Reinforcement Learning and Policy Mirror Descent an introduction

Feiyang Wu

College of Computing, School of Computer Science
Georgia Institute of Technology

August 12, 2022

Outline

Intro

Suppose you are a robot trying to explore the world. Your decision-making process is to optimize your gain from the exploration. Such a process can be modeled as a reinforcement learning problem.

The heart of the problem is to **evaluate** the current situation (”how good am doing right now”), and **improve** your plan in each situation (”what should be next best move”).

Suppose that somehow we come up with an algorithm that can solve all these, how many **sample** does it need to find a nearly optimal solution?

Mathematically, we use the following notations,

- $\mathcal{S}$: a set of possible states of the world, for example, a set of every position on a 2d grid.
- $\mathcal{A}$: a set of actions you can take, for example, moving from $(0, 0)$ to $(1, 1)$.
- $p(\cdot|s, a) \in \mathbb{R}$: probability of transitioning to another state when you take action a at state s. Notice $\sum_{s'} p(s'|s, a) = 1$. This is the so-called *transition kernel*, or transition probability.
- $\pi(a|s) \in \mathbb{R}$: probability of taking action $a \in \mathcal{A}$ given state $s \in \mathcal{S}$; this is the so-called *policy* $\pi : \mathcal{S} \to \mathcal{A}$. Notice $\sum_a \pi(a|s) = 1$.
- $r(s, a, s') \in \mathbb{R}$: a real number if you are at state s and take action a, and you arrived at state s'.
- $s_0, a_0, r_0, s_1, a_1, \cdots$: a sequence of elements termed as a *trajectory*, or the history of samples.
- $G_t := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$: the discounted return, note $r_t = r(s_t, a_t)$, and that $G_t = r_t + \gamma G_{t+1}$.

Intro

At each time step, the agent observes the current state s and subsequently takes an action a, which leads to an instantaneous reward $r(s, a, s')$ as well as the transition to the next state according to an unknown transition kernel $p(\cdot|s, a)$. Notice that for each state-action pair, the expected reward is constant, $r(s, a) = \sum_{s'} p(s'|s, a) r(s, a, s')$.
The eventual goal of the agent is to learn a policy to optimize the reward accrued over time.

Intro

Since the world and the policy are probabilistic, we always consider the expected return. Here we consider the **expected return** starting at a certain state s and follow policy π.

$$V^{\pi}(s) := \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^k r_t | s_0 = s\right].$$

Another very useful term is the expected return starting at a certain state s but **take action** a, and then follow policy π.

$$Q^{\pi}(s,a) := \mathbb{E}_{\pi}\left[G_t \mid s_t = s, a_t = a\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_t \mid s_0 = s, a_0 = a\right]$$

Problem formulation

1. Prediction problem:

$$\text{Evaluate } V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

2. Control problem:
Define $\Delta_{|\mathcal{A}|} := \left\{ p \in \mathbb{R}^{|\mathcal{A}|} \mid \sum_{i=1}^{|\mathcal{A}|} p_i = 1, p_i \geq 0 \right\}, \forall s \in \mathcal{S},$

$$\begin{array}{ll} \max_{\pi} & \mathbb{E}_{s \sim \nu} \left[V^{\pi}(s) \right] \\ \text{s.t.} & \pi(\cdot \mid s) \in \Delta_{|\mathcal{A}|}, \forall s \in \mathcal{S}. \end{array}$$

3. Bound number of samples required to find

$$\mathbb{E}_{s \sim \nu} \left[V^{\pi}(s) - V^{*}(s) \right] < \epsilon$$

where ϵ can be any desired accuracy, e.g. $\epsilon = 10^{-3}$, or $\epsilon = 10^{-3} \mathbb{E}_{s \sim \nu} \left[V^{*}(s) \right]$.

