Reinforcement Learning: Coursework 2

s1667278

28 March 2017

Section 1: Actor-Critic Architecture

Description

Briefly, actor-critic methods are a type of temporal difference (TD) method that explicitly separate the policy and the value function. The *actor* is a process that selects actions and maintains the policy by maximizing reward r, often in a way that is greedy and deterministic. The *critic* is a process that estimates the value function and finds a value δ_t called the TD error—often in a way that is *not* deterministic. The TD error is the critic's "critique" of the actions of the actor and it drives the learning of *both* the actor and the critic. It is calculated as

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t).$$

This is used to update the actor's "preference" $p(s, a)$ for choosing some action a in state s using

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

where β is a learning rate hyperparameter [1].[1]

This separation combines the benefits of actor-only methods, which allows for a continues spectrum of actions, and critic-only methods, which have low variance in value estimates but often must discretize the action space due to computational intensity [2]. This grants greater generality and complexity in strategy—e.g., using a explicitly stochastic policy in competitive situations—when compared to other approaches to action selection.

Application

One team of researchers used a least squares temporal difference (LSTD) actor-critic to determine how to efficiently dispatch forklifts in a warehouse. This problem has 200 discrete states once accounting for the combination of the number of products, forklifts and depots. The result performed 20% better than a method that solved for a smaller, more tractable problem and then scaled the solution to the larger problem [4]. This, however, is not a typical case, according to one survey of the actor-critic literature. Most other applications seem most popular in the field of robotics, with some notable applications in finance and operations research [2].

[1] This preference $p(s, a)$ can be then used to calculate the policy probabilities $\pi(s, a)$ using some selection process, like softmax, which is performed over the set of all actions.

Relation to SARSA

The SARSA is an *on*-policy update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

that is performed after every time step. The action selection depends on Q, e.g., using ϵ-greedy.

SARSA can be used as a critic for calculating TD error in an actor-critic method. However, when comparing SARSA as a stand-alone "critic-only" agent against an actor-critic agent, there are a few important differences—namely, that the actor-critic method is (or can be) off-policy and can formulate stochastic policies due to an explicit policy representation. However, they are similar in that the update is performed each time step [1].

Section 2: RL with Function Approximation

Question 1

For a naive reproduction of the tabular case of the Q function, the features vector $\phi_{s,a}$ and the parameters vector θ_t can be constructed as

$$Q_t(s, a) = \begin{pmatrix} \theta_t^T \phi_{1,1} & \theta_t^T \phi_{1,2} & \dots & \theta_t^T \phi_{1,K} \\ \theta_t^T \phi_{2,1} & \theta_t^T \phi_{2,2} & \dots & \theta_t^T \phi_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_t^T \phi_{S,1} & \theta_t^T \phi_{S,2} & \dots & \theta_t^T \phi_{S,K} \end{pmatrix}$$

where K is the number of features and S is the number of states. As with the usual Q-learning table, actions are along the horizontal axis, states are along the vertical axis and the values are the entries in the table. The number of states must account for every possible combination of values in $\phi_{s,a}$. E.g., for binary features with $S = 10$, this would mean $2^{10} \approx 1000$ states [5].

Question 2

A full description of all features are below. Note that being "close to a bump" means that (1) the closest car is at most 2 vertical grid cells away, (2) the closest car is at most 1 horizontal grid cell away, and (3) speed is greater than zero. Opponent cars are only considered if they are within a vertical grid distance of 6 and horizontal grid distance of 3 (both inclusive). Closeness to opponent cars is determined using Euclidian distance in the grid.

- `dist_from_center (float)`: Number of horizontal grid cells from the player car to the center of the grid. Because the grid is 10 cells wide, this number will always be an integer plus one half (e.g., 1.5). Always a positive number.
- `toward_center (bool)`: Does the action cause the player car to move *toward* the center of the grid? E.g., is the action `LEFT` while the player car is on the right side of the grid?
- `away_from_center (bool)`: Does the action cause the player car to move *away from* the center of the grid?
- `action_accel (bool)`: Is the action `ACCELERATE`?

