Reinforcement Learning for Economists

Lecture 3

Temporal Difference Learning

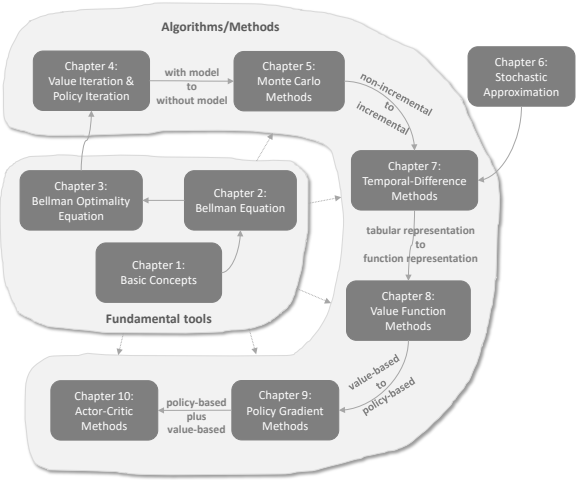
Benjamin Moll

London School of Economics

Plan for the four RL lectures

1. The basic idea of RL
2. Dynamic programming revisited
   - Value function iteration
   - Howard policy iteration
3. Monte Carlo methods
4. Stochastic approximation
5. Temporal difference learning, Sarsa, Q-learning
6. Policy gradient and actor-critic methods

Outline of Shiyu Zhao's book



Note: my slides are mostly a shortened/adapted version of Zhao's great slides

https://github.com/MathFoundationRL/Book-Mathematical-Foundation-of-Reinforcement-Learning

Temporal Difference Learning

Introduction

- This lecture introduces temporal-difference (TD) learning, which is one of the most well-known methods in reinforcement learning (RL).
- Monte Carlo (MC) learning is the first model-free method. TD learning is the second model-free method. TD has some advantages compared to MC.
- We will see how the stochastic approximation methods studied in the last lecture are useful.

Terminology: Incremental vs Episodic Learning

Recall simplest Monte Carlo RL algorithm. Policy evaluation step is:

$$q_{\pi_k}(s,a) = \mathbb{E}_{\pi_k}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0 = s, a_0 = a\right] \approx \frac{1}{N}\sum_{j=1}^{N}\sum_{t=0}^{T}\gamma^t r(s_t^j, a_t^j)$$

Can write as

$$q_{\pi_k}^{j+1}(s,a) = q_{\pi_k}^{j}(s,a) + \frac{1}{j}\left[\sum_{t=0}^{T}\gamma^t r(s_t^j, a_t^j) - q_{\pi_k}^{j}(s,a)\right] \qquad (*)$$

But note: always need to wait until end of episode $t = 0, \ldots, T$ to update estimate of Q function, cannot update estimate in real time

In contrast, this lecture: develop algorithms that update in "real time"

$$q_{t+1}(s_t,a_t) = q_t(s_t,a_t) + \alpha \times \text{stuff}_t, \quad q_t = \text{time-}t \text{ estimate of } q_{\pi_k} \qquad (**)$$

Terminology:

- "Incremental learning" = update in "real time" as in $(**)$
- "Non-incremental (or episodic) learning" = don't update in "real time" as in $(*)$

Note: Sutton-Barto also use "online/offline" but "offline RL" means something else

TD learning is incremental

From Sutton-Barto:

> Equation 2.4). Let us call this method *constant-α MC*. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is G_t known), TD methods need to wait only until the next time step. At time $t+1$ they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method makes the update

"Incremental" vs "episodic" is more general

Second part of lecture = TD learning with function approximation: write algorithms that update some parameter vector θ in real time

$$\theta_{t+1} = \theta_t + \alpha \times \text{stuff}(\theta_t) = \text{incremental algorithm}$$

Next lecture = policy gradient methods:

- Most basic one (REINFORCE): $\theta_{j+1} = \theta_j + \alpha \sum_{t=0}^{T} \gamma^t \text{stuff}_t(\theta_j)$ = episodic
- Actor-critic: $\theta_{t+1} = \theta_t + \alpha \times \text{stuff}_t$ = incremental

TD learning of state values $v_\pi(s)$

First focus on simpler problem: learn value function $v_\pi(s)$ for fixed policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \middle| s_0 = s \right], \quad s_{t+1} \sim p(\cdot|s_t, a_t), \ a_t \sim \pi(\cdot|s_t)$$

