CptS 570 Machine Learning

Homework - 4

Submitted by
Athul Jose P
11867566

School of Electrical Engineering and Computer Science
Washington State University

Contents

1 Q1: Q Learning Implementation

1.1 1: Epsilon Greedy

Q-learning has been implemented in a Grid World environment. The results for the epsilon-greedy strategy are presented below. Among the 284 Q-values generated, a subset is shown in the table 2 for reference. Additionally, epsilon-greedy exploration was performed with epsilon values of 0.1, 0.2, and 0.3. The corresponding plots illustrate the relationship between the number of steps and episodes for each configuration.

ϵ	0.1	0.2	0.3
Number of iterations	2854	2565	2670

Table 1: Number of iterations for various ϵ

Table 1 shows the number of iterations required for a reinforcement learning algorithm to converge under different exploration rates ($\epsilon = 0.1, 0.2, 0.3$). The exploration rate (ϵ) determines the likelihood of taking random actions to explore the environment, with higher values promoting more exploration. The results indicate that a lower exploration rate ($\epsilon = 0.1$) requires more iterations (2854) to converge compared to higher rates, suggesting a balance between exploration and exploitation affects convergence speed.

State	**Action**	**Q-Value (0.1)**	**Q-Value (0.2)**	**Q-Value (0.3)**
(0, 0)	down	-0.649669901	-0.648246707	-0.651488457
(0, 0)	right	-0.642472066	-0.643021839	-0.641492907
(0, 1)	down	-0.612381504	-0.611337794	-0.613816456
(0, 1)	left	-0.621618719	-0.623777322	-0.64170311
(0, 1)	right	-0.596126721	-0.597324167	-0.594577917
(0, 2)	down	-0.571389610	-0.568190143	-0.568050145
(0, 2)	left	-0.579507195	-0.582094825	-0.599464534
(0, 2)	right	-0.541213393	-0.542962977	-0.53983894
(0, 3)	down	-0.479172282	-0.481528934	-0.47807679
(0, 3)	left	-0.541012187	-0.548723726	-0.564816372

Table 2: Q values for first 10 state action pairs

This table 2 shows Q-values for different state-action pairs in a reinforcement learning environment across three learning rates ($\alpha = 0.1, 0.2, 0.3$). States are represented as tuples (e.g., (0, 0)), and actions like "down" or "right" indicate possible moves. Q-values reflect the expected reward or penalty of an action, considering future rewards, with lower values indicating less favorable outcomes. The table highlights how learning rates affect Q-value updates, with higher rates ($\alpha = 0.3$) adapting more quickly. For instance, in state (0, 0), the "down" action consistently yields similar Q-values, suggesting stable outcomes across learning rates.

1.2 2: Boltzman Exploration

Table 3 provides the number of iterations required for a reinforcement learning algorithm to converge using Boltzmann exploration with a decay rate of 0.1. Boltzmann exploration is a strategy where actions are chosen probabilistically based on their Q-values, and the decay rate controls how quickly exploration decreases over time. With a decay rate of 0.1, the algorithm converges in 2527 iterations, indicating the influence of a moderate decay rate on the exploration-exploitation balance and convergence speed.

Decay Rate	0.1
Number of iterations	2527

Table 3: Number of iterations Boltzman Exploration

State	**Action**	**Q-Value**
(0, 0)	down	-0.650494678
(0, 0)	right	-0.644352131
(0, 1)	down	-0.613955515
(0, 1)	left	-0.614413458
(0, 1)	right	-0.597617970
(0, 2)	down	-0.574518436
(0, 2)	left	-0.574733939
(0, 2)	right	-0.542410058
(0, 3)	down	-0.480503703
(0, 3)	left	-0.533440747

Table 4: Q values for state action pairs

2 Q2: CNN Implementation

A Convolutional Neural Network (CNN) was implemented using 50% of the dataset for training and 10% for testing. The model was trained with Stochastic Gradient Descent (SGD) as the optimization algorithm over 20 epochs. As shown in the results, both training and testing accuracies begin to stabilize after 10 epochs. The final observed accuracy is 81.18% for the training set and 83.7% for the testing set.

