ECE 1508: Reinforcement Learning

Chapter 3: Model-free RL

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Summer 2024

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-Based RL
- Bellman Equation
- value iteration
- policy iteration

Model-free RL
- on-policy methods
  - temporal difference
  - Monte Carlo
  - SARSA
- off-policy methods
  - Q-learning

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

$$\boxed{\textit{Let's take a look back at GPI}}$$

Generalized Policy Iteration

We wrote the pseudo-code for GPI as below

```
GenPolicyItr():
1: Initiate two random policies $\pi$ 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*

$$Y = f\left(X\right)$$

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

---

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

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

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

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

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

$$Y_1, Y_2, \dots, Y_K$$

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 $Y_1, Y_2, \dots, Y_K$ 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

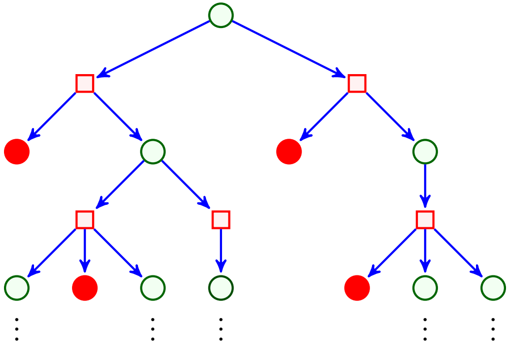
1. *We start at state s and play with policy π until we meet terminal state: say it happens at time T*
2. *We compute the sample return as $G[1] = R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T$*
3. *We repeat this for K episodes and each episode, we collect $G[k]$*

Then, we could estimate the value of state s as

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

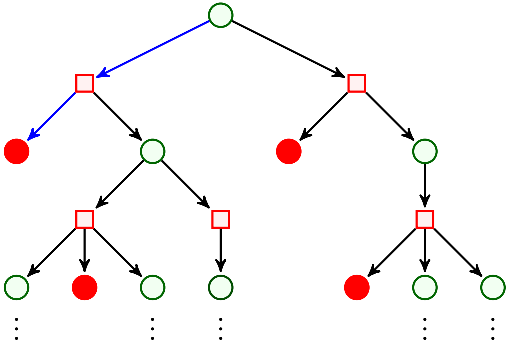
Values via Monte-Carlo: *Trajectory Sampling*

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



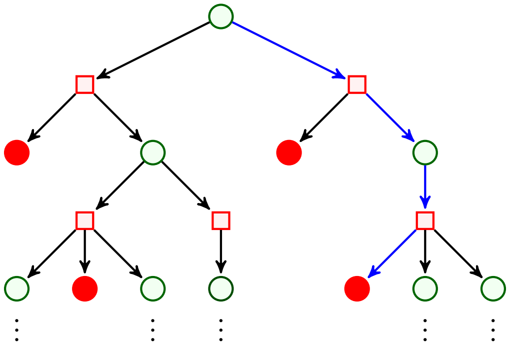
Values via Monte-Carlo: *Trajectory Sampling*

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($\pi$, $s$):
 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

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

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

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

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

But, we can define the previous average as

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

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($\pi, s$):
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
           $S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$: 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 conclude a policy evaluation algorithm based on Monte-Carlo

```
MC_Eval($\pi$):
 1: Initiate estimator of value as $\hat{v}_\pi(s^n) = 0$ for $n = 1 : N$
 2: for episode $= 1 : K$ do
 3:     Initiate with a random state $S_0$ and act via policy $\pi(a|s)$
 4:     Sample a trajectory
            $S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$: 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 estimate of value as $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha(G - \hat{v}_\pi(S_t))$
 9:     end for
10: 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.,

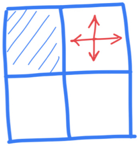
$$\lim_{K \to \infty} \mathcal{C}_K(s^n) = \infty$$

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*



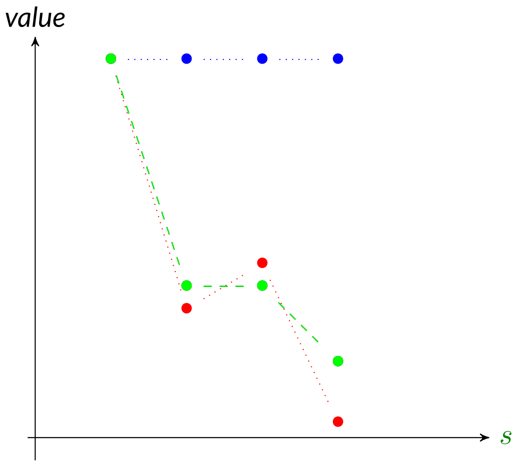
Let's get back to our dummy world: we now use Monte-Carlo method to compute the values for uniform random policy, i.e.,

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

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*

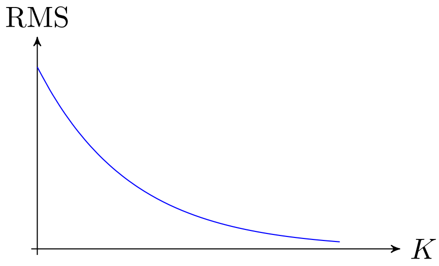
We can compute the error of our estimation in each episode

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

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*

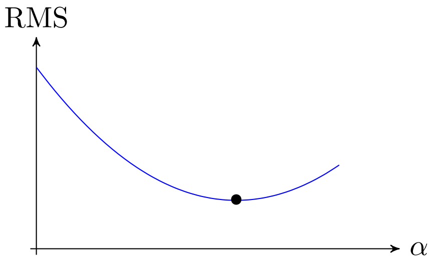
We can compute the error of our estimation in each episode

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

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

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

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($\pi$):
1:  Initiate estimator as $\hat{q}_\pi(s^n, a^m) = 0$ 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 $\pi(a|s)$
4:      Sample a trajectory

        $S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$: 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 $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
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 $\pi$ and $\bar{\pi}$
2: while $\pi \neq \bar{\pi}$ do
3:     $\hat{q}_\pi$ = MC_QEval($\pi$) and $\pi \leftarrow \bar{\pi}$
4:     $\bar{\pi}$ = Greedy($\hat{q}_\pi$)
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
           $$\bar{\pi}(a^m | s^n) = \begin{cases} 1 & m = \operatorname*{argmax}_m \hat{q}_\pi(s^n, a^m) \\ 0 & m \neq \operatorname*{argmax}_m \hat{q}_\pi(s^n, a^m) \end{cases}$$
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

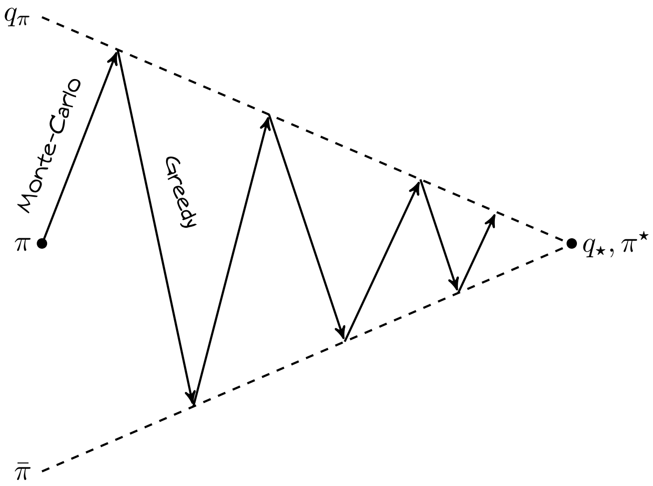
$$\begin{tikzcd}
\pi \arrow[r, bend left, "\text{evaluation}"] & v \arrow[l, bend left, "\text{improvement}"] & \pi \arrow[r, bend left, "\text{evaluation}"] & v \arrow[l, bend left, "\text{improvement}"] & \cdots & \pi^\star \arrow[r, bend left, "\text{evaluation}"] & v_\star \arrow[l, bend left, "\text{improvement}"]
\end{tikzcd}$$

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

$$\begin{tikzcd}
\pi \arrow[r, bend left, "\text{Monte-Carlo}"] & v \arrow[l, bend left, "\text{Greedy}"] & \pi \arrow[r, bend left, "\text{Monte-Carlo}"] & v \arrow[l, bend left, "\text{Greedy}"] & \cdots & \pi^\star \arrow[r, bend left, "\text{Monte-Carlo}"] & v_\star \arrow[l, bend left, "\text{Greedy}"]
\end{tikzcd}$$

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

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

 6:     Initiate with $G = 0$
 7:     for $t = T - 1 : 0$ do
 8:         Update current return $G \leftarrow R_{t+1} + \gamma G$
 9:         if $t < T - W$ then
10:             Update estimate of value as $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha(G - \hat{v}_\pi(S_t))$
11:         end if
12:     end for
13: end for
```

Terminating Monte-Carlo: *Sample Inefficiency*

+ *But, isn't that as you said before sample inefficient?*
– Sure it is!

$$\boxed{\textit{We are loosing lots of states in each sample trajectory!}}$$

A better solution is to use the recursive property of return and do

bootstrapping

This is what we see next!

Testing *Genie in the Box*

Let's think again a bit science-fictional: *assume a genie can tell us the value $v_\pi(s)$ for each state s, We want to test this genie via a numerical algorithm* ☺

Bellman tells us that at any state s, we should see

$$v_\pi(s) = \mathbb{E}\{R_{t+1} | S_t = s\} + \gamma \mathbb{E}\{v_\pi(S_{t+1}) | S_t = s\}$$

Monte-Carlo tells us further that after K sample trajectories $S_0, A_0 \xrightarrow{R_1} S_1$ initiated at $S_0 = s$ and terminated after only one step, we have

$$\mathbb{E}\{R_{t+1} | S_t = s\} \approx \frac{1}{K} \sum_{k=1}^{K} R_1[k]$$

$$\mathbb{E}\{v_\pi(S_{t+1}) | S_t = s\} \approx \frac{1}{K} \sum_{k=1}^{K} v_\pi(S_1[k])$$

Testing *Genie in the Box*

We could hence find an estimator of $v_\pi(s)$ as

$$v_\pi(s) \approx \hat{v}_\pi(s) = \frac{1}{K}\sum_{k=1}^{K} R_1[k] + \gamma v_\pi(S_1[k])$$

And, of course we could simply evaluate this estimator online as

$$\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \frac{1}{K}(R_1 + \gamma v_\pi(S_1) - \hat{v}_\pi(s))$$

if we need linear averaging or alternatively as

$$\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha(R_1 + \gamma v_\pi(S_1) - \hat{v}_\pi(s))$$

if we think of more general weighted averaging

Computing Values via Bootstrapping: *Algorithm I*

We can write our genie-testing algorithm as

