Dynamic programming and Monte Carlo methods

Milica Gašić

Dialogue Systems Group, Cambridge University Engineering Department

In this lecture...

Introduction to model-based RL

Policy iteration

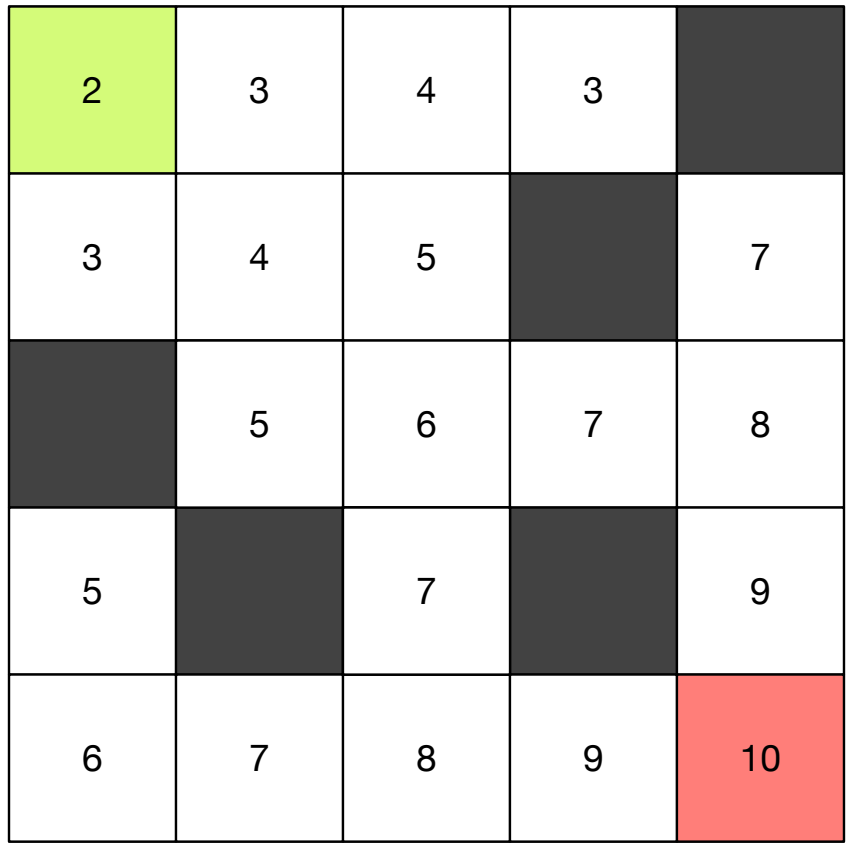
Value iteration

Introduction to model-free RL

Monte Carlo methods

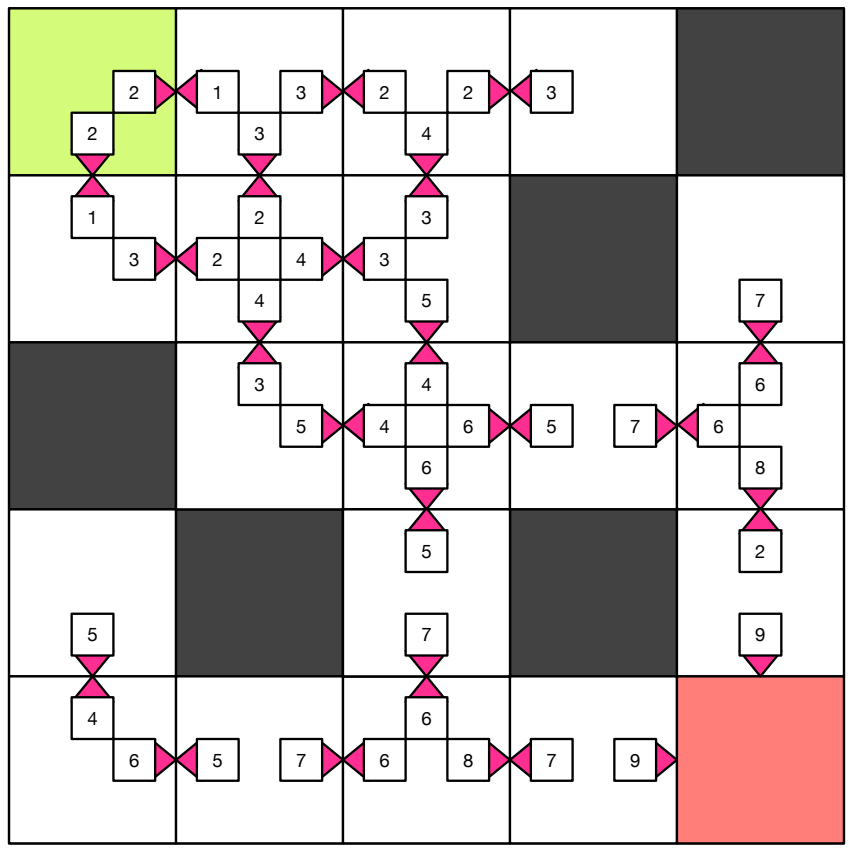
Reminder: Optimal value function

The optimal value function for each state gives highest the expected return that can be obtained from that state.



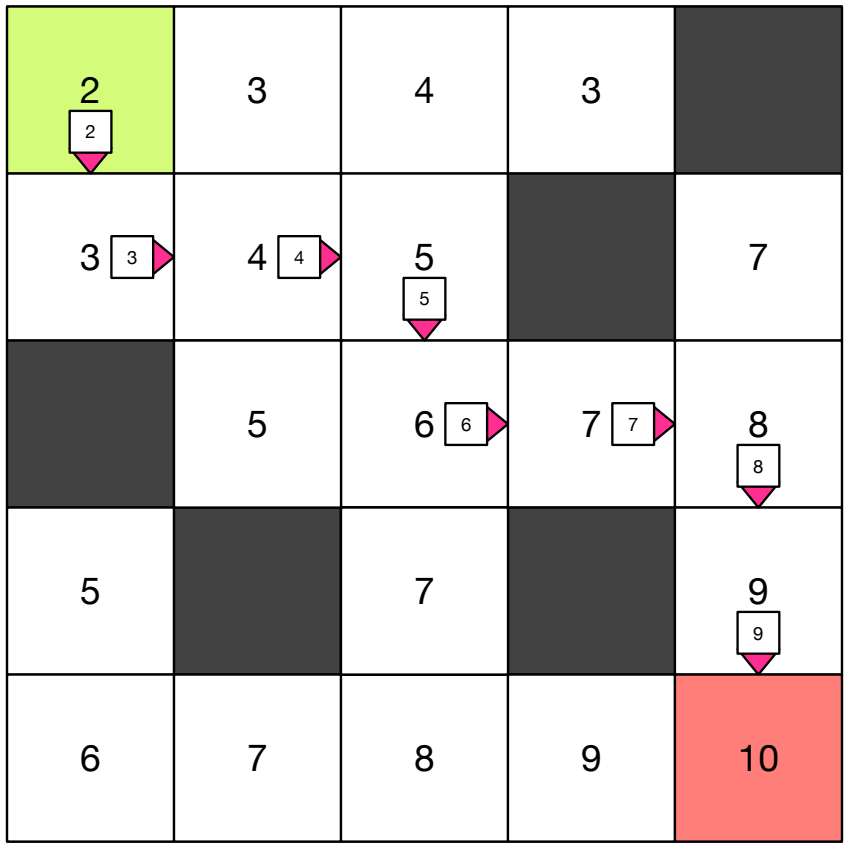
Reminder: Optimal Q-function

The optimal Q-function for each state and action gives the highest expected return that can be obtained from that state when that action is taken.



Reminder: Optimal policy

The optimal policy is the policy associated with the optimal value function or the optimal Q-function.



Dynamic programming

Dynamic programming (DP) algorithms can solve an MDP reinforcement learning task given the model of the environment (the state-transition probabilities and the reward function).

Finite-state MDP A common way of obtaining approximate solutions for tasks with continuous states and actions is to quantize the state and action spaces and then apply finite-state DP methods.

Key idea the use of value functions to organize and structure the search for good policies.

Policy evaluation

How to compute the state-value function for a given policy?

- Choose initial approximation arbitrary, eg $\forall s, V_0(s) = 0$
- Successive approximations are based on the Bellman equation

$$V_{k+1}(s) = \sum_a \pi(a, s) \sum_{s', r} p(s', r|s, a) \left(r + \gamma V_k(s')\right) \qquad (1)$$

- Stop when the value function stops changing
  $\max_s |V_{k+1}(s) - V_k(s)| < \theta$

Policy improvement

The reason for computing the value function for a given policy is to be able to find better policies.

