Project 3: Deep Q-learning Network (DQN)

Reinforcement Learning CS 696

Kathirvel Manivannan Vimaladevi – 131140409

Introduction

This project develops a Deep Q-Learning Network (DQN) to play the Atari Breakout game. The objective is to achieve an average reward exceeding 40 points over 100 episodes, each containing 5 lives. The implementation utilizes PyTorch, OpenAI's Gymnasium, and the Atari wrapper, incorporating improvements such as the dueling architecture and prioritized experience replay.

Model Architecture

The DQN adopts a dueling architecture, where convolutional layers are shared, and separate streams are used to compute value and advantage. This design enables the agent to better differentiate between actions that are crucial for certain situations and those that are less relevant to the current state.

Layers:

Convolutional:

- Conv1: 8*8 kernel, stride of 4, with 32 filters.
- Conv2: 4* 4 kernel, stride of 2, with64 filters.
- Conv3: 3 *3 kernel, stride of 1, with 64 filters.

Fully Connected:

- FC1: 512 units.
- Value stream: Produces a scalar representing the state value.
- Advantage stream: Outputs Q-values corresponding to each action.

Hyperparameters

The replay buffer is set with a maximum size of 100,000 and a minimum size of 40,000. The epsilon decay rate is 0.995, starting with a maximum epsilon value of 1.0 and decaying down to a minimum of 0.01. These parameters are designed to control the exploration-exploitation trade-off during the training process.

Tuning Hyperparameters

Learning Rate:
I experimented with learning rates of 1×10−31 \times 10^{-3}1×10−3, 5×10−45 \times 10^{-4}5×10−4, and 5×10−55 \times 10^{-5}5×10−5. The learning rate of 5×10−55 \times 10^{-5}5×10−5 provided the best balance between convergence speed and stability. Higher rates, such as 1×10−31 \times 10^{-3}1×10−3, led to unstable training with oscillating rewards, while very low rates, like 1×10−61 \times 10^{-6}1×10−6, caused learning to progress too slowly.

Batch Size Selection:
I tested batch sizes of 16, 32, and 64. A batch size of 16 offered the optimal trade-off between performance and memory constraints. Larger batch sizes (64/128) initially increased rewards but slowed down convergence due to higher memory usage.

Replay Buffer Configuration:
The replay buffer was set with a maximum size of 100,000 experiences and a minimum of 40,000 before training began. I used prioritized experience replay, with an alpha of 0.6, and beta annealed from 0.4 to 1.0.

Exploration Strategy:
I applied epsilon decay from 1.0 to 0.01, with a decay rate of 0.995 per episode. This strategy ensured a smooth transition from exploration to exploitation throughout the episodes. Faster decay rates (e.g., 0.99) caused premature convergence to suboptimal policies, while slower rates (e.g., 0.999) delayed exploitation.

Soft Target Update Mechanism:
I used a tau of 5×10−55 \times 10^{-5}5×10−5 for soft updates to the target network, which enabled stable convergence without abrupt shifts in Q-values. Larger tau values (e.g., 1×10−31 \times 10^{-3}1×10−3) made training unstable, while smaller values (e.g., 1×10−61 \times 10^{-6}1×10−6) slowed the updates to the point where recent experiences were not effectively incorporated.

Optimizer

I used the Adam optimizer with a learning rate of 5×10−55 \times 10^{-5}5×10−5, and its adaptive learning rate adjustments contributed to stable convergence during training. While experimenting with RMSProp, I found that it led to slower convergence. Throughout the experiments, Adam consistently provided the most stable and efficient training.

Loss Function

I used Weighted Mean Squared Error (MSE) as the loss function, where the squared TD errors were weighted by importance sampling weights and averaged. MSE outperformed SmoothL1Loss in terms of stability. While experimenting with SmoothL1Loss and Mean Absolute Error (MAE), I found that SmoothL1Loss provided smoother training but resulted in slightly slower convergence. In contrast, MSE enabled faster learning and more consistent performance.

