Reinforcement Learning

Chapter 3: Model-free RL

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Classical RL Methods: *Recall*

Ultimate goal in an RL problem is to find the optimal policy

As mentioned, we have *two major challenges* in this way

1. *We need to compute values explicitly*
2. *We often deal with settings with huge state spaces?*

In this part of the course, we are going to handle the first challenge

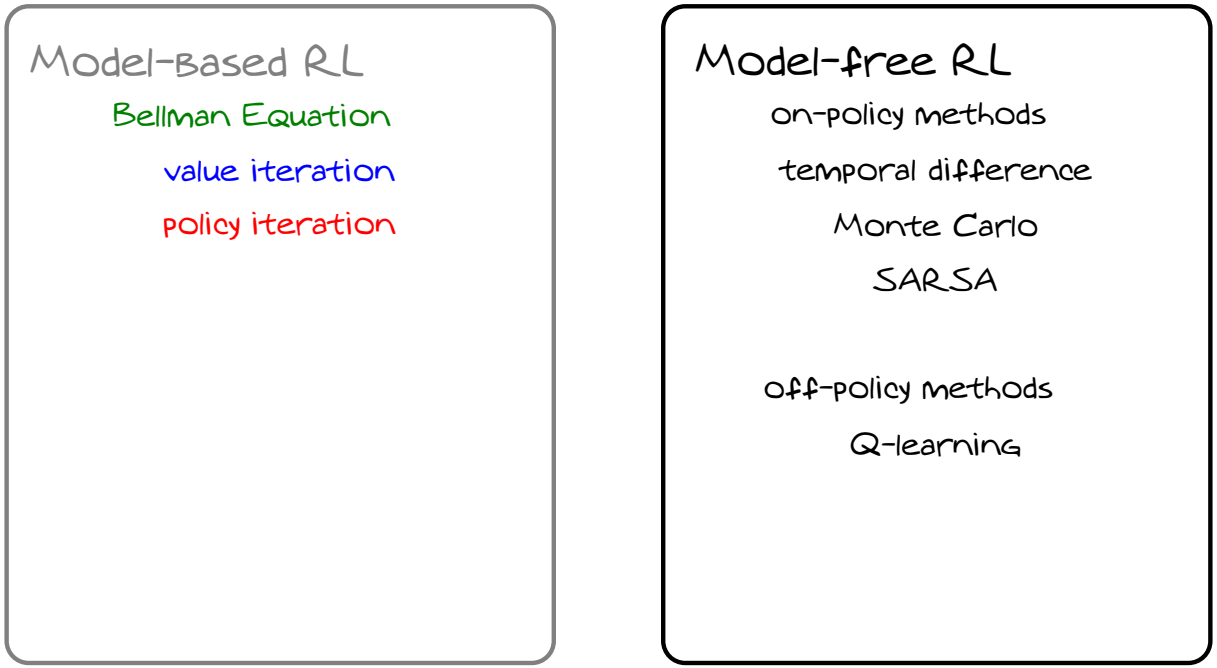
- *Previous chapter* $\rightsquigarrow$ *Model-based methods*
- *This chapter* $\rightsquigarrow$ *Model-free methods*

Finally We Got Serious: *Model-free RL*

In model-free methods

we do not have an analytic model for the behavior of environment

We intend to compute *values from real data collected from environment*



Model-free RL in *Nutshell*

+ *If this is the typical case in RL problems, why did we spend so much time on learning MDPs and finding optimal policy there?*
- Well! We need all those things, since *we are going to do the same thing here only without explicit model*

In a nutshell, we are going to find a way to apply

$$\textit{Generalized Policy Iteration} \equiv \textit{GPI}$$

But, now without knowing the transition-rewarding function

Let's take a look back at GPI

Generalized Policy Iteration

We wrote the pseudo-code for GPI as below

`GenPolicyItr()`:

1: *Initiate two random policies* π *and* $\bar{\pi}$
2: ***while*** $\pi \neq \bar{\pi}$ ***do***
3: $v_\pi = \texttt{GenPolicyEval}(\pi)$ *and* $\pi \leftarrow \bar{\pi}$
4: $\bar{\pi} = \texttt{PolicyImprov}(v_\pi)$
5: ***end while***

Let's recall where we had to use environment's model

1. *In policy evaluation phase when we compute values via Bellman equations*
2. *In policy improvement when we compute action-values out of values*

How can we do these tasks without knowing transition-rewarding model?

