Sviluppo di un Algoritmo di Trading utilizzando il Q-learning per Massimizzare i Profitti

Giuseppe Genito, Rosaria Leone

December 19, 2023

1 Obiettivi

L'obiettivo del presente progetto è stata la creazione di un algoritmo di trading automatizzato. L'agente coinvolto nel progetto assume la responsabilità delle decisioni di acquisto e vendita di asset finanziari, facendo uso dell'algoritmo Q-learning. È stato pertanto necessario condurre l'addestramento dell'agente utilizzando dati storici relativi all'asset, seguito da una fase di test delle prestazioni utilizzando dati di verifica. Il fine ultimo di questo processo di addestramento è la massimizzazione delle performance finanziarie, valutate attraverso il Return on Investment (ROI).

2 Metodologia di implementazione

2.1 Raccolta dei dati storici di Mercato

L'acquisizione dei dati storici dell'asset finanziario S&P500 è avvenuto attraverso
l'uso della piattaforma Yhaoo finance. Il dataset fornito è composto dalle seguenti feature

- **Date:** Giorno in cui è stata rilevato l'andamento azionario
- **Open**: Prezzo di apertura dell'asset
- **High**: Prezzo più alto raggiunto dall'asset
- **Low**: Prezzo più basso raggiunto dall'asset
- **Close**: Prezzo di chiusura dell'asset
- **Adj Close**: Prezzo di chiusura aggiustato in base a dividenti o altro
- **Volume**: Volume in termini di denaro

Il dataset in questione vede la sua durata andare dal 1973-02-19 al 2023-12-15, avendo dunque una durata complessiva di 50 anni. Va sottolineato che le osservazioni sono settimanali e non giornaliere.

Date	Open	High	Low	Close	Adj Close	Volume
1973-02-19	0.000000	116.260002	112.769997	113.160004	113.160004	58920000
1973-02-26	0.000000	113.260002	109.449997	112.279999	112.279999	85860000
1973-03-05	0.000000	115.230003	111.330002	113.790001	113.790001	79910000
1973-03-12	0.000000	115.610001	112.839996	113.540001	113.540001	72060000
1973-03-19	0.000000	113.500000	107.410004	108.879997	108.879997	77390000
1973-03-26	0.000000	113.220001	108.290001	111.519997	111.519997	78120000
1973-04-02	0.000000	111.699997	107.440002	109.279999	109.279999	62080000
1973-04-09	0.000000	113.650002	108.739998	112.080002	112.080002	76150000
1973-04-16	0.000000	112.930000	109.989998	112.169998	112.169998	52630000
1973-04-23	0.000000	112.660004	106.760002	107.230003	107.230003	72310000

Figure 1: Dataset asset S&P500.

2.2 Progettazione dell'agente di Q-learning

L'agente che è stato implementato per svolgere il task di reinforcement learning sfrutta come algoritmo per prendere decisioni il Q-learning. Le azioni che l'agente può svolgere sono due, buy and sell, mentre l'azione hold avviene fin quando sceglie di fare la stessa azione, e dunque termina quando passerà da acquisto a vendita o viceversa. Il portafoglio che deve gestire l'agente è di 100.000 euro, e il budget per ogni investimento è il 2%, all'inizio dell'episodio coinciderà dunque con 2000 euro. Alla chiusura di un trade, che sia esso positivo o negativo, il prossimo budget dell'investimento verrà ricalcolato al 2% del portafoglio, in questo modo se l'agente perde soldi investirà di meno, se guadagna investirà di più. Su ogni trade che l'agente fa viene calcolata una commissione di ingresso ed una di uscita. La tassa di ingresso è pari al 0.1% all'ingresso, mentre lo 0.05% in uscita, entrambi calcolati sul budget investito.

