# Intelligenza Artificiale

Reinforcement Learning per operazioni di trading

Cavaliere Mattia; Citro Carmine; Nunziata Vincenzo

December 2024

## Indice

# 1 Introduzione

Il presente progetto è mirato allo sviluppo di un algoritmo di Intelligenza Artificiale che permetta di compiere operazioni di trading su diversi assets finanziari, come ad esempio azioni stock o cryptovalute. Nello specifico l'agente intelligente è stato costruito utilizzando uno dei principali paradigmi del Machine Learning, il Reinforcement Learning.

## 1.1 Obiettivo del progetto

L'obiettivo del progetto è quello di utilizzare un algoritmo di Reinforcement Learning per massimizzare i profitti ottenuti tramite la compravendita di assets finanziari. Si vuole dimostrare l'efficacia del Reinforcement Learning come paradigma di Machine Learning nell'ambito di problemi in cui il feedback non è istantaneo ma potrebbe essere fornito successivamente nel tempo, e in problemi in cui il tempo è un fattore rilevante per la convergenza ad una soluzione ottima.

## 1.2 Contesto

Il contesto in cui è stato sviluppato il progetto è principalmente il mercato finanziario. Nello specifico, per lo sviluppo e il testing dell'agente intelligente si è posto il focus sul mercato azionario, tenendo in considerazione dati storici relativi a diverse azioni stock. Nonostante ciò, l'agente può essere utilizzato su diversi asset finanziari (si pensi alle cryptovalute).
I dati utilizzati sono dati storici dell'andamento di mercato delle azioni selezionate, i quali sono stati ottenuti tramite la libreria Python yfinance, la quale sfrutta le API fornite da Yahoo Finance per ottenere gli storici di mercato degli assets selezionati.

# 2 Background teorico

Esistono diversi tipi di classificazione degli algoritmi di Machine Learning, utilizzati per definire tipologie diverse di algoritmi in base alle loro caratteristiche. Una delle classificazioni più diffuse identifica i seguenti paradigmi:

- Apprendimento supervisionato (Supervised Learning)
- Apprendimento non supervisionato (Unsupervised Learning)
- Apprendimento per rinforzo (Reinforcement Learning)

Ognuno di questi paradigmi risulta efficiente per la risoluzione di specifiche tipologie di problemi. Ad esempio, l'apprendimento supervisionato risulta particolarmente utile quando ad ogni dato viene associata un'etichetta e i dati a disposizione per l'addestramento sono già etichettati; in questo modo l'agente intelligente può apprendere quali sono i pattern all'interno dei dati di training per predire il valore dell'etichetta di nuovi dati.

Nel caso del presente progetto, i dati utilizzati sono dati non statici, con correlazioni temporali e prettamente numerici, i quali non offrono la possibilità di definire delle etichette o dei gruppi di appartenenza (si pensi ad esempio al clustering che è un tipo di apprendimento non supervisionato). Pertanto, la soluzione più adeguata per il tipo di problema in questione è il Reinforcement Learning.

## 2.1 Reinforcement Learning

Il Reinforcement Learning (RL), o apprendimento per rinforzo, è un paradigma del **Machine Learning** in cui un agente intelligente impara a compiere decisioni ottimali attraverso l'interazione diretta con un ambiente dinamico. A differenza degli approcci supervisionati e non supervisionati, il Reinforcement Learning non richiede un dataset statico per l'addestramento, ma si basa su un ciclo iterativo di esplorazione e feedback.

Nel RL, il problema viene formalizzato come un **processo decisionale sequenziale**, spesso modellato mediante **Processi di Decisione di Markov (MDP)**. Gli elementi chiave sono:

- **Agente**: Il sistema che apprende e prende decisioni
- **Ambiente**: Il contesto esterno con cui l'agente interagisce
- **Stato ($s$)**: La rappresentazione delle condizioni attuali dell'ambiente
- **Azione ($a$)**: Le possibili scelte che l'agente può effettuare
- **Ricompensa ($r$)**: Un feedback numerico fornito dall'ambiente per valutare l'efficacia di un'azione
- **Politica ($\pi$)**: Una strategia che definisce la probabilità con cui l'agente sceglie le azioni in base allo stato corrente

L'obiettivo dell'agente è massimizzare la **ricompensa cumulativa** nel tempo, che può essere rappresentata mediante una funzione di valore scontata:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

dove $\gamma \in [0, 1]$ è un fattore di sconto che attribuisce maggiore importanza alle ricompense immediate rispetto a quelle future.

I principali approcci nell'RL si dividono in:

- **Basati su valori**: Algoritmi come il Q-Learning, che apprende una funzione $Q(s, a)$ per stimare la qualità di una determinata azione $a$ in uno stato $s$
- **Basati su politiche**: Algoritmi come Policy Gradient, che ottimizzano direttamente la politica $\pi(a|s)$ per migliorare le decisioni

- **Approcci ibridi**: Tecniche come Actor-Critic, che combinano i vantaggi dei metodi basati su valori e politiche.

Il processo iterativo di apprendimento per rinforzo può essere schematizzato come segue:

1. L'agente osserva lo stato corrente dell'ambiente ($s_t$)
2. Seleziona un'azione ($a_t$) basata sulla politica corrente
3. Riceve una ricompensa ($r_t$) e osserva il nuovo stato ($s_{t+1}$)
4. Aggiorna la politica o i valori stimati in base al feedback ricevuto

Grazie alla sua capacità di apprendere da interazioni e adattarsi a condizioni variabili, il Reinforcement Learning rappresenta una soluzione ideale per problemi caratterizzati da decisioni sequenziali e ambienti dinamici, come nel caso del progetto descritto in questa relazione tecnica.

## 2.2 Q-Learning

Il **Q-Learning** è un algoritmo di apprendimento per rinforzo off-policy che permette a un agente di apprendere una politica ottimale per massimizzare la ricompensa cumulativa estesa. La politica ottimale si basa sulla funzione $Q(s, a)$, che rappresenta il valore atteso della ricompensa che l'agente riceverà nel lungo termine eseguendo l'azione $a$ nello stato $s$.

L'algoritmo si basa sull'equazione di Bellman, che esprime una relazione ricorsiva tra i valori $Q$ di uno stato attuale e i valori $Q$ dello stato successivo:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

dove:

- $r$ è la ricompensa immediata ricevuta dopo aver eseguito l'azione $a$
- $s'$ è lo stato successivo
- $\gamma \in [0, 1]$ è il fattore di sconto, che determina l'importanza delle ricompense future
- $\max_{a'} Q(s', a')$ rappresenta il valore $Q$ massimo nello stato successivo $s'$, indicativo della politica ottimale

La funzione $Q(s, a)$ rappresenta quindi il "valore" di scegliere un'azione specifica $a$ in uno stato $s$, assumendo che in seguito venga adottata una politica ottimale.

