A Guided Tour of Chapter 12: Reinforcement Learning for Control

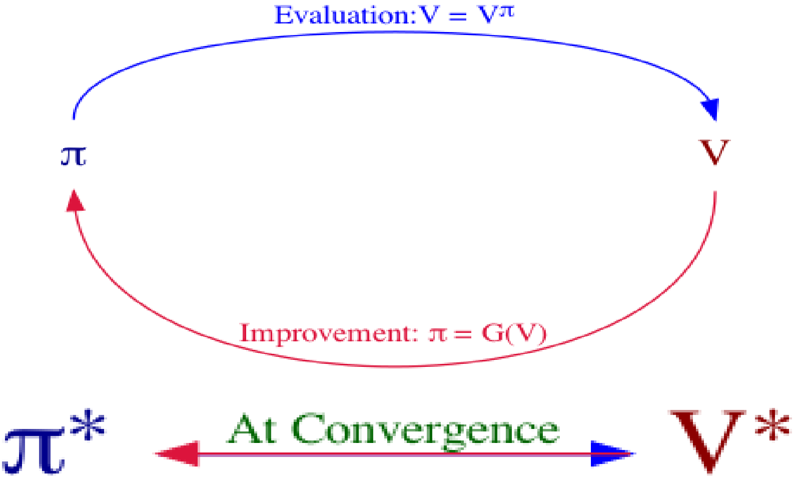
Ashwin Rao

ICME, Stanford University

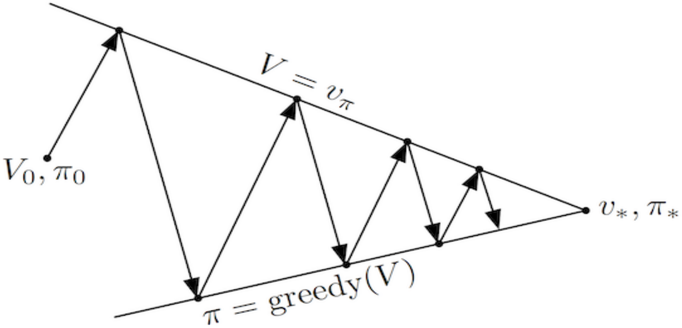
RL does not have access to a probability model

- DP/ADP assume access to probability model (knowledge of $\mathcal{P}_R$)
- Often in real-world, we do not have access to these probabilities
- Which means we'd need to *interact* with the *actual environment*
- Actual Environment serves up individual experiences, not probabilities
- Even if MDP model is available, model updates can be challenging
- Often real-world models end up being too large or too complex
- Sometimes estimating a *sampling model* is much more feasible
- So RL interacts with either *actual* or *simulated* environment
- Either way, we receive *individual experiences* of next state and reward
- We saw how RL Prediction learns from individual experiences
- Now we extend those ideas to RL Control: Learning Optimal VF

Let us recall the Policy Iteration algorithm

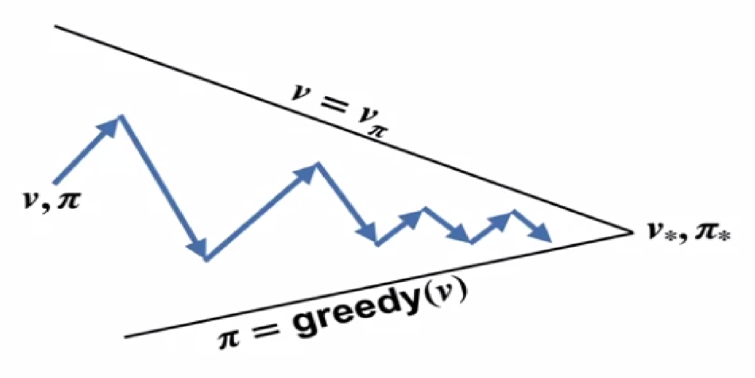


Policy Iteration



- Policy Evaluation estimates V^π, eg: Iterative Policy Evaluation
- Policy Improvement produces $\pi' \geq \pi$, eg: Greedy Policy Improvement
- Policy Evaluation and Policy Improvement alternate until convergence

The idea of Generalized Policy Iteration (GPI)



- *Any* Policy Evaluation method, *Any* Policy Improvement method
- For instance, *Partial* Policy Evaluation and *Partial* Policy Improvement