Computing Statistics from Data

Let's start with a very simple problem: *assume we have an unknown signal generator which returns signals at random; this generator is connected to a device and we can only see the output of this device, i.e., we see*

$$Y = f(X)$$

where X *is the random signal and* $f(\cdot)$ *denotes transform by the device*

We want to know the expected output of our device, i.e.,

$$\mu_Y = \mathbb{E}\{Y\} = \mathbb{E}\{f(X)\}$$

If we knew the model of the generator's model, we could write

$$\mu_Y = \mathbb{E}\{f(X)\} = \underbrace{\sum_{x \in \mathbb{X}}}_{\text{all outcomes}} f(x) \underbrace{p(x)}_{\text{model}}$$

Monte-Carlo Method

Now what can we do if we don't know the model

+ *Well! Shouldn't we evaluate it by a simple numerical simulation?*
– Exactly! This is what we call it *Monte-Carlo method*

In Monte-Carlo method, we sample our device K times independently as

$$Y_1, Y_2, \ldots, Y_K$$

Then we estimate the expected value as

$$\hat{\mu}_Y = \frac{1}{K} \sum_{k=1}^{K} Y_k$$

Monte-Carlo Method

+ *Why does Monte-Carlo work?*
- Simply because of central limit theorem

Since the sequence $Y_1, Y_2, \ldots, Y_K$ contains independent samples of identical process, we could say that

$$\hat{\mu}_Y \sim \mathcal{N}\left(\mu_Y, \frac{\sigma^2}{K}\right)$$

when K is large enough: so we could think of it as

$$\hat{\mu}_Y \approx \mu_Y + \frac{\varepsilon}{\sqrt{K}}$$

for some random error term ε: this error vanishes as K goes large

Computing Values via Monte-Carlo

+ *But, how can we apply this idea to RL? I don't see any connection!*
– Well! *Think of rewards and transitions as random signal and value function as device!* We only need to *take enough samples from the environment*

Let's start with a very simple task: we want to compute the value of state s for policy π in an episodic environment. Monte-Carlo suggest that

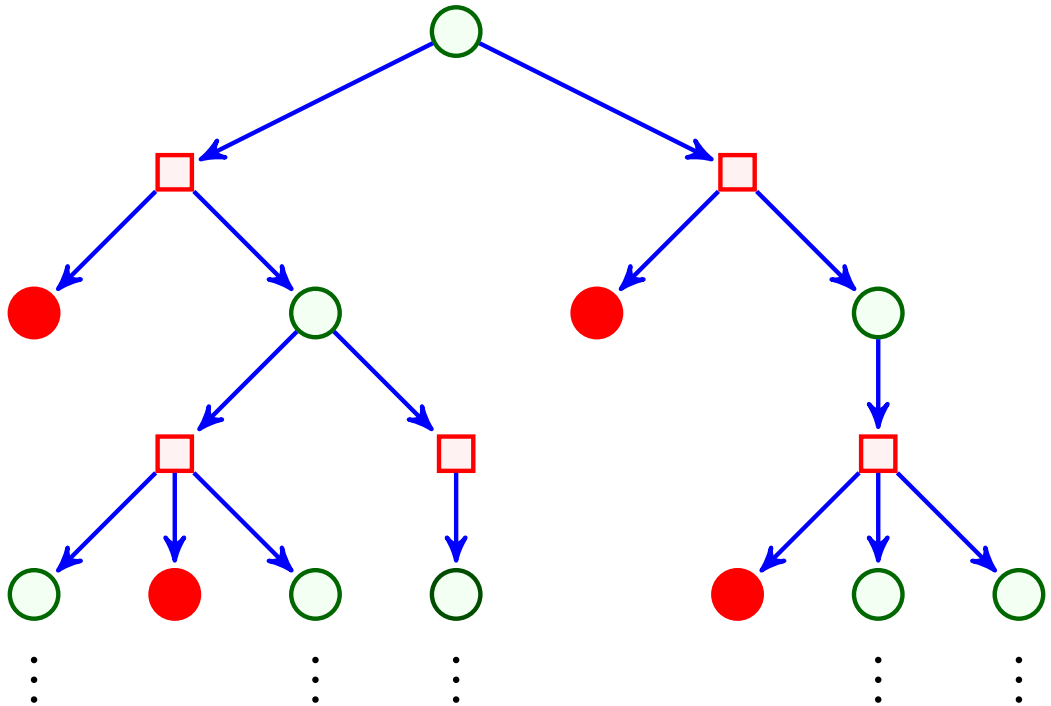
1. *We start at state s and play with policy π until we meet terminal state: say it happens at time T*
2. *We compute the sample return as $G[1] = R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T$*
3. *We repeat this for K episodes and each episode, we collect $G[k]$*