Per lo svolgimento del progetto è stato necessario creare l'agente, e dunque sviluppare una classe, chiamata TradingAgent, composta dai seguenti metodi:

- **init**: Inizializza gli iperparametri
- **get action**: Restituisce la miglior azione secondo il q-value.
- **update**: Aggiorna il q-value di un'azione
- **decay_epsilon**: Diminuisce il valore di epsilon al termine di ogni episodio

2.2.1 __init__

Gli iperparametri inizializzati nell'__init__ sono i seguenti:

- **learning_rate**: Variabile di tipo float che serve a controllare il passo di aggiornamento nella fase di ottimizzazione del modello
- **starts_epsilon**: Variabile di tipo float che indica il valore iniziale del parametro epsilon che determina la percentuale di exploration e exploitation dell'agente.
- **epsilon_decay**: Calcola la velocità con la quale epsilon diventa più piccolo nei vari passi
- **final_epsilon**: Determina quando epsilon deve fermarsi
- **discount_factor**: E' una costante che viene moltiplicata per il calcolo del q-value

Di seguito il codice:

```
def __init__(
    self, learning_rate: float, initial_epsilon: float, epsilon_decay: float,
    final_epsilon: float, discount_factor: float,):

    ## Crea un dizionario di grandezza pari alla lunghezza dell'env
    self.q_values = defaultdict(lambda: np.zeros(env_1.action_space.n))

    ## Inizializzazione iper-parametri
    self.lr = learning_rate
    self.discount_factor = discount_factor
    self.epsilon = initial_epsilon
    self.epsilon_decay = epsilon_decay
    self.final_epsilon = final_epsilon

    ## Inizializza una lista vuota per conservare il TD error
    self.training_error = []
```

Figure 2: Codice __init__.

2.2.2 get_action

Il metodo get_action ha lo scopo di restituire l'azione che deve compiere l'agente, tale azione però potrebbe non essere quella con il q-value più alto, bensì con probabilità 1-epsilon l'agente potrebbe scegliere di compiere un'altra azione che potrebbe portare a risultati migliori in futuro.
Di seguito il codice:

```
def get_action(self, obs: list[float, float]) -> int:

    """
    Restituisce la migliore azione con probabilità (1 - epsilon),
    altrimenti con probabilità epsilon restituisce una azione random
    per esplorare l'ambiente
    """

    if np.random.random() < self.epsilon:
        return env_1.action_space.sample()

    # con probabilità (1 - epsilon) azione greedy (exploit)
    else:
        return int(np.argmax(self.q_values[obs]))
```

Figure 3: Codice get_action.

2.2.3 update

Con questo metodo viene calcolato il q-value dell'azione. Affinché possa essere calcolato è necessario prima calcolare la temporal difference, la quale fungerà anche da parametro per valutare l'andamento dell'errore negli episodi durante la fase di training. Ottenuta la temporal difference viene calcolato il Q-value con la seguente formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\big)$$

Figure 4: Formula calcolo q-value

La formula è composta da due pezzi, il valore del q-value e il learning rate che viene moltiplicato per la temporal difference.
Di seguito il codice che mostra come è stato calcolato il q-value nel progetto:

```python
def update(
    self,
    obs: list[float, float], action: int, reward: float,
    terminated: bool, next_obs: list[float, float],):

    """
    Aggiorna il Q-value per una azione.
    Viene calcolato il temporal difference, poi il q-value, e il temporal
    difference viene salvato nella lista per vedere l'andamento
    dell'errore negli episodi.
    """

    future_q_value = (not terminated) * np.max(self.q_values[next_obs])
    temporal_difference = (
        reward + self.discount_factor * future_q_value - self.q_values[obs][action]
    )

    self.q_values[obs][action] = (
        self.q_values[obs][action] + self.lr * temporal_difference
    )
    self.training_error.append(temporal_difference)
```

Figure 5: Codice update per il calcolod del Q-value

2.2.4 decay_epsilon

Questo metodo viene chiamato alla fine di ogni episodio per diminuire l'epsilon, ossia la variabile che determina la probabilità di esplorazione. La sua funzione è quella di andare a ridurre epsilon secondo qualche funzione definita all'esterno come iper-parametro
In particolare la funzione in questione è calcolata come

$$\epsilon_{\text{decay}} = \frac{\text{start\_epsilon}}{\frac{\text{n\_episodes}}{2}} \tag{1}$$