Il Q-Learning aggiorna iterativamente la stima dei valori $Q(s, a)$ utilizzando la seguente regola di aggiornamento:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

dove:

- $\alpha \in [0, 1]$ è il tasso di apprendimento, che controlla quanto velocemente i valori $Q$ vengono aggiornati
- $r + \gamma \max_{a'} Q(s', a')$ è il **target**, ovvero il valore atteso aggiornato per la coppia stato-azione

Una componente essenziale del Q-Learning è la strategia di esplorazione $\epsilon$-greedy:

- con probabilità $\epsilon$, l'agente sceglie un'azione casuale (esplorazione)
- con probabilità $\epsilon - 1$, l'agente sceglie l'azione con il valore $Q$ più alto (exploitation)

Questa strategia garantisce che l'agente esplori nuove azioni nelle prime fasi di addestramento, riducendo gradualmente $\epsilon$ per convergere verso una politica ottimale.

Il Q-Learning converge a una politica ottimale, a patto che ogni coppia stato-azione venga visitata un numero sufficiente di volte e che il tasso di apprendimento $\alpha$ diminuisca adeguatamente.

Tuttavia, il Q-Learning diventa impraticabile in spazi di stati e azioni molto grandi, poiché richiede una tabella $Q$ di dimensioni $|S| \times |A|$. Inoltre, non è in grado di gestire direttamente spazi di stati o azioni continui senza una qualche forma di approssimazione della funzione $Q$.

## 2.3 DQN (Deep Q-Network)

Il **Deep Q-Network (DQN)** è un'estensione del Q-Learning standard, progettata per superare le limitazioni dell'algoritmo tradizionale quando applicato a spazi degli stati di grandi dimensioni o continui. Esso utilizza una rete neurale profonda per approssimare la funzione $Q(s, a; \theta)$, dove $\theta$ rappresenta i parametri della rete neurale.

Il DQN utilizza una rete neurale che:

- Prende come input una rappresentazione dello stato $s$ (ad esempio, un'immagine o un vettore di caratteristiche)
- Produce come output una stima dei valori $Q$ per tutte le possibili azioni $a$ nello stato $s$

La rete è addestrata per minimizzare l'errore quadratico tra il valore $Q$ stimato e il valore target calcolato come:

$$\text{Loss} = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a' : \theta^-) - Q(s, a; \theta))^2]$$

dove:

- $\theta$ sono i parametri della rete corrente

- $\theta^-$ sono i parametri della rete target, mantenuti fissi per un certo numero di passi di addestramento e aggiornati periodicamente

Le caratteristiche distintive del DQN sono:

- **Experience Replay Memory**: le transizioni $(s, a, r, s')$ sono memorizzate in un buffer e campionate casualmente durante l'addestramento. Questo approccio riduce la correlazione tra i dati di addestramento e aumenta la stabilità
- **Rete Target Separata**: una rete neurale secondaria (rete target) con parametri $\theta^-$ viene utilizzata per calcolare i valori target $max_{a'}Q(s', a'; \theta^-)$. I parametri della rete target vengono periodicamente sincronizzati con quelli della rete principale $\theta$, riducendo le oscillazioni e migliorando la convergenza
- **Strategia di Esplorazione $\epsilon$-greedy**: come nel Q-Learning standard, il DQN utilizza una strategia $\epsilon$-greedy per bilanciare esplorazione e sfruttamento. L'iperparametro $\epsilon$ viene ridotto gradualmente durante l'addestramento, passando da una fase di esplorazione a una di exploitation

## 2.4 Approccio

Come riportato, una delle principali limitazioni dell'algoritmo di Q-learning è l'impossibilità pratica di lavorare con spazi degli stati o delle azioni troppo grandi. L'ambiente, nel caso del presente progetto, fornisce uno spazio delle azioni discreto e uno spazio degli stati continuo pertanto sarebbe impossibile gestire una tabella $Q(s, a)$ con l'algoritmo di Q-Learning.

I possibili approcci sono due:

- Utilizzare algoritmi in grado di operare con spazi continui (come ad esempio il DQN)
- Applicare delle operazioni di **discretizzazione** dei dati per ridurre la dimensione della tabella $Q(s, a)$

Nel contesto di questo progetto si è scelto di implementare come agente principale un agente che sfrutta DQN come algoritmo di RL. In seguito si è applicata una tecnica di discretizzazione sui dati in modo da poter implementare un secondo agente basato su QL da utilizzare come modello di confronto.

### 2.4.1 Discretizzazione

Per costruire il modello basato su QL, è stato necessario discretizzare lo spazio degli stati per ridurre le dimensioni della tabella dei **q_values**. Esistono diverse tecniche di discretizzazione, che variano in base al tipo di dati in input e output. Inoltre, si possono adottare tecniche basate sulla distribuzione della frequenza dei dati o, in alternativa, tecniche più semplici di discretizzazione lineare.

Nel contesto di questo progetto, i dati che compongono lo spazio degli stati non hanno delle correlazioni di frequenza in quanto rappresentano un dato oggettivo, ovvero il costo di un'azione di un determinato asset e il profitto ottenuto dall'agente. Pertanto, si è scelto di utilizzare una tecnica di discretizzazione semplice e rapida da calcolare, ovvero il binning non supervisionato.

Il **binning non supervisionato** è una tecnica di discretizzazione che raggruppa i dati indipendentemente da qualsiasi relazione con una variabile target. Nello specifico è stato utilizzato un tipo di binning non supervisionato, ovvero l'**Equal Width Binning**, il quale, definito $k$ il numero di bin desiderato, genera $k$ bin della stessa dimensione. Ad ogni bin vengono poi assegnati i dati che rientrano nel suo range.

# 3 Formalizzazione del problema

Indipendentemente dall'approccio scelto, la formalizzazione del problema richiede di definire chiaramente i componenti del problema di Reinforcement Learning, formalizzandolo come un **Markov Decision Process (MDP)**.

Un MDP è definito da una tupla $(S, A, P, R, \gamma)$:

- $S$ (Stati): Insieme di tutti gli stati possibili in cui l'agente può trovarsi
- $A$ (Azioni): Insieme di tutte le azioni che l'agente può intraprendere
- $P(s'|s, a)$ (Transizioni): Probabilità di passare allo stato $s'$ eseguendo l'azione $a$ nello stato $s$. Definisce la dinamica dell'ambiente
- $R(s, a, s')$ (Ricompensa): Funzione che assegna una ricompensa al passaggio tra stati. Indica quanto "buona" è un'azione in un dato stato
- $\gamma$ (Fattore di sconto): Valore compreso tra 0 e 1 che determina quanto l'agente considera importanti le ricompense future rispetto a quelle immediate

## 3.1 Obiettivo

L'obiettivo dell'agente è massimizzare i profitti. Tuttavia, per perseguire questo obiettivo, l'agente deve bilanciare il numero di operazioni di trading (`compra` e `vendi`) con i passi in cui sceglie di non agire (azione `hold`). Inoltre, l'agente dovrà limitare le perdite significative e considerare il costo minimo associato a ciascuna operazione.

