Università
Ca'Foscari
Venezia

Corso di Laurea Magistrale
in Economia e Finanza

Tesi di Laurea

Applicazione di Reti Neurali per la Gestione Dinamica del Portafoglio di Investimento

Relatore
Ch. Prof. Marco Corazza

Laureando
Filippo Celeghin
Matricola 892107

Anno Accademico
2022 / 2023

Sommario

INTRODUZIONE

La soluzione del problema di gestione di portafoglio è uno degli argomenti più dibattuti nella letteratura economica, la natura estremamente dinamica e complessa dei mercati finanziari ha reso questi oggetto d'interesse di svariate branche della comunità accademica, da approcci prettamente quantitativi alla finanza comportamentale lo studio dei mercati finanziari ha generato molteplici modelli e soluzioni che cercano di identificare un approccio univoco per la gestione del capitale.

I modelli teorici proposti nel corso degli anni, pur avendo il grande merito di aver formalizzato la materia in modo tale da facilitarne lo studio e la comprensione, si sono dovuti scontrare con la complicatezza derivante dall'applicazione pratica di questi, a causa soprattutto dell'impossibilità di questi modelli teorici di racchiudere, in forma utilizzabile e comprensibile, tutte le dinamiche che caratterizzano i mercati finanziari, la loro continua evoluzione e la complessità che caratterizza non solo il mercato in sé quanto anche le dinamiche psicologiche e sociali che condizionano l'operato degli investitori.

La complessità nel definire un modello pienamente applicabile a livello pratico per la soluzione del problema di gestione di portafogli e lo sviluppo tecnologico hanno aperto una nuova frontiera nella ricerca di nuove soluzioni: l'applicazione di tecniche informatiche avanzate afferenti al campo del *reinforcement learning*, una branca dell'intelligenza artificiale ispirata al processo di apprendimento degli essere umani. Questo offre un approccio promettente per la costruzione di strategie d'investimento dinamiche e adattive, in grado di apprendere dai dati storici e dalle esperienze passate per definire una strategia d'investimento non vincolata a modelli fissi ma in grado di aggiornarsi e adattarsi alle situazioni da affrontare nel corso del tempo.

L'obiettivo principale di questa tesi è l'implementazione di un modello di *deep reinforcement learning*, una tipologia di modelli ispirati al funzionamento di un cervello biologico, in grado di gestire attivamente con successo un portafoglio finanziario composto da un numero limitato di asset, i risultati di questo modello verranno poi confrontati i risultati ottenuti da modelli classici per la gestione di portafoglio al fine di valutare se questo approccio può essere considerato una valida alternativa per la gestione di portafoglio; attraverso un'analisi empirica e l'utilizzo di dati storici questa tesi mira a fornire una panoramica delle potenzialità e limitazioni che caratterizzano questo approccio.

La tesi è suddivisa in cinque capitoli; per fornire un introduzione dei punti fondamentali che il modello proposto nella parte sperimentale della tesi dovrà affrontare, nel primo capitolo viene brevemente esposto il problema di gestione di portafoglio, in particolare verranno esposti i principi fondamentali della *Modern Porfolio Theory*, del modello di selezione di portafoglio proposto dal premio Nobel Harry Markowitz nel 1952 e dei cosiddetti modelli classici di revisione di portafoglio proposti da Smith nel 1967 e da Stone e Hill nel 1979, esponendo pregi e criticità che caratterizzano i modelli di costruzione e revisione di portafoglio e dei fondamenti teorici fondanti della Modern Portfolio Theory. Il capitolo si conclude con una breve disamina delle due macro categorie di approcci pratici alla gestione di portafoglio, la gestione attiva e la gestione passiva, evidenziando le differenze nelle performance storicamente registrate dai due approcci e le limitazioni tipiche della gestione attiva di portafogli.

Il secondo capitolo mira a fornire le basi utili alla generale comprensione del framework caratterizzante il reinforcement learning esponendo i concetti e gli elementi base che ne definiscono l'operatività, viene posta particolare attenzione agli elementi tipici della specifica metodologia del *temporal difference learning*; viene successivamente esposto il funzionamento di una particolare struttura informatica, la rete neurale, e l'applicazione di questa ad un problema di *deep reinforcement learning*.

Nel terzo capitolo viene effettuata una breve revisione della letteratura accademica che ha concentrato i suoi sforzi nell'applicazione di soluzioni attinenti il campo del machine learning a problemi tipicamente affrontati in ambito finanziario, quali la previsione dei prezzi di borsa, il trading e la gestione di portafoglio, riportando per ogni ricerca la metodologia utilizzata ed i risultati ottenuti. La ricerche riportate sono state selezionate secondo l'idea di fornire una visuale il più esaustiva possibile delle diverse modalità e strutture informatiche che possono essere implementate per la soluzione dei problemi preposti andando a selezionare framework differenti per ogni problema affrontato; la parte finale del capitolo è incentrata invece sull'analisi di quelle che sono le applicazioni di machine learning effettivamente implementate nel settore finanziario, evidenziando il grado di utilizzo di queste tecnologie nei vari ambiti che caratterizzano l'attività bancaria e degli istituti finanziari in senso più ampio.

A partire dal quarto capitolo viene introdotto il modello di deep learning, implementato in ambiente Python, proposto per la soluzione del problema gestione di portafogli nella

sezione sperimentale della tesi, in questo capitolo è esposto il funzionamento dettagliato del modello di deep reinforcement learning sviluppato, inizialmente vengono presentatati gli elementi meramente finanziari che caratterizzano il problema affrontato analizzando il processo di selezione dei titoli e le statistiche descrittive di questi registrate nel periodo considerato dall'esperimento. In seguito vengono riportate le specifiche tecniche e informatiche che lo caratterizzano, esplorando il funzionamento degli elementi che lo compongono e le sottostanti formalizzazioni matematiche che contraddistinguono il modello sviluppato.

Nel quinto ed ultimo capitolo vengono confrontati i risultati ottenuti dal modello, implementato con diversi settaggi, con i risultati ottenuti da altri modelli di gestione del portafoglio per avere una visione più chiara delle potenzialità e limitazioni che caratterizzano il modello implementato.

1. SELEZIONE E GESTIONE DI PORTAFOGLIO

Il problema della gestione delle risorse finanziarie, inteso come la ricerca della miglior allocazione di queste, è un problema lungamente dibattuto nella teoria economica. Il problema di gestione e selezione di portafoglio, al suo nucleo, è la necessità di stanziare le risorse disponibili andando a cercare il miglior rendimento possibile ma allo stesso tempo bilanciando i rischi che nascono dall'investimento del capitale. Nel corso degli anni sono stati proposti svariati modelli per la selezione del portafoglio, questo capitolo si concentra sull'analisi della selezione di portafogli à la Markowitz un modello di selezioni di portafoglio che, per quanto abbia dimostrato limiti e criticità, ancora oggi rappresenta elemento fondante della teoria economico finanziaria.

La selezione del portafoglio non è il solo problema da affrontare quando si parla di allocazione delle risorse nel mercato finanziario, l'analisi delle performance, la revisione di portafoglio e la gestione di questo rappresentano altresì elementi fondamentali nella materia, per questo in questo primo capitolo verranno esposti anche quelli che sono definiti i metodi classici di revisione di portafogli. Infine verranno esposti i diversi modelli di gestione di portafogli con le specificità e criticità che li caratterizzano

1.1 Modern Portfolio Theory

Le idee introdotte da Harry Markowitz nell'articolo "*portfolio selection*" da lui pubblicato nel 1952 sono state la pietra miliare di quella che verrà poi definita come *Modern Portfolio Theory* (da qui in avanti MPT). I principi introdotti da Markowitz, nonostante limiti e criticità identificati dallo stesso autore, sono stati lungamente dibattuti all'interno della comunità accademica e, tutt'oggi a oltre 70 anni di distanza, rappresentano una base di partenza essenziale non solo nello studio della materia ma anche nell'applicazione pratica alla gestione di portafoglio. Un altro contributo fondamentale alla MPT viene apportato da William Sharpe nel 1962 con l'introduzione del CAPM, Capital Asset Pricing Model, un importante evoluzione della teoria di portafoglio in quanto introduce il concetto di rischio sistematico (Mangram, 2013). Alla sua essenza, la MPT è un framework d'investimento per la selezione e costruzione di un portafoglio basato sulla massimizzazione dei rendimenti attesi e al contempo sulla minimizzazione del rischio d'investimento (Fabozzi, Gupta e Markowitz, 2002).

Il processo di selezioni di portafoglio introdotto dalla MPT è sintetizzato dal diagramma riportato in Figura 1

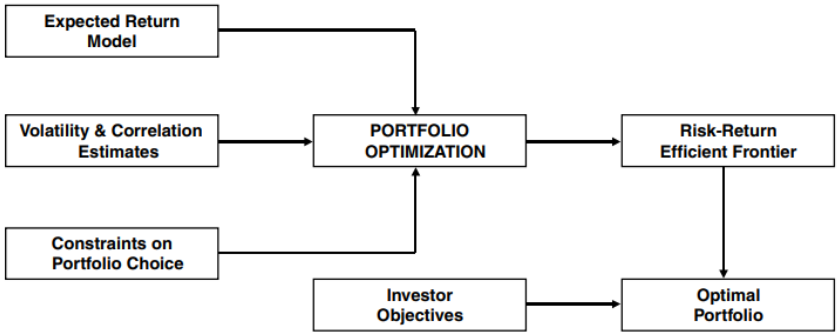

Figura 1 processo di investimento della MPT (Fabozzi, Gupta e Markowitz, 2002).

La prima fase consiste quindi nella valutazione dei rendimenti attesi e della rischiosità dei che popolano il mercato in cui si intende investire; essendo un portafoglio una combinazione di asset differenti il rendimento atteso e la rischiosità di questo dipenderà dalle caratteristiche dei titoli che lo compongono. Nel modello di Markowitz il rendimento atteso di un titolo è identificato con una distribuzione di probabilità normale con media pari alla media dei rendimenti passati, mentre il rischio è rappresentato dalla varianza storica dei rendimenti dei titoli, il modello di portafoglio di Markowitz viene per l'appunto chiamato modello media-varianza in quanto sono le grandezze rappresentative dei titoli. Rendimento atteso e varianza di un portafoglio possono quindi essere definiti rispettivamente come

$$E(R_p) = \sum_{i=1}^{n} r_i w_i \qquad (1.1)$$

$$Var(R_p) = \sum_{i=1}^{n} w_i^2 \sigma_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_i w_j \sigma_i \sigma_j \rho_{ij} \qquad (1.2)$$

Per quanto riguarda il rendimento atteso del portatogli $E(R_p)$ questo dipende dal rendimento atteso r_i dei titoli che lo compongono e dalla frazione di capitale w_i investita in ognuno di questi; la quantità di capitale investita è tale che $\sum_{i=1}^{n} w_i = 1$ in quanto tutto il capitale investito deve essere allocato. Per quanto riguarda la varianza, e quindi la

rischiosità, del portafoglio l'approccio proposto da Markowitz introduce una il concetto di correlazione dei rendimenti, rappresentata, nell'equazione (1.2), dall'indice di correlazione di Pearson ρ calcolato come il rapporto tra la covarianza di due elementi e il prodotto delle rispettive varianze, quindi $\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j}$ tale per cui $-1 \leq \rho \leq 1$ in cui un valore di $\rho = 1$ indica che i rendimenti sono perfettamente correlati, quindi un aumento di un rendimento causa un incremento in ugual misura dell'altro e viceversa, mentre un indice di valore $\rho = -1$ indica che i rendimenti sono perfettamente negativamente correlati quindi ad un movimento dell'uno corrisponderà un movimento uguale opposto dell'altro. L'introduzione del concetto di correlazione dei rendimenti è fondamentale nella definizione della logica di differenziazione proposta da Markowitz, questa permette di considerare la rischiosità di portafoglio secondo un approccio sistemico, un portafoglio non è definito quindi solo dalle caratteristiche intrinseche dei titoli che lo compongono ma anche dalle relazione che intercorrono tra queste, osservando l'equazione (1.2) è possibile affermare che, differenziando facendo attenzione alla correlazione tra i rendimenti nella fase di selezione dei titoli, è possibile costruire un portafoglio con varianza inferiore rispetto al titolo meno rischioso facente parte dell'universo investibile.

Una volta identificati rendimenti e varianze degli asset presenti sul mercato, e stabiliti gli eventuali limiti e vincoli operativi,[1] è possibile definire, utilizzando le equazioni (2.1) e (2.2), rendimenti attesi e varianze dei portafogli possibili. La definizione dei rendimenti e delle varianze di tutti i possibili portafogli ottenibili rende possibile la costruzione della *frontiera efficiente*, ricordando che nella selezione di portafoglio di Markowitz l'investitore è sempre avverso al rischio, la frontiera efficiente è costituita dall'insieme dei portafogli, o singoli asset, che per ogni possibile rendimento offrono il rischio minore, o alternativamente che per ogni livello di rischio offrono il rendimento più elevato. La seguente Figura 2 fornisce un idea grafica di come viene costruita la frontiera efficiente. Idealmente, il piano media-varianza viene diviso per soglie di rendimento atteso $\overline{r_i}$ e per ognuna di queste soglie viene selezionata l'opportunità d'investimento che garantisce il rischio minore, come si nota in figura per i rendimenti attesi $\overline{r_1}$ e $\overline{r_2}$ gli investimenti che portano con sé minor varianza sono rispettivamente X_1 e X_2, questi due investimenti sono quindi parte della frontiera efficiente. Ogni possibile investimento che compone la

[1] Quali, ad esempio, numero massimo e minimo di asset che possono comporre il portafoglio, quota massima e minima di capitale allocabile in un solo asset e così via.

frontiera efficiente è dominante,[2] quindi non sono presenti alternative migliori in termini di media-varianza, rispetto a tutti gli altri investimenti non facenti parte della frontiera efficiente.

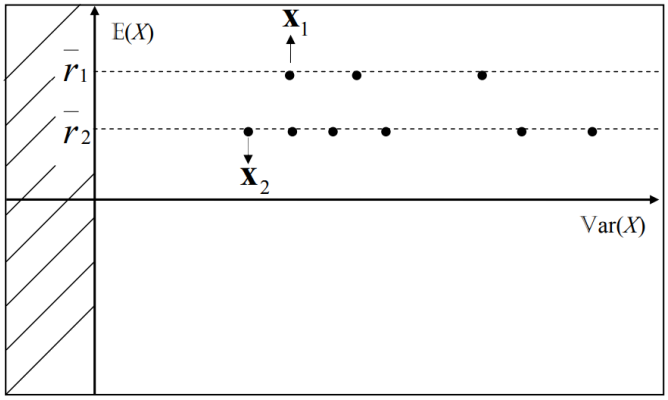

Figura 2 costruzione su piano media varianza della frontiera efficiente.

Una volta definito l'insieme di portafogli che compongono la frontiera efficiente è necessario identificare il portafoglio ottimo per l'investitore, considerando le preferenze dell'investitore con una funzione di utilità $u(\cdot)$, il portafoglio ottimo è quindi il portafogli che massimizza l'utilità attesa $E\big(u(\cdot)\big)$. Graficamente il portafogli scelto dall'investitore sarà quello collocato all'intersezione tra la sua curva di utilità e la frontiera efficiente come rappresentato in Figura 3; da notare come la curva di utilità dell'investitore è concava, rispettando l'assunzione di investitore avverso al rischio.

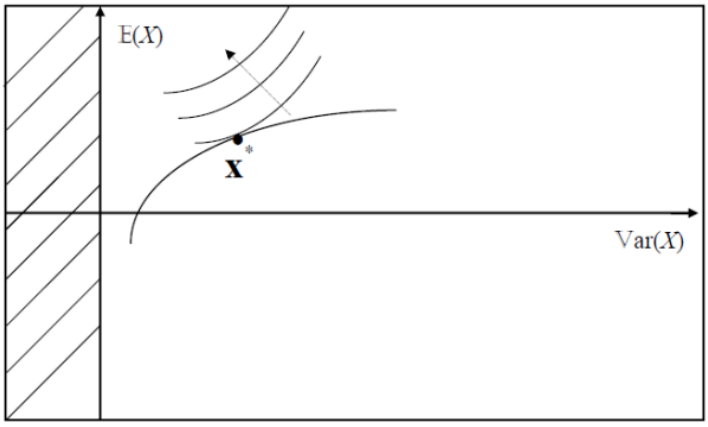

Figura 3 intersezione tra curva di utilità e frontiera efficiente.

[2] Il principio di dominanza indica che $X_1 \succcurlyeq X_2$ se $Var[r_1] \leq Var[r_2] \cup E[r_1] \geq E[r_2]$ e almeno una delle due disuguaglianze è soddisfatta in forma forte.

La selezione di portafogli secondo il criterio media-varianza può quindi essere matematicamente espressa come un problema di minimizzazione della varianza, vincolata all'ottenimento di un certo rendimento π e all'investimento di tutto il capitale disponibile.

$$\min_{x_1 \dots x_n} \sigma_p^2$$
$$s.t. \begin{cases} E(R_p) = \pi \\ \sum_{i=1}^{n} w_i = 1 \end{cases}$$

(1. 3)

1.1.1 Critiche e limiti della MPT

Il modello proposto da Markowitz ha indubbi pregi tra cui la dimostrazione matematica dell'effetto della correlazione dei rendimenti e la consistenza del modello proposto con la logica di massimizzazione dell'utilità attesa (Ingersoll jr., 1987), ma le semplificazioni e assunzioni necessarie affinché il modello sia formalmente corretto rappresentano una forte limitazione all'applicazione pratica del modello, in particolare le principali sono in seguito riportate (Mangram, 2013):

- La MPT assume che l'investitore si comporti sempre in modo razionale in relazione alla sua funzione di utilità ma il comportamento degli investitori si è spesso dimostrato irrazionale andando a seguire comportamenti di massa, ad esempio, orientandosi regolarmente verso settori in forte crescita generando bolle speculative;
- L'ipotesi che gli investitori siano disposti ad accettare rischi maggiori solo a fronte di rendimenti attesi più elevati è spesso contraddetta dal comportamento registrato negli investitori, spesso delle strategie d'investimento adottate prevedono di acquistare titoli, percepiti come rischiosi, per mitigare il rischio del portafoglio senza che questi influenzino il rendimento atteso;
- La presenza di asimmetrie informative tra i vari operatori del mercato è in contrasto con l'assunzione della MPT per cui tutti i partecipanti del mercato hanno accesso a tutte le informazioni reperibili;
- Il modello presuppone che tutti gli investitori possano accedere al mercato del capitale prendendo a prestito denaro illimitatamente al tasso *risk-free*, nella realtà

non è possibile accedere a somme di denaro infinite, in aggiunta, il costo del denaro è sempre superiore al tasso risk-free ottenibile solo dagli stati;

- La MPT poggia sull'assunzione che i mercati siano perfettamente efficienti, nella realtà i mercati sono soggetti a svariati fattori di natura ambientale, personale, strategica o sociale, Inoltre non vengono contemplata la possibili inefficienze di mercato, quali ad esempio costi non contemplati all'interno del prezzo (costi di brokeraggio, *bid-ask spread*,[3] costi amministrativi eccetera). La presenza di costi nascosti può rendere potenzialmente impossibile l'adozione di una strategia d'investimento altrimenti vantaggiosa;
- La MPT presuppone un mercato senza frizioni, quindi oltre all'assenza di costi di transazione e asimmetrie informative si presuppone anche l'assenza di tassazione;
- Un ultima assunzione del modello di Markowtiz prevede la possibilità di selezionare titoli con performance indipendenti dagli altri portafogli, in realtà durante i periodi di recessione gli investimenti presentano livelli di correlazione piuttosto elevati.

Inoltre la valutazione di un investimento secondo la logica di selezione di media-varianza presenta due criticità:

- Il modello presuppone che i rendimenti si distribuiscano secondo una distribuzione normale, si è invece empiricamente dimostrato che i rendimenti non assumono distribuzione normale ma sono leptocurtici, ovvero presentano una maggiore densità distributiva nelle code (rendimenti particolarmente positivi e particolarmente negativi sono più probabili rispetto ad una distribuzione normale) e i valori centrali della distribuzione sono più frequenti. Inoltre i rendimenti presentano generalmente asimmetria positiva, ciò significa che i rendimenti sopra la media sono generalmente più frequenti;
- L'utilizzo della varianza come misura di rischio presenta alcuni problemi, in particolare, essendo una misura di distanza dalla media, considera ugualmente rischiosi rendimenti sotto la media e rendimenti sopra la media, questi ultimi, ovviamente, cercati dall'investitore e non rappresentanti un rischio economico. Inoltre la varianza si può utilizzare su distribuzioni simmetriche,

[3] La differenza tra il prezzo più alto che un compratore offre e il prezzo più basso che un venditore chiede, in pratica la distanza monetaria fra domanda e offerta sul mercato finanziario.

caratteristica come evidenziato in precedenza assente nelle distribuzioni dei rendimenti.

1.2 I modelli classici di revisione di Portafoglio

Oltre alle limitazioni evidenziate nel paragrafo precedente il modello proposto da Markowitz un'ulteriore criticità ovvero la mancanza di dimensione temporale; il modello di selezioni proposto non prevede una vera e propria dimensione temporale tra il tempo *t* in cui il portafogli viene costituito ed il tempo *t+1* in cui viene liquidato, questo implica l'impossibilità, da parte del modello, di considerare processi di modificazione del portafoglio in un tempo intermedio tra l'investimento e il disinvestimento, processo che si rende necessario nel caso in cui le condizioni di mercato subiscano variazioni che possano rendere il portafoglio inizialmente costituito non più in linea con le aspettative dell'investitore, rendendolo non più ottimo o non più efficiente. Per garantire che l'investitore si trovi sempre nella condizione di detenere un portafoglio efficiente, o nella migliore delle ipotesi ottimo, si rende necessario procedere al processo di revisione del portafoglio che si sostanzia nella valutazione del portafoglio attualmente detenuto e nell'eventuale modifica del portafoglio nel caso si renda necessario.

Un primo modello di revisione del portafoglio viene proposto nel 1967 da Keith Smith, il modello proposto è, nella sostanza, un'estensione del modello di Markowitz che lo rende applicabile su base intertemporale. Smith suggerisce quindi un metodo uniperiodale da applicare ad ogni scadenza al verificarsi di date condizioni, portando ad una revisione di portafoglio che tenga conto dei costi di revisione quindi dei costi di transazione e di tassazione (Corazza, 2002).

Questo modello prevede che, ad ogni scadenza temporale predeterminata, venga identificata una nuova frontiera efficiente, se il portafoglio iniziale selezionato al tempo *t* non dovesse più collocarsi lungo la nuova frontiera si rende necessario procedere all'identificazione di un nuovo portafoglio efficiente. Il processo di revisione si limita all'identificazione di un nuovo portafoglio efficiente, non viene quindi considerato se il nuovo portafoglio sia effettivamente un portafoglio ottimo. Alla Figura 4 vengono riportate graficamente le possibilità d'investimento tra le quali l'investitore può scegliere al tempo *t+1* a seguito di uno spostamento della curva d'indifferenza, secondo il modello proposto da Smith l'unica possibile alternativa d'investimento è rappresentata dallo

spostamento dal portafoglio X_t al portafoglio D, questo perché il modello di revisione proposto considera solo la convenienza economica della revisione, non è quindi possibile valutare i portafoglio alternativi secondo una valutazione che consideri sia il rendimento atteso che il rischio che caratterizza i possibili investimenti; in questo modo non è possibile valutare la convenienza di un nuovo portafoglio se presenta una rischiosità differente rispetto al portafoglio iniziale; tutte le alternative d'investimento che presentano diversi valori di varianza rispetto al portafoglio X_t non sono quindi valutabili dal modello.

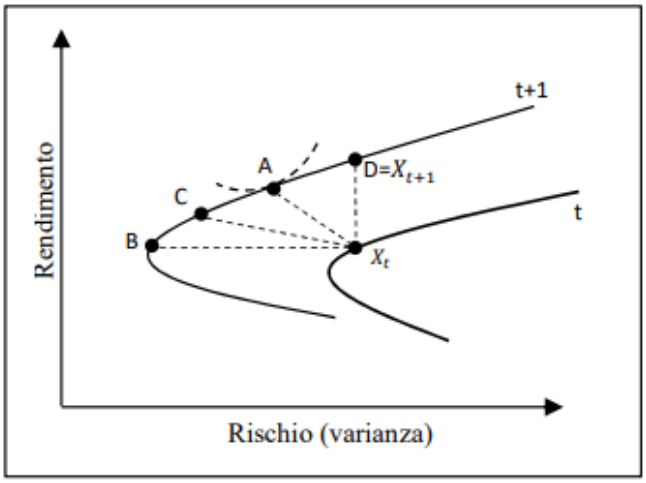

Figura 4 opzioni di revisione di portafoglio secondo il modello di Smith.

Definito il nuovo portafoglio investibile, la revisione è consentita se i costi di revisione non sono superiori all'aumento del rendimento atteso del portafogli, matematicamente la revisione è consentita se viene rispetta la disequazione

$$\Delta E(R_p) * capitale\ investito\ > costi\ di\ revisione \quad (1.\ 4)$$

Dove $\Delta E(R_p)$ indica la variazione del rendimento atteso tra il portafoglio selezionato al tempo *t* e il nuovo portafoglio selezionato al tempo *t+1*. Da notare infine come il problema di revisione di Smith di fatto non consideri la curva si utilità dell'investitore in quanto la variazione avviene solo se il portafogli inizialmente selezionato non è più efficiente.

Per quanto il modello proposto da Smith sia facilmente comprensibile presenta delle criticità che lo rendono difficilmente applicabile, una criticità riguarda la modalità con cui vengono confrontati i costi di transazione con i benefici della revisione di portafoglio

in quanto non sono direttamente considerati dal modello ma vengono applicati solo in un successivo momento, rendendo inoltre impossibile la comparazione tra portafogli con differente rischiosità. Per superare questi limiti Bernell Stone e Ned Hill propongono nel 1979 un modello di revisione strutturato in modo tale da considerare i costi di transazione all'interno del problema di revisione di portafoglio e superare il problema inerente il confronto fra portafogli con varianza differente.

Il modello proposto da Stone e Hill affronta il problema di revisione basandosi su approccio tipicamente utilizzato per la soluzione del problema dello zaino (*knapsack problem*) che spiega come "riempire" in modo ottimale uno zaino in modo da massimizzare l'utilità del contenuto, ogni variazione del contenuto dello zaino, in questo caso del portafoglio, deve quindi considerare la capienza di questo zaino, ovvero il capitale disponibile; la soluzione di questo problema prevede quindi di massimizzare l'utilità del portafoglio vincolando le variazioni del portafoglio al capitale disponibile, ovvero al capitale ottenuto dalla cessione dei titoli liquidati. inoltre, a differenza di Smith, anziché considerare il problema di selezione à la Markowitz seguono l'approccio del CAPM[4] (*Capital Asset Pricing Model*) proposto da Sharpe nel 1963 nel quale il rischio di ogni titolo viene misurato utilizzando il parametro β[5] (Corazza, 2002), in questo modo il rendimento atteso dei singoli asset viene già computato come rendimento aggiustato per il rischio, questo rende possibile superare due limitazioni caratterizzanti il modello di Smith: la possibilità di considerare tra le alternative d'investimento portafogli che presentano livelli di rischio differenti e altresì considerare le preferenze dell'investitore, i rendimenti vengono infatti rappresentati dal seguente vettore

$$c' = (r_1 - \beta_1\theta, \dots, r_N - \beta_N\theta)$$

(1. 5)

dove r_i identifica il rendimento atteso dell'*i*-esimo titolo, β_i è l'indicatore del rischio sistematico del titolo *i*-esimo e θ è un parametro che identifica l'avversione al rischio dell'investitore il cui valore aumenta all'aumentare dell'avversione al rischio.

[4] Modello proposto da Sharpe per cui il rendimento del titolo *i*-esimo può essere espresso in relazione al rendimento atteso di mercato secondo la formula $E(r_i) = \beta_{im}\big(E(r_m) - r_f\big) + r_f$ in cui r_m e r_f rappresentano rispettivamente il rendimento atteso di mercato e il rendimento privo di rischio.

[5] Calcolato per il rendimento r_i del titolo *i*-esimo come $\beta_{im} = \frac{Cov(r_i, r_m)}{Var(r_m)}$.

Il processo di revisione che si sostanzia al tempo *t+1* prevede la possibilità di vendere i titolo presenti all'interno del portafoglio e di acquistarne altri dal mercato, come presentato in precedenza questo processo deve essere svolto in regime di autofinanziamento quindi le risorse disponibili per l'investitore sono solo quelle ottenute dalla cessione di titoli presenti nel portafoglio iniziale, matematicamente il modello viene costruito come un problema di massimizzazione della differenza tra il rendimento atteso, aggiustato per il rischio, ottenibile dai nuovi investimenti e il rendimento atteso, anch'esso aggiustato al rischio, a cui si rinuncia vendendo titoli già presenti nel portafoglio, quindi

$$\max_{z_{b,1},\dots,z_{b,N},z_{s,1},\dots,z_{s,N}} b'z_b - s'z_s$$

$$s.t.\begin{cases} z_b{}'[e + (bc_1, \dots, bc_N)] - z_s'[e - (sc_1, \dots, sc_N)] = 0 \\ z_{b,1} \geq 0 \\ z_{b,i} \leq f_i q'e - q_i \qquad\qquad i = 1, \dots, N \\ z_{s,i} \geq 0 \\ z_{s,i} \leq q_i \end{cases}$$

(1. 6)

in cui:

- $b' = c' - (bc_1, \dots, bc_N)$ e $s' = c' + (sc_1, \dots, sc_N)$ identificano rispettivamente il vettore dei rendimenti aggiustati per il rischio al netto dei costi che si sopportano nell'acquisto dei nuovi titoli e il vettore del "costo opportunità" che si sopporta vendendo titoli presenti nel portafoglio;
- $q' = (q_1, \dots, q_N)$ rappresenta il vettore che indica il capitale investito in ogni titolo presente nel portafoglio prima della revisione;
- $z_b' = (z_{b,1}, \dots, z_{b,N})$ rappresenta il vettore delle quote da acquistare per ogni titolo durante il processo di revisione;
- $z_s' = (z_{s,1}, \dots, z_{s,N})$ rappresenta il vettore delle quote da vendere per ogni titolo durante il processo di revisione.

Per quanto riguarda i vincoli:

- Il primo indica il vincolo di autofinanziamento del portafoglio, il valore delle quote di titoli venduti sottratto il valore delle quote dei titoli acquistato deve essere pari a 0;
- Il secondo e il quarto indicano il divieto di vendite e acquisti allo scoperto;

- Il terzo indica il limite massimo di capitale investibile in un singolo titolo;
- Il quinto indica l'impossibilità di vendere più quote di quante non se ne detengano nel portafoglio iniziale.

La procedura di revisione di portafoglio proposta prevede di acquistare il titolo con la maggior redditività attesa e, allo stesso tempo, di cedere il titolo caratterizzato dalla minor redditività attesa (Corazza, 2002). La selezione dei titoli da acquistare e vendere segue un ordine di utilità, ovvero i titoli detenuti nel portafoglio verranno ordinati in base alla profittabilità della cessione, verranno quindi venduti per primi i titoli la cui cessione garantisce maggior redditività e poi a seguire gli altri, anche per gli acquisti si procede secondo un ordine di utilità, verranno acquistati prima i titoli considerati più redditizi e via seguendo gli altri, il processo di revisione termina quando viene costruito il portafoglio che massimizzi il rendimento *risk-adjusted* atteso investendo tutto il capitale disponibile al tempo in cui avviene la revisione.

1.3 Gestione attiva di portafoglio

Sia il modello di selezione di Markowitz che i modelli di revisione di Smith e di Stone e Hill di cui si è discusso finora nel capitolo rappresentano modelli di selezione e revisione statica, ovvero modelli che vanno a modificare le allocazioni di portafoglio solo in istanti temporali predeterminati. Un approccio largamente utilizzato dai gestori di capitali prevede invece quella che è considerata una gestione dinamica di portafoglio, la gestione e revisione statica infatti non considera le variazioni che il mercato finanziario subisce tra il tempo in cui viene costruito il portafoglio e il tempo in cui questo viene revisionato e modificato, questo approccio può esporre il portafoglio a rischi imprevisti, nel caso in cui le condizioni di mercato cambino repentinamente andando a ridurre il valore del portafoglio o, viceversa, potrebbe portare il gestore di portafoglio a ignorare delle situazioni favorevoli che si presentano sul mercato che potrebbero aumentare la redditività del portafoglio gestito. La gestione attiva, o dinamica, del portafoglio prevede il riaggiornamento dell'allocazione di portafoglio per adattarsi in tempo reale alle dinamiche di mercato per cercare di ottenere un rendimento superiore al mercato stesso, questo approccio è in sostanziale controtendenza con le ipotesi di mercato efficiente in quanto, secondo questa teoria, i valori dei prezzi di mercato sono equi e rispecchiano il valore intrinseco degli asset (Fama, 1969); in questo modo è impossibile per gli agenti

ottenere extraprofitti in quanto, nel caso si presentino opportunità, queste verrebbero immediatamente assorbite dal prezzo di mercato.

L'idea che un investitore sia in grado di "battere il mercato" ovvero sia in grado di ottenere rendimenti superiori rispetto al mercato è in aperta contrapposizione alla teoria del mercato efficiente su cui basano i modelli esposti in precedenza, nel corso degli anni questa teoria è stata lungamente dibattuta, secondo Andrew Lo (2012) le ipotesi di mercato efficiente poggiano sulle seguenti assunzioni:

- La relazione tra rischio e rendimento è lineare;
- La relazione tra rischio e rendimento è statica nel tempo e non influenzata dalle circostanze;
- I parametri che caratterizzano la relazione tra rischio e rendimento possono essere stimati accuratamente;
- Gli investitori hanno aspettative razionali;
- I rendimenti degli asset sono stazionari, hanno quindi una distribuzione costante nel tempo;
- I mercati sono efficienti

Il punto evidenziato da Lo non è tanto capire se queste teorie sono completamente sbagliate o corrette quanto piuttosto, essendo queste teorie semplificazioni della realtà, se gli errori di approssimazione che generano sono sufficienti a generare potenziali vantaggi. Negli anni numerosi economisti hanno presentato le loro critiche alle ipotesi dei mercati efficienti, la nascita dello studio della finanza comportamentale si concentra sulle criticità mostrate dall'ipotesi di razionalità degli investitori (Shiller, 2003; Lo, 2005); altri studi si concentrano sulla confutazione dell'ipotesi che i rendimenti dei titoli quotati seguano un comportamento *random walk*, ovvero siano di fatto imprevedibili andando a studiare correlazioni tra i rendimenti di mercato nel breve periodo ed evidenziando l'esistenza di pattern che spiegano i movimenti di mercato (Lo e MacKinlay, 1999; Shiller, 2000) anche se, come evidenziato da Odean (1999) gli investitori che sfruttano queste correlazioni di breve periodo, il cosiddetto *momentum*, non realizzano extraprofitti. Altre ricerche si sono concentrate, con discreto successo, sulla prevedibilità dei mercati sul lungo periodo utilizzando il rapporto tra prezzi e dividendi (*dividend yeld*) (Fama e French, 1988; Campbell e Shiller, 1988) o utilizzando il rapporto tra prezzo e utili (Campbell e Shiller, 1988), Fama e Schwert (1977) osservano come i tassi di interesse a breve termine sono

correlati ai rendimenti azionari futuri, o ancora Keim e Stambaugh (1986) individuano come i differenziali di rischio tra obbligazioni societarie ad alto rischio e i tassi a breve termine possono avere una certa capacità predittiva. Tuttavia, nonostante in questi lavori venga evidenziata una possibilità di predire i rendimenti futuri del mercato questo non necessariamente identifica inefficienze di mercato, inoltre non è chiaro quanto questa potenziale prevedibilità del mercato possa effettivamente essere utilizzata per generare strategie di investimento redditizie (Malkiel, 2003).

La gestione attiva di portafoglio poggia quindi sugli aspetti critici delle ipotesi di mercato efficiente, come evidenziato dai lavori sopracitati esistono nel mercato potenziali occasioni per generare extraprofitti, siano esse dovute alle inefficienze di mercato, ai comportamenti irrazionali degli investitori o disallineamenti dei valori di mercato. Questo è quindi un approccio flessibile e attivo, basato su scelte tempestive e caratterizzato da frequenti riallocazione del capitale volte a sfruttare le occasioni d'investimento che si presentano; esistono diverse tipologie di strategie nella gestione attiva tra le quali, ad esempio, strategie basate sul *market timing*, ovvero strategie in cui le riallocazioni sono regolate sulla base di previsioni macroeconomiche e sulle fasi cicliche dei mercati; strategie basate sul *momentum* che cercano di sfruttare le tendenze di mercato andando ad investire in titoli che nell'immediato passato hanno mostrato un forte andamento positivo; strategie di *contrarian investing* che puntano ad entrare nel mercato quando questo è in discesa per poter acquistare "a prezzo di saldo" titoli considerati profittevoli, celebre a tal proposito è l'aforisma di Nathan Rothschild "*the time to buy is when there's blood on the streets*".

Nonostante si sia empiricamente dimostrato che i mercati possono presentare possibilità di extraprofitto la gestione attiva degli investimenti e la selezione di singoli titoli, o *stock picking*, raramente riesce a superare i rendimenti di mercato o il *benchmark* utilizzato. Una ricerca, riportata dal quotidiano britannico *The Guardian* nel 2012, ha messo a confronto le performance registrate nello stesso anno da un panel di operatori professionisti del mercato finanziario, da un gruppo di studenti universitari e da un gatto. Ai tre team è stato affidato un capitale di 5.000 sterline a inizio 2012 con la possibilità di cambiarne la composizione ogni tre mesi; al 31 dicembre 2012 il gatto Orlando, che selezionava i titoli lanciando un peluche su una griglia cui ogni spazio corrispondeva ad un azione diversa, ha registrato la performance migliore riuscendo ad aumentare il suo capitale fino a 5.542 sterline, contro le 5.176 ottenute dal panel di professionisti e le 4.840

ottenute dal gruppo di studenti. Questa ricerca, più goliardica che scientifica, pone in evidenza l'effettiva difficoltà nella gestione attiva di portafoglio, in particolare nella selezione dei titoli che possano offrire un rendimento superiore rispetto ad altri.[6]

Un'analisi circa il differenziale di rendimento tra gestione attiva e gestione passiva degli investimenti è stata presentata da Nanigian (2019) in cui vengono confrontati i rendimenti netti aggiustati al rischio ottenuti da fondi comuni d'investimento a gestione attiva e gestione passiva, operanti nel mercato statunitense, nel periodo dal 1991 al 2018. La gestione passiva ha visto un notevole incremento nel suo utilizzo tanto che dal 1994 al 2019 la quota di fondi gestiti passivamente è passata da circa il 2% a poco meno del 40% (Figura 5), nel 2022 la quota di fondi a gestione passiva nel mercato statunitense era del 45%[7] (Figura 6). La ragione di questo aumento risiede nel fatto che, tradizionalmente, i fondi a gestione passiva si sono dimostrati più efficienti e in grado di garantire risultati migliori agli investitori, le principali ragioni possono essere la difficoltà, da parte degli investitori attivi, nel selezionare i titoli da inserire nel portafoglio che garantiscano extraprofitti e i costi che caratterizzano i fondi di gestione attiva; questi ultimi infatti presentano costi di gestione più elevati data la necessità del fondo di essere gestito su base quotidiana, con annessi costi di personale, ricerca e, ovviamente commissioni di trading.

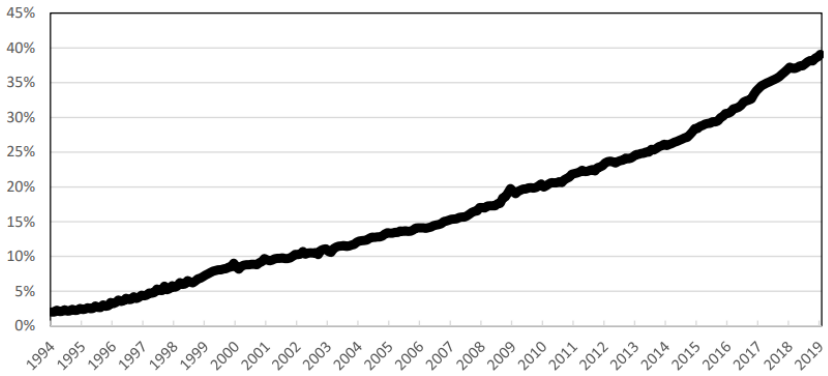

Figura 5 percentuale di fondi gestiti passivamente (Nanigian, 2019).

[6] L'articolo completo "*Investments, Orlando is the cat's whiskers of stock picking*" disponibile al sito https://www.theguardian.com/money/2013/jan/13/investments-stock-picking.
[7] Data dalla somma di fondi basati su un indice e dagli ETF basati su un indice.

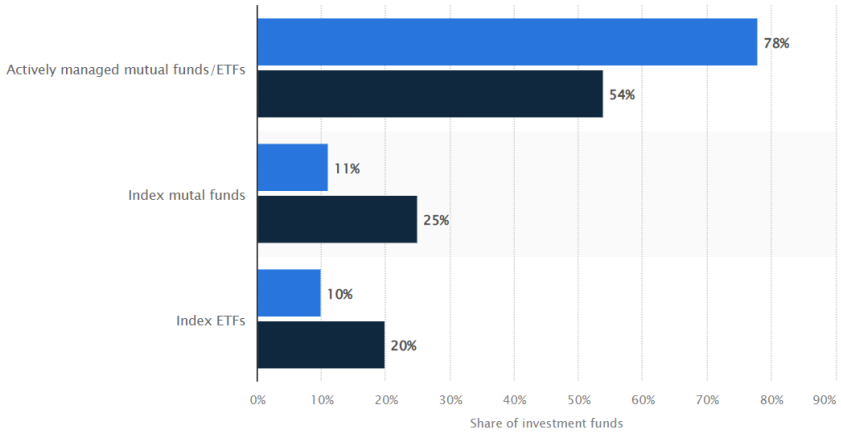

Figura 6 distribuzione dei fondi a gestione attiva e passiva nel mercato USA (Statista, 2023).

