Reinforcement Leaning: Coursework 1

s1667278

27th February 2017

1 Multi-armed bandits

1.1

The formula for softmax with a temperature parameter is:

$$\frac{e^{Q_t(a_i)/\tau}}{\sum_{i=1}^{n} e^{Q_t(a_i)/\tau}}$$

When $n = 2$, the probability of any action a (when the alternative action is a') is

$$\begin{aligned}
&\frac{e^{Q_t(a)/\tau}}{e^{Q_t(a)/\tau} + e^{Q_t(a')/\tau}} \\
&= \frac{1}{1 + e^{\left(Q_t(a') - Q_t(a)\right)/\tau}} \\
&= \sigma\left(\frac{Q_t(a) - Q_t(a')}{\tau}\right)
\end{aligned}$$

where σ is the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The higher the temperature τ, the more similar the action probabilities are. When $\tau \to \infty$, all the action probabilities approach the same value.

The lower the temperature, the greater the difference between the selection probability of the actions given the value estimates. When $\tau \to 0$, the action probabilities approach what they would be under greedy action selection (that is, the highest value action would be chosen).

1.2

The optimistic, greedy method ($Q_0 = 5, \epsilon = 0$) changes its choice of action very often in the beginning since it becomes "disappointed" with its chosen actions. This diversity of actions in the beginning plays introduces variation in the rewards, resulting in spikes and noise in the curve. Also, since it's in exploration mode, it will choose actions that give low rewards instead of exploiting a good option—which results in a lower "% optimal action" in the graph.

In contrast, the realistic, ϵ-greedy method ($Q_0 = 0, \epsilon = 0.1$) chooses a good—but perhaps not optimal—action early on, leading to better performance in the beginning, as well as more stable rewards (since on "average" nine out of ten plays will have the same action and therefore will have similar rewards).

2 Value functions

2.1

The most general equation that we start with is

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s)]$$

Following the optimal policy, we only need to calculate the discounted future rewards when we make the big 10-reward jump (other rewards are 0), which is every five timesteps. Also, calculating every five timesteps, each reward is the same (equal to 10), which further simplifies the calculation. So we have:

$$\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \\
&= (0.9)^0(10) + zeros + (0.9)^5(10) + zeros + (0.9)^{10}(10) + zeros + (0.9)^{15}(10) + ... \\
&= 10\Big(0.9^0 + 0.9^5 + 0.9^{10} + 0.9^{15} + ...\Big) \\
&= 10\left(\sum_{k=0}^{\infty}(0.9)^{5k}\right) \\
&= 10\left(\frac{1}{1-(0.9)^5}\right) \\
&\approx 24.419
\end{aligned} \tag{1}$$

3 Q-learning to play the Enduro game

3.1 Setup

3.2 Discretization

There are two competing effects that arise from discretizing the environment into an 11×10 grid:

- There are a small(er) number of states that are encountered by the agent. This is largely a good thing, since otherwise there would be too many states—and state-action values $Q(s, a)$—to consider, requiring too much training for the agent to encounter each state and learn an effective policy. If the grid was more granular, there would be much more overlap in information between similar grid configurations.
- One grid configuration can represent different situations that give different rewards. For example, when an opponent car is a grid space diagonally ahead, there is not enough granularity to tell whether the player will bump into the other player or if the player will pass, since both the player and the opponent can reside in any part of their respective grid cell. In the first case, the agent will likely get negative rewards (assuming a bump cases the player to be overtaken), and in the other it gets a positive reward of $+1$. However, this loss of granularity is generally not much of a concern: the situation mentioned earlier is perhaps the only one where this granularity would matter, and in most situations it does not matter *exactly* where the opponent cars are.

My state is modeled as the cells surrounding the player. The best agent saw four cells ahead (`forward_visibility = 4`) and seven cells to the side (`side_visibility = 7`). I will discuss why I chose this `forward_visibility` value in section 3.5.

3.3 Random agent

See figure 1 and figure 2. Note that the mean and variance of the distribution are in the histogram title.

3.4 Q-learning agent

The Q-learning agent performed much better than the random agent (figure 1). This is expected: the random agent doesn't learn, and most of the actions in the action set—which are randomly chosen by the random agent—are unlikely to lead to rewards in most situations (e.g., brake). A better naive baseline policy is constant acceleration, which gives a total reward of 14 each episode (figure 1). The Q-learning agent consistently outperforms this "accelerate agent" starting at around episode 50.

