Deep Reinforcement Learning - Exercise 1

Lior Nisimov (325026821), Freddie Grabovski (209258912)

December 4, 2024

1 Section 1

Answers for part 1 of the assignment.

1.1 Why methods such as Value-Iteration cannot be implemented in such environments? Write down the main problem

The reason is that in Value-Iteration, the transition function is needed to calculate the value of each state visited during the algorithm:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[\sum_{s'} P(s' \mid s, a) \left(R(s, a, s') + \gamma V_k(s') \right) \right],$$

This information is unknown to us in model-free environments. Similar methods such as policy iteration are similarly dependent on the transition function. Another pitfall for these algorithms is they require us to explore all the states of the environment, which might not be feasible in environments that are too complicated or have a too large number of states.

1.2 How do model-free methods resolve the problem you wrote in previous question? Explain shortly.

These methods resolve the problem by utilizing Monte Carlo methods, meaning we create trajectories by random sampling. By following the current policy we can generate an episode (a finite set of transitions through the states while following the policy) which we can use the update the Q tables, which in turn, models the transition function implicitly. Thus, theses methods do not require prior knowledge of the environment's dynamics, and instead use the observed outcomes to learn optimal actions.

1.3 What is the main difference between SARSA and Q-learning algorithms? Explain shortly the meaning of this difference

The main difference between SARSA and Q-learning lies in how they update their Q-values during learning. SARSA is an on-policy algorithm, meaning it updates Q-values based on the action chosen by the agent's current policy. In contrast, Q-learning is off-policy, updating Q-values based on the best possible action (highest Q-value), regardless of the current policy. This distinction means that SARSA learns based on the agent's actual behavior, making it more cautious and suitable for environments where exploration carries risks. On the other hand, Q-learning focuses on finding the optimal policy faster, but it may take riskier actions during learning. SARSA is generally more suitable for tasks where safety is critical, while Q-learning is often preferred in deterministic or less risky environments where finding the optimal policy is the main goal.

1.4 Why is it better than acting greedily?

The reason that decaying ϵ-greedy is better than acting greedily is that it balances exploration and exploitation. We usually begin with poor estimation of our environment (the Q values). Therefore it is beneficial to have an exploration coefficient (ϵ) to let us improve our estimations. Slowly capitalizing on our improved estimations by decreasing ϵ allows us to converge on the best policy instead of continuing to take random actions. Using the greedy approach can get ”stuck” on suboptimal actions forever since it never explores or improves its estimations for less-explored regions.

1.5 Results and Analysis

This section presents an analysis of the training process and outcomes.

1.5.1 Hyper-parameters

The final solution was achieved using the following hyper-parameters: a final epsilon of 0.0, an initial epsilon of 1.0, and an epsilon decay of 0.001, which ensured a gradual transition from exploration to exploitation. The learning rate was set to 0.5, allowing for significant updates to the Q-values while maintaining stability in training. A discount factor of 0.999 was chosen to prioritize long-term rewards, and the agent was trained over 5000 episodes for thorough exploration of the environment.

