DESIGNING TRADE LOGIC USING REINFORCEMENT LEARNING

Ashutosh

April 2023

Abstract

The stock market is a perplexing and dynamic framework that is impacted by a great many factors like monetary circumstances, international occasions, and financial backer opinion. Planning productive automated trading procedures requires a profound comprehension of these elements and their effect on stock costs. Venture organizations and multifaceted investments utilize complex calculations and high-level information examination devices to investigate market information and distinguish productive exchanging open doors.

The objective of these methodologies is to amplify venture execution and enhance resource designation by assessing the gamble and likely return of various speculation choices. By utilizing authentic information and factual models, venture organizations can recognize examples and patterns on the lookout and foster prescient models that can be utilized to gauge future cost developments.

Nonetheless, planning a productive procedure is definitely not a simple undertaking. It requires a profound comprehension of market elements, an exhaustive information on measurable models and calculations, and admittance to a lot of excellent information. Besides, the market is continually developing, and new data is continually being integrated into costs, which makes it hard to plan a methodology that can reliably beat the market.

INDEX

1 OBJECTIVE

The main idea of the project is to create an algorithm for a trading strategy using machine learning rewarding concept of reinforcement learning where we train the machine to best fit an action logic to get the highest rewards.

The project has been divided into major sections for clarity of flow and understanding which includes:
1. Understanding the stock market and technical indicator assessment.
2. Understanding Reinforcement Learning Model.
3. Application of the same project using python and its libraries.

An attempt has been made to understand the concepts of the model by carrying out the following:

1. Understanding the reinforcement learning agent.
2. Getting familiar with the concept of Markov Decision Process.
3. Understanding and designing a Markov Reward Process.
4. Getting a hold on the topic of Model Free Prediction.
    - Monte Carlo Reinforcement Learning
    - Temporal Difference Learning
5. Understanding Model Free Control.
6. Setting up action, state and reward functions.
7. Understanding methods to achieve optimal value and policy function.
8. Getting familiar with A2C model.

2 REINFORCEMENT LEARNING

Reinforcement learning (RL) is a type of machine learning that involves training an artificial intelligence (AI) agent to make a sequence of decisions that maximize a reward function. The agent learns to interact with an environment through trial and error, receiving feedback in the form of rewards or punishments based on the actions it takes.

The goal of the agent is to learn a policy that maximizes the expected cumulative reward over time. This can involve exploring the environment to learn more about the reward structure and finding a balance between exploitation (taking actions that are known to lead to rewards) and exploration (trying out new actions to learn more about the environment).

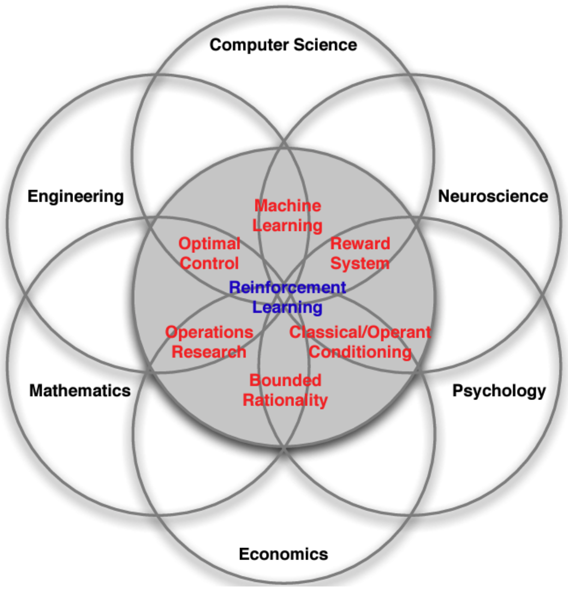


Figure 1: Faces of Reinforcement Learning

2.1 Characteristics of Reinforcement Learning:

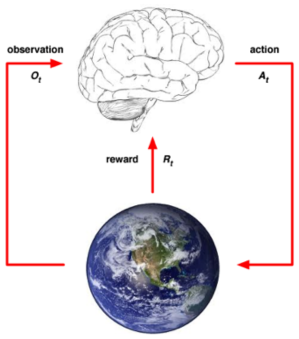
- There is no supervisor, only a reward signal.
- Feedback is delayed, not instantaneous.
- Time really matters (sequential, non-i.i.d data).
- Agent's actions affect the subsequent data it receives.

2.2 Rewards: All goals can be described by the maximization of expected cumulative reward.

- A reward Rt is a scalar feedback signal.
- Indicates how well agent is doing at step t.
- The agent's job is to maximize cumulative reward.

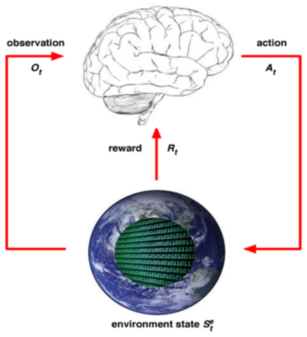
2.3 Agent and Environment Interaction:
At each step t the agent: Executes action = At
Receives observation = Ot
Receives scalar reward = Rt

The environment: Receives action = At
Emits observation = Ot+1
Emits scalar reward = Rt+1
t increments.



