COMP3702 Artificial Intelligence (Semester 2, 2024)
Assignment 3: Reinforcement Learning

Name: Zhiyang Mei

Student ID: s4840033

Student email: Zhiyang.mei@uqconnect.edu.au

Note: Please enter the name, student ID number and student email to reflect your identity. When submitting to Gradescope, ensure you mark the start of each question.

Question 1 (Complete your full answer to Question 1 on the remainder of page 1)

a) Two Key Similarities:

- Both algorithms iteratively update value estimates (Q-values in Q-learning and state values in Value Iteration) based on Bellman equations.
- Both methods converge to the optimal value function or policy given enough iterations and exploration in an environment.

b) One Key Difference:

- Q-learning is a model-free, off-policy algorithm that learns the optimal action-value function by directly interacting with the environment. In contrast, Value Iteration is a model-based, policy-driven algorithm that requires full knowledge of the environment's transition dynamics to update the state values.

Question 2 (Complete your full answer to Question 2 on page 2)

2a) Plotting Function

The following code was implemented to plot the R100 value (100-step moving average episode reward) vs Episode reward. This code is adapted from the official PyTorch Reinforcement Learning Tutorial [**pytorchDQN**]:

```
def plot_r100_vs_episode(all_rewards, r100_list, save_path='r100_vs_episode.png'):
    plt.figure(figsize=(12, 6))
    episodes = range(1, len(all_rewards) + 1)

    plt.plot(episodes, all_rewards, label='Episode Reward')
    plt.plot(episodes, r100_list, label='R100 (Average of Last 100)', linewidth=2)

    plt.xlabel('Episode')
    plt.ylabel('Reward')
    plt.title('R100 vs Episode Reward')
    plt.legend()
    plt.grid(True)
    plt.savefig(save_path)
    plt.close()
```

This function uses the `matplotlib` library to plot the episode rewards and the R100 values, saving the output as a PNG file. The `r100_list` represents the moving average of the last 100 episode rewards.

R100 Calculation

```
if len(all_rewards) >= 100:
    r100 = np.mean(all_rewards[-100:])
else:
    r100 = np.mean(all_rewards)
r100_list.append(r100)
```

The R100 value is calculated as the average of the rewards from the last 100 episodes.

Training Loop

In addition, to record and update logs, I would like to update the plot every 20 episodes to achieve a smoother curve.

```
if episode_no % 20 == 0:
    plot_r100_vs_episode(all_rewards, r100_list)
    print(f"Plot updated at Episode {episode_no}")
```

2b)

