CS 181: Machine Learning
Practical 4: Reinforcement Learning

Sam Lurye, Wanqian Yang, William Gao Yuan
{slurye, yangw, yuan_gao} @college.harvard.edu

May 1, 2018

Technical Approach

We employed both traditional Q-learning as well as approximate Q-learning for this task. For the latter, we used both linear approximation functions as well as deep neural networks. For all of these methods, we modified state representation for better performance, in part due to the large size of the original state space.

Traditional Q-Learning

State Representation: The original state representation had 7 dimensions. This was reduced to 3 as follows: `score` was irrelevant to the mechanics of the game and therefore discarded. Also, since the tree gap and the height of the monkey are constants, we only need to know either the `top` or `bottom` location for both of those objects. Lastly, what matters is not the absolute position of the monkey or the tree, but their relative position, so we proceed with only the difference between `tree.top` and `monkey.top`. For the 3 remaining dimensions (`tree.dist`, `monkey.vel` and `|tree.top - monkey.top|`), the values were split into bins to take away the inessential details of precise position and reduce the state space.

Gravity: Gravity was also added to the state representation, as different games have different gravity and this dimension is crucial to the game mechanics. The precise value of gravity can be inferred from the difference of the monkey's velocity between games ticks before and after a non-action.

The hyper-parameters used were $\alpha = 0.2$, $\gamma = 1$ and $\epsilon = 0.99$ for the learning rate, discount rate and exploitation-exploration ratio respectively.

Approximate Q-Learning

Using Linear Functions

Our first approach to approximate Q-learning is the usage of a linear combination of features as the approximation function, i.e. $Q(s,a) = \sum_{i=1}^{c} w_i f_i(s,a)$, where the weights w_i are modified after each epoch using an off-policy learning strategy:

$$w_i \leftarrow w_i + \alpha \cdot f_i(s,a) \cdot [R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a])$$

Features: We used the original 6 dimensions of the state representation, discarding `score`. The 6 dimensions were also discretized into bins, by dividing the values by a constant D and rounding to the nearest integer. An additional dimension of 1 was added for the bias, as well as a dimension for gravity, inferred in a similar way as done for Traditional Q-Learning.

Experimentation: We experimented with changing the values of α, γ, ϵ and D.

Using Deep Q-Networks

Our next approach was to employ a deep neural network as the approximation function. Our basis vector $\phi : S \times A \mapsto \mathbb{R}^{14}$ is as follows:

$$\phi(s,0) = \big[s_1, s_2, s_3, s_4, s_5, s_6, g, 0, 0, 0, 0, 0, 0, 0\big]$$
$$\phi(s,1) = \big[0, 0, 0, 0, 0, 0, 0, s_1, s_2, s_3, s_4, s_5, s_6, g\big]$$

where s_1 through s_6 are the entries of the state dictionary except for the score, and g is the gravity for the current epoch (obtained by subtracting the velocities between the first two frames in which the monkey does not jump).

To train the network, we followed the strategy described in [1], and the following math is taken from this source. Let $\hat{Q}(s,a;\mathbf{w}_t)$ denote the output of our network for state s and action a at iteration t of optimization. Then the loss for the current set of parameters $\mathbf{w}_t$ is defined as

$$\mathcal{L}(\mathbf{w}_t) = \mathbb{E}_{s,a,s'}\Big[\Big(r(s,a) + \gamma \max_{a'} \hat{Q}(s',a';\mathbf{w}_{t-1}) - \hat{Q}(s,a;\mathbf{w}_t)\Big)^2\Big]$$

Note that our target is given by $r(s,a) + \gamma \max_{a'} \hat{Q}(s',a';\mathbf{w}_{t-1})$ and uses the weights from the previous iteration of optimization. If s' is a terminal state, then our target is simply $r(s,a)$. In practice, this expectation is estimated by using the concept of replay memory. This means that our agent keeps track of the previous m (s_i, a_i, r_i, s_{i+1}) transitions. At each iteration of optimization, we sample a set $\mathcal{S}$ of fixed size of these transitions and update our parameters according to the mean-squared-error loss function

$$\mathcal{L}(\mathbf{w}_t) = \frac{1}{|\mathcal{S}|} \sum_{(s,a,r,s') \in \mathcal{S}} \Big[r(s,a) + \gamma \max_{a'} \hat{Q}(s',a';\mathbf{w}_{t-1}) - \hat{Q}(s,a;\mathbf{w}_t)\Big]^2$$

Note that this is an off-policy learning strategy. To balance exploration and exploitation, we used an ϵ-greedy approach, where $\epsilon(t) = \frac{1}{1+kt}$. We achieved our most consistent results with a network with 4 hidden ReLU layers with 256 neurons each, $m = 128$, $|\mathcal{S}| = 8$, $\gamma = 1$, and $k = 0.01$, using PyTorch's Adam optimizer with learning rate 0.00001.

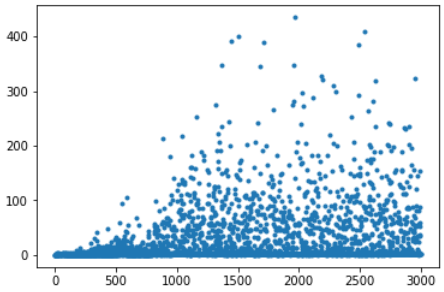
Results

All results are over 1000 training epochs.