```
TD_verI($\pi$, $s$):
1: Initiate estimator of value as $\hat{v}_\pi(s) = 0$
2: Ask genie $v_\pi(\bar{s})$ for all $\bar{s}$ that can be followed after $s$
3: for episode $= 1 : K$ do
4:     Initiate with state $S_0 = s$ and act via policy $\pi(a|s)$
5:     Sample a single-step terminated trajectory
           $S_0, A_0 \xrightarrow{R_1} S_1$
6:     Update estimate of value as $\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha(R_1 + v_\pi(S_1) - \hat{v}_\pi(s))$
7: end for
```

Note that in this algorithm we don't need the environment to be episodic, as we use recursive property of value-function!

This explains the idea of bootstrapping

Bootstrapping: *Using Value Estimates*

+ *But, in practice we don't have genie! And, say we find one, why should we compute values anymore?!*
- Absolutely! But, we may use this property

We can replace those true values with their estimates

- *They are initially bad estimates*
  - ↳ *and thus return in bad estimate for the other state*
- *They gradually improve*
  - ↳ *and therefore return better estimate for the other state*

The key point is that we get rid of need for a terminal state!

Computing Values via Bootstrapping: *Algorithm II*

So, we could get rid of the genie finally

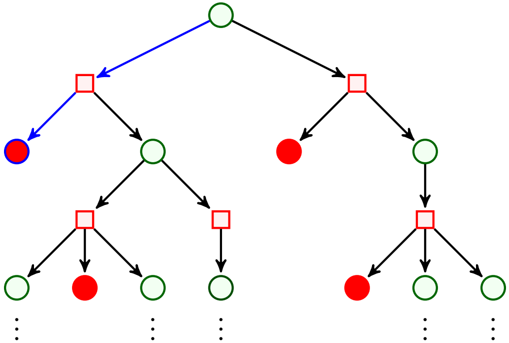
```
TD_verII($\pi, s$):
 1: Initiate estimator of value as $\hat{v}_\pi(s) = 0$
 2: Use available $\hat{v}_\pi(\bar{s})$ for all $\bar{s}$ that follow $s$
 3: for episode $= 1 : K$ do
 4:     Initiate with state $S_0 = s$ and act via policy $\pi(a|s)$
 5:     Sample a single-step terminated trajectory

            $S_0, A_0 \xrightarrow{R_1} S_1$

 6:     Update estimate of value as $\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha(R_1 + \hat{v}_\pi(S_1) - \hat{v}_\pi(s))$
 7: end for
```

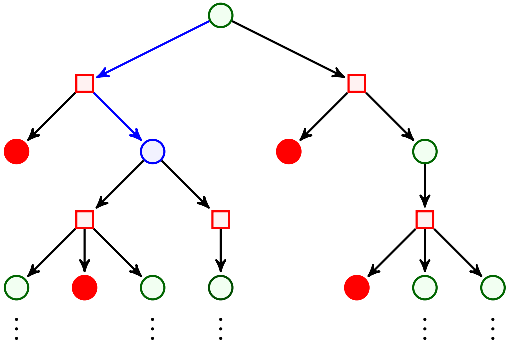
Backup Diagram: *Sampling with Bootstrapping*

Looking at the backup tree, we are now sampling one-step trajectories



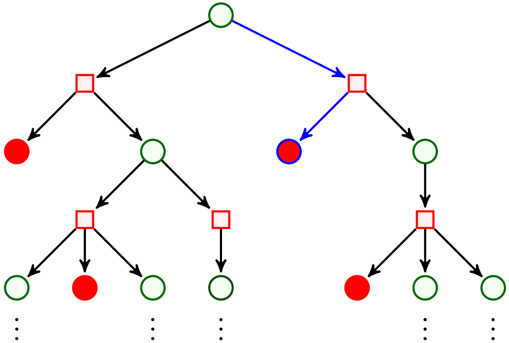
Backup Diagram: *Sampling with Bootstrapping*

Looking at the backup tree, we are now sampling one-step trajectories



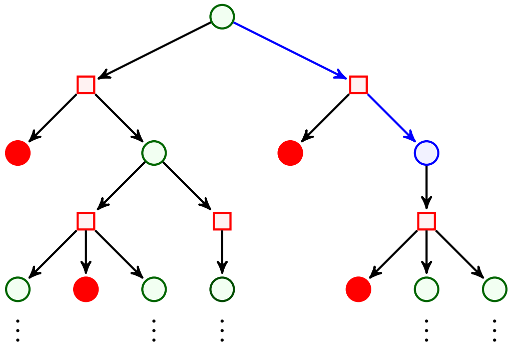
Backup Diagram: *Sampling with Bootstrapping*

Looking at the backup tree, we are now sampling one-step trajectories



Backup Diagram: *Sampling with Bootstrapping*

Looking at the backup tree, we are now sampling one-step trajectories



Temporal Difference

Bootstrapping can replace Monte-Carlo in our evaluation algorithms

1. *We start with some initial value estimates*
2. *We sample a trajectory of finite length T*

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

$\hookrightarrow$ *it can either end with a terminal state if we have any*
$\hookrightarrow$ *or could simply be terminated*

3. *We move over trajectory and update value of each state by bootstrapping*

This idea of estimating values is called

Temporal Difference $\equiv$ *TD*

Temporal Difference: *TD*-0

```
TD_Eval($\pi$):
1: Initiate estimator of value as $\hat{v}_\pi(s^n) = 0$ for $n = 1 : N$
2: for episode $= 1 : K$ do
3:     Initiate with a random state $S_0$ and act via policy $\pi(a|s)$
4:     Sample a trajectory until either a terminal stated or some terminating $T$

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

5:     for $t = 0 : T-1$ do
6:         Update as $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha(R_{t+1} + \gamma\hat{v}_\pi(S_{t+1}) - \hat{v}_\pi(S_t))$
7:     end for
8: end for
```

Attention

With TD, we even don't need to wait till a trajectory is sampled!

Evaluating Action-Values via Bootstrapping

+ *What about the action-values? Can we bootstrap again?*
- Sure!

Recall Bellman equation I of action-value function

$$q_\pi(s,a) = \mathbb{E}\left\{R_{t+1} | S_t = s, A_t = a\right\} + \gamma \mathbb{E}\left\{v_\pi(S_{t+1}) | S_t = s,, A_t = a\right\}$$

So, we can use sample trajectory to estimate action-values as well: at time t, we can update estimate of pair (S_t, A_t) as

$$\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}) - \hat{q}_\pi(S_t, A_t)\right)$$

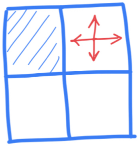
Temporal Difference: *Action-Value*

```
TD_QEval($\pi$):
1: Initiate estimator of value as $\hat{q}_\pi(s^n, a^m) = 0$ 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 $\pi(a|s)$
4:     Sample a trajectory until either a terminal stated or some terminating $T$

