Reinforcement learning

Fraida Fund

Contents

Reinforcement learning

Elements of RL

- An *agent* acts in an *environment*
- The agent sees a sequence of *observations* about the environment
- The agent wants to achieve a *goal*, in spite of some *uncertainty* about the environment.

May need to consider indirect, delayed result of actions.

Elements of RL - environment

- The *state* of the agent at time t is S_t (from $s \in \mathcal{S}$)
- The agent chooses action A_t at time t (from $a \in \mathcal{A}$)
- The agent earns a reward R_t for its actions (possibly stochastic)
- The next state is determined by current state and current action, using a (possibly stochastic) state transition function $\delta(s, a)$:

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

The set of states $\mathcal{S}$, actions $\mathcal{A}$, the reward, and the state transition function, "live" outside the agent - part of the environment.

Elements of RL - observations

Over interactions in T time steps, the agent takes a sequence of actions and observes next states and rewards.

This sequence of interactions is called a *trajectory*:

$$S_1, A_1, R_2, S_2, A_2, \dots, S_T$$

Elements of RL - what agent may learn

- the *policy* π is the agent's mapping from state to action (or probabilities of action). We will always have a policy at the end, but it won't always be explicitly learned.
- We already said that the environment sends a *reward* back to the agent. The agent may learn a *value function* that describes expected total **future** reward from a state.
- The agent may have/learn a *model* of the environment, which we can use to **plan** before or during interactions with the environment

Taxonomy of RL agents

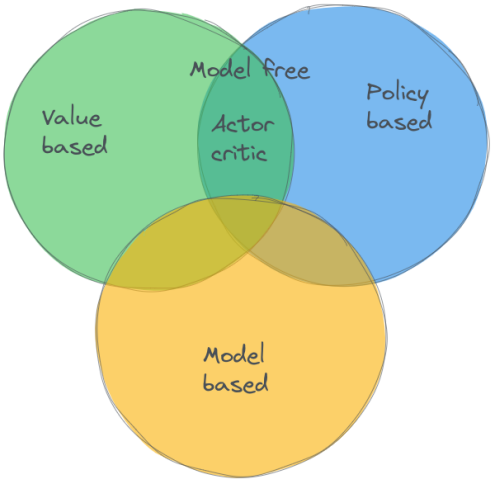


Figure 1: Taxonomy of RL agents.

- Policy-based: learn an explicit representation of policy $\pi : S \to A$.
- Value-based: try to learn what is the expected total reward for each state or state-action pair. We still end up with a policy, but it's not learned directly. For example, we might use a greedy policy: always pick the action that leads to the best expected reward, according to the value function that we learned.
- Actor-critic methods use both policy and value function learning.
- Model-based: uses either a known or learned model *of environment*.
- Model-free: does not know or try to explicitly learn a model *of environment*.
- (Model-free methods interact with the environment by trial-and-error, where model-based methods can plan for future situations by computation on the model.)

The optimization problem

Reward

Suppose the state transition function is

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

the reward for a state-action will be

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s, a)$$

The state transition function gives the probability of transitioning from state s to s' after taking action a, while obtaining reward r.

Policy

We want a *policy*, or a probability distribution over actions for a given state:

$$\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$$

Value function

Let future reward (**return**) from time t on be

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where the discount factor $0 < \gamma < 1$ penalizes future reward.

State-value

The state-value of a state s is the expected return if we are in the state at time t:

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

Action-value

The action value of a state-action pair is

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

Relationship between Q and V

For a policy π, we can sum the action values weighted by the probability of that action to get:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s, a)\pi(a|s)$$

Action advantage function

The difference between them is the action advantage:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

"Taking this action in this state" vs. "getting to this state."