1.5.2 Q-Value Evolution

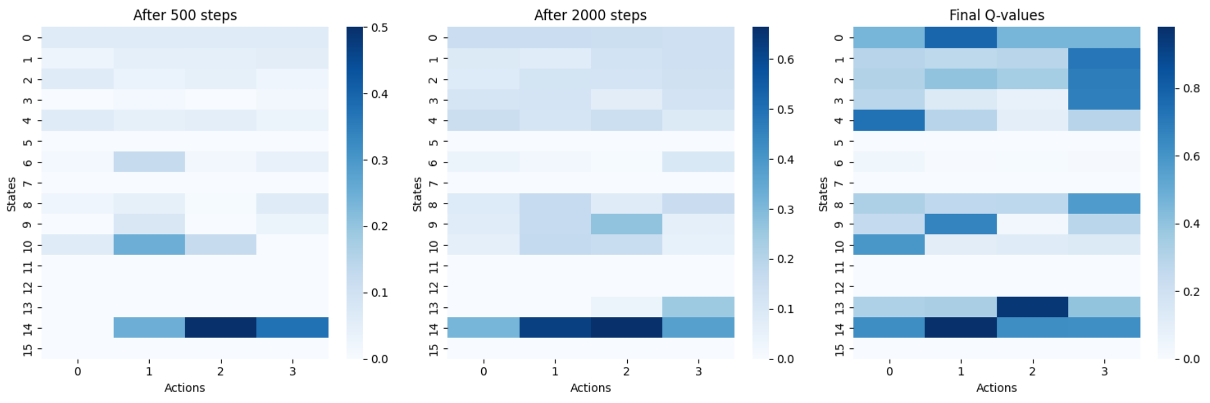

Figure 1: Q-value heatmaps at different stages of training. After 500 steps, the Q-values are sparsely populated and lack structure, reflecting early exploration. By 2000 steps, patterns emerge as the agent shifts towards exploitation, balancing exploration and learned knowledge. The final Q-values show clear and distinct high-value regions, signifying the agent’s convergence to an optimal policy.

1.5.3 Reward Per Episode

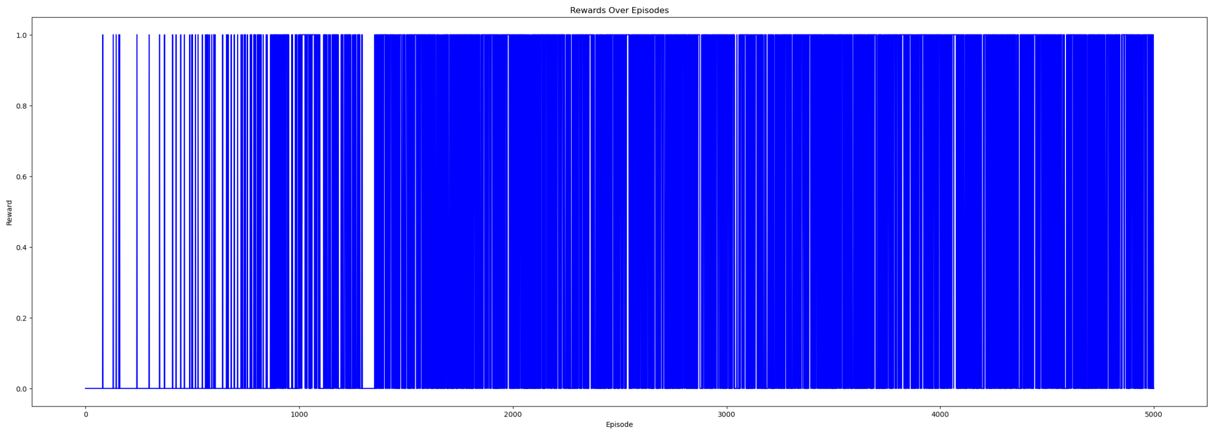

Figure 2: Rewards obtained by the agent in each episode during training. The plot demonstrates the agent's learning progression: sparse rewards during early exploration, fluctuating rewards as it refines its policy, and consistent rewards in the later episodes after convergence to an optimal policy.

1.5.4 Average Steps to Goal Over Episodes

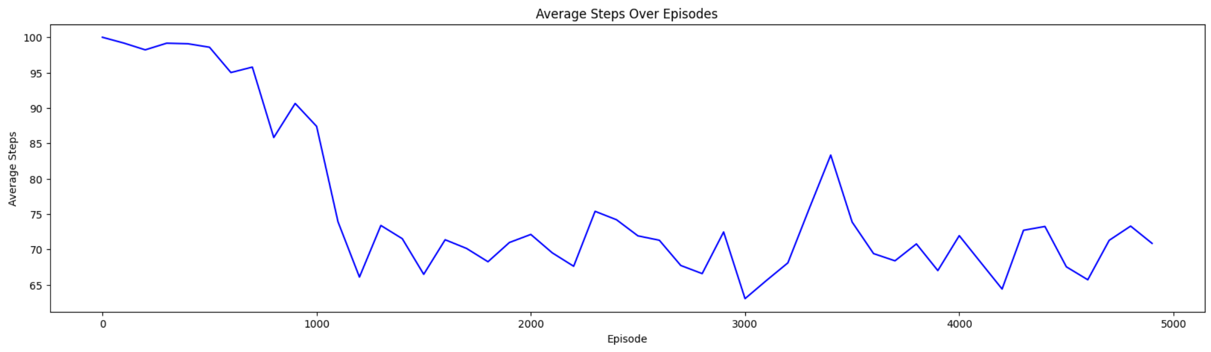

Figure 3: The average number of steps taken to reach the goal over the last 100 episodes, plotted at intervals of 100 episodes. Initially, the agent takes a high number of steps due to exploration, but a sharp decline occurs as the agent learns more efficient paths. After 2000 episodes, the average steps stabilize, indicating convergence to an optimal policy.