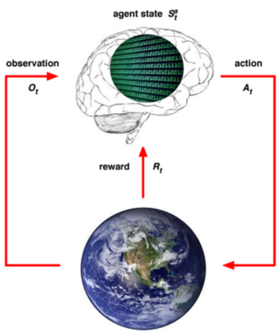
(I) Environmental State S[e,t]: environment's private representation.

- Whatever data the environment uses to pick the next observation/reward.
- The environment state is not usually visible to the agent.
- Even if S[e,t] is visible, it may contain irrelevant information.



(II) Agent State S[a,t]: agent's internal representation.

- Whatever information the agent uses to pick the next action i.e., it is the information used by reinforcement learning algorithms.
- It can be any function of history: S[a,t] = f(Ht)



2.4 Policy
This is how an agent takes an action and decides it's next step.
This is agent's behavior function.

- Deterministic policy: a = π(s)
- Stochastic policy: π(a|s) = P[At=a|St=s]

2.5 Value Function
This function is for prediction of future reward. Value function is used to evaluate the goodness/badness of states and therefore to select actions.

2.6 Model
Model is responsible for prediction of next move of environment.
P = predicts next state. P[a,ss'] = P[St+1=s |St=s, At=a]
R = predicts next reward. R[a,s] = E[Rt+1| St=s, At=a]

3 MARKOV DECISION PROCESS

Markov decision processes formally describe an entirety of the environment for reinforcement learning technique.

3.1 Markov Property

States that the future is independent of the past given the present state.
$P[S_{t+1}|S_t] = P[S_{t+1}|S_1,...,S_t]$ where S is a state that captures all relevant information from the history.

3.2 Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov.

A Markov Decision Process is a tuple$\{S,A,P,R,\gamma\}$.

- S is a finite set of states.
- A is a finite set of actions.
- P is a state transition probability matrix, $P[a,ss'] = P[S_{t+1}=s| S_t=s, A_t=a]$.
- R is a reward function, $R[a,s] = E[R_{t+1}|S_t=s,A_t=a]$.
- γ is a discount factor $\gamma\in [0,1]$.

3.3 Policy

A policy π is a distribution over actions given states, $\pi(a|s)=P[A_t=a| S_t=s]$
MDP policies depend on the current state (not the history).

3.4 Value Function

1. The state-value function $v\pi(s)$ of an MDP is the expected return starting from states, and then following policy π ($v\pi(s) = E\pi[G_t| S_t=s]$)
2. The action-value function $q\pi(s,a)$ is the expected return starting from states, taking action a, and then following policy π ($q\pi(s,a) = E\pi[G_t|S_t=s, A_t=a]$)

3.5 Bellman Expectation Equation

The Bellman expectation equation is a key equation in reinforcement learning that expresses the value of a state or state-action pair as the sum of the immediate reward plus the expected value of the discounted future rewards that can be obtained from that state or state-action pair.

- The state-value function can again be decomposed into immediate reward plus discounted value of successor state, $v\pi(s)=E\pi[R_{t+1}+\gamma v\pi(S_{t+1})|S_t=s]$
- The action-value function can similarly be decomposed,
  $q\pi(s,a) = E\pi[R_{t+1}+\gamma q\pi(S_{t+1},A_{t+1})|S_t=s, A_t=a]$

3.6 Optimal Value Function

The optimal state-value function $v*(s)$ is the maximum value function over all policies
$v*(s)=\max\pi\ v\pi(s)$

The optimal action-value function $q_*(s,a)$ is the maximum action-value function overall policies $q_*(s,a)=\max_\pi q_\pi(s,a)$

3.7 Optimal Policy
For any Markov Decision Process
There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_*\geq\pi, \forall\pi$.
All optimal policies achieve the optimal value function, $v_{\pi_*}(s)=v_*(s)$.
All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$
An optimal policy can be found by maximizing over $q_*(s,a)$.

4 MODEL FREE PREDICTION

4.1 Monte-Carlo Reinforcement Learning
Monte Carlo (MC) reinforcement learning is a type of model-free reinforcement learning that estimates the value of states or state-action pairs by sampling episodes of experience from the environment. Unlike model-based methods that require knowledge of the transition probabilities and rewards of the environment, MC methods learn from direct experience.

In MC methods, the value of a state or state-action pair is estimated as the average of the returns observed from all episodes that visit that state or state-action pair. The return is the sum of the rewards obtained from a given state or state-action pair until the end of the episode. By averaging the returns, MC methods can estimate the expected value of states or state-action pairs.

First visit Monte Carlo Policy
- We take average of the first time we visit a state onwards.
- For example, if you want to estimate the area of a circle, you might generate a random point within a square that circumscribes the circle and then check whether the point is inside the circle. If it is, you count the point as "inside"; if it's not, you count it as "outside." You then estimate the area of the circle as the ratio of the number of points inside the circle to the total number of points generated.

Every visit monte Carlo policy
- Value estimated by mean return.
- For example, if you want to estimate the value of an integral, you might generate a sequence of random points in the integration region and use them to approximate the integral. In every-step Monte Carlo, the estimate of the quantity of interest typically improves as you generate more random samples.

- Goal: learn v_π from episodes of experience under policy π
  $S_1,A_1,R_2,...,S_k\sim\pi$
- Return is the total discounted reward: $G_t=R_{t+1}+\gamma R_{t+2}+...+\gamma^{T-1}R_T$
- Value function is the expected return: $v_\pi(s)=E_\pi[G_t|\ S_t=s]$
- Monte-Carlo policy evaluation uses empirical mean return instead of expected return.