La ricerca di Nanigian confronta le performance utilizzando come misura di rendimento aggiustato al rischio il Four-Factor Model proposto da Carhart nel 1997[8] con lo scopo di catturare il rendimento relativo dei portafogli che hanno dimostrato una tendenza di momentum in passato, la valutazione dei portafoglio basa sul valore α che misura il rendimento in difetto, se negativo, o in eccesso, se positivo, rispetto ai fattori di mercato del modello di Fama e French. Nel periodo considerato i fondi passivi hanno registrato una performance superiore, in termini di α, in 16 dei 28 anni considerati per i fondi equipesati, mentre per i fondi ponderati per valore (*value-weighted funds*)[9] le gestioni passive registrano performance superiori rispetto alle attive per 18 anni su 28, mediamente i fondi a gestione attiva registrano un under performance annua rispettivamente del -0.58% e del -0.97% . La differenza di performance dei fondi può, secondo lo studio, essere spiegata dal diverso peso che hanno i costi di gestione nelle due tipologie di gestione, per questo i fondi vengono divisi per quintili di costo, analizzando le performance registrate dai fondi a gestione attiva e passiva che si pongono, per le rispettive categorie, nell'ultimo quintile di costo, ovvero i fondi che presentano i minori costi di gestione, la differenza di performance media nel periodo si assottiglia notevolmente al -0.24% e -0.13% rispettivamente per i fondi equipesati e per i fondi ponderati al valore, comunque i fondi a gestione passiva registrano performance migliori in 15 anni su 28 e 14 anni su 28 rispettivamente alle due tipologie di fondo. La ricerca mette quindi in luce come buona parte dell'under performance dei fondi a gestione attiva può essere spiegata con i maggiori costi di gestione che la caratterizzano, nonostante

[8] Estensione del Three-Factor Model di Fama e French in cui ai fattori di rischio, dimensione e valore viene aggiunto il fattore momentum.
[9] Fondi in cui la percentuale di capitale investita rispecchia la capitalizzazione di mercato.

questa distinzione i fondi a gestione attiva non riescono a superare sistematicamente le performance dei fondi passivi, indicando come questa modalità di gestione sia, fondamentalmente, ingiustificata dai fatti.

2. REINFORCEMENT LEARNING E DEEP LEARNING

L'idea sottostante lo sviluppo del machine learning è la simulazione del processo di apprendimento di un essere umano compiendo azioni e interagendo con l'ambiente circostante. Un bambino che impara a giocare a calcio recepisce informazioni dalle sue azioni, i movimenti che compie per calciare e fermare il pallone, e dall'ambiente con cui interagisce, la direzione e velocità del pallone; dopo aver esercitato le connessioni tra queste due, semplificate, entità sarà in grado di colpire il pallone e far si che questo raggiunga la destinazione desiderata con precisione maggiore. L'idea che si impari dalle interazioni con l'ambiente circostante è alla base delle teorie dell'apprendimento e intelligenza, il machine learning è il filone di scienze informatiche volte a replicare queste teorie e processi in un ambiente artificiale.

L'idea che una macchina, un computer, possa essere in grado di simulare pensieri umani e possa, autonomamente, imparare processi e metodi per risolvere problemi inizia a essere dibattuta con la nascita dei primi computer nella prima metà del secolo scorso. Già nel 1950 il matematico britannico Alan Turing in un articolo pubblicato sulla rivista *Mind* propone di considerare la domanda "*le macchine possono pensare*?". Qualche anno dopo, nel 1959, il matematico statunitense Arthur Samuel crea quello che viene indicato come il primo algoritmo di machine learning, implementato per essere in grado di giocare una partita di dama. Le ricerche negli anni seguenti e, ovviamente, lo sviluppo tecnologico hanno aperto e continuano ad aprire nuove strade al machine learning e allo sviluppo di intelligenze artificiali capaci di risolvere compiti sempre più complessi con sempre maggior precisione. Ad oggi algoritmi di machine learning vengono applicati in svariati ambiti, dal riconoscimento d'immagini alla traduzione dei testi, dalla previsione delle preferenze dei consumatori alla generazione di testi.

In questo capitolo vengono introdotte le basi teoriche e computazionali alla base del machine learning e del deep learning, con particolare focus su uno specifico approccio a questa tecnica ovvero il reinforcement learning. Presentati brevemente alcuni framework che possono essere implementati per risolvere un problema di reinforcement learning viene poi presentato il framework inerente il deep learning, una particolare tecnica di computazione che, prendendo spunto dal funzionamento di un cervello biologico, viene utilizzata per la risoluzione di problemi di reinforcement learning che presentano particolari complessità a livello dimensionale e computazionale. L'introduzione a questi

concetti e tecniche risolutive è propedeutica alla presentazione del modello di gestione di portafoglio che viene presentato nei capitoli finali di questa tesi.

2.1 Elementi introduttivi del Reinforcement Learning

Come accennato, scopo del reinforcement learning, da qui in avanti RL, è far sì che un computer sia autonomamente in grado di intraprendere decisioni volte alla soluzione di un problema, queste decisioni, in gergo azioni, vengono prese dall'agente che, sulla base dell'esperienza accumulata interagendo con l'ambiente (che possiamo definire come il set informativo di volta in volta fornito), calcola qual è la decisione migliore da prendere in relazione allo stato attuale e l'obiettivo da perseguire.

Il meccanismo di interazione tra agente e ambiente è riassumibile dal seguente schema.

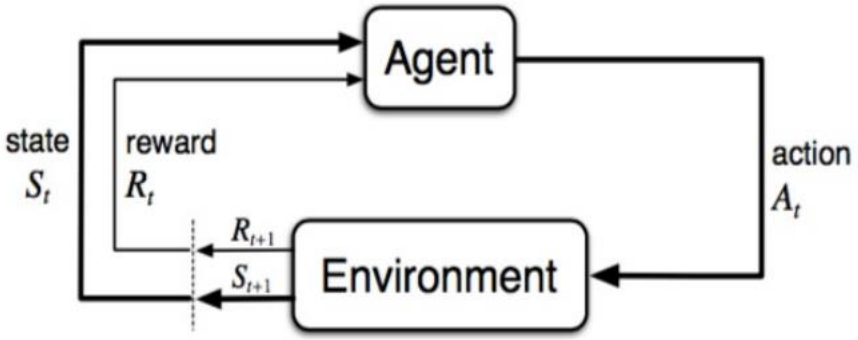

Figura 7 Rappresentazione dell'interazione tra agente e ambiente nel framework RL (Barto e Sutton, 2016).

Ad ogni tempo *t* l'ambiente fornisce come input all'agente lo stato $s_t \in S$ dove *S* rappresenta l'insieme di tutti gli stati possibili, una volta ricevuto l'input l'agente compie un azione $a_t \in A(S_t)$ dove $A(S_t)$ rappresenta l'insieme di tutte le azioni che l'agente può intraprendere nello stato s_t. L'azione intrapresa genera un segnale numerico, la ricompensa $r_{t+1} \in R(S_t)$, dove $R(S_t)$ rappresenta l'insieme delle possibili ricompense ottenibili nello stato s_t. Completato il ciclo si passa al tempo *t+1* in cui l'ambiente fornirà all'agente come input lo stato del mondo s_{t+1} e la ricompensa r_{t+1}. L'azione a_t oltre a generare una ricompensa, dando quindi un indicazione all'agente se la decisione presa era più o meno valida, può, a seconda di come è strutturato il problema, influenzare lo stato successivo andando a modificare direttamente s_{t+1}, è il caso ad esempio del *Cartopole problem* (Barto, 1983) in cui il compito dell'agente è mantenere in equilibrio il più a lungo possibile un asta posizionata verticalmente su un carrello mobile lungo un

percorso limitato, la ricompensa è definita come il tempo in cui l'asta resta in equilibrio mentre l'ambiente è definito da quattro elementi: posizione del carrello, velocità del carrello, inclinazione dell'asta e cambiamento dell'inclinazione dell'asta. In questo framework le azioni che l'agente può intraprendere sono muovere a destra o a sinistra questo carrello, le azioni intraprese dall'agente quindi, oltre a generare la ricompensa, influenzano lo stato allo step successivo.

L'apprendimento per rinforzo è caratterizzato, oltre che dall'agente e dall'ambiente, da quattro ulteriori elementi:

- Una policy π;
- Un sistema di ricompensa;
- Una funzione di valore;
- Opzionalmente, un modello dell'ambiente in cui l'agente opera.

Per policy si intende l'insieme di regole che definiscono la scelta delle azioni da parte dell'agente in un dato momento, in ambito psicologico verrebbe definito come l'insieme di regole di stimolo-risposta (Barto e Sutton, 2016). La policy è la parte centrale di un sistema di RL in quanto generalmente è da sola in grado di spiegare e definire il comportamento dell'agente. Generalmente le policy sono definite come probabilistiche, ovvero come una distribuzione di probabilità su tutte le azioni possibili per ogni stato:

$$\pi: \mathrm{S} \times \mathrm{A} \rightarrow [0,1]$$

Dove la probabilità che l'agente scelta l'azione a dato lo stato s è $\pi(a|s)$.

Il sistema di ricompensa è ciò che definisce l'obiettivo dell'agente, come visto primo ogni interazione dell'agente con l'ambiente restituisce una ricompensa, nel caso del Cartpole problem sopracitato la ricompensa è rappresentata dal tempo in cui l'asta resta in equilibrio, nel caso di gestione di portafoglio la ricompensa può essere generata dal rendimento giornaliero, settimanale o mensile del portafoglio, nel caso del riconoscimento di immagini la corretta identificazione dell'oggetto, e così via. Obiettivo dell'agente è la massimizzazione del computo totale delle ricompense ottenute al termine del periodo. Per quanto siano le azioni dell'agente a generare ricompense il calcolo di queste è esterno all'agente, l'agente può quindi modificare le ricompense cambiando le azioni intraprese ma non può in alcun modo modificare come le ricompense vengono calcolate, non può quindi modificare la funzione generatrice del sistema.

Compito dell'agente non è tanto massimizzare la singola ricompensa quanto piuttosto massimizzare il computo totale di ricompense ottenute nell'intero processo, la funzione di valore viene quindi utilizzata per stimare le ricompense future attese, andando andando ad applicare a queste un dato tasso di sconto; la funzione di valore rappresenta quindi una sorta di valutazione di appetibilità dell'ambiente nel tempo.

La funzione del modello dell'ambiente è quella di sintetizzare l'ambiente, creando quindi un modello che ne mimi l'evoluzione e permetta quindi di ipotizzare i possibili stati futuri. La presenza o meno distingue due categorie di RL, il *model-free* e il *model-based* RL. Un problema che può nascere dall'adozione di un framework model-based è la precisione del modello, essendo questo utilizzato per simulare l'ambiente stesso un modello inaccurato potrebbe portare l'agente a scegliere azioni subottimali, inoltre all'aumentare della complessità dell'ambiente aumenta il costo computazionale sia per la creazione che per la manutenzione del modello, senza contare che più un ambiente è complicato più è difficile strutturare il modello in modo tale che sia in grado di fornire all'agente informazioni utili su di esso, andando quindi ad aumentare il rischio di imprecisione del modello.

2.1.1 Proprietà Markoviana e Processo decisionale à la Markov

Prima di esporre le funzioni di ricompensa e di valore è necessario fare un breve inciso su come la gran parte dei modelli di RL ricavino informazioni dallo stato e su come queste informazioni vengano utilizzate dall'agente. Ad ogni istante temporale *t* il set informativo è rappresentato dallo stato s_t, un sottoinsieme delle informazioni contenute nell'ambiente; difficilmente uno stato può racchiudere tutte le informazioni necessarie all'agente per scegliere l'azione ottima poiché alcune informazioni non sono ricavabili; immaginiamo un agente il cui compito è giocare una partita di blackjack: lo stato è rappresentato dalle carte che ha in mano, per poter calcolare con certezza la ricompensa di questo stato l'agente dovrebbe conoscere anche le carte del banco e quale sarà la prossima carta estratta dal mazzo, entrambe informazioni non accessibili. Ma uno stato non è necessariamente solo la situazione attuale in cui versa l'agente, idealmente uno stato è in grado di racchiudere tutte le informazioni passate utili a definire quale possa essere l'azione ideale, nell'esempio di cui sopra lo stato può contenere anche le informazioni sulle carte estratte durante i turni precedenti, il segnale ottenuto dallo stato in questo caso è sicuramente più informativo per l'agente, che avrà quindi più mezzi per identificare l'azione che potenzialmente garantisce il miglior output.

Un segnale capace di sintetizzare le informazioni rilevanti passate, e quindi far sì che l'azione dell'agente dipenda unicamente dalle informazioni ottenibili da questo segnale, è definito come un segnale con proprietà Markoviana; un esempio in questo senso può essere quello di un agente che gioca una partita a scacchi dove lo stato è rappresentato dalla disposizione dei pezzi sulla scacchiera, in questo caso lo stato è in grado di fornire all'agente tutte le informazioni necessarie per definire la prossima azione, gran parte delle informazioni che hanno portato a questo stato sono perse, l'ordine delle mosse, quali pezzi sono stati usati per magiare i pezzi dell'avversario, ma le informazioni essenziali per definire la prossima mossa sono presenti; in questo caso si parla di proprietà di "indipendenza dal percorso[10]" (Barto e Sutton, 2016).

Considerando la possibile risposta di un ambiente al tempo *t+1* a seguito di un azione intrapresa al tempo *t*, la risposta dell'ambiente può essere espressa come:[11]

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_0, a_0, r_1, \dots, s_{t-1}, a_{t-1}, r_t, s_t, a_t\}$$

(2. 1)

Per ogni *s'*, *r*, e tutti i valori di *a*, *s*, *r* degli stati passati, quindi la risposta dell'ambiente dipende da tutto quanto accaduto dal tempo 0 al tempo *t*. Se il segnale possiede proprietà Markoviana, e quindi la risposta dell'ambiente al tempo *t+1* dipende unicamente dai valori di stato e azione al tempo *t* la stessa equazione può essere espressa come

$$p(s', r \mid s, a) = Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a\}$$

(2. 2)

Per ogni *s'*, *s*, *r*, *a*. La proprietà Markoviana garantisce quindi la capacità di calcolare la probabilità dell'agente di trovarsi nello stato futuro $s_{t+1} = s'$ e ottenere una ricompensa $r_{t+1} = r$ avendo a disposizione le sole informazioni disponibili al tempo *t*.

Un problema di RL che soddisfa la proprietà Markoviana viene definito *processo decisionale à la Markov*, e può essere finito o infinito a seconda che lo spazio di stato e azione siano finiti o infiniti. Un MDP (dall'inglese *Markov Decision Process*) finito è

[10] Traduzione letterale dall'inglese "independence of path".

[11] La formula è espressa, per semplicità, in termini si stati e ricompense finiti discreti ma essere applicata anche a framework infiniti, in tal caso verrebbe espressa come funzione di densità, e non come somme, di probabilità.

definibile dagli stati e azioni possibili e dalla dinamica a un passo esplicitata precedentemente, si possono facilmente calcolare le ricompense attese per le coppie stato azione come:

$$r(s,a) = E[r_{t+1} \mid s_t = s, a_t = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a)$$

(2. 3)

E le probabilità di transizione dello stato come

$$p(s'|s,a) = Pr\{s_{t+1} = s' | \, s_t = s, a_t = a\} = \sum_{r \in R} p(s', r \,|s, a)$$

(2. 4)

2.1.2 Funzione di ricompensa e funzione di valore

Mentre il sistema di ricompense non è altro che il meccanismo attraverso il quale l'ambiente fornisce una ricompensa all'agente, l'agente per raggiungere il suo scopo necessita di una funzione matematica che lo indirizzi verso l'obiettivo, la funzione di ricompensa è quindi definibile come la somma del valore attuale delle ricompense previste future dallo stato s_t al tempo *t*, in cui l'agente si trova, fino allo stato terminale al tempo *T*, ovvero fino a quando non termina l'episodio.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{T} \gamma^k \, r_{t+k+1}$$

(2. 5)

Nell'equazione di cui sopra r_{t+k} indica la ricompensa ottenuta al tempo $t + k$ mentre γ è un parametro rappresentate il fattore di sconto di valore $0 \leq \gamma \leq 1$. Il fattore di sconto indica quanta importanza viene data dall'agente alle ricompense future, un valore pari a 0 porta l'agente ad ignorare le ricompense future, le azioni intraprese punteranno quindi a massimizzare solo la ricompensa immediata r_{t+1}, via via che il valore di γ tende a 1 l'agente sarà al contrario maggiormente influenzato dalle possibili ricompense future, all'aumentare del valore di γ l'agente sarà quindi maggiormente disposto a sacrificare la ricompensa immediata r_{t+1} se questo dovesse portare a maggiori ricompense future. Identificando un framework di gestione di portafoglio si può dire che all'aumentare del valore di γ l'agente sarà più interessato a massimizzare il rendimento finale di portafoglio piuttosto che i rendimenti giornalieri.

Per quanto riguarda invece la *funzione di valore* questa ha il compito di fornire il livello di appetibilità che ha un determinato stato s_t, o, allo stesso modo, il livello di appetibilità che ha intraprendere una data azione a_t nello stato s_t; tale appetibilità è espressa, riprendendo la funzione di ricompensa, in termini di ricompense future attese. Il valore di uno stato $v_\pi(s)$ data la policy π è definito come:

$$v_\pi(s) = \mathrm{E}_\pi[G_t \mid S_t = s] = E_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid S_t = s\right]$$

(2. 6)

Ovvero come il valore atteso E_π, condizionatamente alla policy π, della funzione di ricompensa G_t dato lo stato S_t =s. Allo stesso modo si può definire il valore che ha intraprendere una data azione *a*, nello stato *s*, data la policy π, come la sommatoria delle ricompense future attese, formalmente si può esprimere come da seguente equazione

$$q_\pi(s,a) = E_\pi[G_t|S_t = s,\ A_t = a] = E_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|S_t = s,\ A_t = a\right]$$

(2. 7)

Dove q_π è definito come la *funzione del valore dell'azione a per la policy π*. Da notare come per lo stato terminale *T* sia la funzione di valore $v_\pi(s)$ che la funzione del valore dell'azione a per la policy π $q_\pi(s,a)$ abbiano valore pari a 0 in quanto, terminando l'episodio al tempo *T*, non ci sono ricompense future.

Una caratteristica fondamentale di queste funzioni di valore, assumendo un processo Markoviano discreto, è che soddisfano una relazione ricorsiva, nota come equazione di Bellman, che descrive la relazione che intercorre tra due stati consecutivi ma, ricorsivamente, applicabile per ogni stato $s_t = s$ e ogni possibile stato futuro, matematicamente l'equazione di Bellman è espressa come[12]

$$v_\pi(s) = \mathrm{E}_\pi[G_t \mid S_t = s]$$

$$v_\pi(s) = E_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid S_t = s\right]$$

[12] La formula viene riportata per stati del mondo finiti discreti (t=1,2,3…T) ma è applicabile anche in contesti continui.

$$v_\pi(s) = E_\pi\left[r_{t+1} + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+2} \mid S_t = s\right]$$

(2. 8)

Dove la sommatoria rappresenta di fatto la funzione $v_\pi(s_{t+1})$, e può essere iterata ricorsivamente per tutti i possibili t=1,2,3,…T facenti parte dell'episodio. L'equazione può essere espressa inoltre in termini probabilistici come segue

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma E_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | S_{t+1} = s'\right]\right]$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

(2. 9)

La relazione ricorsiva può essere applicata anche alla funzione $q_\pi(s, a)$ per ogni stato *s*, ogni azione *a* e ogni policy π, ottenendo quindi:

$$q_\pi(s, a) = E_\pi[G_t | S_t = s,\ A_t = a]$$

$$q_\pi(s, a) = E_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} | S_t = s,\ A_t = a\right]$$

$$q_\pi(s, a) = E_\pi\left[r_{t+1} + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+2} | S_t = s,\ A_t = a\right]$$

(2. 10)

Dove, come visto per la funzione $v_\pi(s)$, la sommatoria rappresenta la funzione $q_\pi(s_{t+1}, a_{t+1})$.

Per entrambe le equazioni (2.8) e (2.10) i valori $v_\pi(s)$ e $q_\pi(s, a)$ rappresentano le uniche soluzioni possibili; tali equazioni esprimono la proprietà che il valore dello stato di partenza s_t deve essere uguale alla somma del valore atteso scontato dello stato successivo s_{t+1} con la ricompensa ottenuta nel passaggio dallo stato iniziale allo stato s_{t+1}.

Sia la funzione q_π che la funzione v_π possono essere stimante dall'agente grazie all'esperienza accumulata interagendo con l'ambiente fino al tempo *t* in cui si trova, se

l'agente, seguendo la policy π, calcola e mantiene la media delle ricompense ottenute da uno specifico stato *s*, tale media converge al valore dello stato $v_\pi(s)$ man mano che il numero di volte in cui l'agente incontra tale stato tende all'infinito, allo stesso modo si può stimare il valore della funzione del valore dell'azione $q_\pi(s,a)$ calcolando e mantenendo la media delle ricompense ottenute seguendo una data azione *a.* Questi metodi di stima appartengono alla classe dei metodi *Monte Carlo*, basati sul calcolo delle medie dei valori ottenuti da campioni casuali. L'utilizzo di questi metodi aumentano tuttavia la potenza computazionale richiesta all'aumentare dello spazio degli stati e delle azioni poiché queste devono essere simulate svariate volte al fine di ottenere delle stime precise di $v_\pi(s)$ e $q_\pi(s,a)$; è preferibile quindi come approccio quello di mantenere parametrizzate le due funzioni ed andare a modificarne i parametri ogni volta che viene ottenuta una ricompensa, così da ridurre la distanza tra il valore stimato dalla funzione e l'effettivo valore dello stato *s*.

2.2 Policy ottima e programmazione dinamica

All'interno di un problema di RL l'obiettivo dell'agente è identificare la policy π in grado di garantire il maggior valore della funzione di ricompensa G_t; il processo che porta l'agente ad indentificare la *policy ottima* π_* è definito ottimizzazione della funzione di valore. Innanzitutto una policy π è preferibile ad una policy π' se, per ogni stato *s*, il valore atteso delle ricompense è superiore; matematicamente $\pi \geq \pi'$ se e solo se $v_\pi(s) \geq v_{\pi'}(s)$ per ogni $s \in S$, o alternativamente se $q_\pi(s,a) \geq q_{\pi'}(s,a)$ per ogni $s \in S$ e $a \in A$. La policy che è uguale o superiore a tutte le altre è la policy ottima π_*, è comunque possibile il caso in cui esistano più policy ottime π_*. L'esistenza della policy ottima π_* presuppone quindi l'esistenza di valori ottimi per le funzioni di valore, ovvero $v_*(s)$ e $q_*(s,a)$, definibili come

$$v_*(s) = \max_\pi v_\pi(s) \tag{2. 11}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2. 12}$$

Le equazioni rappresentano quindi le *funzioni ottime di valore* e possono quindi essere espresse senza riferimento ad alcuna policy, inoltre essendo funzioni di valore sono ad esse applicabili le proprietà di Bellman precedentemente esposte.

Definito cosa si intende per policy ottima e per quale ragione un algoritmo di RL deve essere in grado di identificarla è dunque necessario esporre il processo, detto *programmazione dinamica* (Barto e Sutton, 2016) che ne porta all'identificazione attraverso i processi di *valutazione*, *miglioramento* e *iterazione* della policy. Detto che le funzioni ottime di valore rispettano le proprietà di Bellman, queste possono essere riscritte nelle forme ricorsive

$$v_*(s) = \max_a E[r_{t+1} + \gamma v_*(s_{t+1} | s_t = s, a_t = a]$$

(2. 13)

$$q_*(s,a) = E\left[r_{t+1} + \gamma \max_{a'} q_*(s_{t+1}, a') \mid s_t = s, a_t = a\right]$$

(2. 14)

Il processo di *valutazione della policy* si sostanzia secondo la seguente logica: considerando un inizializzazione casuale della funzione di valore $v_k^\pi(s_t)$ ogni successiva approssimazione sarà ottenuta applicando al proprietà ricorsiva dell'equazione di Bellman, quindi

$$v_{k+1}^\pi = E_\pi[r_{t+1} + \gamma v_k^\pi(s_{t+1}) | s_t = s]$$

(2. 15)

Ogni approssimazione successiva sarà ottenuta applicando lo stesso passaggio per ogni stato $s \in S$; il valore precedente di *s* viene sostituito con un nuovo valore stimato dello stesso, ottenuto considerando i valori precedenti degli stati successori di *s*[13] e le ricompense immediate attese. Se v_{k+1}^π esiste, allora la sequenza $\{v_k^\pi\}$ converge al suo valore ottimo per $k \to \infty$, condizionatamente a $\gamma < 1$; essendo eccessivamente impegnativo a livello computazionale ottenere una stima di v_k^π tale per cui $v_k^\pi = v_*$, poiché tale uguaglianza è vera solo in limite generalmente la funzione viene iterata fino

[13] Ovvero i valori delle funzioni di valore associati agli stati futuri immediatamente raggiungibili da un dato stato di partenza.

ad una condizione finale, testando per ogni iterazione la distanza $|v_{k+1}(s) - v_k(s)|$ finché sufficientemente piccola.

Una volta che la policy viene valutata è necessario chiedersi se quest'ultima è effettivamente la policy ideale da seguire per ottenere il massimo risultato possibile, il processo di *miglioramento della policy* è quindi finalizzato a indicare se la policy corrente π è effettivamente una policy ottima π_*. La valutazione della policy permette di conoscere il valore che si ottiene compiendo un azione dato lo stato *s* in cui si trova l'agente, ovvero $v_\pi(s)$, ma non indica se compiere un'azione non prevista dalla policy corrente π può garantire ricompense superiori. L'idea di base è che se, anche se per un solo stato, selezionando l'azione alternativa $a \neq \pi(s)$ si ottiene $q_\pi\big(s, \pi'(s)\big) \geq v_\pi(s)$[14] allora la policy aggiornata π' è necessariamente migliore della policy precedente π.[15] Per la selezione dell'azione si possono adottare approcci differenti, uno di questi prevede il sistema[16]

$$a_t = \begin{cases} \pi(s_t) \text{ con probabilità } 1 - \varepsilon \\ a \in A(s_t) \text{ con probabilità } \varepsilon \end{cases}$$

(2. 16)

Dove $\varepsilon \in (0,1)$ e $\pi(s_t)$ indica l'azione che va a massimizzare l'equazione $q_\pi(s,a)$. Il valore ε indica quindi la probabilità che l'agente non segua l'azione che, in base all'esperienza accumulata al tempo *t*, garantisce la massimizzazione della funzione di valore ma che scelga piuttosto un'azione non ancora selezionata finora, ε indica quindi la probabilità con cui l'agente, ad ogni tempo *t*, esplori alternative alle opzioni finora conosciute al fine di cercare una policy π' potenzialmente migliore.

In ultima, il processo di *iterazione della policy* prevede che, se identificata una policy alternativa π' preferibile a π, si può procede alla valutazione della funzione di valore $v_{\pi'}$ della nuova policy e riapplicare il ciclo valutazione-miglioramento finora esposto, graficamente si può esprimere come una sequenza

[14] La notazione $\pi'(s)$ indica un azione non prevista dalla policy π, tale notazione prende infatti il posto di *a* nella formulazione standard $q_\pi(s,a)$.
[15] La formulazione matematica completa si trova alle pagine 94 e 95 di Barto e Sutton (2016).
[16] Rispetto alla formulazione originale presente in Corazza e Bertoluzzo (2012) si è preferito sostituire $\pi'(s_t)$ con $\pi(s_t)$ per coerenza con la notazione usata finora.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Dove la E indica la valutazione della policy mentre la I indica il miglioramento della policy, questo garantisce che ogni evoluzione della policy sia migliore della precedente a meno che non si tratti già di una policy ottima π_*.

2.3 Temporal Difference Learning e Q-Learning

Un approccio largamente utilizzato nel RL è sicuramente il *temporal difference learning* (o apprendimento per differenza temporali, da qui in poi *TD learning*) che, come visto per la programmazione dinamica, estrapola informazioni dall'esperienza accumulata, in particolare utilizza il metodo bootstrap ovvero stima il valore dello stato *s* senza arrivare allo stato terminale *T*. formalmente, sapendo che il valore di uno stato data una policy $v_\pi(s)$ è

$$\begin{aligned} v_\pi(s) &= \text{E}_\pi[G_t \mid S_t = s] \\ v_\pi(s) &= E_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid S_t = s\right] \\ v_\pi(s) &= E_\pi\left[r_{t+1} + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+2} \mid S_t = s\right] \\ v_\pi(s) &= E_\pi[r_{t+1} + \gamma v_\pi(s_{t+1}) \mid S_t = s] \end{aligned}$$

(2. 17)

Il metodo TD learning utilizza l'ultima equazione (2.17) per la stima di $v_\pi(s)$, la stima è quindi basata sul valore dello stato successivo s_{t+1} piuttosto che sul valore complessivo delle ricompense attese G_t, è un modello di stima basato su un principio ad un passo, ad ogni step temporale viene aggiornata la stima dello step temporale successivo. Non essendo noto il valore $v_\pi(s_{t+1})$ viene usata la stima $V(s_{t+1})$.

Il più semplice metodo TD, definito *TD(0)* segue la seguente logica:

$$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$$

quindi l'aggiornamento del valore dello stato viene eseguito secondo l'equazione

$$V_{k+1}(s_t) = V_k(s_t) + \alpha[r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)]$$

(2. 18)

In cui il parametro α indica il tasso di apprendimento, tale per cui $\alpha \in (0,1]$, dell'algoritmo, γ è il tasso di sconto delle ricompense future, $V_k(s_t)$ è il valore dello stato al tempo *t* mentre $r_{t+1} + \gamma V_k(s_{t+1})$ indica la stima del valore dello stato al tempo *t* calcolata dall'algoritmo; la differenza tra il valore stimato dello stato e il valore effettivamente registrato viene definito *TD-error* (o *errore temporale* in italiano, indicato con δ_t) e indica ad ogni istante di tempo *t* quanto la stima è stata imprecisa, il valore del TD error indica all'agente come direzionare le stime future, se la stima del valore risulta inferiore al valore ottenuto i parametri dovranno essere modificati al fine di ottenere valori di stima più elevati, viceversa se la stima risulta superiore al valore effettivamente ottenuto. Essendo il valore δ_t dipendente dal valore dello stato successivo *t+1* e dalla ricompensa generata dallo stato al tempo *t*, registrata quindi dall'agente al tempo *t+1*, il valore δ_t sarà utilizzato per aggiornare i parametri al tempo *t+1*.

2.3.1 Q-Learning

Proposto per la prima volta nel 1989 nel paper "*Learning from Delayed Rewards*" da Christopher Watkins il *Q-Learning* è un algoritmo di RL off-policy[17] appartenente alla classe dei metodi per differenze temporali. Specificità del Q-Learning è la stima non tanto del valore dello stato $v(s)$ come visto finora ma piuttosto del valore della coppia stato azione $Q(s,a)$. Viene quindi stimato quanto sia vantaggioso compiere un'azione *a* dato stato *s* in cui si trova l'agente. Inoltre, la funzione Q viene stimata approssimando direttamente la funzione ottima q_*. La funzione di aggiornamento della coppia stato-azione segue la seguente regola

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

L'approssimazione della funzione Q alla funzione ottima q_* avviene grazie alla compilazione della *Q-table*, una tabella bidimensionale in cui le dimensioni rappresentano tutte le azioni e tutti gli stati possibili; ad ogni step di aggiornamento l'algoritmo aggiorna la Q-table dove ogni cella di questa tabella rappresenta il Q-value associato ad una data azione dato un certo stato, ad ogni step l'agente seleziona quindi,

[17] Ovvero un algoritmo in cui l'agente è in grado di apprendere da esperienze non necessariamente generate dalla politica di comportamento attuale, quindi un algoritmo in grado di cambiare nel corso del tempo la policy che segue.

dalla Q-table, la coppia stato-azione accessibile con il Q-value stimato più elevato e utilizza gli input ricevuti per aggiornare la tabella stessa. Questa viene utilizzata quindi sia per stimare il valore futuro atteso di coppie stato-azione non ancora esplorate sia per memorizzare l'esperienza accumulata dall'agente.

2.4 Deep learing

La ricerca in ambito di *deep learning* si pone all'intersezione tra i campi inerenti reti neurali, intelligenza artificiale, modellizzazione grafica, ottimizzazione, riconoscimento di pattern e elaborazione di segnali; tre importanti ragioni che spiegano la popolarità del deep learning oggi sono il significativo aumento della capacità computazionale degli strumenti informatici, la possibilità di accedere a grandi quantità di dati per l'allenamento dei modelli e i recenti progressi nella ricerca su machine learning e elaborazione dei segnali e delle informazioni (Deng e Yu, 2014).

Il deep learning consente l'utilizzo di modelli computazionali stratificati composti da molteplici livelli di elaborazione che rendono quindi possibile, per l'agente, apprendere rappresentazioni di dati con molteplici livelli di astrazione. Questo metodo rende quindi possibile l'utilizzo e la rappresentazione di struttura intricate in vasti dataset attraverso l'utilizzo dell'algoritmo di *backpropagation* che permette alla macchina di regolare i parametri interni utilizzati per calcolare la rappresentazione di ciascun strato a partire dalla rappresentazione precedente. Questi approcci hanno avuto un impatto notevole in diversi ambiti, in particolare nel riconoscimento del parlato, nella classificazione di immagini; sono stati utilizzati anche nello sviluppo di farmaci e nella genomica (LeCun, Bengio e Hinton, 2015).

Quanto visto finora sono le nozioni base per il RL applicate ad un modello standard, con il termine deep learning (da qui DL) si intende l'applicazione dei metodi e processi di RL visti finora ad un algoritmo basato su reti neurali. L'utilizzo delle reti neurali garantisce due fondamentali vantaggi: la capacità di utilizzare maggiore capacità computazionale rispetto a modelli standard e quindi la possibilità di usare quantità di dati nettamente maggiori, rispetto a modelli standard, per allenare i modelli (Brunson, 2017).

2.4.1 Le reti neurali

Una *rete neurale artificiale* è un modello computazionale ispirato al funzionamento del sistema nervoso animale progettato per eseguire compiti di machine learning. La struttura

di una rete neurale prevede un insieme di unità di calcolo, chiamate *neuroni* o *nodi*, organizzate in *strati gerarchici* (o *layer*) tra loro collegati sequenzialmente. La più semplice struttura di rete neurale, prevede un layer di input, che recepisce le informazioni necessarie a generare un output, e un layer di output, composto da un solo neurone (chiamato *perceptron* o percettrone), il cui compito è generare l'output richiesto, modelli più complessi possono essere composti da innumerevoli layer intermedi, detti strati nascosti (o *hidden layer*) la cui funzione è quella di garantire un'elaborazione dell'informazione più precisa al fine di garantire un output migliore. Proprio la relativa facilità con cui è possibile modificare una rete neurale nei suoi elementi costitutivi (numero di layer, connessioni tra layer, funzioni di attivazioni dei neuroni ecc…) è uno dei vantaggi che hanno portato all'ampio utilizzo di questi modelli negli ultimi anni.

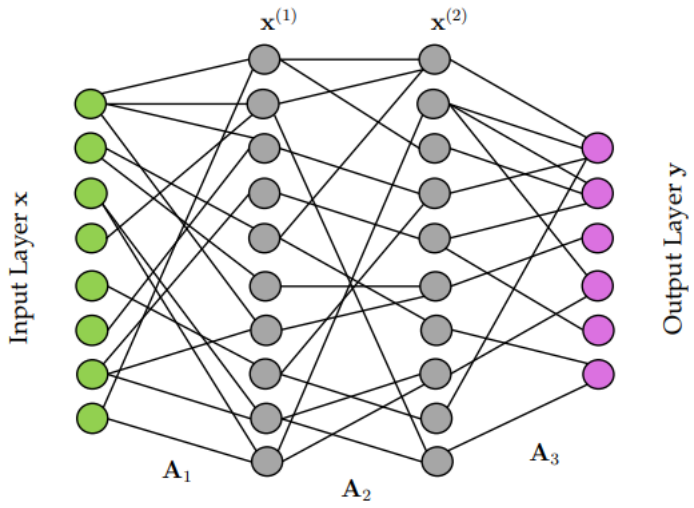

Figura 8 Architettura di rete neurale con due hidden layer (Brunson, 2017).

La Figura 8 illustra una generica struttura di rete neurale, un layer di input x, un layer di output y e due hidden layer $x^{(j)}$ dove j ne indica la posizione sequenziale, le matrici A_i, con i che ne indica la posizione sequenziale, contengono i coefficienti (*pesi* e *bias*) delle connessioni che mappano ogni variabile da un layer all'altro. Nel caso in cui i collegamenti siano lineari la rete neurale presenta le relazioni

$$x^{(1)} = A_1 x$$
$$x^{(2)} = A_2 x^{(1)}$$
$$y = A_3 x^{(2)}$$

A parole si può dire che ogni informazione di input x viene rielaborata e trasmessa al primo layer $x^{(1)}$ dopo essere stata processata secondo i coefficienti della matrice A_1, a

sua volta l'output del layer $x^{(1)}$ viene processato secondo i coefficienti della matrice A_2 e funge da input per il layer $x^{(2)}$, il processo si ripete per tutti i layer della rete fino ad arrivare all'output *y*. Questa struttura di relazioni può essere riscritta per un numero *M* di layer come

$$y = A_M A_{M-1} \dots A_2 A_1 x$$

(2. 19)

Un singolo neurone di un generico layer $x^{(n+1)}$ può essere collegato a un solo neurone (anche se piuttosto rara come struttura), a più neuroni o a tutti i neuroni del layer precedente $x^{(n)}$ quindi il valore recepito da un neurone, e quindi il suo output, è, normalmente, la combinazione di più input. Matematicamente l'informazione ricevuta da un neurone *i* può essere espressa come

$$o_j = \left(o_1 w_{1j} + b_{1j}\right) + \left(o_2 w_{2j} + b_{2j}\right) + \dots + \left(o_n w_{nj} + b_{nj}\right)$$
$$o_j = \sum_{i=1}^{n} w_{ij} o_i + b_{ij}$$

(2. 20)

Ovvero l'informazione o_j ricevuta dal neurone *j* è uguale alla somma di tutti gli input ricevuti $o_1, o_2, \dots o_n$, moltiplicati per i pesi *w* e sommati i bias *b* delle rispettive connessioni tra neuroni[18]. Questo processo si ripete lungo tutta la struttura della rete neurale fino al layer di output.

2.5 Funzioni di attivazione

Finora si è presentato il generico funzionamento di una rete neurale, viene ora introdotta l'idea di mappatura non lineare e di *funzioni di attivazione*. Nel precedente paragrafo la connessione tra due layer è stata presentata come una connessione lineare. Questo genere di modellizzazione rende però difficile, nel caso di relazioni complesse e non lineari tra i dati utilizzati, il riconoscimento da parte della rete dei pattern che caratterizzano gli input, per questo è, spesso, preferibile adottare rete neurali non lineari, che si caratterizzano per

[18] L'equazione presentata è una parziale rielaborazione dell'equazione presentata in Dawson e Wilby (1998), si è preferito adottare la stessa notazione *o* sia per gli input ricevuti grezzi che per l'informazione effettivamente ricevuta dal neurone in quanto l'informazione o_i è essa stessa l'input che riceveranno i neuroni al layer successivo.

la presenza di funzioni di attivazione non lineari tra i neuroni. In questo caso le relazioni tra i neuroni, riprendendo l'esempio riportato in Figura 8, possono essere espresse come

$$x^{(1)} = f_1(A_{1,}x)$$
$$x^{(2)} = f_2(A_2, x^{(1)})$$
$$y = f_3(A_3, x^{(2)})$$

Riscrivibile in forma generale per *M* layer come

$$y = f_M(A_M, \dots, f_2\big(A_2, f_1(A_1, x)\big) \dots)$$

(2. 21)

Si noti come nelle formulazioni di cui sopra vengono utilizzate diverse funzioni non lineari $f_j(\cdot)$, questa non è una regola, tra i vari layer può essere usata sempre la stessa funzione di attivazione, si è preferito utilizzare questa notazione per evidenziare come sia possibile all'interno della stessa rete adottare funzioni di attivazione diverse. Nel caso di connessioni non lineari l'informazione ricevuta dal neurone viene espressa secondo l'equazione

$$o_j = f_j\left(\sum_{i=1}^{n} w_{ij}o_i + b_{ij}\right)$$

(2. 22)

Le funzioni di attivazione maggiormente utilizzate in letteratura sono

$f(x) = \dfrac{1}{1+\exp(-x)}$ – funzione logistica

$f(x) = \begin{cases} 0 \; se \; x \leq 0 \\ 1 \; se \; x > 0 \end{cases}$ – funzione binaria

$f(x) = \begin{cases} 0 \; se \; x \leq 0 \\ x \; se \; x > 0 \end{cases}$ – rectified linear unit (ReLU)

$f(x) = \lambda \begin{cases} x \; se \; x > 0 \\ ae^x - a \; se \; x \leq 0 \end{cases}$ – scaled exponential linear unit (SELU)

$f(x) = \tanh(x)$ – funzione tangente iperbolica (TanH)

Utilizzare una funzione di attivazione non lineare pur portando ai vantaggi esposti in precedenza comporta anche una maggiore complessità della rete neurale, in particolare l'utilizzo di funzioni non lineari rende impossibile le routine di ottimizzazione standard, per questo è necessario adottare la *discesa stocastica del gradiente* o la

backpropagation.[19] Nel proseguo della tesi verrà esposto il solo funzionamento della funzione ReLU oltre al metodo di backpropagation in quanto utilizzati nello sviluppo dell'algoritmo sviluppato per la presente tesi.

