FINSEARCH 2024

Use Deep Reinforcement Learning (Rl) To Optimize Stock Trading Strategy & Thus Maximize Investment Return

MID TERM REPORT

July, 2024

TEAM MEMBERS

Sangeetha D (22B3310)

Hrishikesh S (22B4217)

Jatin Kumar (22B3922)

Oviya S (22B3918)

MENTOR

Likhitha Sree

CONTENTS

1. INTRODUCTION

1.1. Motivation:

The basic motivation behind this project is to optimize stock trading strategies using Deep Reinforcement Learning (RL) to maximize investment returns. In the current trading environment, strategies often rely on heuristics or simpler models which may not capture the complex dynamics of the stock market effectively. By utilizing Deep RL, the project aims to create more sophisticated trading strategies that can adapt to market conditions dynamically.

- **Current Situation:** Traditional optimization methods in stock trading include rule-based systems, technical analysis, and basic machine learning models. These methods have limitations in adapting to the ever-changing market conditions.
- **Need for Optimization:** Effective optimization can lead to better decision-making, reduced risk, and increased returns. It is crucial for traders and investors to employ strategies that not only respond to historical data but also adapt to new information in real-time.
- **Role of Deep RL:** Deep RL combines the power of deep learning with reinforcement learning, enabling the creation of agents that can learn and adapt from their interactions with the market environment.
- **Impact on Trading:** This project aims to demonstrate how Deep RL can be applied to develop trading strategies that outperform traditional methods, potentially leading to higher profitability and more efficient market operations.

1.2. Workflow:

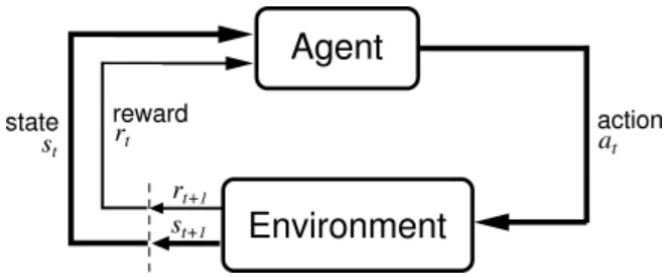
➔ Provide a fundamental understanding of Reinforcement Learning and Deep Learning
➔ Gather and prepare stock data for training and testing
➔ Implement various RL algorithms, including Deep Q-Network (DQN) and others
➔ Optimize models and compare their performance against benchmarks
➔ Test models on live data to validate their efficacy
➔ Summarize findings

2. REINFORCEMENT LEARNING AND DEEP Q-NETWORK

2.1. Reinforcement Learning

2.1.1. Basic Concept:

The basic ideology behind Reinforcement Learning (RL) is to model how human beings learn. Human beings try **actions** and, after a few iterations, learn the best way to achieve a desired goal. The driving force for us to take a particular approach is the **reward** or benefit we get out of it. Similarly, an RL agent tries to maximize its rewards by making decisions at every step. The **environment** plays a crucial role here, providing feedback in the form of rewards or punishments. This **feedback** guides the agent's learning process. During training, the agent learns how to perform tasks independently, and it uses this learned model to achieve the desired behavior in real-world scenarios.



2.1.2. Key Terminologies:

- **Agent**: Decision maker or learner
- **Environment**: A physical world where an agent learns and decides the actions to be performed
- **State (s)**:The current situation of the agent in the environment
- **Action (s)**: An agent's single choice
- **Policy (π)**:the strategy that agent follows to determine it's actions based on the current state
- **Value function (V(s))**: The value of state shows up the reward achieved starting from the state until the policy is executed
- **Q-value function (Q(s,a))**: The expected cumulative reward an agent can expect to receive starting from a particular state, taking a specific action, and then following a specific policy.
- **Discount factor (γ)**: Parameter that determines the importance of future rewards relative to immediate rewards in the agent's decision-making process.
- **Policy gradient**: A method in RL that directly optimizes the policy function by gradient ascent in the direction that increases the expected return.
- **Temporal Difference (TD) Learning**: Type of learning where the value function is updated based on the difference between the estimated value of the current state and the next state.

2.1.3. Algorithms:

In reinforcement learning, the agent learns using trial and error to maximize rewards. There are two models which are widely used:

Traditional RL models:

This is suitable for smaller environments and rely on simpler function approximation. Some examples are Q-learning, temporal difference learning, Markov Decision Process (MDP's), SARSA (State-Action-Reward-State-Action) etc.

Deep Reinforcement Learning Models:

This integrates deep learning techniques with reinforcement learning to solve complex and high-dimensional state and action spaces. Some examples are Deep Q-networks(DQN), Natural Language Processing (NLP), etc.

2.2. Deep Q-Network Algorithm

2.2.1. Overview:

Deep Q-Network (DQN) is an RL algorithm that uses a neural network to approximate the **Q-value function**, enabling the agent to handle high-dimensional state spaces effectively. It is particularly useful for problems where the state space is too large to represent explicitly.

