RL HW4

Thomas Klimek

November 2018

1 Part 1: Compare RL Algorithms

1.1 Problem Statement

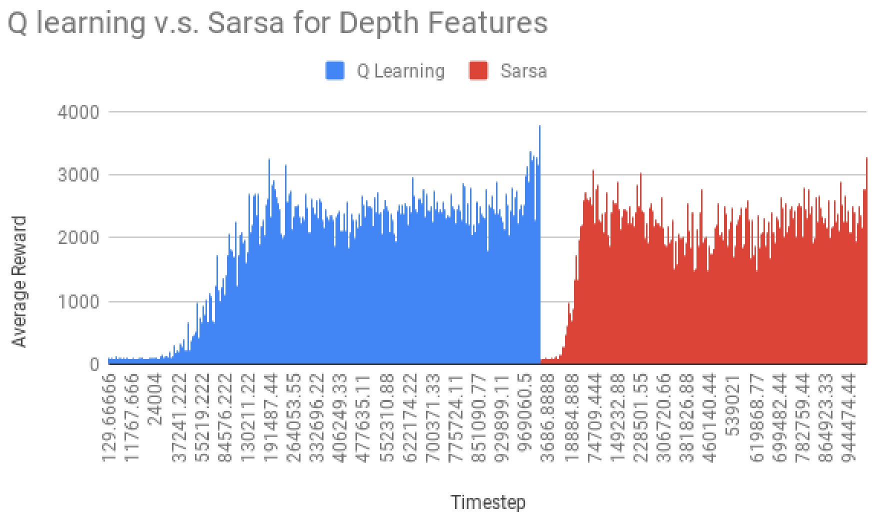
Our goal is to evaluate two different RL approaches for an identical configuration of the Ms. Pac-Man game. We will be comparing the performance of two different RL Learning algorithms, Q-Learning and Sarsa, and comparing the two given feature sets, depth and custom. To make our comparison we will generate plots illustrating the average reward per time-step of training in order to see which algorithm and feature configuration generates the highest rewards and which configuration can most quickly learn a policy to generate high rewards.

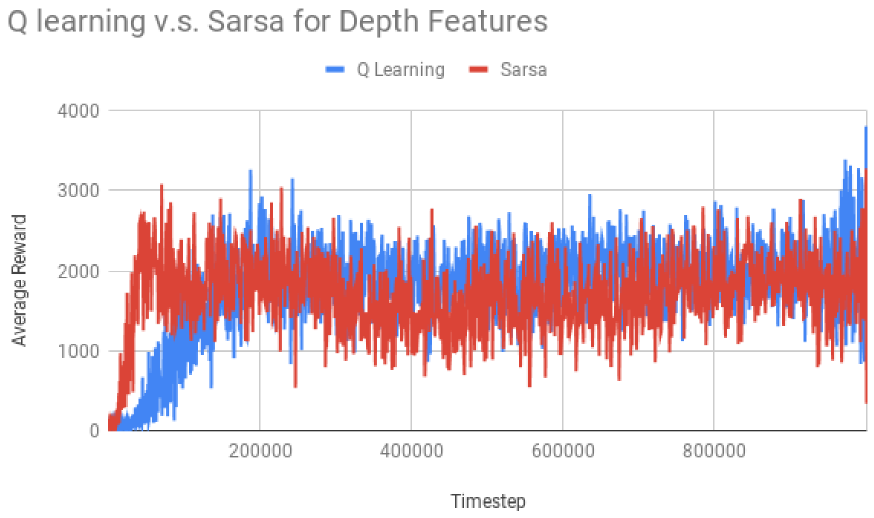
1.2 Implementation

To generate these graphs we will be utilizing a few of the features already implemented within the java RL Ms. Pacman package. We will be changing the values of *public static String STUDENT* to enable different algorithms and features. The other modification we will make to the code is to change the *episode()* function to return *game.getTotalTime()*. This will allow us to keep track of the timesteps taken for each episode. Then within our *train()* function, we will iterate training over a fixed amount of timesteps and accumulate the timesteps from each call to the episode function. For generating the graph I trained each algorithm over 1,000,000 timesteps. Then to account for the variance in performances, I repeated this process 10 times and averaged the results.

