Q-Learning

Farnaz Adib Yaghmaie

Linkoping University, *Sweden*
farnaz.adib.yaghmaie@liu.se

April 6, 2021

What is Q-learning

The most popular *Dynamic Programming* approach to solve an RL problem

- Is based on Bellman principle's of optimality
- Relies on definition of *Quality function* (also called *state-action value function*)
- In Q-learning, we learn the Q function

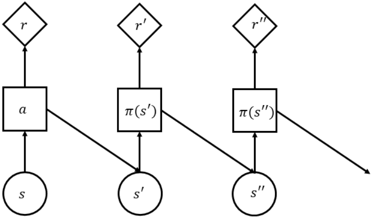
Three main components of an RL agent

- ~~Policy: The agent's decision~~
- Value function: how good the agent does in a state
- ~~Model: The agent's interpretation of the environment~~

Use Bellman's principle of optimality and

- estimate/evaluate the Quality function $Q(s, a)$ for all s, a
- choose a that has the best Quality in s.

Q function or state-action value function: The expected total reward starting from state s, taking an arbitrary action a and then following the policy π.



$$Q(s, a) = r(s, a) + \gamma\, \mathbf{E}[Q(s', \pi(s'))]$$

The action maximizes the expected total reward starting in s

$$\pi = \arg\max_a Q(s, a).$$

Q **function:** The expected total reward starting from state s, taking an arbitrary action a and then following the policy π.

$$Q(s,a) = r(s,a) + \gamma\, \mathbf{E}[Q(s',\pi(s'))] \tag{1}$$

Already in Bellman form!

Policy: The action maximizes the expected reward starting in s

$$\pi = \arg\max_a Q(s,a). \tag{2}$$

Be careful!

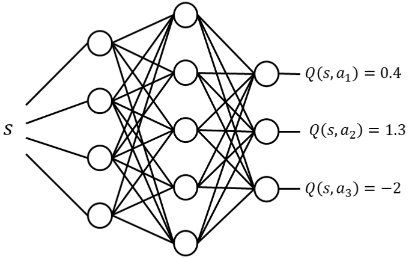
You need to solve an optimization problem!

$$\pi = \arg\max_a Q(s, a).$$

For discrete and continuous action space, the structure of $Q(s, a)$ should be selected carefully to avoid advanced optimization techniques.

Defining Q function in discrete case

- The function takes s as the input and generates $Q(s, a)$ for all possible actions.
- By feeding s the Q function is determined for all possible actions
- The actions are the indices for the vector.
- Policy is the index in which $Q(s, a)$ is maximized.



Defining Q function in continuous action space case

- The Q function takes state and action as inputs and generates a scalar output
- The policy is obtained by mathematical optimization
- Example: Quadratic Q

$$Q(s,a) = \begin{bmatrix} s^\dagger & a^\dagger \end{bmatrix} \begin{bmatrix} g_{ss} & g_{sa} \\ g_{sa}^\dagger & g_{aa} \end{bmatrix} \begin{bmatrix} s \\ a \end{bmatrix} \quad (3)$$

The policy is

$$\pi = -g_{aa}^{-1} g_{sa}^\dagger \, s. \quad (4)$$

Discrete:

- Feed s and generate $Q(s, a)$ for **all** actions

- Policy: by indexing

- Arbitrary structure

Continuous:

- Feed s and a and generate $Q(s, a)$ for that **specific** (s, a)

- Policy: by analytical optimization

- A structure to be optimized analytically e.g. quadratic

Our guess of Q function does not satisfy Bellman and there is an error

$$e = r(s, a) + \gamma \, Q(s', \pi(s')) - Q(s, a). \tag{5}$$

Temporal Difference (TD) learning:

Minimize the mean square error $\frac{1}{2} \sum_{t=1}^{T} e_t^2$.

How to build this error

$$e = r(s,a) + \gamma\, Q(s', \pi(s')) - Q(s,a).$$

For each sample point $s_t,\ a_t,\ r_t,\ s_{t+1}$, do the following

- Find $Q(s_t, a_t)$
- Find $Q_{target}(r_t, s_{t+1}) = r_t + \gamma\ \arg_a \max Q(s_{t+1}, a)$
- Define the error $e_t = Q_{target}(r_t, s_{t+1}) - Q(s_t, a_t)$.
- Minimize the mean square error $\frac{1}{2} \sum_{t=1}^{T} e_t^2$.

- Define a network Q to take s and generate $Q(s, a)$ for all possible a
- Assign a mean square error loss function for it

- Consider a quadratic Q function in s, a:

$$Q(s,a) = \begin{bmatrix} s^\dagger & a^\dagger \end{bmatrix} \begin{bmatrix} g_{ss} & g_{sa} \\ g_{sa}^\dagger & g_{aa} \end{bmatrix} \begin{bmatrix} s \\ a \end{bmatrix} = z^\dagger G z$$

Minimize the mse by batch least squares

$$vecs(G) = \left(\frac{1}{T}\sum_{t=1}^{T} \Psi_t(\Psi_t - \gamma\Psi_{t+1})^\dagger\right)^{-1}\left(\frac{1}{T}\sum_{t=1}^{T} \Psi_t r_t\right), \qquad (6)$$

where

$$z = \begin{bmatrix} s \\ a \end{bmatrix}, \ \Psi = [z_1^2, 2z_1z_2, ..., 2z_1z_n, z_2^2, ..., \ 2z_2z_n, \ ..., z_n^2]^\dagger.$$

It is called Least Squares Temporal Difference Learning (LSTD).

Both minimize mse

Discrete:

Numerically by a Gradient algorithm

Pro: Can have arbitrary structure

Con: Hyper parameters should be set

Continuous:

Analytically by batch least squares

Con: Should be quadratic

Pro: No hyper parameter at all

How to select a in Q-learning?!??

Example: Eating in town

- **Exploitation:** Go to your favourite restaurant
- **Exploration:** Select a random restaurant

In RL

- **Exploitation only:** will get stuck in a local optimum forever
- **Exploration only:** will try only random things

It is important to balance Exploration *vs.* Exploitation

How to generate a in discrete action space case?

Set a level $0 < \epsilon < 1$ and generate a random number $r \sim [0,\ 1]$

$$a = \begin{cases} \text{random action} & \text{if } r \ < \ \epsilon, \\ \arg\max_a Q(s, a) & \text{Otherwise.} \end{cases}$$

How to generate a in continuous action space case?

Generate a random number $r \sim \mathcal{N}(0, \sigma^2)$

$$a = \arg\max_a Q(s, a) + r.$$

Putting all together

We build/select a network to represent $Q(s, a)$. Then, we iterate:

1. Collect data
   - Observe the state s and select the action a.
   - Apply a and observe r and the next state s'.
   - Add s, a, r, s' to the history.
2. Update the parameter θ
   - We minimize the mean squared error using the history of data.

Q-learning

- Model-free
- Based on Bellman's principle of optimality
- The first approach to try
- Usually good results
- Take a look at explanation and implementation on my Github,

Crash_course_on_RL/q_notebook.ipynb

Email your questions to

farnaz.adib.yaghmaie@liu.se