2.2.2. Key Components:

- **Experience Replay**: A memory buffer that stores the agent's experiences, allowing the agent to learn from past experiences and break the correlation between consecutive samples.
- **Target Network**: A separate network to provide stable Q-value targets, reducing the risk of divergence during training.
- **Neural Network**: Used to approximate the Q-function, mapping states to Q-values for each action.

2.2.3. Advantages:

- **Handles High-dimensional State Spaces**: Capable of processing complex and high-dimensional inputs like images or stock market data.
- **Improved Stability**: The use of experience replay and target networks enhances training stability.
- **Generalization**: Neural networks can generalize across similar states, leading to more robust policies.

2.2.3. Limitations:

- **Sample Inefficiency**: Requires a large amount of data for training, which can be computationally expensive.
- **Exploration vs. Exploitation**: Balancing exploration and exploitation can be challenging, especially in high-dimensional spaces.
- **Overestimation Bias**: Q-value estimates can be biased, potentially leading to suboptimal policies.

3. INVERTED PENDULUM PROBLEM

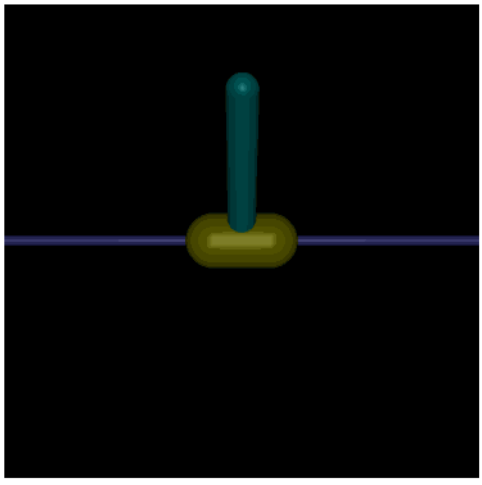
3.1. Dynamics Of The Inverted Pendulum

The inverted pendulum problem involves balancing a pendulum in an upright position by controlling the movement of its base. So we are aiming to use **OpenAI's cartpole environment** to create this model.

System description: The inverted pendulum is mounted on a cart. And it can move only in a single plane and the cart moves horizontally along a track.

State Variables: The state of the system is described by the angle of the pendulum from the vertical position(theta) and the position of the cart along the track(x). These define the configuration of the pendulum-cart system.

Control Input: The control input typically represents the force applied to the cart to move it along the track. This force affects the position of the cart and the angular position of the pendulum.



3.1.1. Equations of motion:

The equations of motion can be derived using Lagrangian mechanics, which explains the systems dynamics

$$\tau = I\ddot{\theta} + mglsin(\theta)$$

where:

- **τ** is the torque applied to the base,
- **I** is the moment of inertia of the pendulum,
- $\ddot{\theta}$ is the angular acceleration,
- **m** is the mass of the pendulum,
- **g** is the acceleration due to gravity,
- **l** is the length of the pendulum,
- **θ** is the angular displacement.

3.1.2. Observation Space:

The observation space in a reinforcement learning environment represents the state of the environment at any given time. It is the input that the agent observes and uses to decide on an action. In the CartPole environment, the observation space consists of the following four variables:

- **Cart Position**: The position of the cart on the track.
- **Cart Velocity**: The velocity at which the cart is moving.
- **Pole Angle**: The angle between the pole and the vertical axis.
- **Pole Angular Velocity**: The rate at which the pole's angle is changing.

3.1.2. Action Space:

The action space is discrete and consists of two possible actions:

- **Move Cart to the Left**: Applying a force to move the cart left.
- **Move Cart to the Right**: Applying a force to move the cart right.

3.2. Reward Function Design

3.2.1. Criteria for reward calculation:

The criteria for reward calculation typically focus on incentivizing actions that stabilize the pendulum upright while keeping the cart within operational bounds. The goal is to design a reward function that guides the reinforcement learning agent towards learning effective control policies that achieve stable and balanced behavior in the system.

Simply put, the system rewards you for each "moment" you keep the pendulum upright with 1 point until it topples over.

3.2.2. Reward function guiding the learning progress:

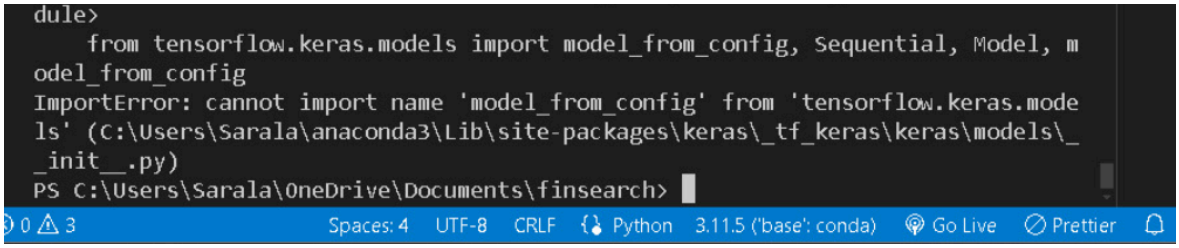
The reward function guides an agent's learning by assigning **values** to actions, encouraging behaviors that stabilize the inverted pendulum and penalizing deviations. It shapes the agent's **exploration-exploitation balance** by reinforcing effective strategies and discouraging ineffective ones.