Then, we could estimate the value of state s as

$$\hat{v}_\pi(s) = \frac{1}{K} \sum_{k=1}^{K} G[k]$$

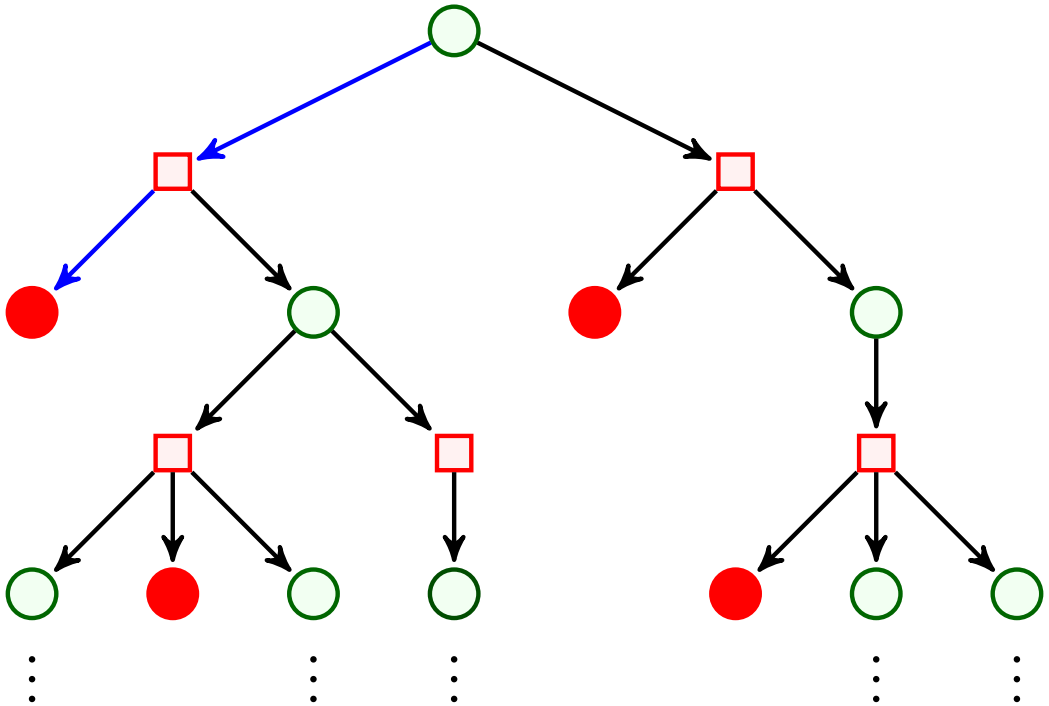
Values via Monte-Carlo: *Trajectory Sampling*

We can look at this approach as estimating values from sample trajectories: with known model, we can compute values by averaging over possible trajectories



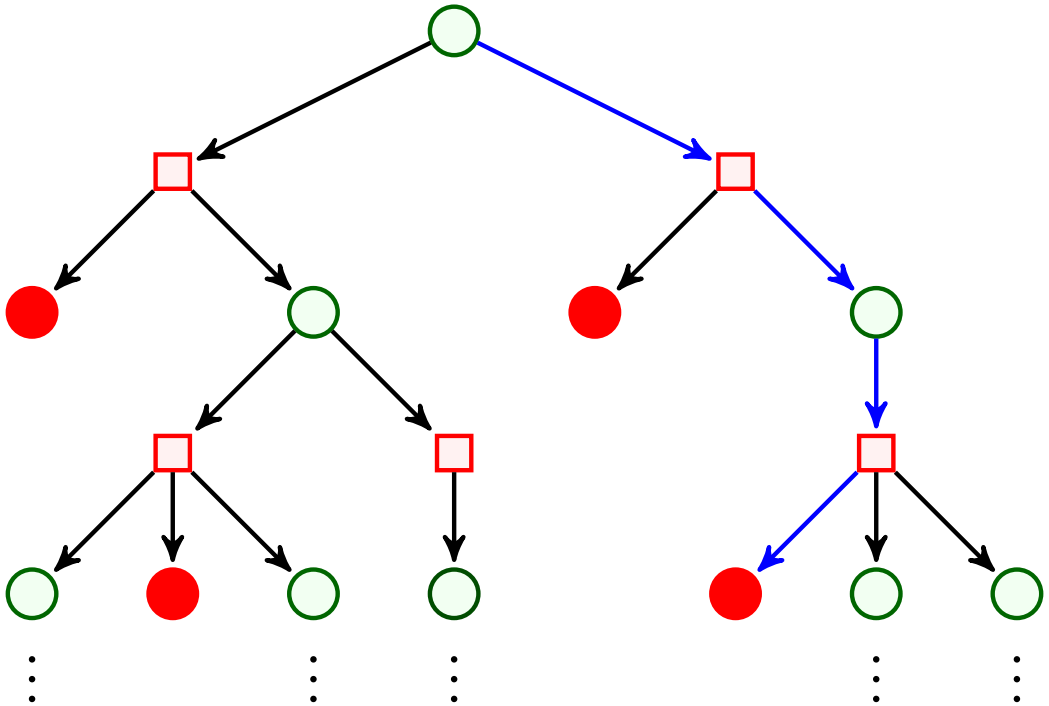
Values via Monte-Carlo: *Trajectory Sampling*

We can look at this approach as estimating values from sample trajectories: without known model, we can sample them and estimate values from them



Values via Monte-Carlo: *Trajectory Sampling*

We can look at this approach as estimating values from sample trajectories: without known model, we can sample them and estimate values from them



Computing Values via Monte-Carlo: *Algorithm I*

Let's put our estimation approach into an algorithm

`MC_verI(`π, s`):`

1: *Initiate estimator of value as* $\hat{v}_\pi(s) = 0$
2: ***for*** *episode* $= 1 : K$ ***do***
3: *Initiate with state* $S_0 = s$ *and act via policy* $\pi(a|s)$
4: *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T \text{: terminal}$$

5: *Compute sample return* $G = R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T$
6: *Update estimate of value as* $\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + G/K$
7: ***end for***

Computing Values via Monte-Carlo

+ *But, doesn't that take too long to compute a single value?*
– Yes! This is in general a problem; however, in our naive algorithm it is too much delayed!