2 Section 2

2.1 Why do we sample in random order?

Sampling in random order from the experience replay ensures that earlier experiences remain influential and are not overshadowed by more recent ones. By shuffling the experiences, the algorithm provides a diverse mix of past events in each training batch, preventing the learning process from focusing too much on recent experiences. This avoids bias toward specific patterns, ensures all experiences contribute to learning, and makes the process more stable and effective.

2.2 How does this improve the model?

Using an older set of weights to compute the targets in DQN helps improve convergence by stabilizing the learning process. If the same network is used for both predictions and target calculations, any updates to the network will simultaneously alter the predictions and the ”labels” (target values), creating a moving target and making convergence difficult. By using a fixed, older set of weights for target calculations, the target values remain more consistent over multiple updates, reducing oscillations and allowing the model to steadily minimize the error. This separation slows down changes in the targets, helping the model converge more reliably to optimal Q-values.

2.3 Results and Analysis

This section provides an analysis of the training results, including the effect of hyperparameters, network architecture, and the agent’s performance in terms of loss, rewards, and steps.

2.3.1 Hyperparameters

The hyperparameters for the final solution were chosen to balance exploration, exploitation, and stable learning. For importance sampling, an α value of 0.6 was selected to give more weight to experiences with higher TD errors while still including lower-priority transitions. A starting β of 0.4, with a gradual increment of 0.001, allowed a smooth progression toward unbiased sampling over time. Additionally, a learning rate of 0.01, an epsilon decay rate of 0.1, a discount factor of 0.95, and a batch size of 1024 were determined through grid search. These parameters provided effective learning dynamics in the CartPole environment while maintaining training stability.

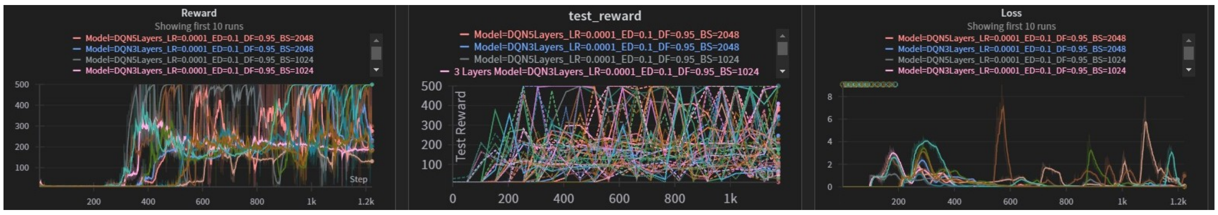

Figure 4: Comparison of 48 different runs across two model architectures to evaluate performance and outcomes.

2.3.2 Network Architecture Comparison

Two network architectures were evaluated: one with 3 hidden layers and another with 5 hidden layers, both configured with 32 neurons per layer and ReLU activation functions. The 5-layer network demonstrated faster convergence, achieving a high test reward within approximately 200 steps, while the 3-layer network required closer to 750 steps to reach comparable performance. This suggests that the additional layers in the 5-layer architecture provided the capacity to model more complex relationships within the environment, allowing the agent to develop an effective policy more efficiently.

