Practical 4: Reinforcement Learning

Zachary Dietz, Jocelyn Tolpin, Theo Walker
zdietz@college, jocelyntolpin@college, twalker@college
github repo: https://github.com/jocelyntolpin/CS-181

May 8, 2019

1 Technical Approach

This practical tasked us with implementing a reinforcement learning strategy for the game *Swingy Monkey*. We decided that the most intuitive way to approach this problem would be to consider a state space with four dimensions: the monkey's horizontal distance from the tree, the monkey top's vertical displacement from the top of the tree, the monkey's instantaneous vertical velocity, and the gravity setting for the current game. Our objective was to learn the optimal binary action (jump or don't jump) corresponding to every state.

Incorporating all possibilities for the above quantities would result in an extremely large state space. Let s denote the vector of an arbitrary state, with entries corresponding to the values above (i.e., s_0 is horizontal distance from the tree). Empirically we found that these entries could take on values in the following intervals:

$$s_0 \in [-150, 500] \qquad s_1 \in [-200, 350] \qquad s_2 \in [-45, 45] \qquad s_4 \in \{1, 4\}$$

The first three dimensions have resolutions of 1, and the last is binary. Hence, letting S_i represent the ith dimensional space, the entire state space S would have

$$|S| = |S_0||S_1||S_2||S_4| = 650 \cdot 550 \cdot 90 \cdot 2 = 64{,}350{,}000$$

We assumed that working with a state space this large would be difficult since our RL algorithm would rarely encounter states it had seen before. Hence we decided to course grain our state space to S', with

$$|S'_0| = 12,\ |S'_1| = 8,\ |S'_2| = 5,\ |S'_3| = 2 \implies |S'| = 12 \cdot 8 \cdot 5 \cdot 2 = 960$$

This new space is more than 67,000 times smaller than the original and is therefore far easier to learn. If $S_i = [a_i, b_i]$, then we assign a vector $s \in S$ to $s' \in S'$ with

$$s'_i = \text{floor}\Big(\frac{|S'_i|(s_i - a_i)}{|S_i|}\Big)$$

We did this for all the states of the monkey in order to convert to our new state space.

1.1 Q-learning

To learn the above state space, we chose to use the technique of Q-learning. This strategy assigns a value $Q(s, a)$ to every state-action pair. The Bellman equations state that for an optimal value matrix $Q^*(s, a)$,

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A}[Q^*(s'|a')],\ \forall s, a$$

Once we have Q^*, we can determine the optimal policy π^* from $\pi^*(s) = \text{argmax}_a Q^*(s,a)$. In order to begin to calculate Q^*, we parametrize it with $\mathbf{w}$ such that $w_{s,a} = Q(s,a;\mathbf{w})$ is a table of estimated Q values. With this new parameterization, we can define a loss function for $\mathbf{w}$ that can be improved via stochastic gradient descent:

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

leading to the gradient descent update

$$w_{s,a} \leftarrow w_{s,a} - \eta(w_{s,a} - [r + \gamma \max_{a' \in A} Q(s',a';\mathbf{w})]) = (1-\eta)w_{s,a} + \eta[r + \gamma \max_{a' \in A} Q(s',a';\mathbf{w})]$$

Therefore, if η is small, the previous value of $w_{s,a}$ will be emphasized in the next value of $w_{s,a}$. Since we'll need to guess $\mathbf{w}$ to begin our agent, we should use a large η value since our initial guess will likely be very poor.

As explained in section, the above update is an off-policy, Q-learning update since our gradient update uses the maximum over a' rather than following what our policy dictated. As we'll show in **Results**, we also considered an on-policy, SARSA approach, but this yielded worse results. Importantly, we assigned death states a Q value of 0, so that their preceding states are only updated based on the reward for death.

1.2 ε-greedy

The ε-greedy approach is a good strategy to balance the exploration/exploitation tradeoff since for any given state, it chooses between taking the action that maximizes our expected reward with probability and randomly choosing an action. We chose the initialization $Q(s,a) = 0$ for all s, a, so some exploration is important for efficient convergence to Q^* this initialization is optimal! So, we take the maximizing action $a_{\max} = \text{argmax}_{a \in A} Q^*(s,a;\mathbf{w})$ with probability $1-\varepsilon$, and with probability ε, we pick an action uniformly at random. We also initialized a matrix $N(s,a) = 0$ for all s, a in order to keep track of the number of times we have visited a state-action pair. This allows for ε to be reduced as a function of the number of times that a state-action pair has been visited; if a state has been reached many times with a corresponding action, it's more likely that it is optimal and thus exploration should be reduced. Specifically, whenever the state-action pair (s,a) was reached, ε for this state action pair became $\frac{e}{k}$ where e is the original ε and k is the number of times we have reached (s,a). We also ultimately decided to decrease η in the same way, since it makes intuitive sense that the learning rate for a state should decrease with the number of times that state has been updated because each consecutive time we visit it we gain less information.