In our algorithm, we need to wait till very end of K episodes to access an estimate, but we rather prefer to have a bad estimate which gradually improves over episodes

We could use the idea of online averaging $\equiv$ incremental averaging

Let's find out what it is!

Online Averaging

Say, we want to compute the average of K samples: *we could write*

$$\eta_K = \frac{1}{K}\sum_{k=1}^{K} G_k = \frac{1}{K}\left(\sum_{k=1}^{K-1} G_k + G_K\right) = \frac{1}{K}\left((K-1)\,\eta_{K-1} + G_K\right)$$

$$= \left(1 - \frac{1}{K}\right)\eta_{K-1} + \frac{G_K}{K}$$

$$= \eta_{K-1} + \frac{1}{K}\left(G_K - \eta_{K-1}\right)$$

But, we can define the previous average as

$$\eta_{K-1} = \frac{1}{K-1}\sum_{k=1}^{K-1} G_k \rightsquigarrow \sum_{k=1}^{K-1} G_k = (K-1)\,\eta_{K-1}$$

Online Averaging: *Geometric Weights*

Online Averaging

We can update the average in online fashion as

$$\eta_K = \eta_{K-1} + \frac{1}{K}\Delta_K$$

where $\Delta_K = G_K - \eta_{K-1}$ *is the deviation in* K*-th episode*

The above expression is given for *uniform averaging weights*, *i.e., all samples have same weights*: *in more general form, we usually update*

$$\eta_K = \eta_{K-1} + \alpha\Delta_K$$

for some $0 < \alpha \leqslant 1$ *that can be fixed or scaled with* K

- *if it is fixed* $\equiv$ *computing weighted average with geometric weights*
- *if it is scaled linearly with* $K \equiv$ *computing linear averaging*

Computing Values via Monte-Carlo: *Algorithm II*

Let's modify our earlier algorithm with online averaging

`MC_verII(`π, s`):`

1: *Initiate estimator of value as* $\hat{v}_\pi(s) = 0$
2: ***for*** *episode* $= 1 : K$ ***do***
3: *Initiate with state* $S_0 = s$ *and act via policy* $\pi(a|s)$
4: *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T\text{: terminal}$$

5: *Compute sample return* $G = R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T$
6: *Update estimate of value as* $\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha(G - \hat{v}_\pi(s))$
7: ***end for***

Now after each episode, we have an estimate of value function at state s

Computing Values via Monte-Carlo: *Improve Efficiency*

In our algorithm: *we go through the whole trajectory to compute the value on the state we started with! This does not sound sample efficient!*

+ *Well! What can we do more?! It seems to be the case!*
– Not really! We can estimate values of other states down the trajectory!

In the following sample trajectory

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

it's not only S_0 whose sample return can be computed! We can also compute sample returns of $S_1, \ldots, S_{T-1}$

All-Visit Monte-Carlo: *Algorithm III*

This concludes a policy evaluation algorithm based on Monte-Carlo

`MC_Eval(`π`):`

Initiate estimator of value as $\hat{v}_\pi\left(s^n\right) = 0$ *for* $n = 1 : N$
for *episode* $= 1 : K$ ***do***
    Initiate with a random state S_0 *and act via policy* $\pi\left(a|s\right)$
    Sample a trajectory

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T\text{: terminal}$$

    Initiate with $G = 0$
    for $t = T - 1 : 0$ ***do***
        Update current return $G \leftarrow R_{t+1} + \gamma G$
        Update estimate of value as $\hat{v}_\pi\left(S_t\right) \leftarrow \hat{v}_\pi\left(S_t\right) + \alpha(G - \hat{v}_\pi\left(S_t\right))$
    end for
end for

All-Visit Monte-Carlo: *Convergence*

It's intuitive to say this algorithm converges to true values after lots of episodes

Asymptotic Convergence of Monte-Carlo

Let $\mathcal{C}_K(s)$ denote number of visits at state s during K Monte-Carlo episodes. Assume that the random state initialization is distributed such that $\mathcal{C}_K(s^n)$ grows large as K increases for $n = 1 : N$, i.e.,

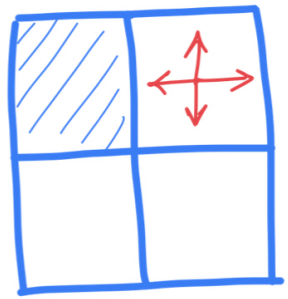
$$\lim_{K \to \infty} \mathcal{C}_K(s^n) = \infty$$