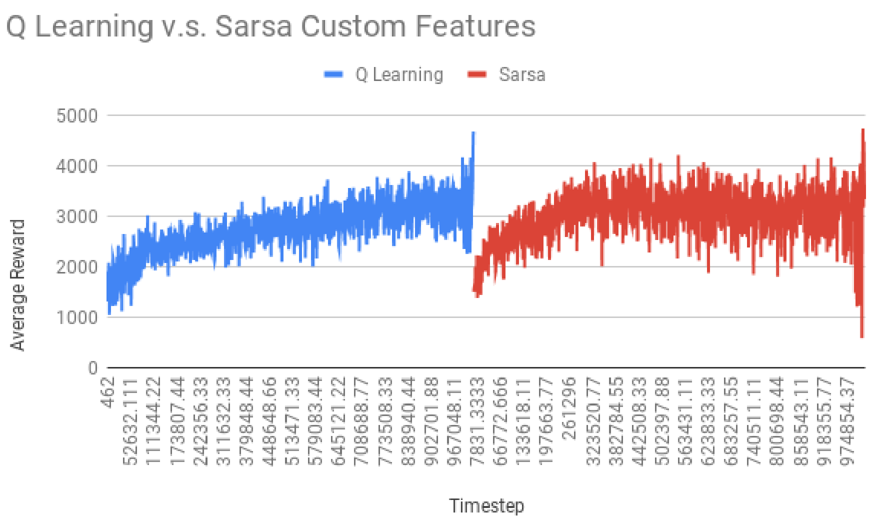
1.3 Plots Analysis

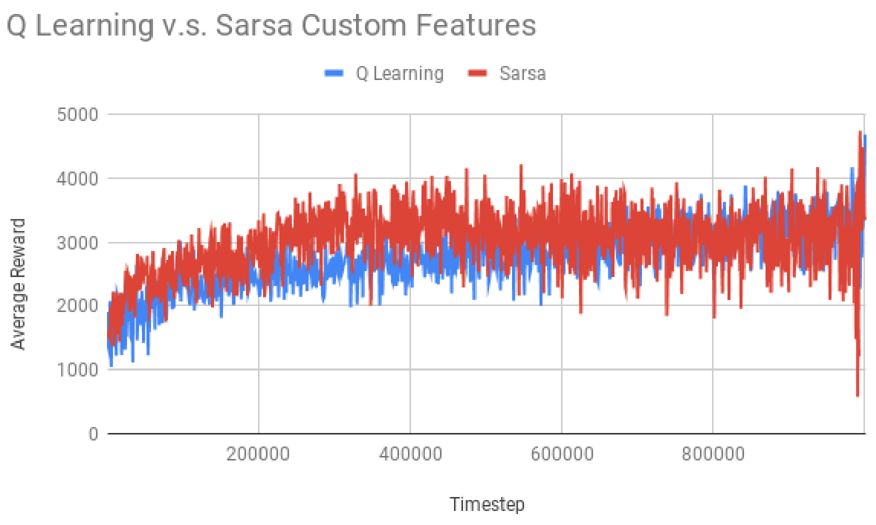
1.3.1 Q Learning v.s. Sarsa





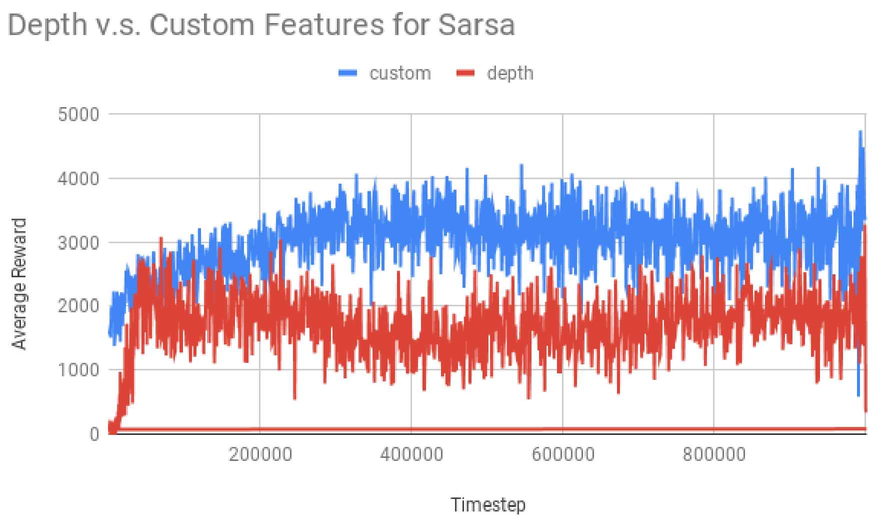
From the first graph we can compare the amount of relative amount of reward generated per episode and see that Q-Learning will on average produce higher rewards per episode as well as having a higher peak of reward produced. From the second chart we can compare the two algorithms at each timestep to see that Sarsa is actually able to initially learn faster than Q-learning and will generate higher rewards for the first 200,000 timesteps of training.