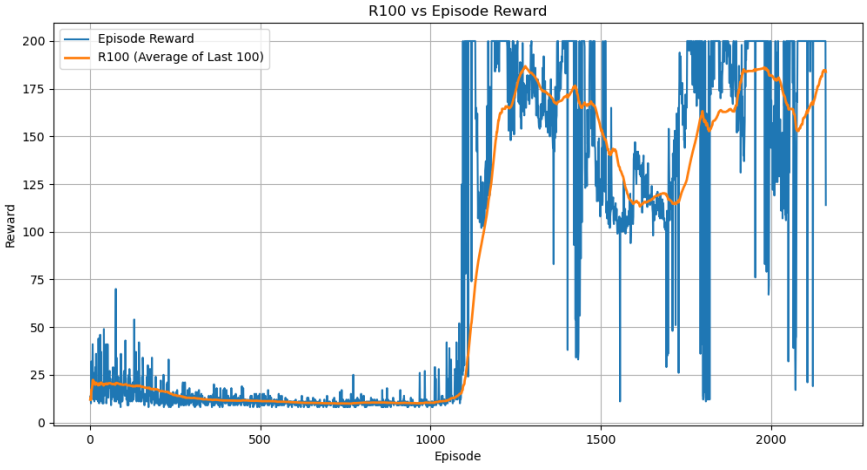

Figure 1: Two hidden layers DQN on CartPole-v0

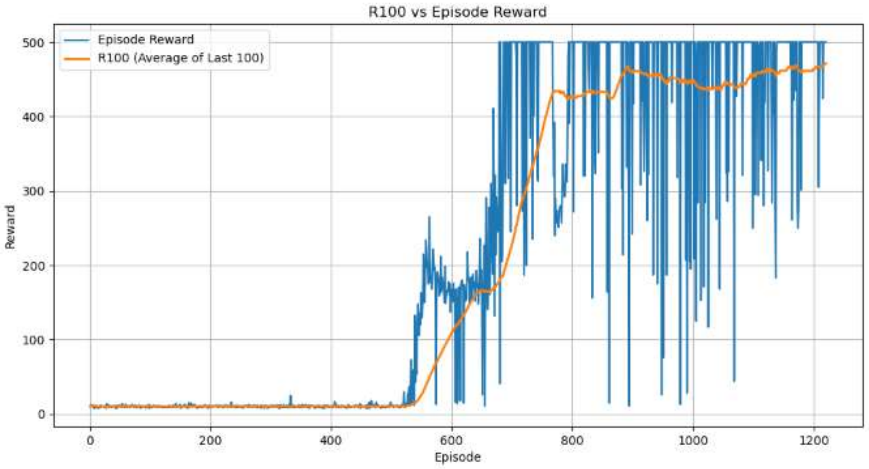

Figure 2: Two hidden layers DQN on CartPole-v1

2c)

Both *CartPole-v0* and *CartPole-v1* taught the system different ways to keep the pole balanced, mainly because of how their rules are set up. The main differences are in how long each version lets it play (episode steps) and the scores it needs to hit (reward threshold).

CartPole-v0: The policy here gets the pole stable fast but doesn't keep it balanced for very long. With a max of 200 steps per episode and needing at least 195 points to succeed, it focuses on quick reactions.

CartPole-v1: This version's policy is better at keeping things steady for a longer time. It's set up for up to 500 steps and needs a much higher score of 475 to pass. It smoothly handles ups and downs over these longer games.

Why the Difference Matters: Jumping from 200 to 500 steps and from 195 to 475 points in CartPole-v1 means the game wants it to get good at balancing for a longer time. This helps train the system to deal with problems better and for longer, kind of like training for harder real-life tasks, where disturbances are frequent and varied. The game becomes not just about quick fixes but about managing more and thinking ahead.

Conclusion: The changes between CartPole-v0 and CartPole-v1 show how the challenges are ramped up to push for more advanced learning and strategies. In v1, the system learns to expect and handle problems better over time, showing a big step up in what it can do.

Question 3 (Complete your full answer to Question 3 on page 3)

a) In Deep Q-Networks (DQN), the loss function is based on Temporal Difference (TD) learning and is defined as:[**mnih2015human**]

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta^-)}_{\text{Target Value } y_i} - \underbrace{Q(s, a; \theta)}_{\text{Current Prediction}}\right)^2\right]$$

- **Target Value (y_i)**:

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

  Combines the immediate reward r and the discounted maximum Q-value of the next state s' using the target network parameters θ^-.

- **Current Prediction**:

$$Q(s, a; \theta)$$

  The neural network's current estimate of the Q-value for state-action pair (s, a).

b)

- **Target Values:**

```
next_state_values = target_net.forward(next_states_v).max(1)[0]

expected_state_action_values =
rewards_v + next_state_values.detach() * params['gamma'] * (1 - dones_v)
```

- **Current State-Action Value Predictions:**

```
Q_s = net.forward(states_v)
state_action_values = Q_s.gather(1, actions_v.type(torch.int64).unsqueeze(-1)).squeeze(-1)
```

c) In Deep Q-Networks (DQN), a separate **target network** is used to improve training stability. The main reasons are:

- **Stable Targets**: The target network provides fixed Q-value targets for several training steps. This prevents the targets from changing too quickly, which helps the learning process remain stable.
- **Reduced Correlation**: By keeping the target network separate and updating it less frequently, the predictions and targets are less correlated. This reduces the risk of oscillations and divergence in the Q-value estimates.

