DQN with model-based exploration: efficient learning on environments with sparse rewards

Paper Review

The paper [1] introduces a method with the objective of enhancing the effectiveness of learning in environments where rewards are scarce. This is achieved by integrating model-free and model-based approaches. The algorithm proposed, known as DQN with model-based exploration, seeks to expand the exploration of different states and expedite the learning process by utilizing environment dynamics to guide exploration. The authors provide evidence that their algorithm surpasses the original DQN when tested on the Mountain Car environment. However, they acknowledge that its effectiveness is restricted to specific types of environments. Additionally, the authors propose potential extensions and enhancements to adapt their method for a wider range of environments.

The objective of the suggested algorithm, called DQN with model-based exploration, is to enhance the effectiveness of learning in environments where rewards are scarce. The authors observe that conventional model-free reinforcement learning algorithms, like DQN, face challenges in such environments due to the absence of informative feedback. In order to tackle this problem, they put forward an approach that integrates model-free and model-based techniques, enabling improved exploration of the environment and more efficient learning. The authors find the significance of this method by recognizing that numerous real-world challenges exhibit sparse rewards. They contend that their algorithm can be employed in diverse tasks beyond Atari 2600 games and holds promise for generating more effective solutions to real-world problems. By enhancing sample efficiency and exploration capabilities, their method has the potential to decrease the volume of data needed for learning and expedite the advancement of intelligent systems.

The main claims of the paper are:

- The enhanced algorithm, DQN with model-based exploration, merges model-free and model-based techniques to effectively learn environments where rewards are scarce.
- By exploring a broader spectrum of states and accelerating the learning process, the suggested algorithm surpasses the original DQN in the Mountain Car environment.
- The efficacy of the method is confined to specific types of environments, as evidenced by its performance in the Lunar Lander environment.
- The authors propose multiple approaches to expand and enhance their method for a wider range of environments with greater diversity.

Within the scope of improving sample efficiency and exploration capabilities in environments with sparse rewards, these assertions carry notable significance. The authors' algorithm, which integrates model-free and model-based approaches, tackles a fundamental hurdle faced by conventional reinforcement learning algorithms that grapple with such environments due to a lack of informative feedback. The observed enhancements in performance on the Mountain Car environment serve as evidence that their method effectively explores a broader range of states and accelerates the learning process. However, the authors duly recognize the limitations of their method, acknowledging that it may not outperform baseline algorithms in certain types of environments. Overall, their proposed algorithm presents a promising avenue for addressing real-world problems characterized by sparse rewards.

<u>Implementation Details</u>

- I start by importing the necessary libraries and initializing the MountainCar-v0 environment and the Q-network. The Q-network is a neural network with three fully connected layers.
- I define the following hyperparameters which includes their rationale:
  - Learning rate (lr): A lower value allows for precise updates, while a higher value speeds up learning.
  - Discount factor (gamma): Determines the importance of future rewards relative to immediate rewards.
  - Exploration and exploitation (epsilon_start, epsilon_end, epsilon_decay): Control the balance between exploration and exploitation during agent-environment interaction.
  - Batch size (batch_size): Number of experiences sampled from the replay buffer for each training iteration.
  - Target network update frequency (target_update_freq): How often the target network is updated with the Q-network's parameters.
- The choice of values for the DQN hyperparameters is based on empirical experience and aims to strike a balance between stability, convergence speed, and exploration-exploitation trade-off.
- I then set up the optimizer (Adam) and the loss function (MSE) to train the Q-network.
- I define a replay buffer class to store and sample transitions for training the Q-network. It has methods to push transitions into the buffer and sample a mini-batch of transitions.
- I add a `select_action` function that implements an epsilon-greedy policy. It randomly selects an action with probability epsilon or selects the action with the highest Q-value based on the current state.
- Following that, I define a `train_model` function to update the Q-network parameters using a mini-batch of transitions sampled from the replay buffer. It calculates the loss between the predicted Q-values and the target Q-values and performs a gradient update using the optimizer.
- It iterates over a specified number of episodes and performs the training loop. It interacts with the environment, selects actions based on the exploration-exploitation policy, stores the transitions in the replay buffer, and trains the model using the `train_model` function. It also updates the epsilon value and the target network periodically.

➢ After training, I test the trained model by running a few episodes without exploration (epsilon = 0). It calculates the average test reward to evaluate the performance of the trained model.

I, thus, implement the DQN algorithm with model-based exploration using a replay buffer to efficiently learn on environments with sparse rewards which combines deep learning (Q-network) with reinforcement learning techniques to train an agent to perform well in the MountainCar-v0 environment.

Evaluation and Results

```python
# Test the trained model
test_episodes = 10
test_rewards = []

for episode in range(test_episodes):
    state = env.reset()
    episode_reward = 0

    while True:
        action = select_action(state, 0.0)
        next_state, reward, done, _ = env.step(action)

        state = next_state
        episode_reward += reward

        if done:
            break

    test_rewards.append(episode_reward)

average_reward = np.mean(test_rewards)
print(f"Average Test Reward: {average_reward}")
```

In the above evaluation loop, I test the trained model after the training process. It performs the following steps:

- Set the number of test episodes to 10 and initialize an empty list `test_rewards` to store the rewards obtained during testing.
- Iterate over the specified number of test episodes using a for loop.
- Reset the environment to start a new episode and initialize the episode reward to 0.
- Enter a loop to interact with the environment until the episode is done.
- Select an action using the `select_action` function with epsilon set to 0, indicating no exploration.
- Take the selected action in the environment and observe the next state, reward, and whether the episode is done.

- Update the current state and accumulate the reward obtained during the episode.
- Check if the episode is done. If it is, break out of the loop.
- Append the episode reward to the `test_rewards` list.
- Calculate the average test reward by computing the mean of the `test_rewards` list using `np.mean()`.
- Print the average test reward.

The average test reward gives an indication of how well the agent performs in the environment after training. The higher the number of episodes, the more reliable the average reward estimation will be. The resulting average reward of -200 gives us an idea of the agent's performance after training.

An average reward of -200 for the MountainCar-v0 environment indicates that the agent is not able to successfully solve the task within the maximum number of allowed steps for the majority of the evaluation episodes. In the MountainCar-v0 environment, the agent receives a reward of -1 at each time step until it reaches the goal state, at which point it receives a reward of 0. The goal of the agent is to learn to navigate the car up the mountain and reach the flag at the top within the allowed steps. It implies that the agent is not able to reach the goal within the maximum allowed steps across the evaluation episodes.

To improve the performance of our DQN agent in the MountainCar-v0 environment, we can experiment with different learning rates (lr), replay buffer capacities, batch sizes, and discount factors (gamma) or train the agent for a longer duration by increasing the number of training episodes. Additionally, we can also adjust the exploration-exploitation trade-off by tuning the epsilon decay rate (epsilon_decay) or implementing alternative exploration strategies.

References:

[1] S. Z. Gou and Y. Liu, “DQN with model-based exploration: Efficient learning on environments with sparse rewards,” arXiv.org, https://arxiv.org/abs/1903.09295 (accessed Jun. 7, 2023).