Reinforcement Learning

Lucas Kuttner Amin (3190713)

Artificial Intelligence (Msc.)
DLMAIRIL01 - Reinforcement Learning
September 30, 2024

Tutorin: Max Pumperla, Dr.

Contents

1 Introduction

1.1 Oral assignment

Please choose one of the topics listed below to write your assignment on. The starting point for your term paper will be the course book, the contents of which will serve as the basis for an in-depth examination of one of the following questions. You are expected to research and cite from sources corresponding to your chosen topic.

1.2 Task 2: Playing Pong

Pong is a simple computer game imitating table tennis originally developed by Atari in 1972. It is now available as a simulation in OpenAI Gym. Devise a reinforcement learning approach using neural networks to learn how to play pong automatically. Include code examples and results where appropriate.

1.3 Pong Game Overview

Pong is one of the first computer games ever developed, and it was released by Atari in 1972. In this game, two players compete against each other following similar rules to a tennis game. Each player controls a paddle, which aims to bounce a ball by hitting it with this object. When one player fails and the ball gets to the end of the screen, the opposing player scores. [Low09]

In reinforcement learning, it is important to define state space and action space for the game. In pong, we can define the state space by the two paddles: their velocity, ball position, speed, and score. But not all those states are important for a player, for the sake of this project, I have selected just the ball position and the player paddle as relevant, given that no matter what the direction and speed of bounce, the ball speed is constant.

As for action space, a player can move the paddle up, and down and stay still.

2 Reinforcement learning

Reinforcement learning is a machine learning algorithm where an agent learns to interact with the environment to maximize a cumulative reward, given a state space and its possible actions. The agent's goal is to choose the actions that lead to the highest future reward. [ADBB17]

For this project, some key concepts were considered:

- Agent: The decision-maker algorithm that interacts with the environment.
- Environment: The external world that the agent interacts with.
- State: The current set of characteristics states the agent is in the environment.
- Action: All the choices and possible interactions an agent can perform.
- Reward: A numerical value that indicates to the agent how well the action was performed.
- Policy: A function that maps states to actions.

3 Q-Learning

Q-Learning is a reinforcement learning algorithm where the agent learns an action-value function Q(S, A), that represents the expected future reward of taking an action A in a state S. [ADBB17]

The Q-learning update rule is as follows:

Q(s, a) ¡- Q(s, a) + [r + * max_a' Q(s', a') - Q(s, a)]

Where each parameter represents:

- is the learning rate
- r represents the reward
- is the discount factor
- s' is the next state
- a' is the next action

Q-learning algorithms go through a "training" phase where the agent performs random actions under several state conditions, and each action performed is saved in a matrix where each row represents a state and each column represents an action. The value of each cell represents the Q-value for that state-action pair. As the agent interacts with the environment, it updates the Q-values based on the rewards it receives.

After a certain threshold of training, the agent starts using just the q-table to predict the next best action possible.

In the case of simple environments with a small number of actions and states, Q-tables can be efficient. But as the environment increases, the size of the Q-table increases exponentially, making it impractical to store and update. [ADBB17]

3.1 Neural networks in reinforcement learning

Neural networks can be used to approximate the Q-value function without a Q-table, making it a more scalable and flexible approach for approximating the Q-value function.

Deep Q-networks are a popular architecture that combines Q-learning with deep neural networks, which can be needed for very complex environments, or simple neural networks can be used if the state and action space are not complex. [TYG17]

4 Methodology

4.1 Pong game setup

To simulate the pong game, I've developed a simple Python script that goes around Pygame [htt00], a library first released in the year 2000. The code is present in the GitHub project present in Appendix A.

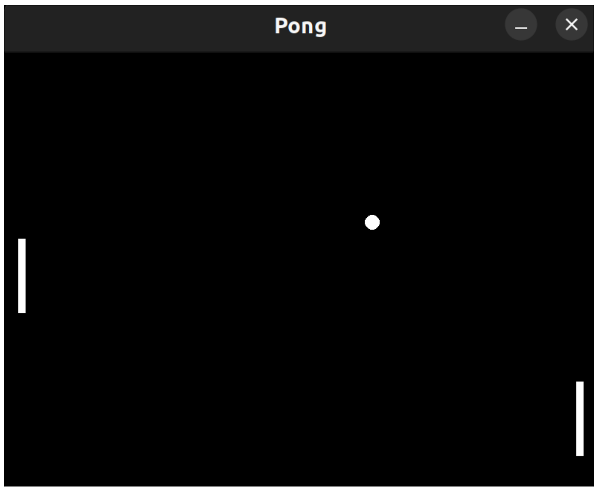

Figure 1: Pong Game

For that, I've created a class called Game with a series of attributes, among them:

- player 1 and 2 paddle coordinates (size, x, y, max_speed, current_speed)
- ball coordinates (Size, x, y, speed_x and speed_y)
- used colors, white (255,255,255) and black (0,0,0)
- screen width and height

With a main loop that contains methods to get the action performed by players and apply them, calculate ball collision, display and score, the whole pong game was built in under 100 lines of code.