- `car_ydist (int)`: The number of vertical grid cells to the closest opponent car. If the number is greater than 6 or if there are no opponent cars, then this number is 6.
- `car1_xdist (int)`: The number of horizontal grid cells to the closest opponent car. If the number of greater than 3 or if there are no opponent cars, then this number is 3.
- `toward_car1 (bool)`: Does the action move the player car (horizontally) *toward* the closest opponent car?
- `away_from_car1 (bool)`: Does the action move the player car (horizontally) *away from* the closest opponent car?
- `close_to_bump_accel (bool)`: Is the player car close to bumping, and is the action `ACCELERATE`?
- `close_to_bump_left_or_right (bool)`: Is the player car close to bumping, and is the action either `LEFT` or `RIGHT`?
- `toward_car2 (bool)`: Does the action move the player car (horizontally) *toward* the second-closest opponent car?
- `away_from_car2 (bool)`: Does the action move the player car (horizontally) *away from* the second-closest opponent car?

The features satisfy the behavioral requirements in the following ways:

- **Collisions should be avoided.** This is accomplished with the variables `close_to_bump_accel` and `close_to_bump_left_or_right`. In theory, `close_to_bump_left_or_right` should have a positive weight, giving preference to horizontal movement when a bump is imminent, while `close_to_bump_accel` should have a negative weight since it will likely result in a bump. `car_ydist` may also accomplish this task, but more "softly" than the two hard binary features, providing a signal for approaching cars that may not be in the immediate vicinity of the player car.
- **Moving faster results in passing by more cars.** This is accomplished with `action_accel`, since this resulted in a bias for acceleration. Using speed alone—or speed combined with actions—as feature definitions was not helpful, often resulting in negative weights.
- **Staying in the centre of the road is preferred when possible.** This is accomplished with `dist_from_center`, `toward_center` and `away_from_center`.

Question 3

Part a

The linear function approximation (LFA) agent learns much more quickly than the Q-learning agent (figure 1). This is due to two (related) reasons: (1) the Q-learning agent's inability to generalize means many of the states in the beginning epoches are seen for the first time, and the agent has to learn an optimal action for each of these separately; and (2) the LFA agent was initialized to have a bias for forward acceleration.[2]

[2]It is a bit harder to do this for the Q-learning agent, since this initial bias must be built into every state where it is appropriate.

The Q-agent suffers from a brief dip in performance around epoch 200; this could be due to a specific state that got linked to a bad policy action, e.g., turning left into the side of the road. This can be hard to escape since negative rewards are not received when the player has not yet passed any cars.

Both agents seem to settle on an average reward of between 20 and 30 at the end of 500 epochs. However, it is not fair to compare performance between the agents since both were very sensitive to the implementation or settings of of the features (in the case of the LFA agent), the state (in the case of the Q-learning agent) and hyperparameters (in the case of both agents)—so small changes to these could greatly influence the results.

Figure 1: Reward by Q-learning and linear function approximation agents. Since the rewards were noisy, a LOESS-smoothed curve was added for each series.

Part b

Figure 2 shows the weights of the 12 features used by the LFA agent. It seems that the most important features are the non-binary ones, since their large weights get multiplied by values in $\phi_{s,a}$ that are often larger than 1—and they therefore have the largest influence on Q. In particular, having a large horizontal distance between the player car and the opponent car, and being close to the center of the road, are most important.

Of the binary features, the most important are ones that represent turning away from the closest opponent car, turning toward the middle of the road, and accelerating. The LFA agent still had an issue with bumping: perhaps this is because `close_to_bump_accel` was unexpectedly (slightly) positive and `close_to_bump_left_or_right` was not quite positive enough. It is possible this could have been remedied by more episodes.

As expected, features that represented turning toward either of the opponent cards and turning away from the center of the road had negative weights—although not strongly so.

Feature type	Feature	Value
Binary	away_from_car1	0.1879
	toward_center	0.1763
	action_accel	0.1395
	close_to_bump_left_or_right	0.0188
	close_to_bump_accel	0.0006
	toward_car1	-0.0035
	away_from_car2	-0.0140
	away_from_center	-0.0339
	toward_car2	-0.0362
Non-binary	car1_xdist	0.1957
	car1_ydist	-0.0044
	dist_from_center	-0.2120

Figure 2: Feature weights.