Policy improvement theorem Let π and π' be any pair of deterministic policies such that, $\forall s \in S$, $Q_\pi(s, \pi'(s)) \geq V_\pi(s)$. Then the policy π' must be as good as, or better than, π.

Policy improvement: constructing a greedy policy π' which actions are better than the original policy π in short term

$$\pi'(s) = \arg\max_a \sum_{s,r} p(s', r|s, a) \left(r + V_\pi(s')\right) \tag{2}$$

If the new policy is not better than the old policy then they are optimal.

Policy iteration

Once a policy π has been improved using V_π to yield a better policy π' we can then compute $V_{\pi'}$ and improve it again to yield an even better π''.

Algorithm 1 Policy iteration

1: Initialise V and π arbitrarily
2: **repeat**
3: Evaluate V using π (Eq 1)
4: Improve π using V (Eq 2)
5: **until** convergence

Value iteration

One drawback of policy iteration is that it involves evaluating a policy at every step. Instead we can perform value iteration:

- Initialise values arbitrarily, eg $V_0(s) = 0$ for every s
- Successive approximations/improvements are based on

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma V_k(s')\right) \tag{3}$$

- Stop when the value function stops changing $\max_s |V_{k+1}(s) - V_k(s)| < \theta$

Value iteration

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.

Algorithm 2 Value iteration

1: Initialise V_0 arbitrarily
2: **repeat**
3: Improve V_{k+1} using the estimate of V_k (Eq 3)
4: **until** convergence

Asynchronous dynamic programming

Drawback of DP methods they involve operations over the entire state set of the MDP. If the state space is very large, this becomes computationally prohibitive.

Asynchronous algorithms back up the values of states in any order whatsoever, using whatever values of other states happen to be available. The values of some states may be backed up several times before the values of others are backed up once.