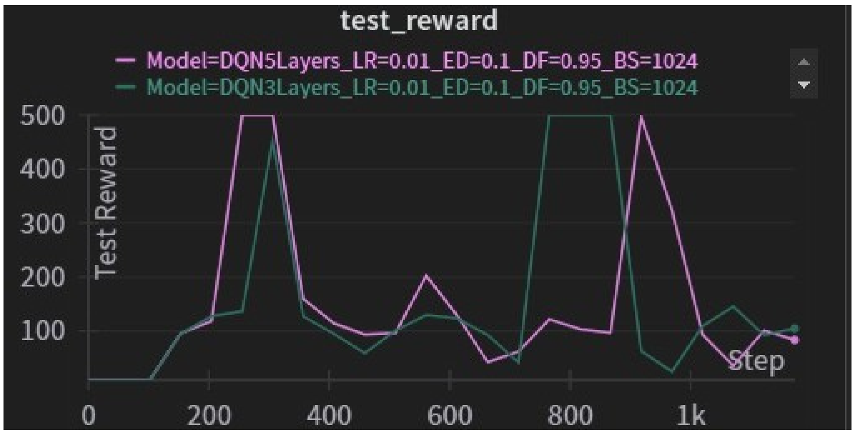

Figure 5: Comparison of test rewards between two network architectures (3-layer vs. 5-layer) across training steps. The 5-layer network achieves faster convergence and exhibits greater stability.

2.3.3 Loss Trend During Training

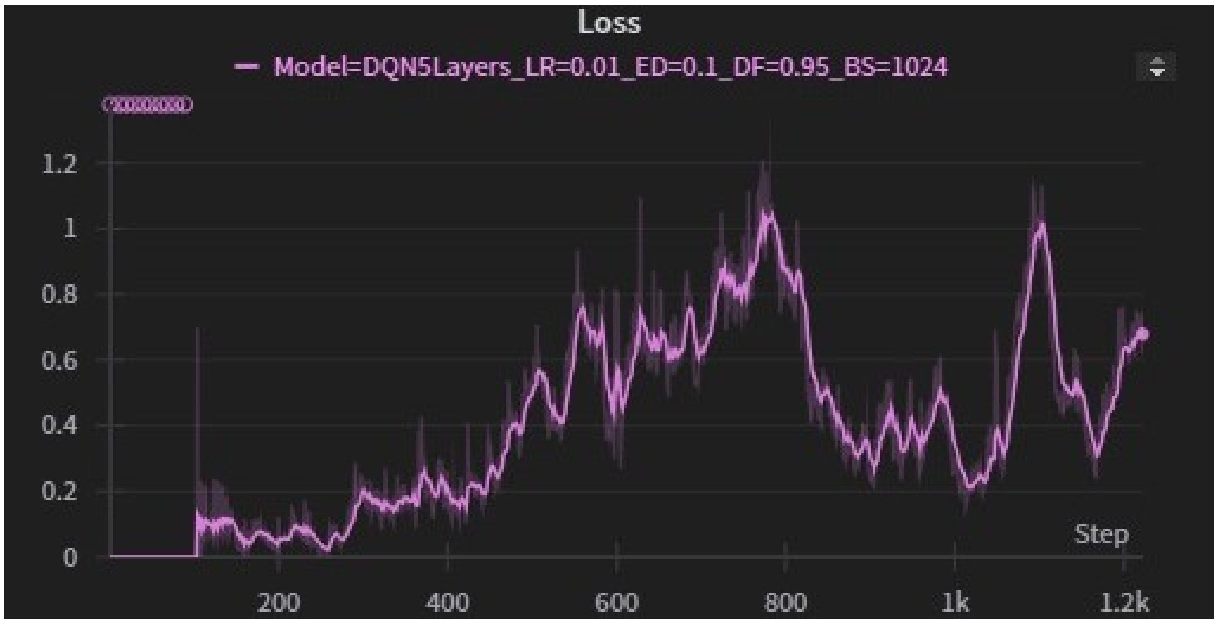

Figure 6: Loss over training steps. The faded line represents the raw loss values, while the clear line shows the moving average.

