Exploring The Performance Impact Of Polynomial Feature Construction For Linear Function Approximation

Holt Spalding

, November 29, 2018

---

1. Background

1.1. Linear Methods In Reinforcement Learning

The goal of any reinforcement learning task is to determine the actions an *agent* ought to take within its *environment* at any given time in order to maximize some cumulative reward. For the purposes of this paper, I will constrain this very broad definition of reinforcement learning to simply mean the process by which one solves a *Markov Decision Process*. A Markov Decision Process (MDP) is just one of many models of how an agent interacts with its environment over discrete time steps, and it is defined in terms of three basic elements: a state space, describing the set of all potential states of the environment, an action space, describing the set of all potential actions the agent can take within that environment, and a reward space, describing the set of all possible rewards that can be achieved as the result of taking some action. In this context, it can said that the primary goal of any RL algorithm is to approximate what is known as a Q function. A Q function tells an agent the expected cumulative rewards that could be attained for performing any given an action a given the state of the environment s. It is on the basis of this Q function that an agent will make decisions in order to maximize rewards.

An RL algorithm which employs so-called *linear function approximation*, is one which approximates the "Q-value"/expected cumulative reward of any given state-action pair as follows:

$$Q(s,a) = \sum_{i=1}^{d} \theta_i \phi_i(s,a)$$

A linear function approximator starts with a mapping ϕ which assigns a finite-dimensional vector to each state-action pair, and then calculates the Q-value of each state-action pair by linearly combining $\phi(s,a)$ with an arbitrary weight vector θ. It is then the job of an RL algorithm to adjust the weights in θ such that Q optimally approximates the so-called "true Q-function", often denoted Q*. An agent which has faithfully approximated Q* should know the optimal action to take given any state of the environment. An agent's capacity to optimize for θ is not the sole determiner in its success in approximating Q*, the choice of ϕ also has an effect on success of an agent. This is the domain of feature engineering, and in some ways it is more art then science. The optimal set of features can't always be learned, often times a human with domain-specific knowledge must use their intuition in order to choose features.

1.2. Feature Engineering

Feature engineering is the process by which domain knowledge is used in order to develop features that allow a machine learning algorithm to work. In the context of simple linear regression for example, an engineer trying to predict housing prices may choose the size of a house and location as their set of features. These would probably be considered a good set of features since they are both relatively strong predictors of housing price. A bad feature might be the age of the previous owners, since this generally isn't a huge factor in price of a house. Feature engineering is not solely concerned with picking and choosing independent features which may or may not have strong predictive power, it is also about combining and affecting features in different ways to see if they may have an interactive effect on one another. In the context of the regression problem, if size, location and housing price have a quadratic relationship, then using a simple line of best fit to predict housing price wont necessarily cut it. The introduction of quadratic features becomes necessary should we want to approximate a quadratic function. Since the true relationship between size, location, and price can never be known for certain, a certain amount of trial and error becomes necessary during the process of feature construction.

1.3. Polynomial Feature Set

One of the simplest methods by which a feature set can be extended in order to capture more complex relationship between features is by creating polynomial features built from the original feature set. For example a two-dimensional feature

set represented by vector (s_1, s_2) could be extended in dimension so as to include $s_1 s_2$, s_1^2, s_2^2. Depending on the context of the problem and the complexity of the function to be approximated, this simple trick can in fact result in vast improvements in the accuracy of approximation.

2. The Experiment

For this experiment I have attempted to evaluate the effect of simple polynomial feature engineering on the performance of two RL agents learning to play Pacman. The Pacman environment I used was provided by Philipp Rohlfshagen, David Robles and Simon Lucas of the University of Essex (no reference). The two agents I compared employed two different RL algorithms, Q-learning and Sarsa (the specifics of those algorithms fall outside the domain of this paper). Both agents utilized the same set of features known as the "Depth Feature Set," which was already built-in to the learning environment I used to run the experiment. The depth feature set assigns a 16-dimensional vector to each state-action pair, representing the proximity of Pacman to surrounding power pills, regular pills, regular ghosts, and edible ghosts. I first assessed the performance of the Sarsa and Q-learning agents on a standard game of Pacman (four ghosts, four powerpills, etc) in terms of in-game score after 1000 training cycles. I then repeated the experiment but with both agents utilizing a modified form of the depth feature set which included the original set of 16 features along with the original 16 features but with each feature squared, and then an additional 120 features representing the product of the set of all possible combinations of the original 16 features.

