Practical 4: Reinforcement Learning

Finale Doshi-Velez
finale@seas.harvard.edu
noep1997, jormagoerns, av777x

May 7, 2019

1 Technical Approach

1.1 Discretizing the State Space

The first step we took was to discretize the state space into bins. The screen size of the game is 600x400, which is a prohibitive number of potential states for us to learn on—virtually continuous. We can approximate the state space by dividing the size of the state space by a fixed number so to convert it into "bins." After some experimentation trying bin sizes of 20, 40, 50, 80, and 100 pixels, we decided to go for bins of 50 pixels for the width and height, which gave us 12 bins for the width and 8 bins for the height. We found that this was a reasonably good trade-off between computational complexity and precision. Through printing it, we concluded that the monkey's velocity varied between -50 and $+30$ and thus decided to go for around 5 bins, each of size 20, to again strike a balance between computational complexity and precision.

1.2 Model-Free vs Model-Based Approaches

We then had to decide between model-free and model-based approaches. Model-based RL is possible and performs well when we can have a reasonably good estimation of the reward model and transition probabilities, which is almost impossible for a state space the size of ours. While it would be practical on our discretized state space, we reasoned that the discretization was already a severe approximation—so we decided to go for a model-free approach instead (to reduce the information loss through building an approximation over an approximation), which skips the estimation step and directly learns the policy. To directly infer this policy, we use the Bellman equations:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} [Q^*(s',a')], \forall s, a \tag{1}$$

We can then take the argmax of those $Q^*(s,a)$ to find the optimal policy $\pi^*(s)$. However, we need to find the Q values that satisfy the Bellman equation. In order to do so, we can parameterize our $Q(s,a;\mathbf{w})$ with parameters $\mathbf{w}$ where $w_{s,a} = Q(s,a;\mathbf{w})$. This is essentially adding a weight vector to our model to help us infer the relative weighting of each potential options. Having done this,

we have the following loss function, a vanilla-enough optimization problem, which we can optimize with stochastic gradient descent.

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

There are two ways to optimize this loss by taking a specific action a': One either stays on the policy π and takes the action that the policy dictates, even if it is not optimal—this is the SARSA algorithm (on-policy)—or one takes the optimal action a' every time, even if it is not on the current policy π—this is Q-Learning (off-policy). SARSA is more computationally tractable, but may not converge to the optimal policy, while Q-Learning behaves in the exact opposite manner. We reasoned that, if we bin nicely, we don't need that much computational power anyway since we are drastically reducing the state space, and since we only have few iterations to train, we do need our agent to learn the optimal policy with few epochs. Therefore, we chose to implement Q-Learning and to focus on optimizing its hyperparameters instead of trying various other models like SARSA, which we expected to perform similarly or worse.

1.3 Off-Policy Learning: Q-Learning

Having settled on a model-free, off-policy approach as the ideal way to make our model converge to the optimal policy, we implemented Q-Learning. Q-Learning is ideal for our purpose since it satisfies the Bellman equation above. We derive the Q-Learning update equation for its weights by remembering that Q-Learning chooses the a' that maximizes $Q(s',a';\mathbf{w})$, optimizing the loss through its derivative and following the standard gradient descent formulation:

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

$$w_{s,a} \leftarrow (1-\eta) w_{s,a} + \eta[r + \gamma \max_{a' \in A} Q(s',a';\mathbf{w})] \tag{4}$$

1.4 Exploration-Exploitation Trade-Off

An important trade-off is that of exploitation and exploration. How long should we spend trying new things (exploitation) with uncertain rewards and how long should we spend performing the things we know are good (exploitation)? We first started with an ϵ-greedy policy, in which we had a fixed probability of taking a random action, and then modified it to a decaying ϵ-greedy policy, in which the probability of taking a random action is reduced every time we take such a random action. The idea is that if we explore a lot to begin with, we need to explore less afterwards (since we have already seen a lot)—and eventually, we will arrive at a full exploitation policy (always taking the greedy-optimal action) because we have now learned enough. We then followed the same logic with our learning rate, and implemented a decaying learning rate for each of our bin states. The idea is that, if you been on a certain state more, Q-Learning should have converged to the optimal action and you want to apply that action instead of modifying it over random behavior of other epochs in the training. For both ϵ and η, we decay by dividing their initial value by the number of times we have visited the active state; thus, the more often we visit a state, the more certain we become and the less we explore.

2 Results

We present two tables. In the first table, we display our multiple experimentations with Q-Learning for a fixed set of hyperparameters. The goal was to observe which additions to Q-Learning would make our model perform best. Note that we would either infer gravity (by observing the different monkey velocities if we do not jump after the first timestep) and use it to parameterize our learning according to the gravity value—or not. We then tuned the hyperparameters for this optimal Q-Learning model (epsilon decay, learning rate decay, gravity), as shown in our second table.