Di seguito il codice del metodo decay_epsilon:

```python
def decay_epsilon(self):

  """
  Al termine di un episodio viene chiamato questo metodo per calcolare
  la diminuizione dell'epsilon.
  """

  self.epsilon = max(self.final_epsilon, self.epsilon - epsilon_decay)
```

Figure 6: Metodo epsilon decay

2.3 Funzione di ricompensa

Una funzione ricompensa dell'agente è definita nell'ambiente, (il cui codice è proveniente dalla repository GitHub: https://github.com/AminHP/gym-anytrading/tree/master). La funzione di calcolo è presente nel metodo calculate reward della classe StockEnv. Tale funzione consiste nel dare un premio all'agente solo nell'occasione in cui passa dall'acquisto di un azione alla vendita, dunque questo implica che se l'agente per n istanti temporali decide di effettuare come azione l'acquisto e poi vende allora viene calcolata la ricompensa, altrimenti se passa dall'azione di vendere a quella di comprare, a prescindere da quello che sia il profitto/perdita non sarà calcolata alcuna ricompensa. Il valore della ricompensa è calcolato come la differenza tra il prezzo di acquisto e quello di vendita. E' stato necessario riscrivere la funzione di ricompensa, questa scelta è avvenuta a seguito di una fase di test in cui è stata riscontrata la discriminazione dell'agente verso l'azione vendere piuttosto che acquistare,

in quanto gli veniva garantito di non ricevere alcuna penitenza. La funzione di ricompensa è stata modificata come segue. Se l'agente sta vendendo, e la differenza di prezzo tra il prezzo attuale e quello precedente va in negativo egli ottiene una ricompensa pari alla differenza di prezzo cambiata di segno, (Da segno negativo a positivo), quando invece la differenza cresce riceve come penalità la differenza di prezzo cambiata di segno, (Da segno positivo a negativo). Lo stesso processo avviene per il caso opposto quindi quando passa dall'acquisto alla vendita. Inoltre è stato implementata la funzionalità stop loss, in questo modo quando l'agente sta perdendo più del 2% del budget viene chiusa la transazione, e prende una ricompensa negativa.
La discrepanza tra la media degli episodi e la lunghezza del dataset è di due valori, che corrispondono al primo ed ultimo stato che nella lunghezza dell'episodio non viene conteggiato.

2.4 Addestramento dell'agente

L'addestramento dell'agente consiste nel minimizzare l'errore del temporal difference in modo da migliorare la scelta dell'azione migliore. La parte difficile di questa fase è trovare i migliori iper-parametri in modo da ottimizzare le performance. In particolare gli iper-parametri a disposizione sono i seguenti: In una prima fase vengono settati tutti gli iper-parametri necessari:

- **discount_factor** Varia tra 0 e 1. Indica quanto peso delle ricompense future. Un basso discount factor farà prediligere all'agente azioni che danno una ricompensa immediata, un alto discount factor farà prediligere azioni che daranno ricompense future maggiori anche a discapito di ricompense negative in una fase iniziale.
- **learning_rate**: E' la grandezza del passo di aggiornamento durante la fase di ricerca dell'ottimo. Un learning rate troppo basso potrebbe non far trovare mai l'ottimo, al contrario uno alto potrebbe far saltare il punto di ottimo.
- **start_epsilon**: Valore epsilon iniziale, determina il fattore di esplorazione dell'agente. Diminuisce del tempo di un fattore pari a epsilon decay
- **epsilon_decay**: Determina quanto velocemente deve decrescere epsilon alla fine di ogni episodio.
- **final_epsilon**: Determina la probabilità minima di esplorazione dell'agente.
- **numero di episodi**: Il numero di episodi è probabilmente uno dei parametri principali del modello. Un numero di episodi troppo basso non permette al modello di apprendere bene, non dando il tempo necessario all'errore di diminuire. D'altra parte un numero di episodi troppo alto da ottime prestazioni nella fase di training, ma pessime nella fase di test a causa dell'overfittig. Rendendo oneroso anche il tempo di addestramento. Per un numero di episodi pari a 10.000 il tempo impiegato da Colab per l'addestramento è circa mezz'ora.