After that: learn action-value function $q_\pi(s, a)$ and combine with policy improvement step

Notation:

$$v_t(s) = \text{time } t \text{ estimate of } v_\pi(s)$$

TD learning of state values – Algorithm description

The TD learning algorithm is

$$v_{t+1}(s_t) = v_t(s_t) + \alpha_t(s_t)\big[[r_{t+1} + \gamma v_t(s_{t+1})] - v_t(s_t)\big], \tag{1}$$

$$v_{t+1}(s) = v_t(s), \quad \forall s \neq s_t, \tag{2}$$

where $t = 0, 1, 2, \ldots$

Here, $v_t(s_t)$ is the estimated state value of $v_\pi(s_t)$; $\alpha_t(s_t)$ is the learning rate of s_t at time t.

- At time t, only the value of the visited state s_t is updated whereas the values of the unvisited states $s \neq s_t$ remain unchanged.
- The update in (2) will be omitted when the context is clear.

TD learning of state values – Algorithm properties

The TD algorithm can be annotated as

$$\underbrace{v_{t+1}(s_t)}_{\text{new estimate}} = \underbrace{v_t(s_t)}_{\text{current estimate}} + \alpha_t(s_t)\overbrace{\Big[\underbrace{[r_{t+1} + \gamma v_t(s_{t+1})]}_{\text{TD target } \hat{v}_t} - v_t(s_t)\Big]}^{\text{TD error } \delta_t}, \tag{3}$$

Here,

$$\hat{v}_t \doteq r_{t+1} + \gamma v_t(s_{t+1})$$

is called the TD target.

$$\delta_t \doteq [r_{t+1} + \gamma v_t(s_{t+1})] - v_t(s_t) = \hat{v}_t - v_t(s_t)$$

is called the TD error.

Observation: The new estimate $v_{t+1}(s_t)$ is a combination of the current estimate $v_t(s_t)$ and the TD error.

TD learning of state values – Algorithm properties

First, why is $\hat{v}_t$ called the TD target?

That is because the algorithm drives $v(s_t)$ towards $\hat{v}_t$.

To see that,

$$
\begin{aligned}
& v_{t+1}(s_t) = v_t(s_t) + \alpha_t(s_t)\big[\hat{v}_t - v_t(s_t)\big] \\
\Longrightarrow \quad & v_{t+1}(s_t) - \hat{v}_t = v_t(s_t) - \hat{v}_t + \alpha_t(s_t)\big[\hat{v}_t - v_t(s_t)\big] \\
\Longrightarrow \quad & v_{t+1}(s_t) - \hat{v}_t = [1 - \alpha_t(s_t)][v_t(s_t) - \hat{v}_t] \\
\Longrightarrow \quad & |v_{t+1}(s_t) - \hat{v}_t| = |1 - \alpha_t(s_t)|\,|v_t(s_t) - \hat{v}_t|
\end{aligned}
$$

Since $\alpha_t(s_t)$ is a small positive number, we have

$$0 < 1 - \alpha_t(s_t) < 1$$

Therefore,

$$|v_{t+1}(s_t) - \hat{v}_t| \leq |v_t(s_t) - \hat{v}_t|$$

which means $v(s_t)$ is driven towards $\hat{v}_t$!

TD learning of state values – Algorithm convergence

Theorem (Convergence of TD Learning)

By the TD algorithm (1), $v_t(s)$ converges with probability 1 to $v_\pi(s)$ for all $s \in \mathcal{S}$ as $t \to \infty$ if $\sum_t \alpha_t(s) = \infty$ and $\sum_t \alpha_t^2(s) < \infty$ for all $s \in \mathcal{S}$.

The proof of the theorem can be found in Shiyu Zhao's book.

Remarks:

- This theorem says the state value can be found by the TD algorithm for a given policy π.
- $\sum_t \alpha_t(s) = \infty$ and $\sum_t \alpha_t^2(s) < \infty$ must be valid for all $s \in \mathcal{S}$.
  - For condition $\sum_t \alpha_t(s) = \infty$: At time step t,
    - ◇ If $s = s_t$, then $\alpha_t(s) > 0$;
    - ◇ If $s \neq s_t$, then $\alpha_t(s) = 0$.

    As a result, $\sum_t \alpha_t(s) = \infty$ requires every state must be visited an infinite (or sufficiently many) number of times.
  - For condition $\sum_t \alpha_t^2(s) < \infty$: In practice, the learning rate α is often selected as a small constant. In this case, the condition that $\sum_t \alpha_t^2(s) < \infty$ no longer holds. When α is constant, it can still be shown that the algorithm converges in an expectation sense.