L'obiettivo finale è quindi massimizzare i profitti, bilanciando tutti i fattori rilevanti.

## 3.2 Spazio degli stati

Lo spazio degli stati $S$ è l'insieme:

$$S = \{s_1, s_2, \ldots, s_n\}$$

dove ogni stato $s \in S$ è una rappresentazione unica e completa del sistema. Un singolo stato $s$ è descritto da una serie di variabili osservabili $x_1, x_2, \ldots, x_k$ che caratterizzano l'ambiente in quel momento:

$$s = (x_1, x_2, \ldots, x_k)$$

dove:

- $x_i$: una variabile o feature rilevante per il problema
- La dimensionalità $k$: dipende dal numero di variabili necessarie per descrivere lo stato

Come anticipato, nel problema del trading, lo spazio degli stati è continuo e multidimensionale. Possiamo quindi descriverlo come:

$$S = \mathbb{R}^k$$

dove $k$ è il numero di caratteristiche rilevanti. Per ogni time step sarà associato uno stato $s$. L'ambiente utilizzato per il presente progetto si basa su dataset composti dai prezzi giornalieri delle azioni, pertanto il time step corrisponde a un giorno. Nell'ambito di questo progetto si è scelto di tenere in considerazione le seguenti informazioni:

- Prezzi di chiusura giornalieri
- Volume giornaliero
- Differenza di prezzo rispetto al time step precedente
- Profitto (o perdita) generato al time step precedente

Possiamo formalizzare lo spazio degli stati del presente progetto come segue:

$$S = \{s_1, s_1, \ldots, s_t, \ldots, s_k\}$$

$$s_t = (P_t, V_t, D_t, \Omega_t)$$

dove:

- $P_t$: array contenente i prezzi di chiusura giornalieri della finestra temporale corrente
- $V_t$: array contenente i volumi giornalieri della finestra temporale corrente
- $D_t$: array contenente la differenza di prezzo (delta) tra il time step $t$ e il time step $t-1$, sempre all'interno della finestra temporale

- $\Omega_t$: profitto (o perdita) ottenuto al time step $t$

Possiamo formalizzare i singoli componenti dello stato $s_t$ nel seguente modo:

$$P_t = [p_{t-29}, p_{t-28,\dots,p_t}] \in \mathbb{R}^{30}$$
$$V_t = [v_{t-29}, v_{t-28}, \dots, v_t \in \mathbb{R}^{30}]$$
$$D_t = [\Delta p_{t-29}, \Delta p_{t-28}, \dots, \Delta p_t] \in \mathbb{R}^{30} \text{ con: } \Delta p_i = p_i - p_{i-1}$$

Per $t-29$, $\Delta p_{t-29}$ può essere calcolato assumendo $p_{t-30}$ come il prezzo immediatamente precedente alla finestra corrente.

$$\Omega_t \in \mathbb{R}$$

Dove $\Omega_t$ rappresenta il profitto o la perdita accumulata al time step corrente $(t)$, calcolata in base alle azioni eseguite dall'agente.

Possiamo dire quindi che ogni stato $s_t \in \mathbb{R}^{91}$.

## 3.3 Spazio delle azioni

Lo spazio delle azioni, a differenza dello spazio degli stati, è uno spazio discreto. Pertanto, definito $k$ il numero delle azioni, lo spazio delle azioni sarà:

$$A = \{a_1, a_2, \dots, a_k\}$$

Nel caso specifico di questo progetto le azioni sono `Compra`, `Vendi` e `Mantieni`, pertanto si ha $k = 3$ e $A = \{\text{Vendi, Compra, Mantieni}\}$.

Ad ogni passo temporale $t$, l'agente seleziona un'azione $a_t \in A$ sulla base dello stato corrente $s_t$ e della politica appresa $\pi(s_t)$. La politica $\pi$ è definita come:

$$\pi(s_t) = \arg\max_a Q(s_t, a)$$

Dove $Q(s_t, a)$ è la funzione $Q$ appresa dal modello DQN.

## 3.4 Funzione di transizione

Le transizioni in un problema di apprendimento per rinforzo basato su un processo decisionale di Markov (MDP) sono formalizzate tramite la **funzione di transizione** $P(s'|s, a)$, che definisce la probabilità di raggiungere uno stato successivo $s'$ dallo stato corrente $s$ eseguendo l'azione $a$.

Nel contesto del DQN, le transizioni possono essere rappresentate come deterministiche o stocastiche, a seconda della dinamica dell'ambiente. Nel caso del presente progetto, le transizioni possono essere trattate come deterministiche poiché lo stato successivo $s'$ è una funzione diretta dello stato corrente $s$, dell'azione $a$, e della dinamica dell'ambiente. In tal caso:

$$s' = f(s, a)$$

Dove $f(s, a)$ è una funzione deterministica che modella:

- L'aggiornamento della finestra temporale (shift dei dati)
- Il valore del portafoglio basato sull'azione scelta
- Gli aggiornamenti relativi ai prezzi, volumi e differenze di prezzo

Considerando la forma dello stato $(P_t, V_t, D_t, \Omega_t)$, le transizioni aggiornano ciascuna componente come segue:

- **Prezzi di chiusura** $(P_t)$:

$$P_{t+1} = [p_{t-28}, p_{t-27}, \ldots, p_t, p_{t+1}]$$

  La finestra temporale viene shiftata in avanti, includendo il prezzo al time step $t+1$

- **Volumi** $(V_t)$:

$$V_{t+1} = [v_{t-28}, v_{t-27}, \ldots, v_t, v_{t+1}]$$

  Anche i volumi seguono lo stesso meccanismo di aggiornamento

- **Differenza di prezzo** $(D_t)$:

$$D_{t+1} = [p_{t-27} - p_{t-28}, \ldots, p_{t+1} - p_t]$$

- **Profitto/Perdita** $(\Omega_t)$:

$$\Omega_{t+1} = \Omega_t + \Delta P_{t+1}$$

  Dove $\Delta P_{t+1}$ è il cambiamento nel valore del portafoglio al time step $t+1$

Possiamo quindi generalizzare determinando la transizione dallo stato $s_t$ allo stato $s_{t+1}$ data l'azione $a_t$, come:

$$s_{t+1} = f(s_t, a_t)$$

Con:

$$f(s_t, a_t) = \begin{cases} P_{t+1}, & \text{prezzi shiftati in avanti} \\ V_{t+1}, & \text{prezzi shiftati in avanti} \\ D_{t+1}, & \text{differenze di prezzo aggiornate} \\ \Omega_{t+1}, & \text{profitto/perdita aggiornato} \end{cases}$$

## 3.5 Funzione di ricompensa

La ricompensa $R(s_t, a_t)$ associata all'azione $a_t$ eseguita nello stato $s_t$ è espressa dalla formula:

$$R(s_t, a_t) = \Delta P_t - \text{penalità}_h - \text{penalità}_{drawdown} - \text{penalità}_{transazione}$$

La ricompensa viene calcolata quindi come la variazione del valore del portafoglio tra il time step attuale e quello dell'ultimo trade eseguito (compra o vendi), sottratta dei tre valori di penalizzazione i quali variano in base al comportamento dell'agente.

