Practical 4: Reinforcement Learning

Abdul Saleh, Dean Hathout, Brendan O'Leary
{abdelrhman_saleh, dhathout}@college.harvard.edu, boleary@g.harvard.edu
abdulsaleh, dhathout, boleary134

May 7, 2019

1 Technical Approach

The game *Swingy Monkey* provides a fertile landscape for the application of reinforcement learning. An agent can learn to play the game on its own by interacting with the game's environment, adjusting its optimal policy based on feedback from the agent's action. It is also inexpensive to get data samples to learn this optimal policy by simply playing more games.

In this practical, we experiment with 3 reinforcement learning approaches for playing Swingy Monkey:

1. **Traditional Q-Learning**: Q values for each state-action pair are recorded in a table.
2. **Linear Q-Approximation** The Q-function is approximated by a linear model.
3. **Deep Q-Approximation**: The Q-function is approximated by a 6 layer network with 256 neurons each and ReLU activation.

A classic approach in reinforcement learning is Temporal-Difference Learning, where we learn the Q function, $Q(s, a)$, by bootstrapping from the current estimate of the value function and updating the Q function to minimize the error between a predicted and target Q value. In our **traditional Q-Learning** approach we model the Q function as a table of state-action pairs such that the optimal policy $\pi(s) = \arg\max_a Q(s, a)$. The Q values are updated via stochastic gradient descent with learning rate α.

$$Q(s,a) \leftarrow Q(s,a) - \alpha[Q(s,a) - (r(s,a) + \gamma \max_{a \in A} Q(s', a'))] \tag{1}$$

We arrive at this update by defining a loss function to minimize similar to that of mean squared error (note we parameterize the table of estimated Q values as $Q(s, a); \mathbf{w})$ with parameters $\mathbf{w}$.

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\mathbb{E}_{s,a}\left(Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q(s',a';\mathbf{w})]\right)^2 \tag{2}$$

#	Learner	Max Score	Mean Score
1	Traditional, $\alpha = 0.3, \epsilon_{max} = 0.5, \epsilon_{min} = 0$	**69**	**4.54**
2	Linear Q-Approx, $\alpha = 1e^{-12}, \epsilon = 0$	18	0.04
3	Deep Q-Approx, $\alpha = 1e^{-6}, \epsilon = 0$, batch size = 8	2	0.03
4	Deep Q-Approx, $\alpha = 5e^{-7}, \epsilon = 0$, batch size = 32	24	0.51
5	Deep Q-Approx, $\alpha = 1e^{-7}, \epsilon = 0$, batch size = 32	2	0.01
6	Deep Q-Approx, $\alpha = 5e^{-7}, \epsilon = 0$, batch size = 8	36	0.98
7	Deep Q-Approx, $\alpha = 5e^{-7}, \epsilon = 0$, batch size = 16	1	0.02

Table 1: Maximum and average scores for *Swingy Monkey* using various Q-learners trained with **standard features**. Best result in bold. $\gamma = 1$ for all.

#	Learner	Max Score	Mean Score
1	Traditional, $\gamma = 1, \alpha = 0.3, \epsilon_{max} = 0.5, \epsilon_{min} = 0$	293	17.79
2	Traditional, $\gamma = 0.7, \alpha = 0.3, \epsilon = 0$	46	2.91
3	Traditional, $\gamma = 1, \alpha = 0.1, \epsilon = 0$	931	51.35
4	Traditional, $\gamma = 1, \alpha = 0.3, \epsilon = 0$	**1389**	**103.24**
5	Traditional, $\gamma = 1, \alpha = 0.4, \epsilon = 0$	296	9.18

Table 2: Maximum and average scores for *Swingy Monkey* using various Q-learners trained with **engineered features**. Best result in bold.

