CSIS, BITS Pilani K. K. Birla Goa Campus

Artificial Intelligence (CS F407)

Programming Assignment 2
Total Marks: 15
Submission Deadline: 9 PM on 16/11/2024 (Saturday)

Each student must individually do this programming assignment. Your program must be written in Python and should run (without errors) on Python 3.8 or later.
Any form of plagiarism will result in 0 marks being awarded to everyone involved. There will be *no* differentiation between minor and major plagiarism.
Note that the deadline is **9 PM** and not midnight. Five marks per day will be deducted for submissions after the deadline. It will be your responsibility to submit the assignment well in advance and avoid unforeseen problems like power failures etc.

Question 1 (15 marks)

Shallow Q-Network (SQN) for Tic-Tac-Toe

Tic-Tac-Toe is a well-known solved game, meaning that it is possible to create a rule-based agent capable of taking the optimal action for any given board position. However, in this assignment, we will take a different approach by using reinforcement learning techniques.

The goal of this assignment is to introduce you to the fundamentals of Deep Q-Networks (DQN) by implementing a simplified version, which we will refer to as a *Shallow Q-Network* (SQN). While DQNs utilize deep neural networks with multiple layers, such architectures can be challenging for beginners to implement and train effectively. Instead, the SQN—a neural network with fewer layers—provides a more approachable way to explore key reinforcement learning concepts. Through this assignment, you will learn the core ideas behind Q-learning and how neural networks can be used for decision-making in games like Tic-Tac-Toe.

You are provided with Python code for the `TicTacToe` class, which simulates the game environment. Your task is to modify the `YourBITSid.py` file to implement the SQN, train it, and evaluate its performance. Ensure that your code correctly imports and interacts with the `TicTacToe` class, following the expected structure.

To test your implementation, run the following command from the terminal and observe the output:

```
$ python YourBITSid.py 0.5
```

Take time to carefully review the provided code in both `YourBITSid.py` and `TicTacToe.py`. Understanding the existing code will help you modify it effectively and implement the SQN correctly.

1. **Data Collection for Training the SQN**
   - Generate training data by allowing your SQN agent to play multiple games against Player 1 using the ϵ-greedy policy.
   - Implement an experience-replay-buffer mechanism to store experiences of the form $(s, a, r, s', done)$, where:
     - s is the current state.
     - a is the action taken.
     - r is the reward received.
     - s' is the next state.
     - If s' is a terminal state, then $done = True$, else $done = False$.
   - Store the most recent experiences (experience-replay-buffer) using the `deque` data structure (as shown in the *NN_training.py* file).
   - Use randomly selected mini-batches from the replay buffer to train the SQN. Use the trained (improved) SQN to collect data for the next set of episodes and so on.
   - You may have to train PlayerSQN using data from thousands of episodes before you see any improvement in its performance.
   - For the initial data collection and training, let probability *smartMovePlayer1* be 0. Increase the probability gradually as the performance of PlayerSQN improves.
2. **Step-by-Step Procedure for experience data collection**
   (a) **Initialize** the neural network with random weights.
   (b) **For** each episode:
      - Reset the game environment to the initial state.
      - While the game is not over:
        - Observe the current state s.
        - Select an action a using the epsilon-greedy policy based on the Q-values predicted by SQN. (Ignore the invalid actions when selecting an action a.)
        - Execute action a and observe the reward r and the next state s'.
        - Store the experience 5-tupe $(s, a, r, s', done)$ in the experience-replay-buffer memory. The difference between $Q(s, a)$ value predicted by the model and $Q_{\text{target}}(s, a)$ is the error in prediction. This error must be minimized during the SQN training process.
      - Decrease epsilon to reduce the exploration rate after PlayerSQN starts performing well.
3. **Note on Calculating Target Q-Values:**
   - The target Q-value for a given action can be calculated using the Bellman equation:
     $$Q_{\text{target}}(s, a) = r + \gamma \max_{a'} Q(s', a')$$
     where:

  - r is the reward received after taking action a in state s.
  - γ is the discount factor (choose a value between 0 and 1, e.g., 0.95).
  - s' is the next state.
  - $Q(s', a')$ are the predicted Q-values for all possible actions in the next state.
  - Important : The action a' selected for finding $Q_{\text{target}}(s, a)$ must be a valid action that can be taken from state s'.
- The Q_{target} value is found when we train the SQN. The Q_{target} value is not stored in the replay buffer because we want to use the most up-to-date Q_{target} value; This will help us correctly determine the error in $Q(s, a)$ value predicted by SQN.

4. **Avoiding Instability while Training**

   Do not train the network while you are collecting experiential data because it can lead to instability issues. To avoid instability issues, use mini-batches of (5-tuple) experiential data. These mini-batches are randomly sampled from the experience-replay-buffer. (Read more about instability issues while training Q-networks and how to avoid them.)

