1 Subjects

- Markov Decision Process
- Bellman Equations and Algorithms

2 Notes

2.1 Elements of reinforcement learning

Exploration vs. exploitation Should we explore new opportunities, or take advantage of the ones we have found that works?

Agent How do we interact with the environment, who do we represent?

Environment What actions can we take and what is currently happening?

Policy How does the learning agent behave? A mapping from the perceived states of the environment to which actions to be taken when in those states.

Reward signal On each time step, the environment send the agent a single number, a reward. The agents sole objective is to maximize the total reward it receives. The agent can change the outcome by the signal by taking actions or changing the environment, but can't change the function that generates the reward signal.

Value function The value of a state is the total amount of reward an agent can expect to accumulate, *starting from that state*. Rewards return the immediate desirability of a state, value returns the long-term desirability of a state.

Model of the environment Something that mimics the behaviour of the environment, so we can infer how the environment will behave. (E.g. could be used to predict the next model). There are both model-based and model-free methods.

2.2 Finite Markov Decision Processes

We can think of the interaction between an agent and the environment as the agent taking some action at time t A_t, and the environment providing the agent with a reward R_{t+1} and a new state S_{t+1} which prompts a new action A_{t+1} from the agent.

At each time step t, the agent implements a mapping from states to probabilities of selecting each possible action $A_t \in A(S_t)$. This mapping is called the agent's policy and is denoted π_t where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Reinforcement learning is about changing the policy as a result of experience.

We seek to maximize the *expected return* $\mathbb{E}(G_t)$, but how do we define G_t? The simplest case is simply the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

If we are dealing with *episodic tasks*, i.e. tasks that are independent like plays of a game or trips through a maze (we call these tasks "episodes"). However, if we are dealing with *continuing tasks*, for example a robot with a long life span or process-control tasks then the return, which we are trying to maximize is potentially infinite. Thus, we will use a definition of return that is slightly more complex conceptually but much simpler mathematically. We introduce the concept of *discounting*, the agent will try to maximize the sum of the discounted rewards, where the rewards are decreasing in value so tasks that are performed immediately are worth more:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Here $0 \leq \gamma \leq 1$ is called the discount rate. Then we know that if $\gamma < 1$ then the infinite sum has a finite value, as long as $R_k \neq \infty$. If $\gamma = 0$, then the agent is only concerned with maximizing immediate rewards. More often than not, we won't have ∞ rewards, so we can simplify this to:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

Where it is possible that $T = \infty$ or $\gamma = 1$ but both cannot be true (since the sum would then be ∞).

2.2.1 The Markov Property

We cannot expect an actor to know everything about the environment, as some information may be hidden. The environment will provide the actor with different sensations, and we might expect the actor to remember all the sensations it has experienced so far.

We call a state signal which succeeds in retaining all relevant information to be *Markov* or to have *the Markov property*. For example, if we were playing chess, then the current configuration of all the pieces on the board would serve as a Markov state, because it summarizes all important details that led to it. Much of the information about the sequence is lost, but all that really matters for the future of the game is retained. It doesn't matter that we don't know how we ended up at that chess board configuration, the current configuration is all that is relevant.

In order to formally define the Markov property in a mathematically simple way, we will assume that there are a finit number of states and reward values. This enables us to work in terms of sums and probabilities instead of integrals and probability densities, but it can easily be extended.

Consider how an environment might respond at time $t+1$ to the action taken at time t. The most general, causal case is where the response depends on everything that has happened earlier:

$$\mathbb{P}\left[S_{t+1} = s', R_{t+1} = r | S_0, A_0, R_1, S_1, A_1, \ldots, R_{t-1}, S_{t-1}, A_{t-1}, R_t, S_t, A_t\right]$$

However, if the state signal has the Markov property, then the environments response at $t+1$ only depends on the state and action at t so we can define it as simply:

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

If this hold for all s' and r then we will be able to predict the result of an action having just s, a just as good as we would be able to do it given the complete history of states and actions.

It is usually useful to think of the state at each time step as an approximation to a Markov state even in non-Markov state signals because of the better performance Markov gives us in reinforcement learning, as long as one keeps in mind that it may not fully satisfy the Markov property.

2.2.2 Markov Decision Process

A reinforcement learning task that satisfies the Markov property is called a *Markov decision process* or MDP. If there is a finite amount of states and actions, then we call it a *finite MDP*. If you understand *finite MDP* then you understand 90% of modern reinforcement learning.

A finite MDP, is defined by uts state and actions sets and by the one-step dynamics of the environment. We use the probability of going to s' and receiving r from s and a as previously:

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

This completely specify the dynamics of some finite MDP. We can then compute anything else we might want to know about the environment such as the expected rewards for state-action pairs:

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

The state-transition probabilities:

$$p(s'|s, a) = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right] = \sum_{r \in R} p(s', r|s, a)$$

And the expected rewards for state–action–next-state:

$$r(s, a, s') = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in R} r \cdot p(s', r|s, a)}{p(s'|s, a)}$$

2.2.3 Value Functions

We will need to estimate value functions for almost all reinforcement learning algorithms. The value functions are functions of states that estimate *how good* it is to for the the agent to be in a given state (or rather, how good is it to perform some action in a given state). The notion of "how good" is defined in terms of what future rewards can we expect to receive.

