CS6700 : Reinforcement Learning
Written Assignment #2

Sai Vinay G
CE17B019
Indian Institute of Technology, Madras

April 3, 2019

Problem 1

(a) Approximating a value function and determining a policy has been proven to be theoretically intractable. Instead of approximating a value function and using that to compute a deterministic policy, we can approximate a stochastic policy using policy gradient.

And in continuous action domains, in value functions methods would require to select an action (ex. using $\epsilon - greedy$), we need to optimize over the action space for every action selected in order to estimate optimal action. In policy gradient we directly get the probability distributions so, it becomes easy to sample actions from them.

policy gradient methods have better convergence properties and effective in high-dimensional or continuous action spaces.

(b) If there has been an optimal policy in the class of policies, then we could sample episodes according to the optimal policy then, value function based would be better than policy gradient methods as value function based methods will learn optimal behaviour quickly than policy gradient methods .

(c) If the state-action space is very small then either of the methods can possibly learn the optimal policy.

Problem 2

(a) Given, MDP with states with s_1, s_2 and actions a_1, a_2
There are there features each for state s_1, s_2 : $\phi_1(s_1) = 1, \phi_1(s_2) = -1, \phi_2(s_1) = -1, \phi_2(s_2) = -1, \phi_3(s_1) = -1, \phi_3(s_2) = 1$

$$\phi(s_1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \phi(s_2) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

Linear function approximator of state value function :

$$V(s) = \theta^T \phi(s)$$

Therefore from above features we get,

$$V(s) = \begin{bmatrix} V(s_1) \\ V(s_2) \end{bmatrix} = \begin{bmatrix} \theta_0 - \theta_1 - \theta_2 \\ -\theta_0 - \theta_1 + \theta_2 \end{bmatrix}$$

Consider $\theta_0 - \theta_2 = a$, Now V(s) becomes

$$V(s) = \begin{bmatrix} a - \theta_1 \\ -(a + \theta_1) \end{bmatrix}$$

Now the terms $a - \theta_1, -(a + \theta_1)$ are linearly independent as

$\alpha(a - \theta_1) + \beta(-(a + \theta_1)) = 0$ gives $\alpha = 0, \beta = 0$

As V is 2-dimensional quantity and formed using independent vectors, it spans the entire 2 dimensional space. So, all the values in 2-dimensional space can be represented using only the given features in a linear function approximator.

(b) Given, the experience is $s_2, a_2, -5, s_1, a_2$
The Linear gradient descent TD(0) update equation is :

$$\theta_{t+1} \leftarrow \theta_t + \alpha\big(R_{t+1} + \gamma V_t(S_{t+1}) - V_t(s_t)\big)\nabla_{\theta_t} V_t(s_t)$$

So , we get

$$\theta_{t+1} \leftarrow \theta_t + \alpha\big(- 5 + \gamma V_t(s_1) - V_t(s_2)\big)\nabla_{\theta_t} V_t(s_2)$$

$$\begin{aligned} V(s_2) &= \theta\phi(s_2) \\ \nabla_{\theta_t} V_t(S_t) &= \phi(s_2) \\ &= \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \theta_{t+1}^{[1]} &\leftarrow \theta_t^{[1]} - \alpha\big(- 5 + \gamma V_t(s_1) - V_t(s_2)\big) \\ \theta_{t+1}^{[2]} &\leftarrow \theta_t^{[2]} - \alpha\big(- 5 + \gamma V_t(s_1) - V_t(s_2)\big) \\ \theta_{t+1}^{[3]} &\leftarrow \theta_t^{[3]} + \alpha\big(- 5 + \gamma V_t(s_1) - V_t(s_2)\big) \end{aligned}$$

Here $\theta_t^{[i]}$ denotes the i^{th} element of θ_t

Problem 3

(a) Implementing TD(λ) using a linear function approximator(FA)
Consider value function of state s as dependent on θ as $V(s;\theta)$ and feature vector of state s as $\phi(s)$ we have

$$V(s) = \theta^T \phi(s)$$

The TD(0) update equation is

$$\vec{\theta}_{t+1} \longleftarrow \vec{\theta}_t + \alpha\delta_t \nabla_{\vec{\theta}_t} V_t(s_t)$$

$$\vec{\theta}_{t+1} \longleftarrow \vec{\theta}_t + \alpha\delta_t \phi(s_t)$$

where δ_t is the TD error

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

Using eligibility traces the update equation is:

$$\vec{\theta}_{t+1} \longleftarrow \vec{\theta}_t + \alpha\delta_t \vec{e}_t$$

For accumulating traces :

$$\begin{aligned} e_t &= \lambda e_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t) \\ &= \lambda e_{t-1} + \phi(s_t) \end{aligned}$$

For replacing traces :

$$e_t^i(s) = \begin{cases} \gamma\lambda e_{t-1}^i(s), & \text{if } \phi^i(s_t) = 0. \\ 1, & \text{if } \phi^i(s_t) = 1\ . \end{cases} \quad For\ all\ i$$

Here e^i denotes i^{th} component of e_t
In normal $TD(\lambda)$ without function approximation the eligibility traces

Algorithm

```
Initialize value function parameters $\vec{\theta}$ arbitrarily

Repeat (For each episode):
    Initialize state s of episode i.e, feature $\phi(s)$
    Initialize eligibility traces $\vec{e}_t = 0$

   Repeat (For each state in the episode):
        Get action from policy $\pi$ for state $s$
        Take action $a$, observe reward $r$, state $s'$ i.e, feature $\phi(s')$
        $V(s) = \theta^T\phi(s)$ , $V(s') = \theta^T\phi(s')$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $\vec{e} \leftarrow \gamma\lambda\vec{e}_t + \phi(s)$
        $\vec{\theta} \leftarrow \gamma\lambda\vec{\theta} + \alpha\delta\vec{e}$
        $s \leftarrow s'$
    until s is terminal state
```

(b) Accumulating traces are better than replacing traces in this case
Replacing traces are not applicable when using function approximation methods as states are not represented uniquely by binary value feature vectors but whereas replacing traces require states to be represented uniquely by binary values.

