NIA - Reinforcement Learning

Notes taken from Professor Martin Pilat and personal research.

The goal of reinforcement learning is to learn an agent's behavior that maximazes the total reward achievable from the environment it acts on. We assume that the agent can observe and influence the environment's states through its actions. At iteration t, the agent observes the current state s_t, selects an action a_t, executes it, and receives a new state s_{t+1} along with a reward r_t. This continues until a terminal state is reached or the episode ends.

Initially, the agent knows little or nothing about the environment. It learns through trial and error, balancing **exploration** (trying new actions) and **exploitation** (using known high-reward actions) to improve its long-term performance.

We can formally describe the environment as a MDP which is specified by the quadruple (S, A, P, R), where:

- S is a finite set of states of the environment.
- A is a finite set of actions.
- $P(s'|s, a)$ is the transition probability function that indicates the probability that by applying action a in state s, the environment will transition to state *s'*. It should fulfill the **Markov condition**, which means that the function should depend only on the current state and action and not on previous states or actions.
- $R_a(s, s')$ is the reward function that gives the expected inmediate reward that the agent receives from the environment if, in state s, it performs action a and transition to state *s'*.

A policy π defines the agent's behavior:

- Deterministic: $\pi : S \to A$ (a single action per state).
- Stochastic: $\pi(a|s)$ gives the probability of selecting action a in state s. Stochastic policies select the greedy action this way: $\pi(s_t) = argmax_a \pi(a_t|s_t)$.

Note: For convenience, we often write $\pi(s) = a$ even for stochastic policies when referring to the most probable (greedy) action.

We aim to find a policy π that maximizes the **expected total discounted reward**:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \right]$$

where $0 < \gamma \leq 1$ is the discount factor which ensures that the sum is finite.

Note: We can represent the policy as a matrix $|S| \times |A|$ where an entry (i, j) represents $\pi(s_i, a_j)$, and the transition probability for each action a_k as a matrix $|S| \times |S|$ where an entry (i, j) represents $P(s_j|s_i, a_k)$.

State-Value Function:

$V^\pi(s)$ under a policy π gives the expected total discounted reward when starting in state s and following π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s \right]$$

This can be recursively defined using the **Bellman expectation equation**:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a) \left[R_a(s,s') + \gamma V^\pi(s') \right]$$

Or, using expectation notation:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} \left[R(s,a) + \gamma \, \mathbb{E}_{s' \sim P(s'|s,a)} \left[V^\pi(s') \right] \right]$$

where $R(s,a) = \sum_{s'} P(s'|s,a) R_a(s,s')$

The idea to obtain the best strategy π is to update state values until reaching convergence and then update the policy as follows:

$$\pi^*(s) = argmax_a \sum_{s'} P(s'|s,a)[R_a(s,s') + \gamma V^{\pi^*}(s')]$$

It is used in model-based approaches (such as policy value iteration) but they need the transition probability function and the reward function. Often those cannot be provided so then we need action-values or also called state-action values.

Action-Value Function:

$Q^\pi(s,a)$ gives the expected total reward starting from state s, taking action a, and then following policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a \right]$$

This is also defined recursively via the Bellman expectation equation:

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a) \left[R_a(s,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right]$$

Or more compactly:

$$Q^\pi(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a) + \gamma\, \mathbb{E}_{a' \sim \pi(a'|s')} Q^\pi(s',a') \right]$$

The idea is that we compute state-action values until reaching convergence and updating the policy at each iteration or at the end as $\pi^*(s) = argmax_a Q^{\pi^*}(s,a)$. Model free methods such as Temporal Difference (bootstrapped, no need of a complete episode) or Monte Carlo (not-bootstrapped, complete episode needed) use it.

The basic idea of MC methods is to learn the optimal policy by averaging the results over multiple episodes. An episode is a sequence of quatriples $(s_{initial}, a, r, s_{next})$ which ends when reaching maximum iterations or a goal. At each step of an episode there is some fixed reward +1 or -1 with the goal of maximizing it.

