Assignment 3

REINFORCEMENT LEARNING

Lomai Mohamed Saadeldin | ID: 20398049 | 7-7-2023

Introduction

This project focuses on training an RL agent to play the "Catch" game using Q-learning. The game involves controlling a paddle to catch falling objects and maximizing the reward (+1 for catching an object, -1 for missing it). The Q-learning algorithm will be enhanced with neural networks (NNs) for function approximation to handle high-dimensional state spaces. A replay buffer will also be implemented to prevent the agent from overfitting to recent experiences and improve its learning efficiency. The performance of the Q-learning agent with and without these enhancements will be compared, and the impact of different hyperparameters on its learning speed and stability will be analyzed.

PROBLEM STATEMENT

The problem statement for this agent is to develop an effective Q-learning algorithm with neural networks and a replay buffer for the game of Catch that can learn an optimal policy for the game and achieve a high return consistently during evaluation. This involves tuning the hyperparameters of the neural network model and the replay buffer and experimenting with different algorithms for prioritized sampling. Additionally, the agent can be further improved by modifying the reward function or the state representation to better capture the game dynamics.

METHODOLOGY

1. Q-Learning agent with Neural Networks:

The _q_func() method computes the Q-value using the neural network model for a given observation and action. The update() method performs the Q-value update using the reward obtained and the maximum Q-value for the next observation, and updates the neural network model using gradient descent. The select_action() method chooses the action to take based on whether a random value is less than epsilon, choosing a random action or the action with the highest Q-value.

Train Q-learning agent with NNs

The first trial:

1. The neural network model has three dense layers with 50, 10, and 1 units respectively, using ReLU activation for the first two layers and no activation for the output layer.
2. The model is compiled with the mean squared error loss function and the Adam optimizer with a learning rate of 0.001.
3. The agent is trained on the Catch environment by running 1000 episodes of the game and updating the policy after each action.

The result:

1. Based on training, the mean return initially fluctuates between positive and negative values, indicating that the agent is still exploring the environment. However, as the training progresses, the mean return gradually increases and eventually stabilizes at around 0.72, indicating that the agent has learned an effective policy for the task.

2. Based on the evaluation, the agent consistently obtains a return of 1.0 for each episode, indicating that it is able to successfully complete the task. The mean return is also 1.0, which indicates that the agent is able to achieve the maximum possible reward for the task.
3. The standard deviation of the returns is 0.0, which indicates that the agent is able to perform the task with a high degree of consistency and reliability.

4. Finally, the agent has learned to perform well in the Catch environment.

The third trial:

(I do the second trial but The agent is not learn the optimal policy.)

1. The neural network model has 4 dense layers with 64, 32, 20, and 1units respectively, using ReLU activation for the first layer and elu activation for the second and third layers and no activation for the output layer.
2. The model is compiled with the mean squared error loss function and the Adam optimizer with a learning rate of 0.001.
3. The agent is trained on the Catch environment by running 1000 episodes of the game and updating the policy after each action.

The result:

1. Based on training, the agent was able to learn and improve its performance over time. At the beginning of training, the agent's mean return was negative, but as training progressed, the agent's mean return improved and became positive.
2. Based on the evaluation, the agent consistently obtains a return of 1.0 for each episode, indicating that it is able to successfully complete the task. The mean return is also 1.0, which indicates that the agent can achieve the maximum possible reward for the task.
3. The standard deviation of the returns is 0.0, which indicates that the agent can perform the task with a high degree of consistency and reliability.
4. Finally, the agent perform very well in the environment.

2. Q-Learning agent with NNs and a Replay Buffer:

The _sample_replay_buffer() method is implemented to sample transitions from the replay buffer for training.

The update() method is implemented to update the Q-values of the neural network based on the sampled transitions.

The qfunc() method is implemented to compute the Q-value as the output of the neural network model.

The select_action() method is implemented to select an action based on the current observation and exploration probability.

Train Q-learning agent with NNs and a Replay Buff

The First Trail :The same model of Trial 1 (Q-learning agent with NNs)

The result:

1. Based on training, the agent's performance started with a mean return of -1.0 and gradually improved over time, with occasional fluctuations. By the end of training, the agent achieved a mean return of -0.4, indicating that it was able to catch some of the falling objects.
2. Based on evaluation, the agent's performance was mixed, with some episodes resulting in a positive return of 1.0 and others resulting in a negative return of -1.0. The mean return over 10 episodes was 0.0 with a standard deviation of 1.0, indicating that the agent's performance was highly variable.
3. Finally, the agent was able to learn and generalize to some extent on the Catch environment.

The Second trial:

- The QLearningNNReplay2, included a flag for prioritized sampling of transitions from the replay buffer. So, the improved performance of the agent in the second trail may be due to the use of prioritized experience replay.
- The secound trail is the best --> Based on training, the agent was able to learn the task successfully, The mean return of the agent increased over time, indicating that the agent was improving its policy. Based on the evaluation, the agent is able to consistently achieve the optimal return of 1.0, indicating that it has learned a good policy.
- The use of prioritized sampling in the updated version of the agent can improve learning speed and performance, especially in scenarios where recent transitions are more informative than older ones.

CONCLUSION

- Q-learning with neural networks can be an effective approach for learning to solve tasks in environments with high-dimensional state spaces.
- The use of a replay buffer can improve the stability and speed of learning.
- Prioritized experience replay can further improve performance by giving more weight to recent transitions.
- The choice of hyperparameters and environment specifics can significantly impact agent performance, and multiple trials may be necessary to find a good policy.
- Using neural networks can be computationally expensive and may require significant computational resources.
- Q-learning with neural networks and replay buffers is a powerful technique that can be used to solve a wide range of tasks, from simple games to more complex real-world problems.
- Careful design and hyperparameter tuning are crucial for achieving good performance in this approach.