Markov Decision Processes and Reinforcement Learning

Day 4 — From Evaluation to Control

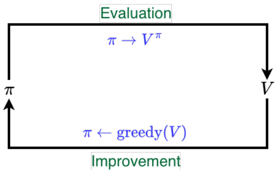
Luiz Felipe Martins

Department of Mathematics and Statistics
Cleveland State University

January 29, 2026

Outline

Generalized Policy Iteration



- Policy iteration alternates two stages:
  - **Evaluation**: Given a policy π, evaluate to V^π.
  - **Improvement**: Given a value function V, compute a policy π greedy with respect to V.
- **Generalized policy iteration** (GPI) algorithm: interleaves these two processes arbitrarily.
  - Value iteration: policy improvement step is performed after $V(s)$ is updated for each state.
  - Asynchronous DP: the two processes are carried out in parallel.

First-visit MC method

- Input:
  - π, a policy to be evaluated.
- Initialization.
  - $V(s)$, initialized arbitrarily.
  - $\texttt{returns}(s)$, an empty list for each state s.
- Repeat a large enough number of times:
  - Generate an episode following π: $(S_0, A_0, R_0), \dots, (S_T, A_T, R_T)$
  - Let $G = 0$
  - Loop for $t = T-1, T-2, \dots, 0$:
    - $G = \gamma G + R_{t+1}$
    - If S_t does not appear in $S_0, S_2, \dots, S_{t-1}$, append G to $\texttt{returns}(S_t)$
- Let $V(s) = \texttt{average}(\texttt{returns}(s))$ for $s \in \mathscr{S}$

First-visit MC method — Example

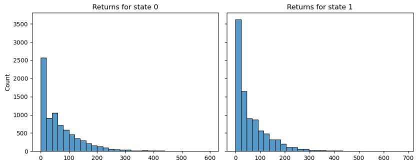
- For the Tri-state MDP, with policy $\pi(0) = a$, $\pi(1) = b$, simulating 10000 episodes we get the estimates:

$$V(0) = 71.18310, \quad V(1) = 63.25460$$

- Actual values:

$$V(0) = 71.25000, \quad V(1) = 63.57143$$

- Distribution of returns:



- $\mathrm{std}[V(0)] = 74.77556$, $\mathrm{std}[V(1)] = 74.56148$

Every-visit MC method

- Input:
  - π, an admissible policy to be evaluated.
- Initialization.
  - $V(s)$, initialized arbitrarily.
  - returns(s), an empty list for each state s.
- Repeat a large enough number of times:
  - Generate an episode following π: $(S_0, A_0, R_0), \ldots, (S_T, A_T, R_T)$
  - Let $G = 0$
  - Loop for $t = T-1, T-2, \ldots, 0$:
    - $G = \gamma G + R_{t+1}$
    - Append G to returns(S_t)
- Let $V(s) = \texttt{average}(\texttt{returns}(s))$ for $s \in \mathscr{S}$

Every-visit MC method — Example

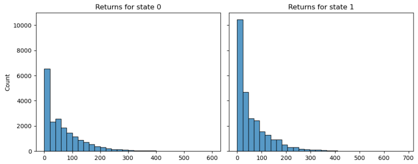
- For the Tri-state MDP, with policy $\pi(0) = a$, $\pi(1) = b$, simulating 10000 episodes we get the estimates:

$$V(0) = 69.84248, \quad V(1) = 62.18433$$

- Actual values:

$$V(0) = 71.25000, \quad V(1) = 63.57143$$

- Distribution of returns:



- $\mathrm{std}[V(0)] = 70.86432$, $\mathrm{std}[V(1)] = 73.64984$

Dynamic update of averages

- Suppose that we have a simulation that gives us a sequence return values $G_1, G_2, \ldots, G_N$. We use the average to estimate the expected return:

$$V_N = \frac{1}{N} \sum_{j=1}^{N} G_N$$

- Now, we compute a new value, G_{N+1}. How should we adjust the average? A simple computation shows that:

$$V_{N+1} = V_N + \frac{1}{N+1} [G_{N+1} - V_N]$$

- In control theory, this process is called *tracking*: The values of V_N are updated in a way that they "track" the values of G_N.
- Engineers like this because *the dynamics of the process G_N may change*, and the values of V_N can still try to "follow" it.