Bellman equation

Let $r_t = r(s_t, a_t, s'_{t+1})$. Given any feasible policy π, we can expand the value function recursively such that for any time step t

$$\begin{aligned} V^\pi(s) &:= \mathbb{E}_\pi \left[G_t \mid s_t = s \right] \\ &= \mathbb{E}_\pi \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s \right] \\ &= \mathbb{E}_\pi \left[r_t + \gamma G_{t+1} \mid s_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s'} p\left(s' \mid s, a\right) \left[r_t + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid s_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a|s) \sum_{s'} p\left(s' \mid s, a\right) \left[r_t + \gamma V^\pi\left(s'\right) \right], \qquad \text{for all } s \in \mathcal{S} \end{aligned} \tag{1}$$

We denote this as the Bellman equation. It is also easy to write the Bellman equation for the action-value function.

$$Q^\pi(s,a) = \sum_{s'} p\left(s' \mid s, a\right) \left[r_t + \gamma V^\pi\left(s'\right) \right], \qquad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}$$

MDP formulation

- Given any policy π, and some starting state s_0, it will generate some state sequence $s_0, s_1, s_2, s_3, \cdots$, which is a discrete-time Markov process (DTMC).
- Markovian property of DTMC: the next state only depends on the last state
- Most of the time we focus on a special kind of DTMC, which is *ergodic*. Such DTMC has a stable distribution ν.
- For ergodic DTMC, whatever is the starting state distribution $s_0 \sim p_0$, it will go to stable distribution $s_t \to s \sim \nu$.
- MDP = DTMC + any feasible policy
- *Uniform ergodicity*: assume ergodic DTMC for any policy computed in MDP (usual assumption).

Optimal Policy

An *optimal policy* is defined as the policy that has the value function better than or equal to any other value functions of any policies for all states. In other words, π is better than π' if and only if $V^{\pi}(s) \geq V^{\pi'}(s), \forall s$. There is at least one optimal policy π^* (I was told the derivation is highly nontrivial, detail is in Siva's notes). Optimal policies share the same value function, the so-called *optimal value function*

$$V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s),$$

and the same *optimal action value function*

$$Q^{\pi^*} = \max_{\pi} Q^{\pi}(s, a)$$

Optimal value functions

The optimal value function and the optimal action-value function itself satisfy the Bellman equation, together with the fact that the best action for any state is to pick the action with the largest Q function, so we have

$$
\begin{aligned}
V^{\pi^*}(s) &:= \max_{a \in \mathcal{A}} Q^{\pi^*}(s, a) \\
&= \max_{a \in \mathcal{A}} \mathbb{E}_{\pi^*} \left[r_t + \gamma G_{t+1} \mid s_t = s, a_t = a \right] \\
&= \max_{a \in \mathcal{A}} \mathbb{E}_{\pi^*} \left[r_t + \gamma V^{\pi^*}(s_{t+1}) \mid s_t = s, a_t = a \right] \\
&= \max_{a \in \mathcal{A}} \sum_{s'} p\left(s' \mid s, a\right) \left[r_t + \gamma V^{\pi^*}\left(s'\right) \right], \qquad \text{for all } s \in \mathcal{S}
\end{aligned}
\tag{2}
$$

Optimal action value functions

Similarly, we have the Bellman optimality equation for Q functions,

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}\left[r_t + \gamma \max_{a'} Q^{\pi^*}(s_{t+1}, a') \mid s_t = s, a_t = a\right] \\ &= \sum_{s'} p(s'|s,a)\left[r_t + \gamma \max_{a'} Q^{\pi^*}(s', a')\right] \end{aligned} \tag{3}$$

So far, this concludes contents from Sutton's book up to Chapter 3.

Dynamic Programming

From now on, we consider solving the RL/MDP problems starting with the prediction problem (evaluate a given policy), and then trying to solve the control problem (find a better policy at the same time). The first policy evaluation method is the *iterative policy evaluation*. Consider the following update rule,

$$\begin{aligned} V_{k+1}(s) &\doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma V_k \left(S_{t+1} \right) \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} p\left(s' \mid s, a \right) \left[r + \gamma V_k \left(s' \right) \right]. \end{aligned} \tag{4}$$

This is essentially treating V_k as the current best estimation of the true value function, and updating the estimation using the Bellman equation.