Vediamo più nel dettaglio quali sono i singoli componenti della funzione di ricompensa.

Il delta del portafoglio ($\Delta P_t$) è definito come la differenza tra il valore del portafoglio al time step attuale $t$ e il valore del portafoglio all'ultimo time step $\tau$ in cui è stata eseguita un'azione `compra` o `vendi`:

$$\Delta P_t = V_t - V_\tau$$

Dove:

- $V_t = \text{cash}_t + (\text{azioni}_t \cdot \text{prezzo}_t)$: il valore del portafoglio al time step $t$
- $V_\tau = \text{cash}_\tau + (\text{azioni}_\tau \cdot \text{prezzo}_\tau)$: valore del portafoglio all'ultimo time step $\tau$ in cui è stata effettuata un'azione di trading

**Penalità**$_h$ tiene conto del numero totale di azioni e del tempo di inattività:

$$\text{penalità}_h = \lambda_h(h_{azioni} + h_{inattivo})$$

Dove:

- $\lambda_h$: coefficiente della penalizzazione $h$ sulla ricompensa
- $h_{azioni} = \beta_a \cdot n_{azioni}$, con $\beta_a$ coefficiente della penalizzazione $h_{azioni}$ e $n_{azioni}$ numero di azioni di trade (compra o vendi)
- $h_{inattivo} = \beta_i \cdot n_{hold}$, con $\beta_i$ coefficiente della penalizzazione $h_{inattivo}$ e $n_{hold}$ numero di time step consecutivi in cui l'azione scelta dall'agente è `hold`

La penalità di drawdown è una penalità applicata quando l'azione scelta dall'agente comporta una perdita. Nello specifico questa penalità viene applicata solamente se il valore del portafoglio $V_t$ scende di più del 50% del massimo valore raggiunto dal portafoglio fino al time step corrente ($V_{max}$). Possiamo, quindi, definire la penalità come:

$$\text{penalità}_{drawdown} = \lambda_d \begin{cases} \alpha \cdot \dfrac{V_{max} - V_t}{V_{max}}, & \text{se } \dfrac{V_{max} - V_t}{V_{max}} > 0.5 \\ 0, & \text{altrimenti} \end{cases}$$

Dove:

- $\lambda_d$: coefficiente di penalizzazione
- $V_{max} = \max_{t' \leq t} V_{t'}$: valore massimo del portafoglio fino al time step $t$

- $\alpha$: coefficiente di penalizzazione per drawdown significativo

La penalità di transazione è una penalità (piccola) applicata per ogni time step in cui viene eseguita un'azione ed è proporzionata al prezzo corrente dell'azione. Viene definita come:

$$\text{penalità}_{transazione} = \lambda_t \cdot 0.05 \cdot \text{prezzo}_t$$

Dove:

- $\lambda_t$: coefficiente di penalizzazione per il costo di transazione

Combinando tutte le componenti, la funzione di ricompensa risultate è quindi:

$$R(s_t, a_t) = (V_t - V_\tau) - \lambda_h(h_{azioni} + h_{inattivo)} - \lambda_d \begin{cases} \alpha \cdot \dfrac{V_{max} - V_t}{V_{max}}, & \text{se } \dfrac{V_{max} - V_t}{V_{max}} > 0.5 \\ 0, & \text{altrimenti} \end{cases} - \lambda_t \cdot 0.05 \cdot \text{prezzo}_t$$

I pesi dei diversi fattori di penalizzazione possono essere modificati andando a variare i valori $\lambda_h, \lambda_d$ e $\lambda_t$.

### 3.6 Fattore di sconto

Il fattore di sconto $\gamma$ è un valore scalare:

$$\gamma \in [0, 1]$$

dove:

- $\gamma = 0$: l'agente considera solo la ricompensa immediata e ignora tutte le ricompense future
- $\gamma \to 1$: l'agente attribuisce maggiore importanza alle ricompense future, considerandole quasi equivalenti a quelle immediate

Nel presente progetto, a seguito di vari test si è trovato il valore ottimale per il fattore di sconto ponendo:

$$\gamma = 0.95$$

## 4 Implementazione del DQN

Come anticipato in precedenza, l'ambiente prevede uno spazio degli stati continuo, con valori che appartengono all'intervallo $[-\infty, +\infty]$. Per tale motivo si è scelto di utilizzare un algoritmo di tipo Deep Q-Network.

## 4.1 Approccio

Per l'implementazione dell'algoritmo DQN è stato necessario definire l'architettura delle due reti neurali (policy network e target network), nonché della experience replay memory. Successivamente è stata svolta una fase di fine-tuning degli iperparametri dell'algoritmo.

## 4.2 Architettura del modello

Come definito al paragrafo 3.2, lo stato è composto da un tensore di dimensione 91x1 di valori reali. Di conseguenza, il layer di input è formato da 91 unità. Seguono poi due layer nascosti completamente connessi, ciascuno composto da 320 unità neurali. Come funzione di attivazione è stata scelta la Rectified Linear Unit (ReLU) per tutti i layer. Infine, l'output layer è composto da 3 unità, una per ciascuna azione ammessa, completamente connesse all'ultimo layer nascosto. Sono stati eseguiti esperimenti con diverse configurazioni di layer nascosti e numero di neuroni; tuttavia, questi valori sono risultati i più idonei per rappresentare la complessità dei dati in input.

Come funzione di loss è stata scelta la `SmoothL1Loss`, che ha il vantaggio di essere poco sensibile agli outlier. Anche in questo caso sono stati effettuati test con altre funzioni di loss, in particolare `MSELoss` e `HuberLoss`. In particolare, anche la `MSELoss` ha riportato buoni risultati in fase di addestramento.

Infine, la dimensione della experience replay memory è stata fissata a 100.000 tuple per garantire una sufficiente diversità nei campioni utilizzati durante l'addestramento.

## 4.3 Addestramento

Sono stati svolti diversi test per il tuning degli iperparametri, in particolare per quanto riguarda i valori di `epsilon_decay` ($\epsilon$), il fattore di sconto `gamma` ($\gamma$) e il `learning_rate` ($\alpha$).

Durante l'addestramento, la rete policy effettua uno step di apprendimento dopo ogni step nell'ambiente utilizzando un minibatch da 128 tuple campionate casualmente dalla experience replay memory. Inoltre, i parametri della rete target vengono sincronizzati con la rete principale ogni 5 episodi.

