Chapter 8 Value Function Approxiamation

TD of state values with function approximation, Sarsa with function approximation, Q-learning with function approximation, Deep Q-learning

TD of state values with function approximation

1. What is the improvement of value function methods based on TD learning? What are the advantages and disadvantages?
2. Explain two main function methods for value estimate.
3. How to find optimal *w* so that the function can best approximate $v_\pi(s)$?
4. Explain the probability distribution of states in objective function.
5. How to minimize objective function?

Sarsa with function approximation

6. Explain the formula of q value update formula.
7. The process of Sarsa with function approximation.

Q-learning with function approximation

8. Explain the formula of q value update formula.
9. The process of Q-learning with function approximation.

Deep Q-learning

10. What is the differece between Q-learning with function approximation and Deep Q-learning?
11. Explain the experience replay method of DQN.
12. Explain the parameter updating method of DQN.
13. Explain the process of DQN.

1. The state values/action values are represented in function form rather than tabular form. We use a function to fit the values. Thus this will decrease the price of storing but sacrify accuracy. In addition, function method has a better generalization ability because updating a value will cause the change in the param of the function which results in updating some other values.
2. First is linear function approximation. *Φ(s)* is the feature vector, *w* is the parameter vector. We should fix *Φ(s)* and update *w* to fit the values. However, it is difficult to decide the length of *Φ(s)* cause we need a pre-understanding of the task. If is combined with TD for function approximation, it can be called TD-Linear.

$$\hat{v}(s,w) = as^2 + bs + c = \underbrace{[s^2, s, 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{w} = \phi^T(s)w$$

Second is artificial nerual networks. The input is a state and out put is the estimation of state value. Artificial neural networks can approximate values as black-box universal nonlinear approximators, which are more friendly to use.

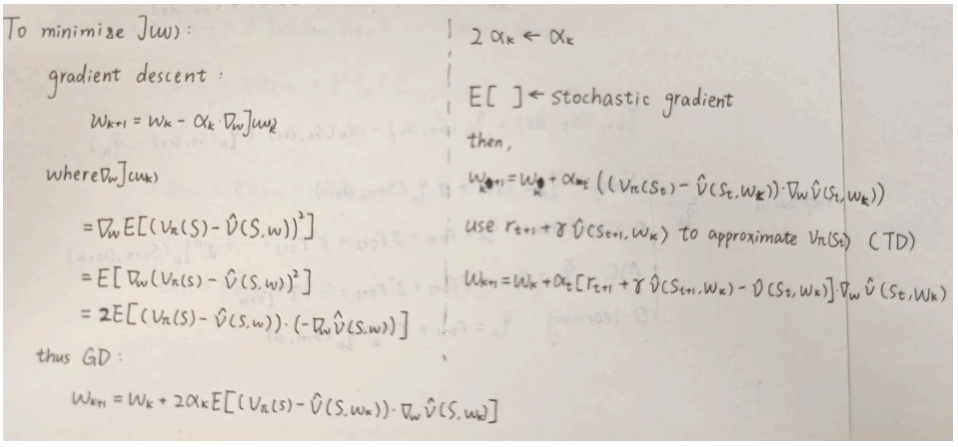
3. We define objective funtion and minimize it:

$$J(w) = \mathbb{E}[(v_\pi(S) - \hat{v}(S, w))^2]$$

4. As the agent executes a policy for a sufficient long time, the probablity of visiting the states will become stable and this is the stationary distribution $d_\pi(s)$. This is the long-term behavior of Markov decision process. Thus, *J(w)* can be written as

$$J(w) = \sum_{s \in \mathcal{S}} d_\pi(s)(v_\pi(s) - \hat{v}(s, w))^2$$

5.



Algorithm 8.1: TD learning of state values with function approximation

Initialization: A function $\hat{v}(s, w)$ that is differentiable in w. Initial parameter w_0.
Goal: Learn the true state values of a given policy π.

