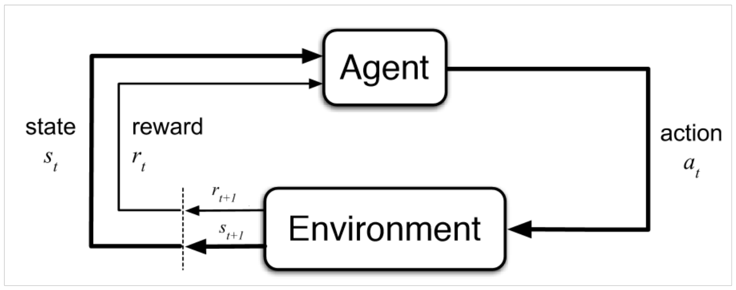
Reinforcement Learning

Jenny Li

May 2019

1 Introduction

Reinforcement learning is a type of machine learning in which an agent learns to perform specific actions in an environment that lead it to maximum reward. Because RL models learn by a continuous process of receiving rewards on every action, it can be trained to respond to unforeseen environments.



A RL setup consists of an agent (the RL algorithm) and an environment (the object the algorithm acts on). Basic reinforcement is modeled by a Markov decision process (MDP), which consists of states, actions, and rewards.

At each discrete time step, the agent receives a state from the environment. Based on the state, the agent chooses an action, and the environment subsequently transitions into a new state. The environment gives a reward to the agent associated with the transition.

1.1 Policy and Value Functions

The strategy that the agent employs to determine its next action is called policy. There are two types of policy functions:

1. Deterministic: always returns the same action at a given state.

$$a = \pi(s)$$

2. Stochastic: outputs a distribution probability over actions.

$$\pi(a|s) = \Pr(a_t = a|s_t = s)$$

The goal of the agent is to maximize its cumulative reward, which for each time step is expressed as a sum of future discounted rewards

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $\gamma \in [0, 1]$ is the discount-rate.

The state-value function tells us the maximum expected future reward the agent will get at each state. Starting with state s and following policy π, the function is

$$V_\pi(s) = E_\pi[R_t | s_t = s].$$

When V is maximized, the policy is most optimal.

It is also useful to define the action-value function, which returns the expected future reward of an action a at state s following policy π.

$$Q_\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a]$$

1.2 Monte Carlo vs. Temporal Difference

There are two different ways of learning. In the Monte Carlo approach, rewards are received when the agent reaches a "terminal state." The agent looks at the total cumulative reward and adjusts its estimates, restarting with new knowledge.

TD methods, however, adjust predictions about the future before the final outcome is known. The algorithm updates value estimates by approximating the rewards at each time step.

1.3 On-Policy vs. Off-Policy

An on-policy agent learns value estimates based on its current policy, while its off-policy counterpart learns them based on another policy. We will see this when examining Q-learning and SARSA.

1.4 Exploration and Exploitation

1. Exploration: finding more information about the environment.

2. Exploitation: using known information to maximize the reward.

The goal of our agent is to maximize the expected cumulative reward. However, without a proper balance between exploration and exploitation, the agent may fall into a trap where it repeatedly exploits rewards that are "closer," even if they are small. This means it will not uncover possible larger rewards.

To address this issue, we must define a rule that handles the exploration-exploitation trade off.

2 Value-Based RL

In value-based RL, the goal is to optimize $V_\pi(s)$ or equivalently, $Q_\pi(s,a)$, without implementing a policy.

2.1 Q-Learning

Q-learning is an off-policy RL algorithm that is used to optimize the action-value function. As the agent explores the environment, the algorithm iteratively updates $Q(s,a)$, giving us better and better approximations for the Q-values. Based on the well-known Bellman equation (which converges to the optimal value), the update function is

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

where α is the learning rate.

2.2 SARSA

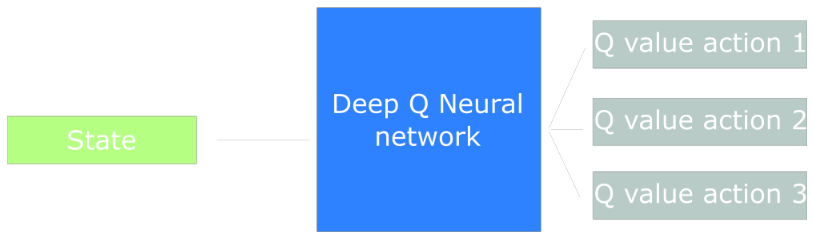
State–action–reward–state–action (SARSA) is similar to Q-learning, but it is an on-policy algorithm. This means that SARSA learns the Q-value based on the current policy instead of the greedy policy. The update equation is

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

where α is the learning rate.

2.3 Deep Q-Learning

With large state spaces, Q-learning or SARSA is impossible. In Deep Q-learning, a Deep Q neural network approximates, given a state, the different Q-values for each action.



The error is calculated by taking the difference between the maximum possible value from the next state and the current prediction of the Q-value.

$$E = (r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1},a,w)) - \hat{Q}(s_t,a_t,w)$$

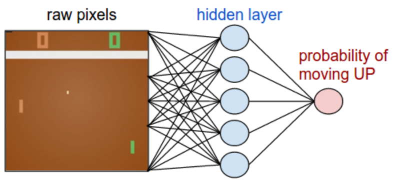
We use backpropagation to update the weights of the neural network so that the error is minimized.

$$\Delta w = \alpha[(r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a, w)) - \hat{Q}(s_t, a_t, w)]\nabla_w \hat{Q}(s_t, a_t, w)$$

3 Policy-Based RL

In policy-based RL, the goal is to directly learn the policy function that maps state to action instead of selecting actions based on a value function.

3.1 Policy Gradients



We have our stochastic policy π that has a parameter θ and outputs a probability distribution of actions.

$$\pi_\theta(a|s) = P[a|s]$$

We introduce a policy score function $J(\theta)$ that measures the expected reward of the policy. The policy score function we use depends on the task.

We want to maximize the score function using gradient ascent. The policy score function can be defined as

$$J(\theta) = E_\pi[R(\tau)]$$

where τ represents the sequence of states, actions, and rewards.

The gradient of the score function is

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau \pi(\tau;\theta)R(\tau).$$

Since we don't want to differentiate a probability function, we can change it into a logarithm.

$$\nabla \log x = \frac{\nabla x}{x}$$

$$\nabla_\theta J(\theta) = \sum_\tau \pi(\tau;\theta)\nabla_\theta(\log \pi(\tau|\theta))R(\tau)$$

Finally, we convert it back to an expectation and find our update function for our parameters.

$$\nabla_\theta J(\theta) = E_\pi[\nabla_\theta(\log \pi(\tau|\theta))R(\tau)]$$

$$\Delta\theta = \alpha * \nabla_\theta(\log \pi(\tau|\theta))R(\tau)$$

3.2 Advantages/Disadvantages over Value-Based

1. Policy-based methods have better convergence. During training, value-based methods can oscillate wildly because small changes in action values may dramatically alter the the choice of actions. Policy gradients always move smoothly toward a local maximum.

2. Policy gradients are more effective in high dimensional action spaces, or when using continuous actions. Deep Q-learning assigns a value to each action, which is unreasonable when dealing with continuous actions.

3. Policy gradients can learn a stochastic policy, while value functions can't. This easily handles the exploration-exploitation trade off.

4. Policy gradients have one major disadvantage. They often converge on a local maximum rather than on the global maximum.