TD learning of state values – Algorithm properties

While TD learning and MC learning are both model-free, what are the **advantages and disadvantages** of TD learning compared to MC learning?

TD/Sarsa learning	MC learning
Incremental: TD learning is incremental. It can update the state/action values immediately after receiving a reward.	**Episodic:** MC learning is episodic. It has to wait until an episode has been completely collected.
Continuing tasks: Since TD learning is incremental, it can handle both episodic and continuing tasks.	**Episodic tasks:** Since MC learning is episodic, it can only handle episodic tasks that have terminal states.

Table: Comparison between TD learning and MC learning.

TD learning of state values – Algorithm properties

While TD learning and MC learning are both model-free, what are the **advantages and disadvantages** of TD learning compared to MC learning?

TD/Sarsa learning	MC learning
Bootstrapping: TD bootstraps because the update of a value relies on the previous estimate of this value. Hence, it requires initial guesses.	**Non-bootstrapping:** MC is not bootstrapping, because it can directly estimate state/action values without any initial guess.
Low estimation variance: TD has lower variance than MC because there are fewer random variables. For instance, Sarsa requires $R_{t+1}, S_{t+1}, A_{t+1}$.	**High estimation variance:** To estimate $q_\pi(s_t, a_t)$, we need samples of $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$. Suppose the length of each episode is L. There are $\lvert\mathcal{A}\rvert^L$ possible episodes.

Table: Comparison between TD learning and MC learning (continued).

TD learning of action values – Sarsa

- The TD algorithm introduced in the last section can only estimate state values.
- Next, we introduce Sarsa, an algorithm that can directly estimate action values.
- We will also see how to use Sarsa to find optimal policies.

Sarsa – Algorithm

First, our aim is to estimate the action values of a given policy π.

Suppose we have some experience $\{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})\}_t$.

We can use the following *Sarsa* algorithm to estimate the action values:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t(s_t, a_t)\Big[[r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})] - q_t(s_t, a_t)\Big],$$
$$q_{t+1}(s, a) = q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t),$$

where $t = 0, 1, 2, \ldots$

- $q_t(s_t, a_t)$ is an estimate of $q_\pi(s_t, a_t)$;
- $\alpha_t(s_t, a_t)$ is the learning rate depending on s_t, a_t.

Sarsa – Algorithm

- **Why is this algorithm called Sarsa?** That is because each step of the algorithm involves $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. Sarsa is the abbreviation of state-action-reward-state-action.
- **What is the relationship between Sarsa and the previous TD learning algorithm?** We can obtain Sarsa by replacing the state value estimate $v(s)$ in the TD algorithm with the action value estimate $q(s, a)$. As a result, Sarsa is an action-value version of the TD algorithm.
- **What does the Sarsa algorithm do mathematically?** The expression of Sarsa suggests that it is a stochastic approximation algorithm solving the following equation:

$$q_\pi(s, a) = \mathbb{E}\big[R + \gamma q_\pi(S', A') \mid s, a\big], \quad \forall s, a.$$

This is another expression of the Bellman equation expressed in terms of action values. The proof is given in Shiyu Zhao's book.

Sarsa – Algorithm

Theorem (Convergence of Sarsa learning)

By the Sarsa algorithm, $q_t(s, a)$ *converges with probability 1 to the action value* $q_\pi(s, a)$ *as* $t \to \infty$ *for all* (s, a) *if* $\sum_t \alpha_t(s, a) = \infty$ *and* $\sum_t \alpha_t^2(s, a) < \infty$ *for all* (s, a).

Remarks:

- This theorem says that the action values can be found by Sarsa for a given policy π.

Sarsa – Implementation

The ultimate goal of RL is to find optimal policies.

To do that, we can combine Sarsa with a policy improvement step.

The combined algorithm is also called Sarsa.

Pseudocode: Policy searching by Sarsa

for each episode **do**

Generate a_0 at s_0 following $\pi_0(s_0)$

for s_t ($t = 0, 1, 2, \ldots$) not the target state, **do**

Collect an experience sample $(r_{t+1}, s_{t+1}, a_{t+1})$ given (s_t, a_t): generate r_{t+1}, s_{t+1} by interacting with the environment; generate a_{t+1} following $\pi_t(s_{t+1})$.