Then, as $K \to \infty$ the estimator of value function converges to its exact expression, i.e.,

$$\hat{v}_\pi(s) \xrightarrow{K \uparrow \infty} v_\pi(s)$$

for any state s

Example: *Dummy Grid World with Random Walk*



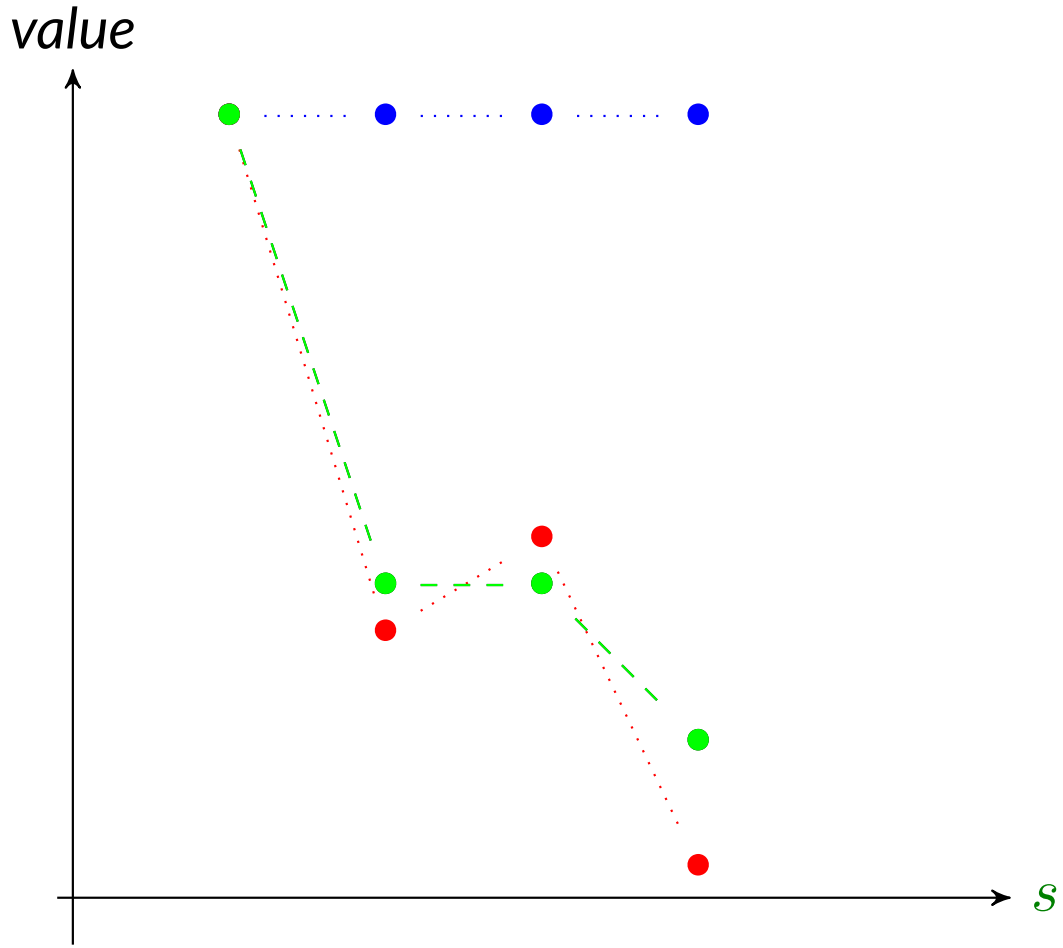
Let's get back to our dummy world: we now use Monte-Carlo method to compute the values for uniform random policy, i.e.,

$$\pi\left(a|s\right) = \frac{1}{4}$$

for all actions and states. From Bellman equations, we have

$$v_\pi(0) = 1 \qquad v_\pi(1) = -4.5 \qquad v_\pi(2) = -4.5 \qquad v_\pi(3) = -6$$

Example: *Dummy Grid World with Random Walk*



Typical Behavior: *Variation Against Number of Episodes*

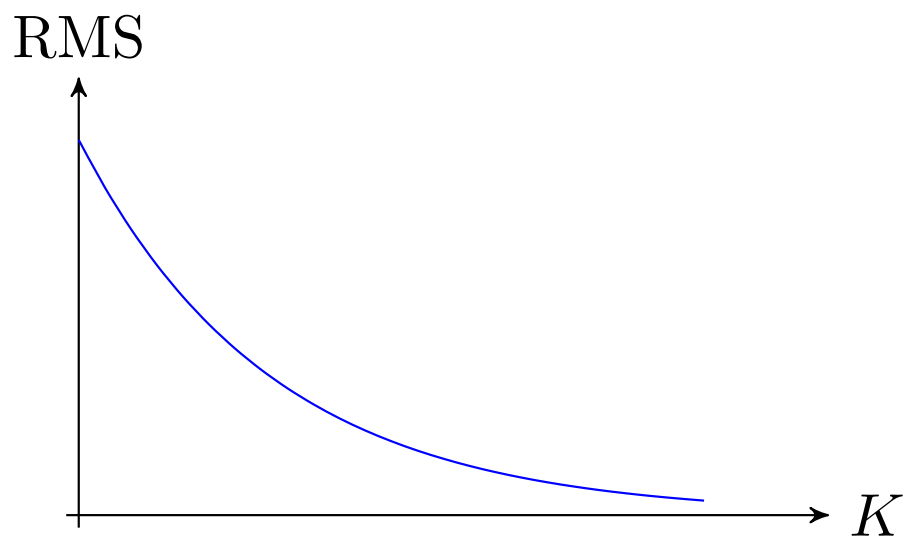
We can compute the error of our estimation in each episode