2.5 Valutazione delle prestazioni

In conclusione è stato necessario testare l'agente su dati a lui sconosciuti, in particolare il dataset iniziale, composto dalle transazioni avvenute negli ultimi 5 anni, è stato diviso in due porzioni, una composta dall'80% ed una dal 20% delle osservazioni, i dati inoltre sono presi senza ripetizione dunque sono unici, e non sono stati mescolati casualmente come avviene nei processi di machine learning, perché altrimenti si perderebbe l'andamento del mercato. In questa fase è stato necessario modificare gli iperparametri durante il training per trovare la migliore configurazione che desse il training error minore e che al contempo nel test set desse il ROI più alto e le ricompense più alte. Dunque in questa fase è stato necessario ottimizzare i valori del q-value, come learning rate, epsilon e numero di episodi al fine di massimizzare i profitti cumulativi.

3 Risultati

3.1 Dimostrazione delle prestazioni ottenute

L'addestramento dell'algoritmo è stato sottoposto a iterazioni multiple, mirando a perfezionare gradualmente gli iperparametri al fine di ottenere le migliori prestazioni possibili. Questa fase di raffinamento, comunemente nota come fine-tuning, è stata cruciale per ottimizzare la capacità predittiva e la robustezza del nostro modello di trading. Durante ciascuna iterazione di addestramento, abbiamo regolato attentamente gli iperparametri chiave, sperimentando varie combinazioni al fine di individuare quelle ottimali. Le modifiche agli iperparametri includevano regolazioni del learning rate, del discount_factor, del final_epsilon e del numero di episodi.

```
## Iper-parametri del TrainingAgent

n_episodes = 5500
discount_factor = 0.98
learning_rate = 0.1
start_epsilon = 1
epsilon_decay = start_epsilon / (n_episodes / 2)
final_epsilon = 0.01
```

Figure 7: Iperparametri

- **Discount Factor**

  Il discount factor, indicato come γ (gamma), nell'algoritmo di training è un parametro cruciale che influenza il peso attribuito alle ricompense future rispetto a quelle immediate. γ è un valore compreso tra 0 e 1, dove:

  - Un γ vicino a 1 indica un'elevata considerazione delle ricompense future, riflettendo una strategia orientata al lungo termine.
  - Un γ vicino a 0 indica una minore considerazione delle ricompense future, enfatizzando maggiormente i risultati immediati.

- **Learning Rate**

  Il learning rate, indicato come α (alfa), è un parametro cruciale nell'algoritmo di training che determina la dimensione dei passi di aggiornamento dei pesi durante il processo di ottimizzazione. Questo valore positivo può variare in un intervallo tipicamente compreso tra 0 e 1.

  - $\alpha = 1$ indica che i pesi vengono aggiornati con la massima ampiezza ad ogni iterazione, favorendo un apprendimento rapido. Tuttavia, un α troppo elevato può causare problemi come divergenza o oscillazioni indesiderate.
  - $\alpha = 0$ implica che i pesi vengono aggiornati con l'ampiezza minima ad ogni iterazione, portando ad un apprendimento lento o al rischio di convergenza in minimi locali.

- **Start Epsilon**

  ϵ_{start} è il valore iniziale del parametro di esplorazione, che determina la probabilità di eseguire un'azione casuale anziché quella predetta dal modello. Inizialmente, durante le prime fasi del training, si permette una maggiore esplorazione per scoprire nuove strategie.

  - $\epsilon_{\text{start}} = 1$ indica un'iniziale massima propensione all'esplorazione, favorendo l'acquisizione di informazioni sulle possibilità dell'ambiente.
  - $\epsilon_{\text{start}} = 0$ suggerisce una bassa probabilità di esecuzione di azioni casuali, privilegiando invece l'uso delle previsioni del modello.