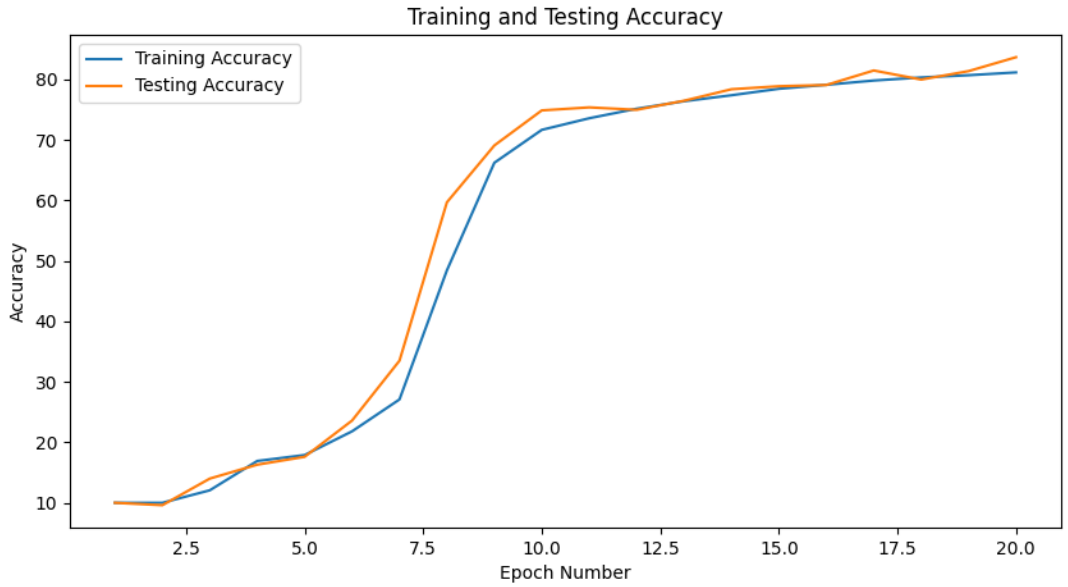

Figure 1: Training and Testing Accuracies

3 Q3: Paper Summary: Hidden Technical Debt in Machine Learning Systems

The paper, Hidden Technical Debt in Machine Learning Systems, sheds light on the unique challenges of maintaining ML systems over time. While developing and deploying ML models is often fast and inexpensive, their long-term maintenance can be costly due to the accumulation of technical debt. This debt arises from ML-specific risk factors that erode abstraction boundaries and create complex interdependencies.

One major challenge is the erosion of abstraction boundaries in ML systems. Unlike traditional software, ML systems rely heavily on external data, making strict encapsulation difficult. This issue is evident in various forms, including entanglement, where ML systems mix signals so thoroughly that changes in one part of the system affect everything else. This is known as the "Changing Anything Changes Everything" (CACE) principle. Mitigation strategies include isolating models, using ensembles, and monitoring prediction behaviors for anomalies. Another issue is correction cascades, where models are modified to solve slightly different problems by adding corrections. This creates dependencies that make future improvements costly. To address this, corrections should be incorporated into the base model or handled by building separate models for new problems. Additionally, undeclared consumers—systems that silently use ML predictions without proper access controls—can create hidden couplings and feedback loops. Implementing strong access controls and service-level agreements can mitigate these risks.

Data dependencies in ML systems also introduce significant challenges. Unlike code dependencies, data dependencies are harder to detect and can lead to instability. For instance, unstable data dependencies occur when input data behavior changes unexpectedly, disrupting model performance. A common mitigation is to version input signals or freeze data mappings to maintain consistency. Underutilized data dependencies, such as legacy, bundled, or correlated features, provide minimal modeling benefits while increasing vulnerability to changes. Regular evaluations, such as leave-one-feature-out tests, can help identify and remove unnecessary features to streamline the system.

Feedback loops further complicate ML system maintenance. These loops occur when live ML systems influence their own behavior, making it challenging to predict future performance. Direct feedback loops arise when models directly influence their training data selection, potentially leading to skewed results. Solutions include randomization and the use of bandit algorithms. Hidden feedback loops, where systems indirectly affect each other, are harder to detect and require careful monitoring and isolation of components.

