Reinforcement Learning: Training Deep Q Networks for Game Environments

Chandan Jyoti Das
ECE dept.
IITG
Guwahati, India
d.chandan@iitg.ac.in

Mir Maiti
ECE dept.
IITG
Guwahati, India
mir.maiti@iitg.ac.in

Pranav Vishal Deshmukh
ECE dept.
IITG
Guwahati, India
pranav.deshmukh@iitg.ac.in

Abstract—**This paper presents the development of a reinforcement learning agent for playing the Cartpole and the Lunar Landing Game using Reinforcement Learning(RL). The agent is trained using Deep Q-Networks (DQN), which efficiently combines Neural Networks with Q-learning. Experimental results show that the agent learns to perform significantly well. It can also achieve scores comparable to human-level expertise. We discuss model architecture, hyperparameter tuning, and the reward system that enables the agent's learning process. Our results demonstrate the viability of Reinforcement Learning approaches in learning gaming strategies.**

- AI in Games
- Reinforcement Learning
- Cartpole, Lunar Landing Game
- Deep Q-Networks

I. Introduction

- **Reinforcement Learning in Games:**
  Reinforcement learning (RL) is a branch of machine learning focused on training agents to make sequential decisions by interacting with an environment to maximize cumulative rewards. This paradigm has found particular success in gaming, robotics, and autonomous systems, where agents learn optimal policies without explicit supervision. Games have proven to be a highly effective benchmark for testing RL algorithms, providing structured yet challenging environments where an agent's ability to learn and adapt can be rigorously evaluated.
- **Role of Deep Q Networks:**
  The development of Deep Q-Networks (DQNs) has significantly advanced reinforcement learning capabilities by combining Q-learning with deep neural networks to approximate optimal action policies. This success has inspired further research using DQNs for tasks requiring strategic decision-making.
- **Cartpole, Lunar Lander:**
  The CartPole environment, a classic reinforcement learning challenge, requires an agent to balance a pole on a moving cart by learning control strategies. Although relatively simple compared to full Atari games, CartPole offers insightful challenges for evaluating the learning performance of RL models, as it requires balancing immediate and delayed rewards, maintaining stability, and optimizing control in a dynamic environment.
  The Lunar Lander problem in OpenAI's Gym is a classic reinforcement learning (RL) task where the objective is to land a lunar module smoothly on a landing pad. This problem is more complex than CartPole and is a good candidate for advanced algorithms like Deep Q-Networks (DQN)
- **Problem Statement and Objectives :**
  This study seeks to develop a reinforcement learning model using DQNs tailored for the CartPole environment, addressing challenges of learning stability and convergence. Our objectives include designing an effective DQN architecture, implementing a training strategy that balances exploration and exploitation, and evaluating the model's performance against baseline methods. This research aims to demonstrate the feasibility and efficiency of DQNs in achieving high performance in the CartPole game.

II. Methodology

- **Problem Statement and Environment Setup:**
  For the Cartpole game, our goal is to balance a pole on a moving cart, whereas in the Lunar Landing game, our task is to land a rover between two poles.
  The agent receives rewards for keeping the pole upright and ends the game if the pole tilts too far or when the cart moves out of bounds or when the rover lands.
- **Defining the DQN Components:**
  1) **Experience Replay Buffer:** The replay buffer stores the agent's experiences (state, action, reward, nextState, done) over episodes. It allows the agent to learn from past experiences and reduces the correlations between consecutive experiences by sampling random mini-batches during training.
  2) **Q-Network:** The Q-network is a neural network that takes the current state as input and outputs a Q-value for each possible action.
  3) **Target Network:** A separate network that is periodically updated to the weights of the Q-network. This target network stabilizes training by reducing oscillations.

4) **Action Selection Policy (Epsilon-Greedy):** This policy helps the agent explore the environment by choosing random actions with a probability of epsilon, which decays over time, eventually leading the agent to exploit its learned knowledge.

- **Implement the DQN Model:**

  1) **Initializing the Environment and Hyperparameters:**
  We have defined hyperparameters, such as learning rate, discount factor, epsilon decay, batch size, and the number of episodes.
  2) **Replay Memory:**
  ReplayMemory provides a mechanism to store the agent's experiences and then efficiently sample random batches of these experiences for training the agent's neural network. This random sampling is crucial in breaking the correlation between consecutive experiences, which helps to stabilize and improve the learning process.
  3) **DQN Class:** The code defines a three-layer fully connected neural network that takes the state of the environment as input and outputs the predicted Q-values for each possible action. This network is the core component of the DQN agent, used to learn the optimal policy for interacting with the environment.
  4) **Implementing the experience Replay buffer:**
  This is often implemented as a deque (double-ended queue) that holds a fixed number of experiences, with older experiences being removed when the buffer is full.
  5) **Defining the DQN training loop:**
  Choosing the epsilon-greedy strategy. After each action, we store the experience tuple (state, action, reward, nextState, done) in the replay buffer.
  6) **Updating the Q-network:**
  We have sampled a mini-batch from the replay buffer, computed the Q-targets, and updated the network.
  7) **Target network update:**
  We have periodically copied the weights from the Q-network to the target network.
  8) **Calculating loss and Backpropagating:**
  For each mini-batch: We have computed the target Q-value: QTarget = reward + gamma * max(QNext) * (1 - done).
  Then we have calculated the loss as the mean squared error between QTarget and QValue.
  Then we used backpropagation to minimize the loss and update the Q-network weights.

- **Hyperparameter Tuning:** We have experimented with different values for learning rate, gamma, epsilon, batch size, and network architecture.
  We have tuned these parameters based on the agent's performance and convergence speed.

III. Experiments and Results

- **Performance Monitoring:**
  We have tracked the performance metrics such as average reward over episodes, the number of steps taken, and epsilon values.
- **Plotting the metrics:**
  We have also plotted these metrics to analyze the improvement over time.
- **Evaluation and Fine Tuning:**
  Once the agent starts achieving a high average score(typically close to 200 in CartPole), we stop the training process and reduce the exploration to let the model exploit its learned policy.
  We have also fine-tuned the network architecture or hyperparameters to improve performance.

IV. Discussions

- Our agent was successful in learning to balance the pole in CartPole and was able to consistently land the lunar module in Lunar Lander with minimal fuel usage.
- One limitation observed was the agent's sensitivity to hyperparameters, especially the learning rate and epsilon decay. Fine-tuning these values was crucial to achieve stable convergence, as inappropriate settings often led to oscillating performance.

V. Conclusions and Future Work

- These results demonstrate the effectiveness of DQN-based approaches in environments with discrete action spaces. The techniques developed here could be extended to robotic control tasks, where balancing stability or safe navigation is critical, or adapted to scenarios where smooth landings are required, such as drone or spacecraft descent.
- Future work could explore the use of prioritized experience replay to focus the agent's learning on transitions with higher impact, potentially accelerating training in environments like Lunar Lander. Additionally, applying PPO or SAC could yield further improvements in the stability and handling of continuous actions, making these methods promising candidates for more complex control tasks.

Acknowledgment

We would like to thank Prof. Teena Sharma for her invaluable guidance and support throughout this research.

We are also grateful for our colleagues at the Indian Institute of Technology, Guwahati for their technical support and encouragement.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, et al., "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," Journal of Artificial Intelligence Research, vol. 47, pp. 253–279, 2013.