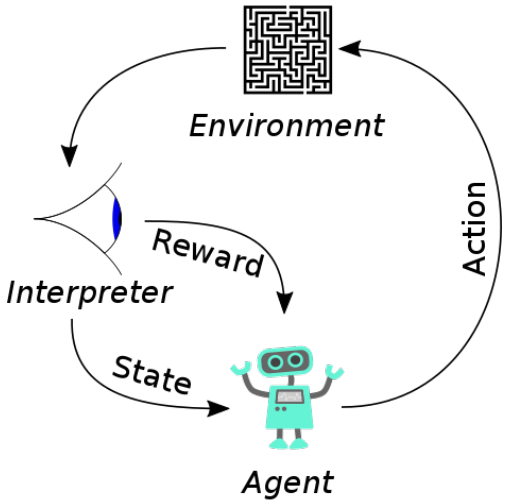
Reinforcement Learning (Part I)

Ce Sui

The concept of Reinforcement Learning is simple: "Learning to make good decisions under uncertainty": in this approach, an artificial agent learns to execute a task through trial and error, by interacting with its environment and receiving rewards or penalties based on its actions. These notes provide an introduction to the fundamental concepts and mathematical framework of Reinforcement Learning. Additionally, one of the most popular forms of deep Reinforcement Learning - the DQN method - is explained. These notes are largely based on the first six lectures of the CS234 course.



1. Basic Concepts
    - **Model:** (description of the world) Mathematical models of dynamics and reward, usually we assume a Markov Decision Process.
        - State: S is a (finite) set of **Markov states**

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t) \tag{2}$$

        - Action: A is a (finite) set of actions
        - Dynamics: P is dynamics/transition model for each action, that specifies

$$P(s_{t+1} = s'|s_t = s, a_t = a) \tag{3}$$

        - Reward: R is a reward function

$$R\left(s_t = s, a_t = a\right) = \mathbb{E}\left[r_t \mid s_t = s, a_t = a\right] \tag{4}$$

    - **Policy:** (our strategy)Function mapping states to actions

$$\pi(a|s) = P(a_t = a|s_t = s) \tag{5}$$

    - **Value function:**(evaluation of given strategy) future rewards from being in a state and/or action when following a particular policy

$$
\begin{aligned}
V(s) &= \mathbb{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots | \mathrm{s}_t = \mathrm{s}] \\
&= R^{\pi}(s) + \gamma \sum_{s' \in S} P^{\pi}\left(s' \mid s\right) V\left(s'\right) \\
&\text{where} \\
&R^{\pi}(s) = \sum_{a \in A} \pi(a \mid s) R(s, a) \\
&P^{\pi}\left(s' \mid s\right) = \sum_{a \in A} \pi(a \mid s) P\left(s' \mid s, a\right)
\end{aligned}
\tag{6}
$$

- Horizon: How many future steps should we consider(Number of time steps in each episode)
- Discount factor γ: trade-off between immediate reward and future reward
- Estimation: Iterative Algorithm

2. Policy Search

   The goal of policy search is to find the optimal strategy that can achieve highest value function, i.e.:

$$
\pi^*(s) = \arg\max_{\pi} V^{\pi}(s) \tag{7}
$$

   We first consider the simplest case where we know how the world works .

   - **MDP**(Markov Decision Process) **Policy Iteration**:
     - State-Action Value Q: Take action a, then follow the policy π

$$
Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P\left(s' \mid s, a\right) V^{\pi}\left(s'\right) \tag{8}
$$

     - Compute new policy:

$$
\pi_{i+1}(s) = \arg\max_{a} Q^{\pi_i}(s, a) \quad \forall s \in S \tag{9}
$$

   - **Value Iteration**:
     - Bellman backup operator: Applied to a value function and Returns a new value function

$$
BV(s) = \max_{a} \left[R(s, a) + \gamma \sum_{s' \in S} p\left(s' \mid s, a\right) V\left(s'\right) \right] \tag{10}
$$

       For a particular policy:

$$
\begin{aligned}
B^{\pi} V(s) &= R^{\pi}(s) + \gamma \sum_{s' \in S} P^{\pi}\left(s' \mid s\right) V\left(s'\right) \\
V^{\pi} &= B^{\pi} B^{\pi} \cdots B^{\pi} V
\end{aligned}
\tag{11}
$$