Natural Idea: GPI with Tabular Monte-Carlo Evaluation

- Let us explore GPI with Tabular Monte-Carlo evaluation
- So we will do Policy Evaluation with Tabular MC evaluation
- And we will do the usual Greedy Policy Improvement
- But Greedy Policy Improvement requires a model of MDP

$$\pi'_D(s) \leftarrow \arg\max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot V^\pi(s')\}$$

- However, it works if we were working with Action-Value Function

$$\pi'_D(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q^\pi(s, a)$$

- This means: Policy Evaluation for Action-Value Function $Q^\pi(s, a)$
- Following a policy π, update Q-value for each (S_t, A_t) each episode:

$$Count(S_t, A_t) \leftarrow Count(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{Count(S_t, A_t)} \cdot (G_t - Q(S_t, A_t))$$

ϵ-Greedy Policy Improvement

- A full Policy Evaluation with MC takes too long
- So we typically improve policy after each episode
- This can lead to some actions not being tried enough
- Which can lead to premature (greedy) domination of an action
- Which can lead to other actions getting "locked-out"
- Same as *Explore v/s Exploit* dilemma of Multi-Armed Bandit problem
- Simple solution: Perform an ϵ-Greedy Policy Improvement
- All $|\mathcal{A}|$ actions are tried with non-zero probability (for each state)
- Pick the greedy action with probability $1 - \epsilon$
- With probability ϵ, randomly choose one of the $|\mathcal{A}|$ actions

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

ϵ-Greedy improves the policy

Theorem

For a Finite MDP, if π is a policy such that for all $s \in \mathcal{N}, \pi(s,a) \geq \frac{\epsilon}{|\mathcal{A}|}$ for all $a \in \mathcal{A}$, then the ϵ-greedy policy π' obtained from Q^π is an improvement over π, i.e., $\boldsymbol{V}^{\pi'}(s) \geq \boldsymbol{V}^\pi(s)$ for all $s \in \mathcal{N}$.

- Applying $\boldsymbol{B}^{\pi'}$ repeatedly (starting with $\boldsymbol{V}^\pi$) converges to $\boldsymbol{V}^{\pi'}$:

$$\lim_{i \to \infty} (\boldsymbol{B}^{\pi'})^i(\boldsymbol{V}^\pi) = \boldsymbol{V}^{\pi'}$$

- So the proof is complete if we prove that:

$$(\boldsymbol{B}^{\pi'})^{i+1}(\boldsymbol{V}^\pi) \geq (\boldsymbol{B}^{\pi'})^i(\boldsymbol{V}^\pi) \text{ for all } i = 0, 1, 2, \ldots$$

- Non-decreasing sequence of Value Functions $[(\boldsymbol{B}^{\pi'})^i(\boldsymbol{V}^\pi) | i = 0, 1, 2, \ldots]$ with repeated applications of $\boldsymbol{B}^{\pi'}$

Base Case of Proof by Induction

The base case of proof by induction is to show that $\boldsymbol{B}^{\pi'}(\boldsymbol{V}^{\pi}) \geq \boldsymbol{V}^{\pi}$

$$\begin{aligned}
\boldsymbol{B}^{\pi'}(\boldsymbol{V}^{\pi})(s) &= (\boldsymbol{\mathcal{R}}^{\pi'} + \gamma \cdot \boldsymbol{\mathcal{P}}^{\pi'} \cdot \boldsymbol{V}^{\pi})(s) \\
&= \boldsymbol{\mathcal{R}}^{\pi'}(s) + \gamma \cdot \sum_{s' \in \mathcal{S}} \boldsymbol{\mathcal{P}}^{\pi'}(s, s') \cdot \boldsymbol{V}^{\pi}(s') \\
&= \sum_{a \in \mathcal{A}} \pi'(s, a) \cdot (\mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \cdot \boldsymbol{V}^{\pi}(s')) \\
&= \sum_{a \in \mathcal{A}} \pi'(s, a) \cdot Q^{\pi}(s, a) \\
&= \frac{\epsilon}{|\mathcal{A}|} \cdot \sum_{a \in \mathcal{A}} Q^{\pi}(s, a) + (1 - \epsilon) \cdot \max_{a \in \mathcal{A}} Q^{\pi}(s, a) \\
&\geq \frac{\epsilon}{|\mathcal{A}|} \cdot \sum_{a \in \mathcal{A}} Q^{\pi}(s, a) + (1 - \epsilon) \cdot \sum_{a \in \mathcal{A}} \frac{\pi(s, a) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} \cdot Q^{\pi}(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(s, a) \cdot Q^{\pi}(s, a) = \boldsymbol{V}^{\pi}(s)
\end{aligned}$$