Part c

Figure 3 shows the weights of the features over time. All weights were initialized to zero except for `action_accel`, which was initialized to one.[3] While guessing at reasonable weight resulted in better performance at the start, it was more interesting to see whether convergence was possible with less coaching.

Interestingly, `action_accel` first decreases before recovering to be above the initialized value, likely in response to the increase in magnitude of the other weights. Most other weights go somewhat smoothly from their initial values to the final values, with the non-binary variables like `car1_xdist` showing a bit more volatility than the binary features.

Unlike the traditional Q-learning algorithm, these weights can offer insight into why the model behaves in a certain way, providing insight into the reason for undesirable behavioral patterns. In the implementation of Q-learning used for this assignment (and in the last assignment), the state had far too many dimensions to visualize changes in the table over time.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[2] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[3] It was discovered that simply using zeros instead of random normal values helped early training.

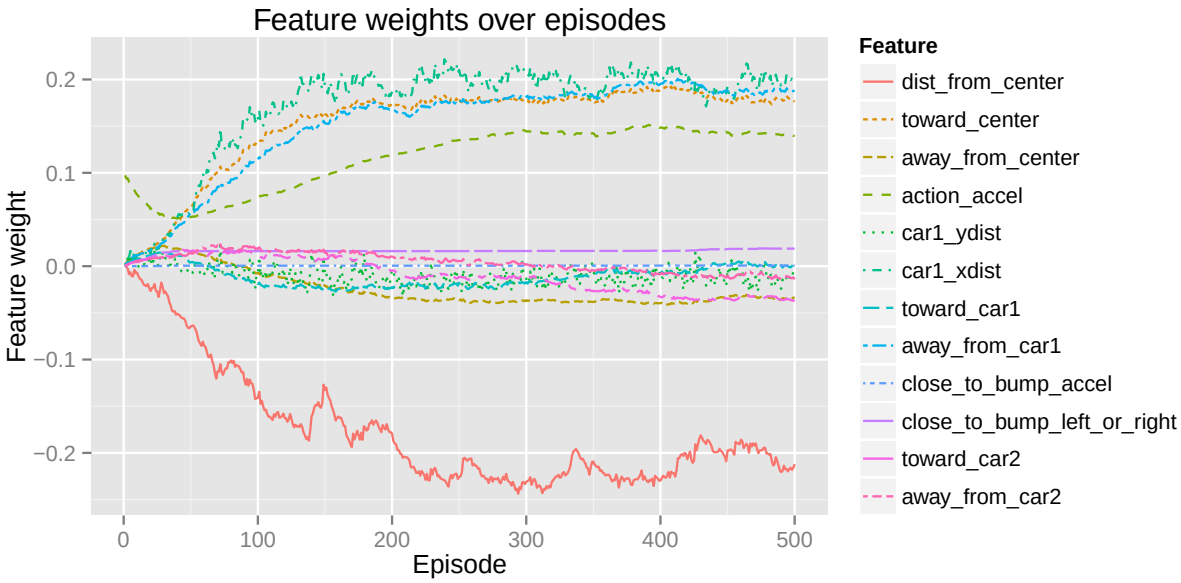

Figure 3: The values of the 12 features used by the LFA agent.

[3] C. Raju, Y. Narahari, and K. Ravikumar, "Reinforcement learning applications in dynamic pricing of retail markets," in *E-Commerce, 2003. CEC 2003. IEEE International Conference on*, pp. 339–346, IEEE, 2003.

[4] I. C. Paschalidis, K. Li, and R. M. Estanjini, "An actor-critic method using least squares temporal difference learning," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pp. 2564–2569, IEEE, 2009.

[5] A. Geramifard, T. J. Walsh, S. Tellex, G. Chowdhary, N. Roy, J. P. How, *et al.*, "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 6, no. 4, pp. 375–451, 2013.