4.2 Temporal-Difference Learning
Temporal-difference (TD) learning is a type of model-free reinforcement learning that estimates the value of states or state-action pairs based on the temporal difference between successive estimates. In TD learning, the value of a state or state-action pair is updated incrementally, based on the difference between the estimated value at the current time step and the estimated value at the next time step.

TD learning combines aspects of both Monte Carlo (MC) and dynamic programming (DP) methods. Like MC methods, TD learning learns directly from experience without requiring a model of the environment. However, unlike MC methods, TD learning updates its value estimates based on the temporal difference between successive estimates, similar to DP methods.

- Goal: learn $v\pi$ online from experience under policy π
- Incremental every-visit Monte-Carlo
  Update value $V(St)$ toward actual return Gt $V(St)\leftarrow V(St)+\alpha(Gt-V(St))$
- Simplest temporal-difference learning algorithm: TD (0)
  Update value $V(St)$ toward estimated return $Rt+1+\gamma V(St+1)$
  $V(St)\leftarrow V(St)+\alpha(Rt+1+\gamma V(St+1) -V(St))$
- $Rt+1+\gamma V(St+1)$ is called the TD target.
- $\delta t=Rt+1+\gamma V(St+1) -V(St)$ is called the TD error.

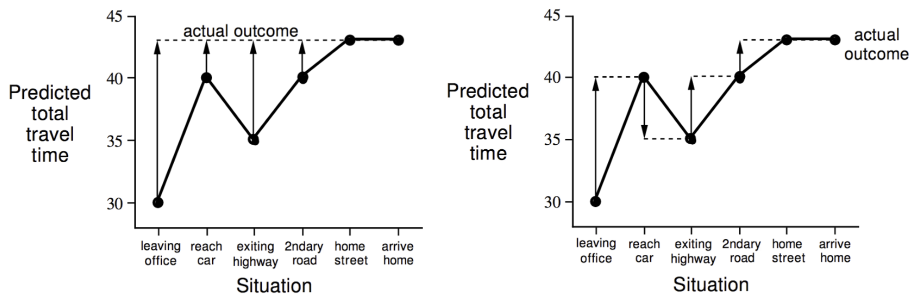
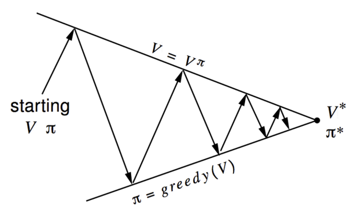


Figure 5: Monte Carlo v/s Temporal Difference

5 MODEL CONTROL PREDICTION

5.1 On-Policy Learning

"Learn on the job" Learn about policy π from experience sampled from π.



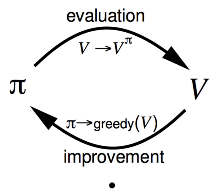


Figure 6: Generalized Policy Iteration

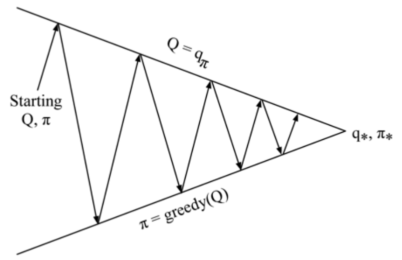


Figure 7: Generalized policy iteration with Action-Value Function

5.1.1 e-Greedy Exploration

At each time step, the agent selects the action with the highest estimated reward with probability 1-ε (where ε is a small positive number called the exploration rate or epsilon), and selects a random action with probability ε. By selecting a random action with some probability ε, the agent is able to explore the environment and learn about other possible actions and their rewards.

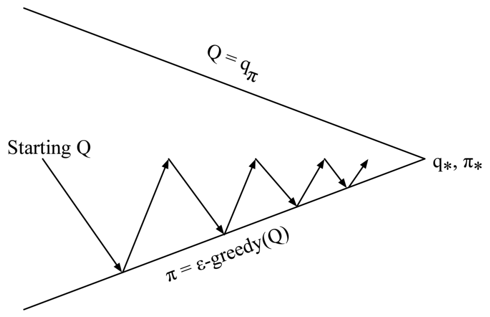


Figure 8: Monte Carlo and SARSA with e-greedy

5.2 Off-Policy Learning

"Look over someone's shoulder". Learn about policy π from experience sampled from μ.

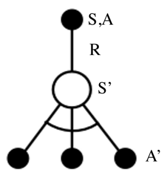
5.2.1 Q-Learning

In Q-learning, the agent maintains an estimate of the optimal action-value function, denoted as Q(s, a), where s is the current state and a is the action taken by the agent in that state. The agent uses this estimate to select the action with the highest Q-value in each state.

Q-learning updates its estimate of the action-value function using the Bellman equation, which relates the value of a state to the values of its successor states:

$Q(s, a) = Q(s, a) + \alpha[r + \gamma \text{maxa}' Q(s', a') - Q(s, a)]$

where α is the learning rate, r is the reward received after taking action a in states, s' is the next state, a' is the next action, γ is the discount factor, and maxa' Q(s', a') is the maximum Q-value over all possible actions in the next states'.