$$\text{RMS} = \sqrt{\sum_{n=1}^{N} \left|\hat{v}_\pi\left(s^n\right) - v_\pi\left(s^n\right)\right|^2}$$

if we know the true value function, e.g., our random walk example

---

If we plot it against K; then, we see



Typical Behavior: *Variation Against Averaging Coefficient*

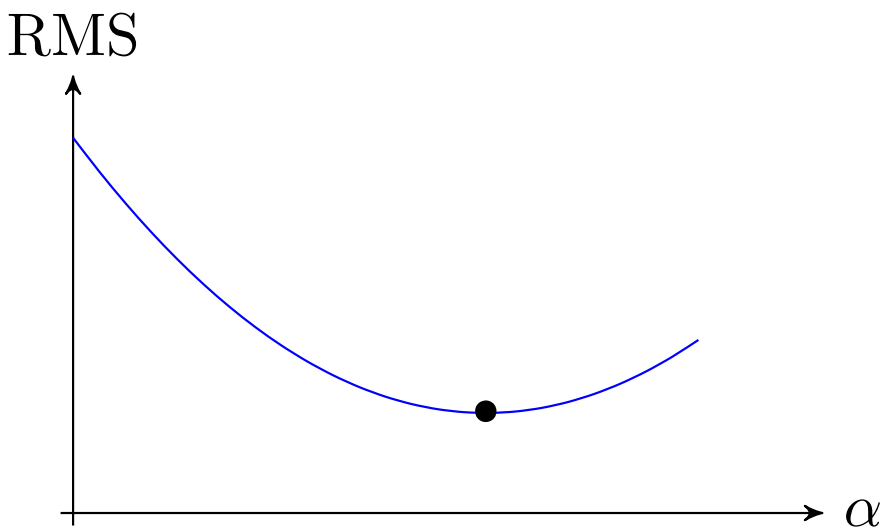
We can compute the error of our estimation in each episode

$$\text{RMS} = \sqrt{\sum_{n=1}^{N} \left|\hat{v}_\pi\left(s^n\right) - v_\pi\left(s^n\right)\right|^2}$$

if we know the true value function, e.g., our random walk example

---

If we plot it against α; then, we could see a minimum



Monte-Carlo Method: *Action-Values*

+ *Now that we have Monte-Carlo algorithm, can we use it in GPI?*
– Not yet! Remember that *we need action-values for policy improvement*

In GPI, we used to use Bellman equation for this

$$
\begin{aligned}
q_\pi(s,a) &= \bar{\mathcal{R}}(s,a) + \gamma \mathbb{E}\left\{v_\pi\left(\bar{S}\right)|s,a\right\} \\
&= \mathbb{E}\left\{R_{t+1}|S_t = s, A_t = a\right\} + \gamma \mathbb{E}\left\{v_\pi\left(S_{t+1}\right)|S_t = s, A_t = a\right\} \\
&= \sum_{\ell=1}^{L}\sum_{n=1}^{N}\left(r^\ell + \gamma v_\pi\left(s^n\right)\right) \underbrace{p(r^\ell, s^n|s,a)}_{\text{transition-rewarding model}}
\end{aligned}
$$

But, now we cannot use it anymore!

Maybe, we ca use Monte-Carlo method to estimate action-values directly

All-Visit Monte-Carlo: *Action-Values*

`MC_QEval(`π`):`

1: *Initiate estimator as* $\hat{q}_\pi\left(s^n, a^m\right) = 0$ *for* $n = 1 : N$ *and* $m = 1 : M$
2: ***for*** *episode* $= 1 : K$ ***do***
3: *Initiate with a random state-action pair* (S_0, A_0) *and act via policy* $\pi\left(a|s\right)$
4: *Sample a trajectory*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T\text{: terminal}$$

5: *Initiate with* $G = 0$
6: ***for*** $t = T - 1 : 0$ ***do***
7: *Update current return* $G \leftarrow R_{t+1} + \gamma G$
8: *Update* $\hat{q}_\pi\left(S_t, A_t\right) \leftarrow \hat{q}_\pi\left(S_t, A_t\right) + \alpha(G - \hat{q}_\pi\left(S_t, A_t\right))$
9: ***end for***
10: ***end for***

We can now apply GPI using the Monte-Carlo method!