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

5:     for $t = 0 : T-1$ do
6:         Set $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}) - \hat{q}_\pi(S_t, A_t))$
7:     end for
8: end for
```

Example: *Dummy Grid World with Random Walk*



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

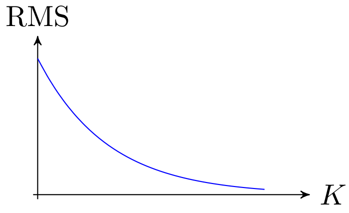
$$\pi\left(a|s\right) = \frac{1}{4}$$

for all actions and states

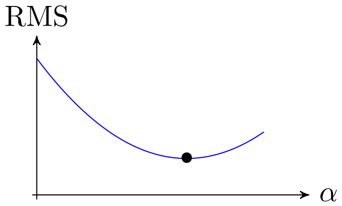
Try it at home ☺

Typical Behavior

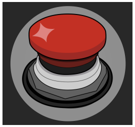
We are going to see the same behavior also with TD: *against K we see*



and against α we have a minimum



Example: *TD vs Monte-Carlo in Single-Button Game*



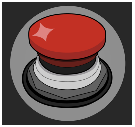
Consider the following dummy game: *we have got a single button to push; each time we push this button,*

- *we either get into green or Blue mode allowing us to push the button again and returns a 0/1 reward*
- *or it gets into red mode which only returns a 0/1 reward and game is over*

Obviously this game has

- *Three states: green, Blue and red which is terminal*
- *a single action, i.e., pushing the button*

Example: *TD vs Monte-Carlo in Single-Button Game*



We play this game 6 episodes and get following sample trajectories

$$\text{Blue} \xrightarrow{1} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Green} \xrightarrow{0} \text{Blue} \xrightarrow{0} \text{red}$$

Let's estimate $v(\text{Blue})$ and $v(\text{Green})$ by both TD-0 and Monte-Carlo

Example: *TD vs Monte-Carlo in Single-Button Game*

$$\text{Blue} \xrightarrow{1} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Green} \xrightarrow{0} \text{Blue} \xrightarrow{0} \text{red}$$

With Monte-Carlo approach, we could say: we have 8 sample trajectories starting with Blue *with 4 returning 1 and 4 returning 0; thus we have*

$$\hat{v}\left(\text{Blue}\right) = \frac{4}{8} = 0.5$$

We also have only one sample trajectory starting at Green *with zero return; thus, we have*

$$\hat{v}\left(\text{Green}\right) = \frac{0}{1} = 0$$

Example: *TD vs Monte-Carlo in Single-Button Game*

$$\text{Blue} \xrightarrow{1} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Green} \xrightarrow{0} \text{Blue} \xrightarrow{0} \text{red}$$

With TD-0, we could say: we have 8 Blue states followed by either Blue or red. If we bootstrap, we then get up to state Blue in the last trajectory

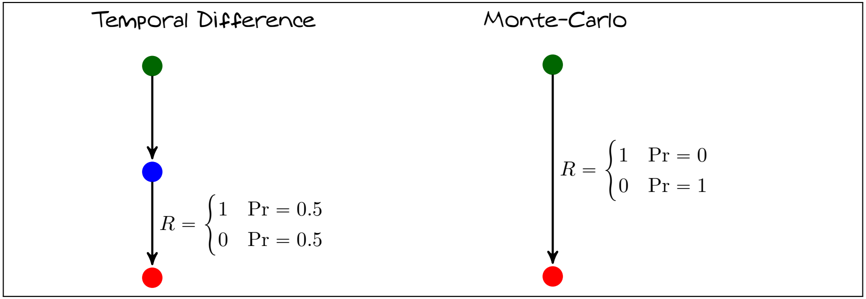
$$\hat{v}\left(\text{Blue}\right) \approx 0.5$$

We then get to the last trajectory which is the only one with state Green: since we have not yet updated, we yet have $\hat{v}\left(\text{Green}\right) = 0$; by bootstrapping we get

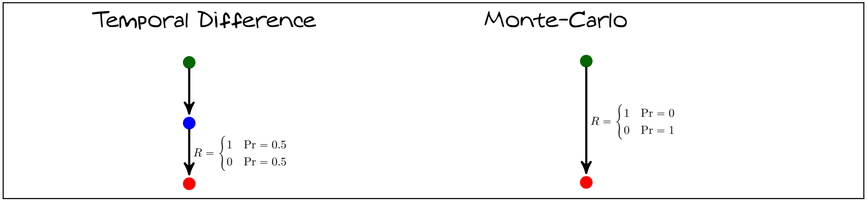
$$\hat{v}\left(\text{Green}\right) \leftarrow \underbrace{\hat{v}\left(\text{Green}\right)}_{0} + \left(\underbrace{R_{t+1}}_{0} + \underbrace{\hat{v}\left(S_{t+1}\right)}_{\hat{v}(\text{Blue})\approx 0.5} - \underbrace{\hat{v}\left(\text{Green}\right)}_{0} \right) \approx 0.5$$

Temporal Difference vs Monte-Carlo: *Note I*

The observed difference follow a fundamental point: *in Monte-Carlo we only look at best approximation given data, without looking into the Markovity of the state, whereas in TD we take into account the fact that we are dealing with a Markov state*


Temporal Difference
$R = \begin{cases} 1 & \Pr = 0.5 \\ 0 & \Pr = 0.5 \end{cases}$
Monte-Carlo
$R = \begin{cases} 1 & \Pr = 0 \\ 0 & \Pr = 1 \end{cases}$

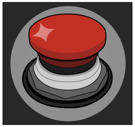
Temporal Difference vs Monte-Carlo: *Note I*


Temporal Difference
$R = \begin{cases} 1 & \Pr = 0.5 \\ 0 & \Pr = 0.5 \end{cases}$
Monte-Carlo
$R = \begin{cases} 1 & \Pr = 0 \\ 0 & \Pr = 1 \end{cases}$

This is common to see in the literature that people say

- *TD finds maximum-likelihood estimate of the values*
  - ↳ *It uses this assumption that the state is a Markov process*
  - ↳ *It's the better option, if we are sure that we have access to the complete (Markov) state*
- *Monte-Carlo finds least-squares estimate of the values*
  - ↳ *It ignores Markovity of the state*
  - ↳ *Maybe better option, when we cannot access the complete (Markov) state*

Back to Single-Button Game: *Side Note on Batch Updating*



What would happen, if we get sample trajectories in the following order

$$\text{Green} \xrightarrow{0} \text{Blue} \xrightarrow{0} \text{red}$$

$$\text{Blue} \xrightarrow{1} \text{Blue} \xrightarrow{0} \text{red}$$

$$\text{Blue} \xrightarrow{1} \text{red}$$

$$\text{Blue} \xrightarrow{0} \text{Blue} \xrightarrow{1} \text{red}$$

$$\text{Blue} \xrightarrow{1} \text{red}$$

$$\text{Blue} \xrightarrow{0} \text{red}$$

Back to Single-Button Game: *Side Note on Batch Updating*

$$\text{Green} \xrightarrow{0} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{red}$$

If we go only once over the batch of all episodes with TD, we get

$$\hat{v}\left(\text{Green}\right) \approx 0$$
$$\hat{v}\left(\text{Blue}\right) \approx 0.5$$

Back to Single-Button Game: *Side Note on Batch Updating*

$$\text{Green} \xrightarrow{0} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{Blue} \xrightarrow{0} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{1} \text{red}$$
$$\text{Blue} \xrightarrow{0} \text{red}$$

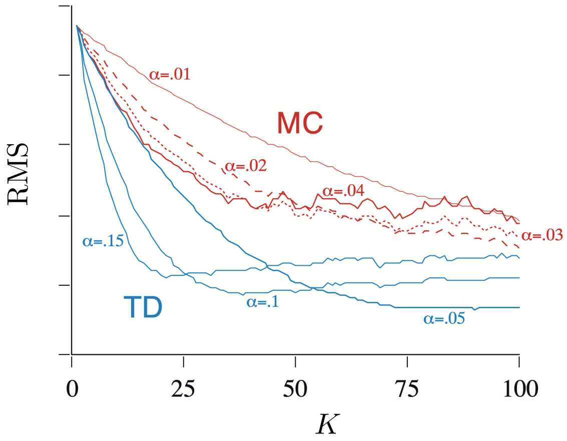
If we go twice over the batch of all episodes with TD, we get

$$\hat{v}\left(\text{Green}\right) \approx 0.5$$
$$\hat{v}\left(\text{Blue}\right) \approx 0.5$$

We get better if we go over the batch od data multiple times!

Temporal Difference vs Monte-Carlo: *Note II*

Let's see how Monte-Carlo performs against TD-0 algorithm for a bit larger example of random walk on a grid[1]



[1] *This figure is taken from Chapter 6 of Sutton and Barto's book*

Recall: *Bias and Variance of Estimator*

At this point, we need to have some clue about bias and variance of an estimator

$\boxed{\textit{If you need to recap, please look at the board}}$

Temporal Difference vs Monte-Carlo: *Note II*

What has been seen in the diagram is a general behavior

- *Monte-Carlo is good in sense of bias but bad in terms of variance*
  - ↳ *It always returns an unbiased estimator of value, i.e.,*

$$\mathbb{E}\left\{\hat{v}_\pi\left(s\right)\right\} = v_\pi\left(s\right)$$

  - ↳ *It's estimation is however high variance, i.e.,*

$$\mathbb{E}\left\{\left(\hat{v}_\pi\left(s\right) - v_\pi\left(s\right)\right)^2\right\} \rightsquigarrow \text{large}$$

- *TD-0 is good in sense of variance but can be bad in terms of bias*
  - ↳ *It can return a biased estimator of value, i.e.,*

$$\mathbb{E}\left\{\hat{v}_\pi\left(s\right)\right\} \neq v_\pi\left(s\right)$$

  - ↳ *It's estimation is low variance, i.e.,*

$$\mathbb{E}\left\{\left(\hat{v}_\pi\left(s\right) - v_\pi\left(s\right)\right)^2\right\} \rightsquigarrow \text{small}$$

Policy Iteration with TD-0

We can use TD-0 to implement another variant of GPI

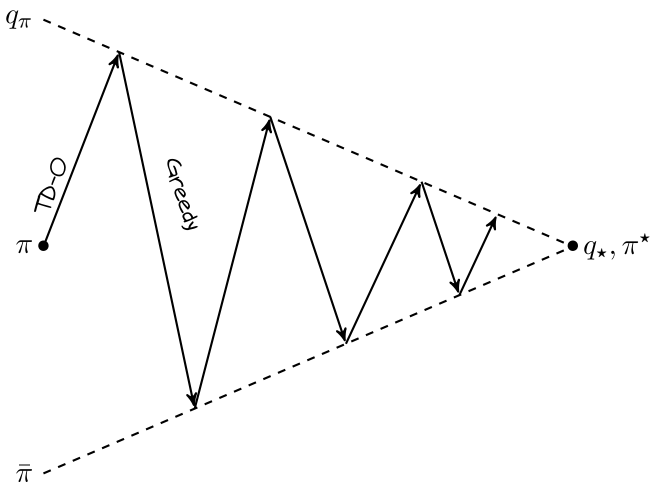
```
MC_PolicyItr():
 1: Initiate two random policies $\pi$ and $\bar{\pi}$
 2: while $\pi \neq \bar{\pi}$ do
 3:     $\hat{q}_\pi$ = TD_QEval($\pi$) and $\pi \leftarrow \bar{\pi}$
 4:     $\bar{\pi}$ = Greedy($\hat{q}_\pi$)
 5: end while
```

And, recall that our greedy algorithm is

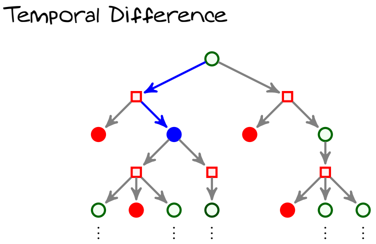
```
Greedy($q_\pi$):
 1: for $n = 1 : N$ do
 2:     Improve the by taking deterministically the best action
            $$\bar{\pi}(a^m|s^n) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_\pi(s^n, a^m) \\ 0 & m \neq \operatorname*{argmax}_m q_\pi(s^n, a^m) \end{cases}$$
 3: end for
```

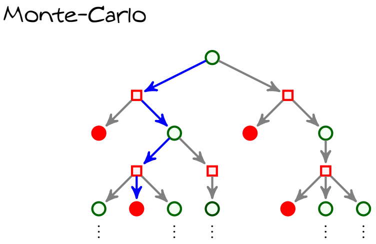
GPI with TD-0: Visualization

We can plot the same figure again in this case



Monte-Carlo vs TD-0: *Spectrum*





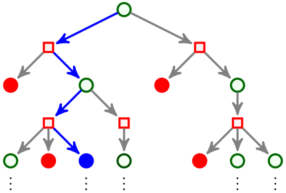
TD-0 and Monte-Carlo are two extreme sides of a *spectrum*

- *In TD-0, we use sample trajectory only for one step*
- *In Monte-Carlo, we use the complete sample trajectory for each state*

Can we draw a solution between these two extreme points?

First Solution: *Bootstrapping with More Steps*

A primary approach to find such a balanced solution is to *extend the idea of bootstrapping to a larger number of steps*



+ *How can we do it?*
– Well! We could simply expand the recursive property of value function

n-Bootstrapping

Looking at a sample return, we can simply write

$$\begin{aligned} G_t &= R_{t+1} + \gamma G_{t+1} \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 G_{t+2} \\ &\vdots \\ &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n R_{t+1+n} \gamma^{n+1} G_{t+1+n} \end{aligned}$$

Now, assuming that *at time step t we are at state s*, *i.e.*,

$$S_t = s, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1} \xrightarrow{R_{t+2}} \cdots \xrightarrow{R_{t+1+n}} S_{t+1+n}$$

we can bootstrap over a longer part of sample trajectory

n-Bootstrapping

$$S_t = s, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1} \xrightarrow{R_{t+2}} \cdots \xrightarrow{R_{t+1+n}} S_{t+1+n}$$

In this trajectory, we can expand Bellman equation with deeper recursion

$$v_\pi(s) = \sum_{i=0}^{n} \gamma^i \mathbb{E}\left\{R_{t+i+1} \middle| s\right\} + \gamma^{n+1} \mathbb{E}_\pi\left\{v_\pi\left(S_{t+n+1}\right) \middle| s\right\}$$

So, if we have K sample trajectory, we can estimate value of state s by n steps of bootstrapping, i.e.,

$$\hat{v}_\pi(s) = \frac{1}{K} \sum_{k=1}^{K} \underbrace{\left(\sum_{i=0}^{n} \gamma^i R_{t+i+1}[k] + \hat{v}_\pi\left(S_{t+n+1}[k]\right)\right)}_{\text{computed on sample } k}$$

n-Bootstrapping $\equiv$ *TD-n*

Let's formulate this approach: *for a given sample trajectory and value function estimator $\hat{v}_\pi(\cdot)$, we define n-bootstrapping return at time t as*

$$G_t^n = \sum_{i=0}^{n} \gamma^i R_{t+i+1} + \gamma^{n+1} \hat{v}_\pi(S_{t+n+1})$$

Given that the sample trajectory was started at state $S_t = s$, we can use online averaging and update the value estimator of state s as

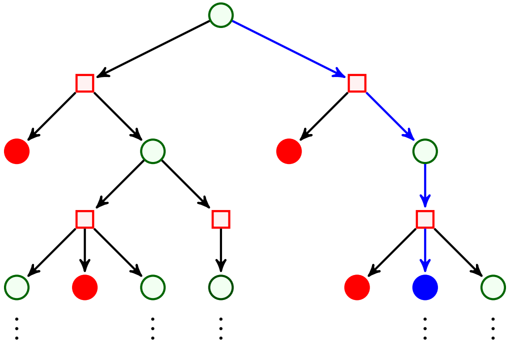
$$\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha(G_t^n - \hat{v}_\pi(s))$$

This is what we call TD-n method of learning

$$\boxed{\text{\textit{This is obviously more general than TD-0! In TD-0, we had simply } } n = 0}$$

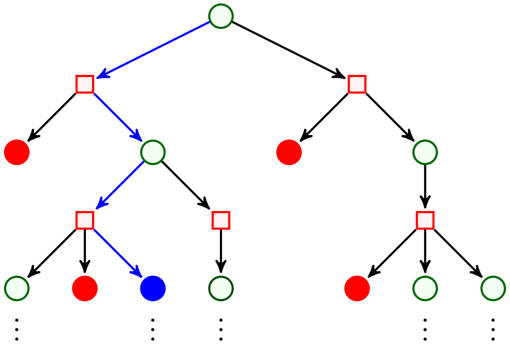
Backup Diagram: *Sampling with n-Bootstrapping*

With n-bootstrapping we sample $(n+1)$-step trajectories from action-state tree of environment



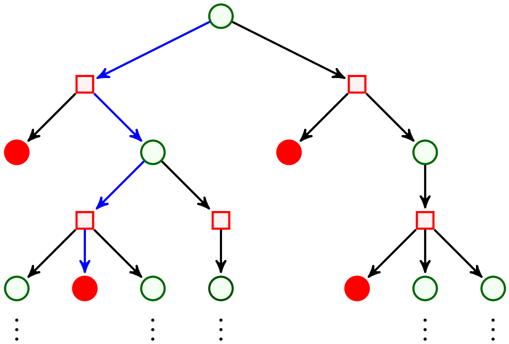
Backup Diagram: *Sampling with n-Bootstrapping*

With n-bootstrapping we sample $(n+1)$-step trajectories from action-state tree of environment



Backup Diagram: *Sampling with n-Bootstrapping*

With n-bootstrapping we sample $(n+1)$-step trajectories from action-state tree of environment



TD-n: *Policy Evaluation*

We can extend our TD-0 evaluation algorithm to TD-n