- **Epsilon Decay**

  L'epsilon decay si riferisce alla riduzione graduale del parametro di esplorazione ϵ nel corso delle iterazioni dell'algoritmo di training. Questo parametro influisce sulla probabilità di eseguire un'azione casuale anziché l'azione predetta dal modello.

- **Final Epsilon**

  Final epsilon, indicato come ϵ_{final}, rappresenta il valore finale che il parametro di esplorazione ϵ assume dopo il processo di epsilon decay. Nello specifico indica la probabilità residua di eseguire un'azione casuale anziché l'azione predetta dal modello.

  - $\epsilon_{\text{final}} = 1$ indica che, nonostante il decay, la probabilità di eseguire azioni casuali rimane massima alla fine del training.
  - $\epsilon_{\text{final}} = 0$ suggerisce che la probabilità di eseguire azioni casuali è ridotta al minimo, privilegiando maggiormente l'utilizzo delle previsioni del modello.

3.2 Addestramento e ROI del train

Il seguente grafico è il risultatod ell'addestramento sui dati di train e in più viene mostrato anche il ROI (Return on Investment), il quale esprime il rendimento di un investimento in percentuale rispetto al suo costo iniziale.

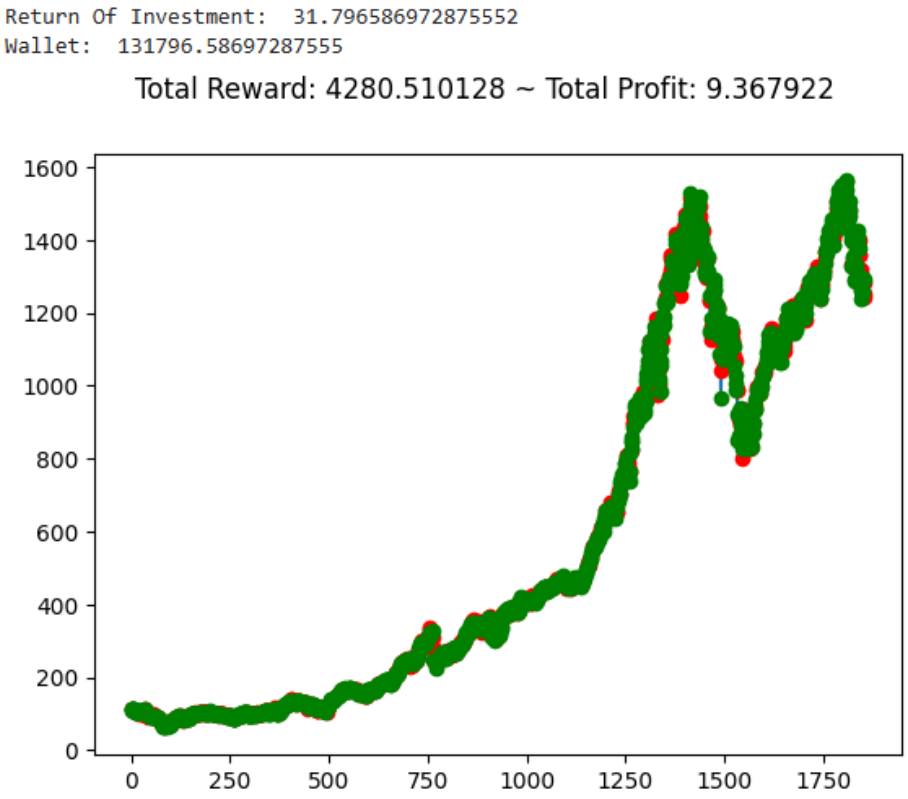

Figure 8: ROI sul Train del 31.8%

Un ROI del 31.8% indica che l'investimento ha generato un guadagno del 31.8% rispetto al suo costo iniziale durante la fase di addestramento.
La formula del ROI è data da:

$$\text{ROI} = \left(\frac{\text{Guadagno Netto}}{\text{Costo Iniziale}} \right) \times 100$$

Dove:

- Il Guadagno Netto è la differenza tra il valore attuale dell'investimento e il suo costo iniziale.
- Il Costo Iniziale rappresenta l'importo iniziale investito.