Policy Iteration with *Monte-Carlo*

We can use Monte-Carlo method to compute the action-values

- *We then improve in each iteration by selecting best action for each state*
  - ↳ *This is what we typically call greedy improvement*

```
MC_PolicyItr():
1: Initiate two random policies π and π̄
2: while π ≠ π̄ do
3:     q̂_π = MC_QEval(π) and π ← π̄
4:     π̄ = Greedy(q̂_π)
5: end while
```

Policy Iteration with *Monte-Carlo*

Algorithmically, we can write the greedy update as

`Greedy`$(\hat{q}_\pi)$**:**

1: ***for*** $n = 1 : N$ ***do***

2: *Improve the by taking deterministically the best action*

$$\bar{\pi}\left(a^m | s^n\right) = \begin{cases} 1 & m = \underset{m}{\operatorname{argmax}}\, \hat{q}_\pi\left(s^n, a^m\right) \\ 0 & m \neq \underset{m}{\operatorname{argmax}}\, \hat{q}_\pi\left(s^n, a^m\right) \end{cases}$$

3: ***end for***

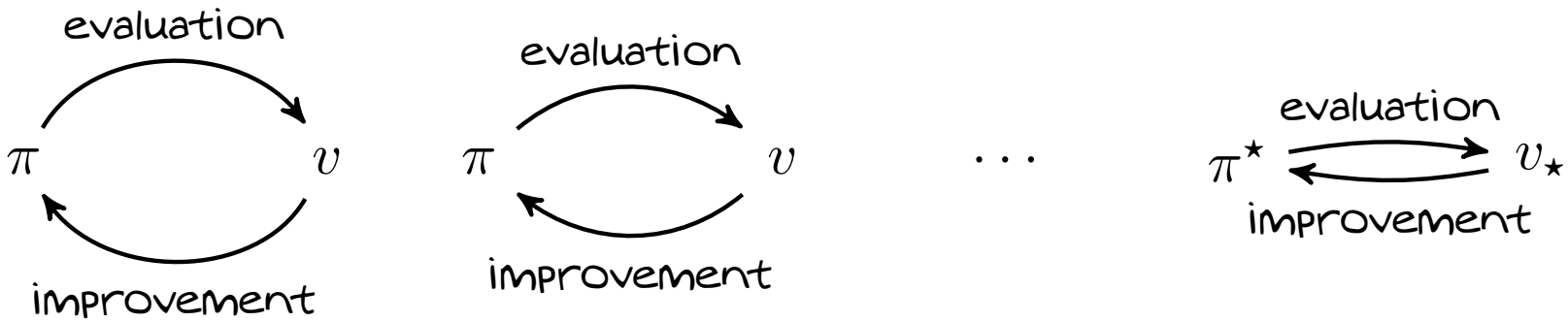
This is however not the best we could do!

We are going to have a whole lecture about it

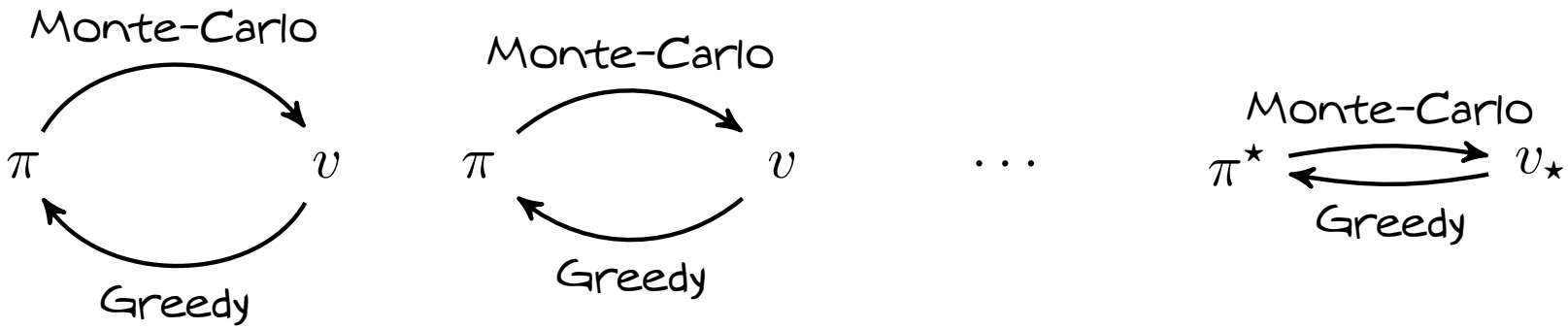
Stay tuned! We get back to this point in Section 4

