Reinforcement Learning-Based Pong Game from Atari with Deep Q-Network Agent : A Detailed Report

1. Introduction

This RL project is based on the interative arcade game called Pong created by Allan Alcorn, an engineer at Atari,Inc, an American Video game developer company in 1972. Here, the game is played and learned by an AI agent through Reinforcement Learning techniques. Using the Deep Q-Network(DQN) algorithm, the agent gets trained to control one of the paddles and compete against a human opponent. The agent learns over time progressively, interacting with the environment and updates its policy using the experience it gains from training/playing.

Here, a feature is also added for the agent to learn from an human opponent while playing against them in real time. Thus, enhancing the ability of the agent to play through dynamic and continuous learning.

2. What is Pong game?

Pong is a classic 2D arcade game in which two paddles move vertically on either side of the screen to hit a ball back back and forth continuously. The goal is to score points by sending the ball past the other paddle that is controlled by another opponent. The game mechanics include:

- **Ball Movement**: The ball moves with a constant velocity and bounces off the paddles and the top and bottom walls.
- **Paddle Movement**: Paddle moves with a constant velocity and vertically to intercept the ball
- **Scoring Points**: A point is scored if the any of the players fail to hit the ball and let it pass through the paddle boundaries

3. Reinforcement Learning (RL) in Pong

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment and receiving rewards or penalties based on the outcomes of these actions. In our Pong game:

- the agent's goal is to maximise the cumulative reward by learning how to move the paddle to successfully hit the ball and prevent the opponent from scoring the point.
- The only actions here to be performed is the paddle movements i.e up and down
- Rewards are given bases Rewards are given based on whether the agent hits the ball (+1), loses a point (-10), or scores (+10).

Reinforcement Learning General Framework

- **Agent**: An entity that learns and makes decisions based on the observed environment and the rewards it receives.
- **State (S)**: The current state of the environment, which is represented by the ball's position,its velocity, and the paddle position.
- **Action (A)**: The set of actions (here it is only two actions) available to the agent.
- **Reward (R)**: The feedback recieved from the environment after taking an action from the action pool based on the circumstance/state of the environment. Positive rewards are for success(ex, hitting the ball), negative rewards are for losing a point or letting the ball go.
- **Policy (π)**: The policy is a strategy that will be used by the agent to follow a set of actions based on the current state to maximise rewards.
- **Value Function(V)**: The expected cumulative reward that the agent will receive, starting from a given state and following the policy.

The Pong game environment for this project is modelled as a Markov Decision Process(MDP) where:

- The next state depends on the current state and action taken which is the Markov Property
- The agent learns to approximate the **Q-value function** using *Neural Networks*.

4. Deep Q-Network (DQN) for Pong Game

This project uses Deep Q-Network(DQN) algorithm to train the AI Agent.

4.1. Q-Learning

Q-learning is a form of model-free reinforcement learning where the agent learns a Q-function $Q(s, a)$, which estimates the value (expected future rewards) of taking action a in state s. The update rule for Q-learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Where:

- α is the learning rate.
- r is the reward obtained from the environment.
- γ is the discount factor, which determines the importance of future rewards.
- s' is the new state after taking action a.

4.2. Deep Q-Network (DQN)

DQN extends Q-learning by using a neural network to approximate the Q-function $Q(s, a)$. The agent interacts with the environment, collects experiences, and uses these experiences to train a neural network to predict the Q-values.

Network Architecture for DQN

The neural network in DQN has the given architecture:

- **Input Layer**: The state of the Pong environment (ball position, velocity, paddle positions).
- **Hidden Layers**: Several fully connected layers that process the input to extract useful features.
- **Output Layer**: Two output values, each representing the Q-value of one possible action (move up or move down).

4.3. Experience Replay

One crucial feature of DQN is **experience replay** where instead of updating the Q-values after each action, the agent stores the experiences it gains (s, a, r, s') in a memory buffer which it periodically samples as a batch of experiences to update the neural network. This helps break the correlation between consecutive experiences and stabilizes training.

4.4. Target Network

DQN also uses a **target network** to stabilize the Q-value updates. This target network is a copy of the Q-network, but its weights are updated less frequently which reduces oscillations and divergence during training to maintain a consistent agent and its plays.

4.5. Epsilon-Greedy Policy (Exploration vs Explotation condition balance)

To balance exploration and exploitation, the DQN agent follows an **epsilon-greedy policy** where it explores by taking random actions with probability ϵ, and exploits its current knowledge (choosing the action with the highest Q-value) with probability $1 - \epsilon$. Over time, ϵ decays to reduce exploration as the agent becomes more confident in its policy.

5. Tech Stack

- **Turtle Graphics**: Used for rendering the Pong game environment, including paddles, ball movement, and the game screen.
- **PyTorch**: To implement the DQN model and for building and training the neural network that approximates the Q-function.
- **NumPy**: For numerical operations and handling state representations.
- **Collections (Deque)**: For managing experience replay by storing past experiences.