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S,A) \right)$$

Figure 9: Q-Learning Control Algorithm

6 TRADE STAGE SETUP

6.1 SPY

"SPY" is the ticker symbol for the SPDR S&P 500 exchange-traded fund (ETF), which tracks the performance of the S&P 500 index, a broad benchmark of the U.S. stock market. As an ETF, SPY allows investors to gain exposure to a diversified portfolio of stocks representing a wide range of sectors and industries.

Investors often look to SPY as a barometer of the overall health of the U.S. stock market. Since its inception in 1993, SPY has become one of the most popular ETFs, with assets under management (AUM) in the hundreds of billions of dollars.

- Diversification: The S&P 500 index includes 500 of the largest and most influential companies in the U.S., across a variety of sectors. Investing in SPY provides exposure to a diversified portfolio of stocks, which can help reduce risk.
- Liquidity: SPY is one of the most actively traded ETFs in the world, with high trading volumes and tight bid-ask spreads. This means that investors can easily buy and sell shares of SPY without having to worry about liquidity issues.
- Low costs: SPY has a very low expense ratio compared to many other mutual funds and ETFs, which means that investors can keep more of their returns.
- Historical performance: The S&P 500 has historically delivered strong long-term returns, and SPY provides investors with a way to capture this performance.
- Accessibility: SPY can be traded like a stock, which means that investors can buy and sell shares through their brokerage accounts. This makes it easy for investors of all sizes to access the benefits of investing in the S&P 500 index.

6.2 Technical Indicators

Technical indicators are mathematical calculations based on a security's price and/or volume that are used to analyze and forecast price movements in financial markets. Technical analysts use these indicators to identify potential buying or selling opportunities, to confirm price trends, and to assess market strength or weakness.

(1) Simple Moving Average (SMA)

Simple Moving Average is a technical indicator used in financial analysis to identify trends in a security's price over a specified period. It is calculated by taking the average price of a security over a specific number of time periods, such as 10, 20, 50, or 200 days.

For example, a 50-day SMA is calculated by adding up the closing prices of a security for the past 50 days and dividing the sum by 50. As the SMA is calculated using past price data, it is considered a lagging indicator and is commonly used to confirm trends and support/resistance levels.

SMA can be used in conjunction with other technical indicators and chart patterns to help traders make informed decisions about when to buy or sell a security. For example, if the price of a security is above its 50-day SMA, it may be seen as a bullish signal, indicating a potential upward trend, while if the price is below its 50-day SMA, it may be seen as a bearish signal, indicating a potential downward trend.

(2) Relative Strength indicator (RSI)

Relative Strength Index is a technical indicator used to measure the strength of a security's price action over a specified period. It is a momentum oscillator that compares the magnitude of recent gains to recent losses in an attempt to determine overbought and oversold conditions of the security.

The RSI is typically calculated using 14 periods and ranges from 0 to 100. A reading above 70 is considered overbought, indicating that the security may be due for a price correction, while a reading below 30 is considered oversold, indicating that the security may be due for a price rebound.

Traders use the RSI to identify potential buy and sell signals. For example, if the RSI reaches an overbought level of 70, a trader may consider selling the security, and if the RSI reaches an oversold level of 30, a trader may consider buying the security. However, it is important to note that the RSI is just one indicator, and traders should use it in conjunction with other technical indicators and fundamental analysis to make informed trading decisions.

(3) Commodity Channel Index (CCI)

Commodity Channel Index is a technical indicator used to measure the variation of a security's price from its statistical average. It was initially developed for commodity markets but is now commonly used for other financial instruments as well.

The CCI is calculated by taking the difference between the typical price (average of high, low, and closing prices) and its simple moving average over a specified period, and then dividing that difference by a constant multiple of the mean absolute deviation of the typical price. The result is a standardized value that typically ranges between -100 and +100.

A reading above +100 is considered overbought, indicating that the security may be due for a price correction, while a reading below -100 is considered oversold, indicating that the security may be due for a price rebound. Traders use the CCI to identify potential buy and sell signals, with buy signals generated when the CCI crosses up from below -100, and sell signals generated when the CCI crosses down from above +100.

(4) Average Directional Index (ADX)

Average Directional Index is a technical indicator used to measure the strength of a security's trend. It was developed by J. Welles Wilder, Jr., and is typically used in conjunction with the Directional Movement Indicator (DMI) to generate buy and sell signals.

The ADX is calculated by taking the difference between the directional movement indicators (+DI and -DI) and dividing that difference by the sum of the directional movement indicators over a specified period, usually 14 days. The result is a standardized value that ranges from 0 to 100.

A high ADX reading indicates a strong trend, while a low ADX reading indicates a weak trend or sideways market. Traders use the ADX to identify potential buy and sell signals, with buy signals generated when the ADX is rising from below 25 and sell signals generated when the ADX is falling from above 25.

(5) Moving Average Convergence Divergence

Moving Average Convergence Divergence is a technical indicator used to identify potential trend reversals and momentum in a security's price action. It is calculated by subtracting a 26-period exponential moving average from a 12-period exponential moving average, and then plotting a 9-period exponential moving average of the difference as a signal line.

