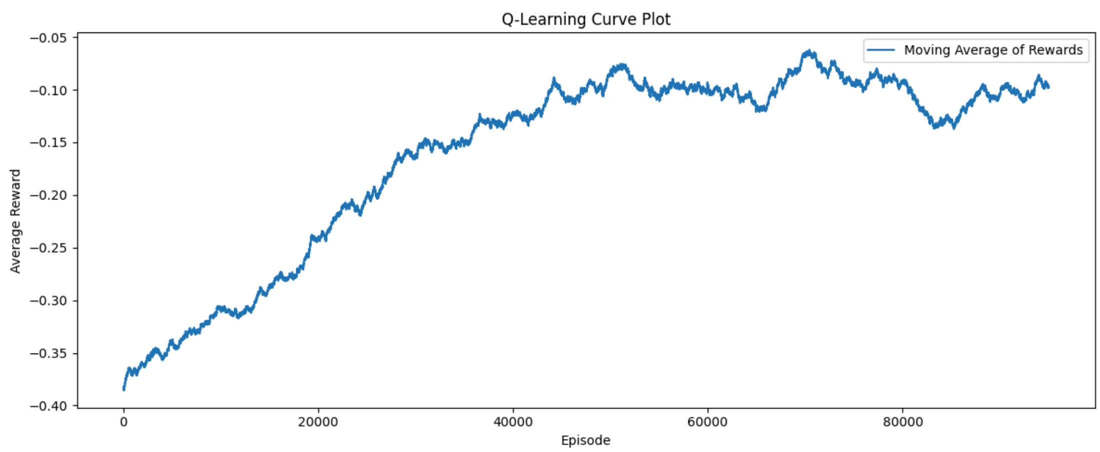
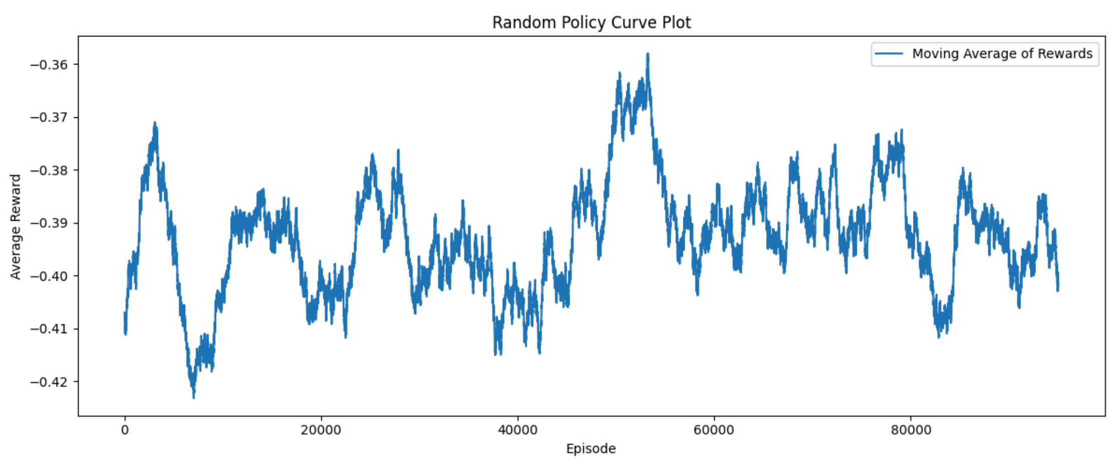
HW4: Q-Learning and DQN

Yeaseen Arafat and Jainta Paul

Part 1: Tabular Q-Learning to Win Big at Blackjack!!

The output graphs:





Comparison of Q-Learning vs. Random Policy:

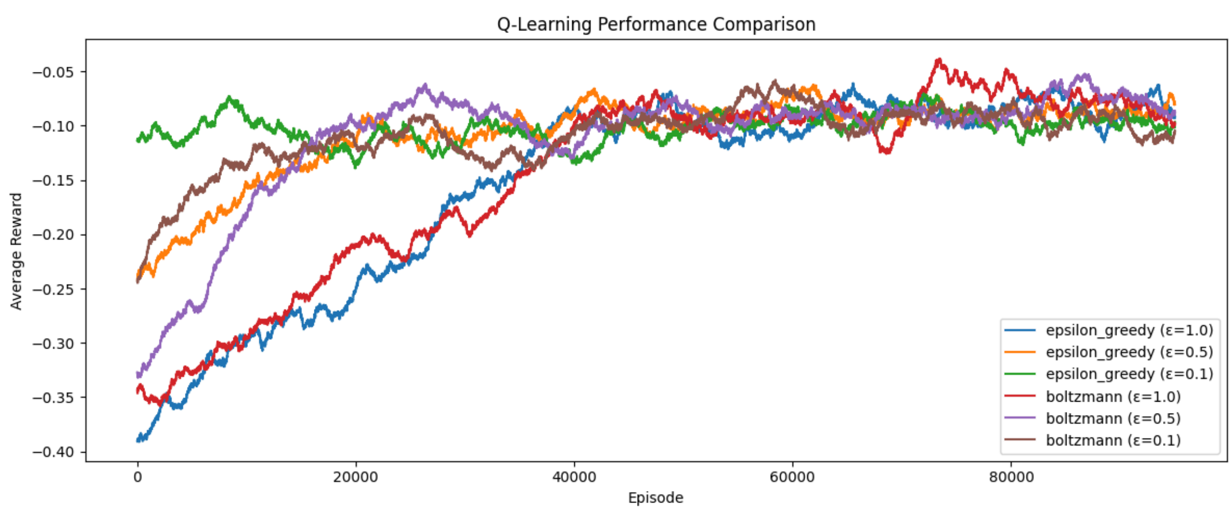
- **Learning Trend:** Q-learning shows a steady improvement over episodes, while the random policy fluctuates without a clear trend, indicating no learning.
- **Average Reward:** Q-learning achieves higher rewards over time, while the random policy maintains consistently lower rewards without improvement.
- **Stability:** Q-learning stabilizes after many episodes, whereas the random policy remains unstable with high variance in rewards.