Dynamic update in Monte Carlo methods

- Monte Carlo evaluation creates a list of returns for each state s:

$$[G_1, G_2, \dots, G_N]$$

- The value function is estimated by:

$$V_N(s) = \frac{1}{N} \sum_{j=1}^{N} G_j$$

- Suppose we generate a new episode and get another return. Then:

$$V_{N+1}(s) = \frac{1}{N+1} \sum_{j=1}^{N+1} G_j = \frac{1}{N+1} \sum_{j=1}^{N} G_j + \frac{1}{N+1} G_{N+1}$$
$$= \frac{N}{N+1} V_N(s) + \frac{1}{N+1} G_{N+1} = V_N(s) + \frac{1}{N+1} [G_{N+1} - V_N(s)]$$

First-visit Monte Carlo with dynamic updates

- Input:
  - π, a policy to be evaluated.
- Initialization.
  - $V(s)$, initialized arbitrarily.
  - $\texttt{visits}(s) = 0$ for each state s.
- Repeat a large enough number of times:
  - Generate an episode following π: $(S_0, A_0, R_0), \ldots, (S_T, A_T, R_T)$
  - Let $G = 0$
  - Loop for $t = T-1, T-2, \ldots, 0$:
    - $G = \gamma G + R_{t+1}$
    - If S_t does not appear in $S_0, S_2, \ldots, S_{t-1}$, increment $\texttt{visits}(s)$ by one and update:

$$V(s) \leftarrow V(s) + \frac{1}{\texttt{visits}(s)}[G - V(s)]$$

Target tracking

- The update rule:

$$V_{N+1} \leftarrow V_N + \frac{1}{N+1}[G_{N+1} - V_N]$$

  is an example of *tracking* a non-stationary sequence G_N.
- The general form of this update rule is:

$$\texttt{new\_estimate} = \texttt{old\_estimate} + \texttt{step\_size}[\texttt{target} - \texttt{old\_estimate}]$$

- A wide variety of learning algorithms fit this pattern.

TD Learning, Monte Carlo and DP

- TD Learning is a combination of Dynamic Programming and Monte Carlo methods.
- Monte Carlo aspect: learn from experience, does not need a model for the environment dynamics.
- DP aspect: update estimates based on other learned estimates (bootstrap).
- TD is the basis for many current RL algorithms.

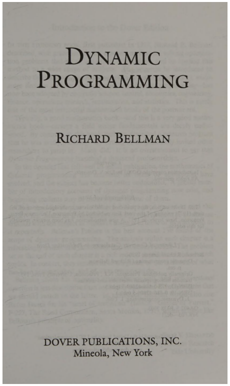
Framework for policy evaluation

- Suppose we have:
  - An estimate $V(s)$ for the state value function for a policy π.
  - An episode $S_0, S_1, \ldots, S_T$ generated with the policy π.
- Denote by G_t the total discounted return for this episode after time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

- For every visited state S_t, we update $V(S_t)$ to move in the direction of G_t.
- Different update strategies lead to distinct algorithms: Monte Carlo, TD(0), etc.

A little bit of philosophy


DYNAMIC PROGRAMMING

RICHARD BELLMAN

DOVER PUBLICATIONS, INC.
Mineola, New York

"Life can only be understood backwards, but it must be lived forwards." (Danish philosopher Soren Kierkegaard)

From Monte Carlo TD(0)

- Monte Carlo update rule:

$$V(S_t) \leftarrow V(S_t) + \frac{1}{\texttt{visits}(S_t)}[G_t - V(S_t)]$$

- Tracking with constant stepsize:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- TD(0): Instead of waiting for a whole episode to end, we use a one-step lookahead estimate of the value function:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD relationship to to DP

- Bellman Equation for state value function V of a policy π:

$$V(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1}) \,|\, S_t = s]$$

  The policy π may be stochastic or deterministic.

- Value iteration is a *fixed point iteration* for the exact value of $V(s)$:

$$V(s) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1}) \,|\, S_t = s]$$

- TD(0) is a *stochastic fixed point iteration* that tracks a one-step lookahead:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

  where $S_0, S_1, \ldots, S_T$ is an episode generated following policy π.