Conclusion: Using a separate target network ensures that the Q-value updates are more stable and reliable, leading to better convergence of the DQN algorithm.

d)

- **Periodic Synchronization (Alpha Sync = False, Target Net Sync = 1000):**
  - The Figure 11 (please see appendix) plot shows results with periodic updates of the target network. Here, the target network parameters are completely replaced with the parameters of the main network every 1000 steps.
  - The training shows significant variability in rewards but eventually reaches and maintains a high performance level, indicating effective learning over time.
- **Soft Updates (Alpha Sync = True, Tau = 0.005):**
  - The Figure 12 (please see appendix) plot represents the soft update method where the target network parameters are slowly adjusted towards the main network parameters, controlled by a factor $\tau = 0.005$.
  - This approach shows a smoother increase in rewards and less variability in performance, suggesting a more stable learning process throughout the training episodes.

Conclusion: The comparison of the two methods shows that soft updates provide a more stable and consistent improvement in performance, likely due to the gradual integration of new knowledge into the target network. Periodic updates, while effective, show more fluctuations, which might complicate convergence in environments with more complex or varied dynamics.

Question 4 (Complete your full answer to Question 4 on page 4)

a)

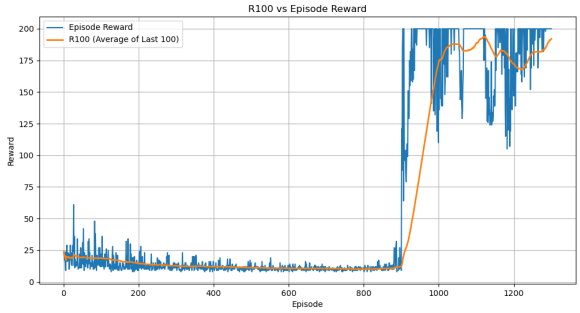

Figure 3: lr:1.0e-4 with Two hidden layers DQN on CartPole-v0

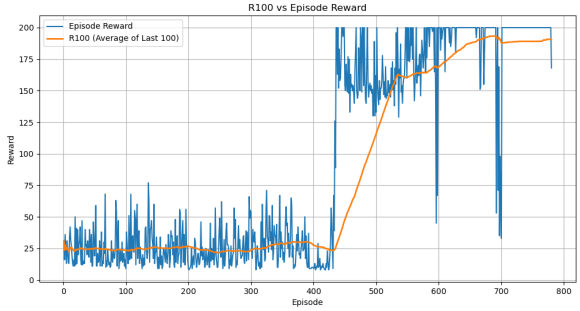

Figure 4: lr:1.0e-3 with Two hidden layers DQN on CartPole-v0

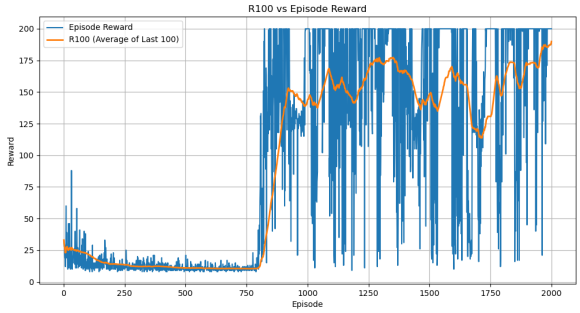

Figure 5: lr:1.0e-2 with Two hidden layers DQN on CartPole-v0

b) From the plots, we can observe two key factors influenced by the learning rate:

- **Learning Speed**: A higher learning rate ($1e{-}2$) leads to faster learning initially, but it causes unstable updates, resulting in fluctuations in performance. A moderate learning rate ($1e{-}3$) balances learning speed with stability, resulting in more consistent improvements. A lower rate ($1e{-}4$) ensures stability but at the cost of slower learning.
- **Stability and Convergence**: The highest learning rate ($1e{-}2$) shows signs of divergence, where the agent struggles to maintain high rewards. The moderate rate ($1e{-}3$) provides the best balance, leading to smooth convergence. The lowest rate ($1e{-}4$) is stable but slow to converge.

c) What Happens When the Learning Rate Is Too High?

