

Lecture 8: Reinforcement Learning

Scribe: *Sayan Paul, Yash Shahapurkar*

October 23rd, 2018

1 Earlier

A Markov decision process is characterized by $\langle S, A, T, R \rangle$

S: set of states, A: set of action, T: transition function, R: reward function,

2 Reinforcement Learning

In reinforcement learning, T and R are not defined. A policy is learned only from experience.

2.1 Learn Value Function for particular policy

Learn new estimate of Value Function, given old estimate and step size,
error = target - old estimate

$$\begin{aligned} V(s_t) &= V(s_t) + \alpha(R + \gamma V(s_{t+1}) - V(s_t)) \\ G_t &= R_t + \gamma V(s_{t+1}) \\ G_t &= \text{target}, R_t : \text{reward} \\ \therefore V(s_t) &= V(s_t) + \alpha(R_t + \gamma V(s_{t+1}) - V(s_t)) \end{aligned}$$

This algorithm is called **Temporal Difference Learning (TDL)**

Initially, $V(s) = 0$ for all s

Loop

Take action a at state s

Observe r,s'

$$V(s_t) = V(s_t) + \alpha(R + \gamma V(s_{t+1}) - V(s_t))$$

$s < -s'$

until s at terminal state

In dynamic programming, the estimated value was expected as

$$V_{\pi}(s_t) = E[R(s_t, a_t) + \gamma V_{\pi}(s_{t+1})]$$

Here we don't take expectation, but we sample (one sample of next state) Thus we approximate the expected value obtained by DP

2.2 State action value function

The state action value function $Q_{\pi}(s_t, a_t)$ is defined as the expected reward by performing action a , in state s , and following policy π

$$Q_{\pi}(s_t, a_t) = R(s_t, a_t) + \gamma E[V_{\pi}(s_{t+1})]$$

$$\begin{aligned} V_{\pi}(s_t, a_t) &= R(s_t, \pi(s_t)) + \gamma E[V_{\pi}(s_{t+1})] \\ Q_{\pi}(s_t, a_t) &= Q(s_t, a_t) + \alpha [R(s_t, a_t) + \gamma Q_{\pi}(s_{t+1}, a_{t+1}) - Q_{\pi}(s_t, a_t)] \end{aligned}$$

To do this we need $s_t, a_t, R_t, s_{t+1}, a_{t+1}$ for updating. This is called **SARSA**

1. Initialize s
2. Repeat for N runs
3. Choose A from S using policy from Q
4. Take action A , observe R, S'
5. Choose A' from S' using policy from Q
6. $Q(s, a) = Q(s, a) + \alpha [R + \gamma Q(s', a')]$
7. $s < -s', a < -a'$

Q can be, for example, an epsilon greedy policy

Epsilon greedy: Policy where action that maximizes Q

$a = \operatorname{argmax}_{a'} Q(s, a)$ with $p = 1 - \epsilon$.

Usually, $\epsilon = 1/t$, as initially we want to explore more and then exploit.

For optimal policy

$$\begin{aligned} Q^*(s_t, a_t) &= R(s_t, a_t) + \gamma E[V^*(s_{t+1})] \\ &= R(s_t, a_t) + \gamma \max_a E[Q^*(s_{t+1}, a_{t+1})] \\ V^*(s_t) &= \max_a R(s_t, a_t) + E[\gamma V^*(s_{t+1})] \\ V^*(s_t) &= \max_a Q^*(s_t, a_t) \end{aligned}$$

During training the relation between time steps and number of episodes changes as follows:

- At starting phase, it is non-linear as it is learning
- When near optimal Q is reached, the relation becomes linear

NOTE: RL implicitly learn transition dynamics.

3 Off policy control

On policy control- learn policy by using the policy.

Off policy control- change target to optimal Q function. **NOTE:** Has less stronger convergence constraint — Just requires longer history of observations.

Q - learning: Change the update rule of SARSA as follows:

$$Q(s, a) = Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

3.1 Markov Game - By Littman et. al

$\langle S, A_r, A_o, T, R \rangle$

$T : S \times A_2 \times A_0 \rightarrow \pi(S)$

$R : S \times A_2 \times A_0 \rightarrow \mathfrak{R}$

$$\begin{aligned} V^*(s) &= \max_a Q(s, a_2) \\ &= \max_{\pi_2} \min_{a_0} \sum_{a_2} R(s, a_2, a_0) \pi(s, a_2) \end{aligned}$$

