HW4: Q-Learning in GridWorld

Lubah

December 2024

Q1: Exploration Policy Analysis and Convergence

For the following questions, the convergence tables can be found on the juptyr notebook

1. Epsilon-Greedy Exploration

The performance of the agent was evaluated using three different epsilon values in an epsilon-greedy exploration strategy. The results are as follows:

- **Epsilon = 0.1:** The agent converged in **52 episodes**.

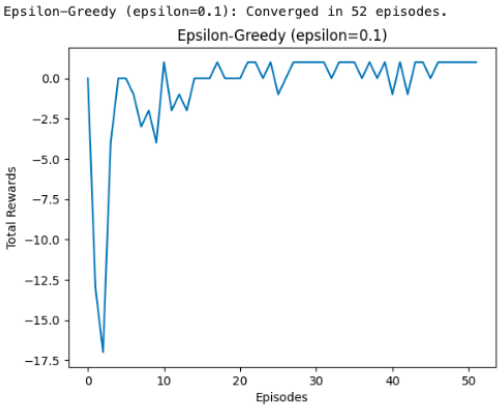

Figure 1: Epsilon-Greedy Exploration with $\epsilon = 0.1$. Convergence achieved in 52 episodes.

- **Epsilon = 0.2:** The agent converged in **108 episodes**.

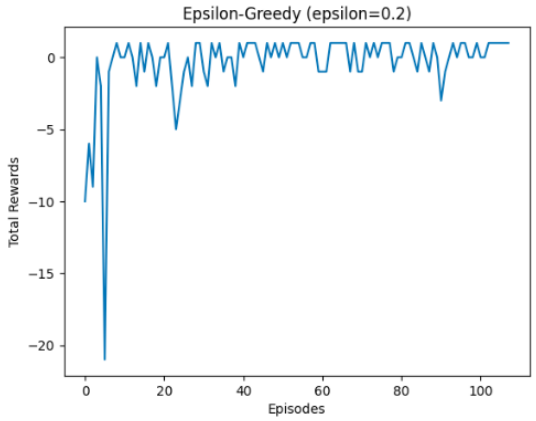

Figure 2: Epsilon-Greedy Exploration with $\epsilon = 0.2$. Convergence achieved in 108 episodes.

- **Epsilon = 0.3:** The agent converged in **128 episodes**.

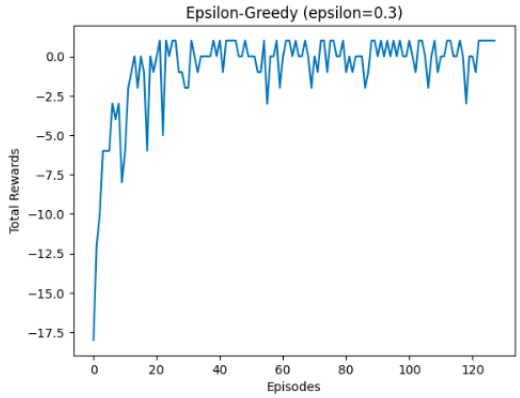

Figure 3: Epsilon-Greedy Exploration with $\epsilon = 0.3$. Convergence achieved in 128 episodes.

Observations: Lower epsilon values ($\epsilon = 0.1$) allowed the agent to exploit known information more effectively, resulting in faster convergence. Higher epsilon values ($\epsilon = 0.3$) encouraged more exploration, delaying convergence but ensuring a more robust policy.

2. Boltzmann Exploration

Boltzmann exploration was tested with an initial temperature $T = 100$. Despite the agent's efforts, no convergence was observed within the allowed number of

episodes.

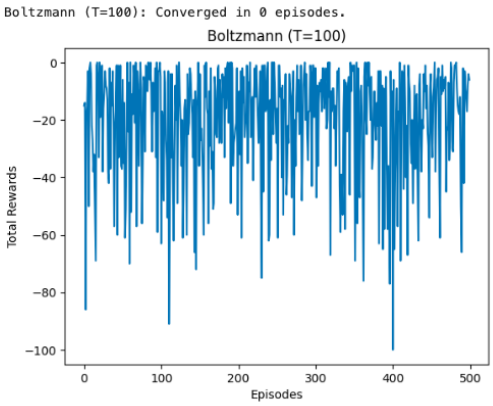

Figure 4: Boltzmann Exploration with $T = 100$. No convergence detected.

Reasoning: The lack of convergence in the Boltzmann exploration case can be attributed to:

- **Insufficient Temperature Decay:** The temperature T determines the exploration-exploitation trade-off. With $T = 100$, the agent prioritized exploration, making it challenging to settle on a stable policy. A faster decay rate or lower initial temperature may have helped the agent focus more on exploitation.
- **State Space Complexity:** The large state-action space of GridWorld increases the difficulty of achieving convergence when the exploration strategy remains too stochastic for too long.