5. **Guidance on Using Keras for training SQN**

   - Install Keras and TensorFlow if not already installed:

```
pip install keras tensorflow
```

   - Use the `Sequential` model from Keras to build your network:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Dense(64, input_dim=9, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(9, activation='linear'))
model.compile(loss='mse', optimizer=Adam(learning_rate=0.001))
```

   - The input dimension is 9, corresponding to the 9 positions on the TicTacToe board.
   - Use Rectified Linear Unit (ReLU) activation functions for hidden layers and a linear activation for the output layer.
   - The output layer has 9 units, each representing the Q-value of taking an action ($Q(s, a)$). (Allow the model to predict Q-values for invalid actions. The invalid actions will be ignored by PlayerSQN when selecting an action using ϵ-greedy action selection.)
   - The state vector currently has 0, 1 and 2 which makes it more human readable. (1 and 2 correspond to Player 1 and 2 respectively.) However, it would be

better if the input vector to SQN contains -1, 0 and 1. This could help in faster learning of patterns in the states.

6. **Evaluation of the SQN**
   - After training, evaluate your SQN by playing a series of games against Player 1 with `smartMovePlayer1` set to 0, 0.5 and 1.
   - Record the number of wins, losses, and draws over at least 20 games.
   - Plot graphs to show how the performance of your SQN improves over time (e.g., win rate vs. number of training episodes).
7. **Submission Requirements**
   - You must submit three files. (See Instructions for submissions for more details.)
   - **Code:** Submit your modified Python code including the implementation of the SQN.
   - **Trained Model:** Submit your trained SQN model saved in an appropriate format (e.g., `.h5` file using Keras's `model.save()` method).
   - **Report:** Submit a report in PDF format containing:
     - A description of your implementation and any design choices.
     - Graphs showing the performance improvement over time.
     - Any observations, challenges faced, and how you addressed them.

Instructions for Submission

- **Code Submission**
  - Name your main Python file as `YourBITSid.py`. (e.g. `2020A7PS0001G.py`) This file contains the PlayerSQN class that you need to modify. Feel free to add additional classes and functions in `YourBITSid.py` file.
  - **No need** to submit the `TicTacToe.py` file; You must submit the `YourBITSid.py` file.
- **Model Submission**
  - Save your trained model using Keras's `model.save('YourBITSid_MODEL.h5')`. (e.g. `2020A7PS0001G_MODEL.h5`)
  - Include the `YourBITSid_MODEL.h5` file in your submission.
- **Report Submission**
  - The report should be a PDF document named `YourBITSid.pdf`. (e.g. `2020A7PS0001G.pdf`)
  - Include the following sections in your report:
    1. **Introduction**: Brief overview of the problem and your approach.
    2. **Methodology**: Detailed explanation of your SQN implementation and training process.

3. **Results**: Present your findings with graphs and charts.
4. **Discussion**: Analyze the results and discuss any challenges or interesting observations.
5. **Conclusion**: Summarize your work and suggest possible improvements.

- **Evaluation Criteria**
  - 7 marks for report. Marks will be proportional to the effort you have put in.
  - 4 marks will be awarded after we run your code with `smartMovePlayer1=0`. (i.e., Player 1 plays randomly.) Marks will be based on the outcome of three episodes:
    - 4 marks if PlayerSQN gets total reward of at least 1 (e.g. 2 wins and 1 loss).
    - 2.5 marks if PlayerSQN gets total reward of 0 (e.g. 3 draws).
    - 1.5 marks if PlayerSQN gets total reward less than 0 (e.g. 1 win and 2 losses)
    - 0 mark if your program crashes or goes into an infinite loop.
  - 4 marks will be awarded after we run your code with `smartMovePlayer1=0.8`. (i.e., Player 1 plays smartly with probability 0.8.) Marks will be based on the outcome of three episodes:
    - 4 marks if PlayerSQN gets total reward of at least 1 (e.g. 2 wins and 1 loss).
    - 2.5 marks if PlayerSQN gets total reward of 0 (e.g. 3 draws).
    - 1.5 marks if PlayerSQN gets total reward less than 0 (e.g. 1 win and 2 losses)
    - 0 mark if your program crashes or goes into an infinite loop.
- **Submission Process**
  - Submit the three submission files (code, model and report) separately. **Don't** zip the files.
- **Late Submission Policy**
  - Late submissions will incur a penalty of 5 marks per day.

Additional Notes

- Start early to ensure you have ample time to train your SQN, as training may take several hours.
- Feel free to experiment with different network architectures and hyperparameters to improve performance.
- If you encounter any issues, seek assistance well before the deadline.

Policy on LLM Usage

Students can use Large Language Models (LLMs) for learning purposes. However, students cannot copy and paste program code from LLMs into their assignment file. Students can get suggestion from LLMs to improve their reports. However, students cannot copy and paste any text from LLMs into their report. **A** more detailed policy will be communicated shortly.

Acknowledgment

This assignment on Shallow Q-Networks was conceptualized by the Instructor-in-Charge (IC). Assistance from ChatGPT was utilized in composing parts of the assignment description and the boilerplate code. Students are required to adhere to the policies regarding the use of LLMs, which will be communicated separately.