

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 finite 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:

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

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:

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

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 its 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:

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

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:

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

The state-transition probabilities:

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

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

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

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:

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

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 :

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

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:

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

We can read this equation as a sum over the triples a , s' and r . For each triple, we compute its probability $\pi(a|s)p(s', r|s, a)$ 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 policy 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:

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

2.2.5 Dynamic programming

We can use dynamic programming (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:

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

or

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

We will see that we obtain DP algorithms by turning these Bellman equations into assignments, 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:

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

Now consider a sequence of approximate value functions v_0, v_1, \dots , 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 :

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

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

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:

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

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:

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

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:

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

But this was the optimal Bellman equation from earlier, therefore $v_{\pi'}$ 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 until 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.

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

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

In generalized policy iteration (GPI), we maintain both an approximate policy and an approximate value function. The value function is repeatedly altered to approximate the value function for the current policy, and the current policy is repeatedly improved with respect to the current value function. In a way they move against each other, as they create a moving target for each other, but they cause both policy and value function to approach optimality.

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 some 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

Up until now, we have assumed complete knowledge of the environment (finite states and actions). Monte Carlo algorithms, allows us to learn from just *experience* – we sample sequences of states, action and rewards without any prior knowledge of the environment’s dynamics.

In the Monte Carlo methods, we want to solve the reinforcement learning problem based on averaging sample returns. Furthermore instead of computing the value functions (as we did in DP), we here want to learn the value functions instead.

What if we looked at just the state-value function, how would we try to estimate that? The obvious solution here, is to average all the returns observed after visits to that state. As we observe more and more returns, the average should converge to the expected value. This is the underlying idea of all Monte Carlo algorithms.

In order to ensure well-defined returns are available, we will only consider Monte Carlo for episodic tasks.

2.3.1 Estimating $v_\pi(s)$

$v_\pi(s)$ is the value of a state s under policy π , given a set of episodes obtained by following π and passing through s . Each occurrence of state s in an episode is called a *visit* to s . It may happen that we visit s multiple times in an episode, so we will look at the *first visit* to s . Let’s first look at a theoretical algorithm which computes the value of s in the first visit to s :

We could also have explored the *every-visit MC method*, which has some different properties and extends more naturally to function approximation, but First-visit MC has been more widely studied.

First-visit MC converges quadratically to $v_\pi(s)$ as the number of first-visits to s goes to infinity. An alternative, finite option, is to loop for some time or some number of episodes.

2.3.2 Estimation of action values

If a model is available, we can simply look ahead in the next possible states and pick the action that leads to the best state. If no model is available, then we

Algorithm 1 First-visit MC policy evaluation

```
// Initialize:
 $\pi \leftarrow$  policy to be evaluated
 $V \leftarrow$  an arbitrary state-value function
 $Returns(s) \leftarrow$  an empty list, for all  $s \in S$ 

loop forever
    Generate an episode using  $\pi$ 
    for all state  $s$  in the episode do
         $G \leftarrow$  return following the first occurrence of  $s$ 
        Append  $G$  to  $Returns(s)$ 
         $V(s) \leftarrow \text{average}(Returns(s))$ 
    end for
end loop
```

have to estimate the values of the action (e.g. speed of vehicle) we should pick in order to get to the best state and maximize reward.

In other words, one of the primary goals for Monte Carlo methods is to estimate q_* , which is the optimal value we get when we take action a from state s . In order to estimate this, we may run a First-visit MC method as before, just visiting state-action pairs instead of just states. Again, this will converge quadratically as before.

The complication is that many state-action pairs may never be visited. If π is a deterministic policy, then following π one will only observe returns from one action from each state (as it is deterministic). Therefore the Monte Carlo estimates of the other actions from that state will not improve with experience. This is the general problem of *maintaining exploration* i.e. we must assure that we are continually exploring other options in order to attempt to find a better one. One way to solve this, is to say that we start in a state-action pair, and every state-action pair has a nonzero probability of being selected as the start. We call this the assumption of *exploring starts*.

Exploring starts, is not really useful when we learn by actual interacting with an environment as we cannot simply start from any state we want. In this case, the most common alternative approach is to only considering stochastic policies with only nonzero probabilities.

2.3.3 Approximating optimal policies

Let us consider a Monte Carlo version of classical policy iteration. We can perform policy evaluation as described in the previous section (estimation of action values). If we assume that:

- We observe an infinite number of episodes.
- The episodes are generated with exploring starts.

Then the Monte Carlo methods will compute each q_{π_k} exactly for arbitrary π_k .