Iterative policy evaluation

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining the accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(\text{ terminal })$ to 0
Loop until $\Delta < \theta$

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{a} \pi(a \mid s) \sum_{s'} p\left(s' \mid s, a\right)\left[r + \gamma V\left(s'\right)\right]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

Policy improvement

Suppose you have given a policy π and its value functions V^π and Q^π, we consider the following update rule to find a better policy, guaranteed by the policy improvement theorem (not discussed today),

$$\begin{aligned}\pi'(s) &\doteq \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a \mathbb{E}\left[r_t + \gamma V^\pi\left(s_{t+1}\right) \mid s_t = s, a_t = a\right] \\ &= \arg\max_a \sum_{s'} p\left(s' \mid s, a\right)\left[r + \gamma V^\pi\left(s'\right)\right]\end{aligned}$$

The policy found is **greedy** because we are taking $\max$ from the previous policy. An example of a non-greedy policy would be, with probability $\epsilon << 1$, we take a random action a uniformly from the action space, and with $1 - \epsilon$ we pick actions according to the greedy policy.

Policy Iteration

How about we combine them? Let's introduce *policy iteration*: a single algorithm that performs value iteration and policy improvement at the same time.

Algorithm 1: Policy Iteration

Initialize V and π arbitrarily;
Loop until π stop changing;
    Perform Policy Evaluation;
    Perform Policy Improvement;

However, such an algorithm takes long to converge, as each step requires a policy evaluation step.

Value Iteration

How about we perform one step of Value Iteration and take the newly updated value function for policy improvement, instead of waiting for Value Iteration to converge?
Consider the following algorithm:

Algorithm 2: Value Iteration

while $\Delta \geq \theta$ **do**
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} p\left(s' \mid s, a\right)\left[r + \gamma V\left(s'\right)\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end

Notice the difference with the rule from policy evaluation:

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p\left(s' \mid s, a\right)\left[r + \gamma V\left(s'\right)\right]$$

Asynchronous Algos and GPI

- So far, all algorithms require a sweep for all states in a single iteration.
- This can be computationally challenging if the state space is large and we don't have easy access to all states. Suppose we only have a single trajectory of states $(s_0, s_1, s_2, s_3, \cdots)$.
- *Asynchronous* version of algorithms update $V(s_t)$ whatever s_t is. Thus we avoid waiting for all states to finish.

Generalized Policy Iteration

- In policy iteration, we decouple policy evaluation and optimization.
- How about we only perform a few steps of policy evaluation (PE) and then improve the policy (instead of waiting for PE to converge)?
- Sutton termed this as *generalized policy iteration* (GPI).
- GPI takes middle points of two extreme: value iteration that updates policies after one step of evaluation, and policy iteration that updates policies until PE converges.
- Turns out this type of algorithm converges. With each step, we are making progress towards the optimal policy.

Summary of Chapter 4

So far we covered Chapter 4.

- Policy Evaluation (PE), an algorithm that evaluates any given feasible policy π.
- Policy Improvement, an update rule to find a better policy when you know the value functions.
- Policy Iteration, an algorithm that combines policy evaluation and policy iteration iteratively.
- Value Iteration, an improved policy iteration that performs PI after one policy evaluation.
- Generalized policy iteration, a middle point framework between policy iteration and value iteration.

Monte Carlo Methods

- Dynamic Programming methods assume we know complete knowledge of the environment. This is a very restrictive assumption.
- For example, how do you build a model of a driving car that contains possible states transitioned from every known state and every action?
- Thus, we must perform estimation without the assumption of knowing the model.
- The first model we are going to introduce is the Monte Carlo method of estimating $V(s)$.

Monte Carlo Prediction

Given any feasible policy, we consider the following operation.