GPI with *Monte-Carlo*

For any GPI, we said that we can think of

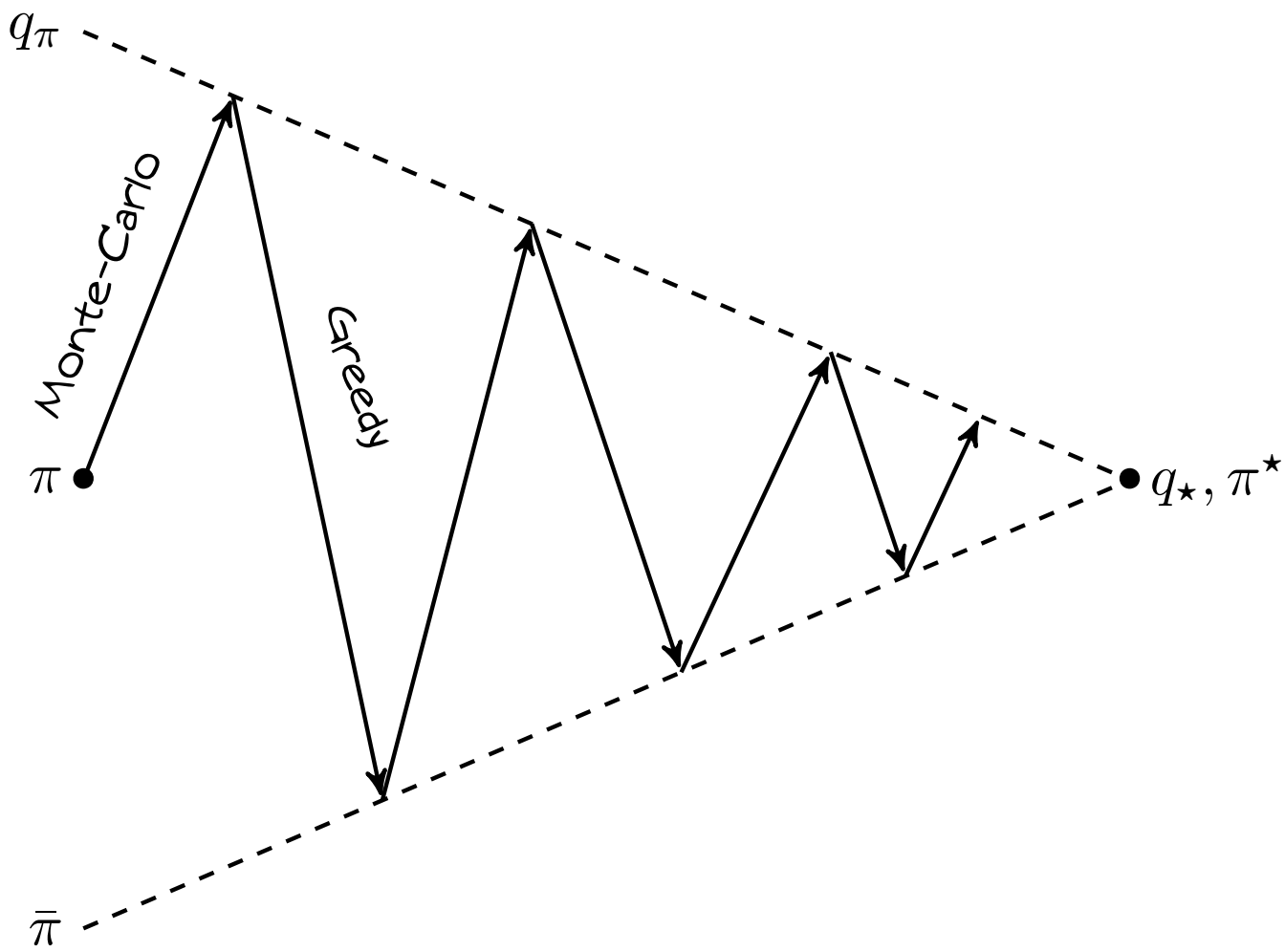


With Monte-Carlo evaluation, we can show this procedure as



GPI with *Monte-Carlo*

Another way to visualize this procedure is to think of following diagram



Non-Episodic Monte-Carlo: *Terminating Trajectory*

+ *We only discussed episodic scenarios! Don't we use model-free RL in non-episodic environment?*
– Sure we do! But, Monte-Carlo is not the best approach

A basic idea in this case is to terminate sample trajectories

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

- *With long enough T and $\gamma < 1$ the very later terms are ineffective*
- *But, we cannot use all the states in the trajectories*
  ↳ *Sample returns of those who are close to time T are not reliable!*

$$G_{T-1} = R_T + \underbrace{\gamma R_{T+1} + \cdots}_{\text{we terminated them!}}$$

Terminating Monte-Carlo

`TerminMC_Eval(`π`):`

Initiate estimator of value as $\hat{v}_\pi\left(s^n\right) = 0$ *for* $n = 1 : N$
Choose very large T *and* W *that satisfy* $W < T$
for *episode* $= 1 : K$ ***do***
    Initiate with a random state S_0 *and act via policy* $\pi\left(a|s\right)$
    Sample a trajectory and terminate after T *time steps*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T\text{: terminated}$$

    Initiate with $G = 0$
    for $t = T - 1 : 0$ ***do***
        Update current return $G \leftarrow R_{t+1} + \gamma G$
        if $t < T - W$ ***then***
            Update estimate of value as $\hat{v}_\pi\left(S_t\right) \leftarrow \hat{v}_\pi\left(S_t\right) + \alpha(G - \hat{v}_\pi\left(S_t\right))$
        end if
    end for
end for