FinSearch MidTerm Report

Using Deep Reinforcement Learning (RL) to Optimise Stock Trading Strategy and thus Maximise Investment Return

Name: Aadit Sule
Roll Number: 23B4204
Team Number : D3

1. Introduction

Deep Q-Network (DQN) is an advanced reinforcement learning (RL) algorithm that combines Q-Learning with deep neural networks to handle environments with high-dimensional state spaces. Developed by researchers at DeepMind, DQN was a breakthrough in the field as it demonstrated the ability to learn complex policies directly from raw sensory input (e.g., pixels from video games).

2. Theory Behind DQN

Q-Learning is a value-based method of RL that aims to learn the optimal action-selection policy by estimating the action-value function, $Q(s, a)$. This function represents the expected return (future rewards) of taking action a in state s and following the optimal policy thereafter.

The Q-Learning update rule is:

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

where:

- s and s' are the current and next states,
- a and a' are the current and next actions,
- r is the reward received after taking action a,
- α is the learning rate,
- γ is the discount factor.

In DQN, a neural network (Q-network) approximates the Q-values, and the network weights are updated based on the Temporal Difference (TD) error:

$$\delta = r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta),$$

where:

- θ are the weights of the Q-network,
- θ^- are the weights of the target network, which are periodically copied from θ to stabilize training.

3. Core Concepts of DQN

- **Experience Replay**: The agent stores its experiences, (s, a, r, s'), in a replay buffer. During training, random samples from this buffer are used to break the correlation between consecutive experiences and stabilize the training process.
- **Target Network**: A separate target network is used to generate the target Q-values. This network's weights (θ^-) are updated less frequently to provide a stable target for the Q-learning updates.
- **Q-Network**: A deep neural network that approximates the Q-value function $Q(s, a; \theta)$. The network is trained using gradient descent to minimize the mean squared error (MSE) between the predicted Q-values and the target Q-values.

4. Algorithm

1. Initialize the replay buffer D.
2. Initialize the Q-network with random weights θ.
3. Initialize the target network with weights $\theta^- = \theta$.
4. For each episode:
   - Initialize the state s.
   - For each time step:
     - With probability ϵ, select a random action a; otherwise, select $a = \arg\max_a Q(s, a; \theta)$.
     - Execute action a and observe reward r and next state s'.
     - Store transition (s, a, r, s') in the replay buffer D.
     - Sample a random mini-batch of transitions from D.
     - Compute the target $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$.
     - Perform a gradient descent step on $(y - Q(s, a; \theta))^2$ with respect to the network parameters θ.
     - Update $s = s'$.
5. Periodically update the target network $\theta^- = \theta$.

5. Advantages of DQN

- **High-Dimensional Input Handling:** DQN can learn policies directly from raw sensory inputs like images, which traditional Q-learning cannot handle efficiently.
- **Stabilized Training:** Experience replay and target networks significantly stabilize the training process, allowing DQN to learn in complex environments.
- **Scalability:** The algorithm scales well to large state spaces due to the use of deep neural networks.

6. Limitations of DQN

- **Sample Inefficiency:** DQN requires a large number of training samples and interactions with the environment, making it computationally expensive.
- **Overestimation Bias:** The max operator in the Q-learning update can lead to overestimation of Q-values, resulting in suboptimal policies.
- **Limited to Discrete Actions:** DQN is designed for discrete action spaces and requires modifications for continuous action spaces.

7. Differences Between DQN and DDPG

Deep Deterministic Policy Gradient (DDPG) is another RL algorithm designed to handle continuous action spaces, unlike DQN. Here are the key differences:

- **Action Space:**
  - DQN: Works with discrete action spaces.
  - DDPG: Handles continuous action spaces using deterministic policy gradients.
- **Algorithm Type:**
  - DQN: Value-based algorithm that estimates the action-value function.
  - DDPG: Actor-Critic algorithm combining value-based and policy-based methods. It uses an actor network to map states to actions and a critic network to evaluate the action-value function.
- **Policy Representation:**
  - DQN: Does not explicitly represent a policy; derives the policy indirectly from the Q-values.

  - DDPG: Uses an actor network to explicitly represent the policy, making it suitable for continuous actions.

- **Exploration Strategy:**
  - DQN: Uses ϵ-greedy exploration.
  - DDPG: Uses noise processes (e.g., Ornstein-Uhlenbeck process) to explore the action space.

8. Conclusion

DQN represents a significant advancement in RL, enabling agents to learn from high-dimensional sensory inputs through deep neural networks. While it has its limitations, such as sample inefficiency and being constrained to discrete actions, its combination of experience replay and target networks has paved the way for more stable and scalable learning. Comparing DQN to DDPG highlights their suitability for different types of action spaces, with DDPG being better suited for continuous actions due to its actor-critic architecture.