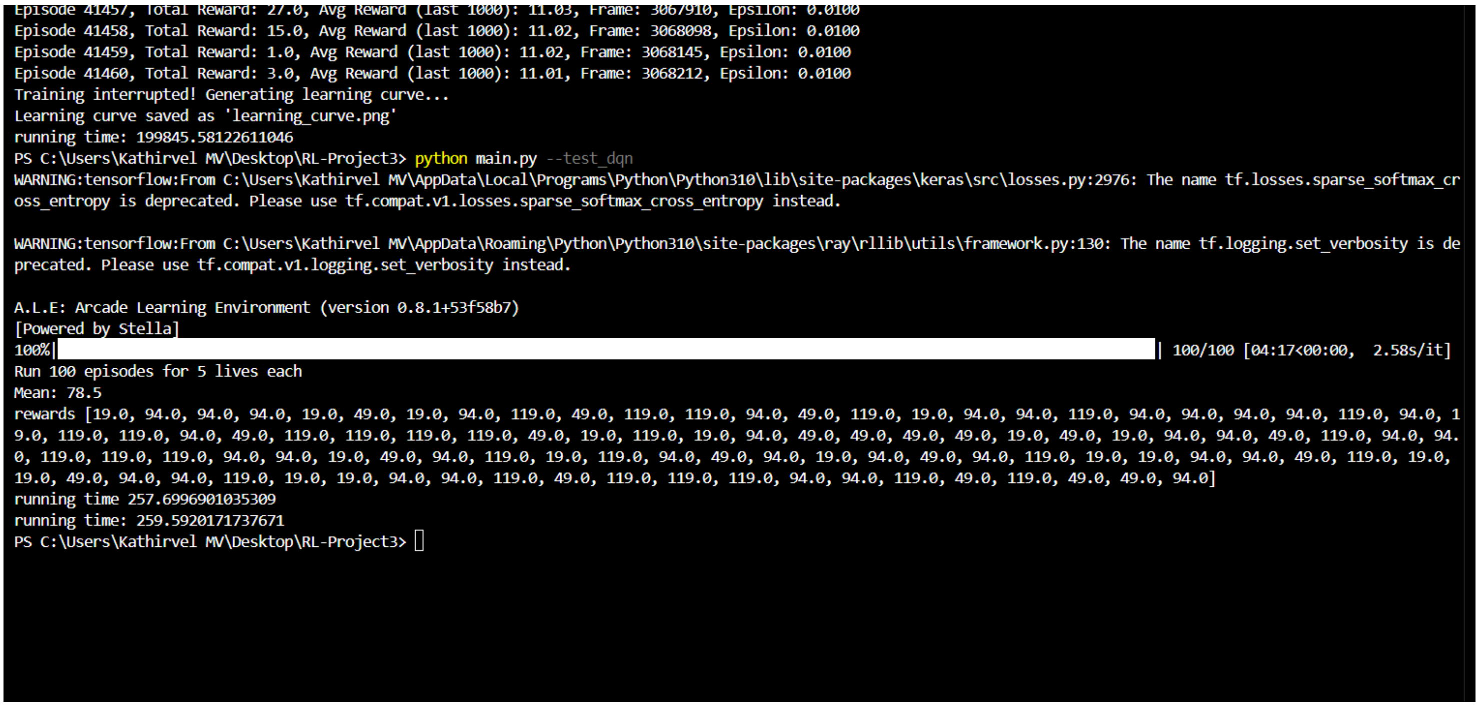
Execution and Observations

My execution was hampered multiple times, as my system crashed during the training phase about 3 times, 2 times locally and once on JupyterLab also. I had to optimize CPU usage and close all the other draining processes to prioritize the training of the model. Once it had run above 10,000+ episodes and then crashed, which was extremely disappointing.

The mean reward I was able to achieve was **78.5** after running for 100 episodes of testing phase which can be improved if I could train the model for longer number of episodes, given better computational resources.

I currently ran training for **41,460 episodes** which itself took 2.5 days of local computation.

Max Reward I got was **119.0,** which can also be increased with better training of model.