Minimizing this loss with respect to the weights, we can approximate the gradient to be used in the learning update as:

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r + \gamma \max_{a' \in A} Q(s',a';\mathbf{w})] \tag{3}$$

In addition to recording the Q values in a table, we experiment with parametrizing Q in a functional form using both linear models and deep neural nets. We call these approaches **Linear Q-Approximation** and **Deep Q-Approximation**). This proves necessary in cases where the learning problem becomes large, particularly when the state space is continuous. We provide the functional form of the linear version for illustration:

$$\hat{Q}(s,a) = \mathbf{w}^\top \boldsymbol{\phi}(s,a) \tag{4}$$

where weights $\mathbf{w}$ are learned through gradient descent and $\boldsymbol{\phi}$ is an arbitrary basis function.

We run experiments using both the default **standard features** describing the state such as position of bottom of monkey, position of tree trunk etc. We also experiment with **engineered features** such as distance between top of monkey and tree trunk and distance between bottom of monkey and top of tree trunk. We additionally included a gravity feature, based on the falling velocity of the monkey, in both the standard and engineered feature sets. For the traditional Q-learning approach, the features were discretized into 12 bins to shrink the space size.

features

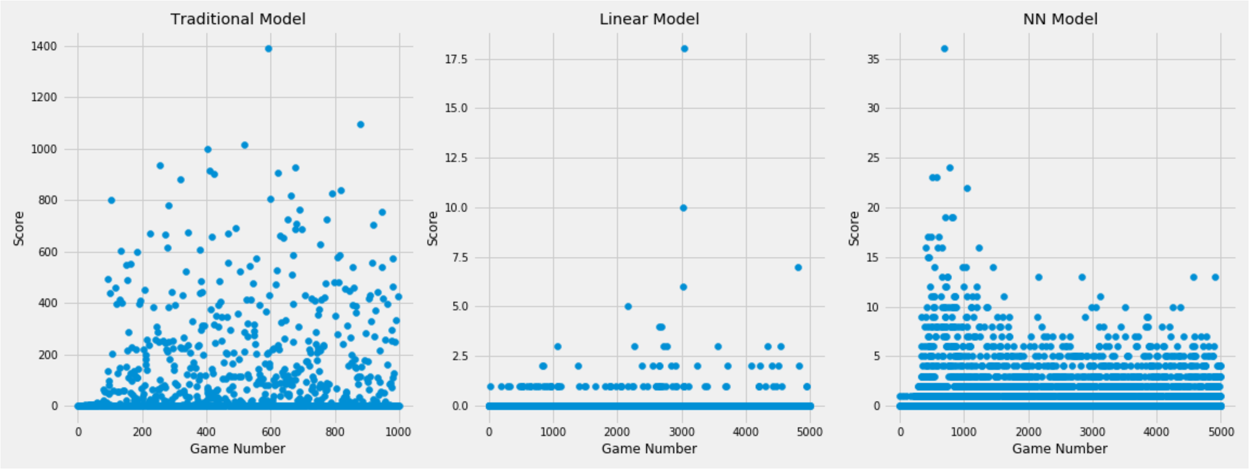

Figure 1: Results for the best Traditional Q-Learning, Linear Q-Approximation, and Deep-NN Q-Approximation (Learner 3, 8, and 12, respectively).

2 Results

Table 1 summarizes our results for the 3 approaches trained on the standard features. As the table shows, the traditional approach achieves the highest score so we decide to test it on the engineered features. Table 2 shows scores achieved by the traditional model using the engineered features for different hyperparameter settings.

We compare a decaying ϵ-greedy policy for exploration, starting at ϵ_{max} and falling to ϵ_{min} for exploration. However, we find that a completely greedy policy with $\epsilon = 0$ achieves better performance. We also find that a discount factor of $\gamma < 1$ hurts performance, possibly because it minimizes the value of longer term rewards which are important for this task. Our results also show that learning is highly sensitive to parameter initialization. So for example, performance noticable drops between $\alpha = 0.3$ and $\alpha = 0.4$. Figure 2 shows the effect of changing α on the scores achieved at each iteration.

The best performing was achieved through traditional Q-Learning with engineered features. It outperforms all other models as seen in Table 1 and Table 2, achieving a maximum score of 1389 and average score of 103.24.