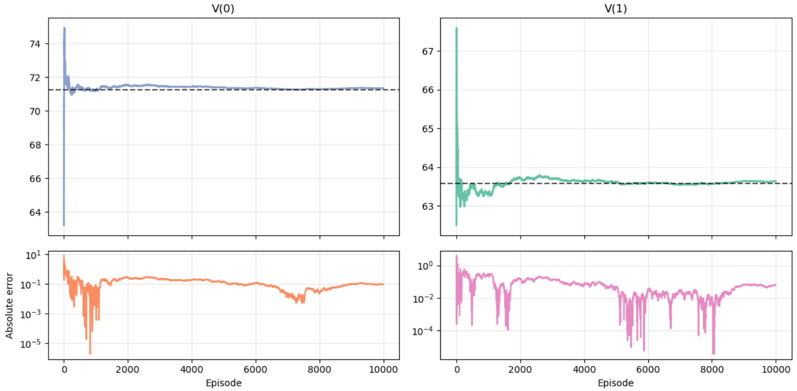
TD(0), Monte Carlo and DP

$$V_{\text{new}}(s) = V_{\text{old}}(S_t) + \alpha[r + \gamma V_{\text{old}}(s') - V_{\text{old}}(s)]$$

- Relationship to Monte Carlo:
  - Can learn directly from experience, independently on a model.
  - Monte Carlo algorithms have no bias and high variance.
  - TD(0) has bias controlled by the step size α and low variance.
- Relationship to DP
- TD methods update estimates based on already learned estimates (bootstrap)
- The TD(0) update rule is a form of *stochastic fixed point iteration*

Estimation by the Monte Carlo method

- Monte Carlo estimates for the Tri-state MDP with policy $\pi(0) = \mathtt{a}$, $\pi(1) = \mathtt{b}$ with 10000 episodes.
- Do 100 runs of the method and average.

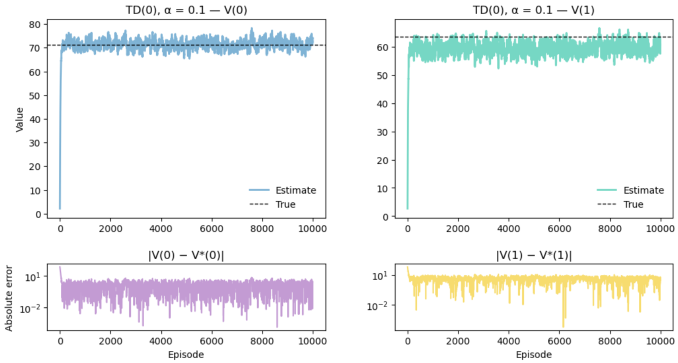


Interpretation of results for MC

- The value function estimates exhibit large transient oscillations, but stabilize around the true values. This behavior is consistent with an unbiased estimator.
- The absolute errors stabilize around 10^{-1}.
- The errors do not converge to zero at finite time due to the intrinsic variance of Monte Carlo returns and the finite number of effective samples per state.
- The magnitude of the error is governed by the standard deviation of the return distribution (empirically on the order of 70 in this model), with error scaling approximately as $\sigma_G/\sqrt{N}$.
- The sharp "dips" in the error correspond to episode indices where the mean estimate of $V(s)$ happens to be very close to the true value; this reflects bias cancellation rather than simultaneous accuracy across runs and is expected behavior.

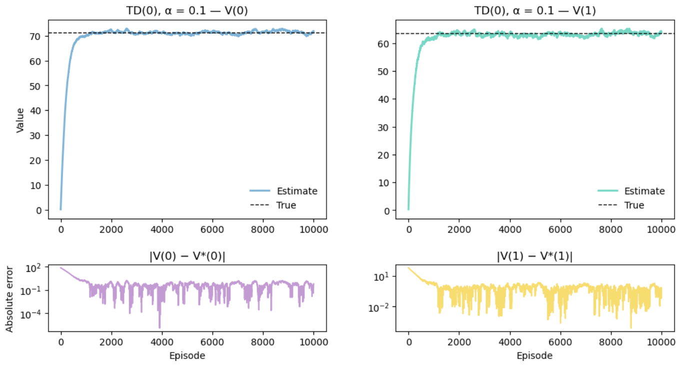
Estimation using TD(0) — $\alpha = 0.1$

- TD(0) estimates for the Tri-state MDP with policy $\pi(0) = \mathtt{a}$, $\pi(1) = \mathtt{b}$ with $\alpha = 0.1$ and 10000 episodes.
- Do 100 runs of the method and average.