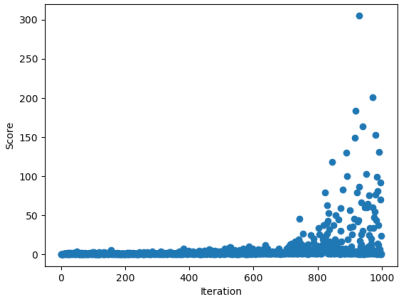
Model	Max Score	Mean Score
1) Traditional, $\alpha = 0.2$, $\gamma = 1$, $\epsilon = 0.99$	214	N/A
2) Linear Approx, $\alpha = 0.005$, $\gamma = 1$, $\epsilon = 1$, $D = 50$	305	6.32
	159	5.70
	41	2.88
3) Linear Approx, $\alpha = 0.0025$, $\gamma = 1$, $\epsilon = 1$, $D = 50$	76	2.91
4) Linear Approx, $\alpha = 0.005$, $\gamma = 0.8$, $\epsilon = 1$, $D = 50$	63	3.60
5) Linear Approx, $\alpha = 0.005$, $\gamma = 1$, $\epsilon = 0.8$, $D = 50$	4	0.47
6) Linear Approx, $\alpha = 0.004$, $\gamma = 1$, $\epsilon = 1$, $D = 100$	62	4.15
7) Linear Approx, $\alpha = 0.0055$, $\gamma = 1$, $\epsilon = 1$, $D = 30$	3	0.37
8) Deep-NN Approx, $k = 0.01$, $\alpha = 0.00001$, $m = 128$, $\lvert\mathcal{S}\rvert = 8$, $\gamma = 1$	202	N/A
9) Deep-NN Approx, $k = 0.01$, $\alpha = 0.00001$, $m = 128$, $\lvert\mathcal{S}\rvert = 8$, $\gamma = 1$	~ 175	N/A
10) Deep-NN Approx, $k = 0.01$, $\alpha = 0.00001$, $m = 128$, $\lvert\mathcal{S}\rvert = 16$, $\gamma = 0.8$	$\sim$150	N/A
11) Deep-NN Approx, $k = 0.01$, $\alpha = 0.00001$, $m = 128$, $\lvert\mathcal{S}\rvert = 32$, $\gamma = 1$	$\sim$160	N/A

Table 1: Results for various models. We ran 3 attempts for model (2). For model (8), 6 hidden ReLU layers of size 256 were used. For models (9) to (11), 4 hidden ReLU layers of size 256 were used.

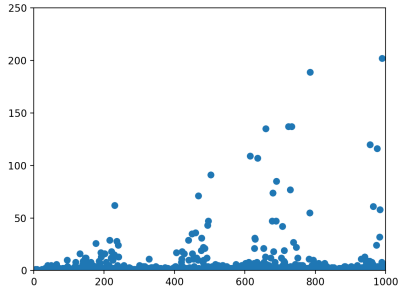
Scatterplots of the score against training epoch for the best performing model for each method are displayed below:



(a) Model 1. *(Refer to first 1000 epochs.)*



(b) Model 2.



(c) Model 8.

Overall, model (2) was the best performing model in 1000 epochs.

Discussion

All 3 methods of reinforcement learning that we employed (traditional Q-learning, linear approximate Q-learning and deep-NN approximate Q-learning) managed to learn the task adequately, with a significant number of epochs generating scores above 20 and with the maximum scores reaching the hundreds range.

One immediate and important conclusion is that there is no significant difference in performance between the 3 methods, with the best attempt for each method generating similar maximum scores after 1000 epochs. This is further supported by the high variance of scores despite using the same hyper-parameters, in evidence for this, we ran model (2) on 3 separate attempts, and the maximum score ranged from 41 to 305 (note that this was the best performance amongst all models), which implies that we should not give too much credence to the maximum score, which is highly variable. Furthermore, the variance is not merely limited to the maximum score, indeed, a stronger proof for the high variance of reinforcement learning for this game is the fact that the mean score (over 1000 epochs) of the 3 attempts of model (2) also ranged widely from 2.88 to 6.32. One possible hypothesis is that the performance of earlier epochs significantly affects how the Q-values are modified and subsequently the performance of later epochs, hence the entire model as a whole involves significant uncertainty ("luck") over learning earlier epochs. Another possible reason is that the distribution of gravity affects learning, since epochs with a gravity value of 1 are harder to learn than those with a gravity value of 4.

For linear approximation functions, it is clear that the optimal discount rate and ϵ-value are both 1, which respectively makes sense because there is no penalty for staying in the game longer, and because it is so easy to die in the game, hence there is a higher penalty for taking random actions (i.e. exploration). Through experimentation, we also found that the optimal bin width is 50, and the optimal learning rate over 1000 epochs is around 0.005. With more epochs, it might make sense to reduce the learning rate.

One interesting observation is the scatterplot for model (8) (deep Q-network approximation), where we can see that the trials in which this agent actually did well were relatively sparse and follow no clear pattern, unlike models (1) and (2) where there is a clear positive correlation between epoch and score, with the scores rising significantly in the last 200 epochs. Ultimately, the scholarly consensus is that the theory behind the convergence of the optimal Q-function using neural networks is poorly understood. This meant that our experimentation with various sets of hyperparameters was basically random. As can be seen in the scatterplot for model (8), in which the Q-function converges to some better-than-random policy, diverges, and then reconverges. We believe that the sparsity in good scores among all agents is a result of the different settings of gravity and the inability of the neural network to encode this. Our next steps would be to maintain separate networks for the different levels of gravity and choose which network to use based on the observed gravity.

References

[1] Mnih, V., Kavukcuoglu, K., Silver D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. *Playing Atari with Deep Reinforcement Learning*. NIPS Deep Learning Workshop 2013. arXiv:1312.5602. `https://arxiv.org/pdf/1312.5602.pdf`.