2.5.1 Rectified Linear Unit (ReLU) e Leaky ReLU

Introdotta per la prima volta nel 2000 in problemi di RL (Hahnloser et al., 2000), ad oggi la funzione ReLU è largamente utilizzata come funzione di attivazione nelle reti neurali. come suggerisce il nome, la caratteristica principale è la rettificazione dei valori negativi, questa funzione si esprime quindi come

$$ReLU(x) = \max(0, x)$$

(2. 23)

La funzione restituisce quindi unicamente i valori non negativi di *x*, nell'ambito di una rete neurale questa funzione "attiva" il neurone solo se l'input è positivo, altrimenti l'input sarà pari a 0 e, conseguentemente, l'output del neurone sarà 0, riprendendo la formula l'attivazione del neurone segue l'equazione

$$o_j = max\left(0, \sum_{i=1}^{n} w_{ij} o_i + b_{ij}\right)$$

(2. 24)

L'utilizzo di una funzione di tipo ReLU presenta una serie di vantaggi:

- Semplicità di calcolo rispetto ad altre funzioni di attivazione;
- Sparsità rappresentazionale ovvero la capacità della funzione di generare valori di vero zero, questo rende la funzione particolarmente efficiente per l'apprendimento rappresentazionale;[20]
- Comportamento lineare poiché, di fatto, per i valori non negativi l'attivazione è lineare.

La funzione ReLU permette inoltre di ovviare al problema di evanescenza del gradiente, che rende impossibile la minimizzazione globale dell'errore, che invece può affliggere

[19] Il termine italiano consigliato è "retro propagazione dell'errore" o "aggiornamento dei pesi basato sull'errore", per sinteticità della forma viene preferito il termine inglese.
[20] Si pensi ad esempio ad un algoritmo per il riconoscimento d'immagini, tutti i pixel forniti all'algoritmo che non restituiscano un certo valore d'interesse, in pratica che non contengono feature utili per il riconoscimento dell'immagine, possono essere immediatamente scremati attribuendogli il valore di 0.

altre funzioni di attivazione come ad esempio la sigmoide o la TanH. Inoltre si è dimostrato come l'utilizzo di questa funzione di attivazione garantisca un miglioramento, rispetto alle funzioni TanH e sigmoide, nell'allenamento di reti profonde (Glorot, Bordes e Bengio, 2011).

Questa funzione presenta comunque alcuni svantaggi, in particolare il "*dying ReLU*"[21] ovvero il fatto che alcuni neuroni non vengano mai attivati perché ricevono input negativi, rendendo quindi inutilizzata una parte della rete stessa, per ovviare a questo fenomeno nel 2013 viene proposta una variante della funzione ReLU, chiamata *Leaky ReLU*, formalizzata come[22]

$$h^i = \max\left(w^{(i)T}x, 0\right) = \begin{cases} w^{(i)T}x & \text{se } w^{(i)T} > 0 \\ 0.01w^{(i)T}x & \text{se } w^{(i)T} \leq 0 \end{cases}$$

(2. 25)

Nella formula riportata i valori di output del neurone precedente sono rappresentati da *x*, mentre $w^{(i)T}$ rappresenta l'effettivo input ricevuto dal neurone, h^i infine è l'output del neurone. Utilizzando la nomenclatura utilizzata in precedenza per l'equazione della funzione ReLU si ottiene

$$o_j = \max\left(0, \sum_{i=1}^{n} w_{ij}o_i + b_{ij}\right) = \begin{cases} \sum_{i=1}^{n} w_{ij}o_i + b_{ij} & \text{se } \sum_{i=1}^{n} w_{ij}o_i + b_{ij} > 0 \\ 0.01\left(\sum_{i=1}^{n} w_{ij}o_i + b_{ij}\right) & \text{se } \sum_{i=1}^{n} w_{ij}o_i + b_{ij} \leq 0 \end{cases}$$

(2. 26)

La funzione di attivazione Leaky ReLU quindi, andando ad attribuire un valore di output diverso da 0 anche se l'input è negativo, riesce ad ovviare al fenomeno dying ReLU, in Figura 9 viene riportato il confronto grafico fra le funzioni.

[21] Traducibile in italiano con "morte del neurone".
[22] Equazione riportata da Maas el al. 2013.

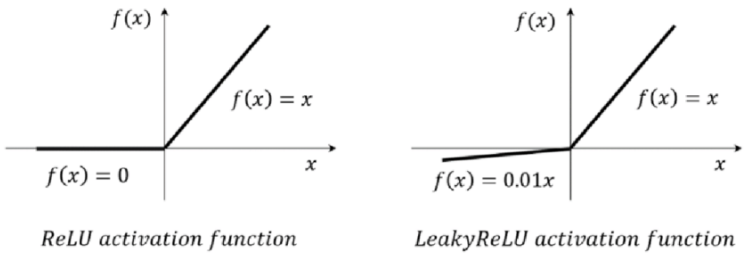

Figura 9 confronto grafico tra le funzioni di attivazione ReLU e Leaky ReLU.

2.6 Funzione di costo e Backpropagation

In linea generale il funzionamento di una rete neurale può essere sintetizzato nei seguenti passaggi:

- Creazione della struttura o *architettura*, ovvero la definizione del numero di layer, numero di neuroni per layer, collegamenti tra essi (pesi e bias possono essere inizializzati casualmente o con valori definiti) e definizione delle funzioni di attivazione;
- *Feedforward* (o propagazione in avanti). La fase di feedforward prevede l'inizializzazione degli input numerici utilizzati al fine di ottenere un output dalla rete, in questa fase gli input vengono quindi propagati dal primo layer di input sequenzialmente in tutti i layer che compongono la rete fino al layer di output, questi input ad ogni passaggio subiscono le variazioni dovute ai coefficienti di connessione e alle funzioni di attivazione, ogni neurone *j* riceve quindi un input, composto dagli output dai neuroni precedenti i ad esso collegati $o_j = \sum_{i=1}^{n}(o_i w_{ij} + b_{ij})$ e restituisce un output $f_j(o_j)$ che verrà a sua volta trasmesso al neurone successivo seconda dei parametri che definiscono il collegamento; nel caso non sia presente nessuna funzione di attivazione $f_j(\cdot)$ l'output del neurone sarà semplicemente o_j;
- *Calcolo dell'errore* nel quale l'output complessivo generato dalla rete viene confrontato con il risultato desiderato utilizzando una *funzione di costo*, che misura la discrepanza tra il risultato ottenuto e la verità nota;
- *Backpropagation*, si può, sinteticamente, descrivere come l'inverso della fase di feedforward, in questa fase infatti l'errore calcolato dalla funzione di costo viene utilizzato per calcolare il gradiente di errore che "risale" la rete neurale partendo

dal layer di output fino al layer di input utilizzando una catena di derivate parziali. Questo processo permette di aggiornare i pesi e bias della rete;

- *Iterazione* ovvero la ripetizione del processo finora descritto per un numero definito di epoche o finché l'errore non raggiunge un valore considerato accettabile.

Nel processo appena descritto la fase di calcolo dell'errore e di backpropagation assumono quindi un ruolo fondamentale poiché sono il cuore del processo di apprendimento della rete neurale, funzionando quest'ultima infatti come una sorta di "scatola nera" rende particolarmente difficile per l'utente capire perché viene generato un output indesiderato e dove si trovi l'errore che lo genera, quindi definire correttamente la funzione di costo e l'utilizzo che la rete fa di questa informazione è di vitale importanza per ottenere l'output desiderato.

Il processo di backpropagation sfrutta la natura composita delle reti neurali per formulare un problema di ottimizzazione volto a determinare il valore dei coefficienti della rete, in particolare sviluppa una formulazione adatta all'ottimizzazione mediante discesa del gradiente basandosi sul principio matematico della catena per differenziazione (Brunson, 2017).

Viene ora riportata la formalizzazione matematica del processo di backpropagation per una rete composta da tre layer composti da un neurone ciascuno, un layer di input, un hidden layer ed infine un layer di output come da seguente Figura 10

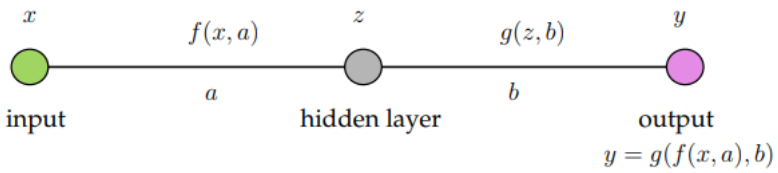

Figura 10 esempio di connessione tra due neuroni successivi (Brunson, 2017).

La relazione di input-output può essere espressa quindi secondo l'equazione che evidenzia la struttura composta della rete in questione[23]

[23] Le equazioni da qui in avanti utilizzate in questo paragrafo si discostano dalla nomenclatura finora usata, la scelta dipende dalla necessità di mantenere la nomenclatura il più comprensibile possibile dati i passaggi matematici che verranno esposti

$$y = g(z, b) = g(f(x, a), b)$$

(2. 27)

dove $f(\cdot)$ e $g(\cdot)$ rappresentano le funzioni di attivazione dei neuroni mentre a e b rappresentano le costanti di pesi e bias, data la formula l'errore E viene descritto dall'equazione

$$E = \frac{1}{2}(y_0 - y)^2$$

(2. 28)

dove y_0 rappresenta l'output desiderato mentre *y* rappresenta l'output stimato dalla rete neurale, l'obiettivo è quindi identificare i valori di a e b che minimizzano la distanza $y_0 - y$; essendo un problema di minimizzazione si segue l'equazione

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz}\frac{dz}{da} = 0$$

(2. 29)

Da notare come la natura composita della rete neurale assieme al principio della catena per differenziazione forza il processo di ottimizzazione a propagare l'errore attraverso la rete. I termini $\frac{dy}{dz}$ e $\frac{dz}{da}$ dimostrano come la propagazione avviene (Brunson, 2017). Il processo di backpropagation risulta nella regola iterativa di discesa del gradiente

$$a_{k+1} = a_k + \delta\frac{\partial E}{\partial a_k}$$

(2. 30)

$$b_{k+1} = b_k + \delta\frac{\partial E}{\partial b_k}$$

(2. 31)

dove δ identifica il *tasso di apprendimento* della rete mentre $\frac{\partial E}{\partial a_k}$e $\frac{\partial E}{\partial b_k}$ possono essere calcolate utilizzando l'equazione (2.29). L'iterazione dell'algoritmo è strutturata in modo da convergere al valore desiderabile di a e b. Il processo di backpropagation è quindi fondamentale per l'allenamento della rete e si può sintetizzare nei seguenti passaggi:

I. Definizione della rete neurale e del *training set*, ovvero dei dati con output y_0 noti utilizzati per allenare la rete;
II. Inizializzazione dei parametri, ovviamente una inizializzazione dei parametri a e b già vicina ai valori che minimizzano l'errore E garantisce un processo di apprendimento più rapido ma questo è raramente possibile, importante invece è non settare i parametri a 0 poiché questo causerebbe valori dei gradienti, e quindi dei parametri a e b identici per ogni neurone. Inoltre spesso le reti neurali restano bloccate in punti di minimo locale, con gradiente 0, che non necessariamente sono minimi globali, rendendo incompleto il processo di allenamento;
III. I dati del training set sono utilizzati per ottenere un output *y*, avendo questi dati un output desiderato noto vengono utilizzati per calcolare l'errore E come da formula();
IV. Calcolo delle derivate esposte all'equazione (2.29);
V. Calcolo dei pesi aggiornati secondo l'equazione (2.30) e (2.31);
VI. Il processo viene ripetuto a partire dal terzo passaggio per un numero definito di iterazioni o finché non viene raggiunta la convergenza $E = 0$.[24]

Per quanto riguarda invece la funzione di costo ottimale questa è strettamente legata alla tipologia di problema che si affronta, nel problema affrontato in seguito nella tesi si è adottato come funzione di costo la differenza pesata tra il Q-value stimato dalla rete neurale e il Q-value effettivo.[25]

2.7 Deep Q-Learning, teoria e applicazione

Come visto finora il DL può essere semplificato come l'applicazione di metodi di RL a modelli basati su reti neurali, questo significa che potenzialmente ogni algoritmo di RL può essere implementato su questa struttura di calcolo. Nello specifico si farà riferimento al *Deep Q-Learning* (a cui ci si riferisce anche come *Deep Q-Network*, d'ora in poi *DQN*) che prevede quindi l'implementazione di un algoritmo di Q-learning tramite reti neurali.

L'implementazione di un algoritmo DQN nasce dalla necessità di poter applicare modelli di Q-learning ad ambienti particolarmente estesi. Il Q-learning infatti si è dimostrato

[24] La convergenza $E = 0$ è ideale e difficilmente raggiungibile, generalmente si definisce raggiunta la convergenza quando la funzione di costo si stabilizza su un valore minimo.
[25] La formulazione completa viene riportata nel Capitolo 4 in cui viene esposto l'esperimento.

particolarmente efficiente perché in grado di stimare, attraverso l'esplorazione dell'ambiente, una funzione Q convergente la funzione ottima q_* utilizzando la Q-table. Il problema di applicabilità del Q-learning nasce quando si utilizzano framework con elevata dimensionalità di stati e azioni, in questo caso l'utilizzo del Q-learning si è dimostrato inefficace per due principali ragioni: il tempo richiesto per ottenere un livello di esplorazione accettabile e la capacità computazionale e di memoria necessaria per identificare e stimare i Q-value per ogni coppia stato-azione. Il DQN nasce quindi per risolvere questi problemi, l'utilizzo di una struttura basata su reti neurali permette infatti di approssimare la funzione Q senza necessariamente stimare tutti i Q-value come con il Q-learning tradizionale, l'utilizzo di una rete neurale, essendo di fatto un metodo di stima, non garantisce la convergenza della funzione Q con la funzione ottima ma permette di sfruttare i vantaggi garantiti dal Q-learning anche in presenza di spazi dimensionali particolarmente estesi.

Il DQN viene proposto per la prima volta nel 2015 in una ricerca compiuta dal team *DeepMind*, obiettivo della ricerca era strutturare un algoritmo di RL in grado di imparare, autonomamente, a giocare ai giochi *Atari 2600*. L'input fornito all'agente è rappresentato dai punteggi ottenuti durante il gioco e da sequenze di pixel che rappresentano l'input visivo che riceve il giocatore, questi input permettono all'agente di "vedere" la variazione dello stato generato dall'azione eseguita, la risposta dell'ambiente e la ricompensa ottenuta, formalmente la funzione Q utilizzata viene riportata secondo l'equazione

$$Q^*(s,a) = \max_{\pi} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi]$$

(2. 32)

data quindi dalla massimizzazione della ricompensa cumulata r_t ottenibile, scontata per il fattore γ, ad ogni step temporale *t*, ottenibile seguendo la policy $\pi = P(s,a)$ dopo aver osservato lo stato *s* e aver eseguito un azione *a*. Per ovviare all'instabilità che un'approssimazione non lineare, la rete neurale, può generare è stato utilizzato un meccanismo di ispirazione biologica, definito *experience replay*, che rende randomico l'utilizzo delle esperienze passate, andando quindi a rimuovere la correlazione tra le osservazioni; questo meccanismo consiste nell'immagazzinamento dell'esperienza dell'agente $e_t = (s_t, a_t, r_t, s_{t+1})$ ad ogni step temporale *t* in un dataset $D_t = \{e_1, \ldots, e_t\}$; durante la fase di allenamento del modello gli aggiornamenti dei Q-value vengono

effettuati su campioni di esperienza $(s, a, r, s') \sim U(D)$ estratti casualmente e uniformemente dal dataset D, infine la funzione di costo è data dall'equazione

$$L_i(\theta_i) = E_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i) \right)^2 \right]$$

(2. 33)

dove γ è il fattore di sconto, θ_i rappresenta i parametri della rete neurale all'interazione i mentre θ_i^- identifica i parametri utilizzati dalla rete per calcolare il Q-value all'interazione i; quest'ultimi sono aggiornati utilizzando i parametri θ_i ogni C step temporali[26]. La struttura della rete neurale utilizzata viene graficamente presentata come segue

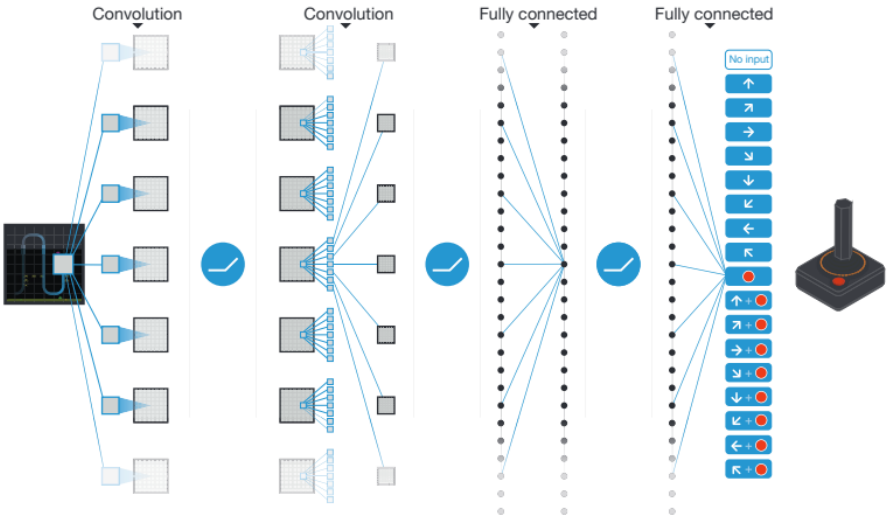

Figura 11 rappresentazione grafica del framework utilizzato (Mnhi et al., 2015).

L'utilizzo di questo modello innovativo ha permesso all'agente di imparare a giocare a tutti i 49 giochi proposti, mantenendo invariata la struttura della rete e i parametri utilizzati. I risultati mostrano come l'algoritmo DQN sia in grado non solo di ottenere risultati migliori di un classico algoritmo di Q-learning in larga parte dei giochi proposti ma addirittura di ottenere risultati migliori di giocatori umani professionisti come riportato dal grafico in Figura 12.

[26] La metodologia completa e la spiegazione viene lasciata al paper pubblicato da Mnih et al., 2015 di cui si sta discutendo

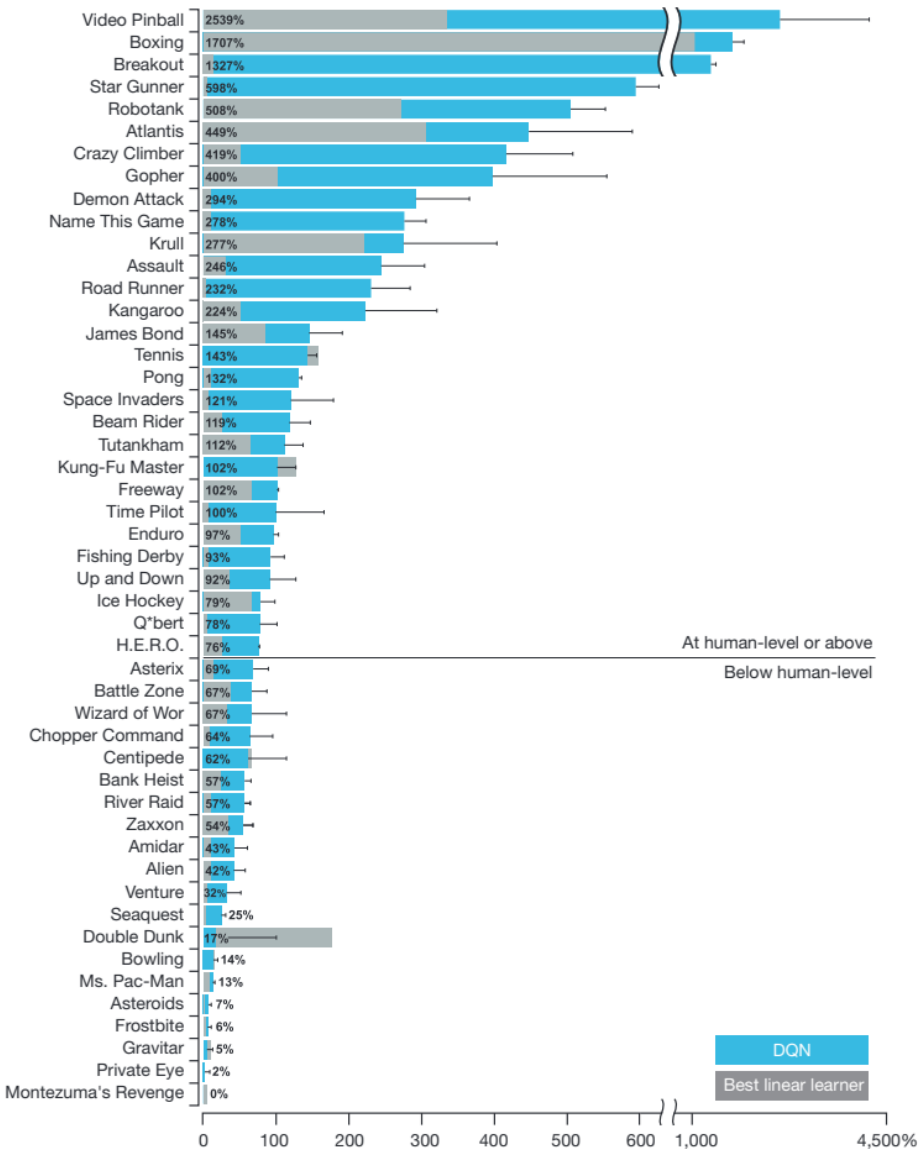

Figura 12 confronto dei risultati ottenuti dal framework DQN, in blu, e da un normale framework Q-learning, in grigio, nei 49 giochi Atari 2600 testati (Mnih et al., 2015).

3. MACHINE LEARNING E APPLICAZIONI FINANZIARIE

L'applicazione di tecniche di machine learning in ambito finanziario è un filone di ricerca relativamente recente se comparato con altri ambiti di ricerca in cui il machine learning ha ormai assunto un'importanza rilevante nello sviluppo di nuove tecnologie anche già largamente distribuite al pubblico. La ragione di questo ritardo, se così si può definire, è dovuto alla complessità, incertezza e continua evoluzione delle dinamiche e relazioni che caratterizzano il mercato finanziario. La gran parte della letteratura che tratta infatti l'utilizzo del machine learning a problemi di trading o di gestione di capitali, nasce infatti a partire dai primi anni del nuovo millennio quando la stabilizzazione dei modelli di deep learning ha reso possibile lo sviluppo di modelli di apprendimento profondo in grado di gestire con maggior accuratezza ambienti estremamente estesi e complessi come per l'appunto l'ambiente rappresentato dai mercati quotati. L'applicazione del machine learning in questo contesto è quindi un area di studio in rapida espansione data la potenzialità già dimostrata da questi metodi in altri ambiti dove il *labeling* dei dati, la previsione dei trend e la capacità di risposta a input imprevisti sono fondamentali.

Obiettivo di questo capitolo è analizzare brevemente parte della letteratura accademica che ha concentrato i suoi sforzi nello sviluppo di algoritmi di machine learning negli ambiti della previsione di mercato, del trading e, più approfonditamente della gestione di portafogli. Infine vengono sinteticamente riportate le attuali applicazioni pratiche delle tecniche di machine learning e intelligenza artificiale negli istituiti finanziari e bancari.

3.1 Previsione di mercato

Nell'ambito della previsione dei prezzi di mercato le tecniche di RL e DL vengono utilizzate con lo scopo di identificare, filtrare e utilizzare le informazioni, derivanti sia dai dati di mercato che da altre fonti, che possono impattare sui prezzi degli asset quotati ed utilizzarle quindi per costruire un modello in grado di prevedere l'andamento futuro dei prezzi.

Pur rudimentale e molto lontano dagli standard di efficienza informatica attuali ritengo comunque rilevante riportare il primo lavoro finalizzato a sfruttare le reti neurali per prevedere i movimenti futuri dei prezzi mercato presentato da Halbert White (1988); in questa ricerca viene sviluppata una rete neurale feed-forward, a tre layer con l'obiettivo di prevedere i rendimento futuro del titolo IBM per poter prevedere e sfruttare distorsioni

di mercato. Strutturato per simulare una regressione lineare dei rendimenti futuri il framework non si è dimostrato particolarmente efficiente nelle previsioni ma è, di fatto, la prima applicazione di questo framework a questa classe di problemi. Negli ultimi anni sono state presentate ricerche che hanno riscontrato maggiori fortune grazie a framework informatici estremamente avanzati rispetti a quanto presentato da White. Chong et al. (2017) sviluppano un framework di DL con l'obiettivo di prevedere i rendimenti di 38 titoli facenti parte dell'indice KOSPI (Korea Composite Stock Price Index). Gli input utilizzati nel modello sono i prezzi di mercato, e tre trasformazioni di questi effettuati tramite altrettante reti neurali: una rete PCA (*Principal Component Analysis*), un metodo per la riduzione della dimensionalità che punta a trasformare un insieme di variabili correlate in un insieme di variabili non correlate, chiamate appunto componenti principali; una rete AE (*Aurtoencoder*), utilizzata per apprendere codifiche di dati non etichettati; ed infine una rete RBM (*Restricted Boltzmann Machine*), utilizzata per l'apprendimento e modellazione di distribuzioni di probabilità complesse. Questo framework si è rilevato più efficiente dei un modelli autoregressivi lineari con cui è stato confrontato nella previsione dei rendimenti. Shen et al. (2015) sviluppano una DBN (*Deep Belief Network*), una particolare tipologia di rete neurale in cui l'aggiornamento dei parametri avviene mediante RBM, per la previsione dei tassi di cambio. Testata sui tassi di cambio tra sterlina e dollaro si è dimostrata superiore ai modelli statistici standard come il modello random walk e il modello ARMA.

Oltre alla classica valutazione matematico-economica sono stati proposti approcci alternativi per la predizione dei valori di mercato, Mittal e Goel (2011) presentano un modello di RL in grado di prevedere l'andamento dell'indice DJIA (*Dow Jones Industrial Average*) utilizzando come input di un modello SOFNN (*Self Organizing Fuzzy Neural Network*)[27] il sentimento pubblico ricavato dai dati della piattaforma social Twitter e l'andamento di mercato recente dello stesso indice. Data l'enorme quantità di "*tweet*" da analizzare vengono selezionati soltanto quelli che contengono parole associabili ad uno stato d'animo, questi tweet vengono poi classificati in quattro classi di "sentimento pubblico" utilizzando una lista di parole di riferimento basata sul questionario POMS.[28] Il modello viene allenato per associare a ognuna di queste quattro classi di sentimento

[27] Ovvero una rete neurale che unisce i concetti di auto-organizzazione, ovvero di apprendimento non supervisionato, e logica fuzzy (un approccio logico che, a differenza dell'approccio classico basato su verità binarie, consente l'esistenza di gradi intermedi di verità).
[28] Profile of Mood State, un questionario psicometrico utilizzato per definire lo stato d'animo di un soggetto.

pubblico una reazione del mercato azionario, i risultati presentano una precisione della previsione del 75.56%. Una ricerca simile viene presentata da Matsubara et. al (2018) in cui propongono un modello di DL il cui obiettivo è prevedere se il prezzo dei titoli, quotati sul mercato giapponese *Nikkei 225*, subiranno un rialzo o un ribasso sulla base dell'analisi di notizie pubblicate su media di informazione in combinazione con prezzi di mercato registrati; l'idea sottostante è sfruttare la sensibilità del mercato azionario a eventi quali lanci di nuovi prodotti, report aziendali e simili. Il modello proposto pur non essendo in grado di dare una previsione dei prezzi puntale si è dimostrato più efficiente nella previsione di rialzo o ribasso rispetto a modelli basati sui soli dati di mercato nonostante abbia mostrato una tendenza all'*overfitting*.[29]

3.2 Trading azionario

Per quanto i modelli previsionali abbiano dimostrato una buona capacità nella previsione dell'andamento dei prezzi non sempre vengono integrati in modelli di machine learning votati al trading di titoli, la ragione di questo è dovuta al fatto che un modello di trading basato sulla previsione dell'andamento futuro dei titoli è strettamente legato, nel suo funzionamento, alla bontà della previsione stessa; questo, ad esempio, può causare la mancata reazione dell'agente ad eventi particolarmente importanti ma non previsti dal modello previsionale sottostante (i cosiddetti *cigni neri*). Si sono sviluppati quindi modelli basati non tanto sulla previsione dei prezzi e rendimenti futuri dei titoli quanto piuttosto sulla capacità dell'agente di interpretare *step-by-step* gli input forniti.

Bertoluzzo e Corazza (2012) propongono due approcci di RL applicati al trading: un algoritmo *Kernel-based Reinforcement Learning* e un algoritmo di Q-learning standard, che non fanno quindi affidamento su reti neurali per la computazione dell'output. In questo algoritmo lo stato è definito dagli ultimi cinque rendimenti registrati sul mercato e le azioni possibili per l'agente sono acquisto, vendita e restare fuori dal mercato,[30] quest'ultima azione prevede la chiusura delle posizioni attualmente aperte. La funzione di ricompensa dell'algoritmo è invece strutturata sulla massimizzazione dello Sharpe ratio degli investimenti effettuati calcolato sui dati degli ultimi cinque e ventidue giorni. Data

[29] Ovvero una tendenza del modello a "imparare a memoria" l'evoluzione dei dati durante la fase di allenamento e quindi a riproporre gli stessi pattern memorizzati durante l'allenamento anche su dati differenti durante la fase di test.
[30] Traduzione dall'originale "stay-out-of-the-market signal".

la complessità computazionale richiesta per simulare le azioni in un ambiente di mercato, e dati i limiti del Q-learning riportati da questa tesi nel Paragrafo 2.7, la funzione di valore allo step k viene approssimata come $\hat{V}_k(s_t) = \hat{V}_k(s_t, \theta_t)$ in cui θ_t rappresenta un vettore di parametri i cui valori ottimi sono ottenuti mediante minimizzazione dell'errore quadratico medio tra il valore effettivo $V^{\pi}(s_t)$ e la sua stima $\hat{V}^{\pi}(s_t, \theta_t)$. Gli algoritmi testati 1000 volte sui titoli Banca Intesa e Fiat hanno registrato, nel periodo considerato e per il miglior settaggio di parametri utilizzato, un rendimento cumulato medio del 99.66% e 133,82% anche se con una varianza piuttosto elevata, rispettivamente 139,71% e 147,53%.

Nel paper di Dash e Dash (2016) viene proposto un framework di RL basato su una rete CEFLANN (*Computational Efficient Functional Link Artificial Neural Network*)[31] e regole di analisi tecnica. In questo paper il problema di trading dei titoli viene articolato in un problema di classificazione con tre classi di valori rappresentanti i segnali di acquisto, vendita e mantenimento della posizione attuale. Una particolare caratteristica riguarda il processo di allenamento del modello, questo infatti non viene effettuato mediante backpropagation ma ogni layer viene allenato mediante un modello ELM (*Extreme Learning Machine*).[32] Il modello sviluppato estrapola dal database di valori forniti sei indicatori tecnici (MA_{15}, $MACD_{26}$, K_{14}, D_3, RSI_{14}, WR_{14}) che vengono poi utilizzati per effettuare l'analisi tecnica utile poi a fornire il segnale numerico di trading; il segnale di trading così fornito viene classificato in una delle tre classi definite (acquisto, vendita e mantenimento della posizione) ed infine l'agente, a seguito della classificazione, sceglie l'azione da perseguire con l'obiettivo di massimizzare il rendimento. Il modello, testato sugli indici S&P 500 e BSE SENSEX (indice della borsa di Bombay, India) ha fornito un rendimento, rispettivamente, del 24.28% e del 47.2% annuo battendo gli altri modelli testati (*support vector machine* o SVM, *Naïve Bayesian model*, *K nearest neighbor model* o KNN e *decision tree model* o DT).

[31] Ovvero una rete con bassa complessità computazionale in cui il vettore di input viene espanso funzionalmente per ottenere soluzioni non lineari (Pujari e Majhi, 2018).
[32] Modello di allenamento che prevede la definizione dei parametri di rete direttamente sull'output desiderato, questi parametri una volta definiti non vengono più aggiornati riducendo significativamente il tempo di allenamento della rete.

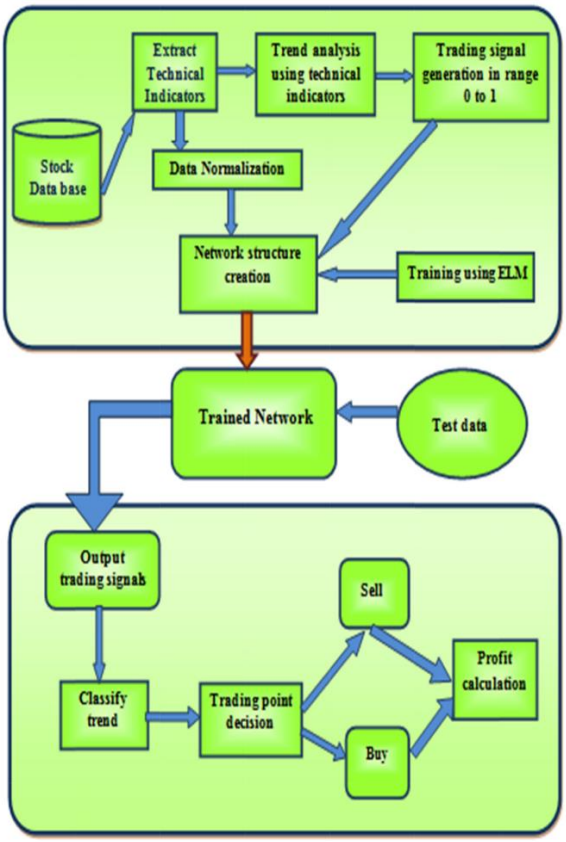

Figura 13: architettura del modello utilizzato (Dash e Dash, 2016).[33]

In Liu et al. (2019) viene proposto un algoritmo afferente alla classe DDPG *(Deep Deterministic Policy Gradient*) per il trading su trenta titoli quotati presso la borsa statunitense Dow Jones Industrial. Suddetto algoritmo è composto da tre componenti chiave: un framework *actor-critic*[34] per la modellizzazione degli stati e delle azioni, un rete target per la stabilizzazione del processo di training e un *experience replay buffer* per eliminare la correlazione tra i campioni di esperienza.[35] Il processo di trading è formalizzato come un MDP (*Markov Decision Process*),[36] il compito dell'algoritmo è massimizzare i rendimenti dell'investimento; lo stato è definito come $s = [p, h, b]$ composto quindi da i prezzi delle azioni *p*, il numero di azioni possedute *h* per ogni titolo e la disponibilità liquida restante *b*; le possibili azioni *a* che l'agente può compiere sono

[33] Da notare, nel diagramma, l'assenza del segnale di mantenimento della posizione, questo segnale viene recepito se il segnale al tempo *t* coincide col segnale al tempo *t-1*.
[34] Un particolare framework di RL che prevede l'implementazione di due reti cooperanti, la rete actor con il compito di apprendere la politica da perseguire, la rete critic con il compito di apprendere la funzione di valore, durante l'addestramento quindi la rete actor prende decisioni sulla base degli input ricevuti dallo stato mentre la rete critic fornisce feedback sulla bontà delle azioni intraprese.
[35] Pratica utilizzata per migliorare l'efficienza della fase di training in quanto la correlazione tra campioni può influire negativamente sulla convergenza dell'algoritmo e sulla stabilità di apprendimento.
[36] Processo che rispetta la proprietà di Markov.

definite in termini di aumento, diminuzione o stazionarietà di h, ovvero nelle tre possibilità di acquisto, vendita o mantenimento della posizione per ogni titolo considerato; la ricompensa $r(s, a, s')$ ottenuta dall'agente è data dalla variazione della sua ricchezza ovvero nella differenza tra $p_t h + b$ tra lo stato *s* e lo stato successivo *s'* in conseguenza all'azione *a* intrapresa. Il modello actor-critic utilizzato prevede una rete actor $\mu(s|\theta^{\mu})$ con la funzione di analizzare lo stato corrente e identificare l'azione *a* da intraprendere dato lo stato *s* in cui si trova secondo i parametri di rete θ^{μ} mentre la rete critic $Q(s, a|\theta^{Q})$ restituisce i valori di quelle azioni secondo il set di parametri θ^{Q}, per permettere l'esplorazione viene aggiunto un rumore all'output della rete actor campionato da un processo casuale N, l'architettura della rete è presentata dalla seguente Figura 14.

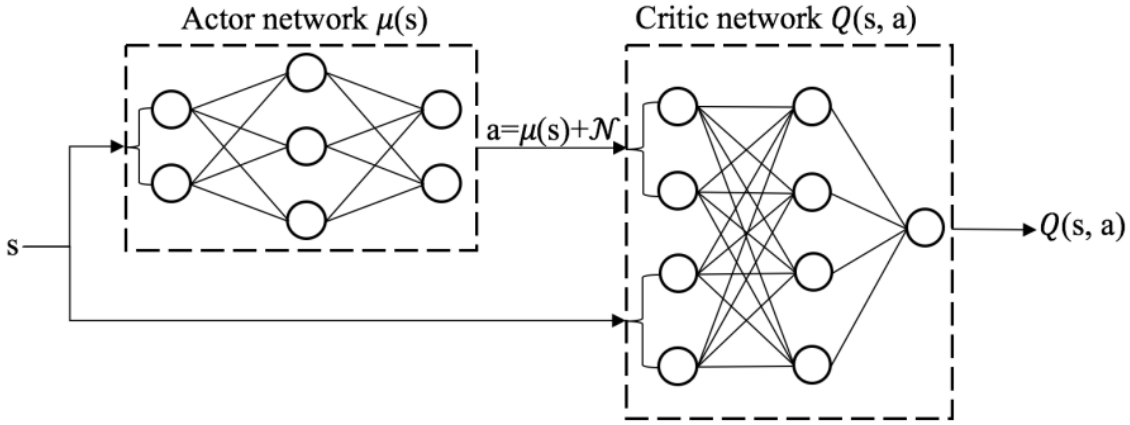

Figura 14 architettura della rete actor-critic (Liu et al., 2019).

Viene infine creata una rete gemella *Q'* e *μ'* utilizzata come rete target con la funzione di fornire backup temporali per l'aggiornamento dei parametri di rete. Entrambe le reti vengono aggiornate in modo iterativo. Ad ogni istante, l'agente DDPG esegue un'azione su s_t, quindi riceve una ricompensa basata su s_{t+1}. La transizione $(s_t, a_t, s_{t+1}, r_{t+1})$ viene quindi memorizzata nel buffer di riproduzione *R*. Le *N* transizioni estratte da *R* vengono quindi utilizzate per calcolare $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu}, \theta^{Q'})\right), i = 1, \dots, \mathrm{N}$ utilizzata per aggiornare la rete critic il cui processo di aggiornamento prevede la minimizzazione della differenza attesa $L(\theta^{Q})$ calcolata come differenza tra gli output delle reti *Q'* e *Q*

$$L(\theta^{Q}) = E_{s_t, a_t, r_{t+1}, s_{t+1} \sim buffer}[(y_t - Q(s_t, a_t|\theta^{Q})^2]$$

(3. 1)

Il modello proposto, testato nel periodo che va dal 04/01/2016 al 28/09/2018 ha offerto un rendimento cumulato del 97.91%, un rendimento annualizzato del 25.87% e uno Sharpe ratio di 1.79, contro un rendimento cumulato del DJIA del 54.28%, con un rendimento annualizzato del 16.4% e uno Sharpe ratio di 1.27.

3.3 Gestione di portafogli

Nei precedenti paragrafi sono stati esposti alcuni lavori che, personalmente, ritenevo interessanti per l'idea sottostante lo sviluppo del modello o per la formalizzazione del modello stesso senza però aver modo di riportare dettagliatamente i lavoro svolto. In questa parte vengono invece analizzati in modo più approfondito modelli sviluppati per la gestione dei portafogli, in quanto più inerenti alla parte sperimentale della tesi che verrà presentata nel prossimo capitolo, andando a riportare con maggior precisione le caratteristiche che il modello presenta. I modelli scelti per l'analisi sono i modelli proposti da Jiang, Xu e Liang (2017) in quanto parte dell'algoritmo sviluppato per la sezione sperimentale della presente tesi è stata sviluppata prendendo ispirazione da quanto presentato nel loro paper "*A deep reinforcement learning framework for the financial portfolio management problem*". Rispetto ai modelli di trading visti in precedenza i modelli di gestione dei portafogli presentano una difficoltà aggiuntiva in quanto la singola azione *a* presa al tempo *t* è, di fatto, divisibile in un insieme di differenti azioni di acquisto e vendita scelte contemporaneamente per i titoli considerati all'interno del problema di gestione di portafoglio.

Jiang, Xu e Liang (2017) propongono un algoritmo per la gestione di un portafoglio finanziario implementato poi con tre diverse strutture di DL: una rete CNN (*Convolutional Neural Network*), una rete costruita con layer di convoluzione[37] utilizzata solitamente per il riconoscimento di immagini grazie alla capacità di apprendere gerarchie e pattern spaziali ma che ha dimostrato ottime potenzialità anche nell'analisi di serie storiche (Zhao et al., 2017); una rete RNN (*Recurrent Neural Network*), utilizzata soprattutto per dati sequenziali e serie temporali grazie alla capacità di identificare le dipendenze sequenziali nei dati di input, caratteristica distintiva è la capacità di mantenere una memoria a breve termine delle informazioni passate tramite un meccanismo noto

[37] Operazione che combina due funzioni matematiche per ottenerne una terza, nell'ambito delle reti CNN l'operazione permette di applicare filtri o kernel a immagini di input per estrarne le caratteristiche spaziali.

come *stato ricorrente*; una rete LSTM (*Long-Short Term Memory*), una variante della RNN progettata per ovviare al problema di scomparsa, o esplosione, del gradiente tipico delle RNN, grazie ad una struttura di memoria più complessa.