- The central idea is to use G_t from one trajectory $(s_0, a_0, r_0, s_1, \cdots)$ as an estimation of $V(s_0)$.
- This is exactly sampling from the trajectory distribution, hence the class of the method, Monte Carlo.
- We first describe the so-called *first-visit* MC prediction, which only takes the first estimation for a given state in a trajectory. The algorithm is described in the next slide.

First-visit MC prediction

Algorithm 3: First-visit MC Prediction for given π

Initialize $V(s)$ arbitrarily, and empty list $Returns(s)$
while *True* **do**
    Generate an episode from π
    $G \leftarrow 0$
    for t *from* $T-1$ *to* 0 **do**
        $G \leftarrow \gamma G + r_{t+1}$
            If s_t does not appears in $(s_0, s_1, \cdots, s_{t-1})$:
                Append G to $Returns(s_t)$
                $V(s_t) \leftarrow average(Returns(s_t))$
    end
end

Monte Carlo method for control problem

Now we consider an approach to find the optimal policy π^*. The intuition is to consider the GPI framework just as we did in DP, alternating between policy evaluation and policy improvement.

Algorithm 4: First-visit MC Control

Initialize $\pi, Q(s,a)$ arbitrarily, and empty list $Returns(s,a)$
while *True* **do**
  Generate an episode following π from specified initial state s_0 and action a_0
  $G \leftarrow 0$
  for t *from* $T-1$ *to* 0 **do**
    $G \leftarrow \gamma G + r_{t+1}$
    If s_t, a_t does not appears in $(s_0, a_0, s_1, a_1 \cdots, s_{t-1}, a_{t-1})$:
      Append G to $Returns(s_t)$
      $Q(s_t, a_t) \leftarrow average(Returns(s_t, a_t))$
      $\pi(s_t) \leftarrow \arg\max_a Q(s_t, a)$
  end
end

MC with Exploration

The previous method does not always work, because some environment does not provide trajectory sampling from a given start state and action pair. But in real-life scenarios this is impractical. For example, self-driving cars can't have enough exploration. Here we

propose a simple strategy that alleviates the problem: we still use the same MC framework for learning the Q function, but generate the trajectory using the ϵ-greedy policy. Namely, the policy has ϵ probability to choose a random action, while having $1 - \epsilon$ probability to choose $\arg\max_a Q(s_t, a)$.

Off-policy prediction

The problem of exploration brings another challenge: how can we evaluate the performance of a policy from the samples generated by another policy? In the previous method, we use π_b a non-greedy version of π, to generate samples, but try to evaluate π itself by learning its Q function. This brings the topic of *off-policy learning*.

Conceptually it is the process of learning the transition dynamics of the environment and using that to evaluate the policy at hand, or even try to improve the policy.

Importance Sampling

Now let's introduce some new terms. Given a starting state s_t, the probability of the subsequent state–action trajectory, $a_t, s_{t+1}, a_{t+1}, \cdots s_T$, occurring under any policy π is

$$\begin{aligned}
&\Pr\left\{a_t, s_{t+1}, a_{t+1}, \ldots, s_T \mid s_t, a_{t:T-1} \sim \pi\right\} \\
&\quad = \pi\left(a_t \mid s_t\right) p\left(s_{t+1} \mid s_t, a_t\right) \pi\left(a_{t+1} \mid s_{t+1}\right) \cdots p\left(s_T \mid s_{T-1}, a_{T-1}\right) \\
&\quad = \prod_{k=t}^{T-1} \pi\left(a_k \mid s_k\right) p\left(s_{k+1} \mid s_k, a_k\right)
\end{aligned}$$

Let's say the trajectory is generated by a *behavior policy* b, and the *importance sampling ratio* between b and any given π is defined as

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi\left(a_k \mid s_k\right) p\left(s_{k+1} \mid s_k, a_k\right)}{\prod_{k=t}^{T-1} b\left(a_k \mid s_k\right) p\left(s_{k+1} \mid s_k, a_k\right)} = \prod_{k=t}^{T-1} \frac{\pi\left(a_k \mid s_k\right)}{b\left(a_k \mid s_k\right)}$$