Q-Learning Model	Hyperparameters	Max Score	Average Score
Fixed ϵ	$\epsilon = 0.1, \eta = 0.2, \gamma = 0.9$	4	0.0081
Fixed ϵ, gravity	$\epsilon = 0.1, \eta = 0.2, \gamma = 0.9$	6	0.1233
Decaying ϵ	$\epsilon = 0.1, \eta = 0.2, \gamma = 0.9$	38	0.4633
Decaying ϵ, gravity	$\epsilon = 0.1, \eta = 0.2, \gamma = 0.9$	51	0.8931
Decaying ϵ, gravity, decaying η	$\epsilon = 0.1, \eta = 0.2, \gamma = 0.9$	107	1.4428

Table 1: Results for the various types of Q-Learning models we experimented with. The parameters listed correspond to initial values in case of decaying parameters models. We trained for 500 epochs.

Hyperparameters	Max Score	Average Score
$\epsilon = 0.1, \eta = 0.1, \gamma = 0.6$	5	0.2067
$\epsilon = 0.001, \eta = 0.1, \gamma = 0.6$	33	1.1233
$\epsilon = 0.05, \eta = 0.1, \gamma = 0.8$	57	1.7800
$\epsilon = 0.01, \eta = 0.1, \gamma = 0.8$	46	2.0330
$\epsilon = 0.01, \eta = 0.1, \gamma = 0.2$	13	0.3500
$\epsilon = 0.1, \eta = 0.1, \gamma = 0.9$	132	2.5567
$\epsilon = 0.05, \eta = 0.2, \gamma = 0.9$	79	1.8033
$\epsilon = 0.05, \eta = 0.1, \gamma = 0.9$	610	11.960

Table 2: Results for the grid searching of parameters for the optimal Q-Learning model with decaying. Only some of the most instructive grid search results are displayed, as we had 5x5x5 combinations. We trained for 500 epochs.

3 Discussion

The first step in our implementation was to do a careful analysis of how to reduce the state space. As described above, we opted for discretizing the state space into bins of 50 pixels for height and width of the screen, which seemed like a good complexity-simplicity trade-off—this was confirmed by our results. We then considered potential approaches and deemed model-free to be more practical than model-based due to its ability to directly learn the policy—we preferred this since we already had a complex environment and were already approximating it into bins. We opted for Q-Learning due to its guarantee of optimal policy and convergence in fewer epochs, since

computational complexity was not much of a problem after discretizing the space. We reasoned that implementing other non-optimal models would prove to be worse for our results than to work at improving Q-Learning.

We decided to use an ϵ-greedy policy to explore new actions as a way to balance our learning, to contrast the greedy policy of Q-Learning. We first implemented Q-Learning with a fixed ϵ and found the results extremely disappointing. The agent would barely learn, if at all, and our highest score was only 4. We then decided to further model the game by inferring gravity by letting the monkey fall its first timestep and calculating the gravity as the difference between the velocities at time 0 and at time 1. We then used this to discretize our learning over the gravity, so the monkey would learn differently for each gravity. However, we saw small results, with a max score of 6.

We then decided to modify our ϵ policy to decrease the ϵ value over time, dividing it by the number of times we had encountered a state. This way, we had a state-specific ϵ. This made intuitive sense to us as the next step, since an agent might want to explore different approaches with more or less probability depending on the situation they are in, and as they encounter a situation more and more, they will be more likely to take a similar action every time if that action worked. As such, we want to learn more at the beginning (exploration) and transition slowly into exploitation. This made a significant improvement, reaching a max score of 51 when combining it with our gravity implementation. We decided to follow the same reasoning as in our ϵ decay with our learning rate, inferring that when you get more confident after being in a state more often, you want to not only not do random actions, but also not deviate from your good action by learning unnecessary elements, and thus we also decayed our learning rate by dividing it by the number of times we had been in a state. This gave us a state-specific learning rate and boosted our progress significantly, to a max score of 107. We then performed grid search over the initial hyperparameters ϵ, η, γ of our Q-Learning model with gravity, ϵ decay and learning rate decay to fine tune it even further. We found that $\epsilon = 0.05, \eta = 0.1, \gamma = 0.9$ were the optimal parameters, although there were high variance in the results due to the randomness of the monkey. This is not surprising, a high gamma is expected since future rewards are still important to achieve a high score and a small initial eta is ideal to learn iteratively. We were surprised by the very small ϵ (we also performed well, score of 351, with an epsilon of 0.001), which seems to imply that exploration is not very important here. This is likely because there are only two states, jump and don't jump, and thus there is a small sample space of new actions to discover.

There is not much of a concept of overfitting here as the game space is quite large and the agent is specifically trained and optimized for this game. That being said, we note that there is a lot of randomness in the performance of the agent due to how it jumps and the general framework of the game. As such, we could argue that the agent is effectively overfitting some of its Q-values to specific scenarios, especially as we decrease the learning rate as we visit the states. We only trained our agent for 500 epochs, and we expect that as we increase the number of epochs (1,000+) the behavior would become more consistent, although there would always be this randomness factor. In terms of future work, we would like to store the Q-values into a text file so to never lose information and be able to do "experience replay", which is a way in which we can randomly re-sample from previous epochs to decorrelate the training batches, which has been shown to massively increase performance in Atari-agents (DeepMind). We would also like to generalize our Q-Learning work to train on a neural network, building a DQN, which is much more computationally expensive to train, but can achieve super-human performances.