Curriculum Learning for RL Agents in a Pac-Man Environment

EN.705.741 - Reinforcement Learning

Matthew Renze
Johns Hopkins University
mrenze1@jhu.edu

Simran Shinh
Johns Hopkins University
sshinh2@jhu.edu

Abstract

In this research project, we trained a series of reinforcement learning (RL) agents to play the game of Pac-Man using curriculum learning (CL). First, we created a custom version of the Pac-Man game to simplify the environment's dynamics. Next, we trained four RL agents with and without CL. Then, we ran each treatment for 100 episodes and analyzed the results. Our analysis revealed that CL improved the performance of the SARSA and Q-learning agents but failed to improve the Approximate Q-learning and Deep Q-learning agents. All code and data are available on GitHub at: https://github.com/matthewrenze/jhu-reinforcement-learning/

1 Introduction

1.1 Background

As humans, we learn to perform complex tasks by first learning a set of simple foundational tasks. For example, we learn to walk before we learn how to run. We learn to read individual words before we learn to read sentences. And, we learn how to grasp objects before we learn fine-motor control.

Our society structures our learning using a set of curricula that progress from primary school, through secondary school, and into university. With each stage of learning, we transfer our previously learned skills to progressively more complex problems. This allows us to progress from performing simple tasks as children to mastering complex tasks as adults.

With this in mind, our research project attempted to answer the question: can reinforcement learning (RL) agents *also* benefit from a curriculum?

1.2 Pac-Man

Pac-Man is a classic arcade game created by Namco in 1980 [1]. As the player, you control Pac-Man, a yellow circle with a wedge-shaped mouth. Your objective as Pac-Man is to navigate a maze and eat all of the 240 dots, called *pac-dots*, while avoiding the four ghosts that roam the maze (see Figure 1).

The maze contains Pac-Man, several pac-dots, four power pellets, and four roaming ghosts. The ghosts are initially contained in the *ghost house* at the center of the maze and are released at timed intervals. The maze has no dead ends; however, there are two tunnels on either side that allow Pac-Man to warp from one side of the screen to the other [2].

When Pac-Man consumes a power pellet, he becomes invincible to the ghosts and can eat them. The ghosts immediately transition into *frightened* mode and try to evade Pac-Man. If eaten, Pac-Man will receive a bonus, and the ghost will re-spawn in the ghost house.

The ghosts are named Blinky, Pinky, Inky, and Clyde. Each ghost has its own unique color and attack pattern. For example, Blinky pursues Pac-Man by directly targeting his current location, while Pinky attempts to ambush Pac-Man by targeting four tiles ahead of his current location [2, 3].

Ghosts transition through three modes. In *chase* mode, they pursue Pac-Man using their specific attack strategy. In *scatter* mode, they return to their respective home corners. In *frightened* mode, they reverse direction to run away from Pac-Man and then choose a random direction at each intersection [2, 3].

For our research project, we created a custom implementation of Pac-Man that simplifies gameplay by converting the original game into a discrete environment (see Figure 2). Our simplified game uses a squarified version of the first level of the original game.

In addition, it also incorporates several other simplifications to keep the complexity low. For example, it only includes the first level of the original game, Pac-Man has only one life per episode, and there are no bonuses. These simplifications allow us to focus on the more relevant aspects of RL and CL.

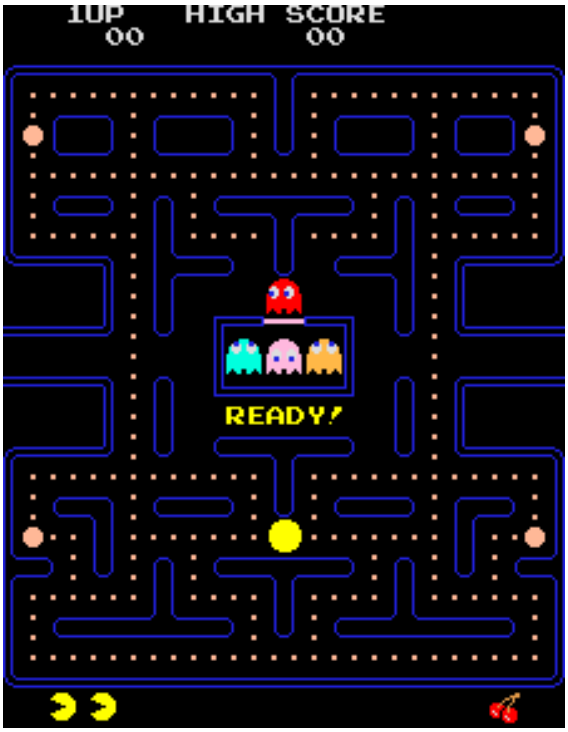

Figure 1: The original Pac-Man game.

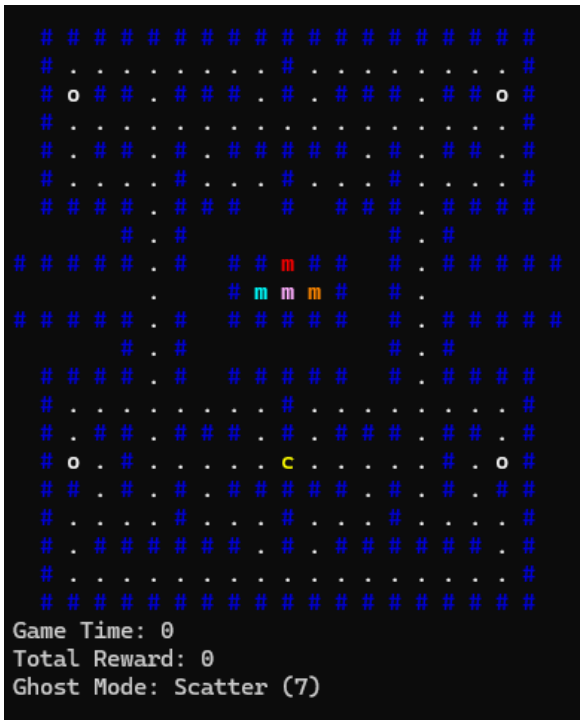

Figure 2: Our modified Pac-Man game.