Off-policy prediction

The spirit of MC is based on the assumption that

$$\mathbb{E}\left[G_t \mid s_t = s\right] = V^b(s)$$

This implies that

$$\mathbb{E}\left[\rho_{t:T-1} G_t \mid s_t = s\right] = V^\pi(s)$$

Notice that for MC method, we can write estimation of the value function as

$$V^b(s) = \frac{\sum_{t \in \mathcal{T}(s)} G_t}{|\mathcal{T}(s)|}$$

where $\mathcal{T}(s)$ is the set of time steps that will be averaged for computing new estimates. For first-time MC methods, this would include visiting a given state across multiple trajectories for the first time. Following this, we can derive off-policy value estimation.

Off-policy prediction

- *ordinary importance sampling*

$$V^{\pi}(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1} G_t}{|\mathcal{T}(s)|}$$

  Unbiased estimator (for first-visit MC) but unbounded variance. Easier to analyze.

- *weighted importance sampling*

$$V^{\pi}(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:\mathcal{T}(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:\mathcal{T}(t)-1}}$$

  Biased estimator but variance converges to 0.

This can lead to a general policy evaluation algorithm for estimating Q or V, with $b = \pi$ as a special case for on-policy prediction.

Incremental Update

For ordinary importance sampling, one can re-use the previous first-visit MC prediction algorithm and simply replace the averaged return with a weighted average return. For weighted importance sampling, however, we can improve the update rule more efficiently. Let $W_i = \rho_{t_i:\mathcal{T}(t_i)-1}$. We wish to estimate

$$V_n^\pi = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}$$

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n]$$

$$C_{n+1} = C_n + W_{n+1}$$

Off-policy Control

Using incremental updates, we arrive at the following control method:

Algorithm 5: Off-policy MC control

Initialize $\pi, Q(s,a)$ arbitrarily, and empty list $Returns(s,a)$
while *True* **do**
    $b \leftarrow$ any soft policy (such as ϵ-greedy)
    Generate an episode following b $G \leftarrow 0$ $W \leftarrow 1$
    for t *from* $T-1$ *to* 0 **do**
        $G \leftarrow \gamma G + r_{t+1}$
        $C(s_t, a_t) \leftarrow C(s_t, a_t) + W$
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{W}{C(s_t, a_t)}[G - Q(s_t, a_t)]$
        $\pi(s_t) \leftarrow \arg\max_a Q(s_t, a_t)$
        If $a_t \neq \pi(s_t)$, exit inner loop and proceed to next episode
        $W \leftarrow W \frac{1}{b(a_t|s_t)}$
    end
end

Summary

Let's summarize Monte Carlo methods.

- Monte Carlo prediction for policy evaluation by generating episodes.
- Monte Carlo control for estimating Q function using essentially the same method.
- Off-policy learning using importance sampling, both in prediction and control.
- However these methods require updating after generating the entire episodes. Can we do better?

Temporal difference: intro

In MC methods, we update the value function using

$$V(s_t) \leftarrow V(s_t) + \alpha \left[G_t - V(s_t) \right]$$

where G_t is the sum of discounted returns after one episode. Recall that if we know s_t, a_t, r_t, s_{t+1}, from the Bellman equation, we have

$$V(s_t) = r_t + \gamma V(s_{t+1})$$

Then the question arises,

can we use $r_t + \gamma V(s_{t+1})$ *as an estimate of* G_t*?*

(luckily the answer is yes)