When the learning rate is too high, such as $1e{-}2$, the parameter updates become too large, leading to instability. This results in oscillations in the reward as the agent overshoots optimal policy improvements, often leading to divergence or poor long-term performance. The agent fails to converge to a stable policy, as shown in the plot for $1e{-}2$, where the R100 values fluctuate widely even after many episodes.

In gradient descent, a high learning rate can cause the loss function to oscillate or increase instead of decreasing steadily. This leads to inefficient learning and can prevent the agent from converging to an optimal solution.

Question 5 (Complete your full answer to Question 5 on page 5)

a)

The epsilon (ϵ) hyperparameter in reinforcement learning governs the exploration-exploitation trade-off in ϵ-greedy policies. It determines the probability of the agent exploring (choosing a random action) versus exploiting (choosing the action with the highest predicted Q-value).

- When $\epsilon \approx 1$, the agent explores extensively by taking random actions, which helps it discover new strategies and avoid local optima.
- When $\epsilon \approx 0$, the agent mainly exploits its learned policy by taking the action it believes to be optimal, which improves performance as the agent learns.

The goal is to balance sufficient exploration in the early stages of learning while transitioning to more exploitation as the policy improves.

b)

- **Epsilon Start**: This is the initial value of ϵ, typically set close to 1 to promote exploration at the beginning of training. A higher ϵ start ensures that the agent thoroughly explores different actions early on.
- **Epsilon Decay**: This parameter controls the rate at which ϵ decreases over time, transitioning from exploration to exploitation. Faster decay (smaller values) leads to quicker exploitation, which may reduce the chance of discovering optimal strategies in complex environments. Slower decay allows for more exploration but delays exploitation.
- **Epsilon Final**: This is the minimum value ϵ can reach. A higher ϵ final promotes continuous exploration even in later training stages, which may help prevent the agent from getting stuck in suboptimal policies. A very low ϵ final value encourages pure exploitation.

Question 6 (Complete your full answer to Question 6 on page 6)

Duelling DQN

introduces a structural change to the Vanilla DQN architecture by splitting the Q-value function into two separate streams: one for the *state value* and one for the *advantage function*. This modification allows the agent to better evaluate the value of each state independently of the specific action taken.

Key Distinction

In Vanilla DQN, the Q-value for each state-action pair is estimated directly as $Q(s, a)$. However, in Duelling DQN, the Q-value is decomposed as:

$$Q(s, a) = V(s) + A(s, a)$$

Where: - $V(s)$ is the value function representing how good it is to be in state s. - $A(s, a)$ is the advantage function, representing the relative importance of action a compared to other actions at state s.

This separation allows Duelling DQN to more accurately learn the value of each state, especially in environments where some actions are irrelevant in certain states.

Advantages of Duelling DQN

1. **Better State Value Estimation:** By estimating the state value $V(s)$ directly, Duelling DQN can better distinguish between valuable states and less valuable states without needing to account for the effects of each action explicitly. This can improve performance in environments where many actions have little effect on the overall value.
2. **Improved Learning Efficiency:** The decomposition of the Q-value into the state value and advantage allows the model to focus on learning the more critical aspects of state evaluation and action selection, which can lead to faster and more stable learning.

Performance Analysis from the Figures

- The Figure 13 (please see appendix) (*Duelling DQN*) demonstrates a steady increase in the reward after about 800 episodes, with a noticeable improvement in the moving average of rewards (orange line) towards the end. This indicates that the Duelling DQN is able to achieve better performance over time.
- The Figure 14 (please see appendix) (*Two Hidden Layers with Vanilla DQN*) shows that while the rewards begin to improve around the same time (around episode 600), the convergence is less stable compared to Duelling DQN. The fluctuations in rewards suggest that the Vanilla DQN architecture struggles more with stable learning compared to Duelling DQN.
- In conclusion, Duelling DQN improves on Vanilla DQN by better capturing the value of each state, leading to more efficient and stable learning, as shown in the plots.