This game can be played by two people, using the letters W and S for the left player and up and down arrows for the right player.

4.2 Neural network Q-Learning agent

To teach an agent through reinforcement learning using neural networks, I have developed a class using a simple neural network built on Pytorch around the game logic.

After testing some simple networks, I have achieved the architecture:

```
model = nn.Sequential(
    nn.Linear(state_size, 128),
    nn.ReLU(),
    nn.Linear(128, 64),  # Fully connected layer with 24 units
    nn.ReLU(),  # ReLU activation
    nn.Linear(64, 16),  # Fully connected layer with 24 units
    nn.ReLU(),  # ReLU activation
    nn.Linear(16, action_space_size)
)
```

Built with 1 input layer, 2 internal and an output layer, all fully connected with ReLU activations, with Adam optimizer and learning rate = 0.001.

The input of this network consists of the current state of the game, which I have reduced to both paddle and ball y coordinates, as the speed of the ball is smaller than the paddle, the agent won't need to predict the next ball state, it just needs to follow it vertically. The action space controls the paddle up or down, so it outputs a tuple in format (x, y), and it goes up when X is bigger and down if Y is bigger.

4.3 Model training

The training process follows the logic:

- Generate reward values based on the paddle's position relative to the ball, giving 20 if the ball is close to the paddle or moving towards it and -10 otherwise.
- Convert state before running the game loop and a state after running it to tensors
- Predict the Q-values for current and next states

- Calculate the target Q-value
- Compute loss (using MSE)
- Perform backwards propagation and update the model weights

```
def update(self, state, action, reward, next_state):
    # Convert state and next_state to PyTorch tensors
    state_tensor = torch.from_numpy(state).float().unsqueeze(0)
    next_state_tensor = torch.from_numpy(next_state).float().unsqueeze(0)

    # Predict Q-values for current and next states
    q_values = self.model(state_tensor)
    next_q_values = self.model(next_state_tensor)

    # Calculate the target Q-value
    with torch.no_grad():
        max_next_q_value = next_q_values.max(1)[0].item()
    target_q_value = reward + self.gamma * max_next_q_value

    # Compute the loss (difference between predicted and target Q-value)
    q_value = q_values[0][action]
    loss = self.loss_function(q_value, torch.tensor(target_q_value))
# Use L2 loss or Huber loss

    # Backpropagate the loss and update the DQN model's weights
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

After some training tests, I reached the epsilon value of 50 and epsilon_decay of 0.999, being able to train a decent model on around 700 game steps, playing against the reinforcement learning agent using the W and S keys.

5 Conclusion

This research aimed to develop a reinforcement learning agent capable of playing the game of Pong using a neural network. By employing a Q-learning algorithm and a simple neural network architecture, I was able to successfully train an agent that could effectively compete against a human player.

The agent's performance was significantly influenced by hyperparameter tuning, particularly the learning rate and exploration-exploitation trade-off. Through experimentation, I found that a learning rate of 0.001 and an epsilon value of 50 provided optimal results.

While the agent demonstrated promising capabilities, there are opportunities for further improvement. Future research could explore more complex neural network architectures, such as convolutional neural networks, to handle more intricate game states. Additionally, investigating alternative reinforcement learning algorithms, like Double Q-learning, could enhance the agent's learning efficiency and stability.

In conclusion, this research has successfully demonstrated the potential of reinforcement learning and neural networks for developing agents capable of playing games.

References

[ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[htt00] https://www.pygame.org. Pygame, 2000.

[Low09] Henry Lowood. Videogames in computer space: The complex history of pong. *IEEE Annals of the History of Computing*, 31(3):5–19, 2009.

[TYG17] Fuxiao Tan, Pengfei Yan, and Xinping Guan. Deep reinforcement learning: From q-learning to deep q-learning. In *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part IV 24*, pages 475–483. Springer, 2017.

A Appendix

The code developed for this assignment is present at https://github.com/lucas-amin/IU-Q-Learning-Pong-Project.