For each episode $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by π, do
    For each sample (s_t, r_{t+1}, s_{t+1}), do
        In the general case, $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \nabla_w \hat{v}(s_t, w_t)$
        In the linear case, $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T(s_{t+1}) w_t - \phi^T(s_t) w_t\right] \phi(s_t)$

6.

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t)\right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

7.

Algorithm 8.2: Sarsa with function approximation

Initialization: Initial parameter w_0. Initial policy π_0. $\alpha_t = \alpha > 0$ for all t. $\epsilon \in (0, 1)$.
Goal: Learn an optimal policy that can lead the agent to the target state from an initial state s_0.
For each episode, do
  Generate a_0 at s_0 following $\pi_0(s_0)$
  If s_t $(t = 0, 1, 2, \dots)$ is not the target state, do
    Collect the experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given (s_t, a_t): generate r_{t+1}, s_{t+1} by interacting with the environment; generate a_{t+1} following $\pi_t(s_{t+1})$.
    Update q-value:
    $w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t)$
    Update policy:
    $\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$
    $\pi_{t+1}(a|s_t) = \frac{\epsilon}{|\mathcal{A}(s_t)|}$ otherwise
    $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$

8.

$$w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t)$$

9.

Algorithm 8.3: Q-learning with function approximation (on-policy version)

Initialization: Initial parameter w_0. Initial policy π_0. $\alpha_t = \alpha > 0$ for all t. $\epsilon \in (0, 1)$.
Goal: Learn an optimal path that can lead the agent to the target state from an initial state s_0.

For each episode, do
  If s_t $(t = 0, 1, 2, \dots)$ is not the target state, do
    Collect the experience sample (a_t, r_{t+1}, s_{t+1}) given s_t: generate a_t following $\pi_t(s_t)$; generate r_{t+1}, s_{t+1} by interacting with the environment.
    Update q-value:
    $w_{t+1} = w_t + \alpha_t \Big[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \Big] \nabla_w \hat{q}(s_t, a_t, w_t)$
    Update policy:
    $\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1)$ if $a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1})$
    $\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s_t)|}$ otherwise

10. DQN uses a special param updating method and experience replay method. What's more, in terms of derivation, $q_\pi(s,a)$ is approximated by TD spirit before calculating its gradient. It makes it difficult to calculate gradient thus we introduce a special param updating method and experience replay method. The reason why we make this trouble is because DQN is combined with deep networks which has high dimension parameters and will cause unstable learning process.

11. Because it is difficult to calculate the derivative of "max" item in J(w), we denote w in "max" item as w_T. Then we have two networks which are called target network w_T and main network w. w_T and w have the same initial value. We fix the value of w_T every C iterations and then update $w_T \leftarrow w$.

12. We collect all the experience examples into the replay buffer. For ever iteration, select a mini-batch of examples {(s, a, r, s')} following i.i.d and use them for training. There are two reasons for introducing this method: Firstly, the initial sequence gained from π_b has a specific order. If we use this order it may cause over-training of some specific (s,a). Thus it is crucial to break the correlation between samples in the sequence to meet uniform distribution. Secondly, each example may use more than once to increase data efficiency.

13.

Algorithm 8.3: Deep Q-learning (off-policy version)

Initialization: A main network and a target network with the same initial parameter.
Goal: Learn an optimal target network to approximate the *optimal* action values from the experience samples generated by a given behavior policy π_b.

Store the experience samples generated by π_b in a replay buffer $\mathcal{B} = \{(s, a, r, s')\}$
For each iteration, do
Uniformly draw a mini-batch of samples from $\mathcal{B}$
For each sample (s, a, r, s'), calculate the target value as $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$, where w_T is the parameter of the target network
Update the main network to minimize $(y_T - \hat{q}(s, a, w))^2$ using the mini-batch of samples
based on GD algorithm
Set $w_T = w$ every C iterations