Deep Reinforcement Learning on Atari Breakout

SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)
(Established under Section 3 of the UGC Act, 1956)
Re-accredited by NAAC with 'A++' Grade | Awarded Category – I by UGC
Founder: Prof. Dr. S. B. Mujumdar, M.Sc., Ph.D. (Awarded Padma Bhushan and Padma Shri by President of India)

Kaustubh Raykar - 21070126048
Srinivas Kota - 21070126050

Description of the problem

Atari Breakout is a classic arcade game where the player controls a paddle to bounce a ball and break bricks at the top of the screen. The goal is to break as many bricks as possible while preventing the ball from falling below the paddle.

Description of the environment and agent:

- **States:** The environment provides the agent with a state consisting of a stack of four grayscale frames representing the game screen.
- **Actions:** The agent can take one of four actions: move the paddle left, move the paddle right, do nothing, or fire the ball.
- **Objective:** The objective is to maximize the cumulative reward, which is achieved by breaking bricks and preventing the ball from falling.
- **Model:** This is a model-free approach, as the agent learns directly from interaction with the environment without explicitly modeling it.
- **Discount Factor:** The discount factor, gamma, is set to 0.99, determining the importance of future rewards.
- The output layer has four neurons, representing the Q-values for each possible action.
- The network takes the stack of 4 frames as input and outputs the predicted Qvalues for each action.

MDP formulation:

States (S)
The state space in Atari Breakout consists of the positions and velocities of the ball, the position of the paddle, and the configuration of the remaining bricks. Due to the complexity and high dimensionality of directly using raw pixel data, states are represented by stacks of the last four frames from the game. This representation helps capture the dynamics such as the motion of the ball and paddle, which are critical for predicting future positions and making decisions.

Actions (A)
The action space is discrete and consists of the following actions available to the agent at each time step:
NOOP (No Operation): The paddle does not move.

FIRE: Used to launch the ball at the start of the game or after losing a life.
RIGHT: Moves the paddle to the right.
LEFT: Moves the paddle to the left.

Rewards (R)
The reward function is straightforward:
+1 point for each brick broken by the ball.
0 points for all other actions, including moving the paddle or missing the ball.
The agent's objective is to maximize the total score, which corresponds to breaking all the bricks as efficiently as possible without losing the ball.

Transitions (T)
The transition dynamics describe how the current state and action taken by the agent influence the next state. In Atari Breakout, these dynamics are governed by the game's physics:

The ball’s movement follows a deterministic path unless it collides with the paddle, a brick, or the walls.

The outcome of taking an action (moving the paddle left or right) is also deterministic in how it changes the paddle's position.

Discount Factor (γ)
The discount factor, gamma (γ), is typically set close to 1 (e.g., 0.99 in this implementation). This high value emphasizes the importance of future rewards, which encourages strategies that not only score immediately but also preserve the possibility of scoring in the future, such as strategically positioning the paddle to handle subsequent ball bounces effectively.

Policy (π)
The policy π is a strategy that the agent follows to choose actions based on the current state. In the case of the implemented DQN, the policy is derived from the Q-values predicted by the network, where the agent typically selects the action with the highest Q-value (exploitation), but with a probability ε, a random action is chosen (exploration). This ε-greedy strategy facilitates both exploration of the environment and exploitation of the learned values.

Method Description:
The RL agent is trained using a DQN with the following specifics:

- **Exploration vs. Exploitation:** An ε-greedy strategy is used, where ε is initially set to 1 (pure exploration) and decays to a minimum of 0.1, balancing exploration with exploitation over 1,000,000 frames.
- **Learning Algorithm:** The Adam optimizer is used for training the DQN with a learning rate of 0.00025 and a clipnorm of 1.0. The network updates its weights using backpropagation from the loss computed via the Huber loss function.
- **Experience Replay:** Implemented to prevent correlations between consecutive updates. The buffer stores up to 100,000 past experiences, and training samples are drawn randomly from this buffer.
- **Target Network:** To stabilize learning, a target network's weights are updated every 10,000 steps, providing consistent targets during temporal difference updates.

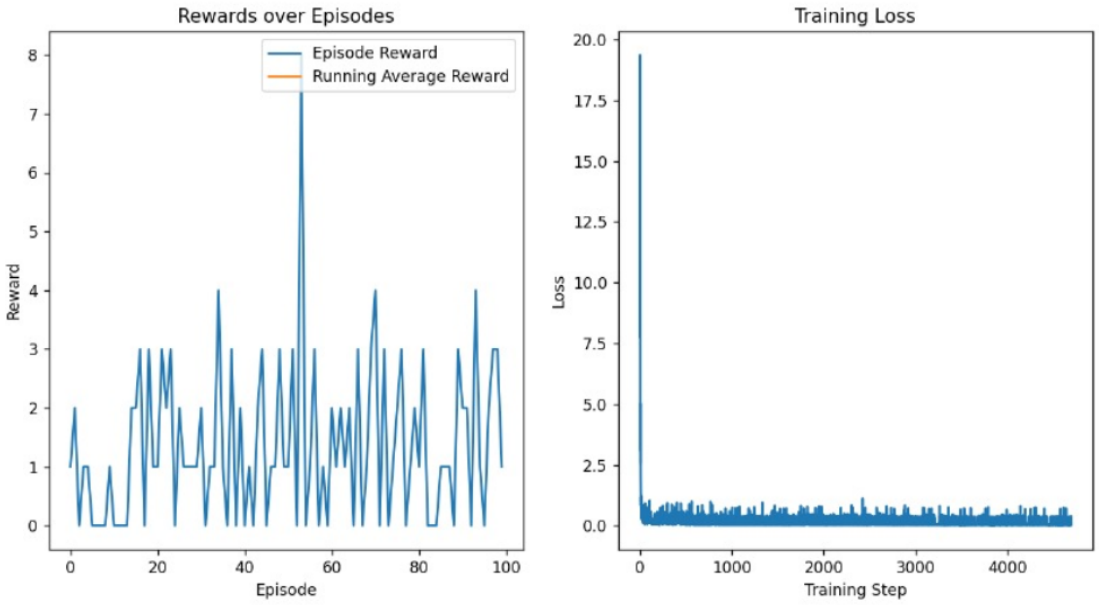
The Deep Q-Network (DQN) algorithm is employed to approximate the Q-values, mapping states to action values. The flowchart of the method is as follows:

1. Initialize the environment and the Q-network.
2. Play episodes of the game, collecting experiences.
3. Store experiences in a replay buffer.
4. Sample batches from the replay buffer for training.
5. Update the Q-network parameters using gradient descent.
6. Periodically update the target Q-network.
7. Repeat steps 2-6 until convergence or a maximum number of episodes is reached.

Results:
The script aims to achieve a running reward of 40 or higher, which is considered solving the Breakout game. The running reward is calculated as the mean of the last 100 episode rewards. The script logs the running reward and episode count periodically to monitor the training progress

The agent learns to play Atari Breakout effectively, achieving a running reward of over 40 within a specified number of episodes. This indicates successful learning and mastery of the game environment.

Graphs:



THE GAME :

Conclusion:

The Deep Q-Network algorithm demonstrates its effectiveness in learning to play Atari Breakout. By iteratively interacting with the environment and updating its Q-values, the agent gradually improves its performance, ultimately achieving a high level of gameplay. This approach showcases the power of reinforcement learning in mastering complex tasks in diverse environments.