We need complete episodes, after obtaining one we go through it taking first time encounters of pairs $(s_{initial}, a)$ and then updating each state-action pair with the average results of $Q(s,a)$ over all episodes.

We use the equality $Q^{\pi^*}(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{N(s,a)} G_t^{(i)}$ where $G^{(i)}$ is the total reward starting from the first encounter of that state-action in the i-th episode, and $N(s,a)$ is the total amount of first times the pair (s,a) appeared in episodes. After each episode it can be updated as $Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)} \left(G^{(i)} - Q(s,a) \right)$.

Monte Carlo Control idea:

```
Initialize Q(s, a) arbitrarily for all s, a
Initialize N(s, a) = 0

For episode = 1 to total_episodes:
    Generate an episode: (S0, A0, S1, R0), (S1, A1, S2, R1), ..., (ST, -, -, -)
    G = 0
    For each (s, a) pair in the episode:
        If (s, a) is first visit in the episode:
            G = total reward from this time onward
            N(s, a) += 1
            Q(s, a) += (G - Q(s, a)) / N(s, a)
```

This strategy helps the agent to choose actions that yield to the best results. However, high variances of $Q(s,a)$ lead to slow convergence elevating the computational time. For each episode, we can updated a few times those "first time" found state-action pairs so this method becomes inefficient for large MDP.

Some problems related to MC methods can be solved with Temporal-Difference (TD) methods. They use the Bellman optimality equation $Q^{\pi^*}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a) + \gamma max_{a' \sim \pi(a'|s')} Q^{\pi_*}(s',a')]$ which means the optimal value of taking the action a in state s is the immediate reward plus the discounted value of the best next acton in the next state s'. Known for its ***off-policy*** property which means that even if the agent selects a random action in practice, it learns assuming the best possible next action.

Q-learning is a Temporal-Difference (TD), model-free algorithm that learns optimal action-values. Q is represented as a matrix, initially with zeros. At each step t, we observe a transition (s_t, a_t, r_t, s_{t+1}), and update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

TD-Error defined as " $r_t + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$ " indicates the how far-close is the estimated $Q^{\pi}(s,a)$ and it corrects it.

Biggest advantage of this method is the need of less episodes compared to MC since it uses each agent-enviroment interaction to update the Q matrix. Second advantage just as important as the first is "**Reward Back Propagation Chain**" showed below with $V^{\pi}(s)$ to keep the example simple:

Suppose you have episodes E_1, E_2, E_3 and in $E_1 = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow goal$ and reaching the goal gives a reward of 1, otherwise 0.

The update in E_1 will be the following:
$V(s_3) = V(s_3) + \alpha(1 + V(s_3))$ surely ≥ 0.

Then somewhere in episode E_2 we have:
$V(s_2) = V(s_2) + \alpha(\gamma V(s_3) - V(s_2))$

and later on at E_3 we have:
$V(s_1) = V(s_1) + \alpha(\gamma V(s_2) - V(s_1))$

The reward was back-propagated! If a state is in the path that leads to the goal then it will be rewarded.

SARSA algorithm is similar to Q-learning but the update of Q is performed by the rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})) - Q(s_t, a_t)]$$

where $\pi(s_{t+1}) = argmax_a Q(s_{t+1}, a)$. SARSA is called ***on-policy*** method since it follows the strategy of its policy.

The traditional implementation of Q-learning using a matrix has the problem that it only works in discrete spaces and with discrete actions. Q-learning with continuous states or actions can't use a matrix to store $Q(s,a)$.

Solutions include:

- **State discretization**: divide continuous space into bins (coarse, limited).
- **Function approximation**: use neural networks to approximate $Q(s, a)$.

Two key techniques to improve convergence and stability:

- **Experience Replay**: store transitions and train from a random batch to break correlation.
- **Target Network**: use a delayed copy of the Q-network to compute targets, improving stability.

These ideas form the basis for **Deep Q-Networks (DQN)**.

TODO: Due September Deep Q-Networks