CS 181 - Machine Learning (Spring 2016)
Practical 4: Reinforcement Learning

Team BridgingtheGAK

Gioia Dominedò (40966234) Amy Lee (60984077) Kendrick Lo (70984997)

April 29, 2016

For this project, we developed multiple agents to play Swingy Monkey, a game in which the player controls a monkey whose objective is to swing on vines and avoid tree trunks while accumulating points. The action space is limited to two options: the monkey can either jump to a new vine, or swing down on the vine that it is currently holding. Points are rewarded for successfully passing tree trunks without hitting them, while also avoiding jumping off the top or bottom of the play area. The difficulty of the task is increased by random variations in: the height of the monkey's jumps; the vertical gaps in the trees; the distances between trees; and the "amount" of gravity. **Our objective was to build an autonomous agent that uses reinforcement learning techniques to learn to play the game and maximize its score.**

1 Technical Approach

Our solution incorporated two different components: a technique to judiciously discretize and reduce the size of the state space, and the implementation of two reinforcement learning agents. We considered also testing model-based agents, but decided to focus our efforts on building and comparing the performance of a model-free Q-learning agent and a hybrid TD-value learning agent. Both agents excelled: the highest score that we achieved was 6,877 (after 600 epochs, which took approximately two hours to complete).

Figure 1: Trained Q-Learning Agent

1.1 Discretizing the State Space

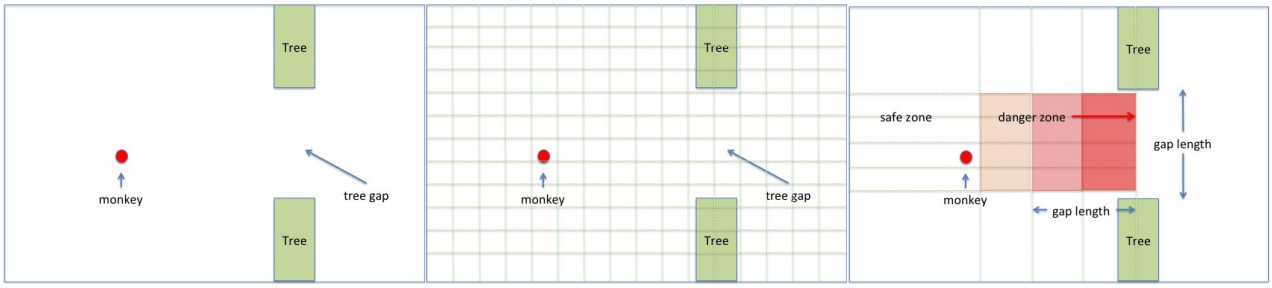
We can frame the learning problem as a **Markov Decision Process (MDP)**, where the agent begins the game without any knowledge of the transition function or reward function. As mentioned above, the action space is restricted to only two options; however, the feedback provided by the environment (in this case, the game interface) at each step is so detailed that we can consider the state space to be almost continuous. Concretely, the information received at each step includes: score (potentially unbounded); number of pixels to the next tree trunk (0-600); screen height of the top of the tree trunk gap (0-400); height of the bottom of the tree trunk gap (0-400); velocity of the monkey (-50-50, based on our observations); screen height of the top of the monkey (0-400); and screen height of the bottom of the monkey (0-400).

Utilizing the information in the format provided (e.g. a tuple with seven elements) would have resulted in very large Q-function, value function, transition, and/or reward matrices (depending on the agent type), which in turn would have required many training rounds in order to fully and accurately populate. To avoid the "curse of dimensionality", we focused a significant part of our efforts on discretizing the state space, trading off the potential information lost through smaller state spaces with the significant improvements in memory and time complexity, as well as training speed.

Naive approach. We initially considered a simplistic approach that splits the anticipated ranges of values for each of the state features into k distinct sections or "bins". While this represents an improvement on the original size of the state space, the number of combinations can still grow unwieldy as k increases. As the state includes seven distinct characteristics, the number of bin combinations is equal to k^7 – even for a seemingly innocuous value of $k = 20$, we would already find ourselves having to track nearly 1.3 billion states! On the other hand, choosing too small a value of k can lead to a discretization that is too coarse-grained and produces sub-optimal results, as two states that call for different actions may end up grouped together as one.

