Reinforcement Learning: Pacman

Mateo Guaman Castro

November 30, 2018

Abstract

In this homework assignment, we focused on comparing the performance of an existing agent using different learning methods, and features for the function approximator. The agent used in this assignment learned to play the classic game of Ms. PacMan. In this environment, the task is for the agent to achieve the maximum score possible after a certain number of time steps given the dynamics of the game. The two learning methods used were Q-Learning and Sarsa. Finally, we varied the existing linear parameters to polynomial parameters.

1 Background

In reinforcement learning, two of the most used control methods for learning are Sarsa and Q-Learning [1]. These two methods are used to determine the Q values, also known as value functions of a state-action pair. The Q values of a state-action pair determine how good it is to take the action at the given state. If the policy of the agent is a greedy one, the agent would then choose the action that has the highest Q value. In case of epsilon-greedy policies, the agent would choose the action with the highest Q value with a probability of $1-\epsilon$ and it would select a random action otherwise. Both Q-Learning and Sarsa are TD (Temporal Difference) methods, meaning that the resulting Q value for a given state-action pair is determined by the resulting states in the future from selecting that action.

For single step Sarsa, the Q values can be found with the following equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Sarsa gets its name from its equation using: S_t, A_t, R_t, S_{t+1}, A_{t+1}. The equation for n-step Sarsa is:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

Finally, it should be noted that Sarsa is an on-policy method, meaning that the optimum policy found includes any biases introduced by exploration.

For Q-Learning, the Q values can be found with the following equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q_{S_t, A_t}]$$

Q-Learning is an off-policy method, meaning that the policy found using this method will not be biased by the exploration of the agent.

The task in this assignment was for an agent to play a game of Ms. PacMan and achieve the highest score possible as determined by the dynamics of the game. Ms. PacMan gets points when it eats pills, power pills, and when it eats the ghosts after eating the power pills. The game is over if Ms. Pacman is eaten by one of the four ghosts, who chase Ms. PacMan around. An image of the game can be seen in Figure 1. Since the state space of the game is too large in this case, and the state-action combinations are even larger, function approximation is used to parameterize the state at a given time step. The reason why parameterizing helps is because it reduces the state space to a set of parameters of a line. The initial features used were OBJECTS*DEPTH, where OBJECTS consists of four elements: regular pills, power pills, regular ghost, and edible ghost; and DEPTH was a number from 1 to 4 that determined the depth at which each object was found.

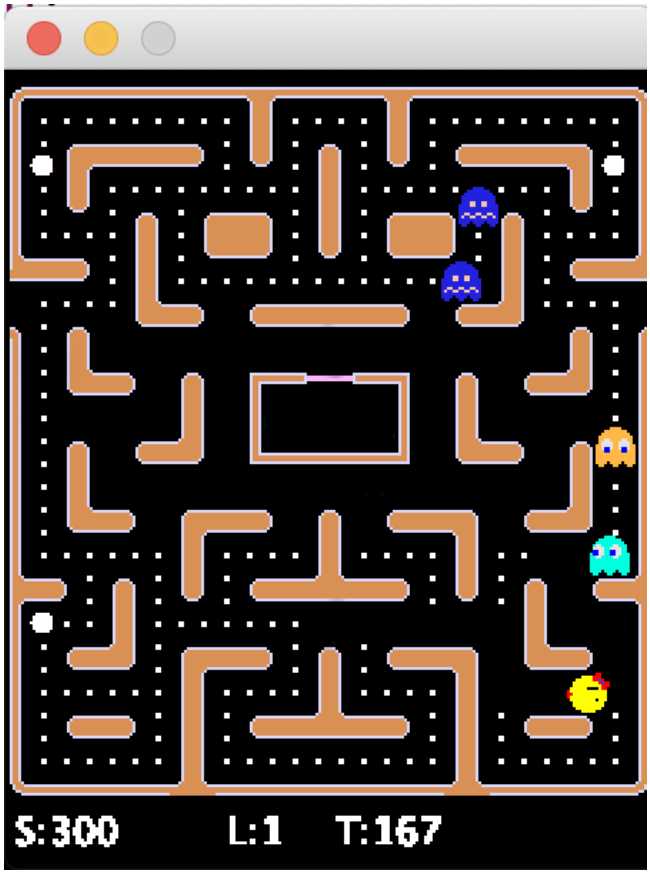


Figure 1: Ms. PacMan

2 Motivation

Having agents learn how to play games has been a popular way to benchmark their performance. This has been done with board games like Backgammon [4]

and Go [2]. More popular has been the use of classic Atari games, like PacMan. In this homework, we play the role of benchmarkers while using an already existing agent. We adjust the learning methods in an effort to determine the best learning method for the given task.