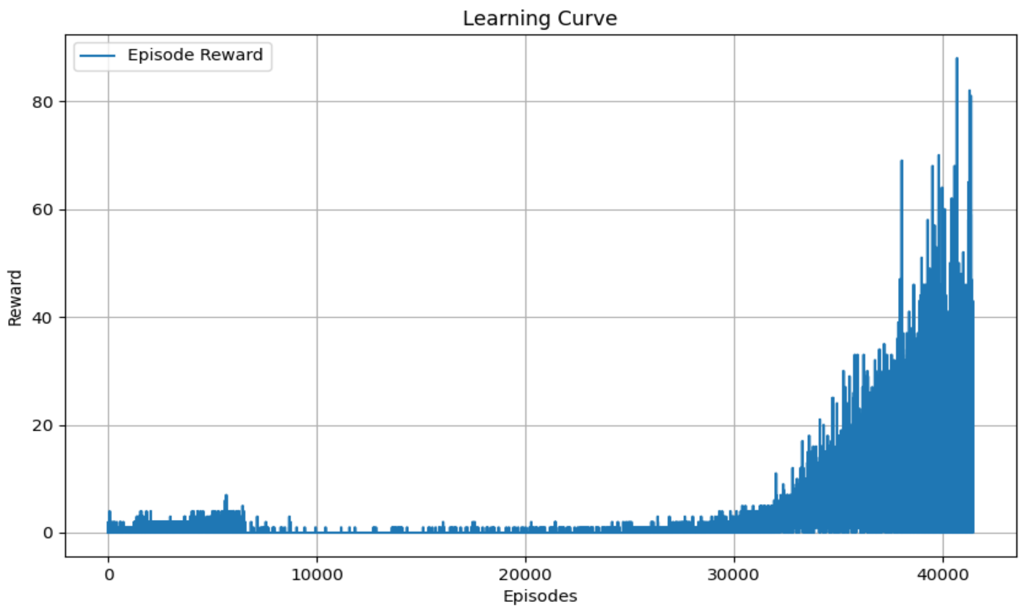
```
Episode 41457, Total Reward: 27.0, Avg Reward (last 1000): 11.03, Frame: 3067910, Epsilon: 0.0100
Episode 41458, Total Reward: 15.0, Avg Reward (last 1000): 11.02, Frame: 3068098, Epsilon: 0.0100
Episode 41459, Total Reward: 1.0, Avg Reward (last 1000): 11.02, Frame: 3068145, Epsilon: 0.0100
Episode 41460, Total Reward: 3.0, Avg Reward (last 1000): 11.01, Frame: 3068212, Epsilon: 0.0100
Training interrupted! Generating learning curve...
Learning curve saved as 'learning_curve.png'
running time: 199845.58122611046
PS C:\Users\Kathirvel MV\Desktop\RL-Project3> python main.py --test_dqn
WARNING:tensorflow:From C:\Users\Kathirvel MV\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cr
oss_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users\Kathirvel MV\AppData\Roaming\Python\Python310\site-packages\ray\rllib\utils\framework.py:130: The name tf.logging.set_verbosity is de
precated. Please use tf.compat.v1.logging.set_verbosity instead.

A.L.E: Arcade Learning Environment (version 0.8.1+53f58b7)
[Powered by Stella]
100%|██████████████████████████████████████████████████████████████████████| 100/100 [04:17<00:00,  2.58s/it]
Run 100 episodes for 5 lives each
Mean: 78.5
rewards [19.0, 94.0, 94.0, 94.0, 19.0, 49.0, 19.0, 94.0, 119.0, 49.0, 119.0, 119.0, 94.0, 49.0, 119.0, 19.0, 94.0, 94.0, 119.0, 94.0, 94.0, 94.0, 94.0, 119.0, 94.0, 1
9.0, 119.0, 119.0, 94.0, 49.0, 119.0, 119.0, 119.0, 119.0, 49.0, 19.0, 119.0, 19.0, 94.0, 49.0, 49.0, 49.0, 49.0, 19.0, 49.0, 19.0, 94.0, 94.0, 49.0, 119.0, 94.0, 94.
0, 119.0, 119.0, 119.0, 94.0, 94.0, 19.0, 49.0, 94.0, 119.0, 19.0, 119.0, 94.0, 49.0, 94.0, 19.0, 94.0, 49.0, 94.0, 119.0, 19.0, 19.0, 94.0, 94.0, 49.0, 119.0, 19.0,
19.0, 49.0, 94.0, 94.0, 119.0, 19.0, 19.0, 94.0, 94.0, 119.0, 49.0, 119.0, 119.0, 119.0, 94.0, 94.0, 119.0, 49.0, 119.0, 49.0, 49.0, 94.0]
running time 257.6996901035309
running time: 259.5920171737671
PS C:\Users\Kathirvel MV\Desktop\RL-Project3>
```

I was also able to generate a video that displays the Breakout gameplay based on the model and it was created for **100 episodes of 5 lives** each in the testing phase, the duration of which is **1.12 hours.**

Learning Curve Graph



Successful training is depicted and the rewards are in increasing trend, especially after 30,000 episodes more rewards were obtained consistently.

Areas of Improvement and Future scope

An area of improvement could be increasing the number of episodes beyond 41,000. Running for more episodes could help the model better fine-tune its policy, leading to more stable and reliable performance. Extended training allows the agent to explore the environment more thoroughly, improving the overall learning process. For future work, enhancing the model's performance can be achieved through several approaches. Exploring architectures like Rainbow DQN could improve reinforcement learning by combining several improvements, such as prioritized experience replay. Additionally, experimenting with variations of Double DQN could address Q-value overestimation, leading to more accurate predictions.

Thank You!