By providing **immediate** feedback, it helps the agent learn the consequences of its actions over time. Iterative refinement based on agent performance enhances the reward function's effectiveness in guiding the agent toward optimal control policies.

- **Stability**: Reward should promote stability of the pendulum in the upright position.
- **Control Effort**: Penalize large control efforts to encourage efficient use of energy.
- **Smoothness**: Encourage smooth and gradual movements to prevent oscillations.

4. CHALLENGES FACED

We encountered challenges while running the code in **VS Code** due to compatibility issues between **Keras** and **TensorFlow**. We also attempted to run the code in Jupyter Notebook but faced similar problems. Eventually, we switched to **Google Colab**, where we successfully ran the code without.



5. CODE

```
import gym

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.optimizers import Adam

from rl.agents.dqn import DQNAgent

from rl.policy import EpsGreedyQPolicy

from rl.memory import SequentialMemory

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,Flatten

from tensorflow.keras.optimizers import Adam

# Set up the environment

env = gym.make('CartPole-v1')

env._max_episode_steps = 500

np.random.seed(123)

env.seed(123)

nb_actions = env.action_space.n



# Build the neural network model

model = Sequential()

model.add(Flatten(input_shape=(1,) +
env.observation_space.shape))
```

```
model.add(Dense(24, activation='relu'))

model.add(Dense(24, activation='relu'))

model.add(Dense(nb_actions, activation='linear'))

print(model.summary())


# Configure and compile the DQN agent

memory = SequentialMemory(limit=50000, window_length=1)

policy = EpsGreedyQPolicy()

dqn = DQNAgent(model=model, nb_actions=nb_actions,
memory=memory, nb_steps_warmup=10,

               target_model_update=1e-2, policy=policy)

dqn.compile(Adam(learning_rate=1e-3), metrics=['mae'])


# Train the DQN agent

dqn.fit(env, nb_steps=50000, visualize=True, verbose=2)


# Test the DQN agent

dqn.test(env, nb_episodes=5, visualize=True)
```

6. RESULTS

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:43: DeprecationWarning: WARN: The argument mode in render method is deprecated; use render_mode during environment initialization instead.
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(

 49280/50000: episode: 313, duration: 7.953s, episode steps: 395, steps per second:  50, episode reward: 395.000, mean reward: 1.000 [ 1.000,  1.000], mean action: 0.494 [0.000, 1.000],  loss: 5.849472, mae: 45.396496, mean_q: 91.245384

/usr/local/lib/python3.10/dist-packages/gym/core.py:43: DeprecationWarning: WARN: The argument mode in render method is deprecated; use render_mode during environment initialization instead.
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(

 49626/50000: episode: 314, duration: 7.011s, episode steps: 346, steps per second:  49, episode reward: 346.000, mean reward: 1.000 [ 1.000,  1.000], mean action: 0.494 [0.000, 1.000],  loss: 8.377810, mae: 46.665852, mean_q: 93.681755

/usr/local/lib/python3.10/dist-packages/gym/core.py:43: DeprecationWarning: WARN: The argument mode in render method is deprecated; use render_mode during environment initialization instead.
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(

 49942/50000: episode: 315, duration: 6.368s, episode steps: 316, steps per second:  50, episode reward: 316.000, mean reward: 1.000 [ 1.000,  1.000], mean action: 0.491 [0.000, 1.000],  loss: 6.921296, mae: 47.243710, mean_q: 94.884026

/usr/local/lib/python3.10/dist-packages/gym/core.py:43: DeprecationWarning: WARN: The argument mode in render method is deprecated; use render_mode during environment initialization instead.
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(

done, took 1013.474 seconds
Testing for 5 episodes ...
Episode 1: reward: 500.000, steps: 500
Episode 2: reward: 500.000, steps: 500
Episode 3: reward: 359.000, steps: 359
Episode 4: reward: 500.000, steps: 500
Episode 5: reward: 382.000, steps: 382
Out[7]: <keras.callbacks.History at 0x7ff1d9c66080>
```

7. REFERENCES

1. https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners/
2. https://www.geeksforgeeks.org/introduction-deep-learning/
3. https://www.youtube.com/watch?v=x83WmvbRa2I
4. https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871
5. https://www.youtube.com/watch?v=oydExwuuUCw
6. https://towardsdatascience.com/deep-deterministic-policy-gradient-ddpg-theory-and-implementation-747a3010e82f
7. https://keras.io/examples/rl/ddpg_pendulum