2.3.4 Reward Per Step

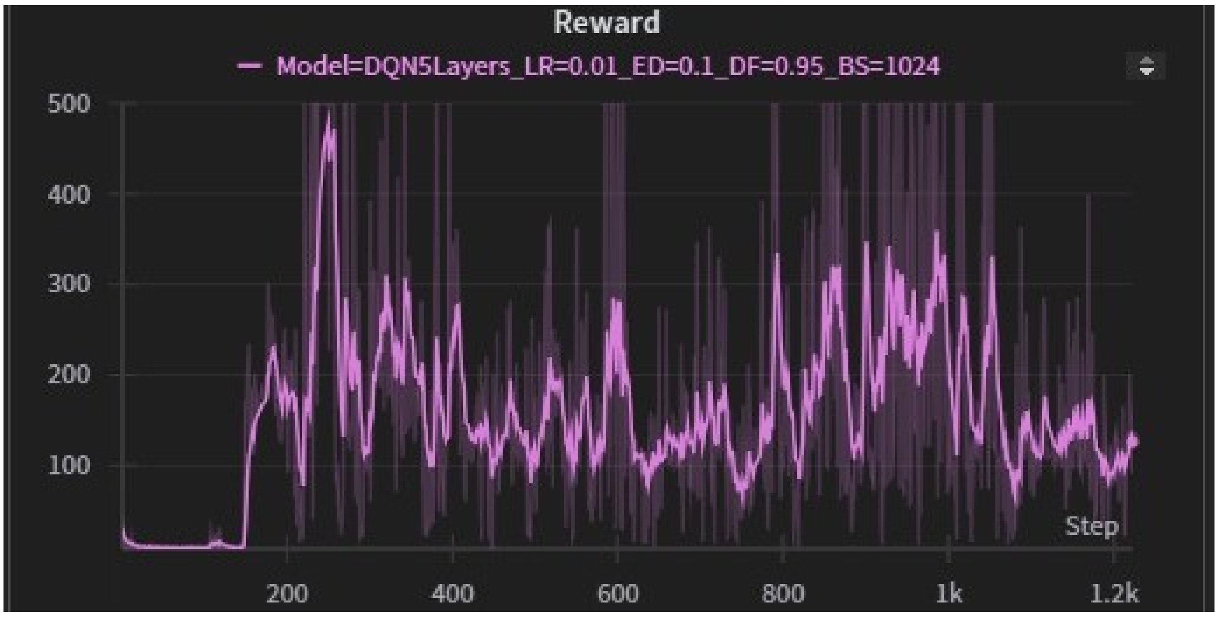

Figure 7: Rewards obtained per episode during training.

2.3.5 Effect of Discount Factor

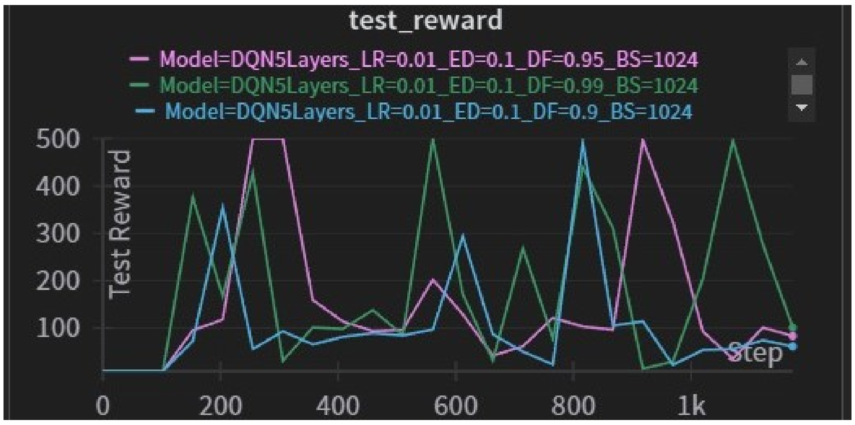

Figure 8: Comparison of performance for different discount factors (0.95, 0.99, and 0.9).

Figure 8 shows the performance of models trained with different discount factors ($\gamma = 0.95, 0.99, 0.9$). The discount factor of 0.95 achieved relatively high and consistent test rewards early in training, indicating a good balance between short-term and long-term rewards. The discount factor of 0.99 demonstrated slower learning and high variability, likely due to the strong emphasis on long-term rewards, which may delay immediate policy optimization. Conversely, the discount factor of 0.9 showed lower and less stable test rewards, suggesting that undervaluing future rewards led to suboptimal policies.