2 Results

Our primary metric for the performance of our Q-learning model was the average score it achieved over 300 epochs. The maximum score is also informative but less so because we found that it was more prone to randomness than the average score. We also viewed a plot of all the scores from each epoch for every model we trained, so we considered other characteristics of the model like how fast it learned and how consistent it was (we did not measure these values quantitatively). To find the optimal values of η, γ, ε for our agent, we performed a grid search in which we simultaneously varied these parameters in the ranges $[0.1, 1]$, $[0.6, 1]$, and $[0.0001, 0.1]$, respectively. For instance, for $\varepsilon = 0.001$ and a subset

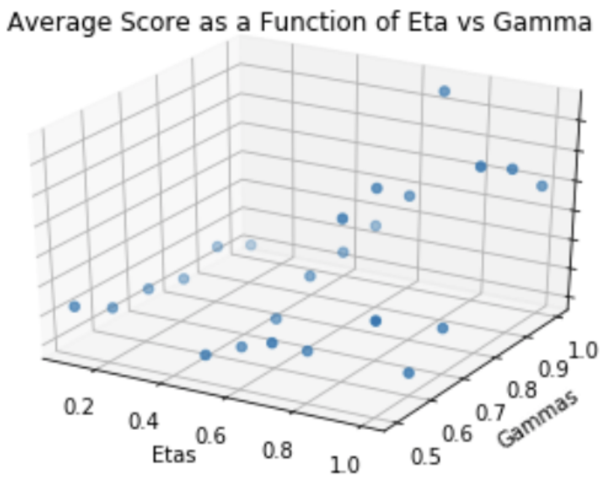

Figure 1

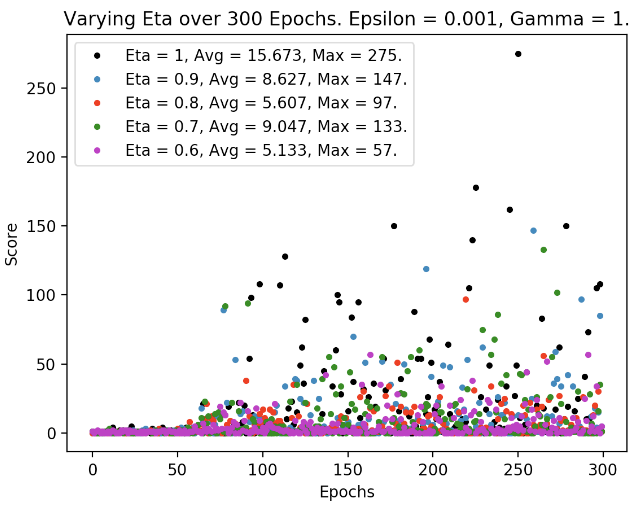

Figure 2

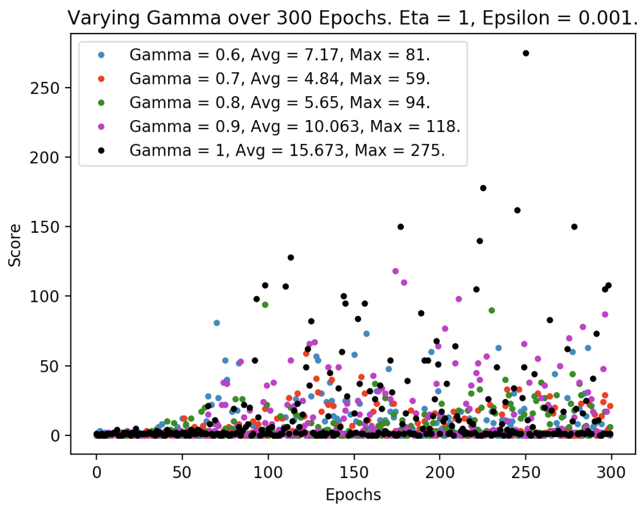

Figure 3

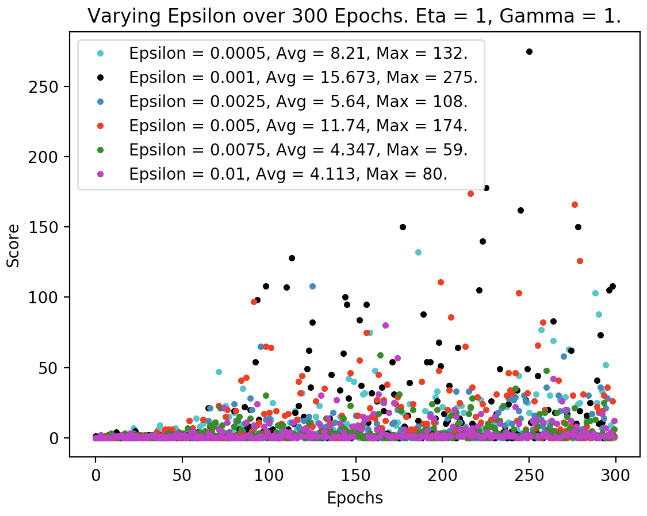

Figure 4

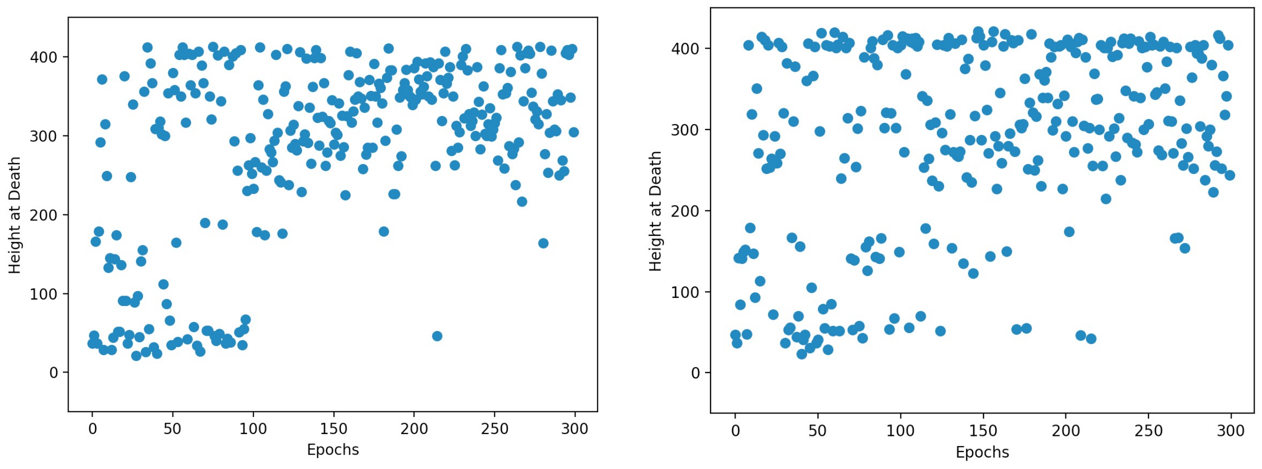

Figure 5

of the η values we tried, we graphed the average scores achieved after 300 epochs in Figure 1 above. With this grid search, we were able to find that $\boxed{\eta = 1, \gamma = 1, \varepsilon = 0.001}$ gave optimal results with an **average score of 15.673 and max score of 275.** Figure 1 is fairly hard to read, and adding the additional dimension of ε would make it even more difficult. Since we don't want to fill up this write-up with tables, instead we created Figures 2-4 above. In these, we permute each parameter around its optimal value found in the grid search while holding the other two constant. This supports our belief that $\eta = 1$, $\gamma = 1$, and $\varepsilon = 0.001$ is a local optima.[1]

Again, note our strategy was to decrease ε and η with the number of times a state-action pair had been visited; the parameters reported above are simply their initializations. As is shown above, with $\eta = 1$, $\gamma = 1$, and $\varepsilon = 0.001$ our agent averaged a score of **15.673** and maxed out at **275** over 300 epochs. Note that these exact values vary on each run, especially the maximum. For instance, on one trial with $\eta = 0.6$, $\gamma = 1$, $\varepsilon = 0.001$ we obtained an average score of 20.173 and a max score of 550 over 300 epochs. However, this was clearly an anomaly run, and the runs we included in Figures 2-4 are all ones that were representative of other runs with those parameters.