Temporal Difference: TD prediction

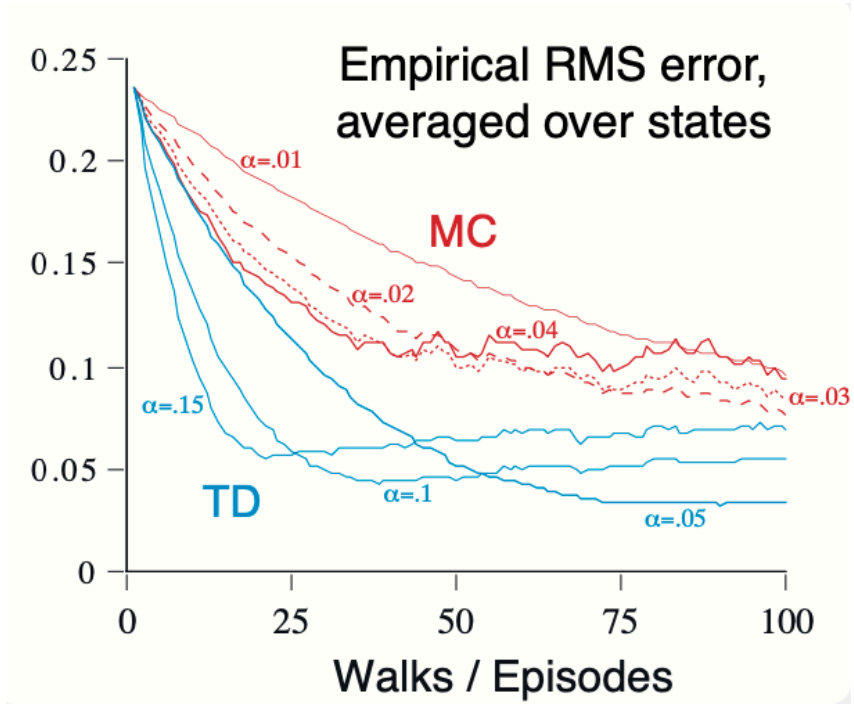
Suppose we are given a set of episodes generated by policy π, the following algorithm is to evaluate this policy.

Algorithm 6: Tabular TD(0) for estimating V^π

Given any policy π, and specify step size parameters $\alpha \in (0, 1]$
Initialize $V^\pi(s) \forall s \in \mathcal{S}$ and $V(\text{terminal}) = 0$
for *each episode* **do**
    Initialize s;
    for (s, a, r, s') *in this episode* **do**
        $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    end
end

TD prediction

- TD methods use the existing estimates, thus a *bootstrapping* method.
- The name $TD(0)$, or *one-step TD* indicates we update the estimate use immediate estimation $r + \gamma V(s')$.
- The term $\delta(s) \doteq r + \gamma V(s') - V(s)$, called *TD error* arises in many forms throughout RL, which has interesting connection to neural science.



Sarsa: on-policy TD Control

Similar to DP and Monte Carlo methods, we use the Q function for the control problem, which implies the following update rule.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Notice that this time we require a tuple of $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, hence the name *Sarsa*.

Sarsa

Algorithm 7: Sarsa for estimating Q^π

Given any policy π, and specify step size parameters $\alpha \in (0, 1]$
Initialize $Q^\pi(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A}$
for *each episode* **do**
    Initialize s;
    Choose action a using policy derived from Q (e.g. ε-greedy);
    for *time step in this episode* **do**
        Take action a, observe r, s';
        Choose action a' using policy derived from Q (e.g. ε-greedy);
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$;
        $s \leftarrow s', a \leftarrow a'$
    end
end

Sarsa

- The convergence of Sarsa (in high probability) is guaranteed if **all** state-action pairs are visited an infinite number of times (thus we need exploration from a non-greedy policy) and ε-greedy converging to the greedy policy in the limit, for example by setting $\varepsilon = 1/t$.
- Sarsa is hard to analyze. An early breakthrough in reinforcement learning is the development of the so-called *Q-learning*, which leads to convergence proofs (some variants are proved very recently)
- Notice a similar update rule of Sarsa can simplify the algorithm

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Q-learning: Off-policy TD Control

Let's examine the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Now the Q function directly approximates Q^{π^*}, the optimal action-value function, independent of the policy being followed. The resulting method thus is an *off-policy* algorithm.