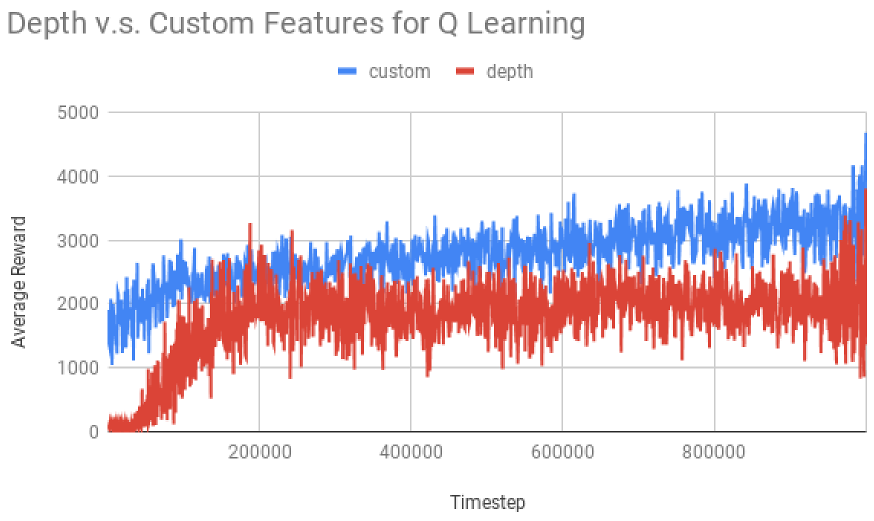


Using the custom feature set, we can see the algorithms are more evenly paced. Sarsa is still achieving higher reward in fewer timesteps however, and by 1,000,000 timesteps Sarsa is no longer performing worse than Q-Learning, and is even slightly doing better. This feature set enhances the learning with both algorithms and makes them more comparable in performance.

1.3.2 Depth v.s. Custom Features



From this graph we can see the difference in reward generated based on learning from the two different feature sets. We know The "custom" features are heavily engineered and generally perform better than the "depth" features. This plot illustrates the importance of feature selection for reinforcement learning, as the agent is able to consistently achieve much higher reward using the same learning algorithm with better engineered features. The same can be seen in the feature comparison for Q Learning.