Several system-level design flaws, termed anti-patterns, contribute to technical debt. For example, glue code, used to integrate general-purpose ML packages with custom systems, often creates inflexible and fragile systems. Wrapping packages in standardized APIs can improve maintainability. Similarly, pipeline jungles, characterized by overly complex data preparation pipelines, make testing and debugging expensive. Holistic pipeline redesigns can significantly reduce long-term costs. Another issue is dead experimental codepaths, where abandoned experimental branches increase system complexity. Regular cleanup of these branches is necessary to maintain system health.

Configuration debt is another source of technical challenges. Large ML systems often have numerous configurable options, such as feature selection, learning settings, and verification methods. These options introduce opportunities for errors and make systems harder to maintain. Transparent configuration management, automated assertions, and version control are essential to mitigate these risks.

Finally, ML systems must adapt to external changes in dynamic environments. Real-time monitoring and automated responses are critical for maintaining system reliability. Key strategies include monitoring prediction bias, enforcing action limits, and tracking upstream processes to ensure data integrity. This proactive approach helps systems adapt to changes effectively while maintaining accuracy.

In conclusion, ML systems accumulate technical debt in multiple dimensions, including data dependencies, feedback loops, and system design. Addressing this debt requires systematic testing, abstraction, and monitoring strategies, as well as cultural changes to prioritize maintainability alongside performance improvements. The paper emphasizes the importance of recognizing these risks and adopting mitigation strategies to build sustainable and reliable ML systems.

4 Q4: Paper Summary: The MLTest Score: A Rubric for ML Production Readiness and Technical Debt Reduction

The paper, The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction, addresses the challenges of deploying machine learning (ML) systems in production environments. Unlike traditional software systems, ML models rely heavily on data and configuration, making it difficult to specify their behavior a priori. To ensure production-readiness and mitigate technical debt, the authors propose a comprehensive rubric comprising 28 actionable tests. These tests are categorized into data, model, infrastructure, and monitoring, offering a structured approach to improving the reliability of ML systems.

Data testing and monitoring form the foundation of the rubric. The authors emphasize the need to define a schema that encodes data expectations, derived from statistical analysis of training data and refined using domain knowledge. This schema automates data checks and ensures validity. Every feature in the system should be evaluated for its predictive utility, computational cost, and compliance with privacy policies. Privacy controls are critical, with measures to prevent sensitive data leaks and ensure user data deletions are propagated through the pipeline. Additionally, training-serving skew must be monitored to address inconsistencies between feature computations during training and inference.

For model testing, the rubric highlights the importance of version control and reproducibility. Proper versioning ensures models are auditable, and reproducibility guarantees consistent behavior when training on the same dataset. Hyperparameter tuning is a key step to uncover hidden issues and enhance model performance. The rubric also emphasizes the need to understand how model staleness impacts prediction quality, enabling teams to determine appropriate update frequencies. To promote fairness, tests should identify biases in the model and ensure inclusivity across diverse user groups and scenarios.

Infrastructure testing is another critical component of the rubric. End-to-end integration testing validates the entire ML pipeline, from data preprocessing and feature engineering to training and serving. Canary testing is recommended for gradually introducing models into production, enabling teams to identify potential issues like incompatibilities or instability. Rollback mechanisms are essential for quickly reverting to previous stable versions when problems arise. Debuggability is also emphasized, with tools for step-by-step examination of model computations on individual examples to diagnose complex errors.

The rubric underscores the importance of continuous system monitoring. Automated dashboards and alerts track key metrics such as data quality, model performance, and computational efficiency.

Effective monitoring includes detecting training-serving skew, tracking model aging, and measuring system performance metrics like latency and throughput. By monitoring these aspects, teams can ensure the system remains reliable, scalable, and responsive to changes in the environment.

The authors advocate for a cultural shift to reduce technical debt and improve ML production practices. The ML Test Score incentivizes teams to adopt rigorous testing and monitoring by providing a quantifiable measure of production-readiness. Teams earn points for implementing and automating tests, with the minimum score across four sections determining the overall readiness. This scoring system encourages even experienced teams to systematically address blind spots and prioritize areas for improvement.

In conclusion, the proposed rubric offers a robust framework for ensuring the reliability, reproducibility, and maintainability of ML systems in production. By combining actionable testing strategies, comprehensive monitoring protocols, and cultural incentives, the authors provide a practical roadmap for addressing the unique challenges of ML deployment. This holistic approach helps minimize technical debt and optimize the long-term performance of ML systems.