All of the above results are for the off-policy, Q-learning model we developed. We also tried an on-policy, SARSA approach but found this usually did slightly worse given the same parameters as the corresponding off-policy model. For instance, our SARSA model with $\eta = 1$, $\gamma = 1$, and $\varepsilon = 0.001$ achieved an average score of 14.22 and maxed out at 187 over 300 epochs. This is extremely similar to our off-policy model and the difference could be due to the intrinsic randomness of the game. Intuitively, it makes sense that these models yielded similar results because our optimal epsilon value was very small and we decreased it iteratively. Thus there was a vary small chance that $\max_{a' \in A} Q(s', a'; \mathbf{w}) \neq \max_{a' \in A} Q(s', \pi(s'); \mathbf{w})$. Specifically, with $\varepsilon = 0.001$, we would expect our policy to cause these to quantities to be not equal at most 1 in 1000 times, and even less the more times a state-action pair is visited.

3 Discussion

We were ultimately successful at improving our model throughout several iterations of it. Initially, we implemented a standard Q-learning approach. We then added ε-greedy randomization and finally decreased ϵ and η as a function of state-action pair visits.

One problem we frequently encountered was that the monkey would learn the state space effectively for around the first 100 epochs, but would then fall into a pattern of dying at the top of the screen after repeatedly jumping too high. We weren't ever able to pinpoint the source of this behavior, but we found that it was reduced after we modified our code to iteratively decrease η and ε. Figure 3 graphs death heights as a function of epochs, with our original model on the left and our final model on the right. While the behavior is still present, it is less noticeable, which suggests that decreasing η/ε helped the model to learn the "top-right" section of pixels in our state space.

We played around with the initialization of the Q-matrix, setting it to values of 0, 1, and -1. We also considered associating jumping with a negative reward and not jumping with 0 in an attempt to reduce the death behavior described above. Ultimately we used 0 because we did not not notice an improvement in model performance over these various, less natural initializations.

[1]This is likely a local optima because we do not try the whole space for each parameter and do not try continuous values.