Il framework proposto presenta tre elementi fondamentali: una struttura EIIE (*Ensemble of Identical Independent Evaluators*), un vettore di memoria del portafoglio (PVM, *Portfolio Vector Memory*) e un modello di allenamento in grado di aggiornare online i batch di allenamento della rete (OSBL, *Online Stocastic Batch Learning scheme*).

La struttura EIIE prevede che la valutazione di ogni titolo inserito nel portafogli venga effettuata separatamente dagli altri, questa struttura presenta tre vantaggi fondamentali rispetto ad una rete integrata: riduzione dei tempi di allenamento del modello e maggior efficienza dell'allenamento, per ogni intervallo di tempo l'allenamento dei parametri avviene per *m* volte, dove *m* è il numero di asset considerati dal modello, rendendo possibile utilizzare serie storiche di dimensioni inferiori; infine la capacità di valutare gli asset è universale e non vincolata ad asset specifici permettendo l'aggiornamento delle scelte dell'agente in tempo reale e senza dover addestrare la rete dall'inizio. L'utilizzo di una struttura EIIE permette quindi l'analisi separata degli *m* asset, come se si trattasse di *m* reti neurali separate, ma allo stesso tempo i parametri della rete sono comuni, questo permette all'algoritmo di valutare le prospettive di ogni asset senza che queste vengano influenzate dall'andamento degli altri ma, al contempo, la condivisione dei parametri per tutti gli IIE garantisce l'identificazione di una policy generale valida per tutti gli asset considerati.

Il vettore di memoria PVM viene utilizzato per immagazzinare le informazioni circa le azioni passate dell'agente, essendo lo stato s_t definito da un tensore dei prezzi X_t, composto dai valori normalizzati di chiusura, massimo e minimo per ogni asset e dai pesi di portafogli al tempo precedente w_{t-1}, quindi,

$$s_t = (X_t, w_{t-1})$$

(3. 2)

è necessario che i pesi di portafoglio selezionati dall'agente nei periodi precedenti siano memorizzati dall'agente per poi essere utilizzati nella selezione dell'azione; la memorizzazione dei pesi di portafoglio precedenti permette all'agente di identificare una strategia di selezione di portafoglio più stabile nel lungo periodo, evitando di sconvolgere

le allocazioni di capitale negli asset, inoltre far sì che la rete mantenga in memoria l'ultima allocazione di capitale è necessario affinché siano calcolabili i costi di transazione come verrà esposto in seguito. Il funzionamento del PVM è graficamente rappresentato dalla Figura 15 in seguito riportata, In entrambi i grafici riportati in figura, una piccola striscia verticale sull'asse temporale rappresenta una porzione della memoria contenente i pesi del portafoglio all'inizio di un periodo. Le memorie rosse vengono lette dalla rete per definire un'azione, mentre quelle blu vengono sovrascritte dalla rete all'interno del PVM. I due rettangoli colorati in Figura 15 (a), composti da quattro strisce, sono esempi di due mini-batch consecutivi. Mentre la Figura 15 (a) mostra un ciclo completo di lettura e scrittura per un mini-batch, la Figura 15 (b) mostra un ciclo di aggiornamento del PVM all'interno di una rete (omettendo la parte precedente della rete).

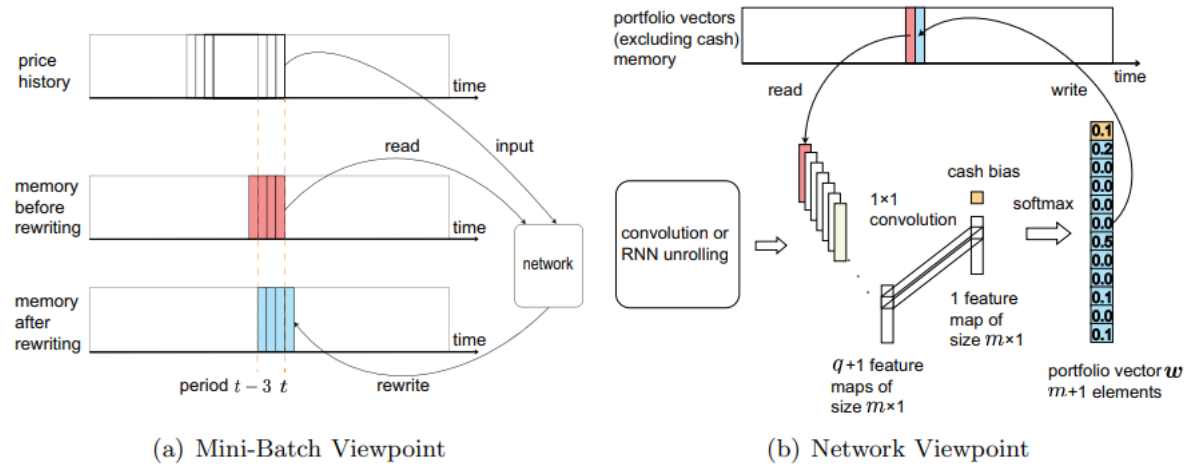

Figura 15 ciclo lettura/scrittura del PVM (Jiang, Xu e Liang ,2017).

Infine, il modello di allenamento OSLB sviluppato in questo framework rende possibile l'allenamento della rete per *mini-batch*,[38] a differenza di un modello di allenamento standard i dati contenuti nei mini-batch devono essere sequenziali per rappresentare l'evoluzione delle serie storiche dei prezzi. L'allenamento online prevede che al termine di un periodo *t*, le variazioni dei prezzi di questo periodo vengono inclusi nei dati di allenamento, in questo modo saranno considerati dall'agente nel periodo *t+1*; una volta che l'agente ha completato gli ordini nel periodo *t+1* la policy seguita verrà allenata confrontandola con N_b mini-batch di dimensione n_b tali per cui il primo dato contenuto

[38] Un mini-batch è, in generale, un sottoinsieme casuale di esperienze prelevate da un buffer di riproduzione che conserva la storia passata delle esperienze dell'agente nella forma $(s_t, a_t, r_{t+1}, s_{t+1})$.

sia ad un tempo $t_b \leq t - n_b$ selezionato secondo la probabilità $P_\beta(t_b)$ distribuita geometricamente come

$$P_\beta(t_b) = \beta(1-\beta)^{t-t_b-n_b}$$

(3. 3)

in cui $\beta \in (0,1)$ è il tasso di decadimento della probabilità che determina la forma della distribuzione di probabilità, quindi, all'aumentare del valore di β viene data maggiore importanza agli eventi di mercato più recenti, andando quindi a selezionare con maggiore probabilità un mini-batch il cui tempo di inizio è più vicino a $t_b \leq t - n_b$.

Come esposto in precedenza lo stato s_t che l'agente riceve come input è definito come $s_t = (X_t, w_{t-1})$, in cui w_{t-1} identifica il vettore dei pesi di portafoglio definiti al tempo precedente, mentre X_t è un tensore contenente i valori di mercato normalizzati, questo tensore è composto dalla concatenazione delle tre matrici V_t, V_t^{hi}, V_t^{lo} rappresentati rispettivamente le ultime *n* osservazioni dei valori di chiusura, massimo e minimo degli asset considerati, normalizzati secondo le seguenti equazioni

$$\begin{aligned}
\boldsymbol{V}_t &= \left[\boldsymbol{v}_{t-n+1} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_{t-n+2} \oslash \boldsymbol{v}_t \middle| \cdots \middle| \boldsymbol{v}_{t-1} \oslash \boldsymbol{v}_t \middle| \mathbf{1}\right], \\
\boldsymbol{V}_t^{(\mathrm{hi})} &= \left[\boldsymbol{v}_{t-n+1}^{(\mathrm{hi})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_{t-n+2}^{(\mathrm{hi})} \oslash \boldsymbol{v}_t \middle| \cdots \middle| \boldsymbol{v}_{t-1}^{(\mathrm{hi})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_t^{(\mathrm{hi})} \oslash \boldsymbol{v}_t\right], \\
\boldsymbol{V}_t^{(\mathrm{lo})} &= \left[\boldsymbol{v}_{t-n+1}^{(\mathrm{lo})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_{t-n+2}^{(\mathrm{lo})} \oslash \boldsymbol{v}_t \middle| \cdots \middle| \boldsymbol{v}_{t-1}^{(\mathrm{lo})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_t^{(\mathrm{lo})} \oslash \boldsymbol{v}_t\right],
\end{aligned}$$

(3. 4)

Ovvero, per ogni titolo, ogni valore di chiusura, massimo e minimo viene espresso come il rapporto tra il suo valore e il prezzo di chiusura al tempo corrente v_t. La concatenazione delle matrici V_t, V_t^{hi}, V_t^{lo} genera quindi il tensore tridimensionale X_t.

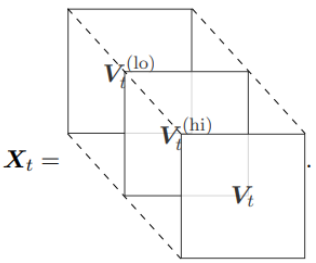

Figura 16 rappresentazione grafica del tensore X_t (Jiang, Xu e Liang ,2017).

La finestra temporale in cui opera l'agente è divisa in periodi di 30 minuti e compito dell'agente è, all'inizio di ogni periodo *t*, riallocare le risorse nel portafogli in modo da massimizzare il rendimento al termine dell'intera finestra temporale considerata (50 giorni); il compito dell'agente è quindi massimizzare il valore della ricompensa finale *R* calcolata come:

$$R\left(s_1, a_1, \cdots s_{t_f}, a_{t_f}, s_{t_f+1}\right) = \frac{1}{t_f}\sum_{t=1}^{t_f+1} r_t$$

(3. 5)

Dove t_f indica il numero di periodi di uguali dimensioni *t* in cui è divisa la finestra temporale in cui l'agente opera mentre r_t indica il rendimento dei portafogli per ogni periodo calcolato come il logaritmo del rapporto tra il valore di portafogli *p* al tempo *t* e al tempo $t-1$ come da seguente equazione, in cui il valore di portafogli al termine del periodo *t* è espresso come $p_t = p_{t-1} y_t \cdot w_{t-1}$

$$r_t = \ln\frac{p_t}{p_{t-1}} = \ln(\mu_t y_t \cdot w_{t-1})$$

(3. 6)

nella seconda uguaglianza dell'equazione (3.6) y_t è il vettore contente i rendimenti degli asset al termine del periodo *t*, calcolati come il rapporto $\frac{v_t}{v_{t-1}}$, per ogni asset, w_{t-1} rappresenta i pesi di portafoglio definiti all'inizio del periodo *t* e μ_t rappresenta i costi di transazione. I costi di transazione sono calcolati secondo l'equazione

$$\mu_t = \frac{1}{1 - c_p w_{t,0}}\left[1 - c_p w'_{t,0} - \left(c_s + c_p - c_s c_p\right)\sum_{i=1}^{m}\left(w'_{t,1} - \mu_t w_{t,i}\right)^+\right]$$

(3. 7)

in cui c_p e c_s rappresentano i costi di commissione per rispettivamente l'acquisto e la vendita di asset, $w'_{t,1}$ indica i pesi di portafogli per l'asset *i* prima che vengano applicati i costi di transazione e $(v)^+$indica l'applicazione di una funzione ReLU per v. Essendo il fattore μ_t interno ad una funzione ReLU l'equazione si può risolvere solo iterativamente, per costi di commissione di vendita e acquisto uguali μ_t può essere approssimato come

$c\sum_{i=1}^{m}|w'_{t,i} - w_{t,i}|$ ovvero l'applicazione della commissione *c* alla differenza dei pesi di portafogli.[39]

La dinamica della variazione di portafogli si può illustrare come in Figura 17 dove p'_t indica il valore di portafogli prima che vengano applicati i costi di transazione.

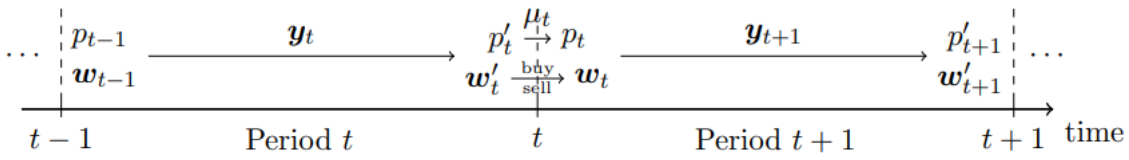

Figura 17 rappresentazione grafica del processo di allocazione di portafoglio (Jiang, Xu e Liang ,2017).

L'agente quindi, nel corso del periodo *t*, dall'allocazione di portafogli w_{t-1} ottiene il rendimento r_t, contenuto nel vettore y_t, come da equazione (3.6), arrivato al termine del periodo *t*, il portafogli, data la variazione del valore dei titoli, presenta i pesi w'_t e valore p'_t. Al termine del periodo *t* l'agente deve riallocare i pesi di portafoglio per ottenere il nuovo portafogli w_t applicando i costi di commissione alla transazione dai pesi w'_t ai pesi w_t si ottiene il nuovo valore di portafogli $p_t = p'_t\mu_t$. Da notare come il valore di portafogli tra la fine del periodo *t* e l'inizio del periodo *t+1* varia soltanto dell'ammontare dei costi di transazione.

Per quanto riguarda la formalizzazione informatica le azioni dell'agente sono esprimibili in termini di pesi di portafoglio per cui $a_t = w_t$ e sono volte a massimizzare la funzione di ricompensa espressa all'equazione (3.5), in cui $\frac{r_t}{t_f}$ rappresenta la ricompensa immediata dell'agente al termine di un singolo periodo *t*, allo stesso tempo il valore $\frac{r_t}{t_f}$ viene utilizzato come valore dell'azione $a_t = w_t$, con fattore di sconto pari a 0. La policy ottima viene identificata dall'agente mediante un algoritmo di discesa del gradiente, la policy viene quindi definita secondo un insieme di parametri θ e l'azione intrapresa al tempo *t* può quindi essere definita in funzione della policy π_θ e dello stato s_t, quindi $a_t = \pi_\theta(s_t)$. La misura di performance $J_{[0,t_f]}$ della policy π_θ ottenuta nell'intervallo temporale $[0, t_f]$ è data dalla funzione di compensa *R* per l'intervallo specificato, quindi

[39] La formulazione completa è riportata al Teorema 1 di Jiang, Xu e Liang (2017).

$$J_{[0,t_f]}(\pi_\theta) = R\left(s_1, \pi_\theta(s_1), \cdots, s_{t_f}, \pi_\theta\left(s_{t_f}\right), s_{t_f+1}\right)$$

(3. 8)

I parametri θ vengono inizializzati casualmente e poi aggiornati con il metodo di discesa del gradiente con tasso di apprendimento λ, il processo di aggiornamento avviene quindi secondo la dinamica

$$\theta \to \theta + \lambda \nabla_\theta J_{[0,t_f]}(\pi_\theta)$$

(3. 9)

Per potenziare il processo di allenamento l'aggiornamento dei parametri non avviene sull'intero processo ma viene effettuato su mini-batch, l'intervallo di dati su cui avviene l'aggiornamento nei mini-batch è quindi $[t_{b_1}, t_{b_2}]$, la dinamica di aggiornamento può quindi essere espressa come

$$\theta \to \theta + \lambda \nabla_\theta J_{[t_{b_1}, t_{b_2}]}(\pi_\theta)$$

(3. 10)

La topologia della prima rete CNN è riportata alla Figura 20, l'input della rete è rappresentato dal tensore X_t presentato in precedenza di dimensioni (3,11,50) ovvero 3 features per 11 titoli per 50 osservazioni; questi input vengono trasformati mediante convoluzione 1×3 e, a seguito di un funzione di attivazione ReLU, integrati in due mappe di dimensioni 11×48. Questo significa che ogni valore ricevuto in input il secondo layer della rete neurale, identificato dalle due mappe di feature 11×48, viene generato dalla convoluzione di tre valori sequenziali di input (ad esempio per l'asset *i* il primo valore ricevuto dal secondo layer è la convoluzione dei valori $(v_{t,i}, v_{t+1,i}, v_{t+2,i})$, il secondo valore è derivato dalla convoluzione degli input $(v_{t+1,i}, v_{t+2,i}, v_{t+3,i})$ e così via). I dati ricevuti dal secondo layer vengono nuovamente trasformati con convoluzione 1×48 e passati così al terzo layer, che riceve come input anche gli ultimi 20 pesi di portafoglio conservati nel PVM il cui funzionamento è stato descritto in precedenza ed è graficamente riportato dalla precedente Figura 17. Il quarto e penultimo layer viene definito *scoring layer*, la sua funzione è, dopo aver ricevuto l'informazione *cash bias*, ovvero l'informazione circa la disponibilità di liquidità dell'agente al tempo *t*, dare per l'appunto un punteggio di appetibilità ad ogni asset compreso il denaro liquido; infine

l'output della rete, quindi la composizione di portafoglio che secondo l'agente massimizza la ricompensa, viene definita mediante un *layer softmax*.[40]

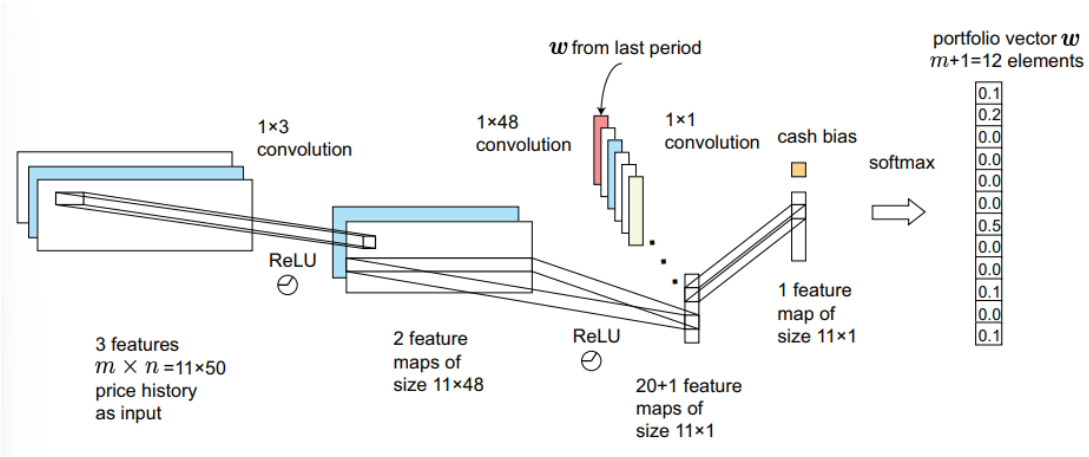

Figura 2 implementazione della rete EIIE con rete CNN (Jiang, Xu e Liang ,2017).

La Figura 20 raffigurante la topologia della rete permette di notare come funzioni nell'atto pratico la rete EIIE, Ogni riga della rete è associata a un asset specifico e fornisce un punteggio di voto al layer softmax, soltanto in quest'ultimo layer la rete utilizza contemporaneamente le informazioni degli 11+1 asset per definire i pesi di portafoglio.

La topologia delle reti RNN e LSTM sono identiche e riportate in Figura 21. Questa rappresenta un approccio più classico alla gestione di serie storiche in quanto viene utilizzata per intero la serie fornita in input, le 50 osservazioni per ogni feature per ogni asset, per identificare le relazioni che intercorrono tra i valori della serie, identificare quindi i pattern che la caratterizzano e restituire un valore indicativo della potenzialità di crescita del valore di ogni asset. Come si nota dalla Figura 21 per ogni asset le osservazioni delle tre feature a ogni istante temporale *t* vengono processate da una rete neurale ricorrente per fornire poi un input al terzo layer, come per la rete CNN i dati relativi ogni asset vengono trattati singolarmente e indipendentemente fino all'ultimo layer di output. Le differenze tra la rete CNN e le reti RNN e LSTM riguardano il processo di trattamento iniziale dei dati di input al fine di fornire un valore indicativo della

[40] I layer softmax vengono generalmente utilizzati per la produzione di probabilità su più classi, in questo caso il layer fornisce di fatto la probabilità, per ogni riga, che il valore dell'asset associato ad una riga aumenti nel prossimo periodo, le probabilità fornite vengono quindi utilizzate come pesi di portafoglio. L'utilizzo di questo layer dipende dal fatto che fornendo un valore probabilistico la somma dei valori è sempre 1, permettendo così la completa allocazione delle risorse, e non sono ammessi valori negativi.

potenzialità di crescita, la seconda parte della rete neurale è identica per entrambi gli approcci considerati.

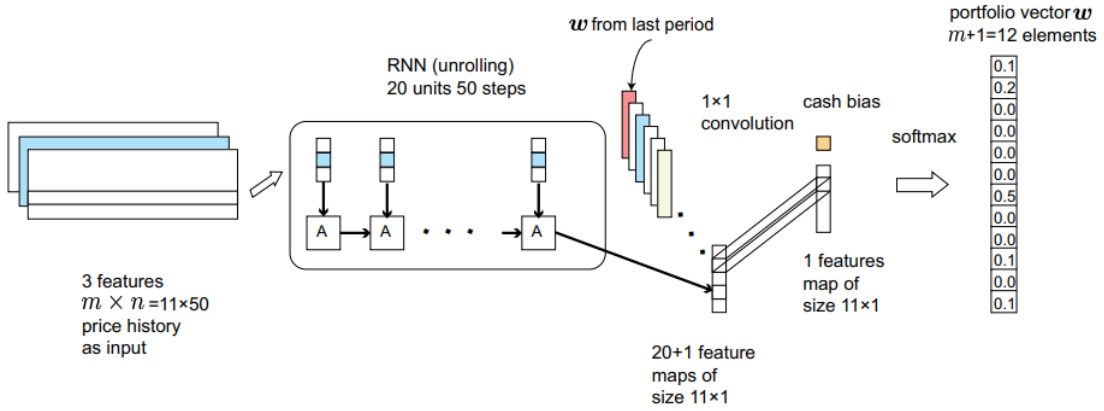

Figura 3 implementazione della rete EIIE con rete CNN (Jiang, Xu e Liang ,2017).

I framework proposti sono stati testati su 11 diverse criptovalute più il Bitcoin che funge da denaro contante, in tre diversi periodi della durata di 50 giorni l'uno, e valutati con tre misure di performance: il valore cumulato finale di portafoglio (fAPV), il valore di indice di Sharpe (SR) ed il valore di Maximum Drowdown (MDD). Il valore cumulato finale di portafoglio è espresso come rapporto tra il valore finale p_f[41] ed il valore iniziale p_0, quest'ultimo posto uguale a 1

$$\frac{p_f}{p_0} = \frac{p_{t_f+1}}{p_0}$$

(3. 11)

Il valore dell'indice di Sharpe, o Sharpe ratio, viene espresso come la media della differenza tra i rendimenti di portafoglio ρ_t e il rendimento privo di rischio ρ_F fratta per la deviazione standard del rendimento di portafogli; in questo framework il titolo privo di rischio è l'asset rappresentante il denaro contante quindi $\rho_F = 0$.

$$SR = \frac{E_t[\rho_t - \rho_F]}{\sqrt{var_t(\rho_t - \rho_F)}}$$

(3. 12)

[41] Da notare come il valore finale di portafoglio p_f sia effettivamente disponibile al tempo $t_f + 1$, questo per la logica temporale evidenziata in Figura 5.

Per il calcolo dell'indice di Sharpe gli autori non utilizzano il rendimento logaritmico r_t utilizzato per la funzione di ricompensa e il calcolo del valore di portafogli ma utilizzano la variazione del valore di portafogli per ogni step temporale ρ_t espresso come

$$\rho_t = \frac{p_t}{p_{t-1}} - 1$$

(3. 13)

Infine il valore di Maximum Drawdown viene espresso come la perdita massima registrata dal portafoglio nel periodo d'investimento, matematicamente espressa come

$$MDD = \max_{\tau>t} \frac{p_t - p_\tau}{p_t}$$

(3. 14)

Queste ultime due misure vengono utilizzate per avere una valutazione del rendimento di portafogli corretta per il rischio, per l'indice di Sharpe il rischio viene espresso in termini di varianza dei rendimenti mentre per il Maximum Drawdown viene espresso in termini di perdite subite. Le performance registrate dai tre modelli vengono quindi confrontate e con i rendimenti realizzati da una rete neurale CNN non basata su EIIE e da altri algoritmi di trading automatico. Alla Tabella 1 sono riportati i risultati ottenuti, gli algoritmi di trading utilizzati per il confronto sono divisi in strategie utilizzate come benchmark (Best Stock,[42] portafoglio equipesato[43] e portafoglio ribilanciato uniformemente in modo costante)[44] algoritmi con strategia *follow-the-leader*, algoritmi con strategia *follow-the-loser* e altri algoritmi. Le prime quattro righe riportano le performance registrate dai tre algoritmi esposti in precedenza, evidenziati in grassetto, e le performance registrate dalla rete CNN non basata su EIIE.

[42] Ovvero il rendimento realizzato se si fosse investito tutto il capitale nel titolo che ha dato il maggior rendimento

[43] Portafogli in cui il capitale iniziale viene equamente distribuito e non viene più ribilanciato, nella tabella con l'acronimo UBAH.

[44] Traduzione letterale di "uniform constant rebalanced portfolio", ovvero un portafoglio periodicamente ribilanciato per far si che le proporzioni d'investimento restino sempre uguali, nella tabella con l'acronimo UCRP.

	2016-09-07 to 2016-10-28			2016-12-08 to 2017-01-28			2017-03-07 to 2017-04-27		
Algorithm	MDD	fAPV	SR	MDD	fAPV	SR	MDD	fAPV	SR
CNN	0.224	**29.695**	**0.087**	0.216	**8.026**	**0.059**	0.406	31.747	0.076
bRNN	0.241	13.348	0.074	0.262	4.623	0.043	0.393	**47.148**	**0.082**
LSTM	0.280	6.692	0.053	0.319	4.073	0.038	0.487	21.173	0.060
iCNN	**0.221**	4.542	0.053	0.265	1.573	0.022	0.204	3.958	0.044
Best Stock	0.654	1.223	0.012	0.236	1.401	0.018	0.668	4.594	0.033
UCRP	0.265	0.867	-0.014	**0.185**	1.101	0.010	**0.162**	2.412	0.049
UBAH	0.324	0.821	-0.015	0.224	1.029	0.004	0.274	2.230	0.036
Anticor	0.265	0.867	-0.014	0.185	1.101	0.010	0.162	2.412	0.049
OLMAR	0.913	0.142	-0.039	0.897	0.123	-0.038	0.733	4.582	0.034
PAMR	0.997	0.003	-0.137	0.998	0.003	-0.121	0.981	0.021	-0.055
WMAMR	0.682	0.742	-0.0008	0.519	0.895	0.005	0.673	6.692	0.042
CWMR	0.999	0.001	-0.148	0.999	0.002	-0.127	0.987	0.013	-0.061
RMR	0.900	0.127	-0.043	0.929	0.090	-0.045	0.698	7.008	0.041
ONS	0.233	0.923	-0.006	0.295	1.188	0.012	0.170	1.609	0.027
UP	0.269	0.864	-0.014	0.188	1.094	0.009	0.165	2.407	0.049
EG	0.268	0.865	-0.014	0.187	1.097	0.010	0.163	2.412	0.049
B^K	0.436	0.758	-0.013	0.336	0.770	-0.012	0.390	2.070	0.027
CORN	0.999	0.001	-0.129	1.000	0.0001	-0.179	0.999	0.001	-0.125
M0	0.335	0.933	-0.001	0.308	1.106	0.008	0.180	2.729	0.044

Tabella 1 confronto delle performance registrate dai modelli di DL proposti e altre strategie di trading.

I risultati riportata in Tabella 1 evidenziano come per tutti e tre i periodi in cui gli algoritmi di DL sviluppati sono stati testati questi hanno registrato performance nettamente superiori agli altri algoritmi di trading proposti per le metriche di SR e fAPV, per quanto riguarda la metrica MDD invece gli algoritmi di DL sviluppati presentano performance positive ma non particolarmente superiori agli indicatori di benchmark. Interessante notare come la rete CNN abbia registrato performance nettamente superiori alle reti LSTM e RNN in due dei tre periodi temporali considerati, questo può dimostrare come un modello di gestione di portafoglio, o comunque di trading, basato sulla previsione dei prezzi futuri dalle serie storiche dei titoli possa presentare delle limitazioni rispetto a modelli in cui la valutazione non è necessariamente legata ad un modello di previsione dei valori. Infine è interessante notare come la rete RNN restituisce sempre performance migliori rispetto alla rete LSTM, questo può essere dovuto alla struttura di memoria di quest'ultima che la rende meno efficiente rispetto ad una rete RNN standard in questo specifico problema; il vantaggio di una rete LSTM risiede infatti nella maggior capacità di identificare e gestire dipendenze a lungo termine, è invece pensiero comune considerare, nelle serie storiche finanziarie, di maggior importanza gli eventi recenti rispetto a quelli più lontani nel tempo, evidentemente la struttura di memoria e aggiornamento delle previsioni più complessa prevista dalla rete LSTM si è dimostrata svantaggiosa rispetto ad un modello più semplice.

3.4 Impiego pratico del machine learning nel settore finanziario

Lo sviluppo tecnologico e l'accesso ai cosiddetti *Big Data* hanno avuto un grande impatto nel settore finanziario non solo dal punto di vista della ricerca accademica, ma anche nello sviluppo e ricerca, da parte delle aziende operanti del settore, di tecnologie che possano garantire un vantaggio sui competitor. Il campo del *Fintech* si pone quindi come un settore dalle enormi potenzialità per aumentare l'efficienza e ridurre i costi delle aziende operanti nel settore finanziario ma non solo, l'applicazione del machine learning e di tecniche di intelligenza artificiale nel tempo hanno acquisito un ruolo fondamentale anche nella gestione degli aspetti più tecnici del settore come ad esempio nella valutazione delle garanzie, nella valutazione del rischio di credito e nella previsione dei ricavi futuri (Bazarbash, 2019).

Con riferimento al mercato domestico sfortunatamente il report più recente riporta l'utilizzo di tecniche di machine learning e intelligenza artificiale per l'anno 2020 quindi i dati riportati possono presentare notevoli differenze con quello che è l'effettivo utilizzo odierno di queste tecniche. I dati riportati fanno riferimento al report "*Rilevazione sull'IT nel settore bancario italiano*" redatto dai centri studi CIPA (Convenzione Interbancaria per l'Automazione) e ABI (Associazione Bancaria Italiana). Dal grafico riportato in Figura 20 si può notare come al 2020 il 56% delle società operanti nel settore finanziario avesse implementato soluzioni di AI, ma soltanto il 12% facesse effettivamente uso rilevante di queste, è comunque previsto un trend in crescita nell'adozione di queste tecniche; il livello di applicazione considerato "Alto" era infatti previsto in forte crescita nel triennio 2021-2023 e questo trend è previsto rafforzarsi anche in futuro.

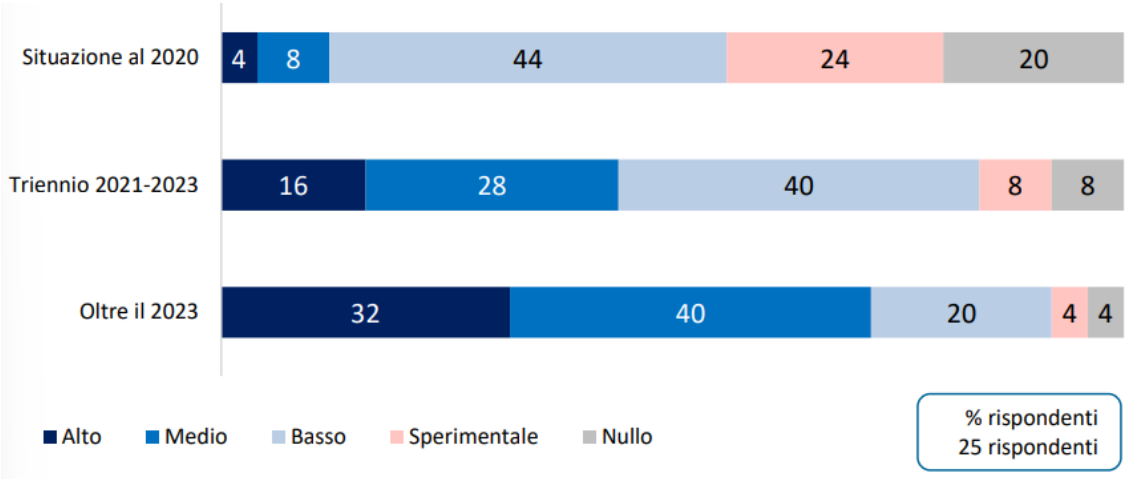

Figura 20 adozione di soluzioni basate sull'intelligenza artificiale (CIPA e ABI, 2021).

Le soluzioni di AI adottate si concentrano maggiormente nell'area marketing e servizi al cliente in cui il 22% dei rispondenti presenta livelli di applicazione tra il medio e l'alto, seguito dai processi di operazioni, mentre per i processi di governance e supporto l'utilizzo di questi processi è piuttosto limitato.

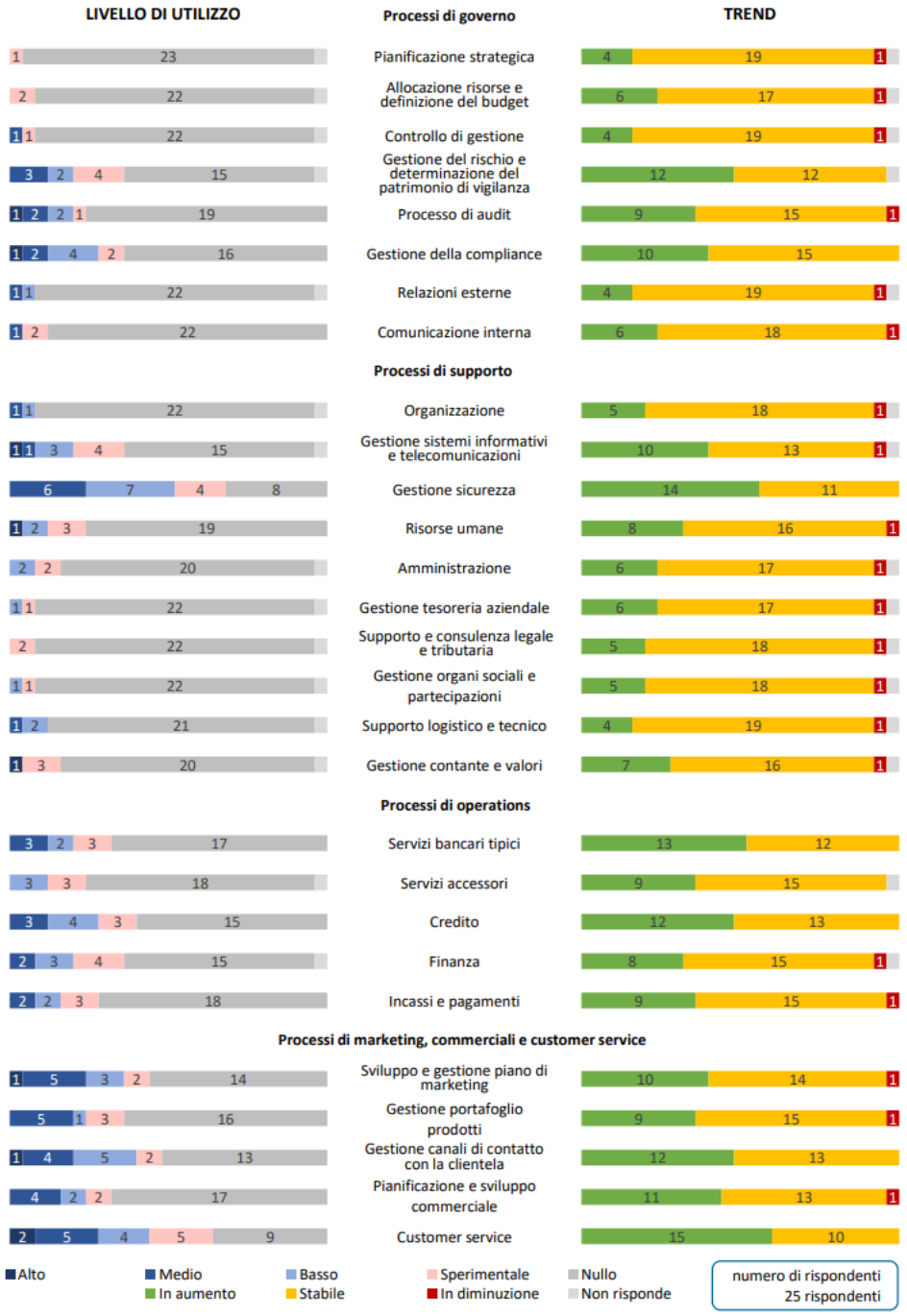

Figura 21 Utilizzo dell'intelligenza artificiale per singolo processo (CIPA e ABI, 2021).

In Figura 21 sono riportati i gradi di utilizzo, e la previsione dei trend di utilizzo, circa l'AI per singolo processo aziendale, la funzione di servizio alla clientela vede l'uso più

intensivo di questa, seguita dalla gestione della sicurezza, proprio queste due funzioni sono quelle per cui sono previsti i maggiori aumenti nelle applicazioni, dimostrando come le soluzioni ad ora adottate hanno dimostrato un buon livello di efficienza ed affidabilità. In generale le applicazioni di AI si concentrano prevalentemente nelle funzioni in cui la gestione e l'analisi di dati nonché la creazione di modelli previsionali ricoprono ruoli di notevole importanza quali ad esempio le funzioni di credito, monitoraggio del patrimonio di vigilanza e funzioni di finanza, tutte funzioni in cui è previsto un sostanziale aumento delle applicazioni. Per quanto concerne invece le funzioni di governance e strategiche l'utilizzo dell'IA sembra destinato a ricoprire un ruolo marginale all'interno del settore finanziario italiano.

Un aspetto interessante da analizzare riguarda le criticità riscontrate dalle aziende intervistate, che già hanno implementato modelli di AI, circa l'applicazione di questi. La gestione più strettamente informatica presenta una delle principali criticità in quanto la complessità di realizzazione, la quantità di impegno e tempo richiesto per l'addestramento dei modelli sembrano essere tre delle principali criticità riscontrate, anche la difficoltà nell'ottenere dati di buona qualità in grande quantità si è dimostrata una criticità limitante per l'applicazione di queste tecnologie. Un ulteriore fattore critico riscontrato dalle aziende riguarda il rispetto del quadro normativo, in particolare con riferimento all'aderenza alla normativa sulla privacy vigente.

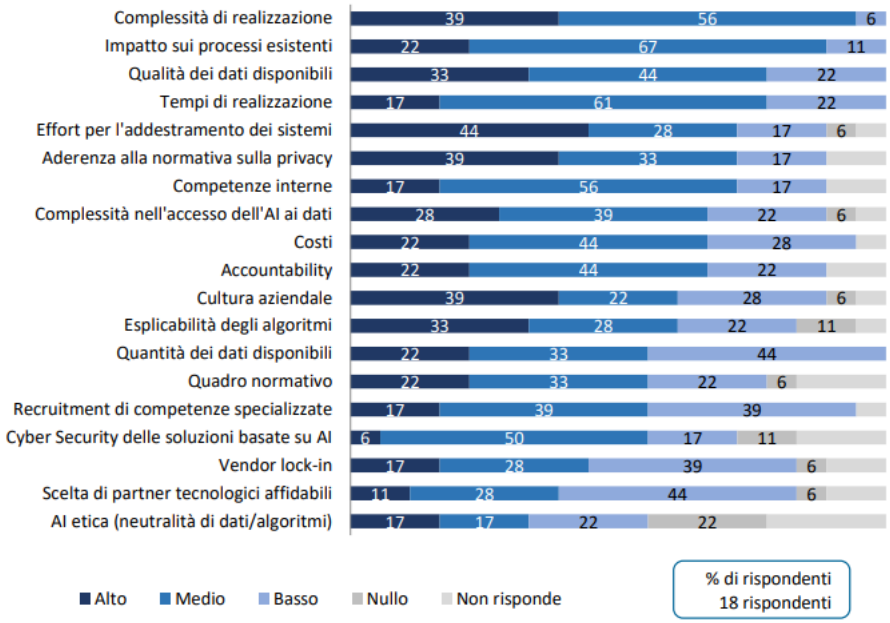

Figura 4 criticità riscontrate nell'adozione dell'AI (CIPA e ABI, 2021).

Per avere una proxy più recente circa l'uso di queste tecnologie nel settore finanziario si è utilizzato il report "*Machine learning in UK financial services*" redatto da *Bank of England*. Secondo suddetto report, strutturato come il precedente come un sondiaggio, il 72% delle aziende britanniche operanti nell'ambito finanziario ha utilizzato soluzioni di machine learning nel 2022, come riportato in Figura 23, rispetto al 67% registrato nel 2019. Il settore bancario è quello che vede il maggior numero di applicazioni di ML, seguito dal settore assicurativo. Un numero minore di applicazioni vede coinvolti invece i prestatori non bancari, le società di gestione di capitale e le società FMI (*Financial Market Infrastructures* di cui fanno parte, ad esempio, sistemi di pagamento e regolamento, intermediari non bancari, ecc…) come mostrato in Figura 23. L'utilizzo di soluzioni di ML e AI è previsto in aumento nei prossimi anni, è infatti stimato che il numero mediano di applicazioni di queste aumenti di tre volte e mezzo tra tutti i settori, con un aumento previsto particolarmente marcato per il settore assicurativo in cui è previsto un aumento del 163% delle applicazioni, seguito dal settore bancario.