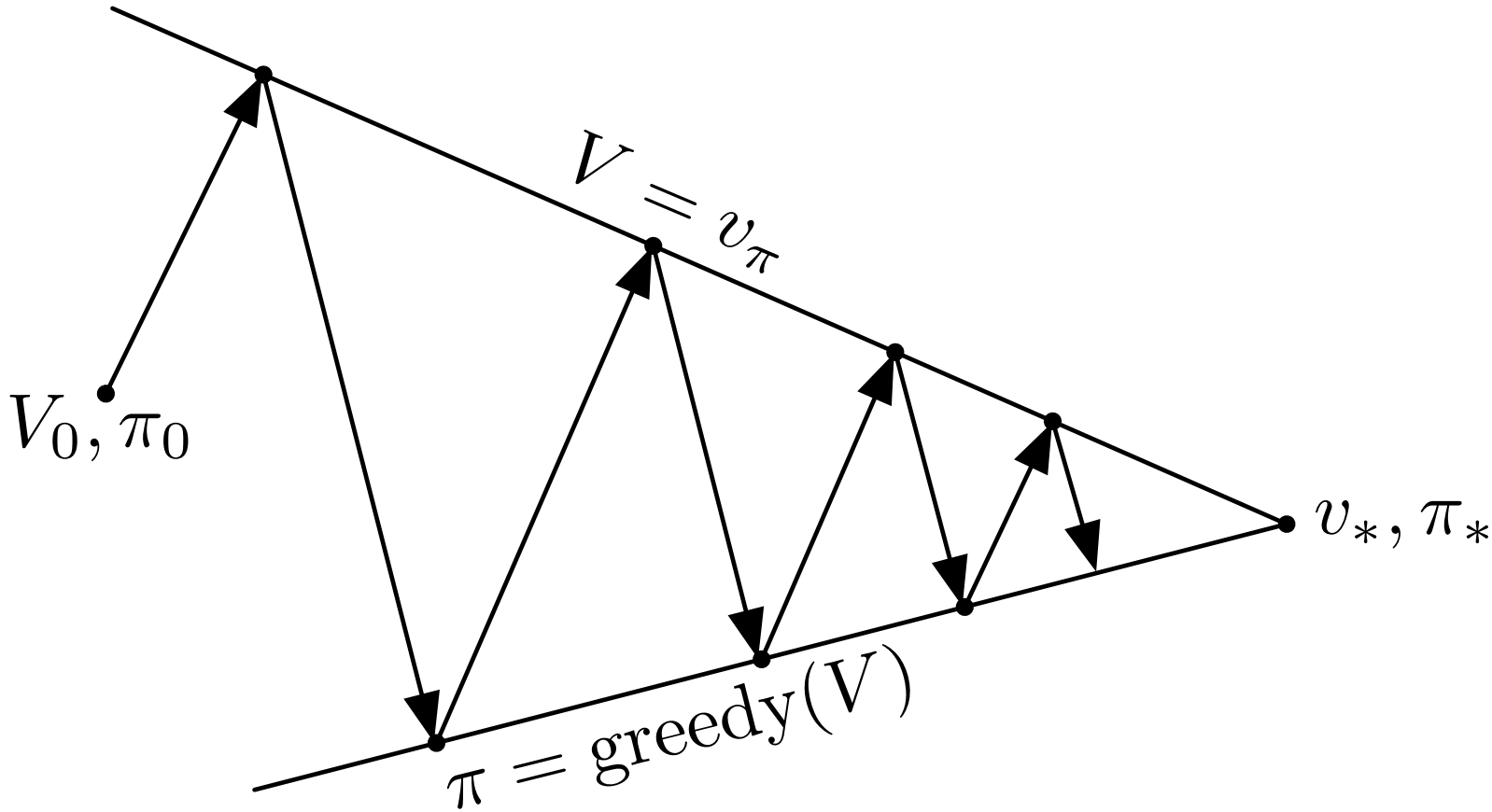
Generalised policy iteration

Policy evaluation and policy improvement processes interact, independent of the granularity of the two processes. The evaluation and improvement processes in GPI can be viewed

competing They compete in the sense that they pull in opposing directions. Making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy, and making the value function consistent with the policy typically causes that policy no longer to be greedy.

cooperating In the long run, however, these two processes interact to find a single joint solution

Generalised policy iteration



Efficiency of dynamic programming

- If n and k denote the number of states and actions, the total number of (deterministic) policies is k^n.
- A DP method takes a number of computational operations that is less than some polynomial function of n and k.
- DP has limited applicability because of the curse of dimensionality, the fact that the number of states often grows exponentially with the number of state variables.
- This is the inherent difficulty of the problem, not of DP as a solution method.

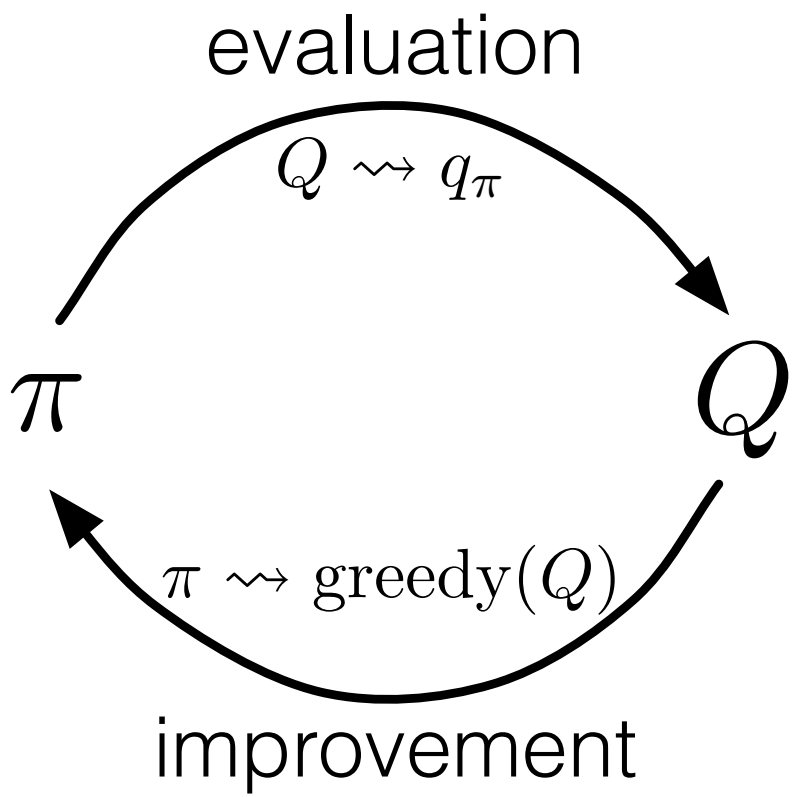
Summary

- Policy evaluation refers to the iterative computation of the value functions for a given policy.
- Policy improvement refers to the computation of an improved policy given the value function for that policy.
- Putting these two computations together, we obtain policy iteration and value iteration, the two most popular DP methods.
- Generalized policy iteration is the general idea of two interacting processes revolving around an approximate policy and an approximate value function.