A policy π is a mapping from each state $s \in S$ and action $a \in A(s)$ to the probability $\pi(a|s)$ of taking action a when in state s. Informally, the following equation is the expected return when starting in state s and following policy π for a MDP:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

Similarly, we can define the value of taking action a in state s under π as the expected return starting from s, taking the action a and then following π:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

We call q_π the action-value function for policy π. We can estimate v_π and q_π using the average of the actual returns, we will return to this later. For now, let's look at a fundamental property of v_π which is used throughout reinforcement learning. Namely, the property that they satisfy particular recursive relationships:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\
&= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_t = s \right] \\
&= \sum_a \pi\left(a|s\right) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_{t+1} = s' \right] \right] \\
&= \sum_a \pi\left(a|s\right) \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right]
\end{aligned}$$

We can read this equation as a sum over the triples a, s' and r. For each triple, we compute its probability $\pi\left(a|s\right) p(s', r|s, a)$ weight the value in the bracket (which is the reward plus the expected reward from the next state s'), which gives us the expected value. This equation is called the *Bellman equation for* v_π

2.2.4 Optimal value functions

In order to "solve" reinforcement learning, roughly means that we need to find a policy that achieves a lot of reward in the long run. For finite MDPs we can define it precisely as follows. A police π is said to be better than or equals to a policy π' if its expected return is better than or equal to that of π'. In other words $\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \forall s \in S$. There will always be at least one policy which is better than or equal to all other policy which is the optimal policy, we denote any of the optimal policies as π_*. We can define their state-value function as:

$$v_*(s) = \max_\pi v_\pi(s)$$

Which is the optimal state-value function. We can define the same for the optimal action-value function:

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

For the state-action pair (s,a), this function gives the expected return of taking action a in state s and then following an optimal policy. We can therefore write q_* in terms of v_* as follows:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right]$$

2.2.5 Dynamic programming

We can use dynamic programmin (DP) to compute the value functions explained earlier. Furthermore, we can easily obtain optimal policies once we have found the optimal value functions v_* or q_* which satisfy the Bellman equations:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_*(s')\right] \end{aligned}$$

or

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \Big| S_t = s, A_t = a\right] \\ &= \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma \max_{a'} q_*(s', a')\right] \end{aligned}$$

We will see that we obtain DP algorithms by turning these Bellman equations into assigments, that is update rules which improve approximations of the desired functions.

2.2.6 Policy Evaluation

First, we will consider computing the state-value function v_π. This is called *policy evaluation* recall that:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right] \\ &= \sum_a \pi\left(a|s\right) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right] \end{aligned}$$

Now consider a sequence of approximate value functions $v_0, v_1, \ldots$, the initial approximation v_0 is arbitrary except the terminal state has to be 0. We can then obtain the approximations by using the Bellman equation for v_π:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s \right] \\ &= \sum_a \pi\left(a|s\right) \sum_{s',r} p\left(s', r|s, a\right) \left[r + \gamma v_k(s') \right] \end{aligned}$$

This approximation does indeed converge to v_π as $k \to \infty$

2.2.7 Policy Improvement

Now that we are able to determine v_π, we are able to evaluate our policies, and thus we will be able to improve upon them. For some state s, we would like to know whether or not we should change the policy such that we deterministically choose an action $a \neq \pi(s)$. We know the quality of following the current policy from s ($v_\pi(s)$), so would it be better or worse to change to another policy? In order to answer this, we could select a in s and otherwise just follow the existing policy:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E} \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a \right] \\ &= \sum_{s',r} p\left(s', r|s, a\right) \left[r + \gamma v_\pi(s') \right] \end{aligned}$$

If this is greater than $v_\pi(s)$ then it is better to select a once in s, but then it would in fact be better to pick s every time:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \implies v_{\pi'}(s) \geq v_\pi(s)$$

We can then use this to define a new greedy policy π' given by:

$$\begin{aligned} \pi'(s) &= \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \mathbb{E} \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a \right] \\ &= \arg\max_a \sum_{s',r} p\left(s', r|s, a\right) \left[r + \gamma v_\pi(s') \right] \end{aligned}$$

I.e. the greedy policy looks one step ahead and picks the action that looks best in the short term. Now suppose π' is as good as, but not better than, the old policy π. The $v_\pi = v_{\pi'}$ and then it follows that:

$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_{\pi'}(s')\right] \end{aligned}$$

But this was the optimal Bellman equation from earlier, therfore $v_{\pi'}$ must be v_* and thus π' must be an optimal policy, thus this policy improvement algorithm must give us a strictly better policy until we hit the optimal one.

So far, we have only considered deterministic policies, that is where $\pi(s)$ always evaluate to the same action. However, all of the ideas so far, easily extend to stochastic policies ($\pi(a|s)$).

Now that we can both evaluate and improve our policies, we can just continue evaluating then improving and evaluating then improve and so on, monotonically improving untill we find a optimal solution.

2.2.8 Value iteration

This algorithm is fairly expensive, so we need to speed it up somehow. One way is implementation specific, where we simply make sure to pick up policy evaluation values from the last evaluation and not 0.

Value iteration provides us with a way to solve both policy evaluation and improvement in one step.

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_k(s')\right] \end{aligned}$$

Here, we find that v_k still converges to v_*.

2.2.9 Drawback of DP

The drawback of DP as described here, is that it involves operations over the entire state set of the MDP, so if there are many states (like backgammon) then it is incredibly expensive.

2.2.10 Generalized Policy Iteration

This is a simple overview of the generalized policy iteration:

- While improving repeat
- Run policy evaluation for some time on some states
- Run policy improvement for som time on some states

Here we may choose not to visit all states, but in most cases we will still converge in polynomial time.

2.3 Monte carlo algorithms