Relative importance of state features. We noted that some of the state features did not appear to be critical to the decision of whether to make the monkey jump or swing down. For example, it seemed reasonable to assume (at least at the outset) that the monkey's action should not be affected by the current score. Similarly, it was not obvious that the monkey's action would be affected by its vertical speed. We did not observe any evidence in our initial test plays that, for example, the monkey did not jump as high if it was previously moving downwards, or that the height of the jump differed based on the current speed. It therefore appeared that we could safely prune the state space by ignoring these features.

Top vs bottom of monkey. The monkey's position is clearly an important consideration, but it was less clear to us that there was an advantage to binning both the top and the bottom of its position. Instead, we created a single feature that measures the center of the monkey's mass by averaging its top and bottom position values, and were able to do away with one more feature.



(a) Simplified screen layout (b) Naive gridding technique (c) Discretization based on gap sizes

Figure 2: Discretizing the state space

Discretization framework. The biggest "wins" from dimensionality reduction followed from a few simple, yet critical observations. Consider, for example, the representation of the state space in Figure 2a. The positions of the top and bottom of the gap in the trees could theoretically span the entire vertical range of the screen. Similarly, the distance between the monkey and the gap could theoretically span the entire horizontal range of the screen. To reflect this, you could consider gridding the entire game screen as shown in Figure 2b, where each grid square represents a state. However, we made three crucial observations that led us to believe that there was a better approach that we could use instead.

- Many of the grid spaces in Figure 2b are unlikely to be visited at all, let alone have any effect on whether or not the monkey can safely maneuver through the tree gap by either jumping or swinging down. It would therefore be a waste of resources to track all of the grid spaces. In addition, the imposition of equally sized grid spaces is overly (and unnecessarily) restrictive.
- We theorized that **the "critical" distance between the monkey and the next tree is heavily dependent upon the size of the gap**. If the gap is small, then the monkey should wait until it is very close to the gap before jumping. Conversely, the monkey can afford to jump earlier if the gap is large, as it runs a lower risk of hitting a tree.
- Similarly, **it is more important to identify the position of the monkey with respect to the height of the gap than the absolute positions of the top and bottom of the trees**. Intuitively, the monkey will need to take a different action if it is close to (or beyond) the gap's edge, compared to when it is vertically aligned with the middle of the gap.

Based on these observations, we chose to redefine the state space into "zones", as shown in Figure 2c. On the horizontal axis, we calculated the distance of the monkey from the gap ($<$ 0.5 gap length, $0.5-1$ gap length, $1-1.5$ gap length, $>$ 1.5 gap length); on the vertical axis, we measured the position of the monkey relative to the size of the gap ($<$ 0.2 gap length, $0.2-0.4$ gap length, $\ldots$, $>$ 0.8 gap length). The monkey's actions are less critical if it is more than 1.5 gap lengths away, as it still has time to take additional (potentially corrective) actions as it gets closer to the gap.

This approach has the advantage of being intuitive, but – more importantly – it also reduces the state space from over 1 billion states to a mere 24! This will result in much faster training times, as each state will be visited "enough" times earlier in the training process.

1.2 Q-Learning

Under this **model-free** approach, the agent learns the Q-function that defines the value of taking a specific action in a specific state, given the optimal policy at the next state. This approach allows the agent to simply maximize the Q-value in the current state at any time, without having to learn a (potentially very large) probabilistic model of the environment or run time-consuming value or policy iteration algorithms. Instead, we simply update the Q-function iteratively every time that the agent transitions from one state to another (i.e. every time the agent transitions to state s' after taking action a from state s, and receives reward r). The update rule is defined as:

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

where γ is the discount factor and α is the learning rate. This is easily implemented, but the agent can take longer to learn an optimal policy than a model-based agent.

1.3 TD-Value Learning

Under this **hybrid model-based/model-free** approach, the agent also learns the transition and reward functions, in addition to the value function. The approach incorporates the best elements of model-based and model-free learning: on the one hand, it gathers additional information on the environment that can be incorporated into decisions; on the other hand, its decision procedure is fast and does not require value or policy iteration. Compared to Q-learning, values are propagated more quickly and actions are selected based on the learned model, which will typically result in the agent learning an optimal policy more quickly.