Model-free reinforcement learning

- The complete model of the environment is not always available.
- In that case, the agent learns from *experience*.
- The experience can be obtained from **interaction** with *simulated* or *real* environment.
- Even-though the agent doesn't have the model of the environment, it can still find the optimal behaviour.

Monte Carlo methods



- Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns.
- Monte Carlo methods sample and average returns for each state-action pair and average rewards for each action.
- They are typically applied to episodic tasks.
- The policy is updated only at the end of an episode.

Monte Carlo prediction

Estimates value function for a given policy.

Algorithm 3 Monte Carlo prediction

1: Initialise V arbitrarily
2: $Returns(s) \leftarrow$ empty list $\forall s \in \mathcal{S}$
3: **repeat**
4: Generate an episode using π
5: **for** s in the episode **do**
6: $Returns(s) \leftarrow$ append return following s
7: **end for**
8: $V(s) = average(Returns(s))$
9: **until** convergence

Each average is an unbiased estimate, and the standard deviation of its error falls as $\frac{1}{\sqrt[2]{n}}$, where n is the number of returns averaged.

Monte Carlo estimation of the Q-function

state-action pairs are followed instead of states

maintains exploration all state-action pairs must be visited

On-policy and off-policy methods

In reinforcement learning we distinguish between

On-policy methods which attempt to evaluate or improve the policy that is used to make decisions

Off-policy methods which evaluate or improve a policy different from that used to generate the data.

On-policy Monte Carlo control

Algorithm 4 On-policy Monte Carlo control

Initialise Q and π arbitrarily
$Returns(s, a) \leftarrow$ empty list $\forall s \in \mathcal{S}, a \in \mathcal{A}$
repeat
  for $s \in \mathcal{S}$ and $a \in \mathcal{A}$ **do**
    Generate an episode using ϵ-greedy π starting with s, a
    for s, a in the episode **do**
      $Returns(s, a) \leftarrow$ append return following s, a
      $Q(s, a) = average(Returns(s, a))$
    end for
    for s in the episode **do**
      $\pi(s) = \arg\max_a Q(s, a)$
    end for
  end for
until convergence

Monte Carlo off-policy methods

In off-policy methods we have two policies

target policy the policy being learned

- The target policy is the greedy policy with respect to Q.

behaviour policy the policy that generates behaviour

- The behaviour policy must have a non-zero probability of selecting all actions that might be selected by the target policy (coverage).
- To insure this we require the behaviour policy to be soft (i.e., that it select all actions in all states with non-zero probability)
- The behaviour policy μ can be anything, but in order to assure convergence of π to the optimal policy, an infinite number of returns must be obtained for each pair of state and action.

Off-policy Monte Carlo control

Algorithm 5 Off-policy Monte Carlo control

Initialise Q arbitrarily, $C(s,a) = 0\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ $\pi \leftarrow$ greedy with respect to Q
repeat
  Generate an episode $[s_0, a_0, \ldots, a_{T-1}, s_T]$ using soft policy μ
  $R \leftarrow 0, W \leftarrow 1$
  for $t = T$ down-to 0 **do**
    $R \leftarrow \gamma R + r_{t+1}$
    $C(s_t, a_t) \leftarrow C(s_t, a_t) + W$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{W}{C(s_t, a_t)} (R - Q(s_t, a_t))$
    $\pi(s) = \arg\max_a Q(s,a)$
    if $a_t \neq \pi(s_t)$ **then**
      Exit for loop
    end if
    $W \leftarrow W \frac{1}{\mu(a_t, s_t)}$
  end for
until convergence

Summary

- **Model-based** vs **model-free** methods
- The Monte Carlo methods learn value functions and optimal policies from experience in the form of sample episodes.
- They do not require the model of the environment and can be learned directly in the interaction with the environment of in simulation.
- They simply average many returns for each state-action pair.
- **On-policy** vs **off-policy** methods.

.

Next lecture

- Temporal difference (TD) learning