Induction Step of Proof by Induction

Induction step is proved by monotonicity of $\boldsymbol{B}^\pi$ operator (for any π):

$$\text{Monotonicity Property of } \boldsymbol{B}^\pi : \boldsymbol{X} \geq \boldsymbol{Y} \Rightarrow \boldsymbol{B}^\pi(\boldsymbol{X}) \geq \boldsymbol{B}^\pi(\boldsymbol{Y})$$

$$\text{So } (\boldsymbol{B}^{\pi'})^{i+1}(\boldsymbol{V}^\pi) \geq (\boldsymbol{B}^{\pi'})^{i}(\boldsymbol{V}^\pi) \Rightarrow (\boldsymbol{B}^{\pi'})^{i+2}(\boldsymbol{V}^\pi) \geq (\boldsymbol{B}^{\pi'})^{i+1}(\boldsymbol{V}^\pi)$$

$\square$

GLIE

Definition

Greedy in the Limit with Infinite Exploration (GLIE):

- All state-action pairs are explored infinitely many times

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

- The policy converges to a greedy policy

$$\lim_{k \to \infty} \pi_k(s, a) = \mathbb{I}_{a = \arg\max_{b \in \mathcal{A}} Q(s,b)}$$

ϵ-greedy can be made GLIE if ϵ is reduced as: $\epsilon_k = \frac{1}{k}$

GLIE Tabular Monte-Carlo Control

- Sample k-th episode using π: $\{S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T\} \sim \pi$
- For each state S_t and action A_t in episode, updates at *episode-end*:

$$Count(S_t, A_t) \leftarrow Count(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{Count(S_t, A_t)} \cdot (G_t - Q(S_t, A_t))$$

- Improve policy at end of episode based on updated Q-Value function:

$$\epsilon \leftarrow \frac{1}{k}$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

Theorem

GLIE Tabular Monte-Carlo Control converges to the Optimal Action-Value function: $Q(s,a) \rightarrow Q^*(s,a)$.

GLIE MC Control with Function Approximation

$$\Delta \boldsymbol{w} = \alpha \cdot (G_t - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

```python
def glie_mc_control(
    mdp: MarkovDecisionProcess[S, A],
    states: NTStateDistribution[S],
    approx_0: QValueFunctionApprox[S, A],
    gamma: float,
    epsilon_func: Callable[[int], float],
    episode_len_tol: float = 1e-6
) -> Iterator[QValueFunctionApprox[S, A]]:
```

GLIE MC Control with Function Approximation

```
q: QValueFunctionApprox[S, A] = approx_0
p: Policy[S, A] = epsilon_greedy_policy(q, mdp, 1.0)
yield q
num_episodes: int = 0
while True:
  trace: Iterable[TransitionStep[S, A]] = \
    mdp.simulate_actions(states, p)
  num_episodes += 1
  for step in returns(trace, gamma, episode_len_tol):
    q = q.update([((step.state, step.action), step.ret
  p = epsilon_greedy_policy(
    q,
    mdp,
    epsilon_func(num_episodes)
  )
  yield q
```

GLIE MC Control with Function Approximation

```python
def epsilon_greedy_policy(
    q: QValueFunctionApprox[S, A],
    mdp: MarkovDecisionProcess[S, A],
    epsilon: float = 0.0
) -> Policy[S, A]:
    def explore(s: S, mdp=mdp) -> Iterable[A]:
        return mdp.actions(NonTerminal(s))
    return RandomPolicy(Categorical(
        {UniformPolicy(explore): epsilon,
         greedy_policy_from_qvf(q, mdp.actions):
             1 - epsilon}
    ))
```