3. Results

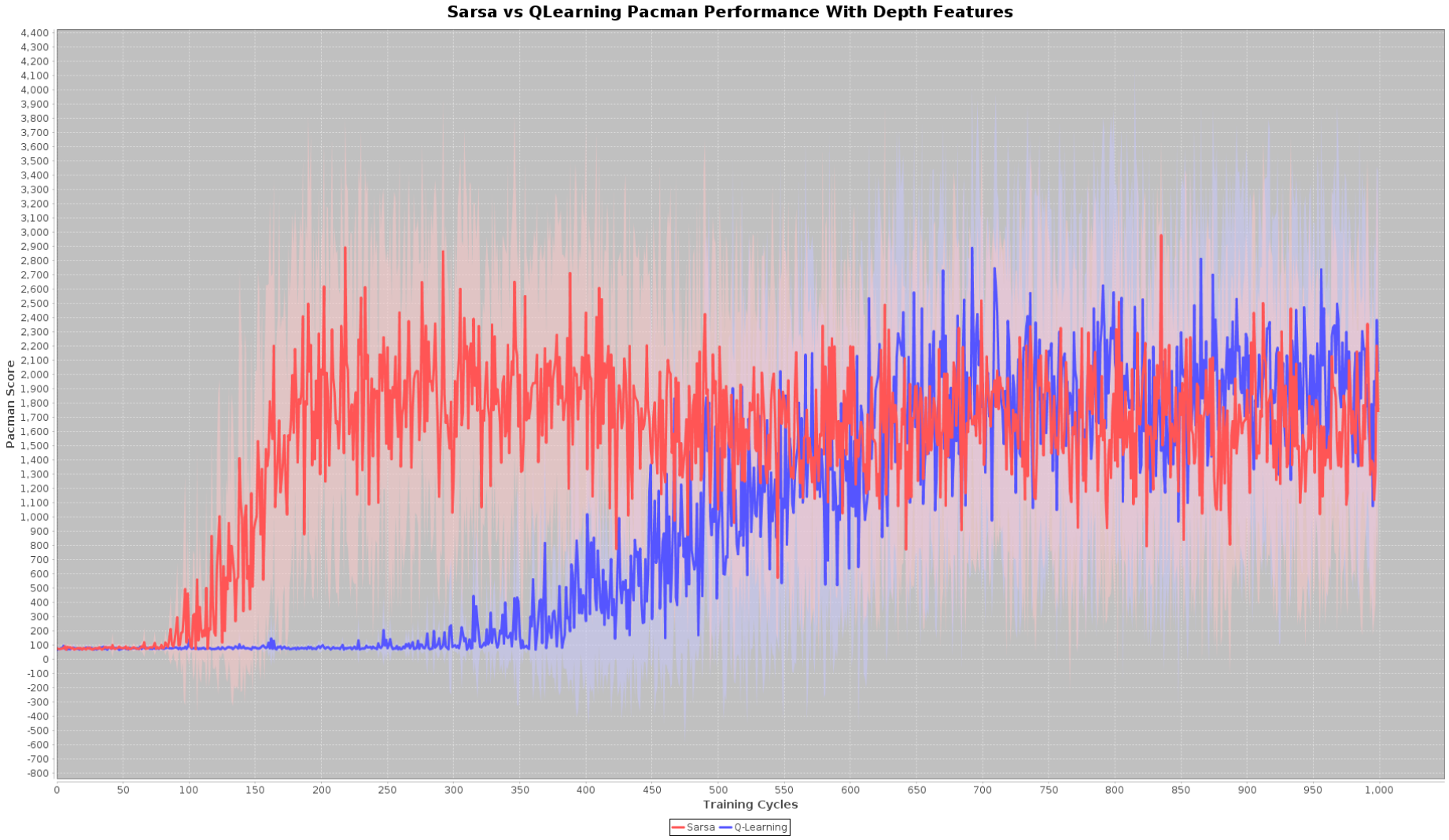

Figure 1: Sarsa Vs. Q-Learning With Depth Features

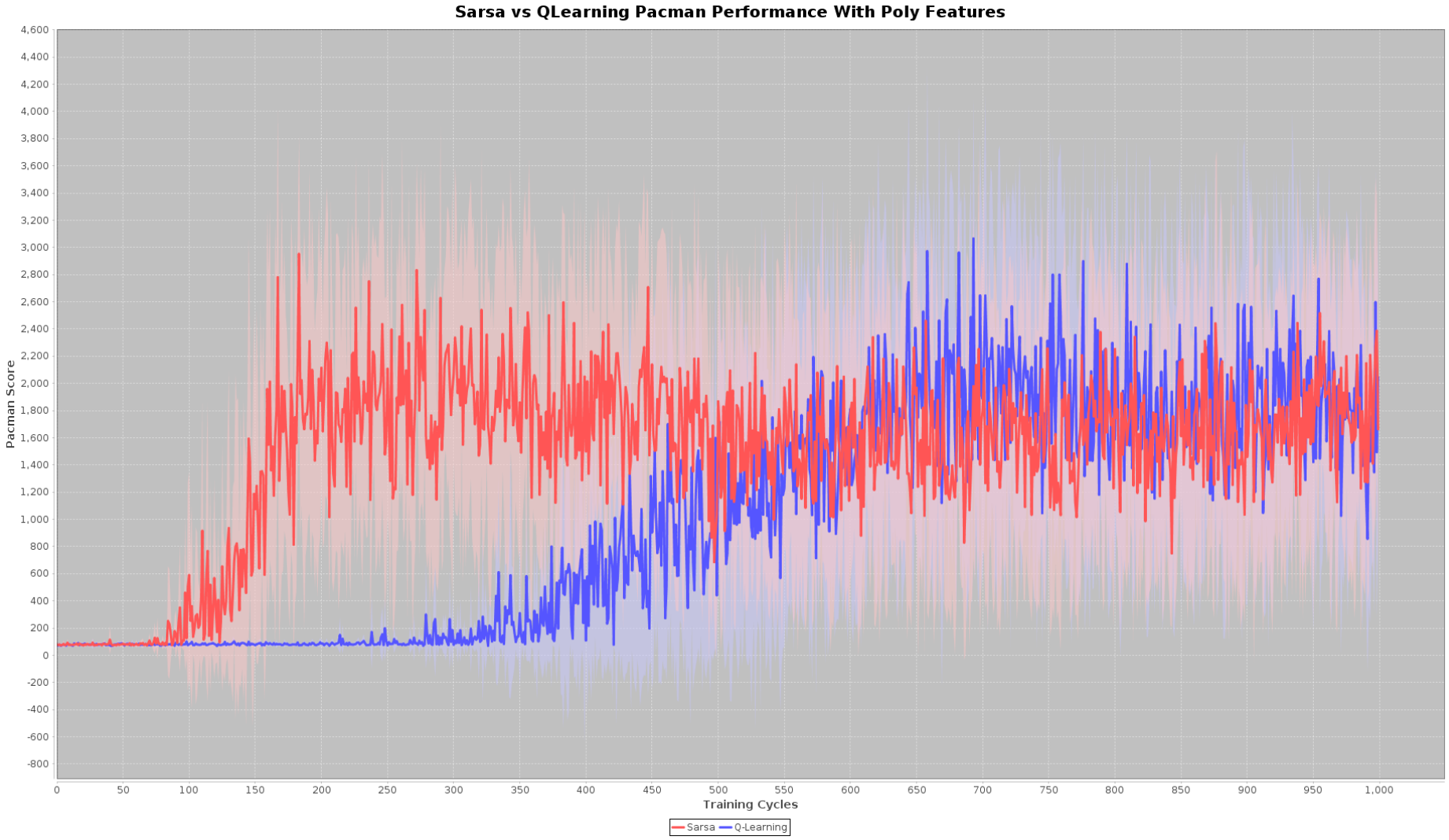

Figure 2: Sarsa Vs. Q-Learning With Polynomial Depth Features

The above graphs display the mean and standard deviation of the in-game scores achieved over ten trials after each training cycle. As the results would suggest, Sarsa (in red), generally converged to a suboptimal solution quicker than Q-learning, achieving an average score of around 2200 after about 200 training cycles. Continuing training for 100,000 cycles did not seem to significantly impact performance at all, suggesting that the depth feature set lacks any strong predictive power. The polynomial feature set had negligible impact on the performance of the two agents, likely due to the lack of power in the original depth feature set.

4. Brief Discussion

While polynomial feature construction did not appear to have a strong impact on the performance of this particular set of agents, I think that this failure demonstrates just how finicky feature construction can be. Assuming both the Q-learning and Sarsa agents optimized for θ as much as they possibly could (as suggested by the plateau in performance), there are only two possible scenarios under which these results could have been observed. The first possibility is that there is a linear relationship between the set of depth features and expected reward. The far more likely scenario is that the depth features we're poorly constructed and therefore did not provide enough relevant information to our Q function approximators. This seems obvious since overall Pacman performance improved when I replaced the depth features with another set of built-in "custom" features. I would like to run the experiment again on the custom feature set to assess if polynomial feature construction would have an impact. There are also much stronger forms of feature construction which could have been employed in this experiment to maximize performance. In completing this experiment, I became very familiar and fascinated with the concept of automatic feature construction. Szita (2012) provides a method by which features can be automatically generated for any game-playing scenario. I would like to learn a lot more about the topic of automatic feature construction in the future because I feel as though these techniques could vastly improve the capabilities of artificially intelligent agents if they are ever perfected.

5. References

1. Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. MIT press.

2.Szita, I. (2012). Reinforcement learning in games. In Reinforcement Learning (pp. 539-577). Springer, Berlin, Heidelberg.