Accumulating traces can be used even when the states are not represented using binary features. Hence accumulating trace is preffered in this case.

Problem 4

- DQN uses semi-gradient methods for updating weights.
  The given update is:
  $$\Delta\theta = \alpha(R_{t+1} + \gamma V(S_{t+1}, \theta) - V(S_t, \theta))\nabla_\theta V(S_t, \theta)$$
  1.The update is wrong. It should be :
  $$\Delta\theta = \alpha(R_{t+1} + \gamma V(S_{t+1}, \theta^-) - V(S_t, \theta))\nabla_\theta V(S_t, \theta)$$
  2.The gradient of $V(s_{t+1}, \theta^-)$ is missing.

  This is because we make the target Q-value's weights independent of the estimated Q-value parameters i.e, θ^- is independent of θ. This is because we don't want to our target to change continuously with the network parameters, then each time we update our parameters the target moves somewhere which doesn't result in successful training. So, the target networks weights are fixed and updated at some intervals to the parameters of estimated Q-values.

- Two ways to update the target network parameters :
  1. periodically update i.e, replace the target network weights by the main training network's weights

  2. frequently update the target network slowly towards the main training network by weighted addition of the weights of target network and main training network .
  $$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^- \quad \textit{where } \tau \ll 1$$
  Second method is better as we slowly move towards the main training network which stabilizes training, when compared to the first method where all the parameters change at once to the main training parameters.

Problem 5

- Experience replay in DQN is used to reduce the problem of correlated data and non-stationary distributions. Deep learning models assume data samples to be independent and fixed underlying distribution but whereas, in RL we generally encounter sequences of highly correlated states and the data distribution changes as we learn more.
  In experience replay we store agent's experience at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ into a replay buffer. While training sample a batch of experiences and Q function is updated.

  So, as we sample a batch from replay memory, the behavior distribution is averaged over many of its

previous states which leads to smoothing out learning
As we sample uniformly at random from the replay buffer, the data is more independent of each other and is closer to i.i.d.

- Higher TD-error δ transitions, which indicates how far is the current value and its next-step bootstrap estimate in the transition. So, these are the transitions are more significant in learning of agent as the estimates are not good for the values in the current transition.

  As, previously while sampling uniformly at random we were not looking for any importance of transitions for learning purpose, which may lead to redundant updates. Using this way we can learn more quickly the significant transitions for more number of times.

Problem 6

Policy Gradient method: In this method, the policy is explicitly represented using its own function approximator, independent of the value function, and is updated according to the gradient of the expected reward with respect to the policy parameters.
The parameters update in modern actor-critic is:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$$

Reinforcement comparison method: In this method we maintain a measure of their preference for each action. The preferences are updated using the reward generated and reference reward. The preferences might be used to determine action-selection probabilities.
The parameter update equation is :

$$p(s_t,a_t) \leftarrow p(s_t,a_t) + \beta[r_t - \overline{r_t}]$$

In **original version** of actor-critic, actions are generated using gibbs softmax method, where preferences of actions are used to determine action-selection probabilities.

$$\pi_t(s,a) = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}}$$

The preferences are updated as:

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

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

Consider the same gibbs softmax methods using **modern actor-critic**, the preferences are parametrized so action selection probability is :

$$\pi(s,a) = \frac{e^{\phi(s,a)^T}\theta}{\sum_b e^{\phi(s,b)^T}\theta}$$

The parameters are updated as :

$$\theta_{t+1} = \theta_t + \alpha\big[\phi(s,a) - E_{\pi_\theta}[\phi(s,a)]\big]Q_w(s,a)$$

We can see that the update equations are different for old and modern version

In the old version we are updating the preferences using the TD error i.e , difference between target (sampled return) and current value function in the value estimate whereas In policy gradient, we update the parameters based on the gradient of expected return . As the old version doesn't involve any gradient of expected returns with respect to policy parameters, the old version is not a policy gradient method.

References

[1] K. Främling, “Replacing eligibility trace for action-value learning with function approximation.” in *ESANN*, 2007, pp. 313–318.

[2] J. Hui. Rl dqn deep q-network. [Online]. Available: https://medium.com/@jonathan$_{h}ui/rl-dqn-deep-$ $q-network-e207751f7ae4$

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[6] H. Seijen and R. Sutton, “True online td (lambda),” in *International Conference on Machine Learning*, 2014, pp. 692–700.

[7] D. silver. Value function approximation. [Online]. Available: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching$_{f}iles/FA.pdf$

[8] R. sutton. Gradient-descent methods. [Online]. Available: http://www.incompleteideas.net/book/ebook/node87.html

[9] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[10] S. Zhang and R. S. Sutton, “A deeper look at experience replay,” *arXiv preprint arXiv:1712.01275*, 2017.