Nel nostro caso la formula è stata:

$$\text{ROI} = \left(\frac{31.8}{100}\right) \times 100 = 31.8\%$$

Questo risultato indica un rendimento molto positivo dell'investimento.

3.3 Andamento dell'errore del train

L'andamento dell'errore durante la fase di addestramento del modello mostra un pattern interessante: inizialmente, l'errore presenta una fase di aumento, seguita da una fase di decrescita fino a stabilizzarsi. Questo andamento può essere interpretato in diversi modi.

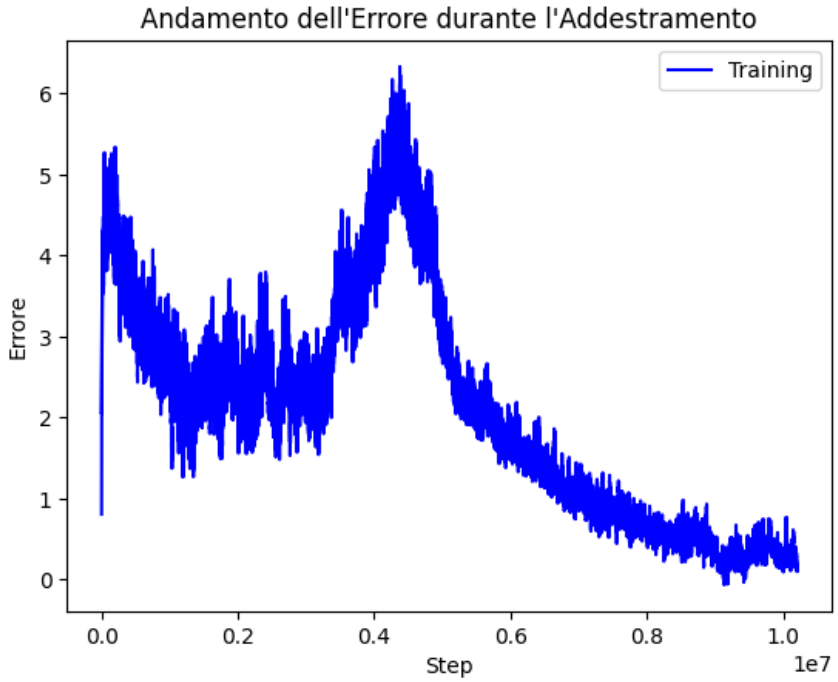

Figure 9: Errore sul Train

L'incremento iniziale potrebbe indicare che il modello ha inizialmente difficoltà a comprendere i pattern nei dati di addestramento. La fase successiva di decrescita suggerisce che il modello sta iniziando ad adattarsi meglio ai dati di addestramento, potendo catturare in modo più efficace i pattern rilevanti. La stabilizzazione dell'errore indica che il processo di ottimizzazione dei pesi del modello ha raggiunto un punto in cui ulteriori modifiche ai pesi non portano a miglioramenti significativi. In questa fase il modello potrebbe aver raggiunto una configurazione dei pesi che ben rappresenta i dati di addestramento.

3.4 Andamento delle ricompense del train

Durante la fase di addestramento inoltre abbiamo valutato anche l'andamento delle ricompense. Le "ricompense" rappresentano gli output ottenuti dall'algoritmo in risposta alle azioni intraprese durante il processo di apprendimento.