GLIE MC Control with Function Approximation

```
def greedy_policy_from_qvf(
    q: QValueFunctionApprox[S, A],
    actions: Callable[[NonTerminal[S]], Iterable[A]]
) -> DeterministicPolicy[S, A]:
    def optimal_action(s: S) -> A:
        _, a = q.argmax((NonTerminal(s), a)
            for a in actions(NonTerminal(s)))
        return a
    return DeterministicPolicy(optimal_action)
```

MC versus TD Control

- TD learning has several advantages over MC learning:
  - Lower variance
  - Online
  - Can work with incomplete traces or continuing traces
  - Generic interface of `Iterable` of atomic experiences allows for serving up atomic experiences in any order (eg: atomic experience replays)
- So use TD instead of MC in our Control loop
  - Apply TD to $Q(S, A)$ (instead of $V(S)$)
  - Use ϵ-greedy Policy Improvement
  - Update $Q(S, A)$ after each *atomic experience*
  - ϵ-greedy policy **automatically updated** after each atomic experience

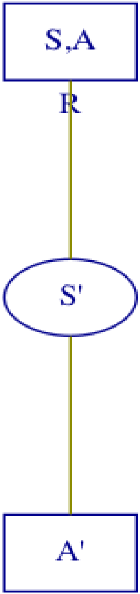
SARSA Algorithm

- SARSA is our first TD Control algorithm
- Like MC Control, Policy Improvement is ϵ-greedy
- But here Policy Evaluation is with a TD target, as below:

$$\Delta \boldsymbol{w} = \alpha \cdot (R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- Note that parameters $\boldsymbol{w}$ are updated after each atomic experience
- ϵ-greedy policy automatically updated after each atomic experience
- Action A_t is chosen from State S_t based on ϵ-greedy policy
- Action A_{t+1} is chosen from State S_{t+1} based on ϵ-greedy policy
- Unlike MC Control, trace experience is incrementally generated
- Note: Instead of ϵ-greedy, we could employ a more sophisticated exploratory policy derived from Q-value function (ϵ-greedy is just our default simple exploratory policy derived from Q-value function)

SARSA Visualization



Convergence of Tabular SARSA

Theorem

Tabular SARSA converges to the Optimal Action-Value function, $Q(s, a) \rightarrow Q^*(s, a)$, under the following conditions:

- *GLIE sequence of policies $\pi_t(s, a)$*
- *Robbins-Monro sequence of step-sizes α_t:*

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

n-step SARSA

- SARSA bootstraps the Q-Value Function with update:

$$\Delta \boldsymbol{w} = \alpha \cdot (R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- So it's natural to extend this to bootstrapping with 2 steps ahead:

$$\alpha \cdot (R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot Q(S_{t+2}, A_{t+2}; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- Generalize to bootstrapping with $n \geq 1$ time steps ahead:

$$\Delta \boldsymbol{w} = \alpha \cdot (G_{t,n} - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- $G_{t,n}$ (call it n-step bootstrapped return) is defined as:

$$\begin{aligned} G_{t,n} &= \sum_{i=t+1}^{t+n} \gamma^{i-t-1} \cdot R_i + \gamma^n \cdot Q(S_{t+n}, A_{t+n}; \boldsymbol{w}) \\ &= R_{t+1} + \gamma \cdot R_{t+2} + \ldots + \gamma^{n-1} \cdot R_{t+n} + \gamma^n \cdot Q(S_{t+n}, A_{t+n}; \boldsymbol{w}) \end{aligned}$$

λ-Return SARSA

- Instead of $G_{t,n}$, a valid target is a weighted-average target:

$$\sum_{n=1}^{N} u_n \cdot G_{t,n} + u \cdot G_t \text{ where } u + \sum_{n=1}^{N} u_n = 1$$

- Any of the u_n or u can be 0, as long as they all sum up to 1
- The λ-Return target is a special case of weights u_n and u

$$u_n = (1 - \lambda) \cdot \lambda^{n-1} \text{ for all } n = 1, \ldots, T - t - 1$$

$$u_n = 0 \text{ for all } n \geq T - t \text{ and } u = \lambda^{T-t-1}$$

- We denote the λ-Return target as $G_t^{(\lambda)}$, defined as:

$$G_t^{(\lambda)} = (1 - \lambda) \cdot \sum_{n=1}^{T-t-1} \lambda^{n-1} \cdot G_{t,n} + \lambda^{T-t-1} \cdot G_t$$

$$\Delta \boldsymbol{w} = \alpha \cdot (G_t^{(\lambda)} - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