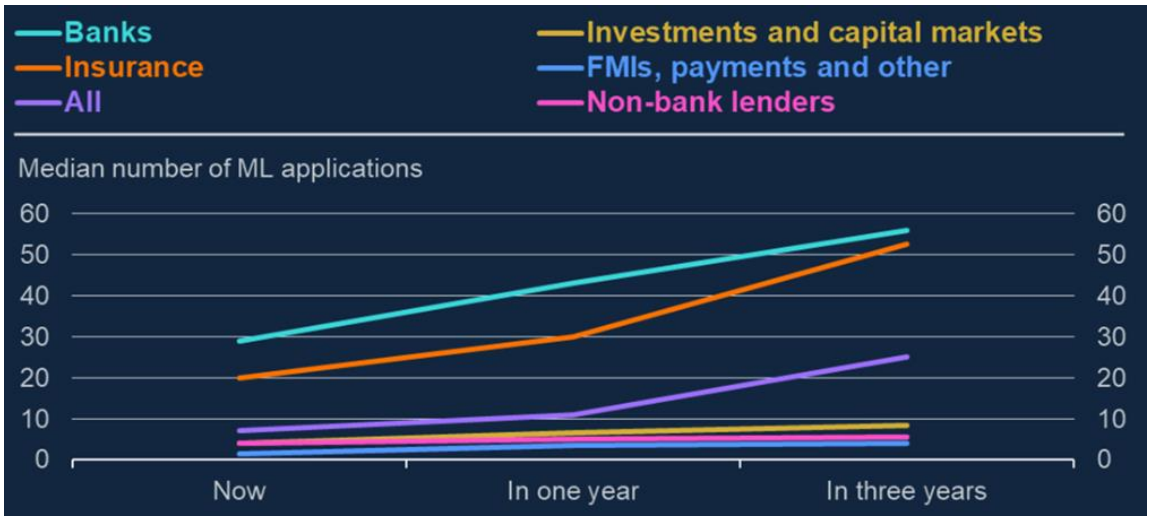

Figura 23 Numero mediano di applicazioni di tecniche di machine learning previste nei prossimi 3 anni (Machine learning in UK financial services, 2022).

Pur essendo il settore che vede il maggior numero di applicazioni attive ed in via di sviluppo il settore bancario è comunque quello in cui ad oggi il ML viene utilizzato, in proporzione, dal minor numero di società, al contrario il settore dei prestatori non bancari è quello che vede una maggior diffusione tra le società che ne fanno parte di tecniche di ML (Figura 24). L'elevato numero di società che già utilizzano ML nei settori dei

prestatori non bancari, gestori di capitale e FMI giustifica in parte il relativamente ridotto aumento previsto di applicazioni di questi processi.

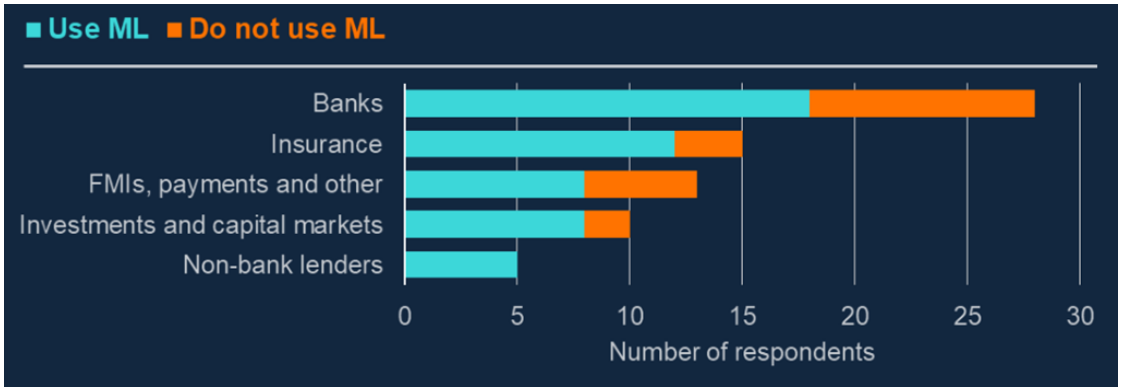

Figura 24 Quota di aziende operanti in ambito finanziario che utilizzano tecniche di machine learning (Machine learning in UK financial services, 2022).

Per quanto riguarda lo sviluppo e impiego del ML, il 79% delle applicazioni sviluppate nel settore finanziario sono già in fase di impiego, in particolare il 65% delle applicazioni è utilizzato in una quota considerevole di settori aziendali mentre il 14% è considerato impiegato in funzioni cruciali per l'area di business di riferimento. Per quanto riguarda lo sviluppo di nuove applicazioni solo il 10% di queste risulta essere nella fase finale di sviluppo, pre-impiego o comunque utilizzate in una quota limitata di settori aziendali; nel 2019 la quota di applicazioni in questa categoria era del 44% suggerendo che il ML ha assunto, negli ultimi tre anni, un ruolo sempre più preponderante nell'implementazione di funzioni aziendali (Figura 25). In particolare le aziende che fanno maggiore uso del machine learning in settori considerati essenziali per il business appartengono alla categoria di prestatori non bancari (42% delle applicazioni) e FMI (26% delle applicazioni) come esposto in Figura 26, questo spiega parzialmente i dati riportati al grafico in Figura 23, essendo soluzioni di ML già impiegate per quote considerevoli di business ulteriori sviluppi riguarderebbero aree marginali di business per cui lo sviluppo potrebbe non essere considerato una priorità. Di contro le società d'investimento sono le società in cui tecniche di ML vengono applicate prevalentemente in aree di business marginali evidenziando come queste soluzioni, per questo settore, non hanno ancora raggiunto un livello di affidabilità sufficiente perché possano essere impiegate su larga scala.

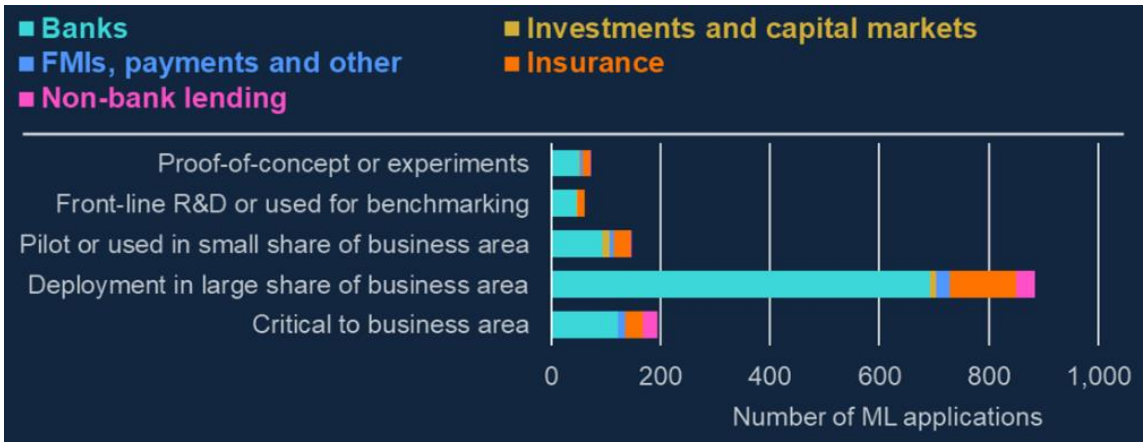

Figura 25 sviluppo e impiego di tecniche di machine learning (Machine learning in UK financial services, 2022).

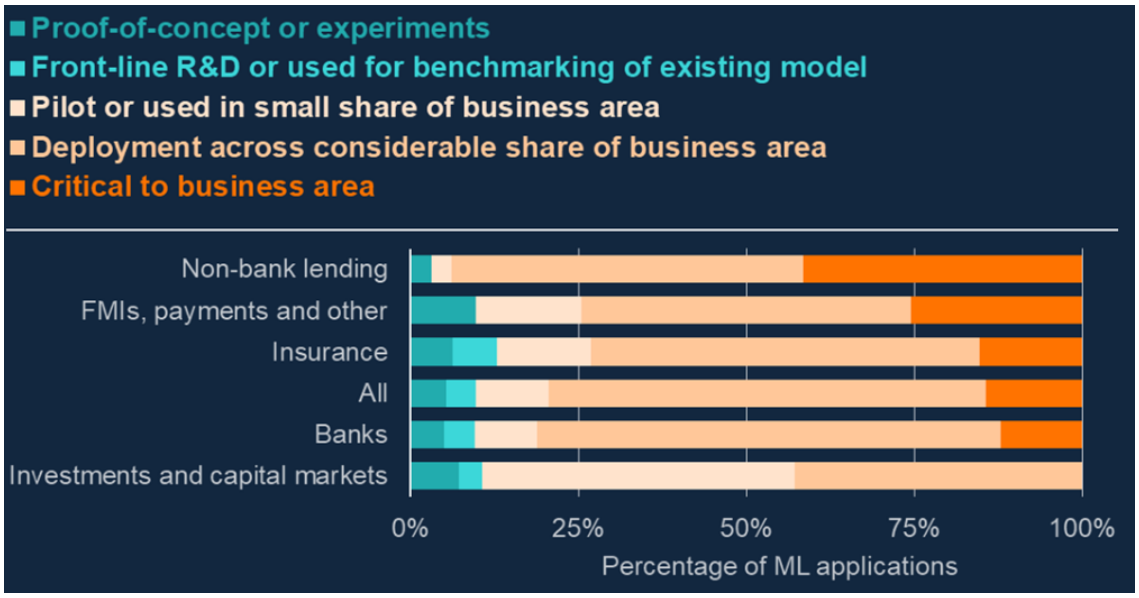

Figura 26 sviluppo e impiego di tecniche di machine learning per tipologia di azienda (Machine learning in UK financial services, 2022).

In termini di varietà di casi d'uso di ML viene evidenziato come le aziende applichino questi processi ad una vasta gamma di ambiti aziendali, il ML viene prevalentemente impiegato nelle interazioni tra azienda e clientela e nella gestione del rischio e compliance, mentre le aree che vedono la minor applicazione sono asset management, investment banking e tesoreria. Il grafico in Figura 28 evidenzia l'importanza che viene attribuita alle soluzioni di ML per ogni area di business, è interessante notare come per l'area di business inerente la tesoreria il queste vengano impiegate per il 40% in mansioni che sono considerate essenziali per la questa area di business nonostante, a livello di numero di applicazioni, sia un segmento in cui questi metodi sono implementati

marginalmente (solo lo 0.4% delle applicazioni di machine learning sono impiegate in questa area), questo dato può indicare come per le mansioni di tesoreria le applicazioni di ML implementate abbiano raggiunto un buon grado di efficienza, limitando la necessità di svilupparne nuove e standardizzando quelle in uso. Con riferimento all'area di tesoreria è altresì interessante notare l'approccio estremamente differente registrato nell'uso di queste tecniche rispetto a quanto avviene in Italia (Figura 21), tenendo comunque presente che i dati riportati fanno riferimento ad anni differenti. In Figura 27 è anche possibile notare come l'implementazione del ML ricopra maggiore importanza nelle aree più tecniche in cui quindi le funzioni di calcolo e classificazione ricoprono maggiore importanza, per l'appunto nelle aree di tesoreria, gestione del credito, gestione dei pagamenti e gestione del rischio, si pensi ad esempio alle funzioni di previsione dei flussi e della liquidità, analisi del rischio e *credit scoring*, valutazione delle garanzie prestate e così via. Interessante è anche notare come per l'area inerente il coinvolgimento della clientela il 97% delle applicazioni di ML sono utilizzate in gran parte delle funzioni del settore ma solo una minima parte di queste sono considerate essenziali, questo si può identificare, ad esempio, nello sviluppo da parte delle aziende operanti nel settore finanziario di applicazioni dirette alla clientela come le applicazioni di *home banking* o comunque allo sviluppo di servizi digitali che permettono alla banca di migliorare le interazioni con il cliente mantenendo comunque un ruolo aggiuntivo e di supporto all'operatore umano. In ultima è importante notare come per le aree di business di asset management e investment banking il numero di applicazioni è generalmente limitato e comunque non considerato vitale per queste aree anche se nel caso dell'asset management oltre il 60% delle applicazioni ricopre un numero considerevole di funzioni attribuite a questa area di business; d'altro canto il fatto che per queste aree nessuna delle applicazioni del ML sia considerata di importanza critica può essere causato da un grado di sviluppo ancora acerbo per queste aree, ipotesi che può essere supportata anche dalla percentuale particolarmente elevata di applicazioni di machine learning considerate in sviluppo, ma non ancora impiegate, per l'investment banking (58%).

Figura 27 applicazione del machine learning per business area aziendale (Machine learning in UK financial services, 2022).

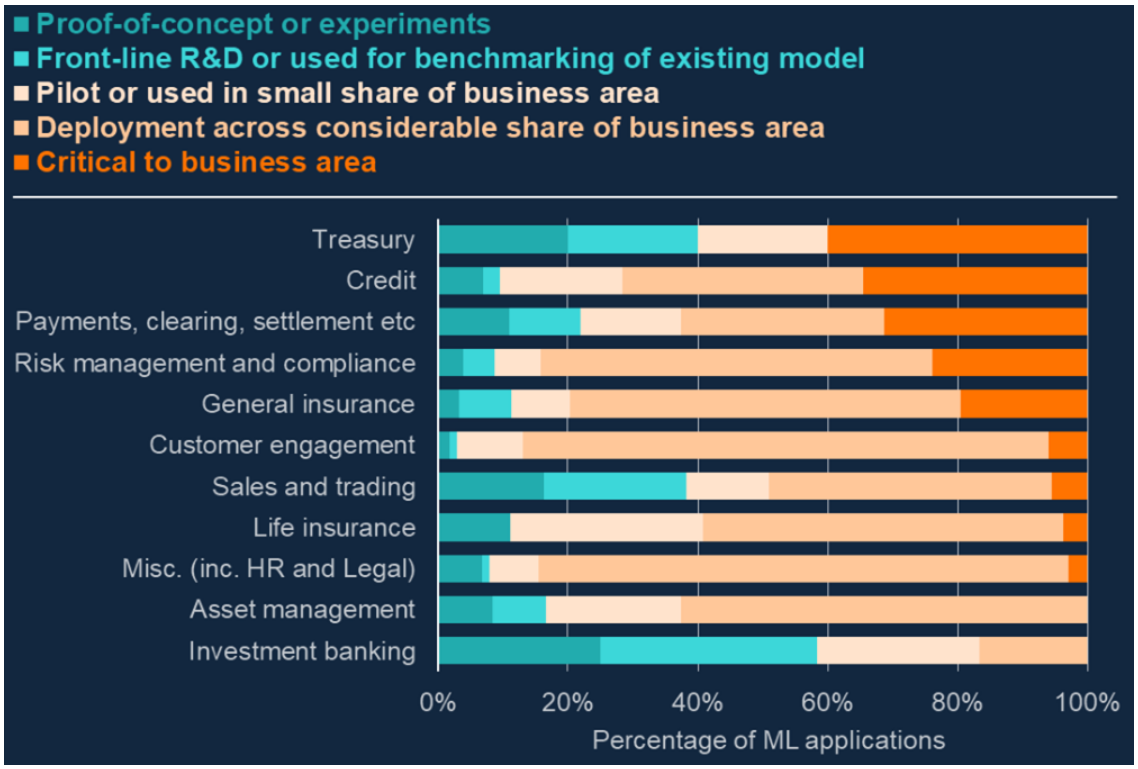

Figura 28 importanza attribuita alle applicazioni di machine learning per business area aziendale (Machine learning in UK financial services, 2022).

Per quanto riguarda strettamente la gestione di capitali, secondo il report OCSE "*Artificial intelligence, machine learning and big data in finance*" l'utilizzo del machine learning nell'ambito della gestione di portafogli è prevalentemente finalizzato alle fasi di ideazione e costruzione di portafogli mentre l'utilizzo per l'effettiva gestione del portafoglio è più limitato con il 27% delle applicazioni di ML finalizzate all'esecuzione degli ordini di acquisto e vendita (Figura 29). Per quanto riguarda invece l'importanza che l'AI riveste nel processo decisionale nel 25% dei casi ha un ruolo predominante andando ad influenzare significativamente le decisioni prese, mentre nel 19% dei casi le decisioni della società sono scarsamente influenzate da questa (Figura 30).

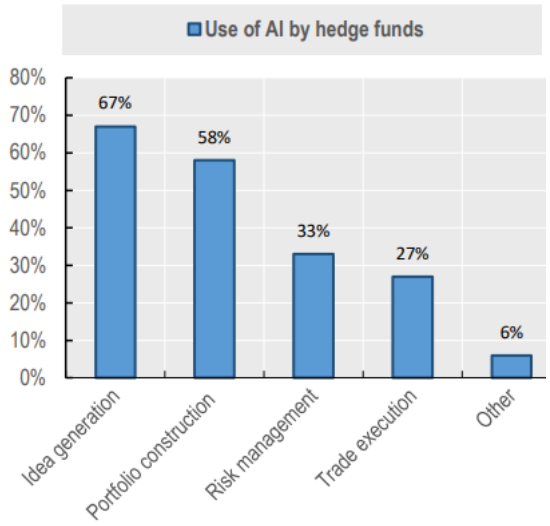

Figura 29 utilizzo dell'intelligenza artificiale nei fondi d'investimento (OCSE, 2021).

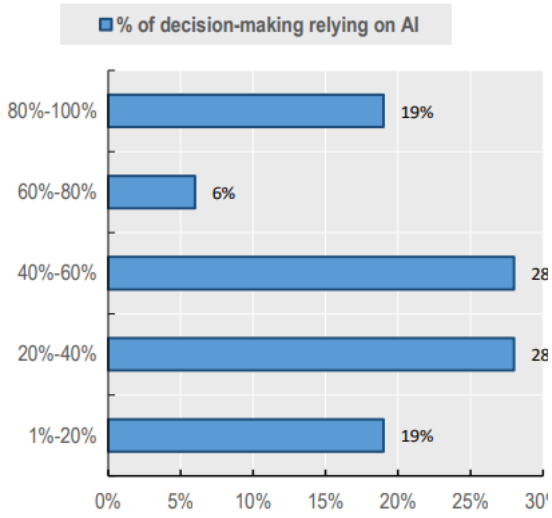

Figura 30 importanza dell'intelligenza artificiale nel processo decisionale (OCSE, 2021).

Per quanto riguarda le performance registrate dai fondi speculativi che utilizzano tecniche di AI e ML queste sono di difficile valutazione, la principale ragione è che ad oggi non sono presenti ricerche accademiche che studiano approfonditamente il fenomeno dei fondi gestiti con intelligenza artificiale, tutte le informazioni in merito possono essere ricavate da fonti redatte dalle stesse società che operano e che, per logiche ragioni, sono riluttanti nel fornire informazioni specifiche sulla gestione dei loro fondi. Inoltre i fondi ETF (*Excahnge-Traded Fund*) basati su AI, nei quali le decisioni di investimento sono eseguite da modelli informatici, non hanno ancora raggiunto dimensioni significative avendo un patrimonio gestito totale stimato di 100 milioni di dollari (OCSE, 2021).

Come rilevato nel report OCSE "*Generative artificial intelligence in finance*" pubblicato nel 2023 nonostante il machine learning e le tecniche di intelligenza artificiale abbiano

iniziato ad essere implementati anche nel settore finanziario il loro utilizzo resta comunque limitato a funzioni di supporto per il lavoratore umano.

Ad oggi tecniche di AI vengono largamente utilizzate per funzioni di back-office per l'implementazione dei processi andando a migliorare sia l'efficienza che l'accuratezza dei processi in essere, mansioni manuali e ripetitive possono essere oggi svolte da questi algoritmi andando a evitare errori umani. Altri casi d'uso dell'AI riguardano il supporto alla fornitura di servizi finanziari o ancora l'uso di modelli di intelligenza artificiale generativa (GenAI)[45] a scopi informativi (come se fossero dei motori di ricerca), come strumento per il supporto al cliente, come modo per semplificare e "umanizzare" la segnalazione dei dati interni ed esterni sulla base dei dati dell'azienda utilizzatrice. E ancora tecniche di ML e AI vengono utilizzate come supporto alle funzioni di gestione del rischio per aziende operanti nell'asset management o per investitori istituzionali data la capacità di questi modelli di simulare le performance di portafoglio ipotizzando svariati scenari di mercato e macroeconomici. Algoritmi sono stati sviluppati anche per la gestione del trading, nei mercati altamente digitalizzati come il mercato azionario o monetario si sono dimostrati utili nella gestione della liquidità, ad esempio andando ad ottimizzare ordini di larghe dimensioni di acquisto o vendita suddividendoli in ordini minori per poter compiere sfruttare condizioni di mercato migliori. L'AI può avere un impatto notevole nel migliorare l'esperienza del cliente essendo già oggi utilizzata per la creazione di interfacce personalizzate con l'utilizzo di *chatbot* sviluppati tramite modelli GenAI, proprio questo genere di modelli possono essere utilizzati anche nello sviluppo di nuovi prodotti e nel miglioramento delle caratteristiche di prodotti già esistenti o nella creazione di campagna di marketing mirate. La combinazione delle potenzialità espresse da queste tecnologie nell'assistenza al cliente e nell'analisi finanziaria è l'area in cui l'intelligenza artificiale ha il maggior impatto immediato, un esempio può essere la creazione di programmi di *robo-advisory* in grado di fornire al cliente una consulenza personalizzata e mirata, consigliando all'utente una soluzione in linea con le sue aspettative ed esigenze (OCSE, 2023).

Nonostante le potenzialità dimostrate l'implementazione e applicazione di processi di AI e GenAI, per l'automatizzazione completa senza che sia necessario l'intervento umano sono ancora in fase di sviluppo. La lenta implementazione delle tecniche di intelligenza

[45] Il ramo di intelligenza artificiale che si occupa della creazione di modelli e sistemi in grado di generare autonomamente nuovi contenuti, come ad esempio Chat GPT.

artificiale, in particolare delle applicazioni di GenAI, nel settore finanziario è in parte giustificata dal fatto che l'attività di mercato è un settore altamente regolamentato e l'applicazione di queste tecniche potrebbe entrare in confitto con tale regolamentazione, in particolare per quanto riguarda gli obblighi di trasparenza, il trattamento dei dati la sicurezza di questi (OCSE, 2023). Un ulteriore fattore che ha limitato l'uso dell'intelligenza artificiale nella finanza è il problema di "spiegabilità" delle decisioni di un agente informatico, come brevemente accennato nel Capitolo 2 della presente tesi una rete neurale, elemento fondante di tutti gli algoritmi di intelligenza artificiale oggi utilizzati, può essere definita come una scatola nera, può risultare quindi difficile per l'operatore umano comprendere pienamente e, per l'appunto, spiegare una decisione presa da un modello di intelligenza artificiale. Questo fenomeno può generare problemi di gestione dei modelli stessi, andando a riporre estrema fiducia nell'output generato dal modello senza metterlo in dubbio o, al contrario, limitandone l'uso in quanto gli output generati possono risultare di difficile comprensione.

4. APPLICAZIONE DEL DEEP LEARNING ALLA GESTIONE DI PORTAFOGLIO

Nel precedente capitolo di questa tesi si sono esposte alcune ricerche che hanno focalizzato i loro sforzi nella creazione di un framework di machine learning per la soluzione di problemi finanziari. In questo capitolo viene presentata la parte sperimentale sviluppata nella tesi con lo sviluppo di un algoritmo di deep learning per la costruzione di un algoritmo di deep learning il cui obiettivo è la gestione attiva di un portafoglio azionario. L'algoritmo viene sviluppato in ambiente Python e nella sua struttura è in larga parte ispirato al paper "*Application of Deep Q-Network in Portfolio Management*" (Gao et al., 2020); per lo sviluppo di questo algoritmo sono stati contattati i professori Zhengyong Jiang e Jonglong Su, docenti presso l'istituto *Xi'an Jiaotong-Liverpool University* per poter avere un confronto in particolare per quanto riguarda la struttura del codice utilizzato ed alcuni aspetti critici inerenti la programmazione dello stesso. La scelta del loro lavoro come riferimento per lo sviluppo del modello dipende da due motivazioni prevalenti: una struttura del codice tale da non dover necessitare eccessiva potenza computazionale per la fase di allenamento e sfruttamento del modello; l'utilizzo di un particolare algoritmo di DL non presente in altri paper analizzati nello studio della materia. Al codice originariamente utilizzato dai professori Jiang e Su sono state apportate comunque alcune modifiche, alcune di carattere meramente informatico con modifiche di alcune stringhe di codice in modo da renderlo utilizzabile sulle ultime piattaforme di coding,[46] altre di natura operazionale avendo cercato di modificare parte del funzionamento del codice per una migliore operatività dello stesso. Il modello viene testato con diverse configurazioni di input e con la presenza o meno dei costi di transazione senza però modificare la struttura informatica del codice, la ragione di questo è valutare l'effettiva adattabilità della struttura informatica proposta a diversi scenari operativi.

In generale, il modello sviluppato vuole rispondere a due questioni emerse nei precedenti capitoli della presente tesi: valutare se è possibile sviluppare un modello di gestione attiva in grado di superare sistematicamente le performance di portafogli a gestione passiva e valutare l'effettiva possibilità di affidare ad un modello pienamente automatizzato la

[46] Nello specifico, il codice sviluppato da Jiang e Su è basato sul linguaggio di programmazione TensorFlow 1.14, questo linguaggio di programmazione non è più completamente operativo in quanto alcuni comandi e funzioni sono stati sostituiti e non sono più compatibili con le più recenti versioni (nello sviluppo del codice viene utilizzato TensorFlow 2.15).

gestione di capitali delegandogli ogni fase della gestione di portafoglio, dalla selezione dei titoli alla previsione dei rendimenti futuri alla gestione delle operazioni di cessione e acquisto dei titoli.

Nel presente capitolo vengono quindi esposte le principali caratteristiche del codice, andando ad analizzare gli elementi informatici e finanziari che lo caratterizzano, i risultati ottenuti dall'algoritmo verranno poi confrontati i risultati ottenuti con altri modelli di gestione del portafogli e con il rendimento dell'indice di mercato S&P 500. Il codice Python sviluppato viene riportato in coda alla presente tesi nella sezione Appendice.

4.1 Elementi finanziari dell'esperimento

4.1.1 Assunzioni dell'esperimento

Nello sviluppo del codice sono state adottate alcune semplificazioni della realtà per ridurre la complessità del problema che si andava ad affrontare:

- Assunzione 1: le azioni dell'agente non influenzano in alcun modo i prezzi nel mercato finanziario;
- Assunzione 2: I titoli sono sufficientemente liquidi da garantire l'immediata realizzazione vendita o acquisto;
- Assunzione 3: assenza di Bid/Ask spread;
- Assunzione 4: il portafoglio rimane invariato tra la fine di un periodo e l'inizio del successivo;
- Assunzione 5: non esistono altri asset oltre ai cinque utilizzati nell'esperimento;
- Assunzione 6: assenza di costi di transazione.

4.1.2 Preselezione dei titoli

Per garantire che le assunzioni 1, 2 e 3 non generino un'eccessiva distorsione rispetto ad un mercato azionario reale i titoli selezionati per lo sviluppo del modello presentano elevati volumi nelle transazioni. I titoli in cui è possibile investire sono stati selezionati in modo da rappresentare la più ampia porzione possibile del mercato finanziario, andando a selezionare titoli leader in cinque diversi settori economici.

I titoli utilizzati nell'esperimento sono: Bank of America (BAC), Ford Motor Company (F), The Coca-Cola Company (KO), Microsoft Corporation (MSFT) e Pfizer Inc. (PFE); rappresentanti di cinque diversi settori economici, rispettivamente il settore bancario

finanziario, il settore automotive, il settore dei beni di consumo, il settore tecnologico ed infine il settore farmaceutico. Tutte e cinque le società sono quotate presso la borsa NYSE (*New York Stock Exchange*) e rientrano nell'indice S&P 500 rappresentante le 500 società con maggiore capitalizzazione quotate presso il mercato statunitense. La selezione di titoli quotati presso lo stesso mercato ha la duplice finalità di semplificare, parzialmente, la struttura di rischio di portafoglio avendo tutti i titoli un rischio di mercato sottostante uguale, e permettere l'utilizzo di un indice già strutturato, lo S&P 500, come benchmark, seppur indicativo, nella valutazione della gestione di portafoglio.

Non è invece stata posta particolare attenzione alla correlazione delle serie storiche dei rendimenti presentate dai titoli selezionati in quanto si voleva valutare la capacità del modello di apprendere autonomamente i principi base della differenziazione.

4.1.3 Serie storiche di prezzi e rendimenti

I dati utilizzati dall'algoritmo sono stati estratti dal sito *Yahoo Finance*, le serie storiche complete sono state suddivise in quattro dataset composti da 365 osservazioni giornaliere, due utilizzati per l'allenamento del modello, due utilizzati per testare il modello allenato. Come esposto in seguito il modello viene quindi implementato su una serie di 365 osservazioni in modo che, attraverso il processo di backpropagation, possa settare autonomamente i parametri del modello in modo da identificare una strategia di gestione ottima, in seguito viene testato su dati non noti. Il primo dataset di allenamento è composto dalle osservazioni giornaliere dal 02/01/2018 al 17/06/2019, viene poi testato sul periodo dal 18/06/2019 al 24/11/2020; il secondo dataset è composto dalle osservazioni giornaliere dal 03/11/2020 al 19/04/2022 per la fase di allenamento e dalle osservazioni dal 20/04/2022 al 02/10/2022 per la fase di test.

La ragione della scelta di finestre temporali di 365 anziché le classiche finestre annue di borsa (quindi di 252 giorni) dipende prevalentemente dalla gestione dei dati mancanti e dalla struttura del modello stesso. Il modello originalmente sviluppato prevedeva l'utilizzo di finestre di durata annuale comprendenti i dati mancanti in cui per i giorni in cui il mercato è chiuso venivano riportati gli ultimi dati disponibili. In questa tesi si è preferito invece utilizzare solo dati di borsa rappresentanti quotazioni giornaliere, andando a evitare di riportare gli stessi dati più volte, inoltre, la struttura implementata per la gestione della memoria interna del codice e utilizzata nella fase di allenamento era vincolata al numero di osservazioni date in input, per evitare di andare a modificare il codice utilizzato in modo particolarmente invasivo, che avrebbe portato a potenziali

problemi di programmazione, si è preferito mantenere lo stesso numero di osservazioni giornaliere utilizzato nel lavoro originale.

Nelle tabelle riportate in seguito sono riportate le statistiche descrittive dei titoli selezionati per l'applicazione per il primo periodo di analisi, dal 02/01/2018 al 24/11/2020, alla Tabella 2; per il secondo periodo di analisi dal 03/11/2020 al 02/10/2022 nella Tabella 3. In entrambe le tabelle le statistiche sono state divise per i dati utilizzati nella fase di allenamento del modello (etichetta train nella tabella) e nella fase di test.

		BAC	F	KO	MSFT	PFE
train	rend. medio[47]	-0.0176%	-0.0650%	0.0328%	0.1188%	0.0440%
	varianza	0.02425%	0.03285%	0.01081%	0.02782%	0.01500%
	asimmetria	-0.02674	0.318795	-1.91128	-0.0394	-0.49528
	curtosi	2.672566	5.514404	14.50809	2.270133	2.179062
test	rend. medio	0.0090%	-0.0149%	0.0100%	0.1309%	-0.0281%
	varianza	0.09928%	0.09200%	0.03606%	0.05706%	0.03843%
	asimmetria	-0.06411	0.523591	-0.66007	-0.4773	-0.2177
	curtosi	7.671594	8.24603	5.501587	9.363774	4.530849

Tabella 2 statistiche descrittive dei titoli per il primo periodo di analisi (dal 02/01/2018 al 24/11/2020).

		BAC	F	KO	MSFT	PFE
train	rend. medio	0.0815%	0.1961%	0.0756%	0.0884%	0.1036%
	varianza	0.03071%	0.07564%	0.01115%	0.02320%	0.03107%
	asimmetria	0.046145	0.266354	-0.01921	-0.07248	0.90282
	curtosi	0.820444	1.750421	4.361787	0.590966	3.908875
test	rend. medio	-0.0929%	-0.0744%	-0.0437%	0.0330%	-0.1071%
	varianza	0.03291%	0.06721%	0.01116%	0.03967%	0.02135%
	asimmetria	0.31291	-0.55453	-0.79391	0.095881	0.124035
	curtosi	1.496915	2.220684	6.107886	1.559714	0.901971

Tabella 3 statistiche descrittive dei titoli per il secondo periodo di analisi (dal 03/11/2020 al 02/10/2022).

Oltre ai cinque titoli azionari presentati l'universo di titoli in cui è possibile investire è composto da un ulteriore titolo rappresentante il denaro contante, quest'ultimo è settato in modo da avere sempre valore unitario e non offrire alcun rendimento. Alle pagine

[47] Calcolato come la media dei rendimenti logaritmici ovvero $\frac{1}{n}\sum_{t=1}^{n}\ln\frac{p_t}{p_{t-1}}$ dove p_t e p_{t-1} indicano il prezzo del titolo al tempo *t* e al tempo $t-1$.

seguenti vengono riportati i grafici rappresentati le serie storiche dei prezzi e dei rendimenti per i cinque titoli selezionati per il primo e per il secondo periodo di analisi (da Figura 31 a Figura 35), in ogni grafico la linea verticale rossa indica la separazione tra i dati utilizzati per l'allenamento (a sinistra) e i dati usati per il test (a destra).

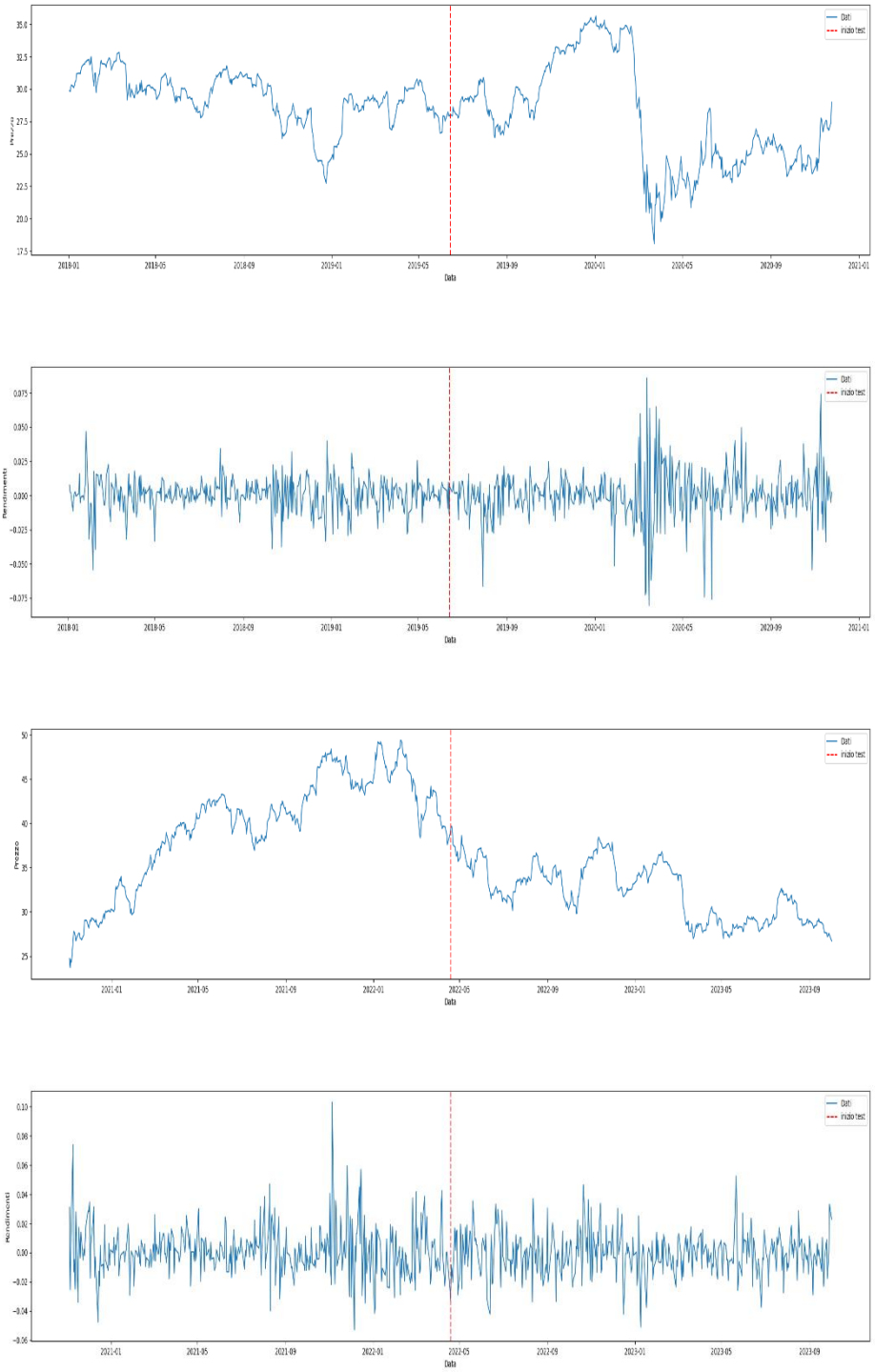

Figura 31 grafico dei prezzi e dei rendimenti per il titolo BAC nei due periodi di analisi.

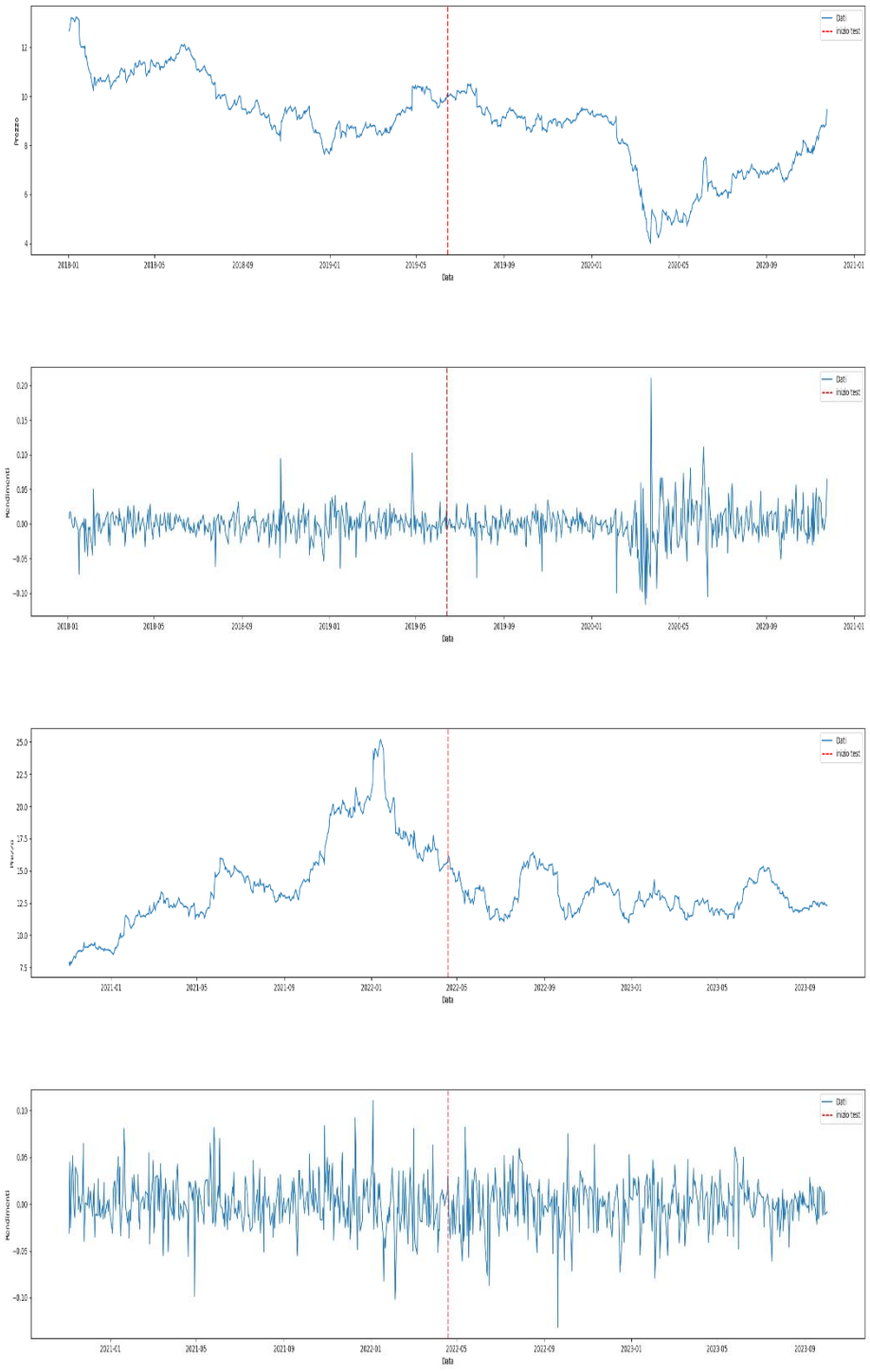

Figura 32 grafico dei prezzi e dei rendimenti per il titolo F nei due periodi di analisi.