- **Effectiveness of Learning:** Q-learning progressively improves decision-making, while the random policy remains inefficient and suboptimal.
- **Final Outcome:** Q-learning outperforms the random policy by maximizing rewards and converging to an optimal strategy, while the random policy fails to improve over time.

Extra Credit 1:

- Tested two **exploration strategies**:
  - **Epsilon-Greedy**: Selects a random action with probability ε, otherwise selects the action with the highest Q-value.
  - **Boltzmann Exploration**: Uses a softmax function over Q-values to probabilistically select actions.
- Ran experiments with **three different epsilon values**:
  - **ε = 1.0** (high exploration)
  - **ε = 0.5** (moderate exploration)
  - **ε = 0.1** (low exploration, more exploitation)

The comparison graph:

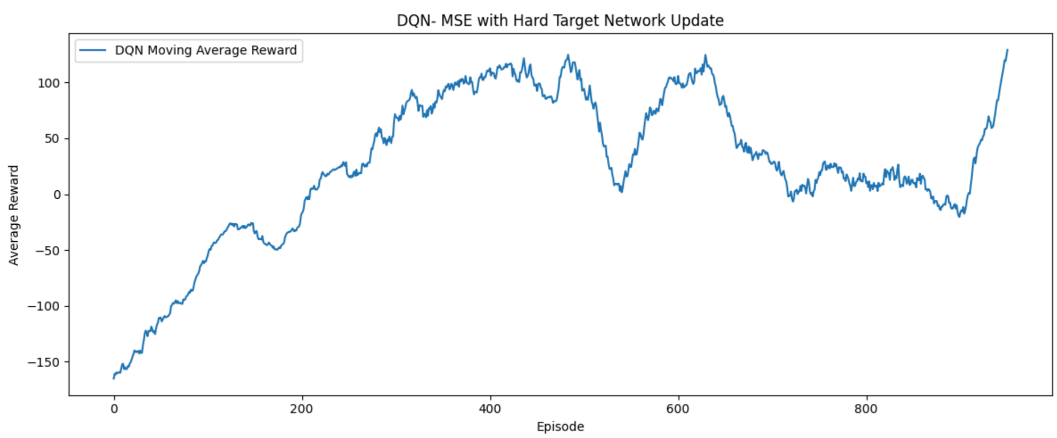


Insights:

- All methods improve over time, showing that both exploration strategies enable learning.
- **Epsilon-Greedy vs. Boltzmann:**
  - **Boltzmann (ε = 1.0, red)** starts slower but eventually **catches up**, indicating a smoother exploration process.
  - **Epsilon-Greedy (ε = 1.0, blue)** starts with rapid fluctuations but gradually converges.

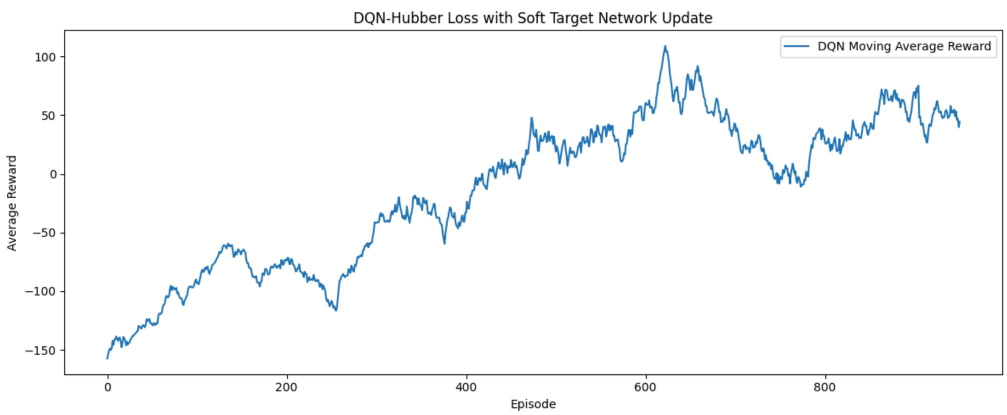
    - **Boltzmann (ε = 0.5, purple)** stabilizes earlier than epsilon-greedy with the same ε, suggesting **more balanced learning**.
    - **Epsilon-Greedy (ε = 0.1, green)** and **Boltzmann (ε = 0.1, brown)** have less fluctuation, showing **more exploitation but slower adaptation**.
- **Final Outcome:**
    - Moderate ε (0.5) performed the best, balancing exploration and exploitation effectively.
    - High ε (1.0) took longer to stabilize, while low ε (0.1) was too conservative, limiting learning speed.