SARSA(λ)

- λ can be tuned from SARSA ($\lambda = 0$) to MC Control ($\lambda = 1$)
- Note that for $\lambda > 0$, λ-Return SARSA is an Offline Algorithm
- SARSA(λ) is online "version" of λ-Return SARSA
- Similar to TD(λ) for Prediction, SARSA(λ) uses Eligibility Traces
- Eligibility Traces $\boldsymbol{E}_t$ for a given trace experience at time t defined as:

$$\boldsymbol{E}_0 = \nabla_{\boldsymbol{w}} Q(S_0, A_0; \boldsymbol{w})$$

$$\boldsymbol{E}_t = \gamma\lambda \cdot \boldsymbol{E}_{t-1} + \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- SARSA(λ) performs following update at each time step t:

$$\Delta \boldsymbol{w} = \alpha \cdot (R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})) \cdot \boldsymbol{E}_t$$

Off-Policy Learning

- Tension in Control Algorithms: Wanting to learn Q-Values contingent on *subsequent optimal behavior* versus wanting to explore all actions
- So separate these concerns into two different policies
- Estimate VF for *target policy* π while following *behavior policy* μ

$$\{S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T\} \sim \mu$$

- Behavior Policy meant to explore to collect data for all actions
- Target Policy is the policy to learn (driving towards Optimal Policy)
- Why is this important?
    - Learning from observing humans or other agents
    - Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
    - Learn about *optimal* policy while following *exploratory* policy
    - Learn about *multiple* policies while following *one* policy

Off-Policy Control with Q-Learning

- Q-Learning performs Off-Policy learning of action-values $Q(s, a; \boldsymbol{w})$
- The current action is chosen using behavior policy: $A_t \sim \mu(S_t, \cdot)$
- The next action is chosen using target policy: $A' \sim \pi(S_{t+1}, \cdot)$
- Update $Q(S_t, A_t; \boldsymbol{w})$ towards value of *targeted action*

$$\Delta \boldsymbol{w} = \alpha \cdot (R_{t+1} + \gamma \cdot Q(S_{t+1}, A'; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

Q-Learning

- We now allow both behavior and target policies to **improve**
- The target (deterministic) policy π_D is **greedy** w.r.t Q-Value Function

$$\pi_D(S_{t+1}) = \arg\max_{a' \in \mathcal{A}} Q(S_{t+1}, a'; \boldsymbol{w})$$

- The behavior policy μ is also improving, eg: ϵ-greedy w.r.t. Q
- The Q-learning target then simplifies to:

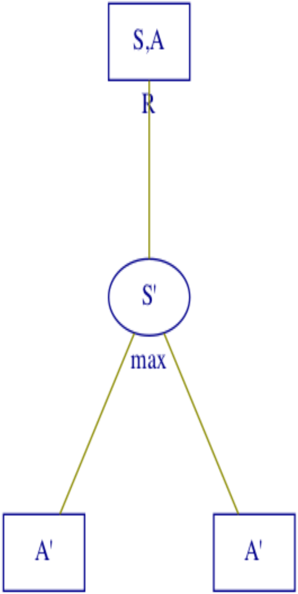
$$\begin{aligned}
& R_{t+1} + \gamma \cdot Q(S_{t+1}, A'; \boldsymbol{w}) \\
=& R_{t+1} + \gamma \cdot Q(S_{t+1}, \arg\max_{a' \in \mathcal{A}} Q(S_{t+1}, a'; \boldsymbol{w})) \\
=& R_{t+1} + \gamma \cdot \max_{a' \in \mathcal{A}} Q(S_{t+1}, a'; \boldsymbol{w})
\end{aligned}$$

- Thus the update to Q-Value Function after each atomic experience is:

$$\Delta \boldsymbol{w} = \alpha \cdot (R_{t+1} + \gamma \cdot \max_{a' \in \mathcal{A}} Q(S_{t+1}, a'; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w})) \cdot \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- Tabular convergence proofs require infinite exploration of all (s, a) pairs & appropriate stochastic approximation conditions for step sizes.