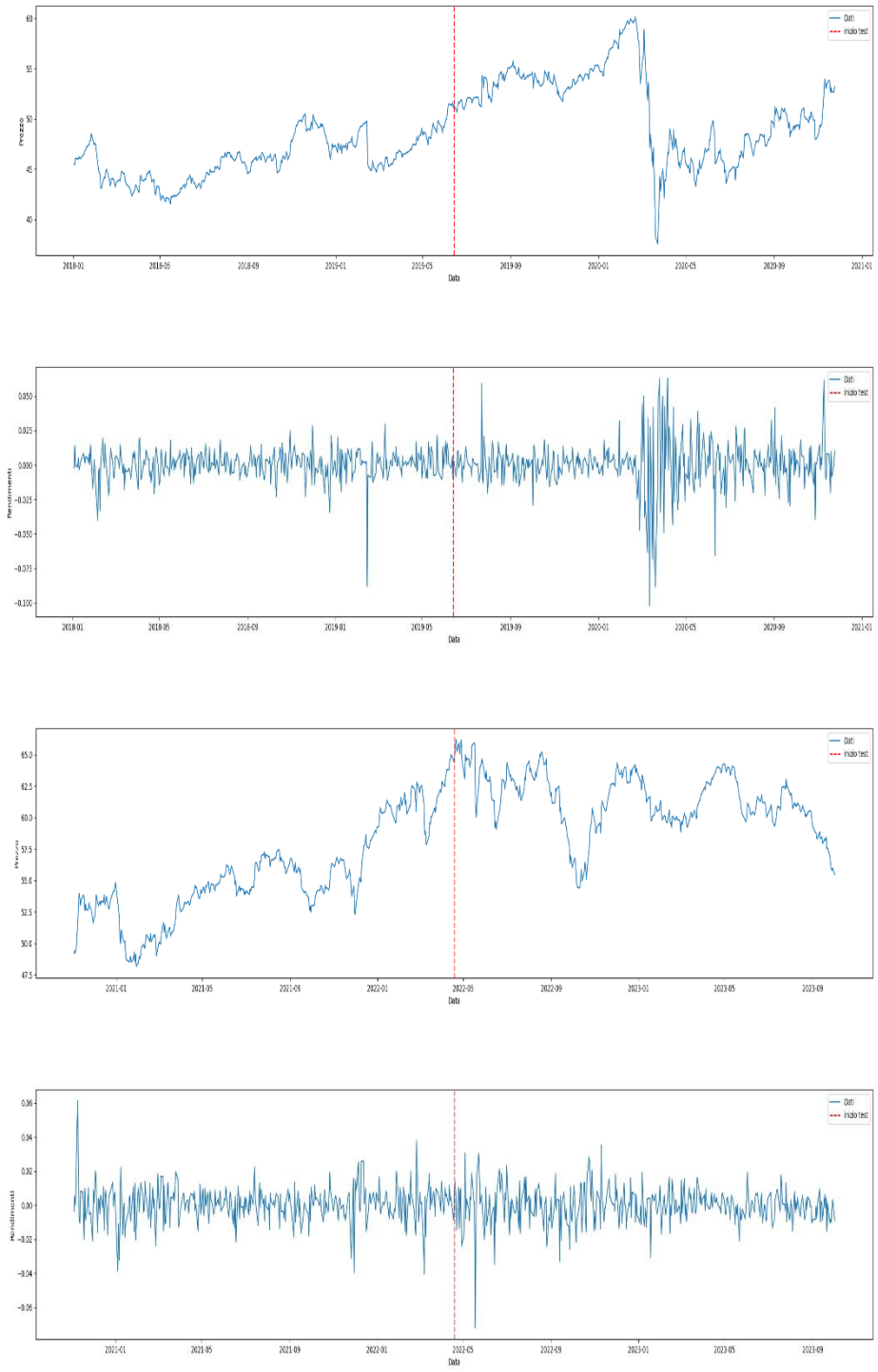

Figura 33 grafico dei prezzi e dei rendimenti per il titolo KO nei due periodi di analisi.

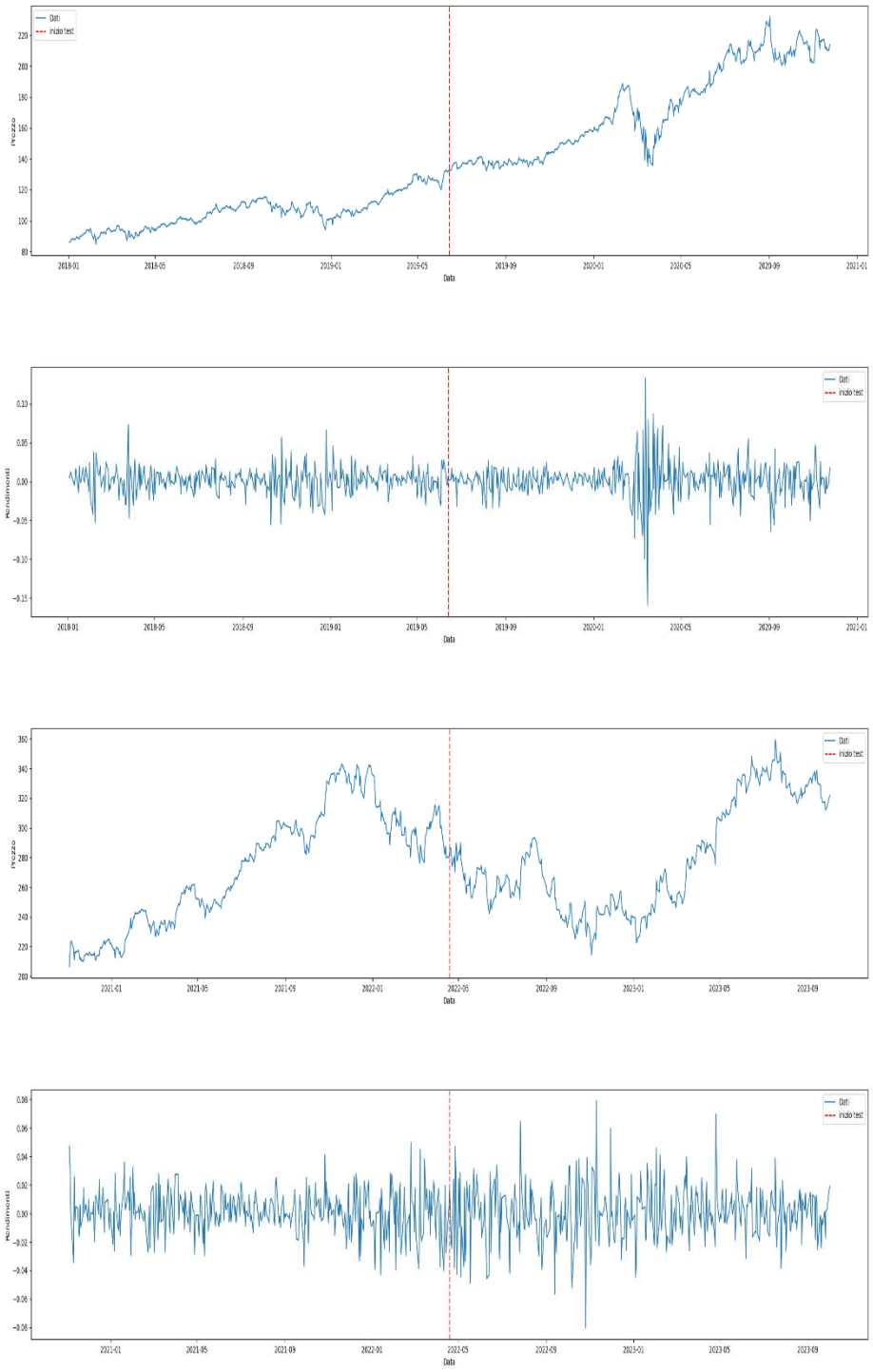

Figura 34 grafici dei prezzi e dei rendimenti per il titolo MSFT nei due periodi di analisi.

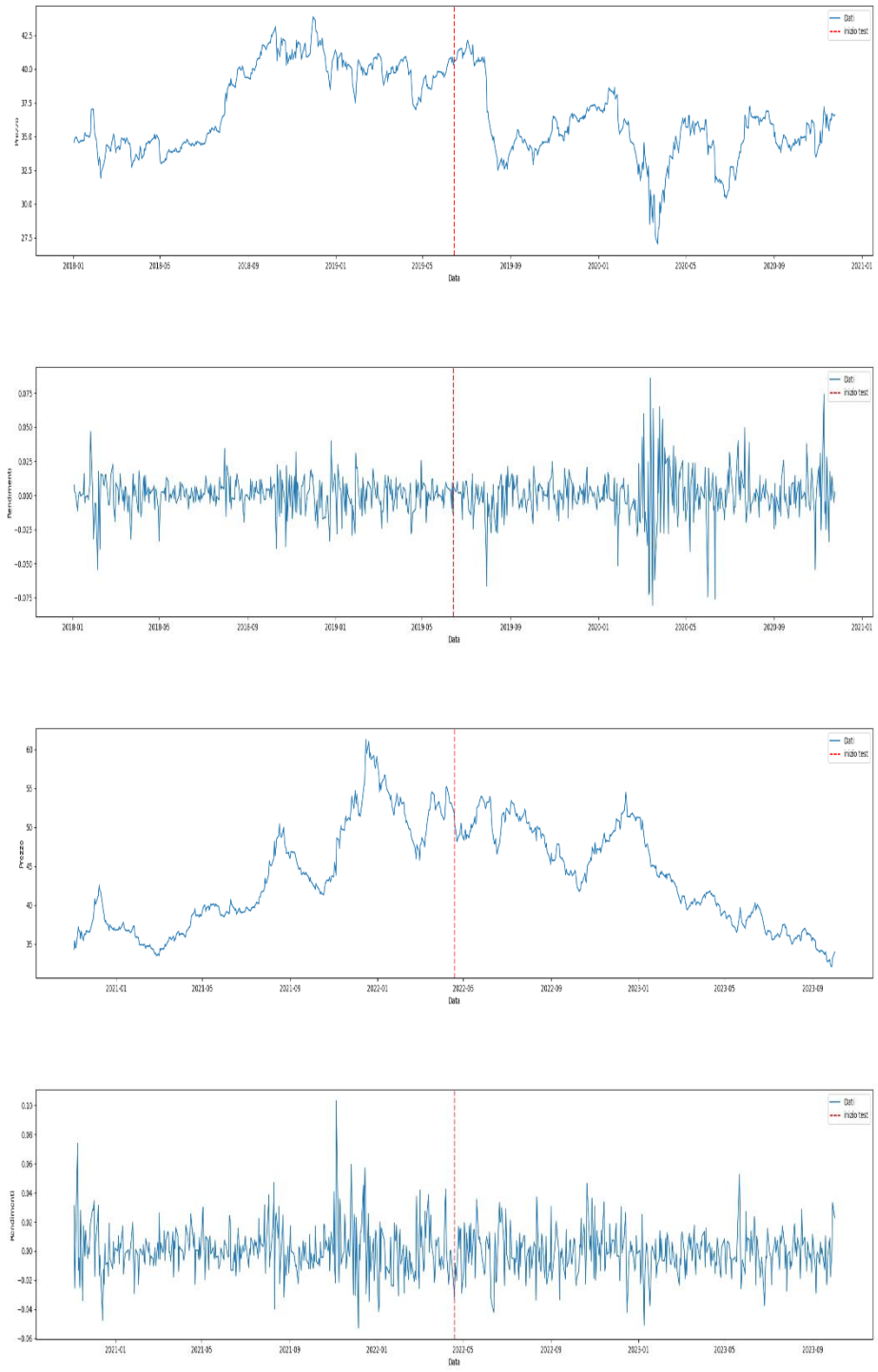

Figura 35 grafici dei prezzi e dei rendimenti per il titolo PFE nei due periodi di analisi.

4.2 Elementi dell'algoritmo

4.2.1 Trattamento dei dati e input della rete

L'input utilizzato dalla rete è composto dalle osservazioni giornaliere dei cinque titoli per i valori di apertura, chiusura, prezzo massimo e prezzo minimo. L'utilizzo dei prezzi come input presenta due problemi: un problema di scala dei dati in quanto questi si presentano

i scale di valori non omogenee e un problema di apprendimento in quanto le reti neurali si sono dimostrate più efficienti con input a media nulla e varianza unitaria. Dati questi fattori ed il fatto che nel computo delle ricompense è basato sulle variazioni percentuali dei prezzi di chiusura e apertura dei titoli i dati sono stati normalizzati secondo la formulazione riportata alla seguente equazione

$$P_t = [p^c_{t-n+1} \oslash p^c_t \mid p^c_{t-n+2} \oslash p^c_t \mid \dots \mid p^c_{t-1} \oslash p^c_t \mid 1]$$

$$P^o_t = [p^o_{t-n+1} \oslash p^c_t \mid p^o_{t-n+2} \oslash p^c_t \mid \dots \mid p^o_{t-1} \oslash p^c_t \mid p^o_t \oslash p^c_t]$$

$$P^{hi}_t = [p^{hi}_{t-n+1} \oslash p^c_t \mid p^{hi}_{t-n+2} \oslash p^c_t \mid \dots \mid p^{hi}_{t-1} \oslash p^c_t \mid p^{hi}_t \oslash p^c_t]$$

$$P^{lo}_t = [p^{lo}_{t-n+1} \oslash p^c_t \mid p^{lo}_{t-n+2} \oslash p^c_t \mid \dots \mid p^{lo}_{t-1} \oslash p^c_t \mid p^{lo}_t \oslash p^c_t]$$

(4. 1)

Dove $\oslash$ indica una divisione vettoriale, o divisione di Hadamard, P^c_i identifica la matrice dei prezzi di chiusura al tempo *i*, P^c_t gli ultimi valori dei prezzi di chiusura registrati nella finestra temporale, P^o_i, P^{hi}_i e P^{lo}_i indicano rispettivamente le matrici dei prezzi giornalieri di apertura, massimo e minimo al tempo *i*. Le quattro matrici così ottenute sono di dimensioni (*M,N*) dove *M* è il numero di asset e *N* la lunghezza della finestra temporale considerata; le quattro matrici vengono poi a loro volta concatenate in un tensore X_t contente quindi tutti gli input normalizzati inseriti nell'algoritmo, formalmente:

$$X_t = [P^c_t, P^o_t, P^{hi}_t, P^{lo}_t]$$

(4. 2)

In definitiva, l'input inserito nella rete neurale è un tensore tridimensionale di dimensioni (*M,N,4*) rappresentato dalla seguente Figura 36.

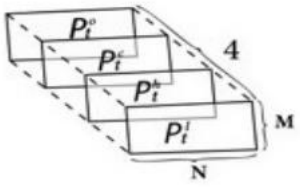

Figura 36 tensore dei prezzi (Gao et al., 2020).

Nel paper di riferimento (Gao et al., 2020) viene applicato un ulteriore passaggio nel trattamento dei dati, infatti ad ogni valore della matrice viene sottratto il valore di 1 e

viene in seguito moltiplicato per un coefficiente di espansione α, il tensore di input viene quindi trasformato nel tensore X_t^* come da seguente equazione (4.3)

$$X_t^* \triangleq \alpha(X_t - 1)$$

(4. 3)

Quest'ulteriore trasformazione dei dati, mantenendo invariato il valore di α e applicata al dataset utilizzato nell'esperimento, sembra generare instabilità nel processo di allenamento del modello che registrava una varianza, calcolata sull'evoluzione del valore del portafoglio, nettamente maggiore a quella registrata nel modello presentato in questa tesi; si è quindi preferito mantenere la formulazione utilizzata in Jiang, Xu e Liang (2017) riportata all'equazione (4.2).

Il tensore X_t non contiene le informazioni dell'intero periodo selezionato ma soltanto le informazioni inerenti le ultime *N* osservazioni come evidenziato dall'equazione (4.1), ciò comporta che ad ogni istante temporale il tensore venga aggiornato con una *rolling window*, quindi al tempo t conterrà le osservazioni dal tempo *t-n+1* al tempo *t*, al tempo *t+1* conterrà le informazioni dal tempo *t-n+2* al tempo *t+1* e così via. In questo processo di aggiornamento il valore del titolo al tempo corrente sarà sempre pari a 1.

L'input fornito alla rete, e quindi lo stato, è composto, oltre che dal tensore dei prezzi X_t anche dai pesi di portafogli del periodo precedente w_{t-1}. Lo stato *s* ad un determinato step temporale *t* può quindi essere espresso secondo l'equazione

$$s_t = (X_t, w_{t-1})$$

(4. 4)

4.2.2 Topologia e tipologia della rete neurale

L'algoritmo utilizzato è strutturato come una *Dueling Q-Network* applicato ad una *Convolutional Neural Network* (CNN), questa tipologia di rete neurale riprende in larga parte la struttura di un algoritmo di Q-Learning classico ma utilizza una struttura a doppio canale per stimare separatamente il valore dello stato *V(s)* ed il valore del vantaggio generato dalla selezione dell'azione *a* dato lo stato *s A(s,a)*, il calcolo del Q-value di ogni coppia stato azione segue quindi l'equazione

$$Q(s.a) = V(s) + A(s.a)$$

(4. 5)

Una struttura Dueling Q-Network presenta molteplici vantaggi rispetto ad una classica rete Q-Network, innanzitutto si è dimostrata più performante, a parità di condizioni, della rete Q-Network standard utilizzata da Mnih et al. nel paper "*Human-level control through deep reinforcement learning*" presentato al Paragrafo 2.7 della corrente tesi; inoltre si è dimostrata più adatta nell'applicazione a problemi model-free di RL. In ultima la possibilità di valutare separatamente stato e azione permette all'agente di poter identificare se un dato stato è vantaggioso o meno senza dover necessariamente imparare l'effetto che avrebbe ogni possibile azione compiuta in quello stato (Wang, 2016).

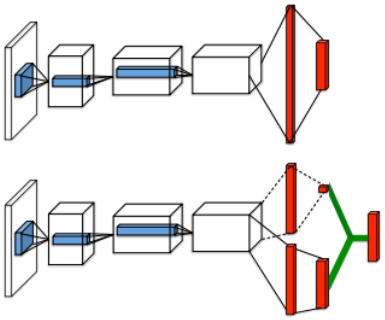

Figura 37 confronto strutture tra una rete Q-Network (sopra) e una rete Dueling Q-Network (sotto) (Wang, 2016).

La topologia della rete è la stessa presentata in Gao et al. (2020) e riportata in Figura 40, l'unica differenza sostanziale riguarda la differente scelta delle funzioni di attivazione, non utilizzando infatti il metodo di trattamento dei dati riportato all'equazione (4.3) è possibile utilizzare una funzione di attivazione ReLU anziché la funzione di attivazione SELU utilizzata in Gao et al. (2020). Infatti utilizzando il processo indicato all'equazione (4.3) la rete riceve in input dei valori negativi, che come visto al Paragrafo 2.5.1 della corrente tesi provocherebbero il fenomeno dying ReLU, essendo tutti i valori normalizzati come rapporti tra valori positivi questi non possono avere valori negativi, si è quindi preferito utilizzare la funzione di attivazione ReLU che ha dimostrato un notevole grado di efficienza nelle reti CNN.

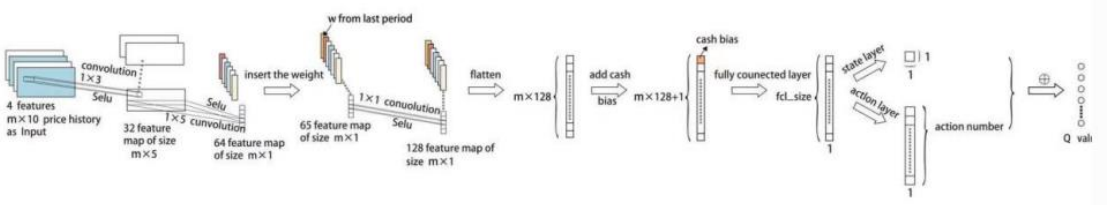

Figura 38 topologia della rete (Gao et al., 2020).

Gli input che definiscono lo stato al tempo *t*, ovvero tensore dei prezzi e allocazione di portafoglio al tempo $t-1$, vengono forniti alla rete in due separati momenti, il tensore dei prezzi X_t viene immediatamente fornito al primo layer, tramite convoluzione 1×3, mentre i pesi dei portafogli precedenti vengono forniti soltanto al terzo layer; i due input vengono forniti separatamente in quanto i primi layer della rete hanno la funzione di definire una sorta di preferenza per i titolo, vengono quindi utilizzati per trattare gli input contenuti nel tensore X_t al fine di ottenere un'indicazione circa le potenzialità di crescita dei singoli titoli, mentre il vettore contente i pesi del portafoglio precedente w_{t-1} sono necessari per definire le azioni da intraprendere considerando l'attuale composizione di portafoglio, il processo logico si può definire quindi in due valutazioni distinte poi integrate, prima vengono definite le potenzialità di crescita dei singoli titoli senza considerare il potenziale guadagno dell'attuale portafoglio, poi viene identificato il potenziale di crescita del portafoglio attualmente detenuto, sulla base delle valutazioni effettuate nei primi due layer; queste due valutazioni confluiscono poi in un layer unidimensionale a cui viene aggiunto un valore di valutazione del contante (*cash bias*), questo layer *fully connected*[48] unidimensionale, di dimensioni $1 \times 128 + 1$, viene infine utilizzato per la generazione dell'output della rete. Utile notare come il primo layer della rete implementata non ha forma unidimensionale ma viene strutturato come una serie di mappe di caratteristiche bidimensionali; questa è la struttura tipica di una rete CNN, sviluppata principalmente per il riconoscimento d'immagini e pattern spaziali. Nel caso specifico l'utilizzo di una struttura a mappa di caratteristiche, anziché di un layer unidimensionale, rende più efficiente il riconoscimento di pattern multipli, quindi agevola il riconoscimento di pattern di dati quanto si manifestano in diverse parti della serie non necessariamente sequenziali. Inoltre rende possibile la scalabilità a dati multi-canale, ovvero rende possibile l'analisi di una serie storica catturando informazioni da più fonti di dati contemporaneamente, nel caso in analisi ogni valutazione sugli asset è influenzata da quattro canali di dati.

L'approccio utilizzato per la valutazione dei titoli ricalca la modalità proposta nella ricerca di Jiang et al. (2017) i dati dei singoli asset vengono trattati separatamente secondo

[48] L'utlizzo di uno strato fully connected (o densamente connesso) permette l'apprendimento di relazioni complesse tra le caratteristiche estratte dai livelli precedenti di rete, inoltre è particolarmente utile nella definizione di pattern globali dei dati, questa tipologia di layer è necessariamente unidimensionale, per questo si procede al processo di appiattimento delle mappe di caratteristiche costruite nei primi due layer della rete.

il metodo degli EIIE, le azioni che popolano il mercato vengono quindi analizzate separatamente e soltanto nella fase di definizione dei Q-value queste vengono utilizzate contemporaneamente per la definizione del portafoglio; la logica implementale è molto simile in entrambi i paper, la rete prima analizza i singoli asset per definirne la potenzialità di crescita e solo seguentemente vengono valutate le performance del portafoglio. Questo approccio si sposa con l'obiettivo dell'algoritmo ovvero la massimizzazione del rendimento, se, ad esempio, nel bacino di asset disponibili ce ne fosse uno che garantisce sempre un rendimento più elevato rispetto agli altri l'algoritmo dovrebbe scegliere di investire tutto il capitale in quel solo asset, ignorando la composizione di portafoglio.

La struttura Dueling Q-Network viene utilizzata soltanto nella parte finale della rete per ottenere l'output, l'ultimo layer della rete è infatti diviso nella valutazione dello stato s_t e nella valutazione dei vantaggi generati da tutte le azioni possibili, la combinazione di questi fornisce il Q-value della miglio coppia stato azione.

È possibile dire che la rete complessiva sia divisa in due parti, la prima parte composta dai primi tre layer utilizza soltanto i dati inerenti lo stato, mentre la seconda parte è utilizzata per la definizione dell'azione migliore, lo stato può essere espresso secondo l'equazione

$$s_t = c_3 \; o \; c_2 \; o \; c_1(X_t, w_{t-1})$$

(4. 6)

In cui c_i rappresenta l'i-esimo layer mentre o identifica la composizione di funzioni che connettono i layer (Gao et al., 2020). Data l'equazione (4.6) il Q-value restituito come output dalla rete viene definito come

$$Q_{(s_t,a)} = Q_s + (Q_a - E[Q_a])$$

(4. 7)

Dove Q_s è il Q-value attribuito allo stato mentre Q_a identifica il Q-value attribuito all'azione, la differenza $Q_a - E[Q_a]$ è il vantaggio generato dall'azione a.[49]

[49] Il vantaggio viene espresso in termini di differenza tra il valore dell'azione scelta Q_a e la media dei valori attribuiti a tutte le azioni possibili $E[Q_a]$.

4.2.3 Selezione dell'azione e funzione di ricompensa

Nel framework proposto le azioni sono definite dall'allocazione di capitale per ogni asset, quindi l'azione a_t è definita come

$$a_t = w_t$$

In un problema di gestione di portafoglio le azioni sono definite in spazio continuo, questo perché le potenziali opzioni d'investimento sono, in linea teorica, frazionabili all'infinito, nel framework proposto le azioni sono definite invece in uno spazio discreto per cui ogni azione viene espressa in frazione del capitale investito; per fare questo il portafoglio iniziale, di valore 1, viene diviso in N parti, in questo modo di ottiene la frazione minima di investibile in ogni asset ovvero $1/N$, le possibili azioni sono definibili come

$$\left(\frac{N}{N}, 0,0, \dots ,0\right)$$
$$\left(\frac{N-1}{N}, \frac{1}{N}, 0, \dots ,0\right)$$
$$\vdots$$
$$\left(\frac{N-1}{N}, 0,0, \dots , \frac{1}{N}\right)$$
$$\vdots$$
$$\left(0,0,0, \dots , \frac{N}{N}\right)$$

(4. 8)

Ad esempio, se $N = 5$ significa che per ogni titolo sono investibili frazioni di 1/5 del capitale. La scelta di uno spazio di azioni possibili discreto è dovuto a due ragioni: la prima di natura informatica, uno spazio di azioni infinito avrebbe portato ad un eccessivo dispendio computazionale, inoltre essendo il set di dati relativamente ridotto alcune azioni potrebbero non venire mai intraprese nella fase di allenamento andando a rendere la fase di allenamento non pienamente impiegata, causando potenziali problemi di overfitting; la seconda ragione è di natura pratica, una gestione di portafoglio che va ad allocare le risorse in frazioni infinitesimali richiede, di fatto, una costante riallocazione di queste,

questo a livello pratico è inapplicabile per i costi di gestione di portafoglio e per il numero elevatissimo di transazione che dovrebbero essere completate.

Le azioni intraprese dall'agente sono volete alla massimizzazione del valore di portafoglio al termine del periodo di trading, il valore di portafoglio al termine del periodo *t* viene definito come

$$y_t' = y_t w_t \cdot \mu_t$$

(4. 9)

dove y_t identifica il valore del portafoglio all'inizio del periodo *t*, w_t identifica il vettore dei pesi di portafoglio all'inizio del periodo *t*, mentre μ_t identifica il vettore dei rendimenti dei singoli asset nel corso del periodo *t*, calcolato come il rapporto tra i prezzi di chiusura e i prezzi di apertura per gli *M* asset che compongono l'universo di titoli selezionabili, matematicamente μ_t viene espresso come

$$\mu_t = p_t^c \oslash p_t^o = \left(1, \frac{p_{t,1}^c}{p_{t,1}^o}, \frac{p_{t,2}^c}{p_{t,2}^o}, \dots, \frac{p_{t,m}^c}{p_{t,m}^o}\right)$$

(4. 10)

in cui il primo valore, rappresentando la disponibilità liquida non ha variazione quindi è sempre pari a 1. Ad ogni time step la ricompensa r_t ottenuta dall'agente è pari al rendimento logaritmico di portafoglio quindi

$$r_t = \ln\left(\frac{y_t'}{y_{t-1}'}\right) = \ln(w_t \cdot \mu_t)$$

(4. 11)

L'agente ha quindi il compito di massimizzare il valore di portafoglio allo stato terminale T, definito secondo l'equazione

$$y_T' = \exp\left(\sum_{t=1}^{T} r_t\right) \cdot y_0'$$

(4. 12)

dove y_0' indica il valore di portafoglio al tempo iniziale 0 in cui il portafoglio viene settato con valore pari a 1 e capitale interamente allocato nella disponibilità liquida quindi

$$w_0 = [1,0,0,\dots,0]$$

(4. 13)

4.2.4 Allenamento della rete e campionamento prioritizzato

Il processo di allenamento della rete prevede la minimizzazione del valore della funzione di costo tramite processo di backpropagation. Prendendo in considerazione che compito di un algoritmo di Q-learning, e quindi della rete neurale utilizzata, è la stima della vera funzione di valore Q l'algoritmo è strutturato in modo da avere una rete di valutazione del Q-value, Q_{eval} e una rete obiettivo Q_{target} con uguale struttura ma parametri diversi; i parametri della rete di valutazione vengono costantemente aggiornati mentre i parametri della rete obiettivo sono fissi, finché non vengono sostituiti da parametri ottenuti dalla rete di valutazione.

Durante il processo di allenamento viene estratto un campione di *n* esperienze passate

$$\{(s_{t_1}, a_{t_1}, r_{t_1}, s_{t_1+1}), \dots, (s_{t_n}, a_{t_n}, r_{t_n}, s_{t_{n+1}})\}$$

(4. 14)

la rete di valutazione Q_{eval} riceve in input lo stato corrente s_t e restituisce in output un Q-value $Q_{eval}(s_t, a)$ per ogni azione $a \in A$, allo stesso tempo la rete obiettivo riceve come input lo stato s_{t+1} e restituisce come output il Q value $Q_{target}(s_{t+1}, a)$ per ogni possibile azione $a \in A$, per entrambe le reti vengono selezionate le coppie stato-azione con il maggior valore quindi

$$Q_{target^*(i)} = Q_{target}\left(s_{t_i+1}, argmax\left(Q_{eval}(s_{t_i+1}, a)\right)\right)$$

(4. 15.a)

$$Q_{eval^*(i)} = Q_{eval}\left(s_{t_i}, argmax\left(Q_{eval}(s_{t_i}, a)\right)\right)$$

(4. 15. b)

dove $i = 1,2,\dots,n$

In questo mod si ottengono i vettori di valori Q per la rete di valutazione e la rete obiettivo

$$\boldsymbol{Q_{target^*}} = [Q_{target^*(1)}, Q_{target^*(2)}, \dots, Q_{target^*(n)}]$$

(4. 16. a)

$$\boldsymbol{Q_{eval^*}} = \left[Q_{eval^*(1)}, Q_{eval^*(2)}, \dots, Q_{eval^*(n)}\right]$$

(4. 16. b)

La stima dei Q-value attraverso le reti di valutazione e stima sono propedeutiche alla stima della vera funzione di valore Q_{real} che viene effettivamente utilizzata dalla rete nel processo di allenamento, i valori Q-value della rete vengono stimati utilizzando la formulazione classica di aggiornamento dei Q-value per una rete Q-learning presentata al Paragrafo 2.3.1 della presente tesi.

L'effettiva stima della funzione Q viene ottenuta e il vettore contenente i suoi valori Q_{real} sono quindi formulati come

$$Q_{real(i)} = r_{t_i} + \gamma Q_{target^*(i)}$$

(4. 17. a)

$$\boldsymbol{Q_{real}} = \left[Q_{real(1)}, Q_{real(2)}, \dots, Q_{real(n)}\right]$$

(4. 17. b)

Dove γ nell'equazione (4.17.a) è il fattore di sconto delle ricompense future di valore $\gamma \in (0,1]$.
Infine la funzione di costo per viene definita come il prodotto vettoriale delle differenze tra i Q-value stimati dalla rete di valutazione e i Q-value reali

$$\boldsymbol{l^*} = (\boldsymbol{Q_{eval^*}} - \boldsymbol{Q_{real}}) \odot (\boldsymbol{Q_{eval^*}} - \boldsymbol{Q_{real}})$$

(4. 18)

Data la difficoltà di ottenere informazioni effettivamente esplicative dal *batch di memoria* il campionamento casuale di queste si può dimostrare inefficace, per accelerare il processo di allenamento viene quindi utilizzato un meccanismo di campionamento prioritizzato, questo processo prevede la selezione delle esperienze passate che registrano il maggior errore temporale (*TD-error*) espresso come

$$TD\text{-}error = |Q_{real} - Q_{eval}|$$

(4. 19)

La selezione delle esperienze che presentano il massimo TD-error è basata su una struttura di dati *SumTree* in modo da rendere maggiormente efficiente l'identificazione delle esperienze. La struttura del SumTree, riportata in Figura 41, prevede che alla base di questo vi siano i singoli TD-error delle osservazioni che vengono tra loro sommati fino ad arrivare alla radice composta dalla somma totale dei TD-error; la ricerca dei singoli TD-error con valore più elevato si concretizza ripercorrendo l'albero dalla "radice" alle "foglie" selezionando sempre la foglia che presenta il valore più elevato, in questo modo si aumentano le probabilità di andare a selezionare singole esperienze con TD-error più elevati; questo processo, pur aumentando la probabilità di selezionare le esperienze più significative rispetto ad un processo di campionamento casuale, non garantisce che venga sempre selezionata l'esperienza che presenta TD-error più elevato.

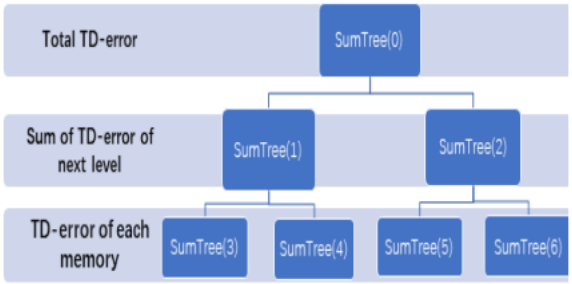

Figura 39 struttura del SumTree (Gao et al., 2020).

Ad ogni esperienza contenuta nel batch di memoria viene associato un peso K calcolato come

$$K_{(i)} = \left(p_{(i)}/p_{(\min)}\right)^{-\beta} \tag{4.20}$$

dove $i = 1,2,\dots,n$

Nell'equazione (4.20) $p_{(i)}/p_{(\min)}$ indica il rapporto tra il TD-error del campione di esperienza i e il valore TD-error minimo registrato in tutta la popolazione di esperienze (identificato sempre tramite il SumTree, in questo caso vengono però selezionati i valori minori per ogni livello) mentre β è una costante tale che $\beta \in (0,1]$[50], il vettore di pesi $\boldsymbol{K}$ è quindi composto da tutti i pesi calcolati per le n esperienze comprese nel batch di memoria, quindi

[50] Nell'algoritmo il valore è stato arbitrariamente settato a 0.05

$$\boldsymbol{K} = \left[K_{(1)}, K_{(2)}, \dots, K_{(n)}\right]$$

(4. 21)

La funzione di costo l da minimizzare è quindi composta dal prodotto tra il vettore dei pesi $\boldsymbol{K}$ e la funzione di costo dei Q-value riportata all'equazione (4.18); la funzione di costo l utilizzata nel processo di backpropagation è quindi

$$l = \boldsymbol{l}^* \cdot \boldsymbol{K}$$

(4.22)

Ovvero l'errore di stima dei Q-value identificato dalla rete di valutazione rispetto al loro vero valore pesato per valore associato all'esperienza che lo ha generato.

4.3 Settaggi utilizzati

Nella sperimentazione dell'algoritmo sono stati testati due differenti settaggi di input dei dati e due differenti settaggi di selezione dell'azione, per quanto riguarda invece i settaggi dei parametri di rete questi sono stati inizialmente testati con diversi settaggi ma le modifiche apportate tra un settaggio e l'altro non hanno portato a sostanziali differenze nel processo di apprendimento della rete né nella fase di test quindi non sono stati riportati.

I settaggi di rete utilizzati sono riportati alla Tabella 4, questi sono stati mantenuti invariati per tutti differenti settaggi utilizzati nelle fasi di allenamento e di test del modello con eccezione dell'impostazione con aumento dello spazio di azione; la ragione per cui i settaggi non sono stati modificati risiede nella volontà di valutare se la stessa struttura dell'algoritmo è in grado di gestire scenari differenti, senza la necessità di procedere ad una fase di *fine-tuning* per ogni differente impostazione. Sono stati comunque provati altri settaggi di rete nella fase di strutturazione del modello ma senza che le modifiche apportate portassero ad effettivi miglioramenti o peggioramenti delle performance, per questa ragione si è preferito mantenere i settaggi proposti da Gao et al. (2020) eccezion fatta per la costante utilizzata nell'equazione (4.20), questa viene arbitrariamente inizializzata con un valore di 0,05.

Trading period[51]	2
Time step di allenamento	40.000
Time step per episodio	160
Tasso di apprendimento α	0,00025
Tasso di esplorazione iniziale ε	1
Periodo di decadenza di ε (in time step)	15.000
Valore minimo di ε	0,001
Dimensione del batch di allenamento (in time step)	100
Tasso di sconto delle ricompense future γ	0,17
Valore iniziale dei pesi di rete	0,01
Valore inziale dei bias di rete	0,01
Valore iniziale di portafoglio	1
Dimensione della memoria (in time step)	3.000
Beta β	0,05

Tabella 4 settaggi di rete utilizzati.

Con particolare attenzione al tasso di esplorazione ε, questo viene inizialmente settato con valore pari a 1 con un periodo di decadimento di 15.000 time step; questo significa che il tasso di esplorazione sarà estremamente elevato nei primi step di allenamento per permettere all'agente di esplorare il più possibile le soluzioni disponibili e mantiene un tasso di decadimento costante nei primi 15.000 step fino a raggiungere il valore minimo di 0,001 che verrà mantenuto costante per i successivi step di training; l'agente quindi adotterà, con una probabilità del 99,9%, le azioni che massimizzano la ricompensa. Mantenere comunque un valore minimo di esplorazione permetterà all'agente di cercare, anche se con probabilità estremamente bassa, soluzioni non identificate come ottime per continuare il processo di esplorazione ed evitare fenomeni di overfitting. I time step per episodio identificano il numero di time step necessari affinché l'agente termini un episodio, rappresentato da indicativamente 365 giorni di quotazioni, nella fase di test del modello, mentre la dimensione del batch di allenamento indica su quanti time step avviene ogni singola sessione di allenamento, l'allenamento complessivo si concretizza quindi in 400 differenti sessioni minori di allenamento. La dimensione della memoria

[51] Per trading period si intende la durata di una sessione di trading, in questo caso trading period = 2 indica che l'agente ha modo di riallocare le risorse del portafoglio ogni 2 step temporali, ovvero ogni due giorni borsa.

indica il numero massimo di esperienze contenute nel batch di memoria, utilizzato per identificare i TD-error necessari per il processo di allenamento, una volta raggiunta la capacità massima di memoria la prima esperienza contenuta viene sostituita dall'ultima esperienza generata, in questo modo la memoria contiene sempre le ultime 3000 esperienze raccolte dall'agente. Infine i valori di tutti i pesi e bias della rete viene inizializzato al valore di 0,01, questa è una pratica comune in cui viene scelto un valore prossimo allo 0 per accelerare il processo di allenamento della rete, se questi parametri venissero inizializzati con valore pari a 0 potrebbero nascere problemi nella fase di allenamento del modello, in particolare il fenomeno della scomparsa del gradiente e un valore degli output dei singoli neuroni sempre uguale per ogni passaggio in avanti delle informazioni di input.

5. RISULTATI DELL'ESPERIMENTO

Dopo aver evidenziato nel precedente capitolo la struttura ed il funzionamento dell'algoritmo in questo ultimo capitolo vengono esposti i risultati ottenuti per i diversi settaggi implementati. L'algoritmo è stato implementato per due diversi settaggi di input e per due diversi settaggi di discretizzazione delle azioni, le performance registrate per ogni settaggio verranno poi confrontate con le performance registrate dall'indice S&P 500 e dalle performance registrate da due modelli di portafogli classici, il portafoglio equipesato e un portafoglio ribilanciato uniformemente in modo costante.

I diversi settaggi riguardano la modifica delle dimensioni del tensore dei prezzi e due diversi settaggi per la discretizzazione della selezione delle azioni. Il tensore dei prezzi viene testato utilizzando 5 e 10 giorni di osservazioni, queste due finestre temporali identificano una e due settimane di quotazioni di borsa e sono dimensioni largamente utilizzate nella costruzione di indici di analisi tecnica. Per quanto riguarda la discretizzazione delle azioni queste sono testate con una divisioni minime del capitale di 20% e 10%, ciò significa che nel primo caso l'agente avrà modo di allocare il capitale per divisioni del 20% del capitale, mentre nel secondo caso avrà modo di allocare il capitale per divisione del 10%; nel secondo caso l'agente dovrebbe quindi essere in grado di agire in modo più puntuale nella scelta delle allocazioni avendo modo di gestire il portafoglio con maggiore libertà.

Una volta allenato l'algoritmo questo viene utilizzato, per ognuno dei settaggi proposti, sulle serie di dati utilizzati nella fase di test. Per ogni diverso settaggio vengono effettuate 10 esecuzioni casuali[52], vengono quindi riportati i risultati medi ottenuti in questa fase di test. La scelta di riportare i risultati medi basa sulla necessità di presentare una visione più robusta delle prestazioni dell'algoritmo. La scelta di riportare i risultati medi basa sulla necessità di presentare una visione più robusta delle prestazioni dell'algoritmo, considerando che la singola esecuzione può essere soggetta a variazioni casuali e fluttuazioni dei risultati, in particolar mode se la fase di allenamento viene effettuata su una base di dati soggetti a variazioni casuali, quali possono essere i rendimenti dei titoli azionari, che rendono più difficoltoso l'ottenimento di informazioni utili distinguendole dal rumore generato dalla variazione dei dati.

[52] In questo modo ogni esecuzione del modello durante la fase di test è indipendente dalle altre, le prestazioni ottenute dal modello in un'esecuzione non influenza in alcun modo le altre esecuzioni.

I risultati ottenuti vengono confrontati sui due diversi periodi di test utilizzati, vengono confrontati tra loro gli algoritmi per i diversi settaggi di input dei dati e per le diverse modalità di discretizzazione delle azioni, infine vengono confrontati globalmente i risultati ottenuti dai diversi settaggi utilizzati.