1.3 Curriculum Learning

Curriculum learning (CL) is a strategy for teaching RL agents how to complete complex tasks by first allowing them to master simple tasks. The curriculum gradually presents the agent with progressively more complicated tasks to allow it to learn incrementally. This approach makes the learning process more efficient, stable, and transferable [4].

CL typically involves structuring the learning process into a series of levels. The agent starts with the simplest task until it reaches a specified level of proficiency. Once the agent has achieved proficiency, it moves onto the next level, where it learns a more difficult task. The curriculum concludes when the agent has mastered all of the tasks.

CL provides several benefits for RL. First, CL can be more efficient than standard RL training processes. Second, a well-structured curriculum can make the learning process more stable. Finally, skills can be transferred from previous levels, allowing the agent to quickly adapt to novel situations [4].

For our research project, we created a series of nine CL levels to incrementally train the agent before testing it on the first level of Pac-Man. Each level introduces new concepts and tasks for the agent, starting with basic navigation, progressing through ghost avoidance, and ending with navigating complex mazes with dynamic ghosts (see Figure 3).

1.4 Hypothesis

Based on our prior understanding of CL in RL, we developed the following hypothesis:

- **Null Hypothesis** (H_0) - CL does not improve the performance of an RL agent compared to a baseline (i.e., non-curriculum) agent in terms of total reward per episode.
- **Alternate Hypothesis** (H_1) - CL improves the performance of an RL agent compared to a baseline agent.

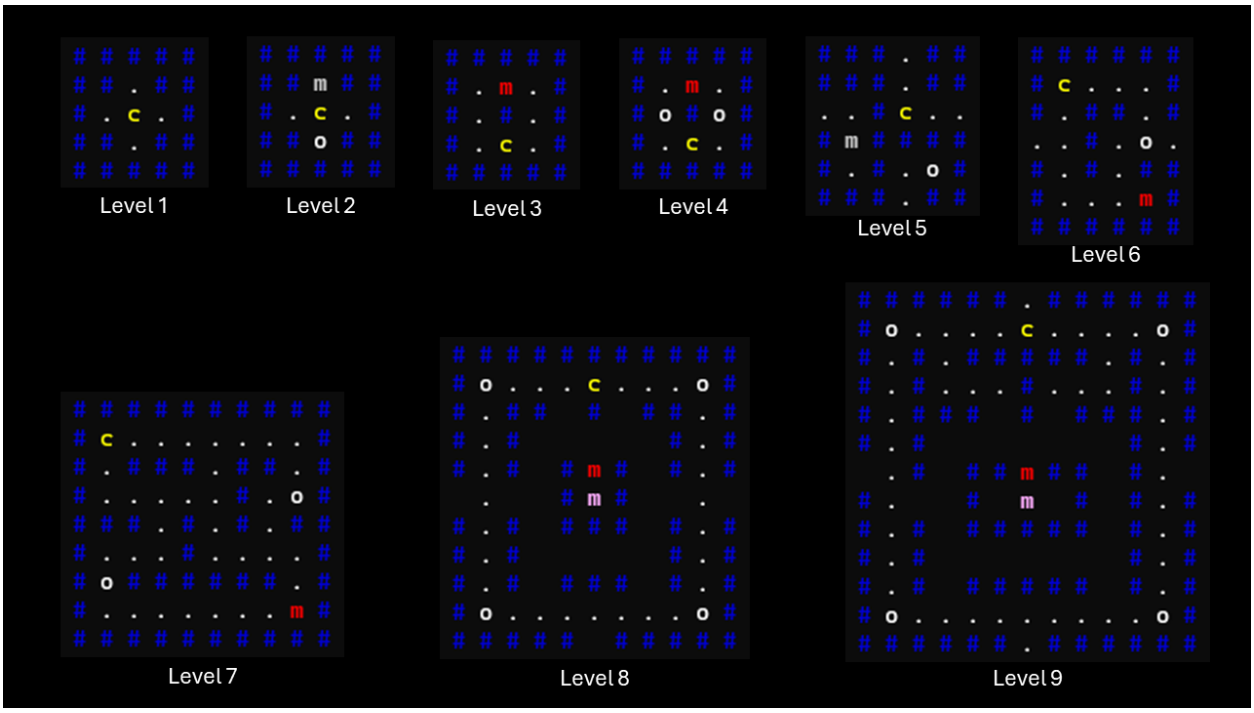

Figure 3: Our nine curriculum levels in order of complexity.

To test our hypothesis, we ran each fully-trained agent for 100 episodes. We compared the means of the total reward obtained from each episode. To assess the statistical significance of our results, we evaluated the mean reward at a significance level (α) of 0.05.

1.5 Prior Literature

Using RL to master video games is not a new endeavor. Atari games, in particular, have been the subject of prior research. Notably, Mnih et al. demonstrated that deep RL can be used to successfully play multiple Atari games [5].

Regarding the game of Pac-Man, Bom et al. used feature extraction to provide seven inputs to a neural network. These features were used to train a high-performing agent, helping to address the challenge of scaling large dynamic environments for RL. Their work also showed that the policy learned was transferable to an unseen maze [6].

Video game emulators have been developed to support coursework in artificial intelligence. These emulators have also been used to perform AI research [7].