Q-Learning Control Visualization



Importance Sampling for Off-Policy Learning

- Importance Sampling refers to methods to estimate properties of a distribution P, given access to samples from a different distribution Q
- We can calculate $\mathbb{E}_{X \sim P}[f(X)]$ given samples from Q as follows:

$$\begin{aligned}
\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X) \cdot f(X) \\
&= \sum Q(X) \cdot \frac{P(X)}{Q(X)} \cdot f(X) \\
&= \mathbb{E}_{X \sim Q}[\frac{P(X)}{Q(X)} \cdot f(X)]
\end{aligned}$$

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from μ to estimate Value Function for π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(S_t, A_t)}{\mu(S_t, A_t)} \cdot \frac{\pi(S_{t+1}, A_{t+1})}{\mu(S_{t+1}, A_{t+1})} \cdots \frac{\pi(S_{T-1}, A_{T-1})}{\mu(S_{T-1}, A_{T-1})} \cdot G_t$$

- Update value towards *corrected* return

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot (G_t^{\pi/\mu} - V(S_t))$$

- Likewise for Q-Value Function for MC Control
- Note: We cannot use this method if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance

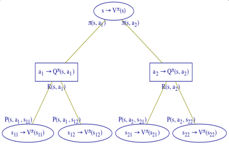
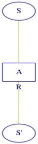
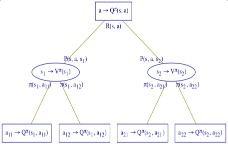
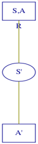
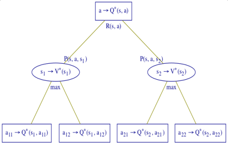
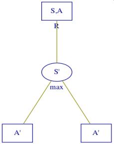
Importance Sampling for Off-Policy Temporal-Difference

- Use TD targets generated from μ to evaluate Value Function for π
- Weight TD target $R + \gamma \cdot V(S')$ with importance sampling
- Here we only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot (\frac{\pi(S_t, A_t)}{\mu(S_t, A_t)} \cdot (R_{t+1} + \gamma \cdot V(S_{t+1})) - V(S_t))$$

- Likewise for Q-Value Function for TD Control
- This has much lower variance than MC importance sampling
- Policies only need to be similar over a single step

Relationship between DP and TD

	Full Backup (DP)	**Sample Backup (TD)**
Bellman Expectation Equation for $V^\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $Q^\pi(s, a)$	 Q-Policy Iteration	 SARSA
Bellman Optimality Equation for $Q^*(s, a)$	 Q-Value Iteration	 Q-Learning

Relationship between DP and TD

Full Backup (DP)	Sample Backup (TD)
Iterative Policy Evaluation: $V(S)$ update $\mathbb{E}[R + \gamma V(S') \mid S]$	TD Learning: $V(S)$ update sample $R + \gamma V(S')$
Q-Policy Iteration: $Q(S, A)$ update $\mathbb{E}[R + \gamma Q(S', A') \mid S, A]$	SARSA: $Q(S, A)$ update sample $R + \gamma Q(S', A')$
Q-Value Iteration: $Q(S, A)$ update $\mathbb{E}[R + \gamma \max_{a'} Q(S', a') \mid S, A]$	Q-Learning: $Q(S, A)$ update sample $R + \gamma \max_{a'} Q(S', a')$

Convergence of Prediction Algorithms

On/Off Policy	Algorithm	Tabular	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Deadly Triad := [Bootstrapping, Function Approximation, Off-Policy]

Gradient Temporal-Difference Learning

- TD does not follow the gradient of *any* objective function
- This is why TD can diverge:
  - when running off-policy, or
  - when using non-linear function approximation
- **Gradient TD** follows true gradient of projected Bellman Error

On/Off Policy	Algorithm	Tabular	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

Convergence of Control Algorithms

Algorithm	Tabular	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-Learning	✓	✗	✗
Gradient Q-Learning	✓	✓	✗

(✓) means it chatters around near-optimal Value Function

Key Takeaways from this Chapter

- RL Control is based on the idea of Generalized Policy Iteration (GPI)
  - Policy Evaluation with Q-Value Function (instead of V)
  - Improved Policy needs to be exploratory, eg: ϵ-greedy
- On-Policy versus Off-Policy
- Deadly Triad := [Bootstrapping, Function Approximation, Off-Policy]