```
TDn_Eval($\pi$):
1: Initiate estimator of value as $\hat{v}_\pi(s) = 0$ for all states
2: for episode $= 1 : K$ do
3:     Initiate with a random state $S_0$ and act via policy $\pi(a|s)$
4:     Sample a trajectory until either a terminal stated or some terminating $T$

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

5:     for $t = 0 : T - n - 1$ do
6:         Compute $G = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n+1} v_\pi(S_{t+n+1})$     ⋆
7:         Update as $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha(G - \hat{v}_\pi(S_t))$
8:     end for
9: end for
```

TD-n: *Policy Q-Evaluation*

Same-wise, we can extend our algorithm for action-value computation

```
TD$n$_QEval($\pi$):
1: Initiate estimator of value as $\hat{q}_\pi(s, a) = 0$ for all states
2: for episode $= 1 : K$ do
3:     Initiate with a random state-action pair $(S_0, A_0)$ and act via policy $\pi(a|s)$
4:     Sample a trajectory until either a terminal stated or some terminating $T$

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

5:     for $t = 0 : T - n - 1$ do
6:         Compute $G = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n+1} v_\pi(S_{t+n+1})$        $\star$
7:         Update as $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
8:     end for
9: end for
```

TD-∞: *Going Back to Monte-Carlo*

+ *You told us that we look for a solution between TD-0 and Monte-Carlo! I see TD-0 is TD-n with $n = 0$, but where does Monte-Carlo stand?*
– Well! You may see already that *Monte-Carlo* is *TD-∞*

Say the environment is episodic: we can say that if we bootstrap very deep; then, at some point we hit a terminal state, i.e.,

$$\lim_{n\to\infty} v_\pi \left(S_{t+n+1}\right) = 0$$

This concludes that at any time t in an episodic environment

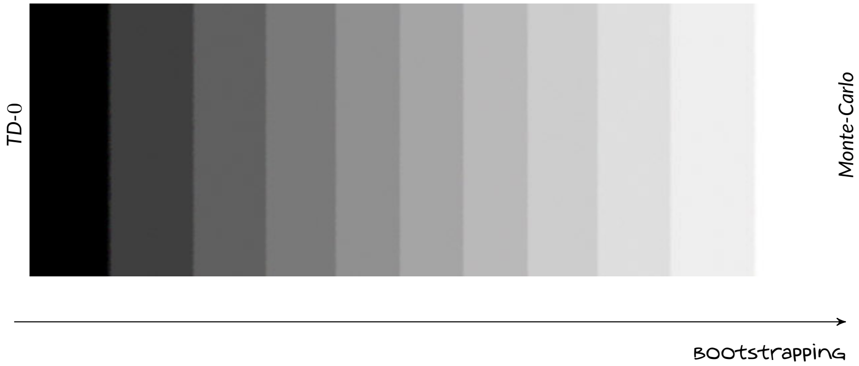
$$G_t^\infty = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} = G_t$$

and hence TD-∞ will update its estimator by

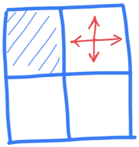
$$\hat{v}_\pi \left(s\right) \leftarrow \hat{v}_\pi \left(s\right) + \alpha \left(G_t - \hat{v}_\pi \left(s\right)\right)$$

TD-n: *A Discrete Spectrum*

We can look at TD-n as a discrete spectrum between Monte-Carlo and TD-0



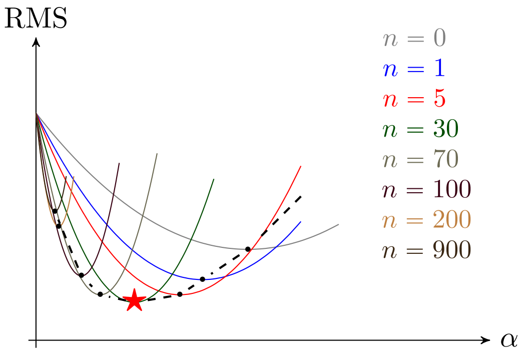
Example: *Dummy Grid World with Random Walk*



Once you got home, try to get back to our dummy world and use TD-n method for multiple n to compute the values for uniform random policy ☺

Typical Behavior: *Variation Against Depth*

If you do some practice with random walk, you will see following curves for different choices of n



We observe a minimum against n: this is a typical behavior! $\boxed{\text{Any illustration?}}$

Averaging Different Depth

+ *How deep we should then bootstrap?*
– Well! We could try to find the best one for each setting, or *we could average the result of multiple bootstrapping depths*

For example, we can

```
1: Initiate $\cdots$
2: for episode $= 1 : K$ do
3:     $\cdots$
4:     for $t = 0 : T - 3$ do
5:         Compute $G_0 = R_{t+1} + \gamma v_\pi\left(S_{t+1}\right)$
6:         Compute $G_2 = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi\left(S_{t+3}\right)$
7:         Set $G = \left(G_0 + G_2\right)/2$
8:         Update as $\hat{v}_\pi\left(S_t\right) \leftarrow \hat{v}_\pi\left(S_t\right) + \alpha\left(G - \hat{v}_\pi\left(S_t\right)\right)$
9:     end for
10: end for
```

Averaging Different Depth: λ-*Return*

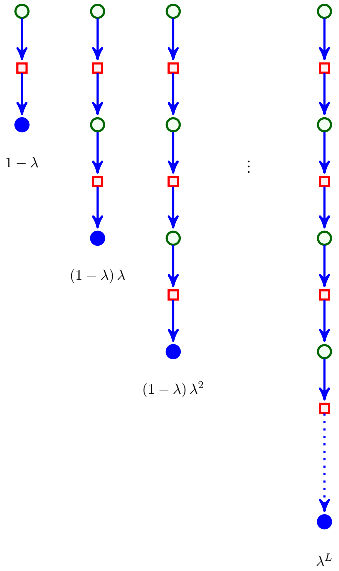
+ *Why should that be a good idea?*
– Because *if the good one is within the average; then, it could dominate and improve* estimation
+ *Then, which ones we should take? We still have no clue!*
– Let's take all of them! We can do it by *geometric weights!*

λ-Return

For any $0 \leqslant \lambda \leqslant 1$, *the* λ*-return at time* t *over* L *steps is defined as*

$$G_t^\lambda = (1-\lambda) \sum_{n=0}^{L-1} \lambda^n G_t^n + \lambda^L G_t^L$$

Averaging Different Depth: λ-*Return*



For each state in the sample trajectory, we can

- *Compute* 0*-bootstrapping return*
  - ↳ *Give it weight* $1-\lambda$
- *Compute* 1*-bootstrapping return*
  - ↳ *Give it weight* $(1-\lambda)\,\lambda$

$\vdots$

- *Compute* $(L-1)$*-bootstrapping return*
  - ↳ *Give it weight* $(1-\lambda)\,\lambda^{L-1}$
- *Compute* L*-bootstrapping return*
  - ↳ *Give it weight* λ^L

TD$_\lambda$: *Policy Evaluation*

We can evaluate policy by averaging its λ-returns over multiple episodes

```
TD_Eval$_\lambda$($\pi$):
1: Initiate estimator of value as $\hat{v}_\pi(s^n) = 0$ for $n = 1 : N$
2: for episode $= 1 : K$ do
3:     Initiate with a random state $S_0$ and act via policy $\pi(a|s)$
4:     Sample a trajectory until either a terminal stated or some terminating $T$

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

5:     for $t = 0 : T - 1$ do
6:         Set $G \leftarrow G_t^\lambda$ towards end of trajectory
7:         Update as $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha(G - \hat{v}_\pi(S_t))$
8:     end for
9: end for
```

TD-λ: *Special Cases*

It is easy to see that $\lambda = 0$ and $\lambda = 1$ are again two extreme cases: *assume we are dealing with an episodic environment*

- *with* $\lambda = 0$
  - ↳ *All weights are zero but that of G_t^0 which is weighted one*

$$G_t^\lambda = R_{t+1} + \gamma G_{t+1}$$

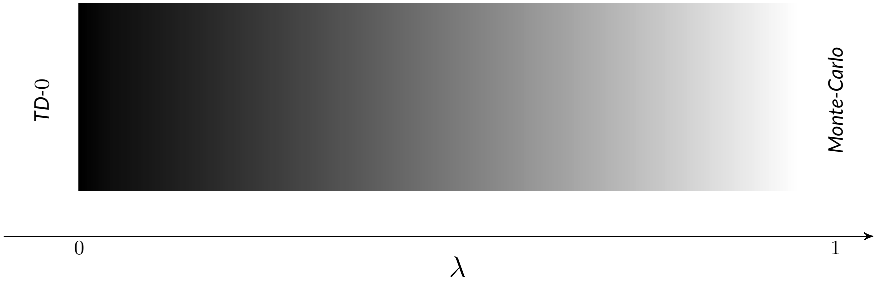
  - ↳ *this is basic bootstrapping: TD$_0$ is hence simply TD-0*
- *with* $\lambda = 1$
  - ↳ *All weights are zero but that of G_t^{T-t} which is weighted one*

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

  - ↳ *this basic Monte-Carlo: TD$_1$ is Monte-Carlo*

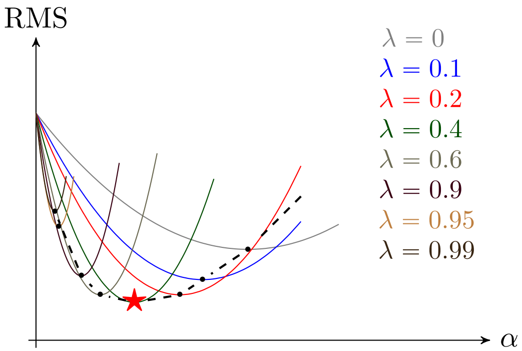
TD$_\lambda$: *A Continuous Spectrum*

We can look at TD$_\lambda$ as a continuous spectrum between Monte-Carlo and TD-0



Typical Behavior: *Variation Against λ*

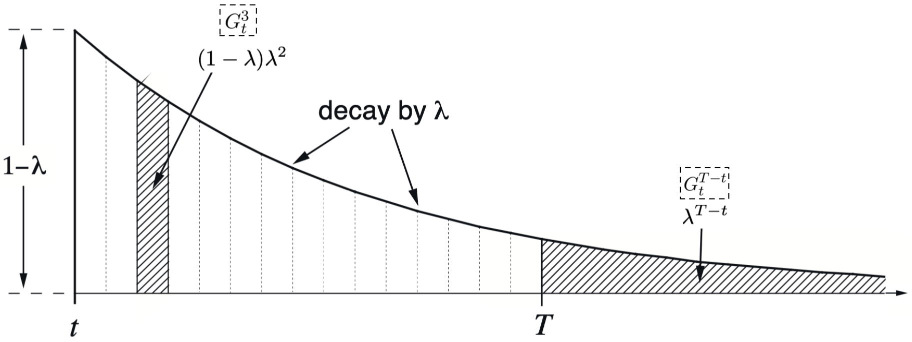
If you do some practice with random walk, you will see following curves for different choices of λ



We observe a minimum against λ: this is analog to TD-n behavior!

TD$_\lambda$: *Weighting Function*

Let's look at the weights decay in λ-return[2]



[2] *From Sutton and Barto's book in Chapter 12*

Assigning Credit for Future by *Weighting*

+ *What is the intuition behind computing the λ-return with those weights?*
– This is a very valid question! Let's see!

Let's look back at TD$_\lambda$ approach: at each time t in the trajectory, we compute

$$G_t^\lambda = \sum_{n=1}^{T-t-1} w_t^n G_t^n$$

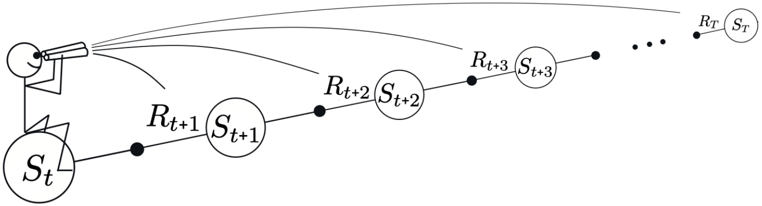
and then update the value of the current state S_t based on these future returns

the more far away this future is $\leftrightsquigarrow$ the less weight its return gets

In fact, we are assigning credit to our current state which will impact our future update: the more we go forward in time, the less this impact will be

TD$_\lambda$: *Forward View*

We can imagine this time advancement *as building impact towards future*[3]



But this approach is hard to be implemented online

we need to wait till the end of episode to compute the λ-returns!

[3]*This figure is taken from Sutton and Barto's book in Chapter 12*

TD$_\lambda$: *Backward View*

+ *But, it does not seem to be another way for credit assignment?*

– Well maybe we could apply the same idea backward

+ *How can we do it?*

– We can invoke the idea of eligibility tracing

Eligibility Tracing in Nutshell

At each time, when we update the value of a state in a sample trajectory, we also update the value of previous states we already met, with a weight decaying as we go back in time

Let's make an algorithm for that!

Eligibility Tracing