CL was first used to assist automated agents in the 1990s [4]. The primary applications were grammar learning and robotics control problems [8–10]. In the late 2000s, CL was applied to supervised learning for classification problems [11].

The first application of CL to RL occurred around the same time [4]. The primary focus was on exploring the use of transfer learning [12, 13]. Since then, there have been more recent attempts to automatically generate curricula for deep RL [14].

1.6 Contributions

Our research builds upon existing CL literature by providing additional insight into the advantages and disadvantages of using CL to train RL agents. In this paper, we have documented the problems we encountered, our solutions, and other valuable information that we have learned over the course of this research project.

In addition, our project provides students with a simple, discrete, and easy-to-use Pac-Man environment for learning RL and CL concepts. We've made this environment available as a free, open-source project on GitHub[1].

[1]The repository for this project can be found at: https://github.com/matthewrenze/jhu-reinforcement-learning/

2 Methods

2.1 Data

To represent each game level, we created a set of 10 plain-text files that represent the elements of each level. Each character in the file represents one of ten possible tile types. These tile types include: [empty, wall, Pac-Man, pac-dot, power pellet, static ghost, Blinky, Pinky, Inky, and Clyde].

We represent each tile using a unique ASCII character. The tile characters are [(space), "#", "c", ".", "o", "s", "b", "p", "i", and "y"] for each respective tile type. Once loaded into memory, these tiles are represented as an integer from 0-9 stored in a 2D array.

We created a 21 $\times$ 21 map to represent the first level of the original Pac-Man game. This map is a squarified approximation of the original level. We refer to this level in our Pac-Man environment as Level 10, since it is the final level of the curriculum (see Figures 4 and 5).

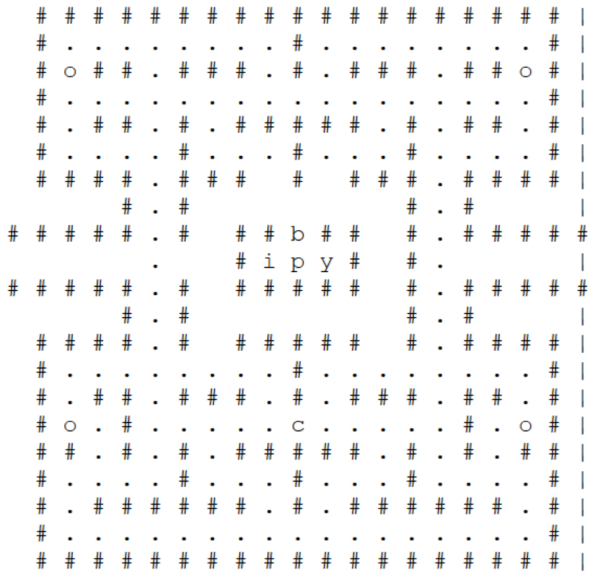

Figure 4: The plain-text map file for Level 10.

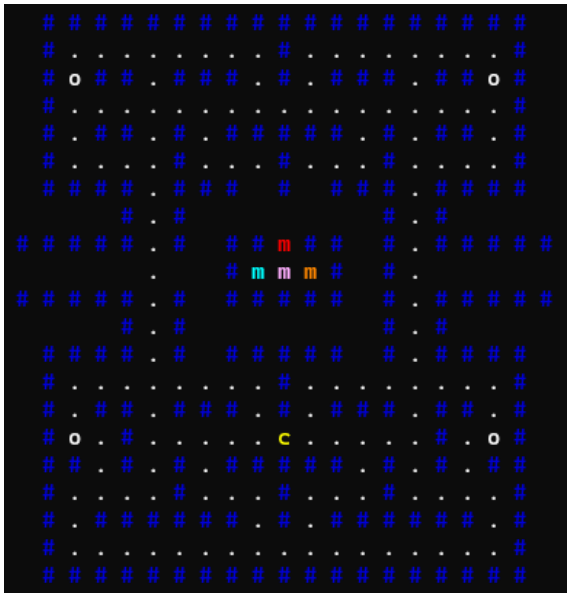

Figure 5: The game rendering of Level 10.

We also created a curriculum of nine game levels (i.e., game maps) of incrementally increasing complexity. The first level teaches the agent how to navigate and eat pellets. The second level introduces a static ghost. Later game levels add more game elements, dynamic ghosts, and navigational complexity (see Figure 3).

To maximize the coverage of all possible states the agent might encounter, we programmatically rotated and flipped maps during each level of the curriculum. This process takes a single curriculum level and transforms it into the six possible orientations of the tiles in that level.

2.2 Agents

First, we implemented a SARSA agent [15]. As input, our SARSA agent uses the four nearest neighboring tiles in the cardinal directions (i.e., up, down, left, and right). We encode the four tiles as an integer from 0 to 10, and we prefix a single 0 or 1 for the *is_invincible* state. This produces a total of 20,000 possible states in the q-table.

Next, we implemented a Q-learning (QL) agent [15]. It also uses the four nearest neighboring tiles and the same feature encoding scheme.

Then, we implemented an Approximate Q-learning (AQL) agent [15]. For AQL, the q-value for each state and action pair is computed by a linear combination of several features, which are defined over the state and action pairs. The set of features used to train the agent was determined through a process of feature engineering. Various subsets of features were evaluated to determine which combination yielded the best performance (in terms of total reward). The states for the AQL agent are simply the coordinates of each tile in the grid, and each feature is calculated based on the value of that feature if action a is taken from tile s.

Finally, we implemented a Deep Q-Network (DQN) agent [5]. Our DQN agent uses a 9×9 receptive field of its 80 nearest neighboring tiles as input. This receptive field is then flattened into a single 1D array of length 810. We also encoded and appended the *is-invincible* state and the *ghost mode* to the input for a total of 814 input values.