The MACD line oscillates above and below the zero line, indicating whether the short-term moving average is above or below the long-term moving average. A positive MACD value indicates that the short-term moving average is above the long-term moving average, indicating upward momentum, while a negative MACD value indicates that the short-term moving average is below the long-term moving average, indicating downward momentum.

Traders use the MACD to generate buy and sell signals. For example, a bullish signal is generated when the MACD line crosses above the signal line, indicating that upward momentum may be increasing, while a bearish signal is generated when the MACD line crosses below the signal line, indicating that downward momentum may be increasing.

(6) Fibonacci Levels

Fibonacci levels are a series of horizontal lines that are drawn on a chart to identify potential support and resistance levels based on the Fibonacci sequence of numbers. The sequence is derived by adding the two previous numbers to get the next number, starting with 0 and 1 (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, etc.).

The most commonly used Fibonacci levels are 23.6%, 38.2%, 50%, 61.8%, and 100%. These levels are calculated by taking the distance between two price points (usually a swing high and a swing low) and multiplying it by the Fibonacci ratios.

Traders use Fibonacci levels to identify potential support and resistance levels. For example, if a security is trending higher and then retraces back to a Fibonacci level (such as the 38.2% or 50% level), it may find support at that level and bounce back up. Conversely, if a security is trending lower and then rallies to a Fibonacci level (such as the 38.2% or 61.8% level), it may find resistance at that level and turn back down.

6.3 Reinforcement Model Setup

6.3.1 Data

Training Data Set: start='2000-01-01', end='2015-12-31'. Close Prices of SPY.
Testing Data Set: start='2016-01-01', end='2022-12-31'. Close Prices of SPY.

6.3.2 State Space

1. 'SPY'
2. 'SMA'
3. 'RSI'
4. 'CCI'
5. 'ADX'
6. 'MACD'
7. 'fib_level1'
8. 'fib_level2'
9. 'fib_level3'
10. 'fib_level4'
11. 'fib_level5'

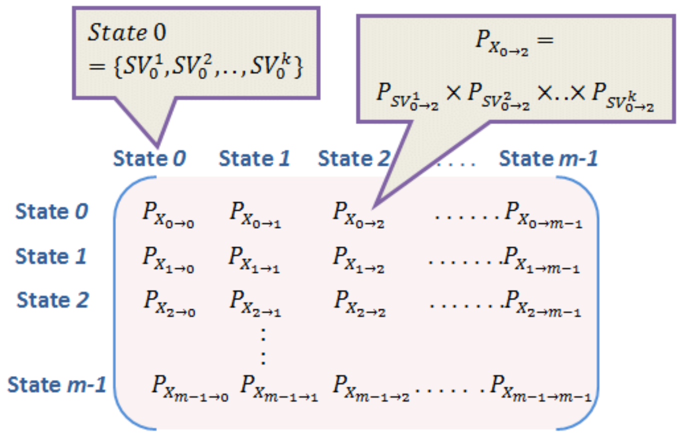

$State\ 0 = \{SV_0^1, SV_0^2, .., SV_0^k\}$
$P_{X_{0\to2}} = P_{SV_{0\to2}^1} \times P_{SV_{0\to2}^2} \times .. \times P_{SV_{0\to2}^k}$
State 0
State 1
State 2
State m-1
$P_{X_{0\to0}}$ $P_{X_{0\to1}}$ $P_{X_{0\to2}}$ ...... $P_{X_{0\to m-1}}$
$P_{X_{1\to0}}$ $P_{X_{1\to1}}$ $P_{X_{1\to2}}$ ...... $P_{X_{1\to m-1}}$
$P_{X_{2\to0}}$ $P_{X_{2\to1}}$ $P_{X_{2\to2}}$ ...... $P_{X_{2\to m-1}}$
$P_{X_{m-1\to0}}$ $P_{X_{m-1\to1}}$ $P_{X_{m-1\to2}}$ ...... $P_{X_{m-1\to m-1}}$

Figure 10: General State Matrix

6.3.3 Action Space
1. Sell
2. Buy

6.3.4 Reward Function
1. +1 -> Profitable position
2. -1 -> Loss position

7 Advantage Actor-Critic (A2C)

Critic-only approach: the critic-only learning approach, which is the most common, solves a discrete action space problem using, for example, Q-learning, Deep Q-learning (DQN), and its improvements, and trains an agent on a single stock or asset. The idea of the critic-only approach is to use a Q-value function to learn the optimal action-selection policy that maximizes the expected future reward given the current state. Instead of calculating a state-action value table, DQN minimizes the mean squared error between the target Q-values and uses a neural network to perform function approximation. The major limitation of the critic-only approach is that it only works with discrete and finite state and action spaces, which is not practical for a large portfolio of stocks since the prices are of course continuous.

Actor-only approach: The idea here is that the agent directly learns the optimal policy itself. Instead of having a neural network to learn the Q-value, the neural network learns the policy. The policy is a probability distribution that is essentially a strategy for a given state, namely the likelihood to take an allowed action. The actor-only approach can handle continuous action space environments.

Policy Gradient: aims to maximize the expected total rewards by directly learning the optimal policy itself.

