1. Reinforcement Learning

The mathmatic behind reinforcement learning is HARD!

1.1 Basic Model

- Reinforcement Learning is about learning **Policy** from the interaction between agent and environment
- A **Policy** function, written like $p = \pi(s, a)$, describes how the agent act. It reads as the probability of agent to take action a at state s.
- The interaction of agent and environment can be described as a sequence of
- $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, …$
- It read as the agent at **State** S_t made an **Action** A_t, the environments gave a instance feedback **Reward** R_{t+1} and subsequent **State** S_{t+1} based on **State** S_t and the **Action** A_t.
- Most of the time, we are more interested for maximizing cumulative future rewards, such as win a game at last. So we denote cumulative rewards at time step t as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, r \in [0, 1]$$

- G_t is also called **Return** at time step t
- γ is **Discount Rate**

1.2 State Value Function

- definition

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

- read: the expected cumulative return from state s following policy π
- consider all following rewards are known in a finite episode
- can be used to evaluate policy, e.g.

$$\pi' > \pi \iff v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}$$

- π' is better than π, when $v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}$

1.3 Action Value Function

- definition

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

- read: the cumulative return from state s when take action a and subsequently following policy π
- a bridge between state value, and action, policy

1.4 Bellman Equation

- Redefine state value function and action value function in an iterative way
- Bellman expectation equation

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(s_{t+1})|S_t = s]$$

- or use general symbol s, s' stands for two states. expend Bellman expectation equation, then we have

$$v_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, \pi(s)) v_\pi(s')$$

- or denote $\mathcal{P}^\pi_{ss'} = p(S_{t+1} = s'|S_t = s, A_t = \pi(s))$, which is more compact for matrix form

$$v_\pi(s) = R^\pi_s + \gamma \sum_{s,s' \in \mathcal{S}} \mathcal{P}^\pi_{ss'} v_\pi(s')$$

- Bellman optimal equation

$$v_*(s) = \max_{a \in \mathcal{A}} \{\, R^a_s + \gamma \sum_{s,s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_*(s')\}$$

2. Convergence for Value-Based Reinforcement Learning

2.1 Bellman Operator

- recall Bellman expectation equation

$$v_\pi(s) = R^\pi_s + \gamma \sum_{s,s' \in \mathcal{S}} \mathcal{P}^\pi_{ss'} v_\pi(s')$$

- for all states, we can rewrite above equation in matrix form like

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

- or more compactly

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

- define Bellman expectation operator $\mathcal{T}^\pi : \mathbb{R}^n \to \mathbb{R}^n$ as

$$\mathcal{T}^\pi v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

- similarly we have Bellman optimality operator

$$\mathcal{T}^* v = \max_{a \in \mathcal{A}} (\mathcal{R}^a + \gamma \mathcal{P}^a v)$$

2.2 Contraction Mappings

- Bellman operator is a contraction mapping. (a lot of read… if I ask why. and I asked)
- v_π and v_* are unique fixed points. By repeatedly applying $\mathcal{T}^\pi$ and $\mathcal{T}^*$ they will converge to respectively

$$\lim_{k \to \infty} (\mathcal{T}^\pi)^k v = v_\pi$$
$$\lim_{k \to \infty} (\mathcal{T}^*)^k v = v_*$$

2.3 Reference for this section

- What is the Bellman operator in reinforcement learning?
- How Does Value-Based Reinforcement Learning Find the Optimal Policy?

3. Q-Learning

- also called maximum SARS
- Asynchronous value iteration
- off-policy learning, use max(Q_next) to learn instead of using orignial policy π
- update Q-table q_π with $(s_t, a_t, r_{t+1}, s_{t+1})$,

$$q_\pi(s_t, a_t) = q_\pi(s_t, a_t) + \alpha(r_{t+1} + \gamma \max q_\pi(s_{t+1}) - q_\pi(s_t, a_t))$$

- or can be rewritten like soft-update form

$$q_\pi(s_t, a_t) = (1 - \alpha) q_\pi(s_t, a_t) + \alpha(r_{t+1} + \gamma \max q_\pi(s_{t+1}))$$

3.2 Temporal Difference Learning

- learning in time, not until episode end.
- I will finished this explaination later. I wish the explanaion intuitive but at the same time mathmatical reasonable.

4. Deep Q Network

- use neural network to fit high dimensional Q-table

4.1 Problem definition

- recall Q-learning

$$q_\pi(s_t, a_t) = (1-\alpha)q_\pi(s_t, a_t) + \alpha(r_{t+1} + \gamma \max q_\pi(s_{t+1}))$$

- considering both contraction mappings and temporal difference.
- let $q_\pi(s_t, a_t)$ be the temporal optimal state value
- let $r_{t+1} + \gamma * \max q_\pi(s_{t+1})$ be the iterative Bellman operator part
- we know that by repeatly applying $\mathcal{T}^*$, $\mathcal{T}^* v$ will converge to v_*
- So we have optimization problem

$$\min ||\mathcal{T}^* v - v_*||$$

- in the view of temporal difference, we redefine the optimization problem within a small interval, like

$$\min_{\text{policy}} ||\text{policynet} - \text{targetnet}||$$

- so we define two networks
- one policy network (also called as evaluation network)
- one target network
- we iterative update policy net to minimize $||\text{policynet} - \text{targetnet}||$ by training neural network.
- and periodically copy policy net to target net, due to temporal difference learning

4.2 Layers

- use dense layers to fit high dimensional Q-table
- in my experiments, I used

```
net = dense(state_size, 64)(x)
net = dense(64, 64)(net)
net = dense(64, action_size)(net)
```

5. Training Deep Q Network

- when hard copy from policy net to target net is too frequent, e.g. every 4 learning steps, the final performance will not be consistence due to my experiements. For example, one run will converge at early stage well, but some other runs will not converge.
- the agent will sometimes stuck, which can be imagined that there could be dead loop inside Q-table or network.

- the network capacity is important, e.g. layers and neurons. The network capacity can be interpreted as Q-table dimension. If the task is complex, then we need a deeper network to fit.
- the reward system or design is one of the most important part in RL.
- here is the evaluation runs from my model
- 100 episodes
- average score is 16
- max score is 25
- min score is 0
- replay the results

  ```
  $ python banana_navigation.py
  ```

6. future works

- Double DQN is designed to solve Q-value explode.
- Prioritized Experience Replay can improve performance of sparse problem.
-

References

- Udactiy Deep Reinforcement Learning Nanodegree
- Udacity Reinforcement Learning by Prof. Charles Isbell and Prof. Michael Littman
- Tutorials from MorvanZhou
- other blogs, articles, a lot