2.3 Feature Engineering

For the AQL agent, many subsets of possible features were first evaluated before performing hyperparameter tuning. The set of features developed included the minimum distances to the closest pac-dot, ghost, and power pellet, as well as the number of frightened and active ghosts 1 and 2 steps away.

A "food-focus" activation feature was developed to indicate whether Pac-Man can focus on eating pac-dots [7]. This feature returned a value of 1 if a pac-dot was in the next space and no active ghosts were within 1 or 2 spaces; otherwise, it was omitted from calculations for weight and q-value updates.

A "safety-mode" activation feature was also included to indicate whether Pac-Man was safe (due to invincibility and no surrounding ghosts). Finally, a bias feature, which always returned a value of 1, was included.

The total amount of pac-dots remaining in the maze above, below, to the left of, and to the right of Pac-Man were also explored as features, but were ultimately omitted.

Figure 9 in the Appendix depicts the impact of the "food-focus" feature, which is feature 12 in the legend. The maximum reward achievable by the AQL agent is greatly impacted by this one feature.

Ultimately, the subset of features yielding the best results were bias, distance to the closest ghost, the number of active and scared ghosts 1 and 2 steps away, and the food-focus feature.

2.4 Hyperparameter Tuning

First, we performed a sweep of the standard hyperparameters for the four agents. These hyperparameters included the learning rate (α), the discount factor (γ), and the exploration rate (ϵ). See Figure 10 in the Appendix for an example of one of the eight hyperparameter sweeps.

Next, we performed a sweep of the DQN learning rate (α_2) using values of 0.01, 0.001, and 0.0001. The top-performing value was 0.001. However, we also explored the space of hidden-layer depth and nodes per hidden layer. The NN architectures we evaluated included layer depths of 1 to 3 layers and nodes per layer from 16 to 1,024. Surprisingly, the best-performing NN architecture was a single-layer NN with 128 nodes in the hidden layer.

We also evaluated the performance of the DQN agent based on the number of nearest neighbors in its $n \times n$ receptive field ranging from 3×3 to 11×11. A receptive field of 9×9 performed best.

Finally, we fine-tuned the number of steps the agent spent training on each level of the curriculum. We evaluated steps-per-level in the range of 100 to 10,000 in base-10 increments and then further refined in increments of 100. The optimal value occurred at 200 training steps per level.

2.5 Agent Training

We trained all eight agents for 1 million steps using the optimal hyperparameters. At the end of the training phase, we saved the final models. This provided us with learning curves for each agent and a persistent model to use for evaluation. See Figure 11 in the Appendix for the learning curves.

2.6 Agent Evaluation

To evaluate our models, we captured and analyzed the following metrics over 100 test episodes:

- **Reward** - the mean of the total rewards collected per episode.
- **Runtime** - the mean runtime duration per episode (in milliseconds).
- **Visited States** - the mean of the percentage of states (i.e., unique tiles) visited per episode.

We used a set of paired t-tests to compare the two treatments (i.e., baseline vs. curriculum) for each agent. We chose a paired t-test because the total-reward-by-agent data appears to be (roughly) normally distributed (see Figure 12).

3 Results

3.1 Performance by Curriculum

In terms of performance by curriculum, we observed that the SARSA agent and the Q-learning agent outperformed their baseline using the curriculum ($p < 0.00001$ for both agents). The AQL and DQN agents both performed worse when given a curriculum for their first 1,800 training steps ($p =$ N/A and $p < 0.0001$, respectively)[2].

Based on our final results, we reject the null hypothesis (H_0) for the SARSA and Q-learning agents in favor of the alternate hypothesis (H_1). However, for the AQL and DQN agents, we failed to reject the null hypothesis (H_0). See Figure 6 and Table 1 for a comparison of the performance by agent and treatment.

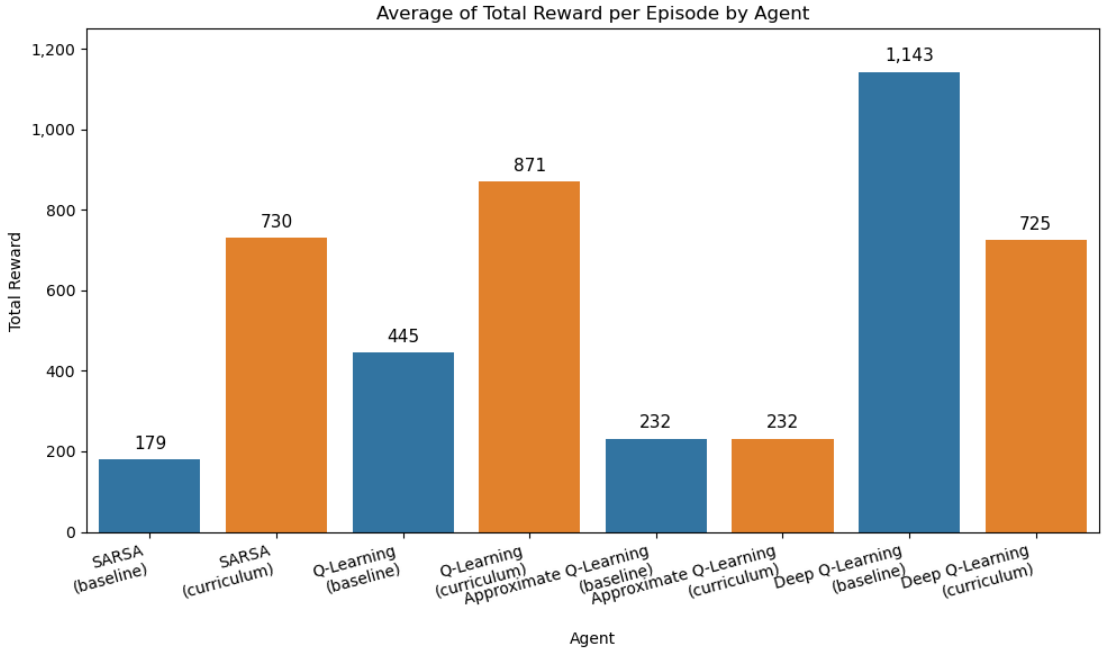

Figure 6: Comparison of average total reward by agent and treatment.