Actor-Critic approach: The actor-critic approach has been recently applied in finance. The idea is to simultaneously update the actor-network that represents the policy and the critic network that represents the value function. The critic estimates the value function, while the actor updates the policy probability distribution guided by the critic with policy gradients. Over time, the actor learns to take better actions and the critic gets better at evaluating those actions. The actor-critic approach has proven to be able to learn and adapt to large and complex environments and has been used to play popular video games, such as Doom. Thus, the actor-critic approach fits well in trading with a large stock portfolio.

A2C: A2C is a typical actor-critic algorithm. A2C uses copies of the same agent working in parallel to update gradients with different data samples. Each agent works independently to interact with the same environment.

mlpPolicy : **mlpPolicy** is a type of policy architecture used in reinforcement learning, specifically in the context of the Proximal Policy Optimization (PPO) algorithm. mlpPolicy stands for multi-layer perceptron policy, and is a neural network architecture that consists of multiple fully connected (dense) layers.

In the mlpPolicy, the input to the neural network is typically the current state of the environment, and the output is a probability distribution over the possible actions that the agent can take in that state. The architecture is called "multi-layer perceptron" because it consists of several layers of neurons, each of which is fully connected to the neurons in the previous and next layers. The number of layers and the number of neurons in each layer can be varied depending on the complexity of the task and the amount of available computing resources.

8 OUTCOMES

8.1 Cumulative Profit

SPY Buy and Hold Strategy returns:
106% on the overall period.
15.14% per annum.
1.26% per month.
Sharpe ratio: 0.647

A2C trading logic returns:
360.86023% on the overall period.
51.55% per annum.
4.3% per month.
Sharpe ratio: 2.658

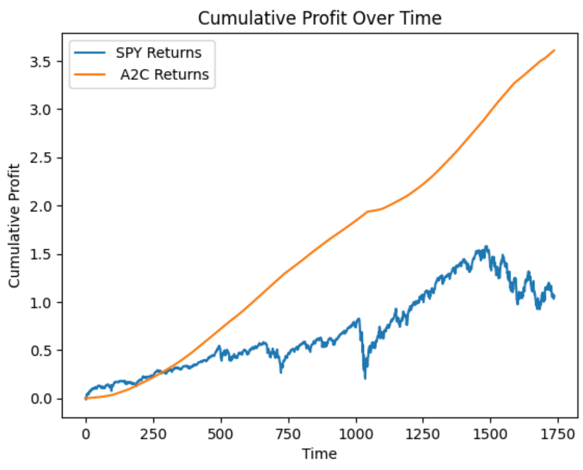


Figure 11: SPY and A2C cumulative returns

8.2 Profits per trading day from model

Mean daily return: 0.2080%

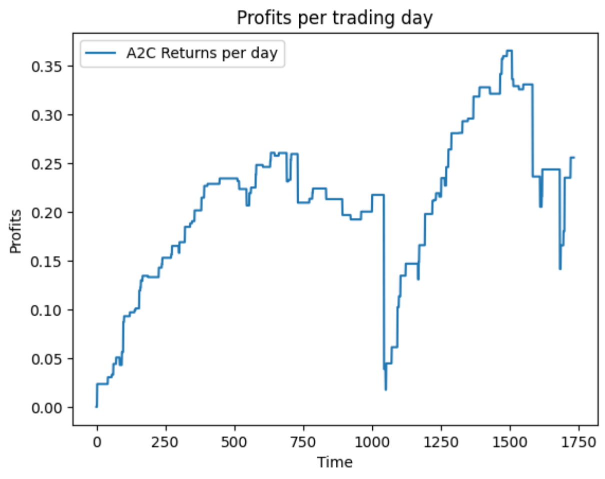
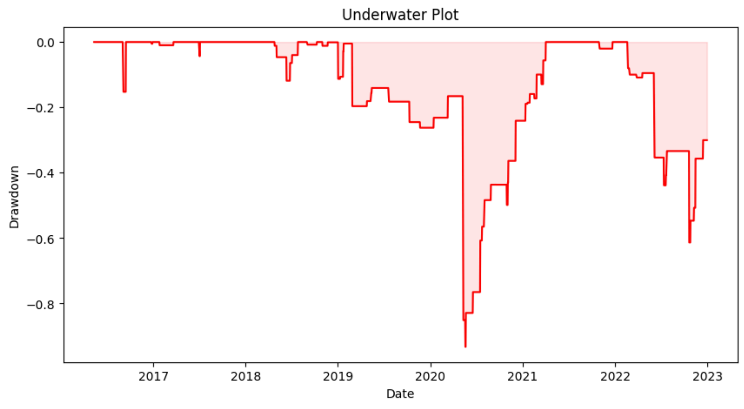


Figure 12: Daily returns from A2C model.