The transition and reward models for the TD-value agent are learned through a simple maximum likelihood estimate, based on averages of past experiences. For the purposes of this agent, we also defined an additional "terminal" state in order to accurately capture transition probabilities. The update rule is defined as:

$$V(s) \leftarrow V(s) + \alpha\Big[(r + \gamma V(s')) - Q(s,a)\Big]$$

and decisions are made based on:

$$\pi(s) = \arg\max_{a' \in A} \big[R(s,a) + \sum_{s'} P(s'|s,a)V(s')\big]$$

1.4 Epsilon-Greedy

The Q-learning and TD-value learning agents each have a simple decision rule for choosing the optimal action in each state, based on the information that they have already learned. However, the agents must take actions while they are still navigating and learning about the environment. In this context, taking the "best" action may not always be optimal; if the ranking of actions is based on very little information, then it is possible that a greater reward can be obtained through an action that the agent has not yet experienced. On the other hand, always taking the least explored path could result in the agent "wasting time" at the expense of potential immediate rewards.

In order to address this **exploration-exploitation tradeoff**, we incorporated the episilon-greedy algorithm into both the Q-learning and the TD-value learning agents. We tested different values of ϵ, which determines the frequency of exploration, but found that all exploration is sub-optimal due to the relative certainty of certain state-action pairs consistently leading to the monkey's death. This is a direct consequence of the manner in which we discretized the state space.

2 Results

As discussed in the previous section, we found that the key driver of performance was our discretization strategy – much more so than the tuning of learning parameters for each of the agents. In fact, the discretization approach even had the unintended side benefit of making exploration redundant and therefore reducing the number of parameters that we had to test. By building upon the strong foundation of the state space discretization, both agents were able to perform very well in the game environment after relatively few epochs. We expect that further improvements could be achieved through longer training times and, to a lesser extent, additional fine-tuning of the learning parameters.

Discretization. In our final implementation, we fine-tuned the discretization strategy by increasing the number of vertical divisions from 6 to 10. We also added a few more state features: two features associated with the direction of the monkey's movement (i.e. "up" or "down"); three features to track whether the monkey is too close to the top or bottom of the screen; and three features to track the monkey's absolute vertical position on the screen, to account for potential asymmetries between jumping and dropping motions. This brought the total number of states to 720 (4 x 2 x 10 x 3 x 3), which is roughly equal to the number of states that would be obtained by setting $k = 3$ under the "naive" binning approach. Unlike that approach, we were able to focus on discriminating between states where it counts the most, instead of simply dividing up wide swaths of pixel space without purpose.

Q-learning agent. While it had less of an effect on overall performance, we did also experiment with tuning the agent's learning parameters. We found that good results were obtained by keeping the learning rate α very low, and keeping the discount rate γ generally high. Specifically, we started with a lower discount rate of 0.7, which steadily increased and gradually approached 1.0 over time. Intuitively, this corresponds to the agent placing additional emphasis on future rewards when it has already been running for a long time and therefore has more to lose.

The Q-learning agent was able to achieve a maximum score of 1,624 with these parameters and a "default" training time of 100 epochs. When extending the training time to 600 epochs, the agent was able to achieve an even higher maximum score of 5,731, and an average score of 345 (see Figure 3). In order to further test the performance of this longer training run, we stored the resulting Q-function values and used them to play again with the default length of 100 epochs. By using this prior knowledge, the agent achieved substantially higher maximum and average scores in a very short period of time (i.e. 6,877 and 742, respectively). These scores exceeded those from all our earlier attempts (see Figure 4).

TD-value learning agent. When testing the TD-value learning agent, we once again found that the discretization of the state space was the key driver of performance and agent behavior. In fact, we discovered some surprising results when testing different combinations of α and γ learning parameters. We had initially posited that the additional information learnt during training would be the main performance driver. Instead, we realized that the only information of value to the agent related to rewards – so much so, that we found that performance was optimized by setting $\alpha = 0$ and $\gamma = 0$, making the agent's decision function simply $\pi(s) = \arg\max_{a' \in A} R(s, a)$! As with the earlier results on the ϵ parameter, we once again concluded that this was a direct consequence of the manner in which we had discretized the state space.