Agent	Baseline	Curriculum	Difference	t-Statistic	p-Value
SARSA	179.3	730.5	551.2	-11.972	< 0.00001
Q-Learning (QL)	445.4	870.6	425.2	-9.985	< 0.00001
Approx. Q-Learning (AQL)	231.8	231.8	0.0	N/A	N/A
Deep Q-Network (DQN)	1142.6	725.4	-417.2	7.716	< 0.00001

Table 1: Comparison of average total reward by agent and treatment

However, during multiple phases of development, we observed results that were different from our final results. So, these results appear to be sensitive to various aspects of the experiment, including the specific agents, environments, curricula, and hyperparameters.

3.2 Performance by Agents

In terms of performance by agent type, the baseline DQN agent outperformed all other agents and treatments (see Figure 6). When comparing the DQN agent against the next highest-performing agent (i.e., the curriculum-trained Q-Learning agent), we saw a difference in the means of 272 points ($p < 0.00001$).

[2]The AQL agent had identical scores for both treatments because the linear model's parameters converged to similar values. In addition, we reset the random seed for each evaluation run, so the two AQL agents chose the same actions at each of the randomly initialized (but pair-wise identical) episodes.

3.3 Performance by States Visited

To further support our findings, we compared our agents by the average number of unique states (i.e., tiles) visited per episode. We observed that the baseline DQN agent performed best, with an average of 20.96 states visited per episode. The Q-Learning agent was next, followed by SARSA, and the AQL agent (see Figure 7).

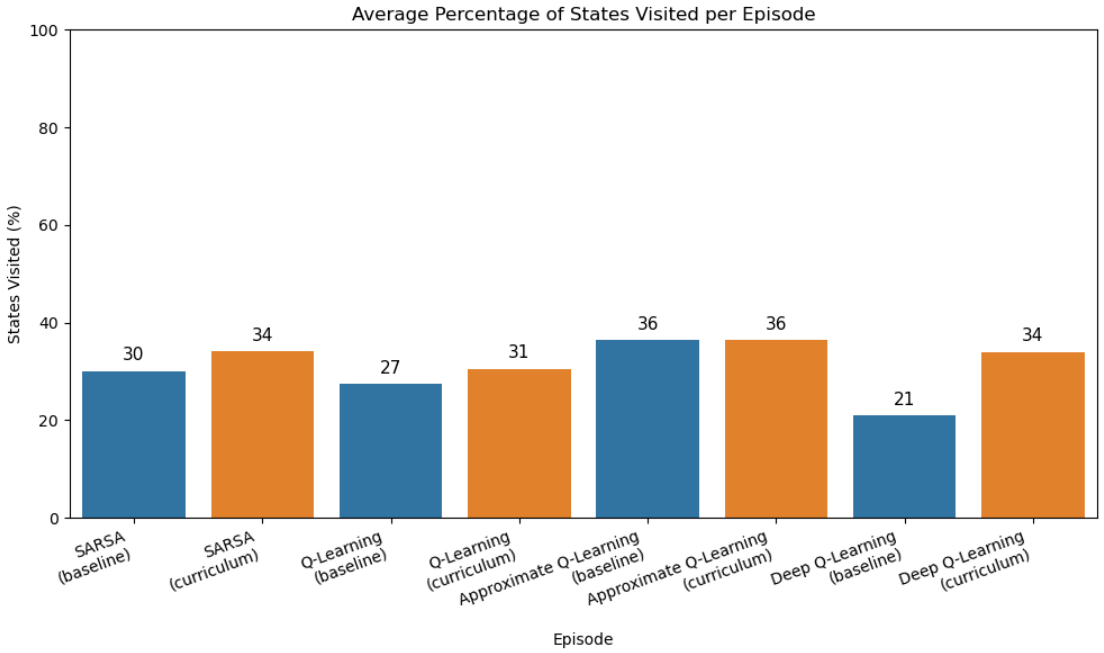

Figure 7: Comparison of average number of states visited by agent and curriculum.

3.4 Performance by Runtime

Finally, we compared the runtime performance in terms of the agents' average runtime duration in milliseconds per episode. The SARSA and Q-Learning agents performed best with an average runtime of 0.325 ms and 0.340 ms per episode (respectively). The DQN agent came in third at 0.917 ms, and finally, the AQL agent with an average runtime duration of 1.314 ms per episode (see Figure 8).

4 Discussion

4.1 Limitations

There were several limitations to this research project.

First, when fine-tuning hyperparameters, we could only train our agents for 100,000 steps per hyperparameter combination. Training all of the hyperparameter combinations took multiple hours, even when parallelized. Due to this limitation, we only used 100,000 steps to predict the long-term trajectory of the agents' performance.

Second, we selected the best hyperparameters based on a visual analysis. We inspected the lift and trajectory of each hyperparameter combination to approximate its long-term trajectory. However, a more rigorous statistical analysis (e.g., Area Under the Curve (AUC) or time-series forecasting) might have led to better hyperparameter selection.

Third, the interaction between hyperparameters introduced significant complexity in the fine-tuning process. Changing one hyperparameter affected several other hyperparameters. As a result, we used an iterative process to try to hone in on the best set of hyperparameters.