Per trovare i valori migliori per gli iperparametri, sono stati effettuati test con diverse combinazioni. Per ogni combinazione, l'agente è stato addestrato su 1000 episodi e poi eseguito su un ambiente di test. L'addestramento è stato eseguito su un ambiente composto da 11 asset finanziari e dati risalenti agli ultimi 4 anni. Per evitare bias nelle ricompense, sono stati scelti asset che mostrassero andamenti costanti o tendenzialmente in salita in tutto il periodo di riferimento. Non è stato possibile, invece, individuare asset che mostrassero andamenti completamente negativi. Gli asset scelti sono elencati nella tabella 1.

Per la valutazione dei risultati, sono stati presi in considerazione sia i risultati sull'ambiente di test sia le curve di addestramento e, in particolare, la ricompensa totale, in quanto questa riassume tutti gli aspetti oggetto di valutazione

| Ticker | Nome Completo |
|---|---|
| **TNDM** | Tandem Diabetes Care, Inc. |
| **JBHT** | J.B. Hunt Transport Services, Inc. |
| **TENB** | Tenable Holdings, Inc. |
| **GSHD** | Goosehead Insurance, Inc. |
| **BRKR** | Bruker Corporation |
| **SNDR** | Schneider National, Inc. |
| **SNAP** | Snap Inc. |
| **WBA** | Walgreens Boots Alliance, Inc. |
| **PII** | Polaris Inc. |
| **APA** | APA Corporation |
| **SWTX** | SpringWorks Therapeutics, Inc. |

Table 1: Ticker e nomi completi degli asset finanziari

dell'agente (profitto totale, perdita media, frequenza delle operazioni di trading ed errori commessi).

Ad eccezione dei tre iperparametri analizzati in dettaglio, tutti gli altri sono stati mantenuti costanti, compresa la dimensione del minibatch e della memoria, nonché il modello della rete, la funzione di loss e di attivazione.

**Learning Rate** Per il tasso di apprendimento $\alpha$, sono stati presi in considerazione i valori 0.01, 0.001, 0.005 e 0.0005. La figura 1 mostra le curve di apprendimento per $\epsilon = 0.95$ e $\gamma = 0.9$.

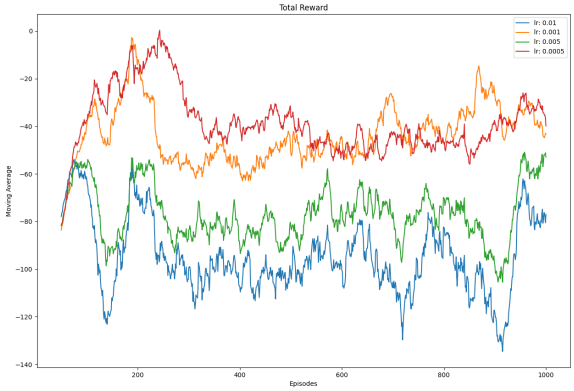
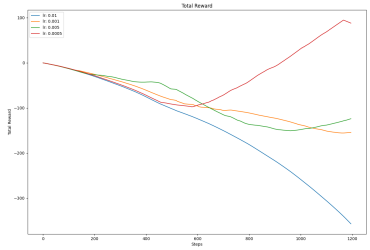


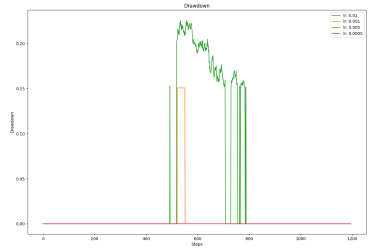

Figure 1: Curve di apprendimento per differenti valori di learning rate, con valori medi su una finestra mobile di 50 episodi.

Come mostrato in figura 1, i risultati migliori sono stati ottenuti con i valori $\alpha = 0.001$ e $\alpha = 0.0005$. I risultati sono quasi comparabili; tuttavia, analizzando l'esecuzione sull'ambiente di test (figura 2), si nota chiaramente che quest'ultimo valore ha ottenuto risultati nettamente migliori in tutti gli aspetti di valutazione.

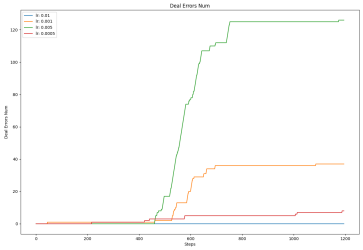



(a) Ricompensa totale


(b) ROI




(c) Drawdown




(d) Numero di azioni non corrette

Figure 2: Risultati sull'ambiente di test per vari valori di learning rate. In arancione $\alpha = 0.001$ e in rosso $\alpha = 0.0005$

.

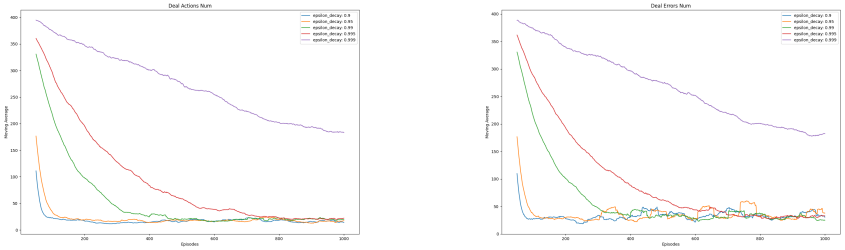
**$\epsilon$-decay** Per bilanciare esplorazione e sfruttamento, sono stati presi in considerazione cinque valori per $\epsilon$: 0.9, 0.95, 0.99, 0.995 e 0.999.

Dalle curve di apprendimento in figura 3, si nota che l'agente apprende rapidamente come massimizzare la ricompensa totale, anche con una ridotta esplorazione (con $\epsilon = 0.9$ ed $\epsilon = 0.95$, l'esplorazione termina in meno di 100 episodi). Un agente che opera in modo prevalentemente casuale tende infatti a eseguire molte azioni e a commettere numerosi errori, ottenendo ricompense fortemente negative. Terminata la fase di esplorazione, l'agente apprende rapidamente a ridurre sia il numero di azioni di trading eseguite sia il numero di errori commessi, come mostrato in figura 4.

Al termine dell'addestramento, l'agente riesce ad ottenere mediamente risultati migliori con una maggiore esplorazione. L'unica eccezione avviene per $\epsilon = 0.999$ in quanto dopo 1000 episodi la fase di esplorazione non è ancora


Figure 3: Curve di apprendimento per diversi valori di $\epsilon$, con valori medi su una finestra mobile di 50 episodi.

terminata.




(a) Numero di azioni (b) Numero di errori

Figure 4: Numero di azioni di trading eseguite ed errori commessi durante l'addestramento per diversi valori di $\epsilon$

.