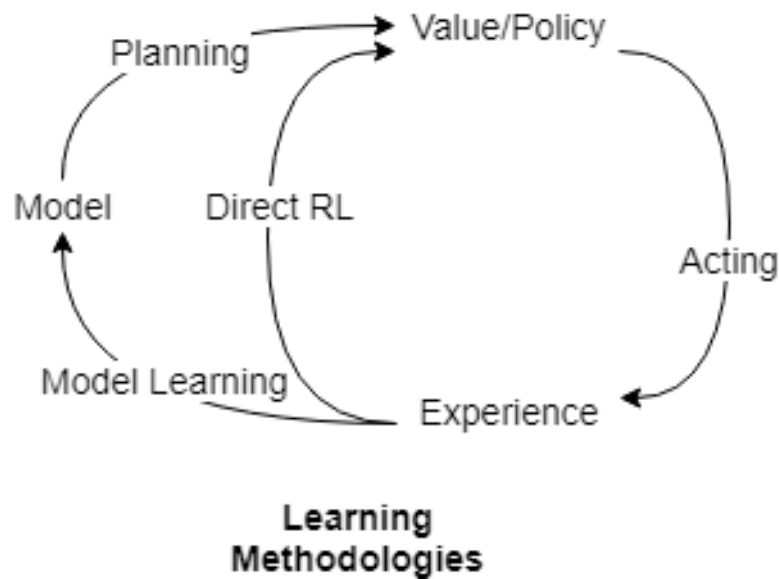
This becomes a linear optimization problem:

$$\begin{aligned} V &\leq \sum_{a_2 \in A_2} \pi(s, a_2) R(s, a_2, a_0), \quad \forall a_0 \in A_0 \\ \sum_{a_2} \pi(s, a_2) &= 1 \\ \pi(s, a_2) &> 0, \quad \forall a_2 \in A_2 \end{aligned}$$

Hence after getting V from solving above,

$$Q(s, a_2, a_0) = Q(s, a_2, a_0) + \alpha [R(s, a_2, a_0) + \gamma V(s') - Q(s, a_2, a_0)]$$

This is **min-max Q-Learning**.



The above figure showcases different learning methodologies. Some methods can be a mix of different learning methods.

4 Dyna-Q

This following algorithm is a mix of RL, Model and Planning methods:

```

Choose action a from Q
 $Q(s, a) = Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
Model (S, A)  $\leftarrow$  R, S'
Loop for N iterations
  S  $\leftarrow$  Sample from history
  A  $\leftarrow$  Sample from history
  R, S'  $\leftarrow$  Sample from Model(S, A)
  Perform update on Q based on R, S'

```

+ Advantage of this method: Much faster learning and convergence.

NOTE: All these learning methods introduced till now are not widely used but approximation is used.

5 Value Approximation

Let $w \in \mathbb{R}^d$,

$$\begin{aligned} L &= \sum_{s \in S} (\text{target value of } S - \text{current value of } S) \\ &= \sum_{s \in S} (V^\pi(s) - \hat{V}(w, s))^2 \end{aligned}$$

Hence, the update process will be:

$$\begin{aligned} w_{t+1} &= w_t - \frac{1}{2} \alpha \nabla_w [V^\pi(s) - \hat{V}(w_t, s_t)]^2 \\ &= w_t + \alpha (V^\pi(s) - \hat{V}(w_t, s_t)) \nabla_w \hat{V}(w_t, s_t) \\ &= w_t + \alpha (R + \gamma \hat{V}(w_t, s_{t+1}) - \hat{V}(w_t, s_t)) \nabla_w \hat{V}(w_t, s_t) \end{aligned}$$

$$\text{Error} = R + \gamma \hat{V}(w_t, s_{t+1}) - \hat{V}(w_t, s_t)$$

5.1 Car Example

Problem formulation:

$$\begin{aligned} f_1 &: \text{speed} \in \{0, 1\} \\ f_2 &: \text{is there other car in front} \in \{0, 1\} \\ \hat{V} &= w \cdot f \text{ (Note: Can be any complex models too.)} \\ \pi &: \text{Move forward with velocity 1} \\ R &= 1, \text{ only when we are driving} \end{aligned}$$

Four interactions:

1. $f_1(s_1) = 1, f_2(s_1) = 0$
2. $f_1(s_2) = 1, f_2(s_2) = 0$
3. $f_1(s_3) = 1, f_2(s_3) = 0$
4. $f_1(s_4) = 1, f_2(s_4) = 1$

Initial: $\bar{w} = 0, \alpha = 0.5, \gamma = 0.9$

1:

$$\begin{aligned} w_1 &= 0 + 0.5(1 + 0.9 * 0 - 0) * 1 = 0.5 \\ w_2 &= 0 + 0.5(1 + 0.9 * 0 - 0) * 0 = 0 \end{aligned}$$

2:

$$\hat{V}(s) = 0.5 * 1 + 0 = 0.5$$

Keep moving, $\hat{V}(s') = 0.5 * 1 + 0 = 0.5$

$$\delta = 1 + 0.9 * 0.5 - 0.5 = 0.95$$

$$w_1 = 0.5 + 0.5 * 0.95 = 0.975$$

$$w_2 = 0 + 0.95 * 0 = 0$$

3:

Keep moving...

$$w_1 = 1.92$$

$$w_2 = 0$$

4:

$$f_1(s) = 0, f_2(s) = 0$$

$$f_1(s') = 1, f_2(s') = 1$$

$$\hat{V}(s) = 1.42$$

$$\hat{V}(s') = 0$$

$$\delta = -1 + 0.9 * 0 - 1.42 = -2.42$$

$$w_1 = 1.42 + 0.5 * \delta * 1 = 0.21$$

$$w_2 = 0 + 0.5 * \delta * 1 = -1.21$$

NOTE: This generalizes to continuous states too. **NOTE:** Deep learning can be used to design model as well as policy for learning.