Homework-2: Reinforcement Learning

-Soham Barman

Reference- This Piece of Code is taken from this repository
https://github.com/dennybritz/reinforcement-learning/blob/master/DQN/Double%20DQN%20Solution.ipynb

1) Writing an abstract that provides a high-level overview of the code's purpose and the overall process it follows.

This Piece of code provides a DQN agent learning to play the Atari Breakout environment. To do so, this relies on state-of-the-art reinforcement learning-a combination of double Q-learning with experience replay-which builds an agent able to extract the best game strategies from pixel-level data. Core components include the three building blocks of the implementation: a state processor responsible for preprocessing raw RGB frames from Atari into grayscale images, a Q-value estimating neural network predictive of values of actions, and a target network architecture responsible for learning stabilization. In a word, the major learning process iterates over the following: processing raw game frames into 84×84 grayscale images, storing an agent replay memory of recently experienced game transitions, sampling mini-batches from the replay memory to train, doing the update by gradient descent, and updating the target network periodically as synchronized with the main Q-network.

2) Identifying the core section (such as a couple of functions about the RL implementations) of the reinforcement learning implementation and add comments to each line, explaining what it is doing. This demonstrates your step-by-step understanding.

I would like to focus on these two components of the code, which in my opinion are the critical implementations in the code, They are:

- Epsilon-Greedy Policy Implementation

```
def make_epsilon_greedy_policy(estimator, nA):
    def policy_fn(sess, observation, epsilon):
        A = np.ones(nA, dtype=float) * epsilon / nA
        q_values = estimator.predict(sess, np.expand_dims(observation,
0))[0]
        best_action = np.argmax(q_values)
        A[best_action] += (1.0 - epsilon)
        return A
    return policy_fn
```

This implementation manages the crucial exploration-exploitation dilemma in reinforcement learning. This function creates a dynamic policy that intelligently balances between random exploration and exploitation of learned knowledge. This is achieved through a sophisticated probability distribution system where each action initially has an equal chance of selection during exploration, while the Q-network predicts action values for the current state, ultimately increasing the probability of selecting the believed best action by a factor of (1 - epsilon). This mechanism can be likened to a basketball player who has developed a reliable jump shot (exploitation) but

occasionally experiments with new moves (exploration), with the frequency of experimentation decreasing as expertise grows.

- Core Learning Loop Implementation

```
samples = random.sample(replay_memory, batch_size)
states_batch, action_batch, reward_batch, next_states_batch, done_batch =
map(np.array, zip(*samples))

q_values_next = q_estimator.predict(sess, next_states_batch)
best_actions = np.argmax(q_values_next, axis=1)
q_values_next_target = target_estimator.predict(sess, next_states_batch)
targets_batch = reward_batch + np.invert(done_batch).astype(np.float32) * \
    discount_factor * q_values_next_target[np.arange(batch_size),
best_actions]
```

The core learning process implements Double Q-Learning, an advanced variant designed to address the overestimation bias inherent in standard Q-learning. This implementation has been rather important for three key components: the Experience Replay Mechanism, the Double Q-Learning Architecture, and the Temporal Difference Target Computation. The Experience Replay Mechanism randomly samples transitions from memory, which breaks the temporal correlations in the training data and thus enables multiple learning opportunities from each experience. The Double Q-learning Architecture makes use of two independent networks for selecting actions and a target network for value estimation that bypasses the self-reinforcing process of generating overoptimistic estimates of values. The Bellman equation implements the value iteration using Temporal Difference Target Calculation in which there is the incorporation of discount factors that balance immediate and future rewards, handling the terminal states through a 'done' flag.

In Atari Breakout, such a setting would enable the agent to learn strategies from simple to complex gameplays. The agent is making random moves at the very beginning due to a high value of epsilon. With that, it builds a diverse memory of experiences while beginning to see basic relations between actions and rewards. In the intermediate learning phase, the agent finds useful patterns in the actions taken and learns simple strategies, such as keeping the paddle under the ball. During his advanced learning stage, he forms intricate strategies toward effective brick-blasting: he learns to aim for the ball at more specific targets and copes with different ball speeds and angles. In my opinion the above learning progression resembles how a chess player would reflect about his or her games: experience replay serves as a record of past positions and moves, Double Q-learning is having two chess coaches where one suggests the moves and the other reviews them; the temporal difference calculation considers both short-term gains and long-term strategies.

The combined implementation of epsilon-greedy exploration and double Q-learning creates a robust learning system capable of mastering complex tasks through trial and error. The issues were those related to learning from detailed visual information, delayed rewards, exploration-exploitation dilemmas, and avoiding common mistakes in deep reinforcement learning. This sophisticated architecture enables the agent to evolve from random paddle movements to skilled gameplay through purely experiential learning, similar to human skill development but with the advantage of learning from thousands of games' worth of experience in a relatively short time. It emphasizes the power of using careful exploration methods in combination with advanced learning techniques in reinforcement learning systems.