   - Value iteration to obtain value function and optimal policy:

     loop until convergence:

$$
V_{k+1} = B V_k \tag{12}
$$

     To extract optimal policy:

$$\pi(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} p\left(s' \mid s, a\right) V_{k+1}\left(s'\right) \right] \tag{13}$$

3. Model-Free Policy Evaluation
    - **Model-Free:** we do not know how the world works (do not have access to dynamics & reward models)
    - **MC Policy Evaluation**:
        - Target :

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\tau \sim \pi}\left[G_t \mid s_t = s\right] \\ G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots + \gamma^{T_i - 1} r_{T_i} \end{aligned} \tag{14}$$

        - Method: Monte Carlo Integration (Expectation over trajectories τ generated by following π)
        - Algorithm:

          Sample episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T_i}$

          Calculate $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots + \gamma^{T_i - 1} r_{T_i}$

          for $t = 1 : T_i$ where T_i is the length of the $i\,th$ episode:

$$V^\pi\left(s_{it}\right) = V^\pi\left(s_{it}\right) + \alpha\left(G_{i,t} - V^\pi\left(s_{it}\right)\right) \tag{15}$$

        - Converges to true value under some (generally mild) assumptions
    - **Temporal Difference Learning**

      (*"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference learning." – Sutton and Barto 2017*)
        - Algorithm:

          Sample tuple $\left(s_t, a_t, r_t, s_{t+1}\right)$

          Update $V^\pi\left(s_t\right) = V^\pi\left(s_t\right) + \alpha\left(\left[r_t + \gamma V^\pi\left(s_{t+1}\right)\right] - V^\pi\left(s_t\right)\right)$
        - Consistent estimator under some (generally mild) assumptions
    - **Certainty Equivalence with dynamic programming**
        - Algorithm:

          Sample tuple $\left(s_t, a_t, r_t, s_{t+1}\right)$

          Recompute maximum likelihood MDP model for (s, a)

$$\begin{aligned} \hat{P}\left(s' \mid s, a\right) &= \frac{1}{N(s,a)} \sum_{k=1}^{i} \mathbb{1}\left(s_k = s, a_k = a, s_{k+1} = s'\right) \\ \hat{r}(s,a) &= \frac{1}{N(s,a)} \sum_{k=1}^{i} \mathbb{1}\left(s_k = s, a_k = a\right) r_k \end{aligned} \tag{16}$$

          Compute V^π
        - Very data efficient and very computationally expensive
        - Consistent
4. Model-free Control

- **Goal:** Learn to select actions to maximize total expected future reward
- **ϵ-greedy Policies:**
  - basic idea:

$$\begin{aligned} a = argmax_a Q(s,a), \quad p = 1-\epsilon+\epsilon/|A| \\ a = \text{other action}, \quad p = \epsilon/|A| \end{aligned} \tag{17}$$

- **Monte Carlo Online Control:**

  Sample episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T_i}$

  Calculate $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots + \gamma^{T_i-1} r_{T_i}$

  for $t = 1 : T_i$ where T_i is the length of the $i\ th$ episode:

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha\left(G_{i,t} - Q^{\pi}(s_t, a_t)\right) \tag{18}$$

  $\pi_k = \epsilon - greedy(Q)$

  - converges to the optimal state-action value function when it is a GLIE MC control
- **Sarsa: On-policy TD Control (Temopral difference learning)**
  - Algorithm:

    Sample tuple (s_t, a_t, r_t, s_{t+1})

    Loop

    Take action $a_{t+1} \sim \pi(s_{t+1})$

    Observe (r_{t+1}, s_{t+2})

    Update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left([r_t + \gamma Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)\right)$

    $\pi(s_t) = argmax_a Q(s_t, a) \quad p = 1-\epsilon \quad \text{else random}$

    $t = t + 1$

    end loop
  - convergence under certain conditions
  - On-policy and off-policy:
    - on-policy: evaluate a policy from experience obtained from following that policy(see Sarsa)
    - off-policy: estimate and evaluate a policy using experience gathered from following a different policy(see Q-learning)
- **Q-learning**
  - Algorithm:

    Loop

    Take action $a_t \sim \pi(s_t)$

    Observe (r_t, s_{t+1})

    Update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left([r_t + \gamma\ max_a Q(s_{t+1}, a)] - Q(s_t, a_t)\right)$