```
ElgTrace($S_t, E(\cdot)$):
 1: Eligibility tracing function has $N$ components, i.e., $E(s)$ for all states
 2: for $n = 1 : N$ do
 3:     Update $E(s^n) \leftarrow \gamma\lambda E(s^n)$ ⇜ choosing $\gamma\lambda$ for equivalency to forward view
 4: end for
 5: Update $E(S_t) \leftarrow E(S_t) + 1$
```

Say we initiate $E(s) = 0$ for all states and get to the following trajectory

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

Assume that we set $\gamma\lambda = 0.1$; then we have

1. *At $t = 0$, we only change $E(S_0) = 1$* ⇜ fresh memory $\equiv$ high impact
2. *At $t = 1$, we change $E(S_0) = 0.1$ and $E(S_1) = 1$*

Eligibility Tracing

```
ElgTrace($S_t$, $E(\cdot)$):
1: Eligibility tracing function has $N$ components, i.e., $E(s)$ for all states
2: for $n = 1 : N$ do
3:     Update $E(s^n) \leftarrow \gamma\lambda E(s^n)$ $\leftsquigarrow$ $\boxed{\text{choosing } \gamma\lambda \text{ for equivalency to forward view}}$
4: end for
5: Update $E(S_t) \leftarrow E(S_t) + 1$
```

Say we initiate $E(s) = 0$ for all states and get to the following trajectory

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

Now say that we see at $t = 2$ the same state as in $t = 0$, i.e., $S_0 = S_2$

3. *At $t = 2$, we change $E(S_1) = 0.1$ and $E(S_0) \leftarrow 0.1E(S_0) + 1 = 1.01$*

Updating with Eligibility Tracing: *Intuition*

+ *What is the use of this algorithm?*
- We can simply *update all previous states each time t weighted by eligibility traces*

We can 0-bootstrap at each time, i.e., compute error as

$$\Delta_t = \underbrace{R_{t+1} + \gamma \hat{v}_\pi \left(S_{t+1} \right)}_{G_t^0} - \hat{v}_\pi \left(S_t \right)$$

and update any state $s = S_0, \ldots, S_t$ that has non-zero trace of eligibility as

$$\hat{v}_\pi \left(s \right) \leftarrow \hat{v}_\pi \left(s \right) + \alpha \Delta_t E_t \left(s \right)$$

with $E_t \left(s \right)$ denoting the eligibility trace that we have updated up to time t

TD$_\lambda$ vs Eligibility Tracing

+ *Is there any concrete reason beside simple intuition that this is a good idea?*
– We can actually see that this is an online form of basic TD$_\lambda$

In fact, we can show by telescopic sum that

$$\sum_{t=0}^{T-1} \Delta_t E_t(s) = \sum_{t=0}^{T-1} \left(G_t^\lambda - \hat{v}_\pi(S_t) \right) \mathbf{1}\{S_t = s\}$$

This means that at the end of episode, we are updating the same!

- *If we set $\lambda = 0$ in the eligibility tracing*
  - ↳ *all eligibility trace remains 0 for all states: only we have $E(S_t) = 1$*
  - ↳ *we are back to TD-0 as it was with TD$_0$*
- *If we set $\lambda = 1$ in the eligibility tracing*
  - ↳ *we are back to Monte-Carlo approach as in TD$_1$*

TD$_\lambda$ with Eligibility Tracing: *Policy Evaluation*

We can evaluate policy by averaging its λ-returns over multiple episodes

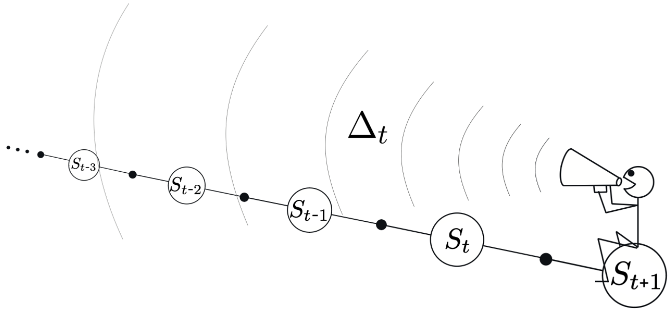
```
ElgTD_Eval$_\lambda$($\pi$):
 1: Initiate value estimator and eligibility traces as $\hat{v}_\pi(s) = 0$ and $E(s) = 0$ for all $s$
 2: for episode $= 1 : K$ do
 3:     Initiate with a random state $S_0$ and act via policy $\pi(a|s)$
 4:     Sample a trajectory until either a terminal stated or some terminating $T$

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

 5:     for $t = 0 : T-1$ do
 6:         $E(\cdot) \leftarrow$ ElgTrace($S_t, E(\cdot)$)
 7:         Compute $G = R_{t+1} + \gamma \hat{v}_\pi(S_{t+1})$ and find error $\Delta = G - \hat{v}_\pi(S_t)$
 8:         for $n = 1 : N$ do
 9:             Update $\hat{v}_\pi(s^n) \leftarrow \hat{v}_\pi + \alpha E(s^n) \Delta$
10:         end for
11:     end for
12: end for
```

Eligibility Tracing: *Backward View*

We can imagine eligibility tracing *as backward assignment of credits*[4]



Now we can update values each time and don't need to wait till end of episode!

[4]*This figure is taken from Sutton and Barto's book in Chapter 12*

Control versus Prediction

What we have done up to now is *prediction*

Prediction

We are given by a policy and intend to evaluate it by sampling

But, in most applications we deal with a *control problem*

Control

We are looking to move towards optimal policy while we are sampling

- + *But, we already talked about GPI with sampling! Didn't we?*
- – Well! What we did makes sense if we learn offline!

GPI in Control Loop: *Offline vs Online Approach*

In *offline* RL, *we sample environment first and find the optimal policy later*

1. *Sample environment with sufficient number of episodes*
2. *Evaluate policies and improve them from the available dataset*
   ↳ *Go over the dataset over and over if needed*

We are however looking for an *online RL approach, we sample environment and learn optimal policy gradually as we sample*

1. *Take a single sample from environment, e.g., a single reward-state pair or a terminating trajectory*
2. *Estimate values and improve policy based on this single sample*
3. *Take a new sample* $\cdots$

+ *I am sure that we always thought about the second case! Isn't that right?!*

– Yes! But, if we want to do complete evaluation in each GPI iteration, *we will be extremely slow! Imagine* 1000 *episodes for each iteration!*

Direct GPI with *Prediction*

Recall a generic form of GPI with a prediction approach `X`

```
X_PolicyItr():
1: Initiate two random policies $\pi$ and $\bar{\pi}$
2: while $\pi \neq \bar{\pi}$ do
3:     $\hat{q}_\pi = \texttt{X\_QEval}(\pi)$ and $\pi \leftarrow \bar{\pi}$
4:     $\bar{\pi} = \texttt{Greedy}(\hat{q}_\pi)$
5: end while
```

and $\texttt{Greedy}(\hat{q}_\pi)$ *is the basic improvement strategy*

```
Greedy($\hat{q}_\pi$):
1: for $n = 1 : N$ do
2:     Improve the by taking deterministically the best action

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

3: end for
```

Direct GPI with *Prediction*

Recall a generic form of GPI with a prediction approach X

```
X_PolicyItr():
1: Initiate two random policies $\pi$ and $\bar{\pi}$
2: while $\pi \neq \bar{\pi}$ do
3:     $\hat{q}_\pi = \texttt{X\_QEval}(\pi)$ and $\pi \leftarrow \bar{\pi}$
4:     $\bar{\pi} = \texttt{Greedy}(\hat{q}_\pi)$
5: end while
```

If we want our evaluation to be accurate enough: we need to keep playing each policy for a large number of episodes

- *This is extremely sample inefficient*
  - ↳ *Imagine how many times we should lose the game to update our strategy!*
  - ↳ *We as human do not really need so many losses*
- *In many practical settings is really cost-inefficient*
  - ↳ *Amount of losses we should pay in each iteration to evaluate the policy is not worth it!*

Online Control Loop via GPI

+ *But, how can we do anything about this?*
– Maybe we could *improve the policy after each update of action-values*

```
X_Control():
1: Initiate two random policies $\pi$ and $\bar{\pi}$
2: while $\pi \neq \bar{\pi}$ do
3:     $\hat{q}_\pi$ = X_QUpdate($\pi$) and $\pi \leftarrow \bar{\pi}$
4:     $\bar{\pi}$ = Greedy($\hat{q}_\pi$)
5: end while
```

Here, `X_QUpdate`(π) *refers to one single update which is typically of the form*

$$\hat{q}_\pi (S_t, A_t) \leftarrow \hat{q}_\pi (S_t, A_t) + \alpha(G - \hat{q}_\pi (S_t, A_t))$$

for some G

Online Control Loop via GPI