Question 7 (Complete your full answer to Question 7 on page 7)

Baseline

Hyperparameter	Final Value
Learning Rate (LR)	1.0e-3
DQN	Duelling DQN
alpha-sync	true
tau	0.005

Table 1: Final Hyperparameter Values

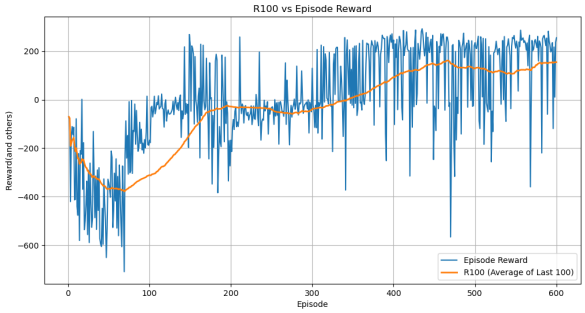

Figure 6: Baseline for LunarLander-v3

Hyperparameters and Experimentation

Learning Rate (LR)

I experimented with different learning rates to observe the impact on convergence speed and policy stability. Figure 7 compares the performance of agents trained with varying learning rates. As shown, a lower learning rate of 0.0005 led to slower convergence, whereas higher rates like 0.001 were unstable. Despite the stable learning, I finally chose the 0.001 to be the optimal value since it saved us much more episodes to solve the environment.

First Change: LR: 5.0e-4

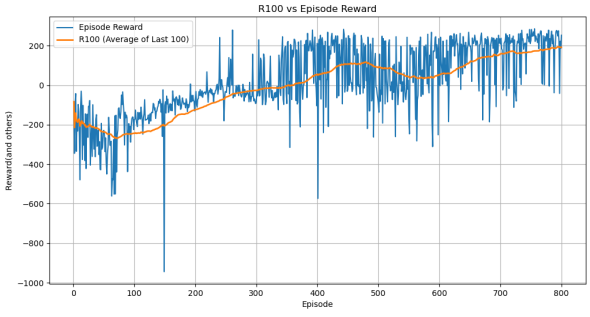

Figure 7: LR Change for LunarLander-v3

Alpha-Sync

According to the plots, we can clearly discover that without soft update, the agent solved the env with less than 500 episodes, whereas, the soft updates resulted in more than 600 episodes, which is a bit strange. This suggests that soft updates may have introduced instability in the learning process or slowed down the convergence if we don't carefully tune tau.

Second Change: Alpha-Sync: false

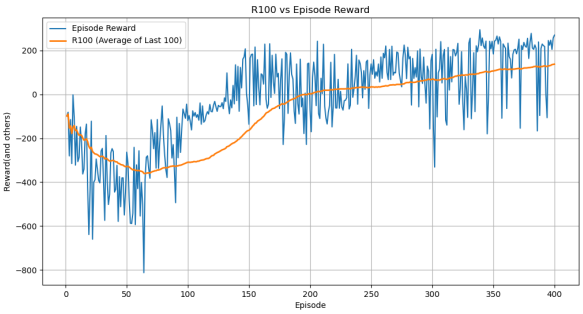

Figure 8: Sync Change for LunarLander-v3

DQN

As illustrated in Figure 9, the two-hidden-layer DQN initially lagged behind the Duelling DQN in terms of convergence speed, especially in the earlier episodes. However, as training progressed, the two-hidden-layer DQN reached similar levels of performance, with comparable R100 values around the later episodes.