**Fattore di sconto** Nel contesto del trading, dove è importante prevedere sia i trend a breve che a lungo termine, il fattore di sconto $\gamma$ riveste un ruolo chiave. Un fattore di sconto più basso, infatti, permette di apprendere tendenze generalmente locali e su un orizzonte temporale di pochi giorni, informazioni particolarmente utili al trader in posizione "corta"[1]. Al contrario, un fattore di sconto alto permette di apprendere le tendenze del mercato nel lungo periodo,

[1]In finanza, un operatore in posizione "corta" ha una visione ribassista del mercato, ovvero prevede che il valore dell'asset cali nel breve periodo.

fino a mesi o anni di distanza; queste informazioni sono ovviamente utili al trader in posizione "lunga"[2].

Nell'ambito del progetto, si è scelto di affrontare il problema considerando operazioni di compravendita classiche, ovvero in cui l'agente cerca di vendere gli asset in suo possesso al prezzo più alto possibile. Si è quindi scelto di addestrare un agente con l'obiettivo di prevedere i trend rialzisti del mercato nel lungo periodo. Per tale motivo, sono stati testati i seguenti valori per il fattore di sconto: 0.9, 0.99 e 0.999. La figura 5 mostra le curve di apprendimento considerando $\alpha = 0.0005$ e $\epsilon = 0.99$.


Figure 5: Cuve di apprendimento per diversi valori di fattore di sconto, con valori medi su una finestra mobile di 50 episodi.

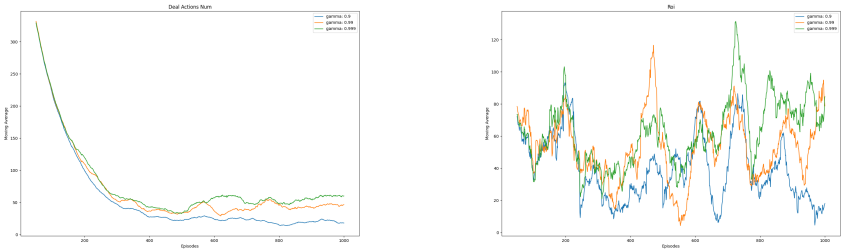
Analizzando esclusivamente le curve di apprendimento, si potrebbe concludere che l'agente ottenga i risultati migliori con il valore $\gamma = 0.9$. Approfondendo l'analisi, si scopre però che le elevate ricompense sono dovute principalmente a un numero minore di operazioni effettuate nell'arco dei quattro anni (figura 6a). Un agente più "miope" tende quindi ad avere un atteggiamento più prudente. Tuttavia, l'analisi del ritorno sugli investimenti (ROI) riportata in figura 6b evidenzia che la strategia adottata dagli agenti più "lungimiranti" consente di ottenere profitti nettamente più elevati.

Poiché l'obiettivo principale dell'agente dovrebbe essere quello di massimizzare i profitti, in questo caso si è scelto di sacrificare il numero di azioni eseguite, privilegiando un approccio più redditizio.

## 4.4 Risultati

Al termine della fase di tuning degli iperparametri, si è concluso che l'agente riesce a ottenere i risultati migliori con i valori $\alpha = 0.0005$, $\epsilon = 0.99$ e $\gamma = 0.999$.

[2] Al contrario, infatti, un operatore in posizione "lunga" ha una visione rialzista del mercato e prevede che il valore dell'asset aumenti nel lungo periodo.




(a) Numero di azioni (b) ROI

Figure 6: Risultati dell'addestramento in dettaglio per diversi valori di $\gamma$

.

Si è quindi proceduto a un test finale, addestrando un agente con gli iperparametri ottimizzati su 2.000 episodi e verificando le prestazioni finali su un ambiente di test. In figura 7 è riportato il risultato delle scelte dell'agente.


Figure 7: Esecuzione dell'agente DQN ottimizzato sull'ambiente di test - Asset di test: Netflix (NFLX)

Dalla figura si osserva un leggero fenomeno di overtrading. L'agente esegue complessivamente 11 operazioni di compravendita nell'arco di 1.200 giorni, con una durata media delle operazioni di compravendita di circa 46 giorni. Questi risultati erano attesi, in linea con la discussione riportata nel paragrafo 4.3. In generale, tuttavia, l'agente esegue operazioni intelligenti, acquistando asset nei momenti di ribasso e rivendendoli quasi sempre a un prezzo superiore, conseguendo, al termine, un ritorno sugli investimenti del 274%.

# 5 Implementazione del Q-Learning con discretizzazione

Nell'ultima fase di progetto si è deciso di addestrare un agente con algoritmo di Q-Learning per verificare se potesse essere applicato con successo al problema del trading e per compararne le prestazioni rispetto all'agente DQN.

## 5.1 Approccio

Come anticipato nel paragrafo 2.4, una delle limitazioni dell'algoritmo di Q-Learning è che può essere applicato solamente a problemi con uno spazio degli stati finito e discreto. Poiché, invece, l'ambiente prevede uno spazio degli stati continuo, i cui valori appartengono all'intervallo $[-\infty, +\infty]$, è stato necessario implementare una tecnica di discretizzazione degli stati.

Un'ulteriore limitazione dell'algoritmo di Q-Learning riguarda la dimensione dello spazio degli stati. Infatti, con uno stato composto da 91 elementi, lo spazio degli stati, dato dall'insieme delle possibili combinazioni di valori, risulta troppo esteso per essere esplorato completamente. Per questo motivo, si è reso necessario semplificare il problema, limitando le osservazioni esclusivamente ai prezzi di chiusura giornalieri nella finestra temporale definita dall'ambiente.

## 5.2 Tecnica di discretizzazione

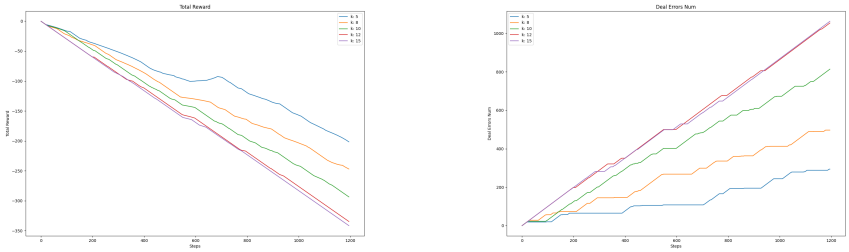
Si è adottata una discretizzazione dello stato basata su Equal Width Binning. A ogni episodio vengono generati $k$ bin di larghezza uniforme nell'intervallo $[min\_price, max\_price]$. Lo stato discretizzato è rappresentato dalla sequenza di indici dei bin in cui ricadono le osservazioni reali.