2 Part 2: Freeform Experiment - Learning using Human Feedback

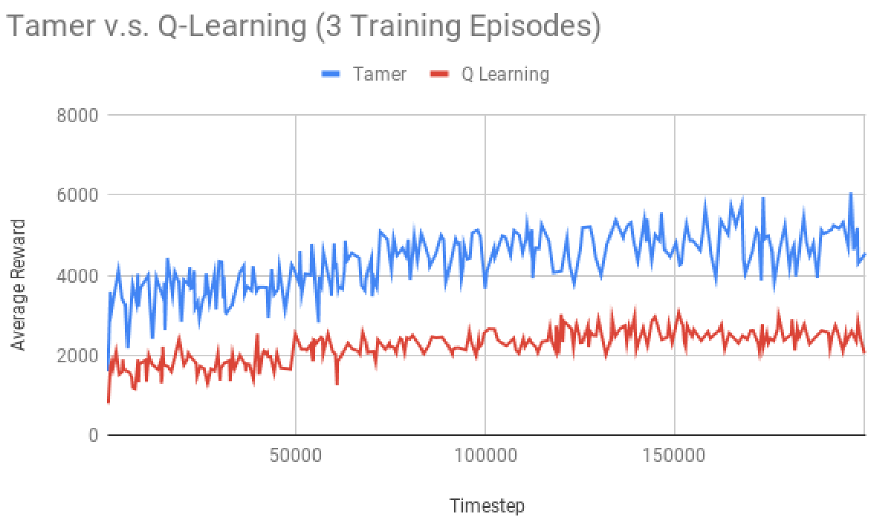
2.1 Problem Statement

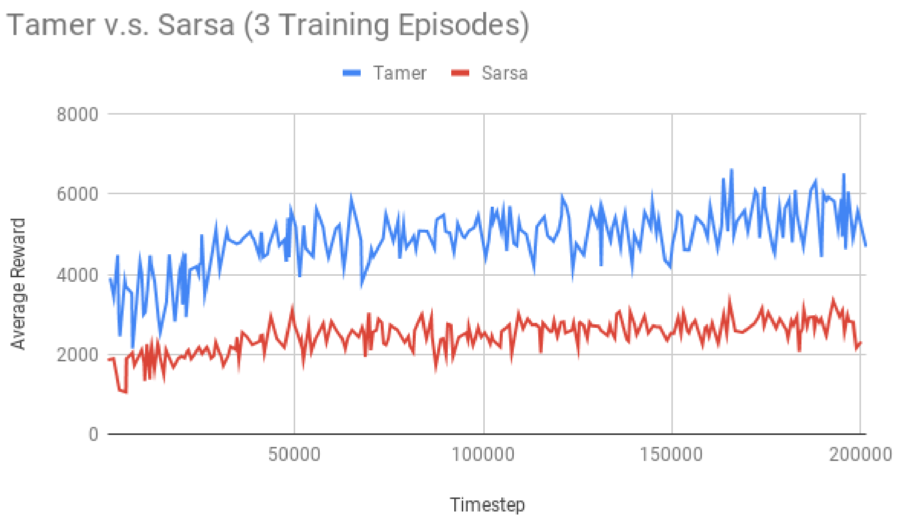
For the second part of this project I implemented a framework for learning based on human feedback based on W.B. Knox and P. Stone's paper *"Interactively shaping agents via human reinforcement: The TAMER framework."*. Using the Tamer Framework, our agent receives a reward signal based on human feedback and learns entirely based on this feedback. Applying the framework to Ms. Pac-man means that there is no reward generated from eating pills, power pills, ghosts, or any aspect of the game environment. Instead a user interacts with the agent pressing "1" to give it positive reward or "2" for negative reward. Since the agent seeks only to maximize human reinforcement, the optimal policy is defined solely by the trainer. We will compare the learning rate of an agent utilizing this framework with our two RL algorithms, Sarsa and Q Learning.

2.2 Implementation

The first step of the implementation involved removing all environmental rewards that will be replaced with human rewards. In the *game* class I introduce a new variable *private static boolean tamer* to indicate whether we are playing the game in "Tamer mode" using human feedback. Then using this variable, I conditionally removed all instances where score is added to the game if tamer is true. The second step of the implementation require creating the key listener to reward feedback realtime while the game was playing. I used the same *train()* function from the *RunTest* class, however I created a new function for simulating an episode of training. In this new function, *tamerEpisode()*, we bind a *KeyListener* object to each frame of the *GameView*. Then in our main loop for playing the game, we look for changes in keyboard input to our two feed back keys "1" and "2' and adjust the game score accordingly. This is done by creating a new member function of the *Game* class to set the score member. Then our agent is able to learn based on these scores using the *processStep(game)* function.

2.3 Plots and Analysis





These figures were generated by using Tamer training for the first 3 episodes, however after these episodes the RL algorithms were re-enabled to learn from natural game reward. This is an extension of the Tamer framework as the agent is not solely learning from human feedback, but it is also learning from environmental reward signal after the first few episodes. From the charts we can see this is an effective technique and generates much higher reward. Both the kickstarted Q-Learning and Sarsa Algorithms were able to receive double to triple the amount of reward than using these RL algorithms alone.