Fourth, due to the long training time, we were only able to train our final agents for 1 million steps. A full training run of all eight agents (in parallel) to 1 million training steps also took over six hours. So, we limited our final training

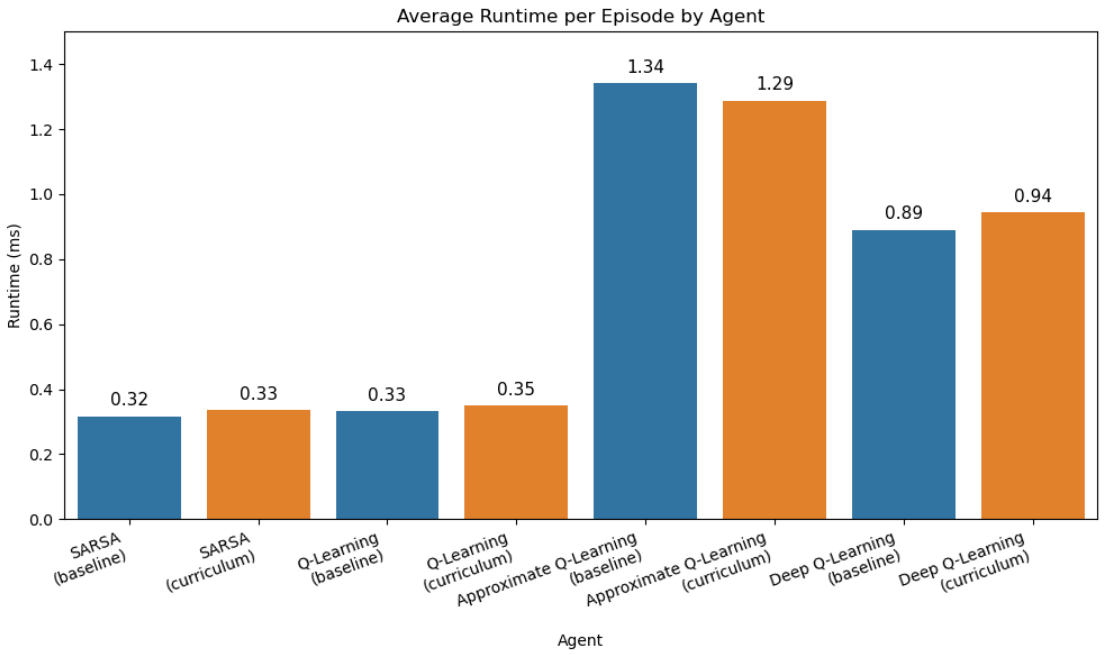

Figure 8: Comparison of average runtime per episode by agent and curriculum.

duration to keep the training process tractable on consumer-grade hardware, but some agents (such as the DQN) may have further improved with additional training.

Finally, the AQL agent had a limited number of parameters. These parameters quickly converged to their optimal values. Once converged, the agent was unable to learn new information from subsequent training steps. As a result, the AQL agent would need additional parameters that better characterize the state space or non-linear function approximation in order to improve performance.

4.2 Implications

This research project has several practical implications.

First, we learned that CL is sensitive to various interacting factors. These include the agents, environments, hyperparameters, features, receptive fields, number of training steps per level, etc. This information is valuable to students and practitioners as it helps set realistic expectations for the difficulty of implementing CL for RL.

Second, our open-source Pac-Man environment may be useful to other students, instructors, and practitioners. We've designed this environment specifically to be used as a learning tool. As a result, we've implemented it as a discrete RL environment that is simple and easy to use. The inclusion of curriculum files can also benefit further CL research.

Finally, our work builds upon the existing body of research on RL and CL. It provides additional evidence as to the power of CL in specific contexts. In addition, it demonstrates cases where CL fails to outperform standard RL methods. Thus, our research provides an incremental advancement to the fields of RL and CL.

4.3 Future Research

First, we recommend repeating this experiment using adaptive curriculum learning. In our implementation, we fixed the number of steps per level to 200. However, it would be better to specify a level of proficiency (in terms of total reward) for each level. Then, once the agent reaches a passing score, it will automatically move on to the next level.

Next, we should repeat this experiment with 1 million training steps for each agent and hyper-parameter combination. This would allow us to get a more accurate assessment of the long-term trajectory of the agent's performance with respect to each hyper-parameter combination.

We also recommend repeating the experiment using a different set of curriculum levels. We found that some curriculum

levels were better at progressively training the agents than others. For example, we might consider a curriculum that always uses Level 1 of Pac-Man but selectively adds more complex features for each curriculum level.

Finally, we should try using different RL agents to see if they can improve upon the performance exhibited by the DQN. In particular, a hierarchical RL agent may be able to achieve higher performance by decomposing the Pac-Man game to smaller sub-tasks, and this approach may scale better with more advanced levels of Pac-Man as well.

5 Conclusion

In this research project, we created an open-source Pac-Man environment for RL. We trained four RL agents with and without CL to analyze their performance. Our results indicated that the performance of the SARSA and Q-Learning agents improved with CL. However, the performance of the AQL and DQN agents did not improve with CL. Finally, we learned that fine-tuning CL to improve RL agent performance is a complex and time-consuming process.

References

[1] B. N. Entertainment, "The official site for pac-man." [Online]. Available: https://www.pacman.com/en/history/

[2] J. Pittman, "The pac-man dossier," 8 2015. [Online]. Available: https://pacman.holenet.info/

[3] C. Birch, "Understanding pac-man ghost behavior," 12 2010. [Online]. Available: https://gameinternals.com/understanding-pac-man-ghost-behavior

[4] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," 3 2020.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv*, 12 2013.