```
X_Control():
1: Initiate two random policies $\pi$ and $\bar{\pi}$
2: while $\pi \neq \bar{\pi}$ do
3:     $\hat{q}_\pi = $ X_QUpdate$(\pi)$ and $\pi \leftarrow \bar{\pi}$
4:     $\bar{\pi} = $ Greedy$(\hat{q}_\pi)$
5: end while
```

+ *It sounds like a loose approach! Why should that work?!*

– We see accurate illustrations about that, *but for the moment*

↳ *we could think of a single update as a low-accuracy estimate*

↳ *we have already said the GPI is very robust against estimation error*

Let's start by a Monte-Carlo control loop

First Try: *Monte-Carlo Control Loop*

We can build a Monte-Carlo control loop for episodic environments

```
MC_Control($\pi$):
 1: Initiate estimator as $\hat{q}_\pi(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state-action pair $(S_0, A_0)$
 4:     Act via $\pi = \texttt{Greedy}(\hat{q}_\pi)$
 5:     Sample a trajectory
            $S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$: terminal
 6:     Initiate with $G = 0$
 7:     for $t = T-1 : 0$ do
 8:         Update current return $G \leftarrow R_{t+1} + \gamma G$
 9:         Update $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
10:     end for
11: end for
```

Monte-Carlo Control: *Updating Action-Values*

Comparing with `MC_QEval(`π`)`, there is only one difference, i.e.,

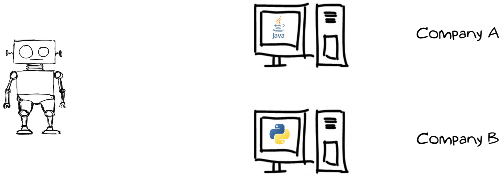
in-loop greedy improvement of the policy: line 4

1. *Estimate action-values over a sample trajectory*
2. *Improve the policy using this estimate by greedy approach*
3. *Sample the next trajectory using the improved policy*

+ *Can we guarantee the convergence of this control loop?*

– In the current state, not really! *Let's see an example!*

Example: *Our Multi-armed Bandit*

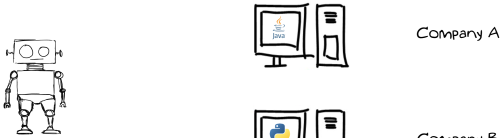


Let's get back to our very first RL problem in which the robot is to decide for a company: say the robot follows Monte-Carlo control loop

1. *it starts with a random decision*
   - ↳ *Say it decides for Company B and receives $150 income*
   - ↳ *Now, we have* $\hat{q}_\pi(\_, A) = 0$ *and* $\hat{q}_\pi(\_, B) = 150$
2. *in the next episode, it would definitely chooses to work at Company B*
   - ↳ *Say it receives $250 income this time*
   - ↳ *Now, we have* $\hat{q}_\pi(\_, A) = 0$ *and* $\hat{q}_\pi(\_, B) = 200$

$\vdots$

Example: *Our Multi-armed Bandit*



The robot keeps working at Company B!

+ *But, can we guarantee that company A is not paying better?!*
– *Well! Not really! In fact, even if we had worked there for a single day or so, we could still not guarantee!*

Greedy Improvement: *Lack of Exploration*

+ *Why is this happening? Why it doesn't happen when we apply direct GPI via Monte-Carlo?*
– In the latter, we do *exploration*; but now, we are only *exploiting*!

This is a general behavior of greedy improvement

Downside of Greedy Improvement

In greedy improvement, we only exploit our knowledge, i.e.,

we always act optimal based on what we know up to now

We thus lack exploration, i.e.,

we remain unaware about states and actions that we have not explored

We may never get the chance to explore them!

Improving via ϵ-Greedy Improvement

A classical approach to handle this issue is to improve by ϵ-greedy approach

ϵ-Greedy Improvement

Choose a small $0 < \epsilon < 1$, and improve after each update of action-values by greedy approach: at begining of each episode

- *with probability $1 - \epsilon$ act by the improved policy*
- *with probability ϵ act randomly*

We implemented this approach for multi-armed bandit in Assignment 1: in this approach we render a trade-off between exploitation and exploration

- *with probability $1 - \epsilon$ we exploit our improved policy*
- *with probability ϵ we explore the environment*

It's hard to find any improvement approach that can beat ϵ-greedy!

ϵ-Greedy Algorithm

We can algorithmically specify ϵ-greedy as

```
ε-Greedy($\hat{q}_\pi$):
1: for $n = 1 : N$ do
2:     Take next step randomly as
           $\bar{\pi}(a^m | s^n) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{M} & m = \operatorname{argmax}_m \hat{q}_\pi(s^n, a^m) \\ \frac{\epsilon}{M} & m \neq \operatorname{argmax}_m \hat{q}_\pi(s^n, a^m) \end{cases}$
3: end for
```

+ *That seems to solve exploration problem! But, is there any guarantee that $\bar{\pi}$ is going to be a better policy? For greedy approach, we could prove that we get always better!*
– Yes! We can actually prove it!

ϵ-Greedy Algorithm

Let's assume we have policy π given after ϵ-greedy improvement, and we improved it again via the ϵ-greedy approach from its action-values: *we can then write the value of new policy $\bar{\pi}$ as*

$$\begin{aligned} v_{\bar{\pi}}(s) &= \sum_{m=1}^{M} \bar{\pi}(a^m|s)\, q_{\bar{\pi}}(s, a^m) \\ &= \underbrace{\frac{\epsilon}{M} \sum_{m=1}^{M} q_\pi(s, a^m)}_{\text{exploration}} + \underbrace{(1-\epsilon) q_\pi(s, a^\star)}_{\text{exploitation}} \end{aligned}$$

We know that for any non-negative $w_1, \dots, w_M$ that add up to one, we have

$$\sum_{m=1}^{M} w_m q_\pi(s, a^m) \leqslant q_\pi(s, a^\star)$$

ϵ-Greedy Algorithm

We have the improved value in terms of the initial action-values as

$$v_{\bar{\pi}}(s) = \frac{\epsilon}{M} \sum_{m=1}^{M} q_\pi(s, a^m) + (1-\epsilon) q_\pi(s, a^\star)$$

Let's now define

$$w_m = \frac{\pi(a^m|s) - \epsilon/M}{1-\epsilon}$$

We note that since π is an ϵ-greedy policy, we have $w_m \geqslant 0$ and

$$\sum_{m=1}^{M} w_m = \sum_{m=1}^{M} \frac{\pi(a^m|s) - \epsilon/M}{1-\epsilon} = 1$$

ϵ-Greedy Algorithm

Now, let us replace this bound in the previous equation

$$\begin{aligned} v_{\bar{\pi}}(s) &= \frac{\epsilon}{M}\sum_{m=1}^{M} q_\pi(s, a^m) + (1-\epsilon) q_\pi(s, a^\star) \\ &\geqslant \frac{\epsilon}{M}\sum_{m=1}^{M} q_\pi(s, a^m) + (1-\epsilon)\sum_{m=1}^{M} \frac{\pi(a^m|s) - \epsilon/M}{1-\epsilon} q_\pi(s, a^m) \\ &= \sum_{m=1}^{M} \pi(a^m|s) q_\pi(s, a^m) = v_\pi(s) \end{aligned}$$

ϵ-Greedy Improvement Theorem

Let π and be an ϵ-greedy policies, i.e., computed from some action-value function using ϵ-greedy algorithm. Assume $\bar{\pi}$ is derived by ϵ-greedy improvement from $q_\pi(s, a)$; then, $\bar{\pi} \geqslant \pi$

Online Control Loop via GPI and ϵ-Greedy Improvement

We can now build our control loop via ϵ-greedy algorithm

```
X_Control():
1: Initiate two random policies $\pi$ and $\bar{\pi}$
2: while $\pi \neq \bar{\pi}$ do
3:     $\hat{q}_\pi = \texttt{X\_QUpdate}(\pi)$ and $\pi \leftarrow \bar{\pi}$
4:     $\bar{\pi} = \epsilon\texttt{-Greedy}(\hat{q}_\pi)$
5: end while
```

Attention

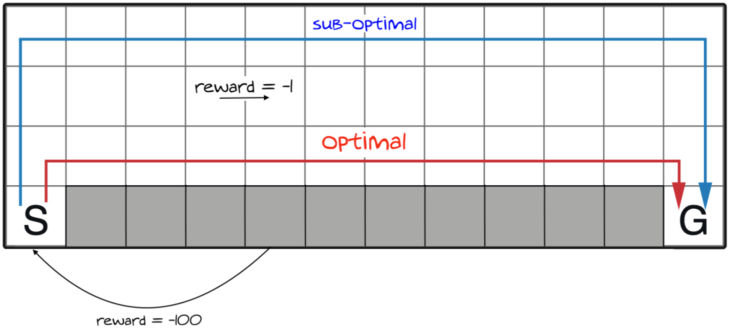
We are still using single update of action-values for policy improvement: this means that we may have bad estimates of action-values at initial iterations!

First Try: *Monte-Carlo Control Loop*

Monte-Carlo control loop for episodic environments is modified as