Particolare attenzione deve essere posta alla scelta del valore di $k$. Poiché le variazioni di prezzo tra giorni consecutivi possono essere minime, nell'ordine di poche unità, un valore di $k$ troppo basso potrebbe far ricadere entrambi i valori nello stesso bin, appiattendo così le tendenze di prezzo. Di conseguenza, si perderebbero informazioni fondamentali che l'agente dovrebbe apprendere autonomamente. Al contrario, un valore di $k$ troppo elevato riporterebbe al problema della dimensionalità dello spazio degli stati. Infatti, l'elevato numero di combinazioni di valori possibili renderebbe impossibile un'esplorazione completa dello spazio degli stati, compromettendo così la capacità di generalizzazione.

I test condotti su diversi valori di $k$ (5, 8, 10, 12, 15) e riportati in figura 8 mostrano chiaramente che il numero di errori commessi nell'ambiente di test – dovuti alla mancata esplorazione di alcuni stati durante l'apprendimento – aumenta con il crescere di $k$, influenzando negativamente la ricompensa totale ottenuta.

## 5.3 Addestramento

Per garantire risultati comparabili, l'addestramento dell'agente Q-Learning è stato realizzato sullo stesso ambiente utilizzato per l'addestramento dell'agente



(a) Ricompensa totale

(b) Numero di errori

Figure 8: Risultati ottenuti sull'ambiente di test per diversi valori del parametro $k$

.

DQN. Si rimanda quindi al paragrafo 4.3 per i dettagli.

Anche in questo caso è stato effettuato il tuning degli iperparametri $\alpha$ (learning_rate), $\epsilon$ (tasso di esplorazione) e $\gamma$ (fattore di sconto). Sono state testate diverse combinazioni di valori e i relativi risultati sono riportati nelle figure 9a, 9b e 9c.

Un aspetto interessante da evidenziare è che l'agente Q-Learning richiede un tasso di esplorazione maggiore rispetto all'agente DQN. Infatti, mentre quest'ultimo approssima la action-state value function `q` principalmente tramite sfruttamento, l'agente Q-Learning deve esplorare in modo più esteso lo spazio degli stati per ottenere un'approssimazione accurata.

### 5.4 Risultati

A seguito del tuning degli iperparametri, si è determinato che l'agente ottiene le migliori prestazioni con $\alpha = 0.0005$, $\epsilon = 0.995$ e $\gamma = 0.9$.

È stato quindi eseguito un test finale, addestrando l'agente con gli iperparametri ottimizzati su 2.000 episodi. In figura 10 è riportato il risultato delle scelte dell'agente sull'ambiente di test.

Durante i 1.200 giorni di simulazione, l'agente esegue complessivamente 12 operazioni di compravendita, con una durata media di circa 18 giorni. Sebbene alcune azioni non siano ottimali, l'agente consegue comunque un ritorno sugli investimenti pari al 71%. Tuttavia, la ricompensa finale risulta fortemente negativa, principalmente a causa dei numerosi errori commessi.

## 6 Confronto tra i due approcci

### 6.1 Analisi comparativa

La tabella 2 riassume i principali risultati ottenuti da entrambi gli agenti sull'ambiente di test.

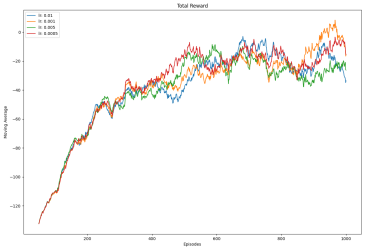
| | **DQN** | **Q-Learning** |
|---|---|---|
| Ricompensa totale | -183.125 | -237.818 |
| ROI | 274.53% | 71.40% |
| Numero di operazioni di compravendita | 11 | 12 |
| Durata media delle operazioni | 46 giorni | 18 giorni |
| Azioni non ammesse (errori) | 163 | 491 |

Table 2: Risultati ottenuti dagli agenti DQN e Q-Learning sull'ambiente di test

Complessivamente, l'agente DQN consegue un ritorno sugli investimenti significativamente superiore, pur eseguendo un numero di operazioni simile, ma con una durata media delle compravendite maggiore. La ricompensa totale ottenuta dall'agente Q-Learning risulta inferiore, principalmente a causa del minor profitto generato e, ancor più, dell'elevato numero di errori commessi.


(a) $\epsilon$


(b) $\gamma$




(c) $\alpha$

Figure 9: Curve di addestramento per diversi valori di $\epsilon$, $\gamma$ e $\alpha$ per l'agente Q-Learning

.

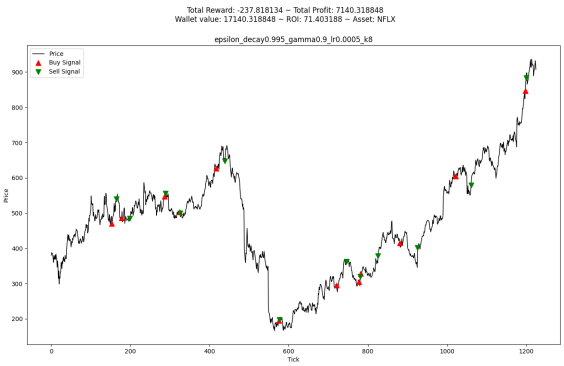



Figure 10: Esecuzione dell'agente Q-Learning ottimizzato sull'ambiente di test - Asset di test: Netflix (NFLX)

## 6.2 Scalabilità

Un confronto tra le curve di addestramento dei due modelli consente di trarre considerazioni significative. Dalla figura 11 si osserva che, durante l'addestramento, l'agente Q-Learning ottiene ricompense significativamente superiori rispetto all'agente DQN, in particolare nella seconda metà della fase di apprendimento. Il numero di errori commessi risulta inferiore, mentre il numero di azioni di trading eseguite è solo leggermente superiore. Il ritorno sugli investimenti è invece comparabile e, in alcuni casi, persino superiore.

L'analisi dettagliata dei risultati ottenuti dai due agenti nell'ambiente di test porta invece a conclusioni opposte. Infatti, dalla figura 12 si osserva che l'agente DQN riesce a ottenere ricompense maggiori rispetto all'agente Q-Learning. Tali ricompense, in particolare, derivano da un profitto significativamente più elevato e da un numero di errori commessi largamente inferiore.

Rispetto all'agente DQN, l'agente Q-Learning mostra una maggiore capacità di adattamento ai dati di addestramento. Durante la fase di sfruttamento, infatti, l'agente Q-Learning approssima con maggiore precisione la action-state value function per gli stati visitati, eseguendo azioni più efficaci.

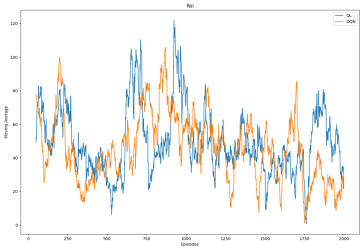
Tuttavia, questo comporta una riduzione della capacità di generalizzazione. Infatti, quando l'agente opera in un nuovo ambiente ed esplora stati non visitati durante l'addestramento, le sue prestazioni peggiorano sensibilmente.