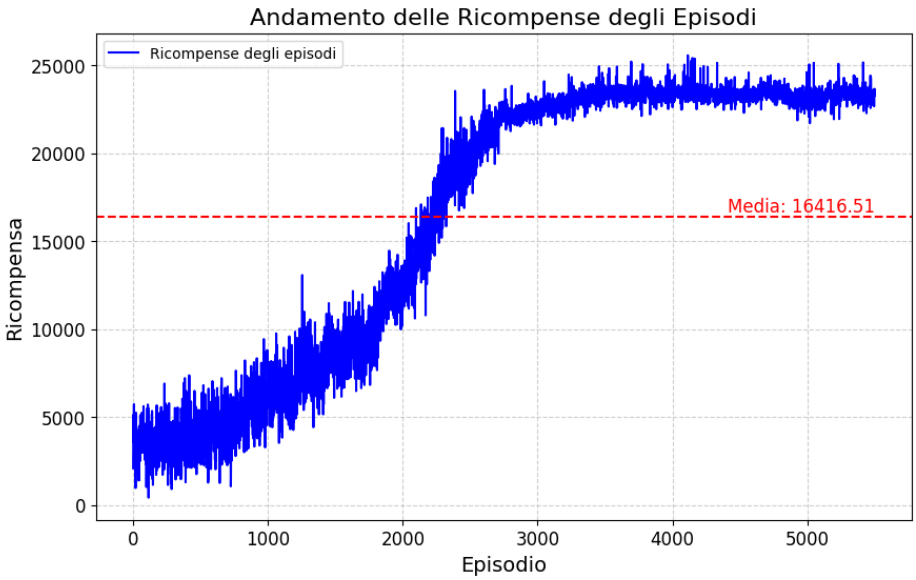

Figure 10: Ricompense sul Train

La somma cumulativa di 16,416.51 indica il rendimento complessivo dell'algoritmo in termini di ricompense. Tale valore suggerisce un rendimento complessivo positivo. Questo numero rappresenta una metrica dell'efficacia dell'algoritmo nel massimizzare le ricompense durante il processo di addestramento.

3.5 Addestramento e ROI del Test

Ed infine il grafico dell'addestramento sui dati di test e ROI del test:

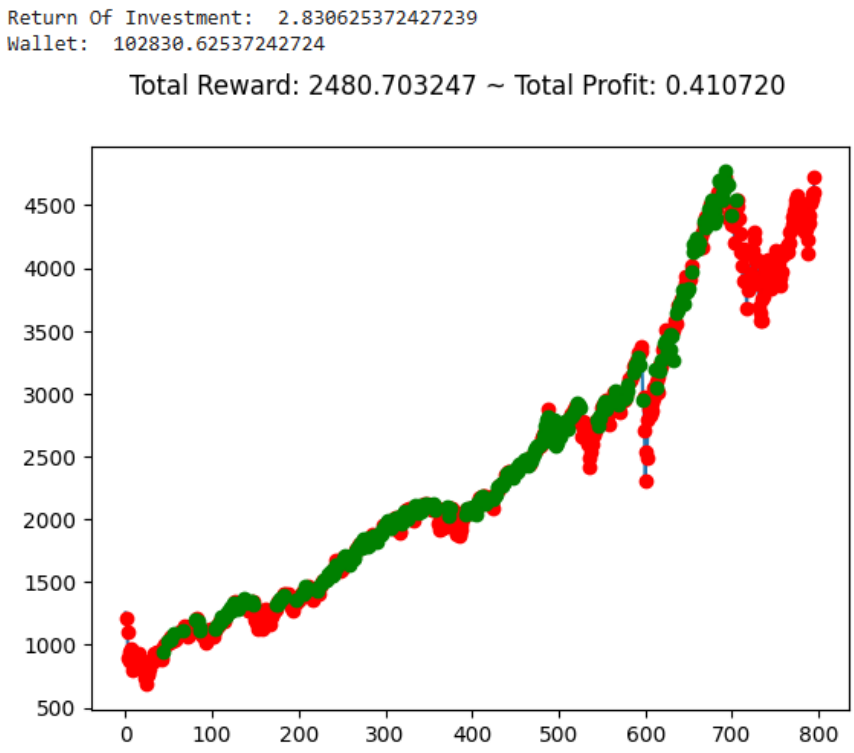

Figure 11: ROI sul Train del 2.83%

anche in questo caso come nel train l'agente guadagna, infatti, un ROI del 2.83% sul test set indica che, rispetto al portafoglio iniziale, il sistema ha generato un guadagno pari al 2.83%.