Update q-value for (s_t, a_t):

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t(s_t, a_t)\big[r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1}) - q_t(s_t, a_t)\big]$$

Update policy for s_t:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1) \quad \text{if } a = \arg\max_a q_{t+1}(s_t, a)$$

$$\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s_t)|} \quad \text{otherwise}$$

$s_t \leftarrow s_{t+1}, \quad a_t \leftarrow a_{t+1}$

TD learning of optimal action values – Q-learning

Sarsa estimates action values of a given policy $\pi \Rightarrow$ must be combined with a policy improvement step. Q-learning directly estimates optimal action values:

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t(s_t, a_t)\Big[[r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)] - q_t(s_t, a_t)\Big],$$
$$q_{t+1}(s, a) = q_t(s, a), \quad \forall (s, a) \neq (s_t, a_t)$$

Very similar to Sarsa, they differ only in the TD target:

- Q-learning: $r_{t+1} + \gamma \max_{a \in \mathcal{A}} q_t(s_{t+1}, a)$ vs Sarsa: $r_{t+1} + \gamma q_t(s_{t+1}, a_{t+1})$

What does Q-learning do mathematically? It is a stochastic approximation algorithm solving

$$q(s, a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_a q(S_{t+1}, a) \,\Big|\, S_t = s, A_t = a\Big], \quad \forall\, s, a.$$

This is the Bellman *optimality* equation expressed in terms of action values $\Rightarrow$ Q-learning estimates q_* directly, no policy improvement step needed

Q-learning – Implementation

Q-learning is "off-policy": the update only needs $(s_t, a_t, r_{t+1}, s_{t+1})$, so the data can be generated by any policy ("behavior policy"), not the one being learned. Simplest implementation (a_t generated by ε-greedy, as in Sarsa):

Pseudocode: Policy searching by Q-learning

for each episode **do**

for s_t ($t = 0, 1, 2, \ldots$) not the target state, **do**

Collect an experience sample (a_t, r_{t+1}, s_{t+1}) given s_t: generate a_t following $\pi_t(s_t)$; generate r_{t+1}, s_{t+1} by interacting with the environment.

Update q-value for (s_t, a_t):

$$q_{t+1}(s_t, a_t) = q_t(s_t, a_t) + \alpha_t(s_t, a_t)\left[r_{t+1} + \gamma \max_a q_t(s_{t+1}, a) - q_t(s_t, a_t)\right]$$

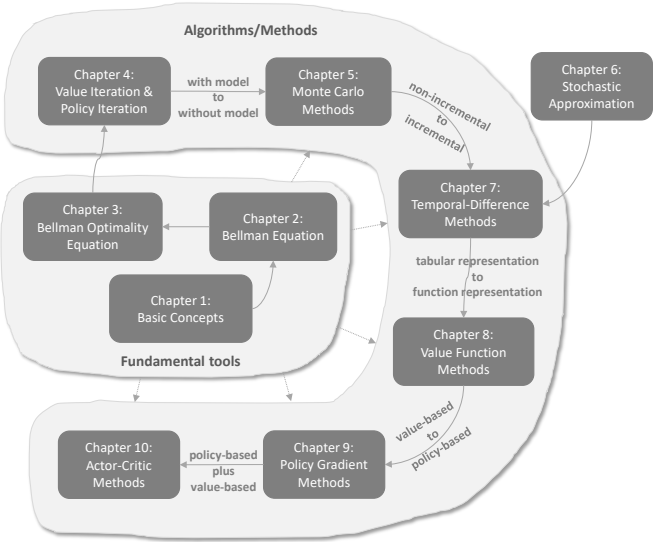
Update policy for s_t:

$$\pi_{t+1}(a|s_t) = 1 - \frac{\varepsilon}{|\mathcal{A}(s_t)|}(|\mathcal{A}(s_t)| - 1) \quad \text{if } a = \arg\max_a q_{t+1}(s_t, a)$$
$$\pi_{t+1}(a|s_t) = \frac{\varepsilon}{|\mathcal{A}(s_t)|} \quad \text{otherwise}$$

Note: no a_{t+1} needed for the update, unlike Sarsa. See Zhao's book for a fully off-policy version (all data from a fixed exploratory behavior policy)

Temporal Difference Learning with Function Approximation

Outline


Algorithms/Methods
Chapter 4: Value Iteration & Policy Iteration
with model to without model
Chapter 5: Monte Carlo Methods
non-incremental to incremental
Chapter 6: Stochastic Approximation
Chapter 7: Temporal-Difference Methods
Chapter 3: Bellman Optimality Equation
Chapter 2: Bellman Equation
Chapter 1: Basic Concepts
Fundamental tools
tabular representation to function representation
Chapter 8: Value Function Methods
value-based to policy-based
Chapter 9: Policy Gradient Methods
policy-based plus value-based
Chapter 10: Actor-Critic Methods

Motivating examples: curve fitting

So far in these lectures, state and action values are represented by tables.