Infine, l'algoritmo proposto nel paper di riferimento non considera costi di transazione quindi l'algoritmo è stato inizialmente sviluppato con le medesime specifiche, si è comunque provato ad adattare l'algoritmo per far sì che potesse considerare i costi di transazione con le modalità identificate da Jiang et al. (2017), i risultati di questa implementazione sono riportati in coda al presente capitolo.

5.1 Confronto per settaggi di input dei dati nel primo periodo

La versione dell'algoritmo implementata dal Gao et al. (2020) prevede un tensore dei prezzi di dimensioni (5,10,4) quindi utilizza come input al tempo t le ultime 10 osservazioni registrate nel tensore dei prezzi X_t. Nella fase di allenamento il modello sviluppato pare esser stato in grado di identificare correttamente i pattern che caratterizzano le fluttuazioni dei titoli considerati andando ad aumentare il valore di portafoglio nel corso dell'allenamento. Essendo un allenamento ripetuto innumerevoli volte sulla stessa base di dati non ha particolare utilità andare a riportare il rendimento ottenuto, viene comunque riportata l'evoluzione grafica del valore di portafoglio alla Figura 40. È interessante notare come il valore del portafoglio aumenti fortemente tra i 10.000 e i 15.000 step, per poi assestarsi seppur con importanti fluttuazioni di valore. Quest'evoluzione è in linea con quanto atteso dato il settaggio del fattore di esplorazione ε riportato alla Tabella 4.

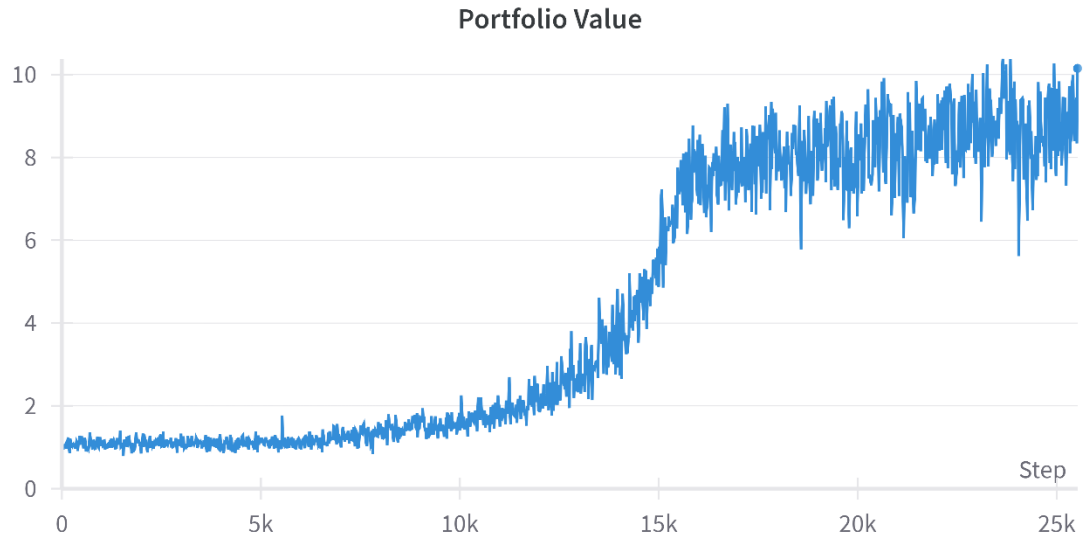

Figura 40 evoluzione del valore di portafoglio nella fase di allenamento perX_t (5,10,4) (elaborazione WandB).

Questo settaggio ha dimostrato buoni risultati nel primo periodo di test andando a registrare un rendimento medio del portafoglio del 12,93% ed un valore dell'indice di Sharpe medio[53] di 2,39. Per quanto riguarda le singole esecuzioni 7 esecuzioni su 10 registrano un rendimento positivo con un rendimento massimo del 21,6% ed un rendimento minimo del -4,79%, la forchetta entro cui si muovono i rendimenti finali lascia intendere che il processo di allenamento non sia riuscito a definire una strategia di trading univoca e precisa, se così fosse gli scostamenti di performance tra un'esecuzione e l'altra dovrebbero dipendere unicamente dal valore del fattore di esplorazione ε, settato per la fase di test al valore minimo di 0,001, che non dovrebbe quindi avere un impatto decisivo nella definizione delle azioni. Il grafico riportato in Figura 41 evidenzia l'evoluzione del valore di portafoglio mediamente registrata nelle 10 esecuzioni utilizzate nella fase di test. La banda chiara indica i valori massimi e minimi registrati nella fase di test.

[53] Calcolato come media dei valori dell'indice di Sharpe per ogni esecuzione considerando che il tasso di rendimento risk-free è pari a 0.

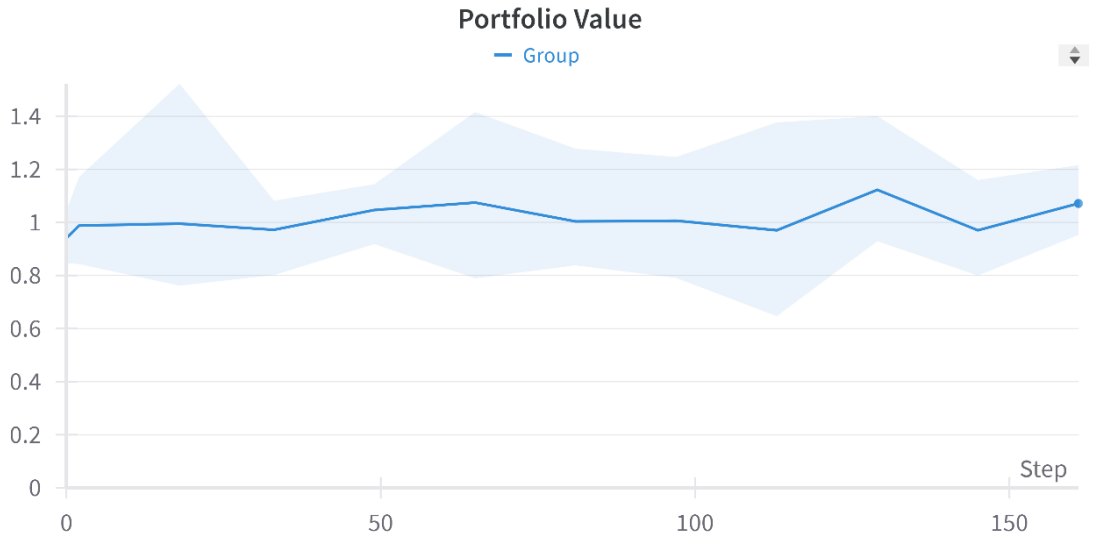

Figura 41 evoluzione del valore di portafoglio medio con tensore di dimensioni (5,10,4) (elaborazione WandB).

Il processo di allenamento sembra comunque essere riuscito a identificare dei pattern nel movimento dei prezzi delle azioni considerati, tale da permettere all'agente di gestire con discreto successo il portafoglio pur non riuscendo a performare in modo costante, valutando singolarmente le 10 esecuzioni della fase di test si nota infatti come raramente le simulazioni seguano percorsi simili. In particolare, analizzando le scelte intraprese dall'agente nelle esecuzioni si nota come solo in pochi casi l'agente seleziona la stessa azione nel medesimo step temporale come si può notare dal grafico in Figura 42.

Figura 42 selezione dell'azione per singola esecuzione con tensore di dimensioni (5,10,4) (elaborazione WandB).

Il grafico precedente riporta la selezione delle azioni per step temporali, le diverse azioni sono etichettate con un numero (da 0 a 251) e sono ordinate sequenzialmente per la variazione di capitale, ciò significa che azioni con numero di etichetta vicino prevedono composizioni di portafoglio non particolarmente dissimili; dal grafico in Figura 42 si può quindi notare che raramente nel medesimo step temporale l'agente ha intrapreso la medesima azione in differenti sessioni di test.

Successivamente sono state testate le performance del modello andando a modificare le dimensioni del tensore dei prezzi perché considerasse solo le ultime 5 osservazioni per ogni step temporale *t*, il tensore presenta quindi dimensioni (5,5,4), questo settaggio si è dimostrato meno efficiente andando a registrare un rendimento medio del 6,7% nella fase di test ed un indice di Sharpe medio di 1.018. Analizzando le singole esecuzioni si nota come 5 su 10 abbiano registrato rendimento positivo, con un rendimento massimo del 27% e minimo del -17,39%. Oltre a peggiori performance questo settaggio ha dimostrato una deviazione standard dei valori di portafoglio alla fine del periodo di test del 15%, superiore rispetto alla varianza calcolata allo stesso modo per il settaggio precedentemente esposto (9,439%). Questi risultati indicano il modello sviluppato per la gestione di portafoglio faccia più fatica ad indentificare le variazioni dei prezzi dei titoli con finestre di dati ridotte.

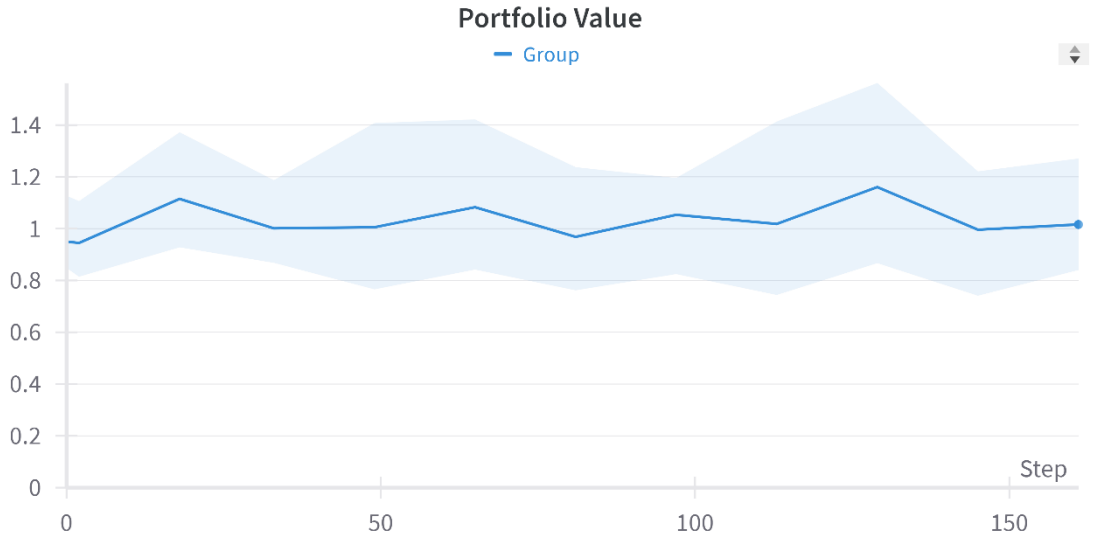

Figura 43 evoluzione del valore di portafoglio medio con tensore di dimensioni (5,5,4) (elaborazione WandB).

L’implementazione del modello utilizzando solo 5 osservazioni per ogni step temporale si dimostra quindi meno efficiente, sia in termini di performance medie registrate, sia in termini di capacità del modello di identificare una strategia di investimento precisa.

Confrontando le performance registrate dal modello implementato, per i due diversi settaggi, e due modelli alternativi di gestione di portafoglio si può notare come, pur non avendo registrato performance estremamente positive, l’algoritmo sia riuscito a superare nello stesso periodo un portafoglio equipesato composto dai 5 titoli considerati ha registrato un rendimento finale negativo del -1,51% con un valore dell’indice di Sharpe pari a -0.7596 mentre un portafoglio ribilanciato uniformemente in modo costante ha ottenuto un rendimento finale del 0,24% e un valore dell’indice di Sharpe di 0,1184. I risultati ottenuti dal modello nelle diverse configurazioni vengono confrontate inoltre con il rendimento ottenuto dall’indice di mercato S&P 500, in questo caso il modello non riesce ad avvicinarsi alle performance registrate dall’indice che registra un rendimento del 22,96% e un indice di Sharpe di 12,27. Confrontare il modello proposto con l’indice S&P 500 non permette un’accurata valutazione delle performance del modello proposto, data la sostanziale differenza che corre tra un indice composto da 500 titoli contro un portafoglio costituito da 5 titoli, ma permette di identificare le performance che il mercato di riferimento ha globalmente registrato nel periodo analizzato. Il valore di indice di Sharpe viene utilizzato nel confronto tra performance per avere un indicazione circa la rischiosità registrata dal modello di portafoglio, i valori ridotti registrati sono in linea con la modalità con cui è stato implementato l’algoritmo in quanto compito dell’agente è gestire il portafoglio con l’unico obiettivo di massimizzare il rendimento disinteressandosi dei rischi sopportati.

L’analisi delle performance registrate rispetto al benchmark devono comunque considerare il periodo di borsa particolare in cui il modello è stato testato, il crollo delle quotazioni di borsa in concomitanza con l’esplosione della pandemia da Covid-19 ha sicuramente inficiato negativamente su portafogli che non prevedendo riallocazioni del capitale, né tantomeno la possibilità di disinvestire. Infatti confrontando la performance del modello con quanto realizzato dall’indice di borsa S&P 500, pur con le dovute cautele dovute al numero limitato di titoli inseriti in portafoglio, si nota come il modello implementato non sia riuscito a ottenere risultati pienamente in linea con l’andamento generale del mercato. I risultati sono sinteticamente riportati alla seguente Tabella 5.

	Rendimento al tempo T	Indice di Sharpe
X_t (5,10,4)	12,93%	2,39
X_t (5,5,4)	6,7%	1,018
Portafoglio equipesato	-1,51%	-0,7596
Portafoglio equipesato ribilanciato	0,24%	0,1184
S&P 500	22,96%	12,27

Tabella 5 confronto performance registrate dai settaggi dell'algoritmo e i benchmark utilizzati.

5.2 Confronto per settaggi di input dei dati nel secondo periodo

Uno dei problemi che potenzialmente affligge i modelli di RL riguarda l'adattabilità del modello a differenti basi di dati, per valutare se il modello sviluppato è efficiente indipendentemente dai dati utilizzati per la fase di test e allenamento i modelli vengono testati con i medesimi settaggi su un altro periodo temporale.

La fase di testing per il modello con tensore dei prezzi di dimensioni (5,10,4) ha realizzato ne periodo dal 20/04/2022 al 02/10/2023, un rendimento medio dell'8,32% e uno Sharpe ratio di 1.0432, registrando performance leggermente peggiori rispetto al periodo precedente con un rendimento minimo del -2,956% e un rendimento massimo del 29,9%, con una varianza dei valori di portafoglio finali del 10,48% nonostante i rendimenti minimi e massimi siano leggermente superiori rispetto a quanto ottenuto nel periodo precedente il rendimento mediano registra un valore del 4,622%, indicando come il rendimento medio sia in larga parte influenzato da pochi rendimenti particolarmente elevati; andando ad analizzare i risultati ottenuti nelle singole esecuzioni infatti si nota come soltanto 5 esecuzioni su 10 abbiano registrato un rendimento positivo, pur restando i rendimenti negativi molto contenuti.

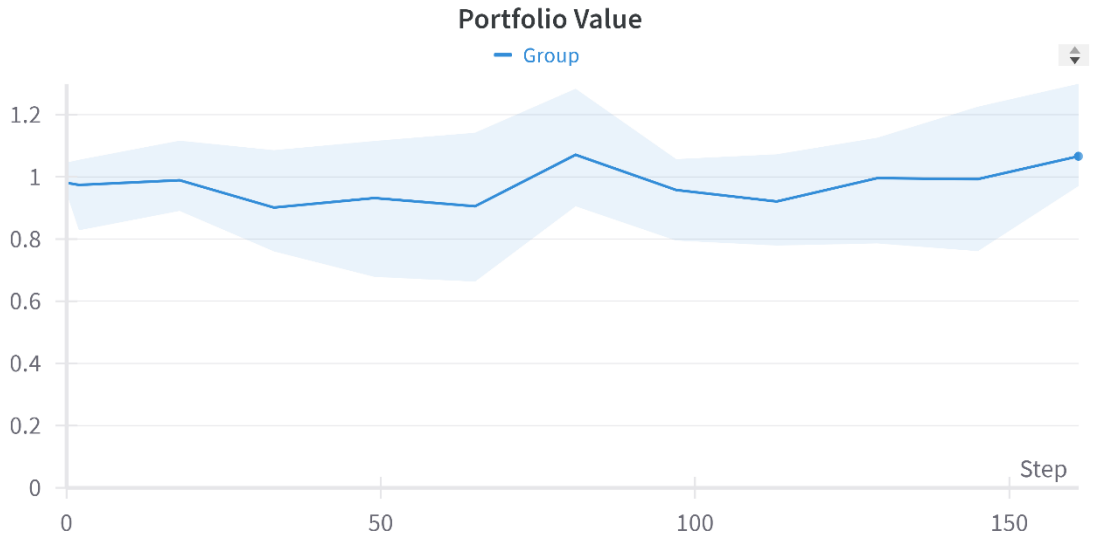

Figura 44 evoluzione del valore di portafoglio medio con tensore di dimensioni (5,10,4) (elaborazione WandB).

Come per il periodo precedente il settaggio con il tensore di prezzi di dimensioni X_t (5,5,4) ha registrato performance inferiori andando a registrare un rendimento medio negativo del -4,32% e un valore di indice di Sharpe del -0,5998; con questo settaggio soltanto 3 esecuzioni su 10 hanno registrato un rendimento positivo con un rendimento massimo dell'8% ed un rendimento minimo del -15,76%. Il valore finale di portafoglio si assesta in un range che va dal valore minimo di 0,8542 ad un massimo di 1,08 facendo registrare una deviazione standard dei risultati del 7,47%, inferiore rispetto al settaggio precedente.

Le peggiori performance registrate da entrambi i settaggi nel secondo periodo possono essere giustificati da una situazione di generale turbolenza del mercato causata dallo scoppio del conflitto tra Russia e Ucraina nel febbraio 2022 in un periodo in cui il mercato finanziario si stava riprendendo dalla crisi da Covid-19.

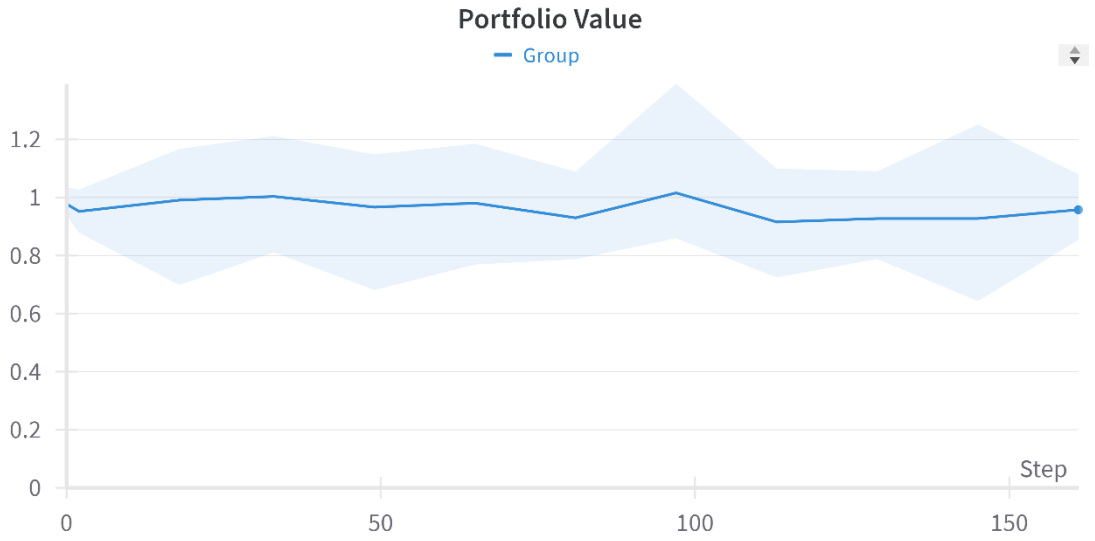

Figura 45 evoluzione del valore di portafoglio medio con tensore di dimensioni (5,5,4) (elaborazione WandB).

Andando ad analizzare il rendimento dell'indice di mercato S&P 500, utilizzandolo come *proxy* di mercato, si evince come il rendimento registrato dall'indice sia negativo (-3,91%) nel periodo selezionato, andando quindi a giustificare il rendimento inferiore rispetto a quanto ottenuto nel periodo precedentemente analizzato. Inoltre, la qualità dei dati inserita in un modello di RL è estremamente importante per garantire una buona performance del modello, i dati riportati alle Tabelle 2 e 3, come i grafici riportati dalla Figura 31 alla Figura 35 della corrente tesi, evidenziano come la fase di allenamento del modello possa essere, per il secondo periodo, influenzata dalla maggior variabilità registrata nei prezzi delle azioni considerate. Inoltre i rendimenti medi dei cinque titoli selezionati per il secondo periodo, riportati sempre alla Tabella 3, sono negativi per quattro titoli su cinque con il solo titolo MSFT in grado di registrare un rendimento medio positivo nel periodo.

Confrontando le performance ottenute dal modello sviluppato rispetto alle performance dei benchmark si nota come l'algoritmo di RL sia stato in grado di operare con successo nel mercato finanziario anche in condizioni di forte turbolenza. In particolare i portafogli a gestione passiva utilizzati come benchmark hanno registrato performance estremamente negative, come nel periodo precedente va comunque considerato che in periodi di forte turbolenza, specialmente se il confronto si concretizza con un'analisi di breve periodo, una gestione di portafoglio in grado di operare disinvestimenti ha uno strumento

aggiuntivo per superare questi periodi senza avere impatti considerevoli sul valore del portafoglio gestito.

	Rendimento al tempo T	Indice di Sharpe
X_t (5,10,4)	8,32%	1,043
X_t (5,5,4)	-4,32%	-0,5998
Portafoglio equipesato	-26,32%	-20,8094
Portafoglio equipesato ribilanciato	-24,99%	-19,225
S&P 500	-3,91%	-3,07

Tabella 6 confronto performance registrate dai settaggi dell'algoritmo e i benchmark utilizzati.

Analizzando nello specifico l'esecuzione che ha generato il maggior valore di portafoglio finale (1,299, con un rendimento del 29,9%) si evince come l'agente sia riuscito ad intercettare il trend in rialzo che ha caratterizzato il titolo MSFT nella seconda metà del periodo andando ad investire la totalità del capitale nel titolo MSFT nell'ultimo mese di transazioni. Date le condizioni di mercato avverse il modello implementato è comunque riuscito ad ottenere risultati mediamente positivi, nonostante ciò una criticità riguarda certamente l'incapacità da parte dell'agente di definire una strategia di trading univoca, causando una certa casualità nei risultati registrati.

5.3 Confronto per diversi settaggi di discretizzazione delle azioni

La difficoltà da parte dell'agente nel definire una strategia di trading univoca e seguita senza grandi variazioni tra un'esecuzione e l'altra può essere in parte dovuta alla scarsa granularità nelle possibilità d'investimento. Con i settaggi finora utilizzati l'agente ha dovuto adattare le decisioni prese ad un framework che lo costringe ad investire larghe quote di capitale contemporaneamente, andando a rendere molto approssimativa la policy seguita rispetto alle previsioni effettuate dalla rete neurale. Per valutare se ampliando le scelte a disposizione dell'agente le performance migliorano viene implementato un diverso settaggio per la discretizzazione delle azioni che prevede la possibilità di investire il capitale per quote del 10% anziché del 20%. Questo settaggio, se da un lato dovrebbe garantire una maggiore vicinanza tra la scelta ideale dato lo stato in cui si trova l'agente e le scelte effettivamente disponibili, dall'altro aumenta notevolmente la complessità dell'ambiente e la dimensione dello spazio stato-azione andando incontro ad una serie di potenziali problematiche in particolare nella fase di allenamento del modello; l'aumento

dello spazio stato-azione può infatti portare alla cosiddetta *maledizione della dimensionalità* che implica un esplorazione meno efficiente, instabilità dell'addestramento (a causa della maggior complessità della convergenza) e uno scarso trasferimento di conoscenza da un ambiente all'altro.

Per mitigare queste problematiche viene modificato il periodo di allenamento che occupa quindi tutte l'intero dataset, ad esclusione dei dati dal 20/04/2022 al 02/10/2023 che verranno utilizzati nella fase di test; la fase di allenamento viene quindi suddivisa in tre diverse fasi di allenamento eseguite su tre diversi dataset,[54] al termine di ogni fase di allenamento i parametri di rete vengono salvati e inizializzati come parametri iniziali per la fase successiva, viene inoltre adottato un rudimentale processo di *learning rate annealing*, un processo di correzione del tasso di apprendimento proposto da Li (2020) che prevede la riduzione graduale del tasso di apprendimento durante la fase di allenamento, per questo il tasso di apprendimento viene inizialmente settato al valore di 0,00075 e poi ridotto prima a 0,0005 e infine a 0,00025; in questo modo l'allenamento del modello si divide in tre fasi ideali: una prima fase di inizializzazione, una seconda fase di accelerazione del processo di allenamento, una terza fase di *fine-tuning*[55] per la definizione dei parametri, anche il valore del tasso di esplorazione ε viene modificato rispetto a quanto esposto in precedenza, viene settato un periodo di decadenza più lungo nel primo set di allenamento pari a 30.000 time step, ridotto a 15.000 time step nel secondo e terzo set di allenamento, nell'ultima fase di allenamento inoltre il valore di ε viene inizializzato a 0,75 anziché 1, andando a ridurre la fase esplorativa a favore di una fase di sfruttamento più marcata. Data la maggior complessità e lunghezza della fase di allenamento[56] questa implementazione del modello avviene soltanto per le dimensioni (5,10,4) del tensore X_t.

L'implementazione del modello con le specifiche descritte si rivela meno performante rispetto al medesimo modello con uno spazio stato-azione ridotto, il rendimento medio ottenuto risulta infatti negativo (-3,49%) con un valore di indice di Sharpe di -0,8396, inoltre solo 4 esecuzioni su 10 hanno ottenuto rendimento positivo. Il rendimento massimo nella fase di test è dell'8,5% mentre la performance peggiore ha ottenuto un

[54] La fase di allenamento avviene quindi sui due dataset di allenamento utilizzati in precedenza e sul dataset utilizzato per la prima fase di test vista nei settaggi precedenti.
[55] Processo di raffinamento dei parametri su un modello già addestrato.
[56] Sono state richieste complessivamente 3 ore e 36 minuti per il processo di allenamento del modello contro i 39 minuti richiesti per l'allenamento del modello con 252 azioni disponibili.

rendimento di -20,69%; va comunque notato come questa sia l'unica prestazione particolarmente negativa, il secondo risultato peggiore risulta infatti essere del -8,35%. La deviazione standard dei valori di portafoglio alla fine del periodo risulta pari a 7,33% risultando leggermente inferiore rispetto a quella registrata in tutti i settaggi precedenti.

Le performance inferiori possono essere spiegate da un fenomeno di overfitting nel processo di allenamento che ha portato l'agente a "imparare a memoria" l'evoluzione dei prezzi nel corso del processo di allenamento, senza riuscire a distinguere correttamene il rumore generato dalle fluttuazioni dei prezzi dai pattern che ne caratterizzano i movimenti; in questo modo l'agente ha replicato informazioni apprese durante la fase di allenamento anche alla fase di test ignorando la differenza di dataset utilizzati. Date le performance positive del modello durane la fase di allenamento si esclude un problema di sottodimensionamento della rete, ovvero che la rete implementata non sia sufficientemente complessa per far sì che l'agente possa raccogliere le informazioni necessarie durante la fase di allenamento.

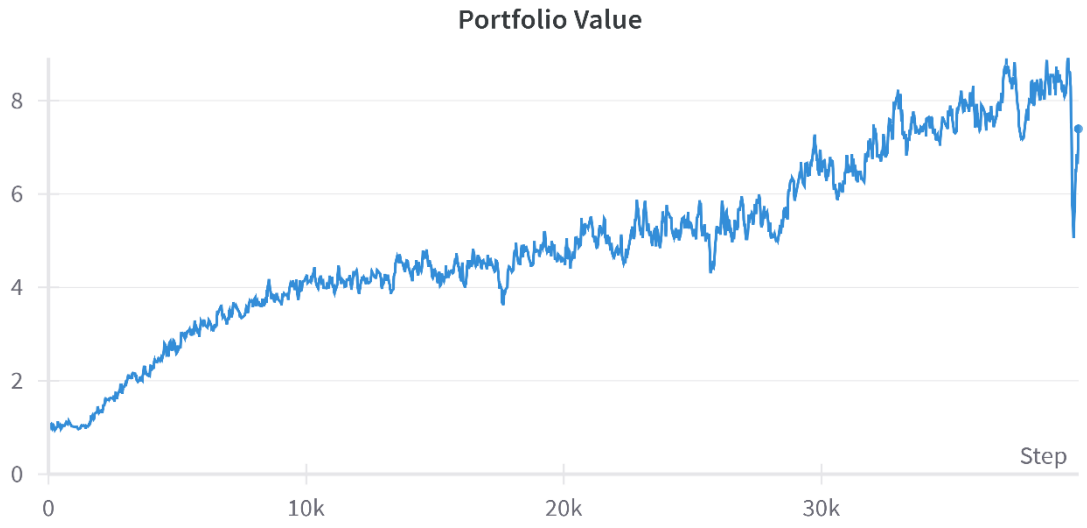

Figura 46 evoluzione del valore di portafoglio nell'ultima fase di allenamento (elaborazione WandB).

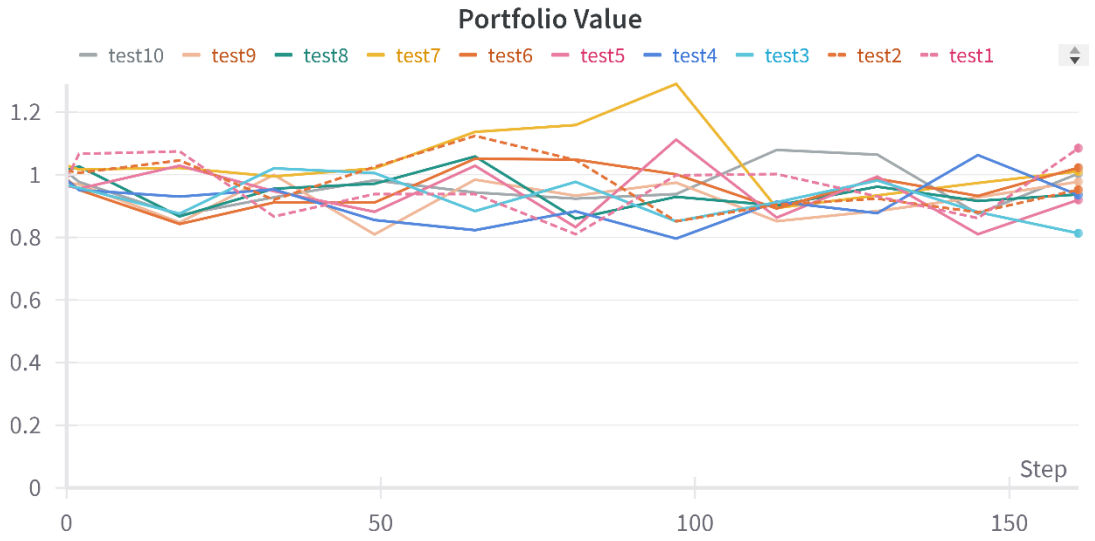

Figura 47 evoluzione del valore di portafoglio per le 10 esecuzioni nella fase di test (elaborazione WandB).

	Rendimento al tempo T	Indice di Sharpe
X_t (5,10,4) a 252 azioni	8,32%	1,043
X_t (5,10,4) a 3003 azioni	-3,49%	-0,8396
Portafoglio equipesato	-26,32%	-20,8094
Portafoglio equipesato ribilanciato	-24,99%	-19,225
S&P 500	-3,91%	-3,07

Tabella 7 confronto performance registrate dai settaggi dell'algoritmo e i benchmark utilizzati.

Nonostante le performance inferiori rispetto al framework con minor numero di azioni disponibili il modello è riuscito a ottenere un risultato superiori rispetto ai modelli utilizzati come benchmark e all'indice di mercato S&P 500.

5.4 Risultati con costi di transazione

Il modello viene infine implementato in modo da considerare i costi di gestione del portafoglio, definiti in percentuale fissa dell'1% del capitale transato. I costi, non essendo considerati nel framework proposto da Gao et al. (2020) vengono applicati con la stessa logica vista nel framework implementato da Jiang et al. (2017). Per questo viene implementato un sistema di calcolo del valore di portafoglio a due step, che ricalca quanto esposto al Paragrafo 3.3 della presente tesi.

Ricordando che al termine del tempo t viene identificato il valore di portafoglio y_t' secondo l'equazione

$$y_t' = y_t w_t \cdot \mu_t$$

(5. 1)

l'applicazione dei costi di gestione comporta la variazione del valore di portafoglio quando l'allocazione di capitale subisce una variazione, per cui il valore del portafoglio all'inizio del periodo $t + 1$ viene calcolato come

$$y_{t+1} = y_t'(1 - c_t) * \langle \mu_{t+1}, w_{t+1} \rangle$$

(5. 2)

dove $\langle \mu_{t+1}, w_{t+1} \rangle$ è il prodotto scalare tra i rendimenti degli asset tra il termine del periodo t e l'inizio del periodo $t + 1$[57] e i nuovi pesi di portafoglio, infine l'elemento c_t indica i costi di portafoglio calcolati, come da seguente equazione

$$c_t = 0{,}01 * \sum_{i=1}^{m} \left| w_{t,i}' - w_{t,i} \right|$$

(5. 3)

Il portafoglio con costi di transazione è stato testato solo con vettore di dimensioni (5,10,4) e nel primo periodo di test. Poiché si voleva verificare se l'algoritmo fosse riuscito a performare anche considerando i costi di transazione si è ritenuto superfluo verificare le performance del modello con tutti i settaggi visti finora.

Le performance registrate in questo framework sono state generalmente negative, la fase di allenamento del modello si è rivelata mediamente meno efficace rispetto ai settaggi utilizzati in precedenza come mostrato dall'evoluzione del valore di portafoglio riportata alla Figura 48.[58] Nella fase di test il modello ha registrato un notevole variazione dei risultati con una deviazione standard del valore di portafoglio finale del 11,99% indicando come l'agente abbia fatto particolarmente fatica ad individuare dei parametri di rete ben definiti, influenzando quindi la scelta della policy da seguire. Per quanto riguarda i

[57] Si ricordi che il framework implementato considera anche i prezzi di apertura, per cui, venendo il portafoglio ribilanciato tra la chiusura del giorno *t* e l'apertura del giorno *t+1* il vettore dei rendimenti μ_{t+1} considera la variazione tra il valore di chiusura e il valore di apertura dei cinque asset considerati nel framework.

[58] Il confronto viene fatto con l'evoluzione del valore di portafoglio riportato in Figura 42.

risultati ottenuti questo framework registra un rendimento medio del -1,99% con un rendimento massimo del 17,8% e un rendimento minimo del -22,68%, un indice di Sharpe medio di -0,2147 e con sole 3 esecuzioni su 10 che concludono con un rendimento positivo.

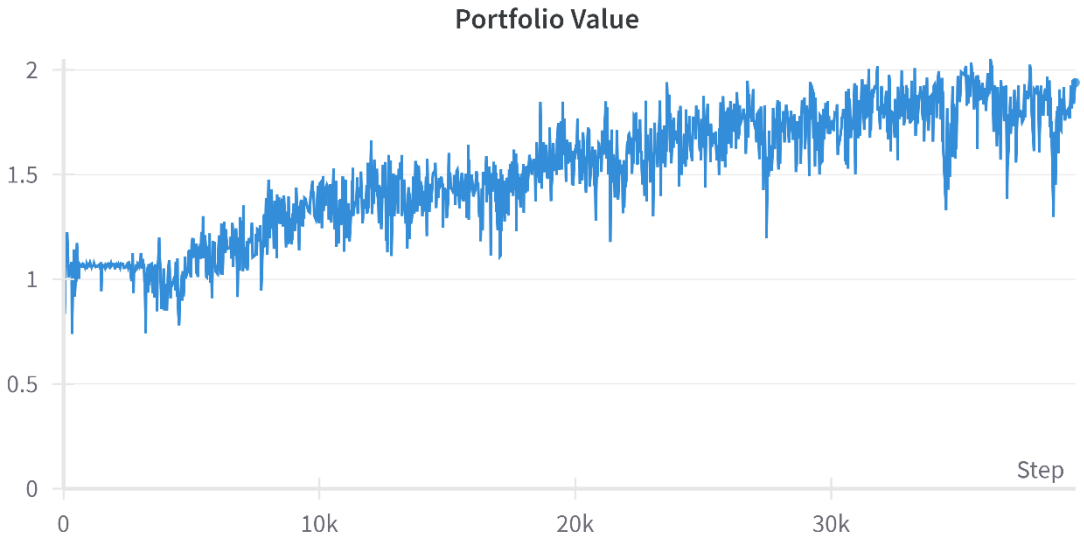

Figura 48 evoluzione del valore di portafoglio nell'ultima fase di allenamento (elaborazione WandB).

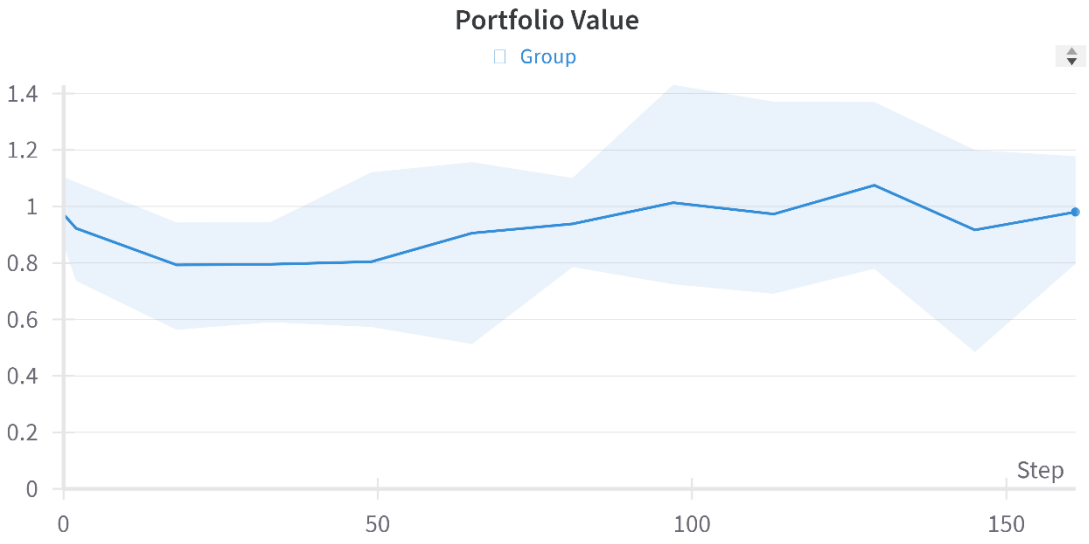

Figura 49 evoluzione del valore di portafoglio nella fase di test (elaborazione WandB).

	Rendimento al tempo T	Indice di Sharpe
X_t (5,10,4)	12,93%	2,39
X_t (5,10,4) con costi di gestione	-1,99%	-0,2147
Portafoglio equipesato	-1,51%	-0,7596
Portafoglio equipesato ribilanciato	0,24%	0,1184
S&P 500	22,96%	12,27

Tabella 8 confronto performance registrate dai settaggi dell'algoritmo e i benchmark utilizzati.

A livello di operatività del modello è interessante notare come questo, pur non riuscendo a restituire risultati particolarmente positivi, sia riuscito ad identificare la negatività derivante dall'applicazione dei costi di transazione, analizzando il numero di riallocazioni eseguite nella fase di test si nota come in questo caso il numero di azioni, e quindi di riallocazioni di capitale sia notevolmente inferiori rispetto agli altri framework implementati, mediamente considerando i costi di transazioni i pesi di portafoglio vengono corretti 7,3 volte nella fase di test del modello contro le 10,7 correzioni compiute dai modelli con dimensione del tensore Xt (5,10,4).

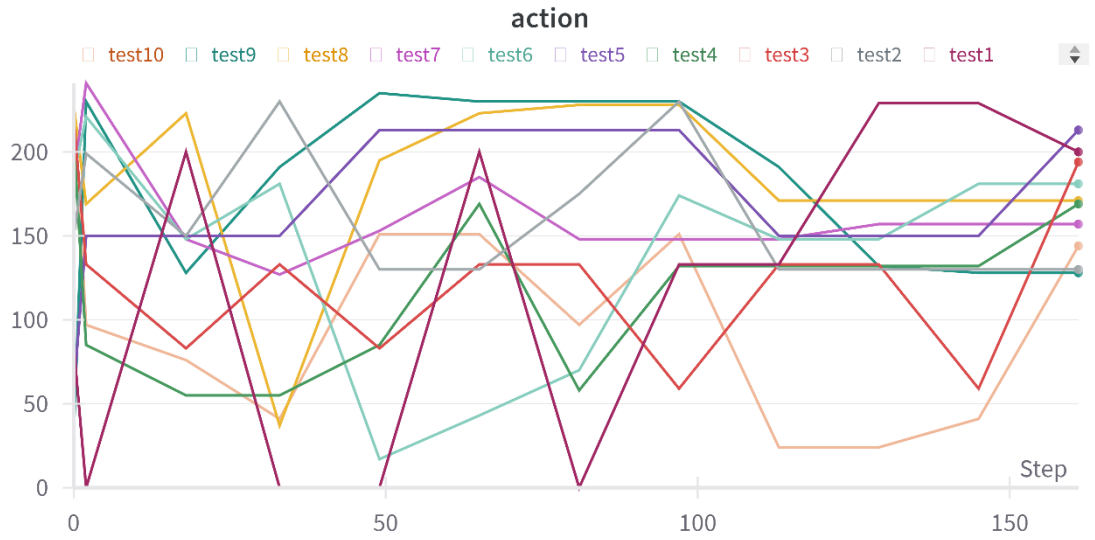

Figura 50 selezione delle azioni per singola esecuzione (elaborazione WandB).