Al contrario, l'agente DQN mostra una maggiore capacità di generalizzazione che, pur sacrificando le prestazioni nell'ambiente di addestramento, consente di ottenere risultati significativamente migliori anche in stati mai esplorati.

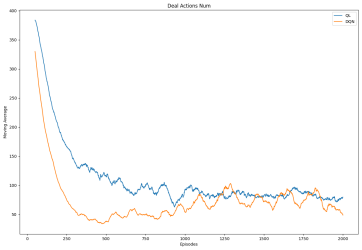
## 6.3 Impatto della discretizzazione

Come già analizzato precedentemente, le ridotte capacità di generalizzazione dell'agente Q-Learning sono imputabili alla discretizzazione dello spazio degli stati. Infatti, la discretizzazione da un lato comporta una perdita di informazioni

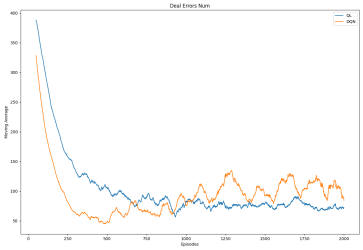

(a) Ricompensa totale




(b) ROI



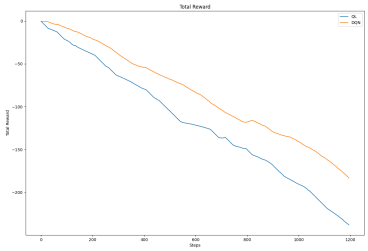

(c) Numero di azioni



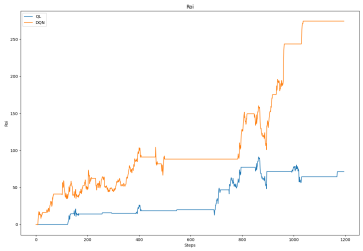

(d) Numero di azioni non corrette

Figure 11: Comparazione degli agenti DQN e Q-Learning sulle curve di addestramento

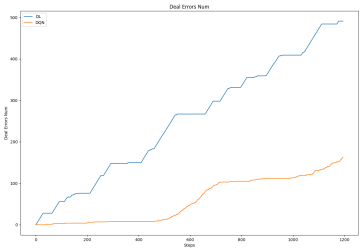
.




(a) Ricompensa totale




(b) ROI




(c) Numero di azioni

Figure 12: Comparazione degli agenti DQN e Q-Learning nell'ambiente di test

.

essenziali, come le piccole variazioni di prezzo giornaliere che delineano i trend, mentre dall'altro, per valori di $k$ più elevati, comporta un aumento della dimensionalità dello spazio degli stati e, di conseguenza, una maggiore probabilità che l'agente debba agire in stati mai esplorati durante l'addestramento.

# 7 Conclusioni e sviluppi futuri

## 7.1 Conclusioni principali

Nell'ambito del progetto in questione è stata valutata la possibilità di applicare tecniche di Reinforcement Learning a problemi di trading.

In particolare, sono stati dapprima definiti gli aspetti dell'ambiente in cui gli agenti si sarebbero trovati a operare, ponendo particolare attenzione alla definizione della funzione di ricompensa. Successivamente, sono stati implementati due agenti: uno basato su reti neurali con l'algoritmo Deep Q-Network e un secondo basato sull'algoritmo di Q-Learning classico con tecniche di discretizzazione degli stati. I due agenti sono stati ottimizzati attraverso una fase di tuning degli iperparametri e, infine, addestrati ed eseguiti su un ambiente di test.

Dai risultati ottenuti si evince che, sebbene l'agente di Q-Learning ottenga risultati appena sufficienti, entrambi gli approcci possono essere utilizzati per affrontare il problema in maniera adeguata.

## 7.2 Problemi affrontati

Il principale problema affrontato durante lo sviluppo del progetto è stato la definizione di una funzione di ricompensa adeguata agli obiettivi posti. Durante i primi esperimenti, infatti, è risultato che, se l'asset di addestramento era tendenzialmente in crescita, anche un agente che opera in maniera completamente casuale riusciva a ottenere profitti significativi, in quanto le piccole perdite ottenute in alcune operazioni di compravendita non incidevano molto sul profitto generale. Di conseguenza, utilizzando una funzione di ricompensa basata esclusivamente sul profitto complessivo, l'agente apprendeva che la strategia ottimale consistesse nell'eseguire un numero elevato di operazioni. Inoltre, con versioni successive della funzione di ricompensa sorgeva un ulteriore problema legato all'azione `Mantieni`: l'agente, infatti, apprendeva che, dopo aver generato un piccolo profitto con una compravendita, l'approccio migliore era non fare nulla per il resto dell'episodio. Sono state quindi sperimentate varie modifiche alla funzione di ricompensa, fino ad arrivare alla formulazione finale proposta in questo progetto.

Un ulteriore problema affrontato riguardava la scarsa variabilità dell'ambiente di addestramento. In un ambiente di Reinforcement Learning classico, infatti, la variabilità deriva dalle differenze nello stato iniziale dell'agente o nelle condizioni dell'ambiente tra episodi consecutivi. Al contrario, nella versione originale dell'ambiente utilizzato, l'agente veniva addestrato ripetutamente sullo

stesso trend per centinaia di episodi, determinando un eccessivo adattamento ai dati di addestramento. Per mitigare questo problema, l'ambiente è stato modificato affinché, ad ogni nuovo episodio, proponesse un asset selezionato casualmente da un pool stabilito.

Infine, un ultimo problema affrontato è stato la scelta dell'architettura della rete neurale. Nei primi esperimenti, infatti, non si riuscivano ad evincere grossi cambiamenti nelle curve di addestramento al variare dei valori degli iperparametri. Si è infine scoperto che il problema risiedeva nella rete neurale troppo piccola. Sono stati allora effettuati vari test per stabilire un'architettura che permettesse di rappresentare correttamente la complessità dei dati.

## 7.3 Possibili miglioramenti

Il trading finanziario è un problema altamente complesso, che coinvolge diversi aspetti, partendo da quello prettamente finanziario fino ad arrivare a fattori sociali e culturali. Per questo motivo, analizzare esclusivamente i dati storici potrebbe non rappresentare la soluzione ottimale al problema. Un possibile miglioramento consisterebbe nel fornire alla rete neurale, attraverso un modulo di comprensione del testo, informazioni aggiuntive quali notizie e articoli su fatti ed eventi che possono influenzare i trend futuri.

Inoltre, l'algoritmo DQN potrebbe rivelarsi ancora troppo semplice per gestire un problema di tale complessità. Un ulteriore miglioramento potrebbe consistere nell'esplorare l'utilizzo di algoritmi di Reinforcement Learning più avanzati, come ad esempio Proximal Policy Optimzation (PPO).