3 Experiment

3.1 Methodology

There are two parts to this experiment. The first one consisted of comparing two different learning methods: Sarsa or Q-learning. Two different feature sets were available in the existing environment, the OBJECT*DEPTH (called depth features) one described above, and a custom, more engineered, set of features. In this case, the depth features were used, since they had the potential to show more of a difference between the learning algorithms. The way their performance was measured was to compare the average score that each algorithm got at the end of each episode for a certain amount of episodes.

The second part of this experiment dealt with creating a new set of nonlinear features of the state space. The goal of this part was to determine whether or not a polynomial feature space affected the performance of the learning agent. Similarly as above, the evaluation metric used was the score obtained per episode. For the purpose of this experiment, we used Sarsa with a modified version of the depth feature set, with added polynomial features.

3.2 Implementation

Since the environment and learning methods for the agent were already implemented, this experiment was mainly concerned with evaluating the performance of the learning methods themselves. In this experiment, we ran ten different runs of 1000 episodes each. We saved the results into CSV files and then we plotted the resulting median and standard deviation.

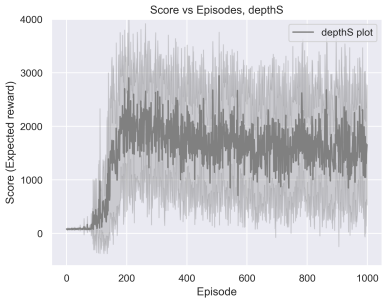
For the second part of the experiment, we created a new set of features based on the features that were already used for the agent. The original features consist of 16 elements, of the 4 different objects looking 1-4 steps ahead. In this assignment, the feature set was expanded with products of the different existing features. So we came up with a quadratic dataset, made up of the original sixteen features and the unique combinations of two features at a time. These are also represented as the elements of the upper triangular matrix where the columns and the rows represent the original feature vectors, and each location in the matrix represents the product of these features. For this experiment, we also ran the game 10 times with episodes of 1000 time steps each. Both Sarsa and Q-Learning were used with their original feature set and with the expanded, polynomial feature set.

4 Results and interpretation

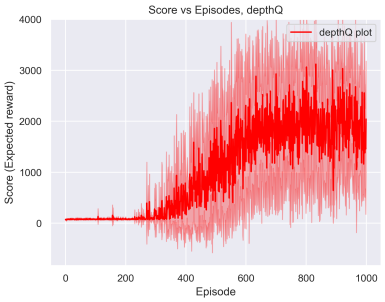
4.1 Comparison of Sarsa and Q-learning using Depth feature set

The results of the comparison between Sarsa and Q-learning using the same feature set can be seen in Figure 2. It can be clearly seen that, while both learning methods achieve a similar performance after 1000 episodes, Sarsa achieves its maximum performance after just around 200 episodes, while Q-Learning takes longer to learn. Q-Learning only starts learning at around the 400th episode.

There are a few reasons that could explain the difference in convergence speed of the two methods. There are two different factors that determine how the agent evaluates how good taking an action was, or more concisely, the Q values of a given state-action pair: the reward received from taking an action, and some measure of how good was the state at which the agent ended up from taking that action. The reward obtained from taking an action is the same in both Sarsa and Q-learning. The second measure is what changes. The way that Sarsa determines how good the next state is, is by looking at the Q value of the action that the agent would take at state S_{t+1} given its current policy. While this should mean that the agent will use the Q-value of the best possible action at state S_{t+1} (acting greedily), we have to remember that the agent's policy also dictates it to explore from time to time. Therefore, it is possible that its policy determines that at S_{t+1}, the agent should take a non-optimal random action, so the Q value of this non-optimal action will not be the highest Q value in the action space at that state. So coming back to S_t, it is possible that the agent uses a non-optimal Q-value as the measure of how good the next state is when taking an action A_t, which would convey wrong information about how good the state S_{t+1} is. On the other hand, the Q-learning method looks at the highest possible Q value at the next state, S_{t+1}, as a measure of how good the next state is when taking an action A_t (to evaluate $Q(S_t, A_t)$. This removes the bias that exploring adds to the Q values in Sarsa. However, Q-learning is still not free of problems. At the beginning of the episode, the agent simply does not know what the best Q-value of a state is, so it uses a random value (or whatever the non-accurate, highest **initial** Q-value is) as what it believes to be the best Q-value in S_{t+1}. Since these values are not accurate at all at the beginning of the episode, the agent will make decisions based on inaccurate information until it discovers the true Q-values at any given state. Once it learns more accurate Q-values at all or most states, then it will always act greedily, and therefore better than if using Sarsa. Therefore, the reason why Sarsa performs better than Q-learning in this task may be because either a)the initial information that the agent has is too incorrect to be useful towards the learning of the agent (Q-learning slows down learning too much by using incorrect information), or b) using the Q values determined by the actual policy (Sarsa), and not hypothesized best Q values (Q-learning), at S_{t+1} might be more effective at transmitting the agent correct information that can then be used to make more informed decisions of which actions to purse next.