I costi di gestione essendo calcolati come riduzione del valore di portafoglio e non come riduzione del rendimento dei singoli asset che compongono il portafoglio dovrebbe garantire all'agente di riuscire a valutare correttamente le azioni di gestione da intraprendere per massimizzare il rendimento, sfortunatamente il framework proposto non è efficace, una soluzione, non implementata in questa tesi, potrebbe essere quella di

considerare il rendimento degli asset già al netto dei costi di transazione, il modello infatti si è dimostrato sufficientemente capace di identificare i pattern che caratterizzano i movimenti dei prezzi dei singoli titoli.

CONCLUSIONI

In questa tesi è stato sviluppato e implementato in ambiente Python un algoritmo di deep reinforcement learning volto alla soluzione di un problema di gestione di portafoglio. Come esposto brevemente nel Paragrafo 3.4 della corrente tesi negli ultimi anni lo sviluppo tecnologico ha permesso l'adozione di soluzioni afferenti il campo del machine learning o dell'intelligenza artificiale per la soluzione, o semplificazione, di problemi tipicamente affrontati dagli istituti finanziari. Nonostante l'utilizzo di queste tecniche informatiche avanzate sia utilizzato in svariati ambiti la gestione di portafogli finanziari resta un ambito in cui l'implementazione di queste tecnologie resta piuttosto limitato con l'uso incentrato prevalentemente a compiti di supporto piuttosto che alla vera e propria gestione del capitale.

Il framework proposto nella tesi giustifica l'utilizzo ancora limitato del machine learning per la gestione di portafogli finanziari, pur avendo dimostrato buone potenzialità il modello sviluppato presenta sostanziali criticità pur essendo un modello estremamente basilare, l'ampliamento del problema di gestione del portafoglio in modo che possa considerare un numero più consistente di asset, se non addirittura tutti gli asset quotati presso un mercato, potrebbe portare a notevoli complicazioni sia dal lato operativo, con tempi di calcolo estremamente lunghi che vanificherebbero di fatto l'utilità del modello, sia per quanto riguarda la precisione del modello. Un ulteriore limitazione all'utilizzo pratico del modello riguarda la restituzione di output differenti durante la medesima fase di test del modello, per quanto i risultati siano mediamente positivi per alcuni settaggi è necessario considerare che non tutte le esecuzioni hanno fornito risultati in linea con le attese e che le esecuzioni che hanno ottenuto risultati positivi si caratterizzano per selezioni dei titoli differenti; a livello pratico il modello proposto non è quindi in grado di fornire una soluzione univoca, e quindi applicabile, al problema affrontato. Infine è necessario considerare i potenziali problemi di adattabilità del modello proposto, la differenza di performance registrate con settaggi diversi dimostra come il modello sia stato implementato e affinato con lo scopo di gestire un portafoglio con determinati settaggi e che, andando a modificare le modalità con cui viene affrontato il medesimo problema, le performance possono variare significativamente; d'altro canto va comunque considerato come il modello proposto sia in grado di adattarsi a diverse situazioni e a diversi mercati.

Ulteriore limitazione riguarda il problema di spiegabilità dei risultati ottenuti, pur avendo descritto in modo dettagliato la struttura informatica del modello non è possibile fornire una spiegazione sul perché l'agente abbia selezionato l'una o l'altra possibilità d'investimento ad ogni step temporale, se la capacità di operare senza la necessità di sottostare a regole d'investimento predefinite può portare a vantaggi operativi, in quanto la scelta della strategia d'investimento non è vincolata e quindi può essere adattata alle diverse situazioni, dall'altro lato della medaglia si presenta la difficoltà di riconoscere agevolmente la logica che ha guidato la selezione delle azioni; mentre per il problema limitato e didascalico affrontato in questa tesi la questione della spiegabilità dei risultati può assumere tratti marginali l'implementazione di modelli simili da parte di istituti finanziari reali, che devono sottostare a vincoli normativi e al diritto degli *stakeholders* di conoscere la ragione per cui si sono ottenuti determinati risultati, può incontrare sostanziali limitazioni.

Nonostante le limitazioni di operatività pratica appena esposte il modello di deep reinforcement learning proposto si è dimostrato sufficientemente efficiente per le finalità di ricerca secondo le quali è stato implementato, riuscendo a registrare performance positive per buona parte dei settaggi proposti. Date le buone performance dimostrate da questa tipologia di modelli lo sviluppo tecnologico e gli sforzi di ricerca possono portare all'implementazione di modelli più complessi e avanzati che meglio si adattano ad una rappresentazione più realistica del problema di gestione di portafogli.

Per concludere, il mercato finanziario si caratterizza per un'elevatissima dinamicità che rende spesso particolarmente complesso adottare principi e modelli statici e predefiniti per la soluzione del problema di gestione di portafogli; la capacità dimostrata dai modelli informatici sviluppati negli ultimi anni di apprendere i pattern che caratterizzano i movimenti dei prezzi di mercato, e la capacità di riaggiornare il processo decisionale che porta alla restituzione degli output, rende lo sviluppo e l'implementazione dei modelli di machine learning un ambito imprescindibile per garantire alle gestioni attive di capitale di ottenere performance sistematicamente superiori rispetto al mercato.

APPENDICE

File “action_discretization.py”.

```
from itertools import combinations
from scipy.special import comb
import numpy as np

asset_num = 6
division = 5
#This function is to build action space by combination
def Action_discretization(asset_num, division):

    item_num = asset_num + division - 1
    action_num = int(comb(item_num, asset_num - 1))
    actions = {}
    pointer = 0

    for c in combinations(np.arange(item_num), asset_num - 1):
        action = np.zeros(asset_num)
        for i in range(len(c) - 1):
            action[i + 1] = c[i + 1] - c[i] - 1
        action[0] = c[0]
        action[-1] = item_num - c[-1] - 1
        actions[pointer] = action / division
        pointer += 1

    return action_num, actions

action_num, actions = Action_discretization(asset_num, division)
```

File “config.py”.

```
import pandas as pd
abspath=""                  # the file path where you want to save the
parameter of the model
tuned_config = {"env": {"window_length":10,
                        "trading_period":2,
                        "norm": 'latest_close',
                        "trading_cost":0,    # Based on the assumptions
in the paper, commision fee is set as 0
                        "asset_num":6,       # 5 stocks and 1 cash
                        "feature_num":4},    # open price, close price,
highest price, lowest price
                "train": {"learning_rate":0.00025,
                          "division": 10,     # we divide the total
portfolio value into 10 equal parts
                          "epsilon": 0.01,
                          "epsilon_Min": 0.001,
                          "epsilon_decay_period":15000,
                          "start_date": pd.to_datetime(""),
                          "end_date":pd.to_datetime(""),
                          "date_range_s":pd.to_datetime(""),
                          "date_range_e":pd.to_datetime(""),
                          "steps_per_episode":160,
                          "reward_scale": 1,
                          "batch_size": 100,
                          "steps": 160,
                          "replay_period": 10,
                          "memory_size": 3000,
                          "upd_tar_prd": 200,
                          "save": True,
                          "save_period": 5000,  # save parameters every
5000 steps
                          "discount": 0.17},
                "net": {"kernels": [[1, 3], [1, 5], [1, 1]],
                        "strides": [[1, 2], [1, 5], [1, 1]],
                        "filters": [32,64,128],
                        "padding": "same",
                        "regularizer": 1e-4,
                        "b_initializer": 0.01,
                        "w_initializer": 0.01,
                        "cnn_activation": "relu",
                        "fc_activation": "relu",
                        "fc1_size": 512,
                        "output_num": 462},
                "mem":{"epsilonM" : 0.001,
                       "alpha":0.05,
                        "beta":0.01,
                        "beta_increment_rate":0.05,
```

```
                        "err_upp":0.05,
                        "capacity":3000}}
```

File “coordinator.py”.

```
import sys
import os
current_path = os.path.abspath(__file__)
sys.path.append("")
from portfolio import PortfolioEnv
import pandas as pd
from dqn_agent import Dqn_agent


df_train = pd.read_csv('', delimiter=';') #file path where the csv
dataset is stored

df_train['Date']=pd.to_datetime(df_train['Date'],
format='%d/%m/%Y').dt.strftime('%Y-%m-%d')

df_train['index_values'] =
df_train['Date'].rank(method='dense').astype(int)



class Coordinator:

    def __init__(self, config, name):

        name = name
        asset_num = config['env']['asset_num']
        feature_num = config['env']['feature_num']
        window_length = int(config['env']['window_length'])
        trading_cost = config['env']['trading_cost']
        trade_period = config['env']['trading_period']
        expan_coe = config['env']['expan_coe']

        network_config = config['net']

        self.total_training_step = config['train']['steps']
        self.replay_period = config['train']['replay_period']
        self.reward_scale = config['train']['reward_scale']
        learning_rate = config['train']['learning_rate']
        epsilon = config['train']['epsilon']
        epsilon_Min=config['train']['epsilon_Min']
        epsilon_decay_period = config['train']['epsilon_decay_period']
        start_date = config['train']['start_date']
        end_date = config['train']['end_date']
        date_range_s = config['train']['date_range_s']
        date_range_e = config['train']['date_range_e']
        steps_per_episode = config['train']['steps_per_episode']
        division = config['train']['division']
```

```
        gamma = config['train']['discount']
        batch_size = config['train']['batch_size']
        memory_size = config['train']['memory_size']
        upd_tar_prd = config['train']['upd_tar_prd']
        save = config['train']['save']
        save_period = config['train']['save_period']

        alpha = config['mem']['alpha']
        epsilonM = config['mem']['epsilonM']
        beta=config['mem']['beta']
        beta_increment_rate=config['mem']['beta_increment_rate']
        err_upp = config['mem']['err_upp']
        capacity=config['mem']['capacity']

        self.config = config

        self.agent = Dqn_agent(asset_num,
                               division,
                               feature_num,
                               gamma,
                               epsilon=epsilon,
                               epsilonM = epsilonM,
                               epsilon_Min=epsilon_Min,
                               alpha=alpha,
                               beta=beta,
                               beta_increment_rate=beta_increment_rate,
                               err_upp=err_upp,
                               capacity= capacity,
                               learning_rate=learning_rate,
                               network_topology=network_config,
                               update_tar_period=upd_tar_prd,
                               epsilon_decay_period=epsilon_decay_period,
                               memory_size=memory_size,
                               batch_size=batch_size,
                               history_length=window_length,
                               save=save,
                               save_period=save_period,
                               name=name)

        self.env_train = PortfolioEnv(df_train,
                                      start_date,
                                      end_date,
                                      date_range_s,
                                      date_range_e,
                                      steps=steps_per_episode,
                                      trading_cost=trading_cost,
                                      window_length=window_length,
                                      trade_period=trade_period,
                                      expan_coe=expan_coe)
```

```
    def train(self):

        training_step = 0
        self.rewards = []
        self.portfolio_values = []

        while training_step < self.total_training_step:
            observation = self.env_train.reset()

            while True:
                action_idx, action, fc_input =
self.agent.choose_action(observation)
                observation_, reward, done, info =
self.env_train._step(action)
                self.rewards.append(reward)
                portfolio_value = info.get("portfolio_value", None)
                if portfolio_value is not None:
                    self.portfolio_values.append(portfolio_value)


                reward *= self.reward_scale
                self.agent.store(observation, action_idx, reward,
observation_)
                observation = observation_
                if self.agent.start_replay():
                    if self.agent.memory_cnt() % self.replay_period == 0:
                        self.agent.replay()
                        training_step = self.agent.get_training_step()


                    break


    def restore(self, name):
        self.agent.restore(name)
```

File “dqn_agent.py”

```
import numpy as np
from memory import Memory
from action_discretization import Action_discretization
import tensorflow as tf
from config import abspath


tf.compat.v1.disable_eager_execution()

class Dqn_agent:
    def __init__(self, asset_num, division, feature_num, gamma,
                 network_topology,
                 learning_rate,
                 epsilon, epsilon_Min,
                 alpha, beta,
                 beta_increment_rate,
                 err_upp,
                 capacity,
                 epsilon_decay_period,
                 epsilonM,
                 update_tar_period,
                 history_length,
                 memory_size,
                 batch_size,
                 save_period,
                 name,
                 save):

        self.epsilon = epsilon
        self.epsilonM = epsilonM
        self.epsilon_min = epsilon_Min
        self.epsilon_decay_period = epsilon_decay_period
        self.asset_num = asset_num
        self.division = division
        self.gamma = gamma
        self.name = name
        self.update_tar_period = update_tar_period
        self.history_length = history_length
        self.feature_num = feature_num
        self.global_step = tf.Variable(0, trainable=False)
        self.lr = learning_rate
        self.cnn_trainable=False
        self.action_num, self.actions =
Action_discretization(self.asset_num, self.division)

        config = tf.compat.v1.ConfigProto()
        self.sess = tf.compat.v1.Session(config=config)
```

```
        network_topology['output_num'] = self.action_num
        self.network_config = network_topology

        self.initialize_graph()
        t_params =
tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.TRAINABLE_VARIABLES,
scope='target_net')
        e_params =
tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.TRAINABLE_VARIABLES,
scope='estm_net')

        self.update_target = [tf.compat.v1.assign(t, l) for t, l in
zip(t_params, e_params)]
        self.sess.run(tf.compat.v1.global_variables_initializer())

        self.memory = Memory(self.action_num, self.actions, memory_size,
capacity, batch_size, epsilonM,
                             alpha, beta, beta_increment_rate, err_upp)

        if save:
            self.save = save
            self.save_period = save_period
            self.name = name
            self.saver = tf.compat.v1.train.Saver()
        else:
            self.save = False

    def initialize_graph(self):
        self.price_his = tf.compat.v1.placeholder(dtype=tf.float32,
                                                  shape=[None,
self.asset_num - 1, self.history_length, self.feature_num],
                                                  name="ob")

        self.price_his_ = tf.compat.v1.placeholder(dtype=tf.float32,
                                                   shape=[None,
self.asset_num - 1, self.history_length, self.feature_num],
                                                   name="ob_")

        self.addi_inputs = tf.compat.v1.placeholder(dtype=tf.float32,
shape=[None, self.asset_num], name='addi_inputs')
        self.addi_inputs_ = tf.compat.v1.placeholder(dtype=tf.float32,
shape=[None, self.asset_num], name='addi_inputs_')

        self.a = tf.compat.v1.placeholder(dtype=tf.int32, shape=[None, ],
name='a')
        self.input_num = tf.compat.v1.placeholder(dtype=tf.int32,
shape=())
```

```
        self.ISWeights = tf.compat.v1.placeholder(tf.float32, [None, 1],
name='IS_weights')

        with tf.compat.v1.variable_scope('estm_net'):
            self.fc_input, self.q_pred = self.build_graph(self.price_his,
self.addi_inputs, self.cnn_trainable)

        with tf.compat.v1.variable_scope('target_net'):
            _, self.tar_pred = self.build_graph(self.price_his_,
self.addi_inputs_, self.cnn_trainable)

        with tf.compat.v1.variable_scope('q_tar'):
            self.q_target = tf.compat.v1.placeholder(dtype=tf.float32,
shape=[None], name='q_target')

        with tf.compat.v1.variable_scope('q_estm_wa'):
            a_indices = tf.stack([tf.range(tf.shape(self.a)[0],
dtype=tf.int32), self.a], axis=1)
            self.q_estm_wa = tf.gather_nd(params=self.q_pred,
indices=a_indices)

        with tf.compat.v1.name_scope('loss'):
            error = tf.abs(self.q_target - self.q_estm_wa)
            self.abs_errors = error
            square = tf.square(error)
            self.loss = tf.reduce_mean(self.ISWeights * square)

        with tf.compat.v1.name_scope('train'):
            self.optimizer = tf.compat.v1.train.AdamOptimizer(self.lr)
            self.train_op = self.optimizer.minimize(self.loss,
global_step=self.global_step)


    def build_graph(self, price_his, addi_input, trainable=False):
        kernels = self.network_config['kernels']
        strides = self.network_config['strides']
        filters = self.network_config['filters']
        fc1_size = self.network_config['fc1_size']

        # Define activation functions
        def set_activation(activation):
            if activation == 'relu':
                activation_fn = tf.nn.relu
            elif activation == 'relu':
                activation_fn = tf.nn.relu
            else:
                activation_fn = tf.nn.leaky_relu
            return activation_fn
```

```
        cnn_activation =
set_activation(self.network_config['cnn_activation'])
        w_initializer = tf.random_uniform_initializer(-0.05, 0.05)
        b_initializer =
tf.constant_initializer(self.network_config['b_initializer'])

        # Define regularizer
        regularizer =
tf.keras.regularizers.l2(self.network_config['regularizer'])

        # Define convolutional layers
        conv = price_his
        conv = tf.keras.layers.Conv2D(filters=filters[0],
kernel_size=kernels[0], strides=strides[0],
                                      trainable = trainable,
                                      activation=cnn_activation,
                                      kernel_regularizer=regularizer,
                                      bias_regularizer=regularizer,
                                      kernel_initializer=w_initializer,
                                      bias_initializer=b_initializer,
                                      padding='same', name=self.name +
'conv' + str(0))(conv)

        conv = tf.keras.layers.Conv2D(filters=filters[1],
kernel_size=kernels[1], strides=strides[1],
                                      trainable = trainable,
                                      activation=cnn_activation,
                                      kernel_regularizer=regularizer,
                                      bias_regularizer=regularizer,
                                      kernel_initializer=w_initializer,
                                      bias_initializer=b_initializer,
                                      padding='same', name=self.name +
'conv' + str(1))(conv)

        addi_input1 = addi_input[:, 1:]
        conv = tf.keras.layers.Concatenate(axis=3)([conv,
tf.expand_dims(tf.expand_dims(addi_input1, axis=-1), axis=-1)])

        conv = tf.keras.layers.Conv2D(filters=filters[2],
kernel_size=kernels[2], strides=strides[2],
                                      trainable = trainable,
                                      activation=cnn_activation,
                                      kernel_regularizer=regularizer,
                                      bias_regularizer=regularizer,
                                      kernel_initializer=w_initializer,
                                      bias_initializer=b_initializer,
                                      padding='same', name=self.name +
'conv' + str(2))(conv)
```

```
        cash_bias = tf.ones((self.input_num, 1))

        conv = tf.keras.layers.Flatten()(conv)
        fc_input = tf.keras.layers.Concatenate(axis=1)([cash_bias, conv])

        fc1 = tf.keras.layers.Dense(units=fc1_size, activation=None,
                                    kernel_initializer=w_initializer,
                                    kernel_regularizer=regularizer,
                                    bias_initializer=b_initializer,
                                    bias_regularizer=regularizer,
                                    trainable = trainable,
                                    name=self.name + 'fc1')(fc_input)

        output_state = tf.keras.layers.Dense(units=1, activation=None,
                                             kernel_initializer=w_initial
izer,
                                             kernel_regularizer=regulariz
er,
                                             bias_initializer=b_initializ
er,
                                             bias_regularizer=regularizer
,
                                             trainable = trainable,
                                             name=self.name +
'output_state')(fc1)

        output_action = tf.keras.layers.Dense(units=self.action_num,
activation=None,
                                              kernel_initializer=w_initia
lizer,
                                              kernel_regularizer=regulari
zer,
                                              bias_initializer=b_initiali
zer,
                                              bias_regularizer=regularize
r,
                                              trainable = trainable,
                                              name=self.name +
'output_action')(fc1)

        output = output_state + (output_action -
tf.reduce_mean(output_action, axis=1, keepdims=True))

        return fc_input, output

    def replay(self):

        observations, action_batch, reward_batch, observations_,
tree_idx, ISWeights = self.memory.sample()
```

```
        q_values_next = self.sess.run(self.q_pred,
feed_dict={self.price_his: observations_['history'],
                                                                self.addi_i
nputs:observations_['weights'],
                                                                self.input_
num:observations_['history'].shape[0]})

        best_actions = np.argmax(q_values_next, axis=1)

        q_values_next_target = self.sess.run(self.tar_pred,
feed_dict={self.price_his_: observations_['history'],
                                                                    se
lf.addi_inputs_:observations_['weights'],
                                                                    se
lf.input_num: observations_['history'].shape[0]})


        targets_batch = reward_batch + self.gamma *
q_values_next_target[np.arange(len(action_batch)), best_actions]


        fd = {self.q_target: targets_batch,
              self.price_his: observations['history'],
              self.addi_inputs: observations['weights'],
              self.a : action_batch,
              self.input_num: observations['history'].shape[0],
              self.ISWeights:ISWeights}

        _, abs_errors, global_step = self.sess.run([self.train_op,
self.abs_errors, self.global_step], feed_dict=fd)

        self.memory.batch_update(tree_idx, abs_errors)

        if global_step % self.update_tar_period == 0:
            self.sess.run(self.update_target)

        if self.save and global_step % self.save_period == 0:
            self.saver.save(self.sess, abspath+'/logs/checkpoint/' +
self.name, global_step=global_step)

        if self.epsilon > self.epsilon_min:
            self.epsilon -= (1 - self.epsilon_min) /
self.epsilon_decay_period



    def choose_action(self, observation, test=True):
```

```
        def action_max():
            fc_input,action_values =
self.sess.run([self.fc_input,self.q_pred],
                                                    feed_dict={self.price_h
is: observation['history'][np.newaxis, :, :, :],
                                                    self.addi_inputs:
observation['weights'][np.newaxis, :],
                                                    self.input_num : 1})
            return np.argmax(action_values), fc_input

        if not test:
            if np.random.rand() > self.epsilon:
                action_idx, fc_input= action_max()
            else:
                action_idx = np.random.randint(0, self.action_num)
                action_idx_, fc_input=action_max()
        else:
            action_idx,fc_input = action_max()

        action_weights = self.actions[action_idx]

        return action_idx, action_weights, fc_input


    def store(self, ob, a, r, ob_):
        self.memory.store(ob, a, r, ob_)

    def get_training_step(self):
        a = self.sess.run(self.global_step)
        return a

    def restore(self, name):
        self.saver.restore(self.sess, abspath+'/logs/checkpoint/'+name)

    def start_replay(self):
        return self.memory.start_replay()

    def memory_cnt(self):
        return self.memory.tree.data_pointer
```

File “main.py”.

```
import os
import sys
current_path = os.getcwd()
sys.path.append(current_path)
from coordinator import Coordinator
from config import tuned_config

model = Coordinator(tuned_config, 'name')
model.train()
model.restore('name-step')
```

File “memory.py”.

```
import numpy as np

#This class is the structure of memory pool
class SumTree:

    def __init__(self, capacity):
        self.capacity = capacity      #capacity is the number of memories
that the memory pool can contain
        self.tree = np.zeros(2 * capacity - 1)     #the number of nodes
of sumtree
        self.data = {}
        self.data_pointer=0

    #The method is to add new memory to the memory pool
    def add(self, p, data):
        tree_idx = self.data_pointer + self.capacity - 1
        self.data[self.data_pointer] = data
        self.update(tree_idx, p)
        self.data_pointer += 1

        if self.data_pointer >= self.capacity:
            self.data_pointer = 0

    #calculate TD-error of each node after new memory added to the memory
pool
    def update(self, tree_idx, p):
        change = p - self.tree[tree_idx]
        self.tree[tree_idx] = p

        while tree_idx != 0:
            tree_idx = (tree_idx - 1) // 2
            self.tree[tree_idx] += change

   #search the memory with the largest TD-error
    def get_leaf(self, v):

        parent_idx = 0

        while True:
            cl_idx = 2 * parent_idx + 1
            cr_idx = cl_idx + 1
            if cl_idx >= len(self.tree):
                leaf_idx = parent_idx
                break
            else:
                if v <= self.tree[cl_idx]:
                    parent_idx = cl_idx
```

```
                else:
                    v -= self.tree[cl_idx]
                    parent_idx = cr_idx
        data_idx = leaf_idx - self.capacity + 1

        return leaf_idx, self.tree[leaf_idx], self.data[data_idx]

    def total_p(self):
        return self.tree[0]


#memory pool
class Memory:

    def __init__(self, action_num, actions ,memory_size, capacity,
batch_size, epsilonM, alpha, beta, beta_increment_rate, err_upp):

        self.memory_size = memory_size
        self.batch_size = batch_size
        self.memory_pointer = 0
        self.action_num=action_num
        self.actions=actions
        self.rewards = []
        self.full = False
        self.tree = SumTree(capacity)
        self.epsilon = epsilonM
        self.alpha = alpha
        self.beta = beta
        self.beta_increment_rate = beta_increment_rate
        self.err_upp = err_upp

    #take a batch of samples form the memory pool
    def sample(self):

        observations = {'history':np.zeros((self.batch_size,
self.history_size[0], self.history_size[1], self.history_size[2])),
                        'weights':np.zeros((self.batch_size,
self.weight_size[0]))}
        #print("obs", observations)

        observations_ = {'history':np.zeros((self.batch_size,
self.history_size[0], self.history_size[1], self.history_size[2])),
                         'weights':np.zeros((self.batch_size,
self.weight_size[0]))}


        actions_idx = []

        rewards = []
```

```
        ISWeights = np.empty((self.batch_size, 1))     #initialize the
weight of each sample, to obtain the loss function

        b_idx=np.empty((self.batch_size,), dtype=np.int32)

        pri_seg = self.tree.total_p() / self.batch_size

        self.beta = 0.01

        self.beta = np.min([1., self.beta + self.beta_increment_rate])

        min_prob = np.min(self.tree.tree[-self.tree.capacity:]) /
self.tree.total_p()

        for i in range(self.batch_size):
            a = pri_seg * i
            b = pri_seg * (i + 1)

            v = np.random.uniform(a, b)
            idx, p, data = self.tree.get_leaf(v)
            prob = p / self.tree.total_p()
            ISWeights[i, 0] = np.power(prob / min_prob, -self.beta)
            b_idx[i] = idx
            observations['history'][i, :, :, :] = data[0]['history']
            observations['weights'][i, :] = data[0]['weights']
            observations_['history'][i, :, :, :] = data[3]['history']
            observations_['weights'][i, :] = data[3]['weights']
            actions_idx.append(data[1])
            rewards.append(data[2])

        return observations, actions_idx, rewards, observations_, b_idx,
ISWeights

    #add a memory to the memory pool
    def store(self, observation, action, reward, observation_):



        if len(self.tree.data)>=self.memory_size:
            self.full = True
            self.history_size = np.shape(self.tree.data[0][0]['history'])
            self.weight_size = np.shape(self.tree.data[0][0]['weights'])

        max_p = np.max(self.tree.tree[-self.tree.capacity:])

        if max_p == 0:
            max_p = self.err_upp
```

```
        transition = (observation, action, reward, observation_)

        self.tree.add(max_p, transition)

        self.rewards.append(reward)

    #update the node of sumtree
    def batch_update(self, tree_idx, abs_errors):

        abs_errors += self.epsilon

        clipped_errors = np.minimum(abs_errors, self.err_upp)

        ps = np.power(clipped_errors, self.alpha)

        for ti, p in zip(tree_idx, ps):
            self.tree.update(ti, p)

    #when the memory pool is full, start experience replay process
    def start_replay(self):
        return self.full
```

File “portfolio.py”.

```
import numpy as np
import pandas as pd

class DataSrc(object):

    def __init__(self, df,
                 start_date,
                 end_date,
                 date_range_s,
                 date_range_e,
                 steps,
                 expan_coe,
                 trade_period,
                 window_length):

        self.steps = steps
        self.trade_period = trade_period
        self.expan_coe = expan_coe
        self.window_length = window_length
        self.df = df
        self.df['Code'] = self.df['Code'].astype(str)
        self.df['Date'] = pd.to_datetime(self.df['Date'])
        self.df = self.df[(df['Date'] >= start_date) & (df['Date'] <=
end_date)]
        self.date_set = pd.date_range(date_range_s, date_range_e)
        self.asset_names = ['asset_1', 'asset_2', 'asset_3', 'asset_4',
'asset_5']
        self.features = ['Close', 'High', 'Low', 'Open']


        self.df.set_index('Date', inplace=True)
        asset_dict = dict()

        for asset in ['asset_1', 'asset_2', 'asset_3', 'asset_4',
'asset_5']:
            asset_data = self.df[self.df["Code"] ==
asset].reindex(self.date_set).sort_index()
            asset_data['Close'] =
asset_data['Close'].fillna(method='pad')
            asset_data['Open'] = asset_data['Open'].fillna(method='pad')
            asset_data['High'] = asset_data['High'].fillna(method='pad')
            asset_data['Low'] = asset_data['Low'].fillna(method='pad')
            asset_data = asset_data.fillna(method='bfill')
            asset_data = asset_data.fillna(method='ffill')
            asset_dict[str(asset)] = asset_data

        length = len(self.date_set) -1
```

```
        asset_data=asset_dict['asset_1']


        V_close = asset_data.loc[:asset_data.index[length], 'Close']

        V_high = asset_data.loc[:asset_data.index[length], 'High']

        V_low = asset_data.loc[:asset_data.index[length], 'Low']

        V_open = asset_data.loc[:asset_data.index[length], 'Open']



        state = []
        for asset in ['asset_2', 'asset_3', 'asset_4', 'asset_5']:
            asset_data = asset_dict[str(asset)]
            V_close = np.vstack((V_close,
asset_data.loc[start_date:end_date, 'Close']))
            V_high = np.vstack((V_high,
asset_data.loc[start_date:end_date, 'High']))
            V_low = np.vstack((V_low, asset_data.loc[start_date:end_date,
'Low']))
            V_open = np.vstack((V_open,
asset_data.loc[start_date:end_date, 'Open']))

        state.append(V_close)

        state.append(V_high)

        state.append(V_low)

        state.append(V_open)

        state_tensor = np.stack(state, axis=2)

        data = state_tensor


        if self.features[3] not in ['High', 'Low', 'Close', 'Open']:
            self.price_columns = self.features[:3]
        else:
            self.price_columns = self.features[:4]

        self.nb_pc = len(self.price_columns)
        self.close_pos = self.price_columns.index('Close')
        self.open_pos = self.price_columns.index('Open')
        self._data = data
```

```
    def _step(self):

        data_window = self.data[:,
(self.step)*self.trade_period:(self.step)*self.trade_period+
                                self.window_length,:self.nb_pc].copy()


        data_window_ = self.data[:,(self.step + 1) *
self.trade_period:(self.step + 1) * self.trade_period +
                                 self.window_length, :self.nb_pc].copy()

        y1 = data_window_[:, -self.trade_period, self.open_pos].copy() /
data_window[:, -self.trade_period,
                            self.open_pos].copy()

        y1 = np.concatenate([[1.0], y1])



        last_close_price = data_window[:, -1, self.close_pos].copy()

        data_window[:, :, :self.nb_pc] /= last_close_price[:, np.newaxis,
np.newaxis]

        history = data_window

        self.step += 1

        done = bool(self.step >= self.steps)

        return history, y1, done



    def reset(self):

        self.step = 0

        self.idx = np.random.randint(low=(self.steps+1)*self.trade_period
+ self.window_length, high=self._data.shape[1])

        self.data = self._data[:, self.idx -
(self.steps+1)*self.trade_period-
self.window_length:self.idx,:self.nb_pc].copy()
```

```python
# this class is trading process

class PortfolioSim(object):


    def __init__(self, asset_names, steps, trading_cost):
        self.cost = trading_cost
        self.steps = steps
        self.asset_names = asset_names
        self._p0_pre = 1.0
        self._p0 = 1.0
        self._w0 = np.array([1.0] + [0.0] * len(self.asset_names))


    def _step(self, w0, y1):

        _p0_pre = self._p0_pre

        _p0 = self._p0

        _w0 = self._w0

        c1 = self.cost * (np.abs(_w0[1:]-w0[1:])).sum()

        w0_ = (y1 * w0) / np.dot(y1, w0)

        p0_pre_=_p0_pre * np.dot(w0, y1)

        p0_ = _p0 * (1 - c1) * np.dot(w0, y1)

        r1 = np.log(p0_ / _p0)

        self._w0 = w0_

        self._p0 = p0_

        self._p0_pre = p0_pre_

        done = bool(p0_ <= 0)

        info = {"reward": r1,
                "portfolio_value": p0_,
                "weights": w0,
                "last_weight":_w0}
```

```
        return r1, info, done

    def reset(self):
        self.infos = []
        self._w0 = np.array([1.0] + [0.0] * len(self.asset_names))
        self._p0 = 1.0
        self._p0_pre = 1.0



class PortfolioEnv(object):
    def __init__(self,
                 df,
                 start_date,
                 end_date,
                 date_range_s,
                 date_range_e,
                 steps,
                 trading_cost,
                 trade_period,
                 expan_coe,
                 window_length):
        self.src = DataSrc(df=df,
                           start_date=start_date,
                           end_date=end_date,
                           date_range_s= date_range_s,
                           date_range_e= date_range_e,
                           steps=steps,
                           trade_period=trade_period,
                           window_length=window_length,
                           expan_coe=expan_coe)
        self.sim = PortfolioSim(asset_names=self.src.asset_names,
trading_cost=trading_cost, steps=steps)
```

```
    def _step(self, action):

        history, y1, done1 = self.src._step()

        reward, info, done2 = self.sim._step(action, y1)

        info['steps'] = self.src.step

        self.infos.append(info)

        return {'history': history, 'weights': info["weights"]}, reward,
done1 or done2, info


    def reset(self):
        self.sim.reset()

        self.src.reset()

        self.infos = []

        action = self.sim._w0

        observation ,reward, done, info =
self._step(action)

        return observation
```

BIBLIOGRAFIA

Bank of England (2022), *Machine learning in UK financial services*, ottobre 2022.

Barto, A., Sutton, R. (2016). *Reinforcement Learning: an Introduction II ed*. Londra: The MIT Press.

Barto, A., Sutton, R., Anderson, C. (1983). *Neronlike Adaptive Elements That Can Solve Difficult Learning Control Problems*, IEEE Transaction on Systems, man, and Cybernetics, Vol. SCM-13, No.5.

Bazarbash, M. (2019). *FinTech in Financial Inclusion: Machine Learning Applications in Assessing Credit Risk*, IMF Working Papers, *2019*(109), A001.

Brunton, S., Kutz, N. (2017). *Data Driven Science & Engineering*, Cambridge University Press.

Campbell, J., Shiller, R. (1988). *Stock Prices, Earnings, and Expected Dividends*, Journal of Finance. 43:3, pp. 661–76.

Campbell, J., Shiller, R. (1998). *Valuation Ratios and the Long-Run Stock Market Outlook*, Journal of Portfolio Management. Winter, 24, pp. 11–26.

Chong, E., Han, C., & Park, F. C. (2017). *Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies*, Expert Systems with Applications, 83, pagg. 187–205.

CIPA, ABI. (2021). *L'intelligenza Artificiale in banca: stato dell'arte e prospettive*, Rilevazioni sull'IT nel settore bancario italiano anno 2020.

Corazza, M. (2002). *La revisione statica del portafoglio azionario: i principali modelli classici*, Quaderni del dipartimento di matematica applicata, Venezia, Dipartimento di Matematica Applicata, Università Ca' Foscari Venezia, pagg. 1-20.

Corazza, M., Bertoluzzo, F. (2012). *Reinforcement Learning for automatic financial trading: Introduction and some applications*. Procedia Economics and Finance, vol. 3, pagg. 68-77.

Dash, R., Dash, P. (2016). *A hybrid stock trading framework integrating technical analysis with machine learning techniques*, The Journal of Finance and Data Science, Volume 2, Issue 1, pagg. 42-57.

Dawson, C.W., & Wilby, R.L. (1998*). An artificial neural network approach to rainfall-runoff modelling*, Hydrological Sciences Journal-journal Des Sciences Hydrologiques, 43, pagg. 47-66.

Deng, L., Yu, D. (2014). *Deep Learning: Methods and Applications*, Foundations and Trends in Signal Processing, 7, pagg. 197-387.

Fabozzi, F., Gupta, F., Markowitz, H. (2002). *The legacy of modern portfolio theory*, Journal of Investing, pagg. 7-22.

Fama, E. (1969). *Efficient Capital Markets: A Review of Theory and Empirical Work*, The Journal of Finance, vol. 25 num. 2, pagg. 383-417.

Fama, E., French, K. (1988). *Permanent and Temporary Components of Stock Prices*, Journal of Political Economy. 96:2, pp. 246–73.

Fama, E., Schwert, W. (1977). *Asset Returns and Inflation*, Journal of Financial Economics. November, 5:2, pp. 55–69.

Gao, Z., Gao, Y., Hu, Y., Jiang, Z., Su, J. (2020). Application of Deep Q-Network in Portfolio Management, Papers 2003.06365, arXiv.org.

Glorot, X., Bordes, A., Bengio, Y. (2011). *Deep sparse rectifier neural network*, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pagg. 315-323.

Hahnloser, R., Sarpeshkar, R., Mahowald, M. et al. (2000). *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit*, Nature 405, pagg. 947–951.

Huang, J., Chai, J. & Cho, S. (2020). *Deep learning in finance and banking: A literature review and classification*, Front. Bus. Res. China 14, 13.

Ingersoll, J. E. (1987). *Theory of financial decision making* (Vol. 3). Rowman & Littlefield.

Jiang, Z., Xu, D., & Liang, J. (2017). *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, ArXiv, abs/1706.10059.

Keim, D. Stambaugh, R. (1986). *Predicting Returns in Stock and Bond Markets*. Journal of Financial Economics. December, 17, pp. 357–90.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). *Deep learning*, Nature 521, pagg. 436–444.

Li, D., Yu, D. (2014). *Deep Learning: Methods and Applications*, Foundations and Trends® in Signal Processing: Vol. 7: No. 3–4, pagg. 197-387.

Lo, A. (2012). *Adaptive Markets and the New World Order*, Financial Analysts Journal, Vol. 68, No. 2, pagg. 18-29.

Lo, A., MacKinlay, C. (1999). *A Non-Random Walk Down Wall Street*, Princeton: Princeton University Press.

Maas, A.L., Hannun, A.Y. and Ng, A.Y. (2013). *Rectifier Nonlinearities Improve Neural Network Acoustic Models*, Proceedings of the 30th International Conference on Machine Learning, Vol. 28, 3.

Malkiel B. (2003). *The efficient Market hypothesis and Its Critics*, Journal of Economic Perspectives, Vol. 17, No. 1, Winter 2003, pagg. 59–82.

Mangram, M. (2013). *A Simplified Perspective Of The Markowitz Portfolio Theory*, Global journal of business research.

Matsubara, T., Akita, R., & Uehara, K. (2018). *Stock price prediction by deep neural generative model of news articles*. IEICE Transactions on Information and Systems, 4, pagg. 901–908.

Mittal, A., Goel, A. (2011). *Stock Prediction Using Twitter Sentiment Analysis*. Tratto da Stanford CS229: Machine learning.

Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2015). *Human-level control through deep reinforcement learning*, Nature 518, pagg. 529–533.

Nanigian, D.(2019). *The historical record on active vs. passive mutual fund performance*, Forthcoming in The Journal of Investing, 2019 Academic Research Colloquium for Financial Planning and Related Disciplines.

Odean, T. (1999). *Do Investors Trade Too Much?*, American Economic Review, December, 89:5, pp. 1279–298.

OECD (2021), *Artificial Intelligence, Machine Learning and Big Data in Finance: Opportunities, Challenges, and Implications for Policy Makers*.

OECD (2023), *Generative artificial intelligence in finance*, OECD Artificial Intelligence Papers, No. 9, Parigi, OECD Publishing.

Pujari, P., Majhi, B. (2018) *What is Functional Link Artificial Neural Network (FLANN)*, Handbook of research on modeling, analysis, and application of nature-inspired metaheuristic algorithms, ed. I, capitolo 19. IGI Global.

Samuel, A. (1959). *Some studies in Machine Learning Using the Game of Checkers*, IBM Journal of Research and Developement, Vol. 3, No. 3.

Shen, F., Chao, J., & Zhao, J. (2015). *Forecasting exchange rate using deep belief networks and conjugate gradient method*, Neurocomputing, 167, pagg. 243–253.

Shiller, R. J. (2003). *From Efficient Markets Theory to Behavioral Finance*. The Journal of Economic Perspectives, Vol. 17, No.1, Winter 2003, pagg. 83–104.

Shiller, R. (2000). *Irrational Exuberance*, Princeton: Princeton University Press.

Smith K.V. (1967). *A Transaction Model for Portfolio Revision*, Journal of Finance, 3, pagg. 425-439.

Stone B.K. e Hill N.C. (1979). *Portfolio Management and the Shrinking Knapsack Algorithm*, Journal of Financial and Quantitative Analysis, 14, pagg. 1071-1083.

The Guardian (2013). *Investments, Orlando is the cat's whiskers of stock picking,* articolo pubblicato su The Guardian, 13 gennaio 2013.

Turing, A. (1950). *Computing Machinery and Intelligence*, Mind, New Series, Vol. 59, No. 236, pagg. 433-460.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H. Lanctot, M., de Freitas, N. (2016). *Deuling Network Architectures for Deep Reinforcement Learning,* Papers arXiv:1511.06581v3.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*.

White, H. (1988). *Economic prediction using neural networks: the case of IBM daily stock returns*, Proceedings of the 2nd Annual IEEE Conference on Neural Networks, pagg. 451–459.

Xiong, Z., Liu, X., Zhong, S., Yang, H., & Elwalid, A. (2018). *Practical Deep Reinforcement Learning Approach for Stock Trading*, ArXiv, abs/1811.07522.

Zhao, B., Lu, H., Chen, S., Liu, J., Wu, D. (2017). *Convolutianl neural networks for time series classification*, Journal of Systems Engineering and Electronics, Vol. 28, No.1, pagg. 162-269.