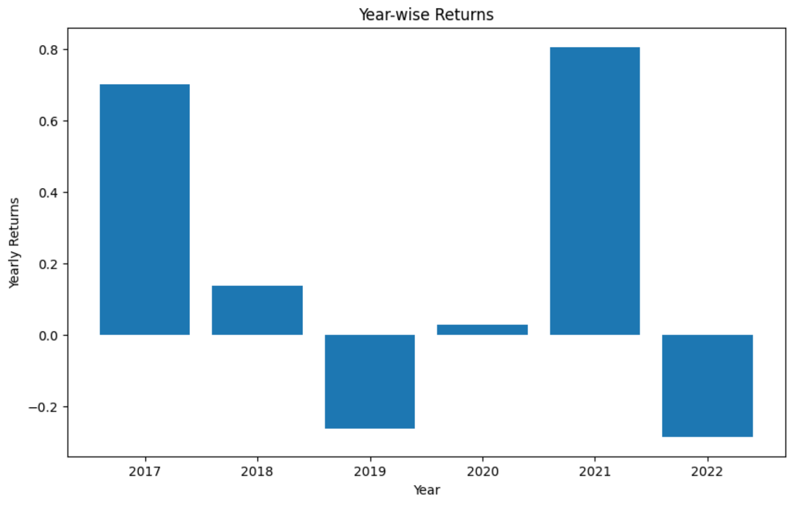
8.3 Maximum Drawdown

Maximum drawdown is a risk metric used to measure the largest loss an investment has experienced over a given time period.

Maximum drawdown of the A2C model is 80%.



8.4 Year wise returns.



9 RESULTS

The project exposes us to varied understandings of reinforcement learning and its related concepts.
The project caters to the following learnings:
1. Research about the stock and technical indicators.
2. Learning about various reinforcement logics and implementation.
3. Modelling a A2C model into practical trading logic.

10 REFERENCE

1. https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02

2. https://www.davidsilver.uk/teaching/

3. https://www.youtube.com/watch?v=KHZVXao4qXs&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&index=7

4. https://finance.yahoo.com/quote/SPY/history?p=SPY

5. https://www.investopedia.com/

APPENDIX:

CODE:

```
import numpy as np
import yfinance as yf
import pandas as pd
import ta
import gym
import stable_baselines3
import matplotlib.pyplot as plt
import pyfolio
import seaborn as sns

from stable_baselines3 import A2C
from stable_baselines3 import PPO
from stable_baselines3 import DDPG

from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3.common.env_checker import check_env

# Training the model using training data


class TradingEnv(gym.Env):
    def __init__(self, data):
        self.df = data
        self.reward_range = (-1, 1)

        self.action_space = gym.spaces.Discrete(2)
        self.observation_space = gym.spaces.Box(low=-50, high=50, shape=(11,))

        self.reset()

    def reset(self):
        self.current_step = 0
        self.profit = 0
        self.bought = False
        self.done = False

        obs = self.df.iloc[self.current_step][['SPY', 'SMA', 'RSI', 'CCI', 'ADX', 'MACD',
'fib_level1','fib_level2','fib_level3','fib_level4','fib_level5']].values / self.df.max().values

        return obs

    def step(self, action):
        if action == 0:  # Buy
            if not self.bought:
                self.bought = True
                self.buy_price = self.df.iloc[self.current_step]['SPY']
        elif action == 1:  # Sell
            if self.bought:
                self.bought = False
            self.sell_price = self.df.iloc[self.current_step]['SPY']
            self.profit += (self.sell_price - self.buy_price) / self.buy_price

        self.current_step += 1
        if self.current_step == len(self.df):
            self.done = True

        obs = self.df.iloc[self.current_step][['SPY', 'SMA', 'RSI', 'CCI', 'ADX', 'MACD',
'fib_level1','fib_level2','fib_level3','fib_level4','fib_level5']].values / self.df.max().values
        reward = self.profit

        return obs, reward, self.done, {}


df = yf.download('SPY', start='2000-01-01', end='2015-12-31', index_col='Date')['Close'].reset_index()
df.drop('Date', axis=1, inplace=True)
df = df.rename(columns={'Close': 'SPY'})

df['SMA'] = df['SPY'].rolling(window=20).mean()
df['RSI'] = ta.momentum.RSIIndicator(df['SPY'], window=20).rsi()
macd = ta.trend.MACD(df['SPY'])
df['MACD'] = macd.macd()
cci = ta.trend.cci(high = df['SPY'], low = df['SPY'], close = df['SPY'], window=20)
df['CCI'] = cci
adx = ta.trend.ADXIndicator(high=df['SPY'], low=df['SPY'], close=df['SPY'], window=20)
df['ADX'] = adx.adx()

#Fibonacci Retracement for support levels
high = df['SPY'].rolling(window=20).max()
low = df['SPY'].rolling(window=20).min()
diff = high - low
levels = [0.236, 0.382, 0.5, 0.618, 0.786]

fib_levels = []
for level in levels:
    fib = high - diff * level
    fib_levels.append(fib)

fib_level1 = fib_levels[0]
df['fib_level1'] = fib_level1
fib_level2 = fib_levels[1]
df['fib_level2'] = fib_level2
fib_level3 = fib_levels[2]
df['fib_level3'] = fib_level3
fib_level4 = fib_levels[3]
df['fib_level4'] = fib_level4
fib_level5 = fib_levels[4]
df['fib_level5'] = fib_level5


df.dropna(inplace=True)
df = df.astype(np.float32)
```

```
env = TradingEnv(df)
check_env(env)

env = DummyVecEnv([lambda: env])
model = A2C('MlpPolicy', env, verbose=1)
model.learn(total_timesteps=3995)


# Testing the model using testing data
# Load the test data
test_data = yf.download('SPY', start='2016-01-01', end='2022-12-31', index_col='Date')['Close'].reset_index()
test_data.drop('Date', axis=1, inplace=True)
test_data = test_data.rename(columns={'Close': 'SPY'})

#test_data = pd.read_excel('SPY_testing.xlsx')
test_data = test_data.astype(np.float32)

test_data['SMA'] = test_data['SPY'].rolling(window=20).mean()
test_data['RSI'] = ta.momentum.RSIIndicator(test_data['SPY'], window=20).rsi()
macd = ta.trend.MACD(test_data['SPY'])
test_data['MACD'] = macd.macd()
cci = ta.trend.cci(high = test_data['SPY'], low = test_data['SPY'], close = test_data['SPY'], window=20)
test_data['CCI'] = cci
adx = ta.trend.ADXIndicator(high=test_data['SPY'], low=test_data['SPY'], close=test_data['SPY'],
window=20)
test_data['ADX'] = adx.adx()

high = test_data['SPY'].rolling(window=20).max()
low = test_data['SPY'].rolling(window=20).min()
diff = high - low
levels = [0.236, 0.382, 0.5, 0.618, 0.786]

fib_levels = []
for level in levels:
    fib = high - diff * level
    fib_levels.append(fib)

fib_level1 = fib_levels[0]
test_data['fib_level1'] = fib_level1
fib_level2 = fib_levels[1]
test_data['fib_level2'] = fib_level2
fib_level3 = fib_levels[2]
test_data['fib_level3'] = fib_level3
fib_level4 = fib_levels[3]
test_data['fib_level4'] = fib_level4
fib_level5 = fib_levels[4]
test_data['fib_level5'] = fib_level5


test_data.dropna(inplace=True)
test_data = test_data.astype(np.float32)

# Initialize the trading environment
```

```
test_env = TradingEnv(test_data)
test_env = DummyVecEnv([lambda: test_env])

# Run the test
obs = test_env.reset()
done = False
profits = []
actions = []

k = 0
while k!=len(test_data)-1:
    action, _ = model.predict(obs)
    obs, reward, done, info = test_env.step(action)
    if not done:
        profits.append(reward)
        actions.append(action)
    k=k+1

cumulative_profits = np.cumsum(profits)


SPY_benchmark = test_data['SPY']
SPY_benchmark_temp = test_data['SPY'].tolist()
buy_and_hold = [(val - SPY_benchmark_temp[0])/SPY_benchmark_temp[0] for val in
SPY_benchmark_temp]


plt.plot(buy_and_hold, label="SPY Returns")
plt.plot(cumulative_profits/100, label=" A2C Returns")
plt.xlabel('Time')
plt.ylabel('Cumulative Profit')
plt.title('Cumulative Profit Over Time')
plt.legend(loc="upper left")
plt.show()


SPY_returns = test_data['SPY'].pct_change().dropna()
SPY_sharpe_ratio = np.sqrt(252) * SPY_returns.mean() / SPY_returns.std()
print("Sharpe ratio of Buy-Hold SPY:",SPY_sharpe_ratio)

Strategy_sharpe_ratio = np.diff(cumulative_profits).mean() / np.diff(cumulative_profits).std()
print("Sharpe ratio of A2C trading strategy SPY:",Strategy_sharpe_ratio)

plt.plot(np.diff(cumulative_profits), label="A2C Returns per day")
plt.xlabel('Time')
plt.ylabel('Profits')
plt.title('Profits per trading day')
plt.legend(loc="upper left")
plt.show()

print("Mean daily return: {:.4%}".format(np.diff(cumulative_profits).mean()/100))
# Calculate daily returns using percentage change formula
```

```
start_date = '2016-5-7'
end_date = '2022-12-31'
dates = pd.date_range(start=start_date, end=end_date, freq='B')

SPY_benchmark = test_data['SPY']
#daily_returns = SPY_benchmark.pct_change().values
SPY_daily_returns = [(val - SPY_benchmark.values[0])/SPY_benchmark.values[0] for val in
SPY_benchmark]
mask = ~np.isnan(SPY_daily_returns)
#daily_returns = daily_returns[mask]

SPY_Sharpe_Ratio = (252**0.5)*np.array(SPY_daily_returns).mean()/np.array(SPY_daily_returns).std()
A2C_returns_series = pd.Series((np.diff(cumulative_profits/100)).astype(float), index=dates[0:])

SPY_daily_returns_series = pd.Series(SPY_daily_returns[2:], index=dates[0:])
SPY_daily_returns_series.index = pd.to_datetime(SPY_daily_returns_series.index.strftime('%Y-%m-
%d'))

A2C_returns_series.index = pd.to_datetime(A2C_returns_series.index.strftime('%Y-%m-%d'))


# Define a function to calculate drawdowns
def drawdown(A2C_returns_series):
    max_return = np.maximum.accumulate(A2C_returns_series)
    return (A2C_returns_series - max_return) / max_return

# Calculate drawdowns
dd = drawdown(A2C_returns_series)

# Plot the result
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(dd, color='red')
ax.fill_between(dd.index, dd, 0, color='red', alpha=0.1)
ax.set(title='Underwater Plot', xlabel='Date', ylabel='Drawdown')
plt.show()

yearly_returns = A2C_returns_series.resample('Y').last().pct_change()

# Plot the yearly returns
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(yearly_returns.index.year, yearly_returns.values)
ax.set_xlabel('Year')
ax.set_ylabel('Yearly Returns')
ax.set_title('Year-wise Returns')
plt.show()
```