Part 2: Landing on the Moon using DQN!



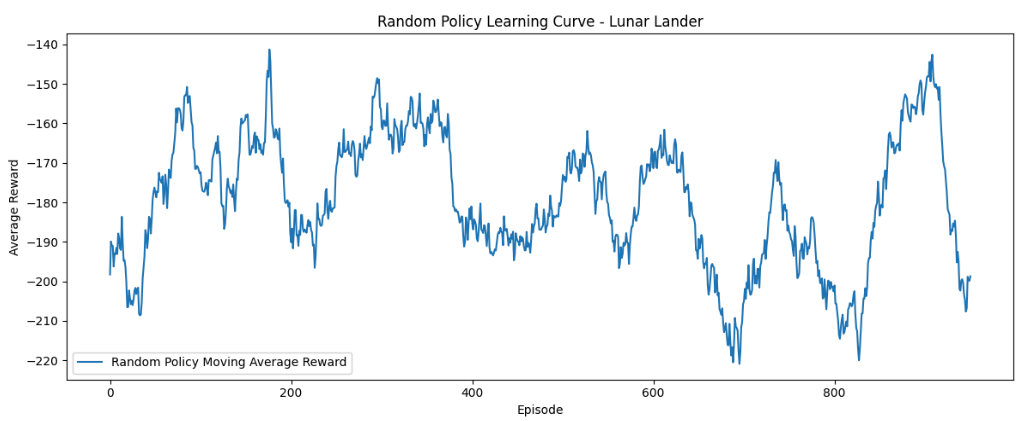
DQN with MSE Loss and Hard Target Update:

- **Learning Trend:** Gradual improvement in reward, with fluctuations.
- **Final Performance:** Reaches a high reward level but exhibits instability in later episodes.
- **Issues:** Large performance dips in the middle and late phases.



DQN with Huber Loss and Soft Target Update:

- **Learning Trend:** Smoother learning curve compared to MSE-based DQN.
- **Final Performance:** More stable and consistent reward progression.
- **Advantages:** Huber loss prevents extreme value swings, and soft updates help maintain stability.



Random Policy:

- **Learning Trend:** No clear learning, with high fluctuations across all episodes.
- **Final Performance:** Consistently poor, remaining at low rewards with no improvement.
- **Issues:** Fails to learn optimal landing strategies.

Key Takeaways

- **DQN with Huber Loss and Soft Updates is the best performer**, with improved stability and reduced variance.
- **DQN with MSE Loss and Hard Updates still learns well but suffers from instability** due to large weight changes.

- **The random policy performs the worst, as expected**, confirming the importance of reinforcement learning for this task.

Extra Credit 2:

Environment: Used **CarRacing-v3** with **discrete action space (`continuous=False`)** from Gymnasium.

Model Architecture:

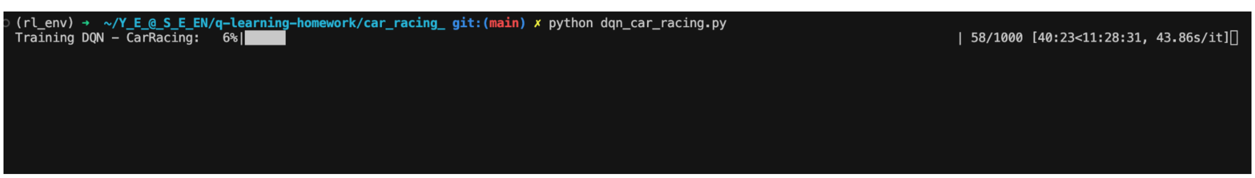
- **CNN-based DQN** with **three convolutional layers** for feature extraction.
- **Fully connected layers** for decision-making.
- **ReLU activations** and **Huber loss** for stable training.

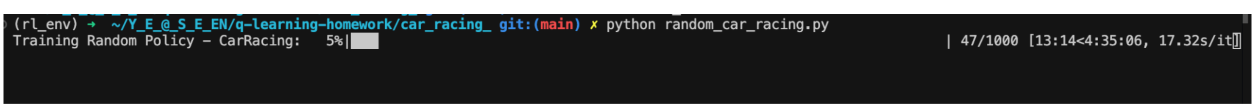
Training Strategy:

- **Replay Buffer:** Stores experiences for mini-batch training.
- **Epsilon-Greedy Exploration:** Balances exploration and exploitation.

The training for both DQN and Random Policy on CarRacing-v3 was extremely slow on Mac's MPS due to Metal's limited deep learning optimizations. Currently, We do not have access to a GPU, making the training process even more time-consuming.

After 4 hours, we got the following:





We have added the codes for these in the zip folder. Ideally, CNN based DQN algorithm should outperform the random policy based training implementation.