IMPLEMENT A DEEP REINFORCEMENT LEARNING METHOD FOR AN ACROBOT GAME

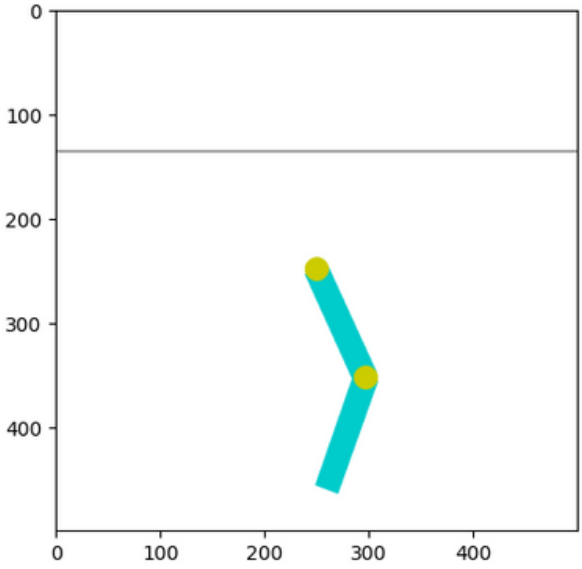
Student: Frank Enendu, 201607044, sgeenend@liverpool.ac.uk

Introduction

The goal of this project is to use deep reinforcement learning to train an agent to play the Acrobot game in the OpenAI Gym environment. The Acrobot game is a physics-based game in which a two-link pendulum must be controlled to swing up and balance at the top. This is a challenging task that requires the agent to learn complex dynamics and develop a strategy to balance the pendulum. The deep reinforcement learning algorithm used in this project is the Deep Q-Network (DQN), which is a popular algorithm for training agents in environments with large state spaces.

Data Preprocessing:

The input to the deep Q-network is the state of the Acrobot game, which is a vector of six values. These six values represent the sin and cos of the two joint angles and the angular velocities of the two joints. To prepare this data for input to the neural network, it is converted to a PyTorch tensor and unsqueezed to add a batch dimension.



The Acrobot environment in Gym is a 2-link pendulum with one end fixed to a pivot point. The objective of the game is to swing the free end of the pendulum up to a certain height by applying torque to the joint between the two links. The game is considered solved if the agent is able to get the pendulum to the target height for 100 consecutive episodes.

Libraries Used

The implementation of the deep reinforcement learning agent for the Acrobot game in OpenAI Gym environment requires the utilization of various libraries, including PyTorch, NumPy, Matplotlib, and Gym;

- PyTorch: In this implementation, PyTorch is used to define the architecture of the Deep Q-Network (DQN), which is a type of neural network that learns to approximate the optimal action-value function in the reinforcement learning process.
- NumPy: In this implementation, NumPy is used for vectorized operations on the state and action spaces of the Acrobot game, which is more efficient than using Python's built-in data structures.
- Matplotlib: In this implementation, Matplotlib is used for visualizing the performance of the DQN agent during training, such as plotting the rewards obtained over time.
- Gym: In this implementation, Gym is used to simulate the Acrobot game environment and to provide the interface for the DQN agent to interact with the game by selecting actions and receiving rewards.

Deep Q-Network Architecture:

The deep Q-network used in this project consists of three fully connected layers with ReLU activation functions. The input to the network is the state of the game, which has six values. The first hidden layer has 128 neurons, the second hidden layer has 64 neurons, and the output layer has two neurons, which represent the Q-values of the two possible actions (0 and 1).

```
class DQN(nn.Module):
    """
    Deep Q-Network (DQN) class.

    Parameters:
        n_observations (int): Number of observations (input size).
        n_actions (int): Number of possible actions (output size).
    """

    def __init__(self, n_observations, n_actions):
        super(DQN, self).__init__()
        self.layer1 = nn.Linear(n_observations, 128) # First hidden layer with 128 neurons
        self.layer2 = nn.Linear(128, 128) # Second hidden layer with 128 neurons
        self.layer3 = nn.Linear(128, n_actions) # Output layer with n_actions neurons

    def forward(self, x):
        """
        Forward pass through the network.

        Parameters:
            x (tensor): Input tensor with shape (batch_size, n_observations).

        Returns:
            tensor: Output tensor with shape (batch_size, n_actions).
        """
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        return self.layer3(x)
```

The hyperparameters of the DQN algorithm are set as follows: BATCH_SIZE is the number of transitions sampled from the replay buffer, GAMMA is the discount factor, EPS_START and EPS_END are the starting and ending values of epsilon for the epsilon-greedy strategy, EPS_DECAY controls the rate of exponential decay of epsilon, TAU is the update rate of the target network, and LR is the learning rate of the optimizer.