Third Change: DQN: Two Hidden Layers DQN

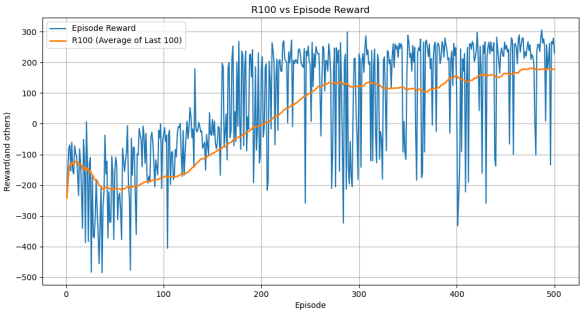

Figure 9: DQN Change for LunarLander-v3

Final Hyperparameter Settings

Combined Change:

Final Settings of Hyperparameter Values (please see appendix)

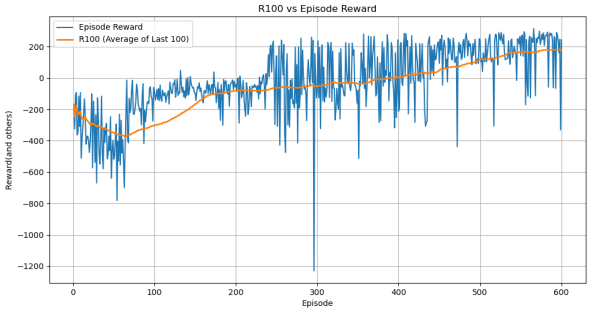

Figure 10: Combined Change for LunarLander-v3

Appendix/References (Include references and usage of Generative AI on the last page)