3.6 Sfide affrontate

Tra le sfide affrontate rientrano:

- **Adattamento dell'Agente al Contesto Specifico**

  Durante l'implementazione dell'algoritmo, abbiamo inizialmente utilizzato un agente basato sul gioco del blackjack seguendo il tutorial link. Tuttavia, la natura specifica del nostro caso d'uso ha richiesto un significativo adattamento dell'agente. Identificando le differenze chiave tra il blackjack e il nostro scenario, abbiamo modificato il codice dell'agente per renderlo compatibile.

- **Scelta del Dataset**

  Nella progettazione dell'algoritmo di training, la scelta del dataset è stata critica per diverse ragioni:

  - **Rappresentatività:** Il dataset doveva essere rappresentativo del problema per evitare problemi di generalizzazione.
  - **Varianza e Diversità:** Doveva catturare la varianza e la diversità dei dati reali per garantire la robustezza del modello.
  - **Quantità e Qualità:** L'abbondanza e la qualità dei dati sono due concetti cruciali. Dati numerosi e ben pre-elaborati (senza la presenza di valore NaN ad esempio) sono essenziali per un addestramento efficace.

- **Gestione delle Ricompense Errate** Un'ulteriore sfida è stata la gestione delle ricompense errate dell'ambiente. Inizialmente, le ricompense incoraggianti spingevano l'agente a concentrarsi esclusivamente sulla vendita, trascurando la possibilità di effettuare acquisti. Questo comportamento distorto era particolarmente evidente nel test set, dove l'agente sembrava propenso a scegliere solo l'opzione di vendita. La correzione di questa anomalia ha richiesto l'implementazione di una nuova funzione di ricompensa che rifletteva più accuratamente gli obiettivi desiderati dell'algoritmo. Questa nuova funzione di ricompensa considera in modo bilanciato sia le azioni di acquisto che quelle di vendita, consentendo all'agente di apprendere strategie più equilibrate. La modifica della funzione di ricompensa ha prodotto un impatto significativo sia sul set di addestramento che su quello di test. Nel test set, in particolare, il comportamento dell'agente è stato migliorato, evidenziando una scelta più ponderata tra acquisto e vendita.

- **Costruzione del ROI (Return on Investment)** Un ulteriore ostacolo significativo affrontato durante l'implementazione ha riguardato la corretta costruzione del ROI, la nostra prima implementazione generava risultati di ROI estremamente elevati, distanti dalla realtà, sollevando preoccupazioni sulla validità del nostro modello, era evidente che la nostra implementazione conteneva errori concettuali che alteravano significativamente la valutazione dell'investimento. Dunque dopo un'analisi approfondita della formula del ROI e delle variabili coinvolte, la nuova implementazione del ROI ha dato i risultati da noi sperati; ha tenuto conto in modo più accurato dei guadagni e delle perdite relative all'investimento, fornendo risultati più realistici e allineati con le aspettative. Questo aggiustamento ha avuto un impatto diretto sulla valutazione delle prestazioni del nostro algoritmo di trading, influenzando positivamente la misurazione del rendimento complessivo.

- **Gestione del Rischio con lo Stop Loss** Al fine di mitigare il rischio di ingenti perdite, abbiamo introdotto la funzionalità di "stop loss" nell'algoritmo di trading. Nella finanza, lo stop loss è una strategia che comporta la chiusura automatica di una posizione quando il prezzo dell'asset raggiunge un determinato livello predeterminato. Nella nostra implementazione, abbiamo stabilito un limite massimo di perdita accettabile per ciascuna transazione. Quando il prezzo dell'asset ha raggiunto questo limite, l'algoritmo ha chiuso automaticamente la posizione, limitando così le perdite potenziali. Questa misura riflette la pratica comune nei mercati finanziari, dove gli operatori stabiliscono livelli di stop loss per proteggere i loro investimenti. L'introduzione dello stop loss ha comportato una gestione più avanzata del rischio nell'algoritmo

di trading, contribuendo a preservare il capitale investito. Questa strategia non solo ha aiutato ad evitare perdite significative, ma ha anche migliorato la stabilità complessiva delle performance dell'algoritmo, consentendo una gestione più efficace del portafoglio.