2.3.6 Effect of Epsilon Decay

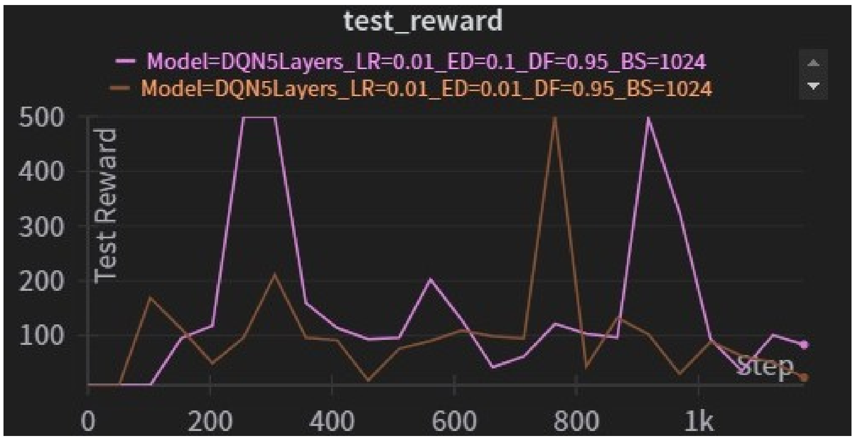

Figure 9: Comparison of performance for different epsilon decay rates (0.1 vs. 0.01).

Figure 9 illustrates the impact of different epsilon decay rates on the agent's training performance. The decay rate of 0.1 allowed the agent to reduce exploration more rapidly, leading to quicker convergence and higher test rewards early in training. In contrast, a slower decay rate of 0.01 caused the agent to spend more time exploring, which delayed convergence and resulted in lower and more variable test rewards. These results suggest that in environments like CartPole, a faster reduction in exploration (via a higher epsilon decay rate) helps the agent focus on exploiting its learned policy sooner, improving both efficiency and performance. However, slower decay may be beneficial in more complex environments where extended exploration is needed.

2.3.7 Section Conclusion

The results demonstrate that the selected hyper-parameters and 5-layer network architecture enabled the agent to learn efficiently. The prioritization mechanism, along with the chosen epsilon decay and discount factor, supported stable learning while ensuring the agent focused on valuable experiences. This highlights the importance of balancing exploration and exploitation in reinforcement learning tasks.

3 Section 3

In this section, we implemented importance sampling to enhance the training process. Importance sampling is a technique used in reinforcement learning to adjust for biases introduced when sampling experiences non-uniformly, such as in prioritized experience replay. By assigning importance weights to each sampled experience, the algorithm can correct for this bias, ensuring that the learning process remains both efficient and unbiased. This is particularly beneficial in focusing the agent's learning on transitions that are more significant while preserving stability.

3.1 Advantages of Importance Sampling

The use of importance sampling provides several key benefits in reinforcement learning. Firstly, it improves efficiency by prioritizing experiences with higher temporal-difference (TD) errors, directing learning toward impactful updates. Secondly, it corrects for the inherent bias in non-uniform sampling, which maintains theoretical convergence guarantees. Lastly, by balancing the learning process, it ensures stability and prevents over-fitting to specific transitions.

3.2 Hyper-parameters for Importance Sampling

For this implementation, we used the following hyper-parameters:

- **Alpha ($\alpha = 0.6$):** This parameter determines the level of prioritization. Setting α to 0.6 provides a balance between uniform sampling ($\alpha = 0$) and fully prioritized sampling ($\alpha = 1$), ensuring that experiences with significant TD errors are emphasized without neglecting lower-priority ones.
- **Beta Start ($\beta = 0.4$):** The initial β value controls the strength of the importance sampling weights. Starting with a lower value allows the algorithm to manage bias early in training when the agent is still exploring the environment's basic dynamics.
- **Beta Increment ($\beta_{\textbf{increment}} = 0.001$):** Gradual increment of β over time moves the training process toward unbiased sampling ($\beta = 1$). This ensures that as the agent converges, the correction for sampling bias strengthens, promoting stability and convergence.

These hyper-parameters work together to create a balanced approach to prioritized experience replay, ensuring efficient learning while maintaining stability throughout the training process.