Estimation using TD(0) — $\alpha = 0.01$

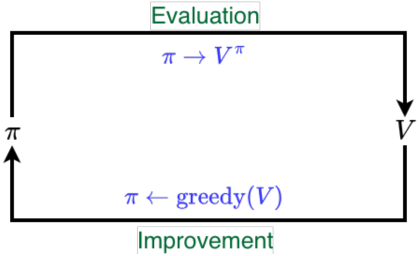
- TD(0) estimates for the Tri-state MDP with policy $\pi(0) = \mathtt{a}$, $\pi(1) = \mathtt{b}$ with $\alpha = 0.01$ and 10000 episodes.
- Do 100 runs of the method and average.



Interpretation of results for TD(0)

- TD(0) “converges” rapidly to a neighborhood “some value”.
- With a fixed step-size, TD(0) does not converge pointwise, but to a stationary distribution around a biased fixed point.
- The stationary bias is *state-dependent* and need not be the same for all states.
- States with lower visitation frequency or closer to termination exhibit both higher variance and larger apparent bias.
- The variance of the estimator does not vanish and is controlled by the step-size: larger step-sizes yield faster learning but higher steady-state variance.

From estimation to control



We have not explored the “Evaluation” branch of the cycle. Next time we will see how this can be incorporated in a policy optimization algorithm.

The exploration $\times$ exploitation dillema

- If we experiment a bit with an environment, we may be able to quickly get reasonable actions for some states.
- If we adopt a greedy policy, we will get good immediate rewards for the actions we tried (**Exploitation**).
- However, if we only stick to the actions we are used to, we may be missing actions that, even though their immediate reward is not great, it will lead to better returns later.
- Every once in a while, we should try some random action, just to learn about unexplored regions of the environment (**Exploration**).

From TD estimation to TD control

- Update rule for TD estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- When doing control, use $Q(s, a)$ instead of $V(s)$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- It is important to guarantee that *all state-action pairs are visited* with probability 1. For example, we can use an ϵ-greedy policy:

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathscr{A}(s)|} & \text{if } a \in \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|\mathscr{A}(s)|} & \text{otherwise} \end{cases}$$

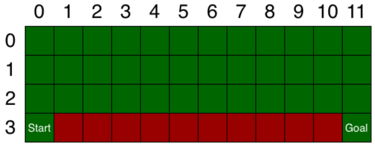
SARSA — On-policy TD control

- Loop for each episode:
    - Initialize S.
    - Choose A from S using policy derived from Q (for example, ϵ-greedy).
    - Loop for each step in the episode:
        - Take action A, observe R, S'
        - Choose A' from S' using policy derived from Q (for example, ϵ-greedy).
        - Let $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S,A') - Q(S,A)]$
        - $S \leftarrow S'$, $A \leftarrow A'$

SARSA is an acronym for:

$$(S, A, R, S', A')$$

Cliff walking



- Environment: 4×12 rectangular grid.
- A cliff runs through cells $(3,1)$ to $(3,10)$
- Starts at cell $(3,0)$, goal is to reach cell $(3,11)$.
- Actions: move up, right, down, left.
- Rewards:
  - Normal step: -1
  - Enter cliff: -100
  - Reach goal: 0
- Notice that model is deterministic.

Gymnasium

- **Gymnasium** provides a standardized API for reinforcement learning environments.
- Predefined classic environments, such as CartPole, CliffWalking, FrozenLake.
- Some environments expose their underlying models via `env.unwrapped.P`.
- Designed to be interoperable with modern RL libraries.
- It provides reproducibility through explicit seeding and controlled random number generation.
- Library is actively maintained and distributed.

Sample of available environments

Environment	Brief description
CartPole	Balance a pole on a moving cart; classic control and stability task.
Acrobot	Swing a two-link pendulum to reach an upright configuration.
Pendulum	Continuous control of a torque-driven pendulum to upright position.
CliffWalking	Gridworld navigation with severe penalties for falling off a cliff.
FrozenLake	Stochastic gridworld with slippery transitions and sparse rewards.
LunarLander	Control a spacecraft to achieve a safe, fuel-efficient landing.
Blackjack	Finite MDP modeling a simplified card game with terminal rewards.
Atari (ALE)	High-dimensional, pixel-based environments for deep RL benchmarks.