[6] L. Bom, R. Henken, and M. Wiering, "Reinforcement learning to train ms.pac-man using higher order action-relative inputs." Institute of Electrical and Electronics Engineers, 2013. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6615002

[7] J. DeNero, D. Klein, and P. Abbeel, "The pac-man projects." [Online]. Available: https://inst.eecs.berkeley.edu/~cs188/sp21/projects/#credits

[8] J. L. Elman, "Learning and development in neural networks: the importance of starting small," *Cognition*, vol. 48, pp. 71–99, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0010027793900584

[9] D. L. Rohde and D. C. Plaut, "Language acquisition in the absence of explicit negative evidence: how important is starting small?" *Cognition*, vol. 72, pp. 67–109, 8 1999.

[10] T. D. Sanger, "Neural network learning control of robot manipulators using gradually increasing task difficulty," *IEEE Trans. Robotics Autom.*, vol. 10, pp. 323–333, 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:35187967

[11] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:873046

[12] A. Lazaric, M. Restelli, and A. Bonarini, "Transfer of samples in batch reinforcement learning." Association for Computing Machinery, 2008, pp. 544–551. [Online]. Available: https://doi.org/10.1145/1390156.1390225

[13] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[14] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep rl: A short survey," C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 4 2020, pp. 4819–4825, survey track. [Online]. Available: https://doi.org/10.24963/ijcai.2020/671

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction, 2nd ed.* The MIT Press, 2018.

A Technical Appendix

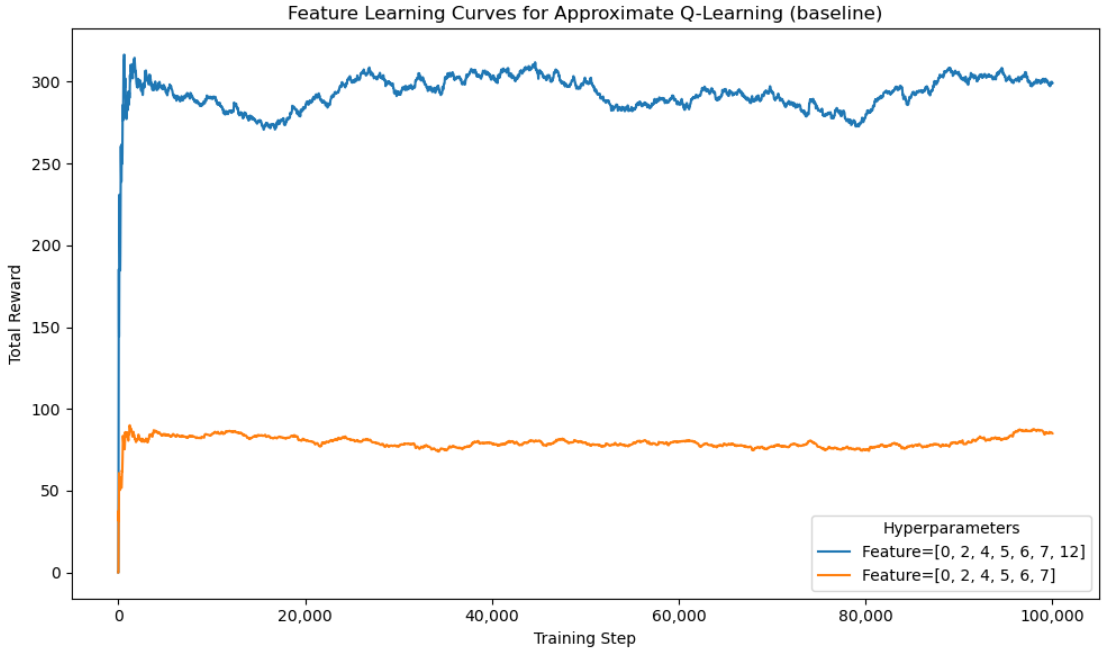

Figure 9: Comparison of learning curves for two feature subsets.

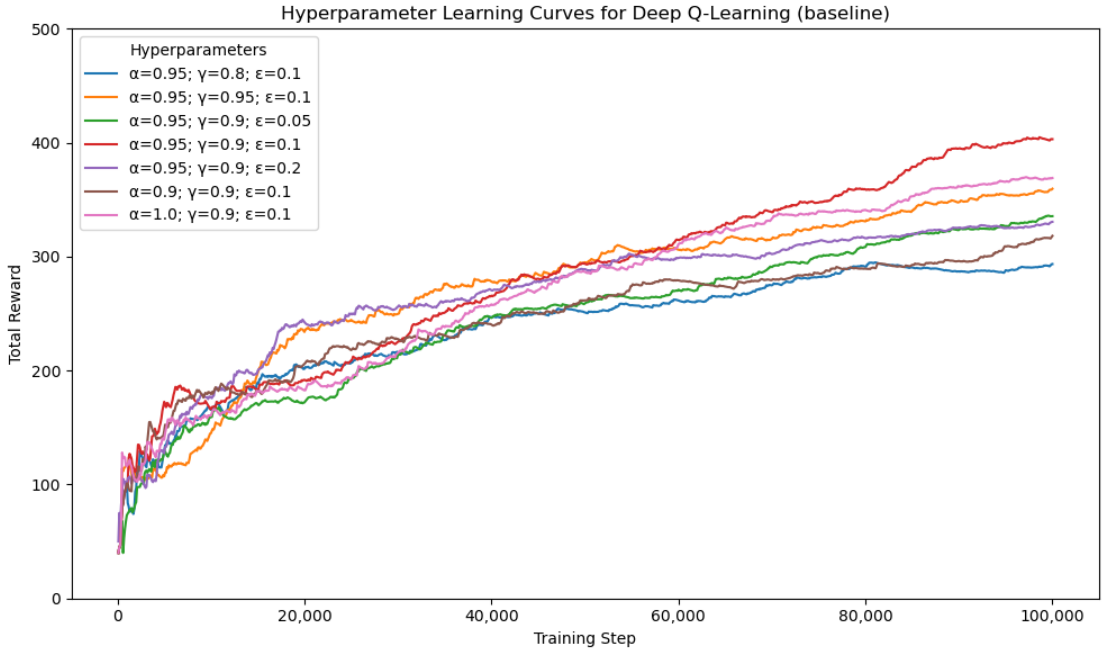

Figure 10: Learning curves from a hyperparameter sweep with the DQN agent[3].