3.3 Results: Number of Steps by Test Rewards

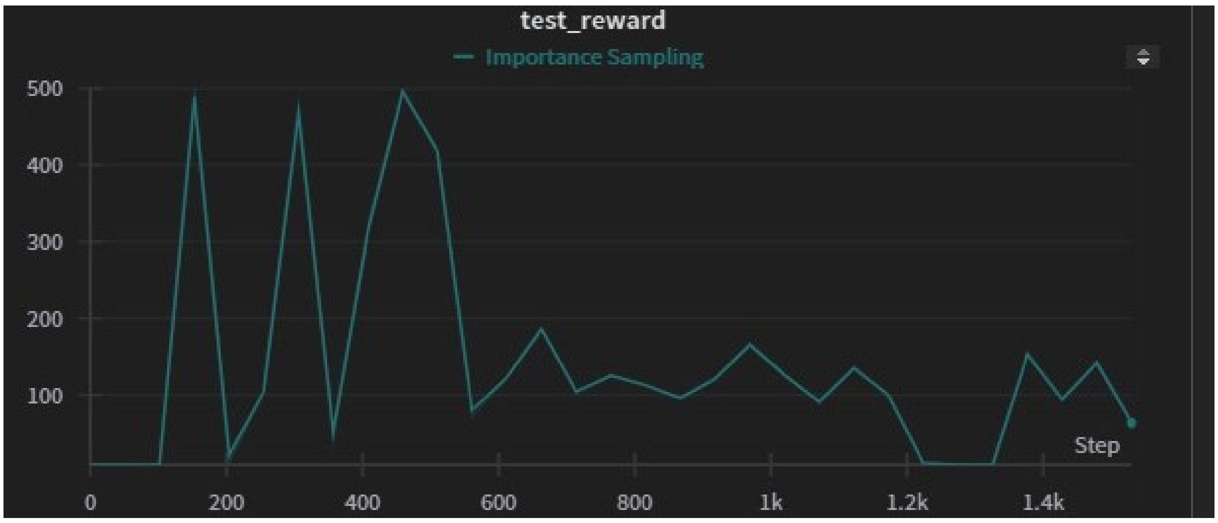

Figure 10: Test rewards over training steps for the agent using importance sampling.

As shown in Figure 10, the agent employing importance sampling demonstrates rapid convergence to near-perfect test rewards, achieving this level of performance after just 150 training episodes. The prioritization of experiences with high TD errors allowed the agent to focus on the most informative transitions, leading to faster learning compared to the baseline agent without importance sampling.

3.4 Results: Loss During Training

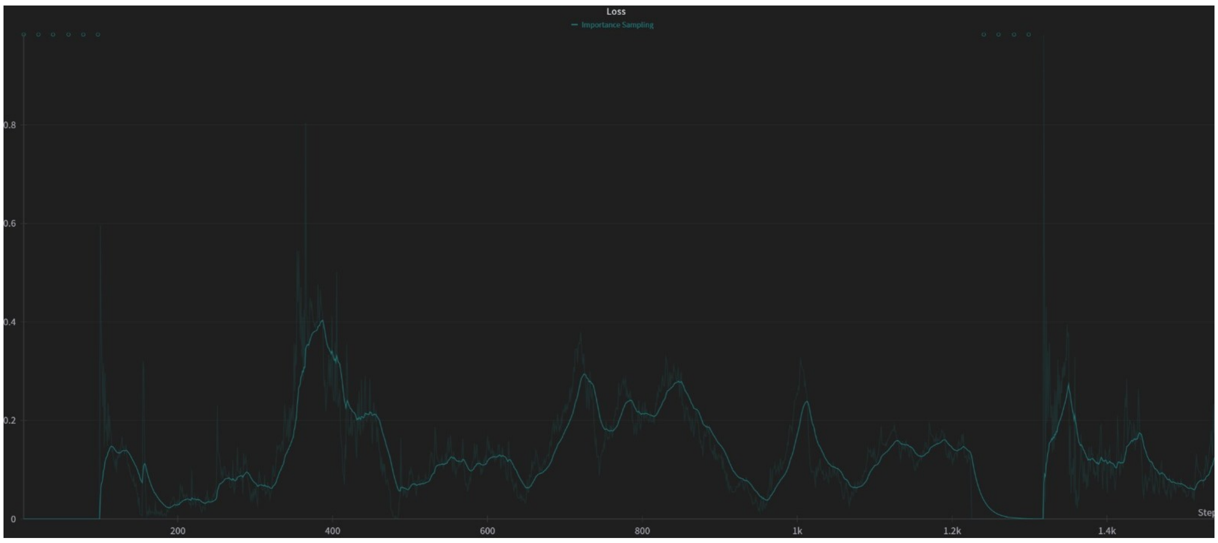

Figure 11: Loss curve during training with importance sampling.