Algorithm 8: Q-learning for estimating Q^{π}

for *each episode* **do**
    Initialize s;
    Choose action a using policy derived from Q (e.g. ε-greedy);
    for *time step in this episode* **do**
        Take action a, observe r, s';
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$;
        $s \leftarrow s'$
    end
end

Q-learning

- Q-learning is a very successful algorithm. It remains important in the landscape of modern RL.
- Modification and variants have been proposed based on Q-learning. For example, Async Q-learning.
- Deep Learning based RL methods rely heavily on the Q-learning update rule. But we will not introduce function approximation today.

Expected Sarsa

Let's introduce one variant of Q-learning: expected Sarsa. The idea is to replace $\max$ operation with expected value based on current state.

$$\begin{aligned} Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \mathbb{E}_\pi \left[Q(s_{t+1}, a_{t+1}) \mid s_{t+1} \right] - Q(s_t, a_t) \right] \\ &= Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \sum \pi(a \mid s_{t+1}) Q(s_{t+1}, a) - Q(s_t, a_t) \right] \end{aligned}$$

- In expectation, Sarsa moves in the same direction as Expected Sarsa, hence the name.
- In some cases, Expected Sarsa is significantly better than Q-learning or Sarsa.
- Expected Sarsa subsumes and generalizes the two TD algorithms, but required additional computations.

Maximization Bias and Double Learning

- In all TD control algorithms discussed, we take $\max$ over the estimated values. In this way, we implicitly use *maximum of estimated values* as *estimate of the maximum value*. This can lead to a significant positive bias.
- For example, for some state s, the true action-value function $Q(s,a) = 0$ for all a. But our current estimate $\widetilde{Q}(s,a)$ has some positive values and negative values. With the $\max$ operation, we implicitly over-estimate. Sutton calls this *maximization bias*.
- One way to view the problem is that it is due to using the same samples both to determine the maximizing action and to estimate values.
- This bring the idea of *Double Q-learning*, which maintains two Q estimates Q_1, Q_2, and replace $\max_a Q_1(s',a)$ with $Q_1(s', \arg\max_a Q(s',a))$.

Double Q-learning

Algorithm 9: Q-learning for estimating Q^π

for *each episode* **do**
  Initialize s;
  Choose action a using policy derived from Q (e.g. ε-greedy);
  for *time step in this episode* **do**
    Take action a, observe r, s';
    With 0.5 probability:
    $Q_1(s,a) \leftarrow$
     $Q_1(s,a) + \alpha\left[r + \gamma Q_1(s', \arg\max_a Q_2(s',a)) - Q_1(s,a)\right]$;
    else:
    $Q_2(s,a) \leftarrow$
     $Q_2(s,a) + \alpha\left[r + \gamma Q_2(s', \arg\max_a Q_1(s',a)) - Q_2(s,a)\right]$;
    $s \leftarrow s'$
  end
end

Summary

- We introduced TD methods, which in spirit update our estimating using correction from steps ahead.
- TD(0) for policy evaluation.
- Sarsa for on-policy control.
- Q-learning for off-policy control.
- Expected Sarsa, a generalized TD control method.

n-step bootstrap

So far the key ingredients of tabular RL are represented. We introduce some new ideas with those methods.

- Instead of one-step look ahead in TD, estimate the TD target from n steps followed.

$$V_{t+n}(s_t) \doteq V_{t+n-1}(s_t) + \alpha[G_{t:t+n} - V_{t+n-1}(s_t)]$$

  where $G_{t:t+n} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n V_{t+n-1}(s_{t+n})$.