Despite this surprising behavior, we found that our TD-value learning agent also performed very well and was able to consistently achieve maximum scores of over 2,000 points after only 100 epochs (see Figure 5). We also noted that the game play was not perceptibly slower, even though the agent was updating and storing additional information at each time step. Once again, this is due to the relatively small size of the discretized state space.

3 Discussion

The discussion above details our thought process and the techniques with which we experimented. Our main takeaways are listed below:

- **Dimensionality:** As with all the other practicals, we were once again faced with the challenge of converting a very high-dimensional problem – in this case, a nearly continuous state space – into a more manageable one. The success of our discretization approach highlighted the importance of applying "domain knowledge" (or simply a careful analysis of the problem) when creating and selecting features, instead of relying on algorithms to do the heavy lifting. In fact, we later found that our intelligent remapping of the feature space had the unintended side benefit of making certain parameters less important for overall performance.
- **Imbalanced probabilities:** Our definition of the state space features resulted in very imbalanced probabilities of outcomes for given state-action pairs. For example, if the monkey is very close to hitting the lower part of a tree, it is highly unlikely (or even impossible) that jumping will result in any outcome other than death. These near-certainties made exploration unnecessary, or even detrimental. Similarly, the same characteristic resulted in the TD-value learning agent relying entirely on its immediate rewards and disregarding all other information. While initially counterintuitive, we found that these results were both a direct consequence of our design decisions for the state features, and that they actually made the training process easier.
- **Possible extensions:** Both our agents excelled at playing the game, but we believe that there is still potential for improvement. For example, Figures 3, 4, and 5 show that games with low scores occur fairly frequently, despite the overall positive score trajectory across epochs. We believe that this may be due not only to the randomness of certain elements of the game, but also to variations in gravity from game to game. Given more time, we would have liked to also incorporate inference on the gravity at each epoch, in order to narrow the gap between the minimum and maximum scores across epochs. For example, we could have trained specialized Q-functions for each of the gravity states.