3 Discussion

Comparing the results of these various models on standard features raises several interesting points, the most salient of which is *the clear instability of the functional approximations.* As can be seen in table 1, Linear Q-Approximation achieves a max score of 18 and an average score of 0.04, failing to pass even the first tree trunk the vast majority of the time. Additionally, we note that essentially all hyperparameter configurations attempted for this approach resulted in divergence, with weights $\mathbf{w}$ exploding in magnitude. We hypothesize that this results from the fact that the same model

divergence

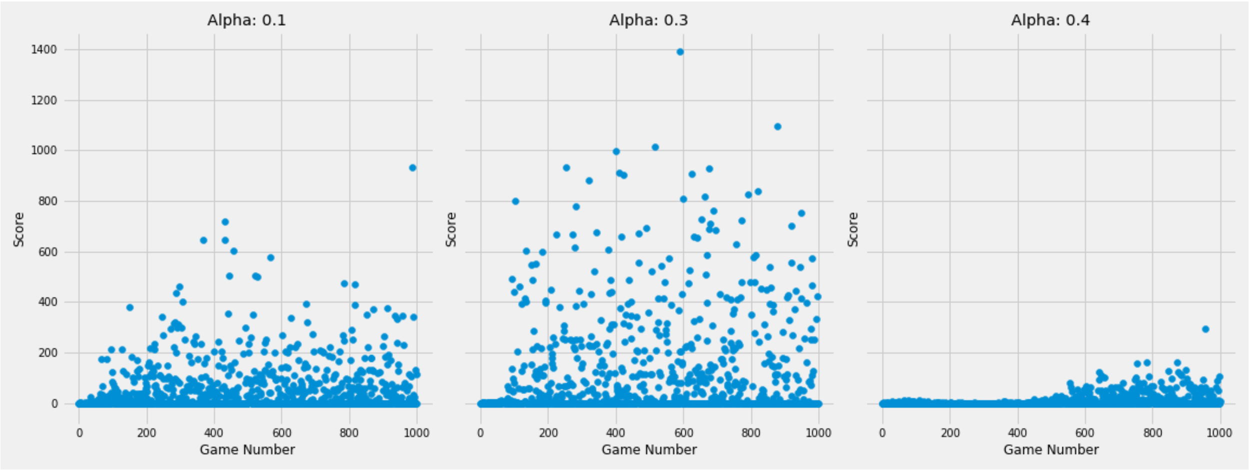

Figure 2: Discrete linear approximations: all models use $\gamma = 1$, $\epsilon = 1$, new features with varying $\alpha : (0.1, 0.3, 0.4)$.

is used to approximate both the predicted and target Q-values when calculating the loss. This simulates a cat chasing its own tail resulting in divergence.

Deep Q-Approximation, while slightly more stable than the linear model, also suffers from poor performance and frequent divergence. The best Deep Q model (with a learning rate of $5e-7$ and a batch size of 8) achieves a maximum score of 36 and a mean score of 0.98, only navigating through the first tree trunk on average. The performance of these functional approximations are visualized in Figure 1, which clearly shows an inability to learn for these models.

In contrast, Figure 1 shows that traditional Q-Learning is able to achieve satisfactory scores within approximately 100 games, with a generally increasing relationship between Score and Game Number. Table 1 shows much better maximum and average performance for this model. As such, we decided to continue experimentation with this approach only. Table 2 shows the performance of the traditional method with various hyperparameter configurations, on engineered features. This feature set consists only of distance to tree, monkey velocity, distance between tree top and monkey top, and distance between tree bottom and monkey bottom. Since the monkey's height and the length of the tree gap remain the same for all games, we are able to shrink the feature space down in this way without losing any information, resulting in a less complex Q-function to estimate. We hypothesize that this is the reason for improved performance of the traditional Q-learner on engineered features relative to standard features.

Thus we conclude that although the Q-function approximation can be more flexible and powerful than using a state-action pair table, this flexibility comes at the cost of instability and sensitivity to hyperparameter settings.

We also note that using a completely greedy strategy with $\epsilon = 0$ unexpectedly outperformed ϵ-greedy strategies which balance exploration and exploitation. We hypothesize this is because forcing the model to explore wastes valuable training time when the task is simple and the training iterations are limited. Instead, we exploit each step to maximize performance while implicitly exploring in the process since the Q-function estimates are noisy during learning.