```
MC_Control($\pi$):
 1: Initiate estimator as $\hat{q}_\pi(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state-action pair $(S_0, A_0)$
 4:     Act via $\pi = \epsilon$-Greedy$(\hat{q}_\pi)$
 5:     Sample a trajectory
            $S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$: terminal
 6:     Initiate with $G = 0$
 7:     for $t = T - 1 : 0$ do
 8:         Update current return $G \leftarrow R_{t+1} + \gamma G$
 9:         Update $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
10:     end for
11: end for
```

Example: *Cliff Walking*

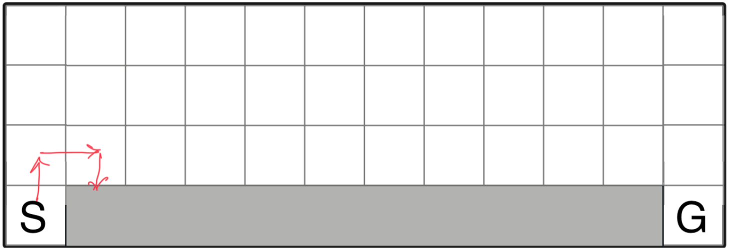


We have seen the cliff walking example in Assignment 1: we want to

- *get from S to G with shortest possible path*
- *avoid hitting the cliff $\equiv$ gray squares*
  - ↳ *each time we hit the cliff, we get back to S with a big negative reward*

Example: *Cliff Walking*

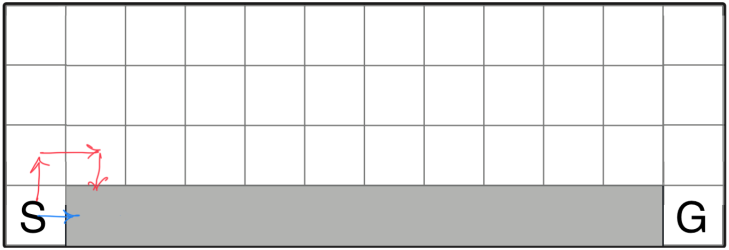
Say we use naive greedy policy: we start sampling trajectory and hit the cliff



We realize that our first action gave bad reward

Example: *Cliff Walking*

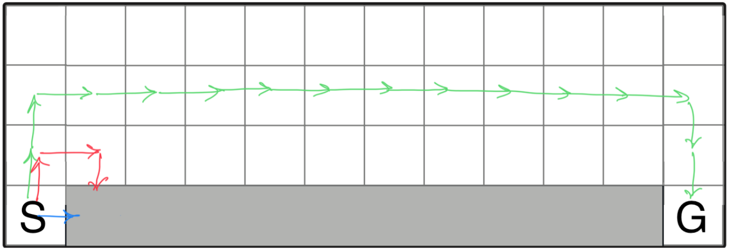
We now follow a better action, but we hit the cliff again



We realize that this action was even worse

Example: *Cliff Walking*

We get back to our first action, but now modify next actions



Say we are lucky and arrive at G

We will never go back to find the optimal path!

But with ϵ-greedy improvement, we get the chance to explore again: we may find the optimal path!

Greedy in Limit with Infinite Exploration Algorithms

+ *Sounds working! But can we guarantee that this approach will converge to optimal path?*
– Under some circumstances: Yes!

Recall that we said the following when we started Monte-Carlo

Asymptotic Convergence of Monte-Carlo

Let $\mathcal{C}_K(s, a)$ denote number of visits at state s followed by action a during K Monte-Carlo episodes. Assume random initialization is distributed such that

$$\lim_{K\to\infty} \mathcal{C}_K(s, a) = \infty$$

for any state s and action a; then, we can guarantee $\hat{q}_\pi(s, a) \xrightarrow{K\uparrow\infty} q_\pi(s, a)$

Greedy in Limit with Infinite Exploration Algorithms

The main idea in this result was that

> *As long as we do enough sampling, so that we see all states and actions enough number of times, Monte-Carlo will converge*

We can claim the same thing here

> *If we keep playing enough, we explore all states and actions; then, eventually we get very sure about optimal values and actions*

But there is a small point here: *if we keep on using ϵ-greedy policy even after we got sure, we can still perform sub-optimal*

> *We should stop exploring once we have visited all states and actions*

This is what we call

Greedy in Limit with Infinite Exploration $\equiv$ GLIE

GLIE Algorithms

GLIE Algorithms

A GPI-type control loop is GLIE, if for any state-action pair (s, a), we have the following asymptotic properties

1. *The number of visits to all state-action pair grows large*

$$\lim_{K\to\infty} \mathcal{C}_K(s, a) = \infty$$

2. *The improved policy in last episode converges to greedy policy*

$$\lim_{K\to\infty} \pi_K(a^m | s) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_{\pi_K}(s, a^m) \\ 0 & m \neq \operatorname*{argmax}_m q_{\pi_K}(s, a^m) \end{cases}$$

GLIE control algorithms converge to optimal policy

GLIE Algorithms

+ *It seems that they contradict! First one needs us to explore and the second to exploit!*
– We could simply get rid of it by scaling ϵ

Say we choose ϵ to scale reversely by the number of episodes, e.g.,

$$\epsilon_k = \frac{1}{k}$$

Then, we have both the constraints satisfied

1. *We keep exploring a lot in initial episodes*
2. *We focus more on exploiting in later episodes*

$\boxed{\textit{This is what we do in practice!}}$

ϵ-Greedy Monte-Carlo is GLIE

It is easy to show that Monte-Carlo with shrinking ϵ-greedy improvement is GLIE

```
MC_Control():
 1: Initiate estimator as $\hat{q}_\pi(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state-action pair $(S_0, A_0)$
 4:     Set $\epsilon = 1/k$ and act via $\pi = \epsilon$-Greedy$(\hat{q}_\pi)$
 5:     Sample a trajectory
            $S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$: terminal
 6:     Initiate with $G = 0$
 7:     for $t = T - 1 : 0$ do
 8:         Update current return $G \leftarrow R_{t+1} + \gamma G$
 9:         Update $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
10:     end for
11: end for
```

Control Loop via Temporal Difference

+ *But still we are not fully online! We need to wait till end of each episode!*
– Well! That’s right! But, we could use TD!

Using TD in the control loop will make our algorithm fully online

- *We update values after each state-action pair*
- *We then improve the policy*

$\boxed{\text{We should yet use } \epsilon\text{-greedy improvement to keep exploration}}$

SARSA: *State-Action-Reward State-Action*

SARSA $\equiv$ *State-Action Reward State-Action*

SARSA algorithms use TD along with ϵ-greedy update for the control loop

In general, we can develop various forms of SARSA

- *We may use TD-0 for updating action-values*
  - ↳ *This is the basic SARSA*
- *We may use TD-n for updating action-values*
  - ↳ *This is n-SARSA*
- *We may use TD$_\lambda$ for updating action-values*
  - ↳ *This is SARSA(λ)*

SARSA: *First Try*

Let's try to make a simple TD-based control loop

```
TD_Control():
 1: Initiate estimator as $\hat{q}_\pi(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state-action pair $(S_0, A_0)$
 4:     for $t = 0 : T - 1$ that is either terminal or terminated do
 5:         Act $A_t$ and observe
                        $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
 6:         Update policy to $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}_\pi)$
 7:         Draw the new action $A_{t+1}$ from $\pi(\cdot|S_{t+1})$
 8:         Compute $\hat{v}_\pi(S_{t+1})$ from $\hat{q}_\pi(S_{t+1}, a)$ and $\pi(\cdot|S_{t+1})$
 9:         Set $G \leftarrow R_{t+1} + \gamma \hat{v}_\pi(S_{t+1})$
10:         Update $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
11:     end for
12: end for
```

SARSA: *Going On-Policy*

In *line 8* of our control algorithm: *we compute $\hat{v}_\pi(S_{t+1})$ as*

$$\hat{v}_\pi(S_{t+1}) = \sum_{m=1}^{M} \pi(a^m | S_{t+1}) \hat{q}_\pi(S_{t+1}, a^m)$$

But, we do know that

1. *our estimates $\hat{q}_\pi(S_{t+1}, a^m)$ are not that good, and also*
2. *our policy has led use to next action A_{t+1}*

So, we could move on our policy and write

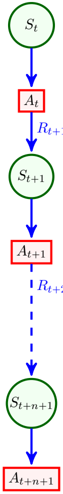
$$\pi(a|S_{t+1}) = \begin{cases} 1 & a = A_{t+1} \\ 0 & a \neq A_{t+1} \end{cases} \rightsquigarrow \hat{v}_\pi(S_{t+1}) = \hat{q}_\pi(S_{t+1}, A_{t+1})$$

We call this approach on-policy, since move on our policy

SARSA: *Basic Algorithm*

```
SARSA():
 1: Initiate estimator as $\hat{q}_\pi(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state-action pair $(S_0, A_0)$
 4:     for $t = 0 : T - 1$ that is either terminal or terminated do
 5:         Act $A_t$ and observe
                        $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
 6:         Update policy to $\pi \leftarrow \epsilon\text{-}\mathtt{Greedy}(\hat{q}_\pi)$
 7:         Draw the new action $A_{t+1}$ from $\pi(\cdot|S_{t+1})$ and move on policy
                        $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1}$
 8:         Set $G \leftarrow R_{t+1} + \gamma \hat{q}_\pi(S_{t+1}, A_{t+1})$
 9:         Update $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
10:     end for
11: end for
```

SARSA: *Deeper Return Samples*



We can use a longer trajectory while we learn on-policy, i.e.,

$$G^n = \sum_{i=0}^{n} R_{t+i+1} + \gamma \hat{q}_\pi \left(S_{t+n+1}, A_{t+n+1} \right)$$

This will however add extra delay!

As a practice, you could

re-write the basic SARSA with n-return ☺

SARSA(λ):*Tracing Eligibility of State-Action Pairs*

We can extend SARSA to the case with λ-return: *we have two options*

- *the case with forward-view*
  - ↳ *We know this is not practical! So, let's skip the details*
- *the case with backward-view and eligibility tracing*
  - ↳ *Let's look into this one*

We first extend eligibility tracing to the case with state-action pairs

```
ElgTrace($S_t, A_t, E(\cdot) \mid \lambda$):
1: Eligibility tracing function has $NM$ components, i.e., $E(s,a)$ for all state-action pairs
2: for all state-action pairs $(s,a)$ do
3:     Update $E(s,a) \leftarrow \gamma\lambda E(s,a)$
4: end for
5: Update $E(S_t, A_t) \leftarrow E(S_t, A_t) + 1$
```

SARSA: *Alternative via TD-λ*

```
SARSA($\lambda$):
 1: Initiate $\hat{q}_\pi(s, a) = 0$ and $E(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state-action pair $(S_0, A_0)$
 4:     for $t = 0 : T-1$ that is either terminal or terminated do
 5:         $E(\cdot) \leftarrow$ ElgTrace($S_t, A_t, E(\cdot) \;|\lambda$)
 6:         Act $A_t$ and observe $R_{t+1}$ and $S_{t+1}$
 7:         Update policy to $\pi \leftarrow \epsilon$-Greedy$(\hat{q}_\pi)$
 8:         Draw the new action $A_{t+1}$ from $\pi(\cdot|S_{t+1})$ and move on policy