A Q3 (d) Plots

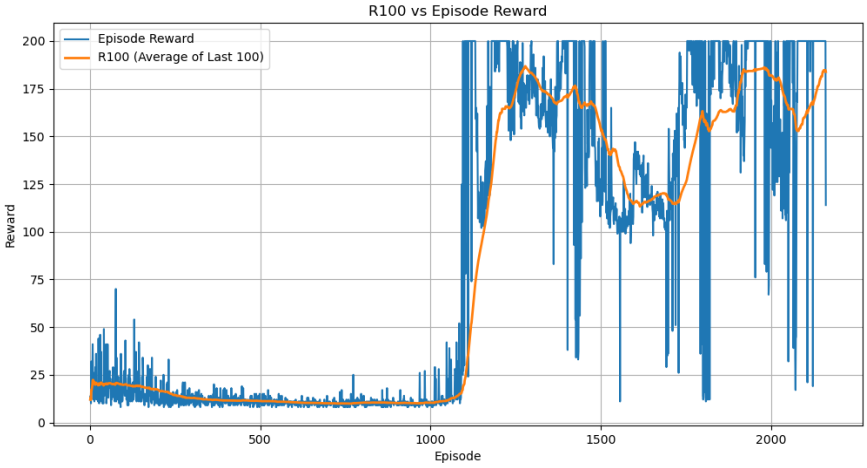

Figure 11: Periodic Synchronization with Two hidden layers DQN on CartPole-v0

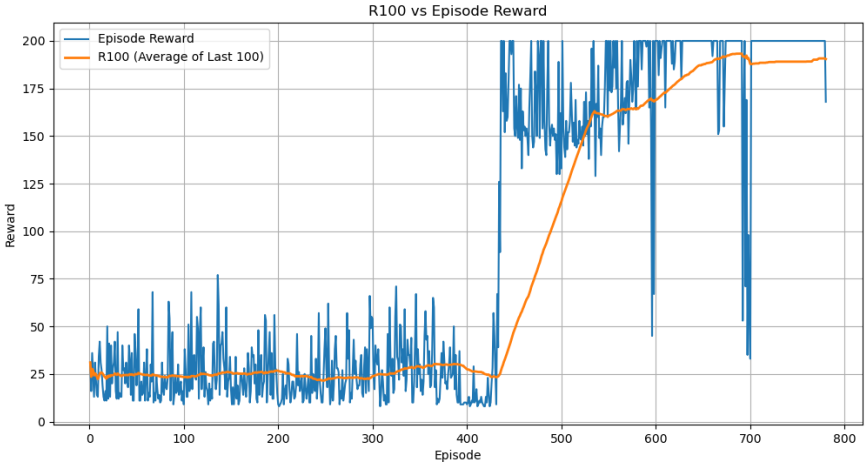

Figure 12: Soft Updates with Two hidden layers DQN on CartPole-v0

B Q6 Plots

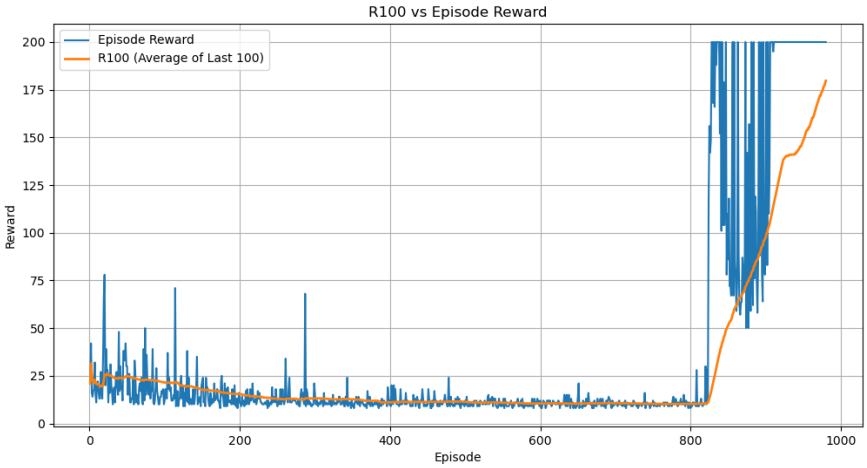

Figure 13: Duelling DQN - CartPole-v0

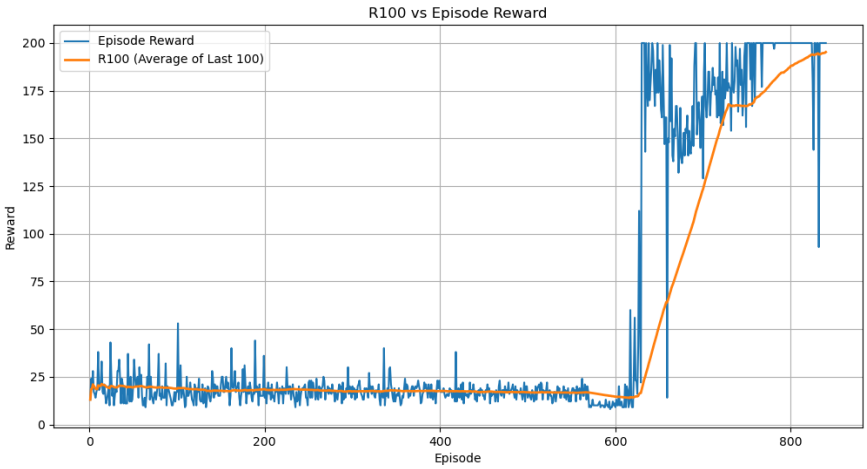

Figure 14: Vanilla DQN - CartPole-v0

C Final settings of hyperparameter values for Q7:

gamma:	0.99
batch_size:	64
hidden_size:	256
hidden_size2:	300
replay_size:	1.0×10^6
replay_size_start:	5000
epsilon_decay:	5000
epsilon_final:	0.001
epsilon_start:	1.0
Learning_rate:	5.0×10^{-4}
target_net_sync:	1.0×10^3
alpha_sync:	true
tau:	0.005
stopping_reward:	200
max_frames:	1.0×10^6
save_path:	saved_models
clip_gradient:	false

References

[1] PyTorch Reinforcement Learning Tutorial: *Deep Q-Learning*. Available at: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

[2] V. Mnih, K. V. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. K. P. Omar, G. Czarnecki, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. https://www.nature.com/articles/nature14236.

[3] OpenAI, *ChatGPT: Optimizing Language Models for Dialogue*, 2023. [Online]. Available: https://openai.com/chatgpt. [Accessed: 24- Oct- 2024].

Declaration of AI Assistance

I declare that I have used OpenAI's ChatGPT to assist me in improving the clarity and expression of my writing, as well as converting content into LaTeX format. I take full responsibility for the content of the work and have reviewed and verified the final output for accuracy and appropriateness.