- For example, action value:

	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

- Advantage: intuitive and easy to analyze
- Disadvantage: difficult to handle large or continuous state or action spaces. Two aspects: 1) storage; 2) generalization ability

Motivating examples: curve fitting

- **Idea:** Approximate the state and action values using parameterized functions: $\hat{v}(s, w) \approx v_\pi(s)$ where $w \in \mathbb{R}^m$ is the parameter vector.
- **Key difference:** How to access and assign the value of $v_\pi(s)$
- **Advantage:**
  1) **Storage:** The dimension of w may be much less than $|\mathcal{S}|$.
  2) **Generalization:** When a state s is visited, the parameter w is updated so that the values of some other unvisited states can also be updated.

Motivating examples: curve fitting

Consider an example:

- Suppose there are one-dimensional states $s_1, \ldots, s_{|\mathcal{S}|}$.
- Their state values are $v_\pi(s_1), \ldots, v_\pi(s_{|\mathcal{S}|})$, where π is a given policy.
- Suppose $|\mathcal{S}|$ is very large and we hope to use a simple curve to approximate these dots to save storage.

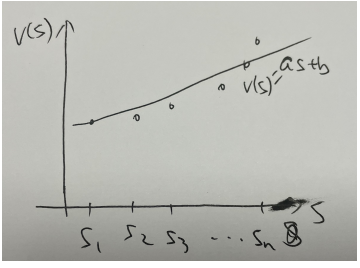


Figure: An illustration of function approximation of samples.

Motivating examples: curve fitting

First, we use the simplest straight line to fit the dots.
Suppose the equation of the straight line is

$$\hat{v}(s, w) = as + b = \underbrace{[s,\ 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s)\, w$$

where

- w is the parameter vector
- $\phi(s)$ the feature vector of s
- $\hat{v}(s, w)$ is linear in w

Approximation with neural networks

Can also use deep neural network to approximate v:

$$\hat{v}(s, w)$$

Now the parameter vector w are the "weights" of the neural network

RL + DNN = Deep reinforcement learning

But important: RL and DNNs are orthogonal / completely separate tools

Optimization algorithms

Objective: find w that minimizes the mean squared error

$$J(w) = \mathbb{E}\big[\big(v_\pi(S) - \hat{v}(S, w)\big)^2\big]$$

How? SGD from last lecture with $f(w) = (v_\pi(s_t) - \hat{v}(s_t, w))^2$ (2 absorbed into α_t):

$$w_{t+1} = w_t - \alpha_t \nabla_w f(w_t) = w_t + \alpha_t \big(v_\pi(s_t) - \hat{v}(s_t, w_t)\big) \nabla_w \hat{v}(s_t, w_t)$$

Problem: we don't know $v_\pi(s_t)$ $\Rightarrow$ replace it by something we can compute:

- **Monte Carlo learning with function approximation:** replace $v_\pi(s_t)$ by g_t, the discounted return starting from s_t in the episode

$$w_{t+1} = w_t + \alpha_t \big(g_t - \hat{v}(s_t, w_t)\big) \nabla_w \hat{v}(s_t, w_t)$$

- **TD learning with function approximation:** by the spirit of TD learning, replace $v_\pi(s_t)$ by the TD target $r_{t+1} + \gamma\, \hat{v}(s_{t+1}, w_t)$

$$w_{t+1} = w_t + \alpha_t \big[r_{t+1} + \gamma\, \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\big] \nabla_w \hat{v}(s_t, w_t)$$

Sarsa with function approximation

So far, we merely considered the problem of state value estimation. That is, we hope

$$\hat{v} \approx v_\pi$$

To search for optimal policies, we need to estimate action values.

The Sarsa algorithm with value function approximation is

$$w_{t+1} = w_t + \alpha_t \big[r_{t+1} + \gamma\, \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \big]\, \nabla_w \hat{q}(s_t, a_t, w_t).$$

This is the same as the algorithm we introduced previously in this lecture except that $\hat{v}$ is replaced by $\hat{q}$.