Now we just need to figure out how to perform policy-improvements. We can improve on our policy with a simple greedy policy with respect to the current value function. We can then compute for each $s \in S$ the policy π :

$$\pi(s) = \arg \max_a q(s, a)$$

We can then construct each π_{k+1} as the greedy policy with respect to q_{π_k} , and we get that the policy improvement theorem applies since:

$$q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}(s, \arg \max_a q_{\pi_k}(s, a))$$

This assures us that each π_{k+1} is uniformly better than or just as good as π_k . If it is just as good as π_k then they are both optimal policies.

2.3.4 Monte Carlo without infinite episodes

We made two unlikely assumptions in order to easily obtain this guarantee of convergence. One was that the episodes has exploring starts, the other was that we could run for infinitely many episodes. We will try to remove the latter assumption here and then return to exploring starts afterwards.

There are two primary ways to combat the infinite episodes assumption, one approach is to estimate some bounds on magnitude and probability of error and run until these bounds are sufficiently small. This will guarantee correct convergence up to some level of approximation, however it is likely to require far too many episodes to be useful in practice.

The second approach is to avoid trying to complete policy evaluation before returning for improvement and simply move the value function *toward* q_{π_k} . One extreme example of this was the value iteration, in which only one iteration of iterative policy evaluation is performed between improvements. Or the in-place version where we alternate between improvement and evaluation steps for single states.

For Monte Carlo policy evaluation it is natural to alternate between evaluation and improvement on an episode-by-episode basis. After each episode we evaluate the policy and then improve it at all the visited states. We can describe this algorithm as follows:

2.3.5 Monte Carlo without Exploring Starts

Now, how can we avoid the unlikely assumption of exploring starts? The only general way to ensure that all actions are selected infinitely often is for the agent to continue to select them. We have two approaches to ensuring that this happens, we call it *on-policy* methods and *off-policy* methods. On-policy attempts to evaluate or improve the policy used to make decision whereas off-policy methods evaluate or improve a different policy. The ES method above is an example of an on-policy method. We will explore on-policy methods more here, and not go into details with off-policy.

Algorithm 2 Monte Carlo ES (Exploring Starts)

```
// Initialize, for all  $s \in S, a \in A(s)$ :  
 $Q(s, a) \leftarrow$  arbitrary  
 $\pi(s) \leftarrow$  arbitrary  
 $Returns(s, a) \leftarrow$  empty list  
  
loop forever  
  Choose  $S_0 \in S$  and  $A_0 \in A(S_0)$  s.t. all pairs have probability  $> 0$   
  Generate an episode starting from  $S_0, A_0$  following  $\pi$   
  for all pairs  $s, a$  appearing in the episode do  
     $G \leftarrow$  return following the first occurrence of  $s, a$   
    Append  $G$  to  $Returns(s, a)$   
     $Q(s, a) \leftarrow \text{average}(Returns(s, a))$   
  end for  
  for all states  $s$  in the episode do  
     $\pi(s) \leftarrow \arg \max_a Q(s, a)$   
  end for  
end loop
```

Here we will look at ϵ -greedy policies, which with probability $1 - \epsilon$ will simply follow the policy π , but with probability ϵ it will select a uniformly random action. So each action has probability $\frac{\epsilon}{|A(s)|}$ of being picked, except the estimated optimal action which also has probability $1 - \epsilon$ of being picked, yielding the algorithm:

This holds under the policy improvement theorem, although I will not prove this here.

Algorithm 3 On-policy first-visit MC control (for ϵ -soft policies)

```
// Initialize
for all  $s \in S$  and  $a \in A(s)$  do
     $Q(s, a) \leftarrow$  arbitrary
     $Returns(s, a) \leftarrow$  empty list
     $\pi(a|s) \leftarrow$  an arbitrary  $\epsilon$ -soft policy
end for
loop forever
    Generate an episode using  $\pi$ 
    for all pairs  $s, a$  appearing in the episode do
         $G \leftarrow$  return following the first occurrence of  $s, a$ 
        Append  $G$  to  $Returns(s, a)$ 
         $Q(s, a) \leftarrow \text{average}(Returns(s, a))$ 
    end for
    for all visited states  $s$  in the episode do
         $A^* \leftarrow \arg \max_a Q(s, a)$ 
        for all  $a \in A(s)$  do
             $\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|}, & \text{if } a = A^* \\ \frac{\epsilon}{|A(s)|}, & \text{if } a \neq A^* \end{cases}$ 
        end for
    end for
end loop
```