                          $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1}$

 9:         Set $\Delta \leftarrow R_{t+1} + \gamma \hat{q}_\pi(S_{t+1}, A_{t+1}) - \hat{q}_\pi(S_t, A_t)$
10:         for all state-action pairs $(s, a)$ do
11:             Update $\hat{q}_\pi(s, a) \leftarrow \hat{q}_\pi(s, a) + \alpha \Delta E(s, a)$
12:         end for
13:     end for
14: end for
```

Going Off-Policy

Let's think about a fundamental question: *while sampling the environment with a specific policy π, can we estimate the values of another policy $\bar{\pi}$?*

+ *Why should this be a fundamental question?*
- Well! There are several reasons
  - ↳ *Maybe we sampled environment with our bad* policy*: can't we use our sample again?*
  - ↳ *Maybe we are looking at other players: can't we learn something about the environment from their samples?*
    - ↳ *Maybe they are good* players*: can't we use this fact to improve our policy?*
    - ↳ *Maybe they are bad* players*: can't we use this fact to avoid doing mistakes?*

This is the idea of off-policy control

$\boxed{\textit{Let's start with some baiscs}}$

Importance Sampling

Consider following problem: *we have random variable X drawn as $X \sim p(x)$ whose mean is*

$$\mu_p = \mathbb{E}_p\left\{X\right\} = \sum_x p(x)\, x$$

We want to know how would be the expectation if we had $X \sim q(x)$: we write

$$\begin{aligned}\mu_q = \mathbb{E}_q\left\{X\right\} &= \sum_x q(x)\, x \\ &= \sum_x p(x) \frac{q(x)}{p(x)} x = \mathbb{E}_p\left\{\frac{q(X)}{p(X)} X\right\}\end{aligned}$$

This gives us possibility to

estimate $\mathbb{E}_q\left\{X\right\}$ using samples drawn from $p(x)$

Importance Sampling

Say we have drawn K samples from $p(x)$, i.e., we have

$$X_1, X_2, \dots, X_K$$

We can use Monte-Carlo to estimate μ_p as

$$\hat{\mu}_p = \frac{1}{K} \sum_{k=1}^{K} X_k$$

We can also use Monte-Carlo to estimate μ_q as

$$\hat{\mu}_q = \frac{1}{K} \sum_{k=1}^{K} \frac{q(X_k)}{p(X_k)} X_k$$

We call this method importance sampling

Off-Policy Control via Importance Sampling

Now, let's get back to our problem: *assume we have played with policy π and collected K sample trajectories of length T all started at state $S_0 = s$, i.e.,*

$$s = S_0[k], A_0[k] \xrightarrow{R_1[k]} S_1[k], A_1[k] \xrightarrow{R_2[k]} \cdots \xrightarrow{R_T[k]} S_T[k]$$

for $k = 1 : K$; then, we could write

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

This is the basic Monte-Carlo

Off-Policy Control via Importance Sampling

But now, we want to use samples to evaluate another policy $\bar{\pi}$

$$s = S_0[k], A_0[k] \xrightarrow{R_1[k]} S_1[k], A_1[k] \xrightarrow{R_2[k]} \cdots \xrightarrow{R_T[k]} S_T[k]$$

We could also use importance sampling to write

$$\begin{aligned}\hat{v}_{\bar{\pi}}(s) &= \frac{1}{K}\sum_{k=1}^{K} \frac{\Pr\{\text{same action sequence with } \bar{\pi}\}}{\Pr\{\text{same action sequence with } \pi\}} G[k] \\ &= \frac{1}{K}\sum_{k=1}^{K} \frac{\bar{\pi}(A_0[k]|S_0[k]) \cdots \bar{\pi}(A_{T-1}[k]|S_{T-1}[k])}{\pi(A_0[k]|S_0[k]) \cdots \pi(A_{T-1}[k]|S_{T-1}[k])} G[k] \\ &= \frac{1}{K}\sum_{k=1}^{K} \prod_{\ell=0}^{T-1} \frac{\bar{\pi}(A_\ell[k]|S_\ell[k])}{\pi(A_\ell[k]|S_\ell[k])} G[k]\end{aligned}$$

Off-Policy Control via Importance Sampling

We can further update the estimate in an online fashion from

$$S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1} \xrightarrow{R_{t+2}} \cdots \xrightarrow{R_T} S_T$$

by online averaging as

$$\hat{v}_{\bar{\pi}}\left(S_t\right) \leftarrow \hat{v}_{\bar{\pi}}\left(S_t\right) + \alpha \left(\prod_{\ell=t}^{T-1} \frac{\bar{\pi}\left(A_\ell | S_\ell\right)}{\pi\left(A_\ell | S_\ell\right)} G_t - \hat{v}_{\bar{\pi}}\left(S_t\right) \right)$$

So, we are evaluating $\bar{\pi}$ via Monte-Carlo

off our policy π

This is off-policy control

Off-Policy Control via Importance Sampling

We can further apply off-policy control via TD

$$\hat{v}_{\bar{\pi}}(S_t) \leftarrow \hat{v}_{\bar{\pi}}(S_t) + \alpha \left(\frac{\bar{\pi}(A_t|S_t)}{\pi(A_t|S_t)} \left(R_{t+1} + \gamma \hat{v}_{\bar{\pi}}(S_{t+1})\right) - \hat{v}_{\bar{\pi}}(S_t) \right)$$

Note that for action-values estimate

R_{t+1} *does not depend any more on policy as we know action* A_t

Therefore, we have for action-value update

$$\hat{q}_{\bar{\pi}}(S_t, A_t) \leftarrow \hat{q}_{\bar{\pi}}(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \frac{\bar{\pi}(A_t|S_t)}{\pi(A_t|S_t)} \hat{v}_{\bar{\pi}}(S_{t+1}) - \hat{q}_{\bar{\pi}}(S_t, A_t) \right)$$

Q-Learning

Q-Learning

Q-learning is an off-policy TD control algorithm, where we sample with ϵ-greedy policy but update the action-values to evaluate greedy policy

This means in Q-learning π is ϵ-greedy policy and $\bar{\pi}$ is greedy. Let's consider basic TD evaluation: so, we can write

$$\hat{q}_{\bar{\pi}}\left(S_t, A_t\right) \leftarrow \hat{q}_{\bar{\pi}}\left(S_t, A_t\right) + \alpha\left(G - \hat{q}_{\bar{\pi}}\left(S_t, A_t\right)\right)$$

where G should be

$$G = R_{t+1} + \gamma \hat{v}_{\bar{\pi}}\left(\bar{S}_{t+1}\right)$$

Since we sample by ϵ-greedy policy π, we use importance sampling and write

$$G = R_{t+1} + \gamma \frac{\bar{\pi}\left(A_t | S_t\right)}{\pi\left(A_t | S_t\right)} \hat{v}_{\bar{\pi}}\left(S_{t+1}\right)$$

Q-Learning

But, we really don't need importance sampling: we can simply observe that

$$\hat{v}_{\bar{\pi}}\left(S_{t+1}\right) = \sum_{m=1}^{M} \hat{q}_{\bar{\pi}}\left(S_{t+1}, a^m\right) \bar{\pi}\left(a^m | S_{t+1}\right) = \max_m \hat{q}_{\bar{\pi}}\left(S_{t+1}, a^m\right)$$

and we do know that

$$\frac{\bar{\pi}\left(A_t | S_t\right)}{\pi\left(A_t | S_t\right)} = \mathbf{1}\left\{A_t = \operatorname*{argmax}_a \hat{q}_{\bar{\pi}}\left(S_t, a\right)\right\}$$

So, we could directly update as

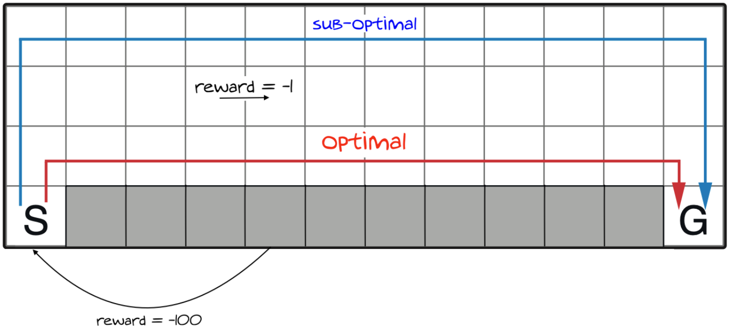
$$\hat{q}_{\bar{\pi}}\left(S_t, A_t\right) \leftarrow \hat{q}_{\bar{\pi}}\left(S_t, A_t\right) + \alpha\Big(R_{t+1} + \gamma \max_m \hat{q}_{\bar{\pi}}\left(S_{t+1}, a^m\right) - \hat{q}_{\bar{\pi}}\left(S_t, A_t\right)\Big)$$

This concludes Q-learning algorithm

Q-Learning: *Basic Algorithm*

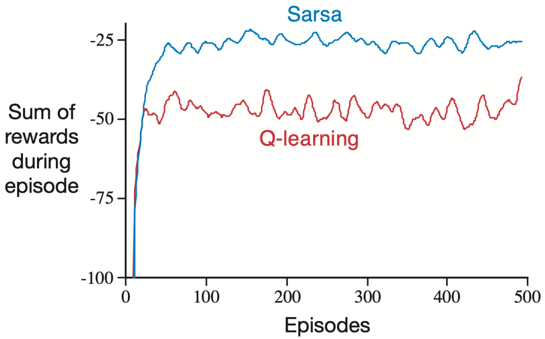
```
Q-Learning():
 1: Initiate estimator as $\hat{q}_\star(s, a) = 0$ for all states and actions
 2: for episode $= 1 : K$ or until $\pi$ stops changing do
 3:     Initiate with a random state $S_0$
 4:     for $t = 0 : T - 1$ that is either terminal or terminated do
 5:         Update policy to $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}_\star)$
 6:         Draw action $A_t$ from $\pi(\cdot|S_t)$ and observe
                          $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
 7:         Set $G \leftarrow R_{t+1} + \gamma \max_m \hat{q}_\star(S_{t+1}, a^m)$
 8:         Update $\hat{q}_\star(S_t, A_t) \leftarrow \hat{q}_\star(S_t, A_t) + \alpha(G - \hat{q}_\star(S_t, A_t))$
 9:     end for
10: end for
```

Example: *Cliff Walking*



Let's compare SARSA to Q-Learning algorithm!

Example: *Cliff Walking*



Don't Mistake!

Q-learning collects less reward since it goes off-policy; however, it estimates optimal action-values: at some point it can start playing optimally

Convergence of SARSA

Recall: *GLIE Algorithms*

A GPI-type control loop is GLIE, if for any state-action pair (s, a), we have the following asymptotic properties

1. *The number of visits to all state-action pair grows large*

$$\lim_{K \to \infty} \mathcal{C}_K(s, a) = \infty$$

2. *The improved policy in last episode converges to greedy policy*

$$\lim_{K \to \infty} \pi_K(a^m | s) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_{\pi_K}(s, a^m) \\ 0 & m \neq \operatorname*{argmax}_m q_{\pi_K}(s, a^m) \end{cases}$$

GLIE control algorithms converge to optimal policy

Convergence of SARSA

+ *But do we really have large number of episodes with SARSA?*
– Not necessarily! We may have only one infinitely long trajectory
+ *What should we do then?*
– We can simply treat it as a large number of episodes of length 1

In (basic) SARSA, we only need one step in the trajectory

$$S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$$

We could hence think of it as one episode

1. *each time step t we update the action-values*
2. *each time step we improve the policy*

Convergence of SARSA

Modification: *GLIE Algorithms*

An online control loop is GLIE, if we have asymptotically in time t

1. *The number of visits to all state-action pair grows large*

$$\lim_{t\to\infty} \mathcal{C}_t\left(s, a\right) = \infty$$

2. *The improved policy converges to greedy policy*

$$\lim_{t\to\infty} \pi_t\left(a^m | s\right) = \begin{cases} 1 & m = \operatorname*{argmax}_m q_{\pi_t}\left(s, a^m\right) \\ 0 & m \neq \operatorname*{argmax}_m q_{\pi_t}\left(s, a^m\right) \end{cases}$$

Convergence of SARSA: *Make it GLIE*

+ *Can we guarantee that both conditions hold with SARSA?*
– The second one is easy: *we need to scale ϵ down with t, e.g.,* $\epsilon_t = 1/t$
+ *What about the first condition?*
– We should scale *the step-size α according to Robbins-Monro*

Robbins-Monro Sequence

Sequence α_t is Robbins-Monro if we have

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \text{and} \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

For instance, $\alpha_t = 1/t$ is a Robbins-Monro sequence

Convergence of SARSA

Convergence of SARSA

SARSA online control loop converges to the optimal action-values if

1. *Step-size is scheduled by a Robbins-Monro sequence*
2. *Exploration factor ϵ decays in time*

In practice however

- ϵ *is a hyperparameter*
  - ↳ *We know that we should schedule it*
  - ↳ *How we should do the scheduling? This is hyperparameter tuning*
- α *is a hyperparameter: some people call it learning rate*
  - ↳ *Its scheduling is again hyperparameter tuning*

Convergence of Q-Learning

Convergence of Q-Learning

Q-learning online control loop with exploration (non-zero ϵ) converges to the optimal action-values as $t \to \infty$

+ *That's it?*
- Yes!

Since we are evaluating off-policy, we don't care about behaving policy

Q-Learning vs SARSA

+ *So! Does it mean that Q-learning is always better?*
– Not always!

In general Q-learning has several benefits

- *Minimal convergence requirements*
- *It converges faster to the optimal policy*
  - ↳ *If we want to make SARSA that fast, we may get to a sub-optimal policy*
- *It has more flexibility and sample-efficiency*

But, SARSA also has some benefits

- *It is better suited for online control*
  - ↳ *Our behaving policy is the one going towards optimal one*
  - ↳ *In Q-learning, the behaving policy is not the optimal one*
- *It has lower complexity*
  - ↳ *We just deal with one policy*

End of Story!

Model-Based RL
- Bellman Equation
  - value iteration
  - policy iteration

Model-free RL
- on-policy methods
  - temporal difference
    - Monte Carlo
    - SARSA
- off-policy methods
  - Q-learning

I would strongly suggest to start with programming part of Assignment 2!

There you solve Froozen Lake with SARSA and Q-Learning