Optimal value function

The optimal value function maximizes the return (future expected reward):

$$V_*(s) = \max_\pi V_\pi(s)$$
$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

Optimal policy

The optimal policy achieves the optimal value functions:

$$\pi_* = \arg\max_\pi V_\pi(s)$$
$$\pi_* = \arg\max_\pi Q_\pi(s,a)$$

i.e. $V_{\pi_*}(s) = V_*(s)$ and $Q_{\pi_*}(s,a) = Q_*(s,a)$.

Optimal policy breakdown

We can also think of it as the policy that maximizes current reward + discounted value of next state:

$$\pi_* = \arg\max_\pi r(s,a) + \gamma V^*_\pi(\delta(s,a))$$

From here on, we are going to assume a value based RL agent, and we will discuss strategies for learning the value function.

But first: what type of policy will the RL agent use?

Exploration and exploitation

If the policy is always to take the best action we know about, we might miss out on learning about other, better actions!

- **exploration**: take some action to find out about environment, even if you may miss out on some reward
- **exploitation**: take the action that maximizes reward, based on what you know about the environment.

ϵ-greedy policy

- With probability ϵ, choose random action
- With probability $1 - \epsilon$, choose optimal

Can decay ϵ over time.

There are many alternatives, e.g. an upper confidence bound policy, where we trade off actions that we know are optimal vs actions about whose effect we are less certain.

In either case: you would still use a greedy policy during inference! It's just during training that you would use a policy that includes both exploitation and exploration. (We call this "off-policy" when we use a different policy during training and inference.)

Now that we have a policy - we need to learn a value function. And of course, we want to learn from *experience*.

Monte Carlo

We could update the value function for a state after the end of a *complete* experience, e.g. after observing G_t.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

This is the Monte Carlo method.

The new value of state S_t is the previous estimate of the value of state S_t, plus learning rate times:

- G_t, the return after step t (observed)
- minus previous estimate of value of state S_t (estimated)

However, this only considers the return of an entire experience - does not consider which states/actions in the experience were useful, and which were not.

TD learning

Instead, we could update the value function after a single step, e.g. after observing R_{t+1} and S_{t+1} but no more.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

This is called temporal difference (TD) learning.

The new value of state S_t is the previous estimate of the value of state S_t, plus learning rate times:

- immediate reward (observed)
- plus discounted value of next state (next state is observed, its value is estimated)
- minus previous estimate of value of state S_t (estimated)

Here we have essentially broken down G_t into R_{t+1} (immediate reward) and $\gamma V(S_{t+1})$ (discounted future reward).

Q learning

As an example, we will look more closely at Q learning, which

- is value-based
- uses TD learning
- and learns the action-value function

Q table

- Each row/column is an action
- Each column/row is a state
- Table stores current estimate $\hat{Q}(s, a)$

Figure 2: Example agent, environment, and Q table.

Iterative approximation

- start with zero values
- observe state S_t, then iteratively:
  - choose action A_t and execute,
  - observe immediate reward R_{t+1} and new state S_{t+1}
  - update $\hat{Q}(S_t, A_t)$ using

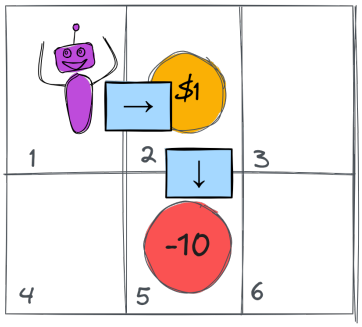
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- go to new state S_{t+1}.

New Q value estimate is the previous estimate, plus learning rate times:

- immediate reward (observed)
- plus discounted estimate of optimal Q value of next state (optimal, using greedy policy - not epsilon greedy!)
- minus previous estimate of Q value

Q table - after two steps



	←	→	↑	↓
1	0	0.1	0	0
2	0	0	0	-1
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

Figure 3: After two steps.

In practice, we want to learn environments with very large state space where we cannot enumerate a Q table like this. But, in place of a lookup table, we can learn underlying relationships using e.g. deep neural networks $\longrightarrow$ deep Q learning.