The loss curve, depicted in Figure 11, shows a steady and consistent decline over the training process. The use of importance sampling smooths out the learning process by focusing updates on critical transitions while gradually incorporating less prioritized experiences. This prevents instability in the loss curve, which is often observed in standard uniform sampling.

3.5 Results: Rewards Per Episode

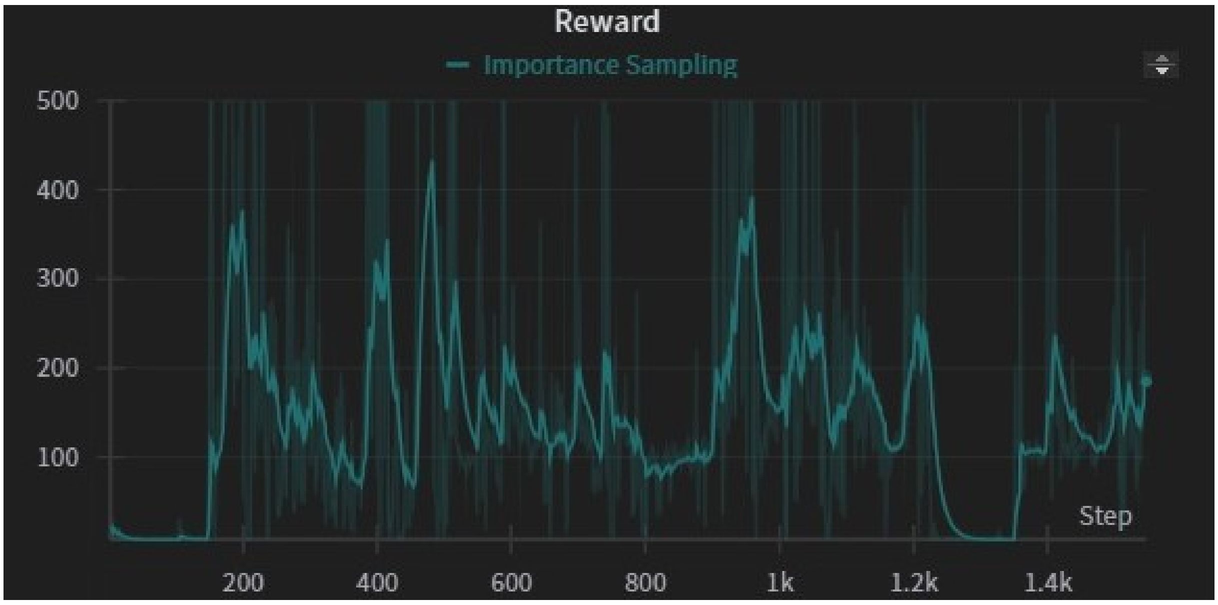

Figure 12: Rewards per episode during training, showing variability early on and stabilization with importance sampling.

Figure 12 highlights the consistent and stable growth in rewards per episode for the agent using importance sampling. Early episodes show high variability as the agent explores the environment, but the rewards quickly stabilize as the agent exploits its learned policy. The gradual increase in β over time ensures a smooth transition from biased sampling to unbiased sampling, which contributes to this stability.

3.6 Comparison to Baseline

Overall, the agent using importance sampling achieved faster convergence compared to the baseline agent without importance sampling, reaching test rewards above 475 in 50 fewer episodes. Additionally, the reward and loss plots demonstrate a much steadier learning curve, highlighting the stability brought by importance sampling. These results confirm that the chosen hyper-parameters and prioritization mechanism improved learning efficiency and stability.

4 Conclusion

This report examined key aspects of reinforcement learning, including model-free methods, hyperparameter tuning, and sampling strategies. A 5-layer network outperformed a 3-layer network, while importance sampling improved efficiency and stability. Critical hyperparameters like discount factor and epsilon decay were tuned to optimize performance.

To reproduce results, run:

```
python [script_name].py
```

Replace `[script_name]` with the relevant file name. This highlights the importance of balancing exploration, exploitation, and targeted updates in reinforcement learning.

Thanks for reading!