    $\pi(s_t) = argmax_a Q(s_t, a) \quad p = 1-\epsilon \quad \text{else random}$

$t = t + 1$

end loop

- Convergence under certain condition

5. Function Approximations
   - **Basic Idea**: replace Tabular $V(s)$ and $Q(s,a)$ with function approximation
   - **Value Function Approximation**:
     - Monte Carlo VFA with Linear Model:

       Loop

       Sample episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T_i}$

       for $t = 1 : T_i$ where T_i is the length of the $i\ th$ episode:

       Calculate $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots + \gamma^{T_i - 1} r_{T_i}$

       Loss $(G_t - \hat{V}(s_t; w))^2 = (G_t - x(s_t)^T w)^2$

       Update (Linear model)

       $$\Delta w = \alpha(G_t - x(s_t)^T w) x(s_t)$$

       end loop

     - Temporal Difference Learning with Value Function Approximation

       Loop

       Sample tuple (s_t, a_t, r_t, s_{t+1})

       Loss $= \left(\left[r_t + \gamma \hat{V}^{\pi}(s_{t+1}) \right] - \hat{V}^{\pi}(s_t) \right)^2$

       Update (Linear model)

       $$\begin{aligned} \Delta w &= \alpha(\left[r_t + \gamma \hat{V}^{\pi}(s_{t+1}) \right] - \hat{V}^{\pi}(s_t)) \nabla_w \hat{V}^{\pi}(s_t; w) \\ &= \alpha(r + \gamma x(s_{t+1})^T w - x(s_t)^T w) x(s_t) \end{aligned}$$

       end loop

   - **Control using Value Function Approximation**

     MC control:

     $$\Delta w = \alpha \left(G_t - \hat{Q}(s_t, a_t; w) \right) \nabla_w \hat{Q}(s_t, a_t; w)$$

     SARSA:

     $$\Delta \boldsymbol{w} = \alpha \left(r + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \boldsymbol{w}) - \hat{Q}(s_t, a_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

     Q-learning:

     $$\Delta \boldsymbol{w} = \alpha \left(r + \gamma \max_a \hat{Q}(s_{t+1}, a; \boldsymbol{w}) - \hat{Q}(s_t, a_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

     - We can use Neural Networks! (This is the basic idea behind the most popular RL approach : Deep Q learning, DQN)
     - Q-learning may diverge : (1) Correlations between samples (2) Non-stationary targets
     - Possible solution:(1) Experience replay (2) Fixed Q-targets

- DQN algorithm(simple version):

  Loop

  At s_t, sample action a_t using $\epsilon - greedy$ policy for current $\hat{Q}\left(s_t, a; \boldsymbol{w}\right)$

  Store transition: $\left(s_t, a_t, r_t, s_{t+1}\right)$

  evaluate : $y_t = r + \gamma \max_a \hat{Q}\left(s_{t+1}, a; \boldsymbol{w}\right)$

  Calculate loss: $\left(y_t - \hat{Q}(s_t, a_t; w)\right)^2$ and do gradient descend
  $\Delta \boldsymbol{w} = \alpha \left(r + \gamma \max_a \hat{Q}\left(s_{t+1}, a; \boldsymbol{w}\right) - \hat{Q}\left(s_t, a_t; \boldsymbol{w}\right)\right) \nabla_{\boldsymbol{w}} \hat{Q}\left(s_t, a_t; \boldsymbol{w}\right)$

  End Loop

- There are many variants of DQN which improve certain aspects of this method: Double-DQN, Dueling-DQN, Distributional DQN...

6. Summary:
   - What you should learn from these notes:
     - Basic concepts of RL(reward, Policy, value function, state-action value)
     - Temporal difference learning, $\epsilon - greedy$ policy, Q-learing
     - Deep-Q learning
   - What's next:
     - DQN is based on greedy algorithm, so it has
       - deterministic behavior
       - can not be applied to continuous action space
     - In part II, we may discuss(cs234, lecture 7-15):
       - Policy-Based Reinforcement Learning
       - Batch Reinforcement Learning
       - ...
   - Resources and materials:
     - CS234
     - Reinforcement Learning: an introduction
     - Reinforcement Learning Learning Tool
     - Packages and codes:
       - Gymnasium
       - CleanRL
       - Unity ML-Agents Toolkit