4 Appendix

A Figures

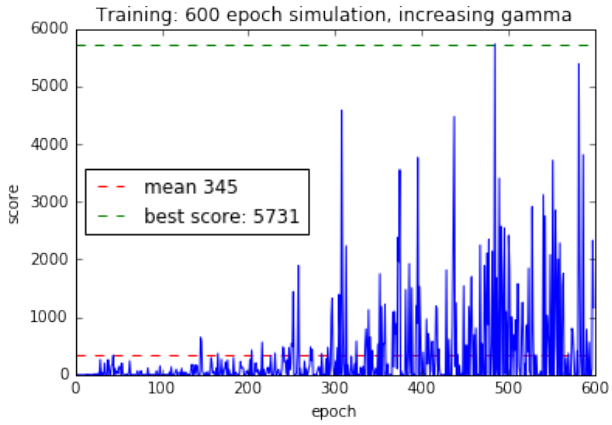


Figure 3: Training the Q-learning agent

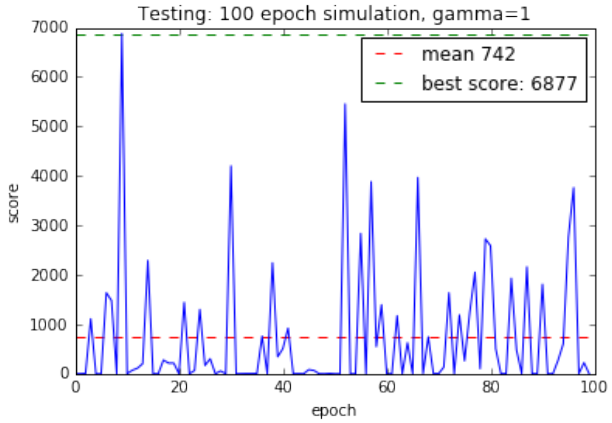


Figure 4: Testing the stored Q-value function

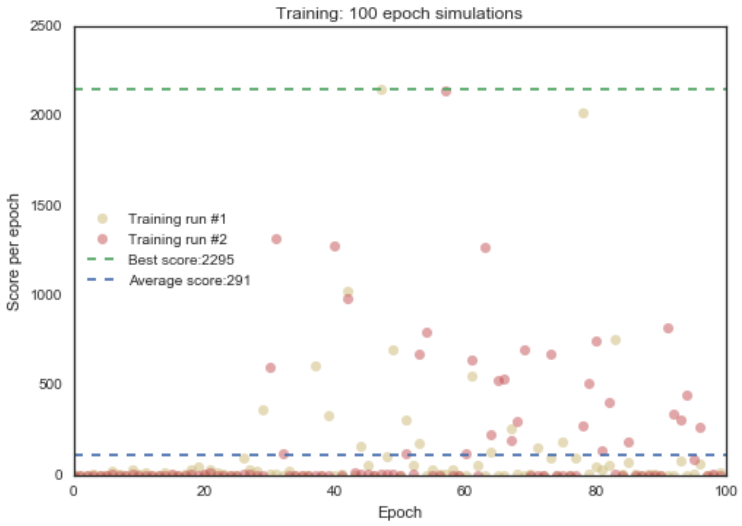


Figure 5: Training the TD-value agent

B Summary Results

B.1 Q-Learning Agent

Initial α	Initial γ	Initial ϵ	Maximum score (Average / Five Individual Runs)	Average score (Average / Five Individual Runs)
0.0	0.0	0.0	0.0 / 0.0; 0.0; 0.0; 0.0; 0.0	0.0 / 0.0; 0.0; 0.0; 0.0; 0.0
0.01	0.0	0.0	325 / 196; 601; 258; 140; 433	21 / 17; 29; 13; 17; 29
0.01	0.5	0.0	405 / 175; 131; 710; 749; 262	27 / 17; 15; 45; 48; 10
0.01	0.7	0.0	707 / 1,028; 318; 196; 368; 1,624	47 / 68; 25; 20; 36; 88
0.01	0.9	0.0	308 / 44; 84; 129; 326; 956	22 / 5; 10; 15; 24; 57
0.1	0.0	0.0	188 / 143; 348; 91; 137; 219	13 / 10; 27; 9; 9; 11
0.1	0.5	0.0	161 / 86; 171; 220; 165; 162	14 / 9; 11; 23; 14; 14

Table 1: Q-Learning Parameter Configurations (100 Epochs)

B.2 TD-Value Learning Agent

Initial α	Initial γ	Initial ϵ	Maximum score (Average / Five Individual Runs)	Average score (Average / Five Individual Runs)
0.0	0.0	0.0	1,282 / 682; 469; 1,325; 676; 2,145	87 / 51; 57; 136; 76; 113
0.01	0.0	0.0	251 / 244; 365; 80; 498; 69	19 / 16; 30; 10; 32; 8
0.01	0.1	0.0	32 / 16; 71; 10; 26; 35	2 / 1; 4; 1; 3; 2
0.01	0.5	0.0	61 / 6; 145; 8; 13; 131	5 / 1; 12; 1; 1; 12
0.01	0.7	0.0	36 / 12; 26; 4; 50; 89	3 / 1; 3; 0; 5; 5
0.01	0.9	0.0	33 / 21; 66; 42; 17; 18	2 / 3; 3; 3; 2; 1
0.1	0.0	0.0	117 / 192; 157; 66; 35; 133	7 / 7; 3; 6; 3; 17
0.1	0.1	0.0	92 / 108; 44; 138; 141; 29	5 / 6; 2; 6; 6; 3
0.1	0.5	0.0	33 / 49; 21; 9; 62; 22	3 / 5; 1; 1; 4; 2
0.1	0.7	0.0	57 / 92; 46; 21; 19; 108	5 / 8; 3; 3; 1; 9
0.1	0.9	0.0	93 / 22; 144; 200; 65; 34	11 / 2; 13; 28; 9; 3

Table 2: TD-Value Learning Parameter Configurations (100 Epochs)