0.1 Q2

The following implementations were applied to CNN architecture and the training: The batch size for training was set to 70, slightly deviating from standard sizes like 32 or 64. By using a batch size of 70, we ensured that the available hardware resources were utilized effectively while maintaining a smooth optimization process, as larger batch sizes stabilize gradient updates and reduce noise in the learning process.

To improve training stability and enhance generalization, batch normalization was introduced after each convolutional layer. Batch normalization works by normalizing the inputs to each layer, reducing internal covariate shifts and making the training process less sensitive to changes in hyperparameters. This modification allowed the network to use higher learning rates and accelerated convergence, which is a widely adopted best practice in modern deep learning architectures.

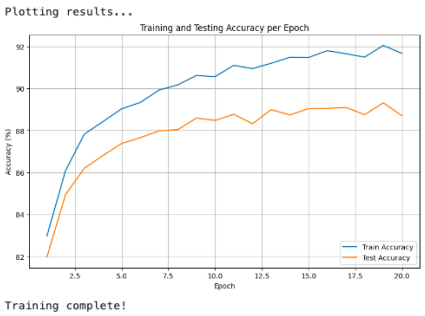

Figure 5: Enter Caption

Random horizontal flipping was applied as a data augmentation technique to diversify the training dataset. Data augmentation artificially increases the variability of the training data, helping the model avoid overfitting. This step was particularly valuable for the Fashion MNIST dataset, which contains a fixed set of grayscale images with limited variability.

The optimizer used was Adam, chosen for its computational efficiency and adaptive learning rates. With a learning rate of 3×1043 \times 10

The training duration was extended from the suggested 10 epochs to 20 epochs. This additional training allowed the model to refine its feature extraction capabilities and reduce underfitting, leading to higher testing accuracy. By providing more time for the network to adjust its parameters, the extended training duration resulted in a more accurate and reliable classification performance.

0.2 Q3

This paper highlights "technical debt" in ML systems, emphasizing the hidden maintenance costs and long-term challenges arising from their unique characteristics. The paper argues that neglected maintenance costs arise from an excessive focus on improving model accuracy, often at the expense of system reliability and maintainability. Over time, these hidden issues compound, resulting in inefficiencies, system breakdowns, and escalating operational costs. To address these challenges, the authors recommend adopting modular system design to reduce component entanglement and implementing automation for comprehensive monitoring and testing. They further advocate for simplifying data pipelines, maintaining clear documentation, and standardizing interfaces to improve scalability and ease of maintenance. Finally, striking a balance between model complexity and operational simplicity is essential to ensure the long-term robustness and sustainability of ML systems. These strategies collectively empower practitioners to confront hidden technical debt and develop resilient, efficient machine learning systems.

The key challenges covered includes data dependency, where changes in data

quality, schema, or distribution can cascade into system failures. Feedback loops, where predictions influence future inputs, can introduce bias or instability. Additionally, ML pipelines often evolve into "pipeline jungles," characterized by overly complex and poorly documented processes.Unlike traditional software systems, ML systems are heavily dependent on data, making them more prone to instability and harder to maintain. This complexity is compounded by glue code—ad hoc scripts connecting various components—and configuration debt, where intricate and unmanageable settings increase the risk of errors. These challenges create tightly coupled, brittle systems that are difficult to update or debug, exacerbating the accumulation of technical debt!

0.3 Q4

The previous paper discusses technical debt in ML systems; this paper introduces the ML Test Score, a framework designed to evaluate the production readiness of ML systems. It addresses technical debt by providing a comprehensive checklist that helps teams ensure their systems are reliable, scalable, and maintainable. The framework focuses on improving the design, deployment, and maintenance of ML systems by systematically addressing common weak points.

The checklist is organized into five key areas. Data tests focus on maintaining the quality and stability of input data, identifying issues like schema mismatches or data skews that can disrupt system performance. Model tests assess robustness, fairness, and generalization, helping teams identify biases or overfitting. Infrastructure tests ensure scalability, version control, and reproducibility, enabling consistent performance in different environments. Monitoring tests aim to detect data drift, anomalies, and prediction errors in real time, allowing teams to fix issues promptly. Finally, business and use-case tests confirm that the system aligns with user needs, business objectives, and compliance standards, ensuring it delivers practical value.

Beyond the checklist, the paper highlights broader challenges ML systems face in production. Many current practices emphasize performance metrics over comprehensive testing, leading to gaps in reliability and maintainability. The ML Test Score framework encourages systematic testing and accountability, helping teams proactively identify weaknesses and reduce technical debt. This structured approach allows teams to prioritize key areas for improvement and make iterative enhancements over time.

The authors also emphasize the framework's role in cutting long-term operational costs. By addressing potential issues early and improving the overall durability of ML systems, teams can avoid expensive failures and inefficiencies. As ML systems become more integral to real-world applications, the ML Test Score provides a timely and practical guide to building scalable, reliable, and production-ready systems.