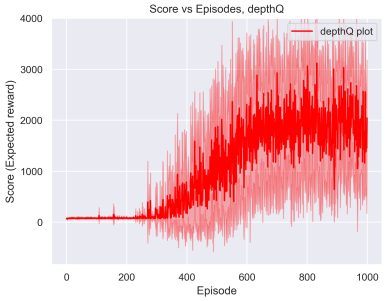


(a) Sarsa

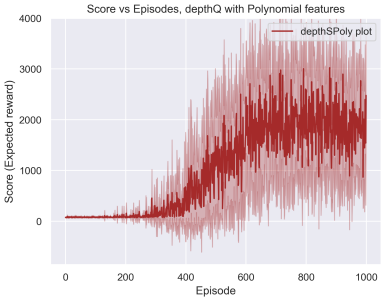


(b) Q-Learning

Figure 2: Learning methods comparison using depth features



(a) Q-learning, linear depth features



(b) Q-learning, polynomial depth features

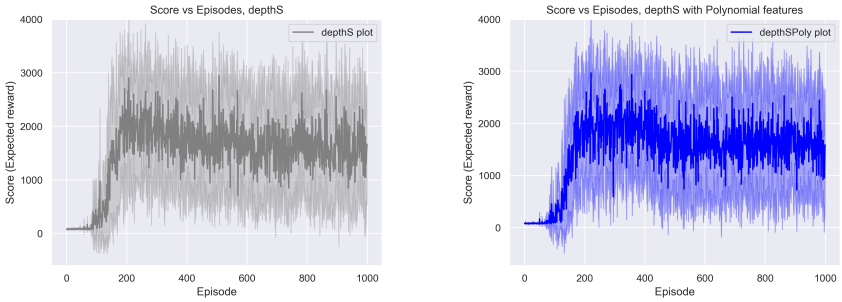
Figure 3: Q-learning comparison between linear and polynomial features

4.2 Comparison of between linear and quadratic features

The results of comparisons between using the linear and quadratic feature sets for both Q-learning and Sarsa can be found in Figure 3 and 4 respectively. The main result from these figures is that adding quadratic features does not help the agent perform better in any significant way, compared to the linear features. One explanation to this could be that there is not enough correlation between features so that the agent could exploit these additional quadratic features to achieve better performance. That is similar to saying that the linear features and weights of these features can provide all the information that additional quadratic features would provide.

5 Conclusion

In this homework assignment, the performance of two learning algorithms, with two different feature sets was analyzed in the task of learning to play the classic game of Ms. PacMan. Between the two learning methods used, Sarsa and Q-Learning, Sarsa achieved a faster speed of convergence to the optimal policy, and both algorithms reached the same asymptotic performance. It was also shown that for this particular task, and given the structure and dynamics of the environment, adding quadratic features to the feature set did not improve



(a) Sarsa, linear depth features

(b) Sarsa, polynomial depth features

Figure 4: Sarsa comparison between linear and polynomial features

the performance of the learning agent compared to the performance achieved using linear features. There are three main take-aways from this homework: Off-policy learning (Q-Learning) is not always better than On-policy learning (Sarsa), adding more complex features does not always mean that using these new features will increase the performance of the agent (important when considering neural networks), and that putting ourselves in the role of benchmarkers can provide great insights into the workings of different learning methods.

References

[1] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: an Introduction.* 2nd ed., The MIT Press, 2018.

[2] Silver, David, et al. "Mastering the Game of Go without Human Knowledge." Nature, vol. 550, no. 7676, 2017, pp. 354–359., doi:10.1038/nature24270.

[3] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[4] Tesauro, Gerald. "Temporal difference learning and TD-Gammon." Communications of the ACM 38.3 (1995): 58-68.