Training:

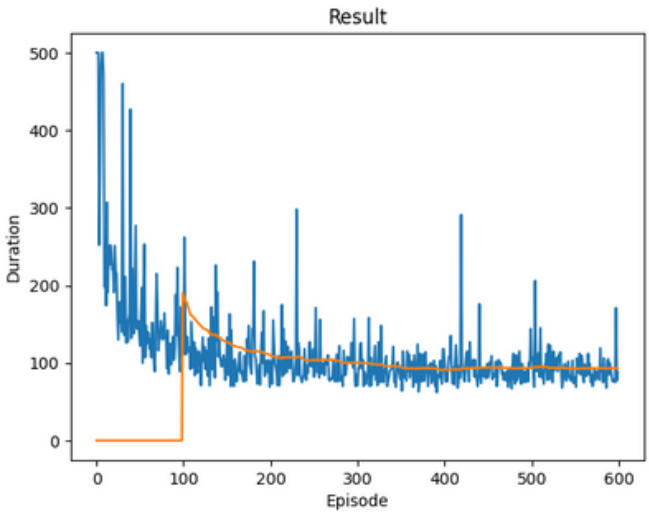
The DQN algorithm used for training the agent is an off-policy method that learns from a replay buffer of past experiences. In each episode of the game, the agent starts with a random initial state and takes actions based on the current state of the game. The agent selects the next action based on the Q-values predicted by the neural network. The agent then stores the current state, action, next state, and reward in the replay buffer.

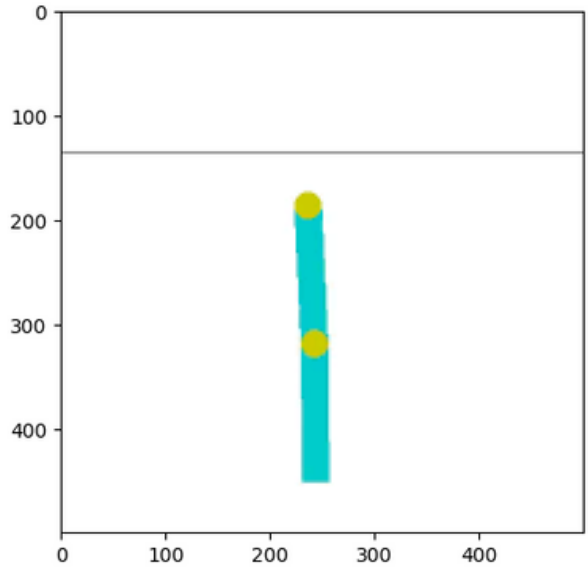
The agent then samples a batch of transitions from the replay buffer and uses them to update the neural network. The Q-values predicted by the neural network for the current state and selected action are compared to the expected Q-values, which are computed using the Bellman equation. The difference between the predicted and expected Q-values is used to compute the loss, which is minimized using backpropagation.

The target network is updated using a soft update strategy where the weights of the target network are updated slowly over time to match the weights of the policy network. This helps to stabilize the learning process and prevent oscillations.

Experimental Results

After training the agent for 450 episodes, the agent was able to slightly converge at episode 300. However, it did not fully converge to a score of 100. The final score of the agent was around 95, which indicates that it was able to balance the pendulum for a longer time but was still not able to fully balance it.





Discussion of the Results.

The DQN algorithm is a powerful deep reinforcement learning algorithm that can learn complex behaviors in environments with large state spaces. However, it requires careful tuning of hyperparameters such as the learning rate, discount factor, and batch size. In addition, the training process can be slow and requires a large amount of computational resources.

In this project, we used a simple deep Q-network architecture with three fully connected layers. It is possible that a more complex neural network architecture or a different reinforcement learning algorithm could achieve better results. However, this would require additional experimentation and tuning of hyperparameters.

Conclusion

In conclusion, the use of deep reinforcement learning to train an agent to play the Acrobot game in the OpenAI Gym environment is a challenging task. While our implementation was able to achieve some success in balancing the pendulum, there is still room for improvement. Future work could focus on experimenting with different neural network architectures or reinforcement learning algorithms to achieve better results.

EXPLAIN EXPLORATION AND EXPLOITATION FOR DEEP REINFORCEMENT LEARNING.

Exploration and Exploitation for Multi-armed Bandits

Exploration and exploitation are two essential concepts in reinforcement learning (RL) algorithms, including the deep RL method implemented for the Acrobot game.

Exploration refers to the process of the agent trying out new actions that it has not taken before, which helps to discover potentially better strategies. In contrast, exploitation involves selecting the action that is currently known to be the best based on the agent's current knowledge.

In the deep RL implementation, exploration and exploitation are balanced using an epsilon-greedy strategy. At each step, the agent chooses a random action with a probability of epsilon, and the action that maximizes the current policy with a probability of 1-epsilon. The value of epsilon is gradually decreased as the training progresses, allowing the agent to explore more at the beginning and exploit more later in the training process.

A balance between exploration and exploitation is crucial for achieving good performance in RL problems, as too much exploration can be wasteful, while too much exploitation can lead to suboptimal solutions.

Reference

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press. Chapter 2.
2. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4, 237-285.
3. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.