Reinforcement Learning

> **Idea**
>
> An agent learns from the environment by interacting with it and receiving rewards for performing actions

> **Example**
>
> Backgammon - given sequences of moves and whether or not the player won at the end, learn to make good moves

An agent can take actions that affect the state of the environment and observe occasional rewards that depend on the state. Let's say a policy is a mapping from states to actions, then our goal is to learn a policy that maximises the expected reward over time.

The Markov Decision Process[1]

#MDP is a framework used to help make decisions on a stochastic environment. Our goal is to find a policy, which is a mapping that gives us all optimal actions on each state of our environment. In order to solve #MDPs we need dynamic programming[2], more specifically the Bellmann equation.

An #MDP is made up of 4 different components; the states s beginning with state s_0, the actions a, the reward function $r(s)$[3], and the transition model $P(s'|s,a)$[4]. The policy is represented as $\pi(s)$, which is the action the agent takes in any given state. Formally this can be written as (S, A, R, P, γ) where γ is the discount factor.

The #MDP Loop

At time $t = 0$, the environment samples the initial state $s_0 \sim p(s_0)$. Then, the following is repeated until "completion":

- The agent selects an action a_t
- The environment samples the reward $r_t \sim R(\cdot|s_t, a_t)$
- The environment samples the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$
- The agent then receives the reward r_t and the next state s_{t+1}

A policy $\pi(s)$ is a function from S to A that specifies what action to take in each state. Our objective is to find a policy π^* that maximises cumulative discounted reward.

Cumulative Discounted Reward

Suppose that the policy π starting in state s_0 leads to a sequence s_0, s_1, s_2 and so on. The cumulative reward of the sequence is $\sum_{t \geq 0} r(s_t)$.

Unfortunately state sequences can vary in length or even be infinite, so typically we define the cumulative reward as the sum of rewards discounted by a factor γ:
$r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) = \sum_{t \geq 0} \gamma^t r(s_t)$ for some γ such that $0 < \gamma \leq 1$.

The discount factor controls the importance of the future rewards versus the immediate ones. The lower the discount factor is, the less important future rewards are, and the agent will tend to focus on actions that yield immediate rewards only. To help the algorithm to converge, the cumulative reward is bounded.

#Reinforcement-learning vs #Supervised-learning

- Supervised Learning
  - Next input does not depend on previous inputs or agent predictions
  - There is a supervision signal at every step
  - Loss is differentiable w.r.t. model parameters
- Reinforcement Learning
  - Agent's actions affect the environment and help to determine the next observation
  - Rewards may be sparse
  - Rewards are not differentiable w.r.t. model parameters

#Supervised-learning Loop

- Get input x, sampled from an i.i.d[5] data distribution
- Use model with parameters w to predict output y
- Observe target output y_i and loss $l(w, x_i, y_i)$
- Update w to reduce loss with SGD[6] $w \leftarrow w - \eta \nabla l(w, x_i, y_i)$[7]

#Reinforcement-learning Loop

- From state s, take action a determined by policy $\pi(s)$
- Environment selects next state s' based on transition model $P(s'|s, a)$
- Observe s' and reward $r(s)$
- Update policy

The Two Main Approaches

#Value-based Methods

The goal of the agent is to optimise the value function $V(s)$. The value of each state is the total amount of the reward an RL agent can expect to collect in the future from the given state.

Value Function

The value function gives the total amount of reward the agent can expect from a particular state to all possible states from that state. With the value function we can find a policy. The value function V of a state s w.r.t. a policy π is the expected cumulative reward of following that policy starting in s: $V^{\pi}(s) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t) | s_0 = s, \pi]$

The optimal value of a state is the value achievable by following the best possible policy:
$V^{*}(s) = max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t) | s_0 = s, \pi]$

This can be used as a measure of how good a state is.

Q-Value Function

It is more convenient to define the value of a state-action pair:
$Q^{\pi}(s, a) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t) | s_0 = s, a_0 = a, \pi]$

The optimal Q-value function tells how good a state-action pair is:
$Q^{*}(s, a) = max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t r(s_t) | s_0 = s, a_0 = a, \pi]$

When the optimal Q-value is found it is used to compute the optimal policy:
$\pi^{*}(s) = argmax_a Q^{*}(s, a)$

Bellman Equation

Recursive relationship between optimal values of successive states and actions:

$$Q^{*}(s, a) = r(s) + \gamma \sum_{s^{*}} P(s^{*} | s, a) max_{a'} Q^{*}(s^{'}, a^{'})$$

$$= \mathbb{E}_{s' \sim P(\cdot | s, a)}[r(s) + \gamma \, max_{a'} Q^{*}(s^{'}, a^{'}) | s, a]$$

If the optimal state-action values for the next time-step $Q^{*}(s^{'}, a^{'})$ are known, then the optimal strategy is to take the action that maximises the expected value.

#Policy-based Method

We define a policy which we need to optimise directly and which defines how the agent behaves. A stochastic policy gives a distribution of probability over different actions: $\pi_{\theta}(s, a) \approx P(a | s)$

---

1. MDP ↩
2. DP ↩
3. which may be negative in some cases ↩

4. following the Markov assumption that the probability of going to s' from s depends only on s and not on any other past actions or states ↩
5. independent and identically distributed ↩
6. stochastic gradient descent ↩
7. unsure what the ' ∇ ' is meant to mean ↩