Figure 1: Reward by agent type.

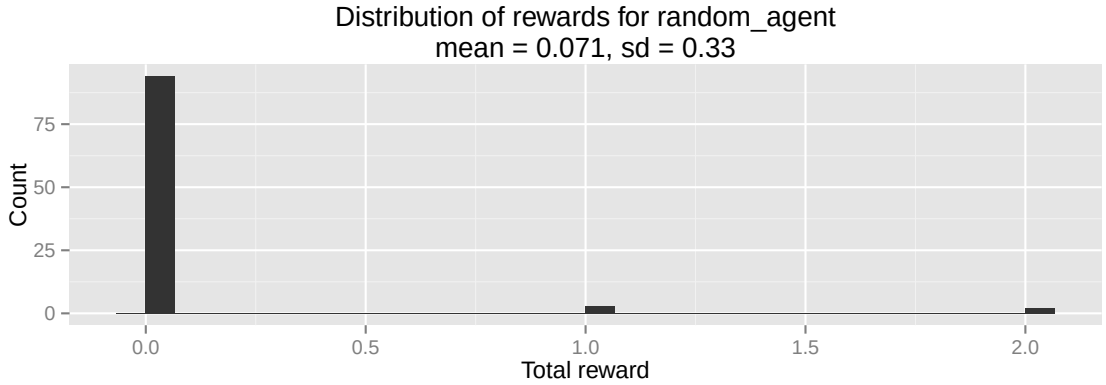

Figure 2: Histogram of rewards for *random_agent*.

3.5 Time horizon

When varying the horizon in either direction, the performance suffers (figure 3).

With close vision, the agent has the benefit of encountering fewer states than the other agents, and therefore has the best performance early on (since more actions will be non-random, and random actions usually result in poor performance). However, the inability to see too far ahead prevents it from learning how to react to more complex state configurations, leading to a dropoff in performance. (I didn't test this, but a quickly accelerating, myopic agent may also not have enough time to prevent a bump.)

With far vision, the complexity of the opponent car configurations increases, meaning more states. These new states, however, may not carry more valuable information; instead, they may confuse the agent since it will always be encountering previously unseen states. This leads to an agent that fails to learn in the 100 episodes—although given more training it *may* eventually overtake the performance of the *q_agent*.

Figure 3: Reward by vision distance. *q_agent* has a vision of 4 grid cells (that is, the agent can see 4 grid cells in front of the player cell), while *q_agent (close vision)* has a vision of 3 and *q_agent (far vision)* has a vision of 6.

3.6 Additional features

Curvature of the road: It seems that the curvature of the road affects the gameplay physics. This information could be incorporated into the model by making the state a combination of the grid

information and the curvature information. If modeling curvature as simply {straight, left right},[1] then I assume this would roughly triple the number of states, likely leading to longer training time but better performance. However, if exploiting left-to-right symmetry (which is what is done in the current build of my *q_agent*), then this would only double the number of states that need to be learned.

Velocity: In the current build, it's impossible for the agent to determine velocity, which may be important for distinguishing between states that would otherwise seem similar. For example, without seeing velocity, the agent can't tell the difference between getting passed and passing another car; if the agent is accelerating but has negative velocity due to a recent bump, it may get passed by an opponent car, therefore penalizing acceleration for all situations when there's an opponent car to the side of the player car. A solution is to take the *difference* of grid states. This would lead to many more states and longer training time, but better performance.

Weather/scenery: I'm unsure if change in weather/scenery (e.g., the white background vs. the green background) has an effect on gameplay. If it does—e.g., if white represented snow and affected the physics of the game—then the state should be modeled as a combination of the grid information and the weather information. This would likely lead to longer training time but better performance.

Optional implementation

See file *q_agent_diff.py* for an implementation that models velocity. It's somewhat hacky in that it requires maximum side visibility (`side_visibility = 9`) to ensure that the subtracted grids are of the same size. Unfortunately I did not save the performance results to file, but I do remember that it did not perform nearly as well as the best-performing *q_agent*. It likely needed more than 100 episodes to train before seeing improvements.

[1] The intermediate curvatures between straight and left and between straight and right do not last very long, and therefore they probably aren't worth modeling separately.