6. Implementation Procedure

6.1. Game Setup

The game environment is created and setup using Python's Turtle module. The screen is initialized, and the paddles and ball are rendered. The game mechanics are coded, including ball bouncing, scoring, and paddle control.

6.2. Agent Training

- The DQN agent interacts with the Pong environment, where the state is represented by a vector of six values (ball position, ball velocity, left paddle position, right paddle position).
- The agent receives rewards based on performance of its actions and the impacts it has on the environment (e.g., hitting the ball or scoring points).
- Over the iteration of episodes, the agent's Q-network is updated using the Q-learning update rule, where the expected future reward is approximated through a neural network.

6.3. Playing Against the Agent

After training the agent, the model is saved and can be loaded for gameplay. The human controls the right paddle using keyboard inputs, while the trained agent controls the left paddle. The agent continues to learn during the game using experience replay, improving its performance as it plays against the human.

Primary Equations Used

Q-Learning Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Where:

- $Q(s,a)$: Current Q-value for state s and action a.
- r: Immediate reward from the environment.
- γ: Discount factor, controlling the weight of future rewards.
- α: Learning rate.

DQN Loss Function:

The goal is to minimize the difference between the predicted Q-values (from the neural network) and the target Q-values (which represent the "true" value the agent should have learned for a particular state-action pair). This difference is captured using the Mean Squared Error (MSE). The loss function used to update the neural network is the mean squared error (MSE) between the predicted Q-values and the target Q-values:

$$L = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)^2 \right]$$

- L: This is the loss, specifically the mean squared error between the predicted Q-values and the target Q-values.
- $\mathbb{E}$: Represents the expected value, or the average over a batch of experiences sampled from the replay memory.
- r: The **immediate reward** received after taking action a in state s. This is the short-term feedback the agent gets from the environment after its action.

- γ: The **discount factor**, which determines the importance of future rewards. A value between 0 and 1, where:
  - $\gamma = 0$ means the agent only cares about immediate rewards.
  - γ close to 1 means the agent values future rewards more, encouraging it to plan ahead.
- $\max_{a'} Q(s', a')$: The maximum predicted Q-value for the **next state** s' across all possible next actions a'. This represents the agent's estimate of the best future reward it can obtain from the next state.
- $Q(s, a)$: The **predicted Q-value** for taking action a in the current state s.
- $r + \gamma \max_{a'} Q(s', a')$: This term represents the **target Q-value**. It's the immediate reward r plus the discounted maximum future reward $\max_{a'} Q(s', a')$. This is the "true" Q-value that the neural network is trying to approximate.

Challenges Encountered and Steps taken

1. **Exploration-Exploitation Tradeoff**: A balance between exploring new actions and exploiting known strategies had to be achieved by using an epsilon-greedy policy.
2. **Training Stability**: Using experience replay and a target network was necessary to stabilize the learning process.
3. **Human-AI Interaction**: Had to ensure a smooth gameplay between the human and AI agent required balancing the game's speed, AI responsiveness, and the agent's continuous learning during gameplay.

Concept of Prioritized Experience Replay (PER)

Prioritized Experience Replay (PER) is an improvement to the standard experience replay mechanism used in DQN which aims to improve learning rate of the agent efficiently. In the standard experience replay system, the agent stores the experiences (state, action, reward, next state, done) into a buffer and samples random mini-batches from this buffer to train the neural network. But, this random sampling treats all experiences equally and its not optimal for our case. Priority-based sampling mechanism increases the learning efficiency by considering only the most "important" experiences, which is where the agent has learnt the most. This is improved using the **Temporal Difference (TD) error**, which measures the difference between the predicted Q-value and the target Q-value. Experiences with a higher TD-error indicates that the agent's prediction is high and it has to learn from experiences more to reduce this error.

Key Concepts:

1. **Temporal Difference (TD) Error**:
   - TD error is the difference between the Q-value predicted by the Q-network and the target Q-value. It reflects how much the agent's prediction deviates from the actual outcome.
   - TD error = |Q(s, a) - (r + γ * max Q(s', a'))|

- Larger TD errors imply that the agent made a larger prediction error, signaling that these experiences are more valuable for learning.

2. **Prioritized Sampling**:

   - Instead of sampling experiences randomly, PER assigns a priority to each experience based on its TD error. Experiences with larger TD errors are more likely to be sampled for training because they offer a higher potential for reducing the agent's prediction error.

   - The probability of sampling an experience i is proportional to its priority p_i:

     $$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

     where α controls the degree of prioritization (when α = 0, prioritization follows uniform random sampling).

Conclusion

The project successfully completed and demonstrates the strength of Reinforcement Learning, especially with Deep Q-Network algorithm, that is applied to train the AI Agent to play Pong game. The agent learnt overtime how to play by interacting with the environment and improved its performance using the Q-Learning framework, neural networks, and experience replay.