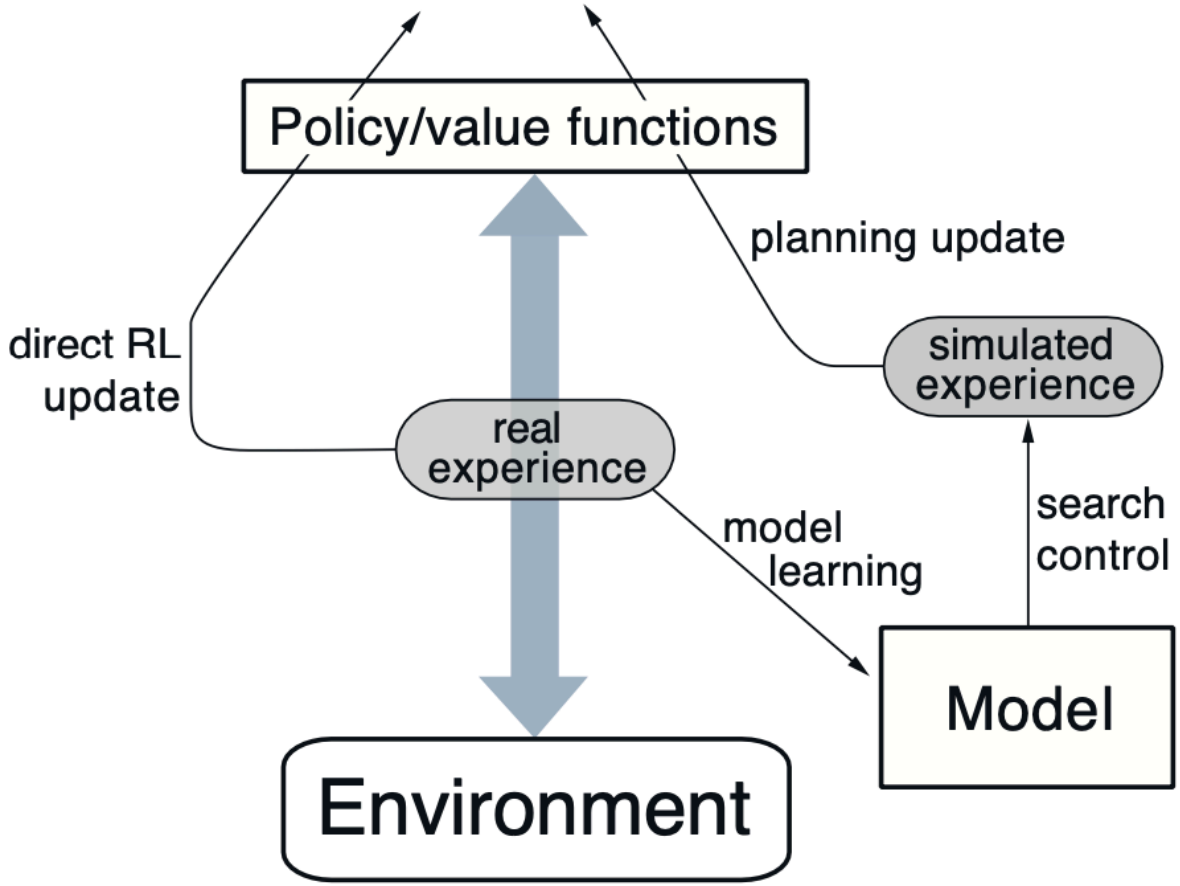
- Same idea can be used for Sarsa and off-policy learning with importance sampling.
- n step Q-learning, termed *n step Tree Backup*, achieves off-policy learning without importance sampling, by following the same recipe.
- Finally we can mix and match: take a convex combination of one-step target and n step target.

Planning and Learning

- So far we discussed DP methods, which requires a model of the environment to learn; and Monte Carlo and TD methods, which are model-free methods that update estimation from current estimates.
- Humans learn things in both ways: we build a model based on our understandings, and simulate experience based on those model, we learn things from both simulated and real experiences.
- We can achieve the same goal by combining both types of methods.

Dyna (1)

Figure: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, aects policy and value functions in much the same way as does simulated experience generated by the model of the environment.



Thank you.

feiyangwu@gatech.edu

References I

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT press, 2018.