[3]The DQN agent performed best with a relatively high learning rate ($\alpha = 0.95$). We believe this is because the agent's neural network uses a small backpropagation learning rate ($\alpha_2 = 0.001$).

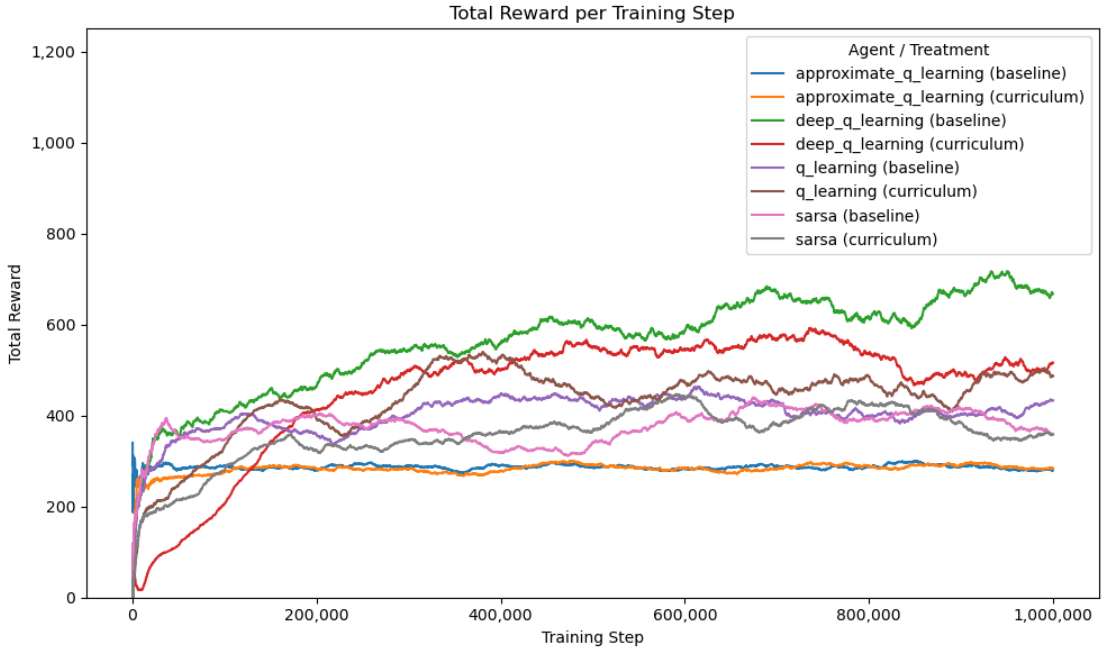

Figure 11: Learning curves for all agents to one million training steps.

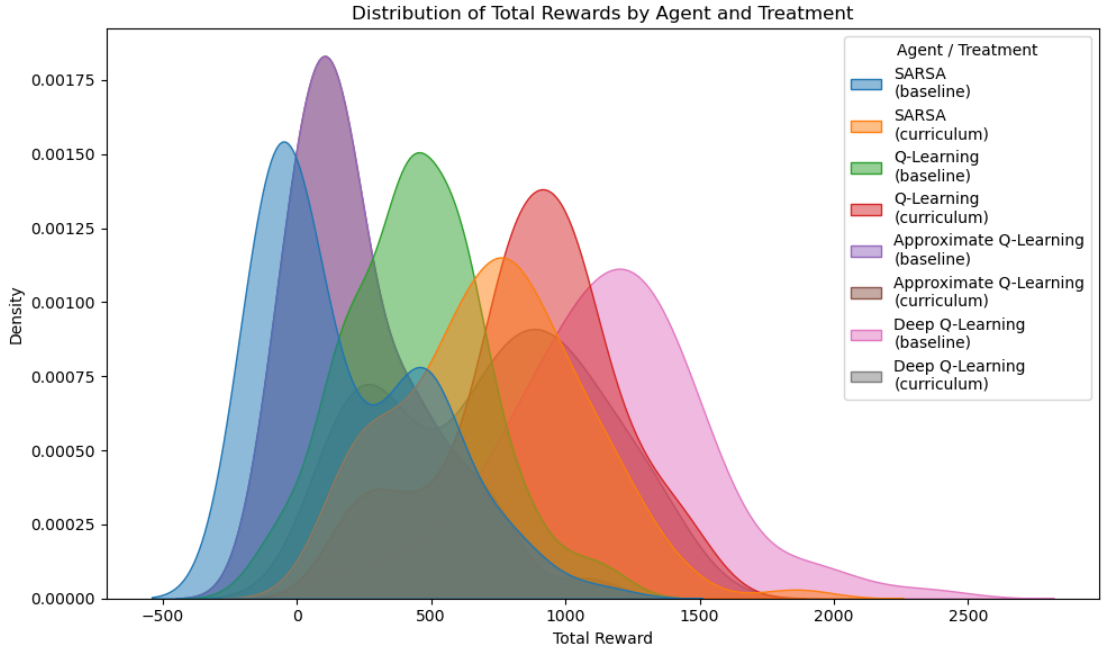

Figure 12: Distribution of total rewards by agent and treatment.