Environment creation and simulation

- Importing package:

```
import gymnasium as gym
```

- Environment creation:

```
env = gym.make("CliffWalking-v1")
```

- Resetting environment (required before starting episode):

```
state, _ = env.reset(seed=77)
```

- Advancing one step:

```
next_state, reward, terminated, _, _ = env.step(action)
```

Example episode

```
Initial state: 36
Step  0: s=36, a=2, r=-1.0, s'=36
Step  1: s=36, a=1, r=-100.0, s'=36
Step  2: s=36, a=2, r=-1.0, s'=36
Step  3: s=36, a=0, r=-1.0, s'=24
Step  4: s=24, a=1, r=-1.0, s'=25
Step  5: s=25, a=3, r=-1.0, s'=24
Step  6: s=24, a=2, r=-1.0, s'=36
Step  7: s=36, a=1, r=-100.0, s'=36
Step  8: s=36, a=2, r=-1.0, s'=36
Step  9: s=36, a=2, r=-1.0, s'=36

Episode finished after 10 steps
Total return: -208.0
```

- Actions: 0–up, 1–right, 2–down, 3–left
- If the walker falls through the cliff, the episode does not end.

SARSA implementation

- Greedy policy derived from $Q^{\pi_\epsilon^*}(s,a)$:

→	→	→	→	→	→	→	→	→	←	↓	→
→	↑	↑	↑	→	→	→	↑	↑	→	↓	→
↑	←	↑	↑	↑	→	↑	↑	↑	↑	→	↓
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑

- Optimal policy:

→	→	→	→	→	→	→	→	→	→	→	↓
→	→	→	→	→	→	→	→	→	→	→	↓
→	→	→	→	→	→	→	→	→	→	→	↓
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	→	→

GLIE policies

- GLIE — Greedy in the limit with infinite exploration: $\epsilon \to 0$

→	→	→	→	→	→	→	→	→	→	↓	↓
→	↑	↑	→	↑	←	↑	↑	↑	↑	→	↓
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	→	↓
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑

- Optimal policy:

→	→	→	→	→	→	→	→	→	→	→	↓
→	→	→	→	→	→	→	→	→	→	→	↓
→	→	→	→	→	→	→	→	→	→	→	↓
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	→	→

Slippery ground

- SARSA for slippery model: probability 2/3 of moving in a direction perpendicular to the one chosen.

↑	↓	→	→	→	←	↑	↓	↑	↓	→	→
←	↑	↑	→	→	↑	↓	↓	→	↓	↑	↓
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	→
←	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑

Where do we go now?

"Where do we go?
Oh-oh, where do we go now?
Oh, where do we go?
Where do we go now?"

(Guns N' Roses, Sweet Child of Mine)

Annotated bibliography

- Sutton and Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Still the best introduction to contemporary issues in RL. Sort of required reading. I have read about 90% of the book, and keep reading it and finding interesting stuff. *Do the exercises*, even if they sound silly. These guys really understand their stuff.
- Powell, *Reinforcement Learning and Stochastic Optimization - A unified framework for sequential decisions* Very wide ranging! Awesome reading with a very broad perspective of applications, theoretical, and computational issues. Its a mammoth book with almost 700 pages. I like to print random sections and read before sleeping.

Annotated bibliography

- Bertsekas, *A Course in Reinforcement Learning*, 2nd ed.
  Bertsekas is, in my opinion, the best author in the mathematical side of Dynamic Programming and Reinforcement Learning. His books "prove everything" and are beautiful. I have not read this latest book entirely, but I would probably choose it for a mathematically oriented course in RL.
- Bertsekas and Tsitsiklis, *Neuro-Dynamic Programming*.
  In the 90's, Bertsekas and Tsitsiklis had the idea of using artificial neural networks to estimate state-action value functions, as a way to get around the "curse of dimensionality". The was an idea waaaaaaaay ahead of its time, because at that time we didn't have the hardware or computer science resources to build these networks! A beautiful example of mathematicians, just with their brains, anticipating technology that would appear decades later.