Sequential Decision Making and Reinforcement Learning

(SDMRL)

Model-free RL

Sarah Keren

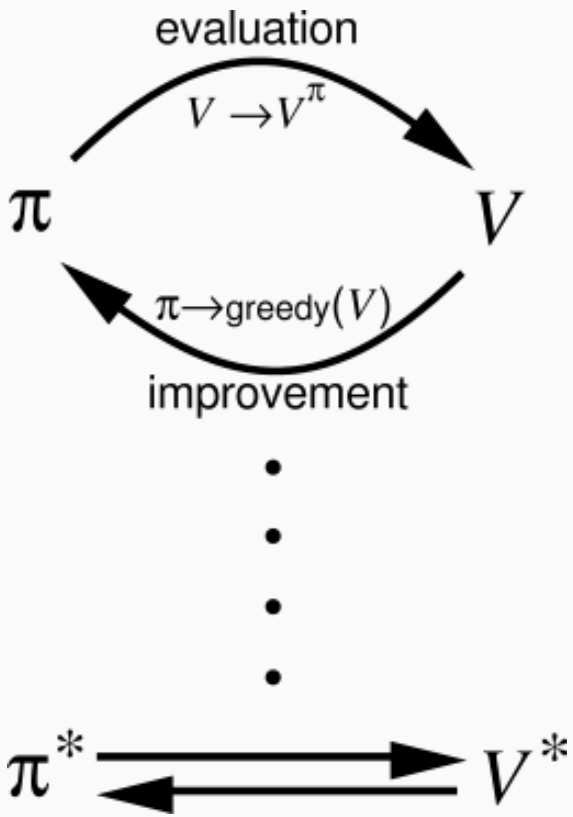
The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Acknowledgements

- David Sliver's course on RL: *https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver*
- Slides by Malte Helmert, Carmel Domshlak, Erez Karpas and Alexander Shleyfman.

Recap

Anatomy of RL Algorithms



Model-Free vs. Model-Based RL

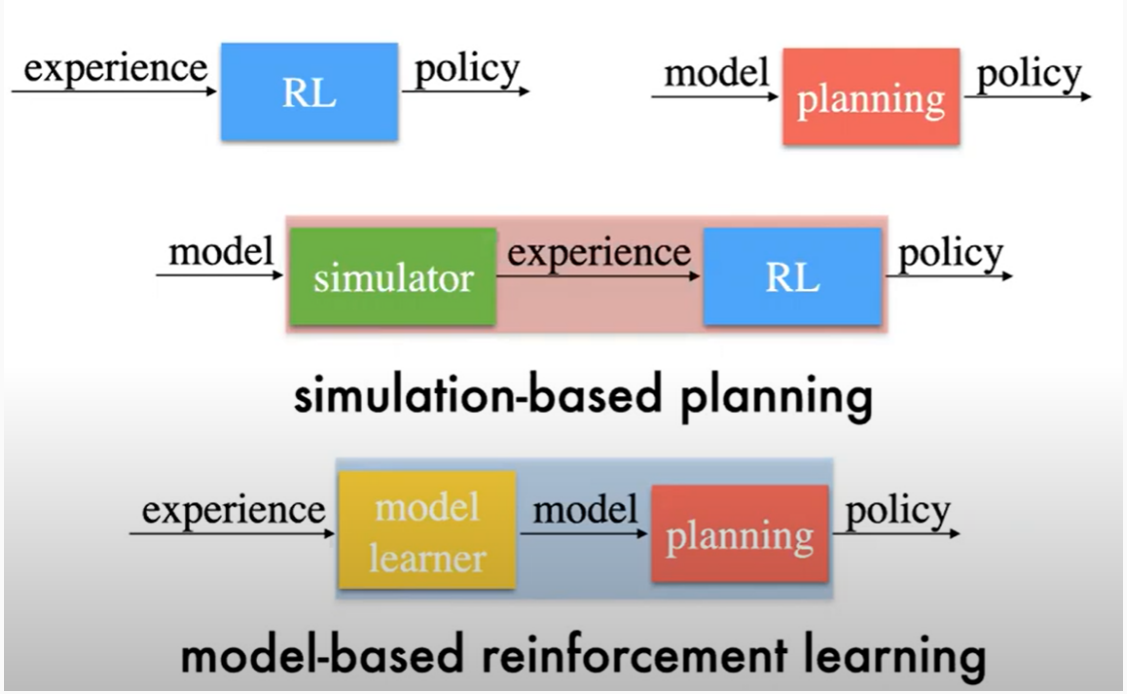


Figure 1: By Michael Littman

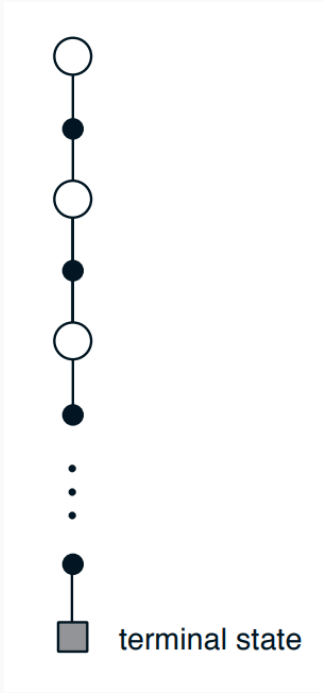
https://www.youtube.com/watch?v=45FKxa3qPHo

Model-Free vs. Model-Based RL

Model Free RL: Monte Carlo

Monte Carlo (MC)

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes: no bootstrapping.
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs: all episodes must terminate



Monte Carlo Simulation-Rollout

- A Monte Carlo simulation is a statistical technique used to analyze the behavior of complex systems and processes that are influenced by random variables.

Monte Carlo Simulation-Rollout

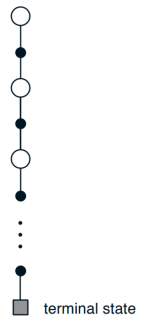
- A Monte Carlo simulation is a statistical technique used to analyze the behavior of complex systems and processes that are influenced by random variables.
- Each rollout involves simulating a sequence of actions, typically by selecting actions according to some policy (e.g., uniformly at random, or based on a heuristic) until a terminal state is reached.

Monte Carlo Simulation-Rollout

- A Monte Carlo simulation is a statistical technique used to analyze the behavior of complex systems and processes that are influenced by random variables.
- Each rollout involves simulating a sequence of actions, typically by selecting actions according to some policy (e.g., uniformly at random, or based on a heuristic) until a terminal state is reached.
- Typically, a large number of simulations are used to estimate the probabilities and distributions of different outcomes.

Monte Carlo Simulation-Rollout

- A Monte Carlo simulation is a statistical technique used to analyze the behavior of complex systems and processes that are influenced by random variables.
- Each rollout involves simulating a sequence of actions, typically by selecting actions according to some policy (e.g., uniformly at random, or based on a heuristic) until a terminal state is reached.
- Typically, a large number of simulations are used to estimate the probabilities and distributions of different outcomes.
- We will use this in different algorithms



Monte-Carlo Policy Evaluation

- **Reminder**:
  - Return is (typically) the total discounted reward:

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

  - Value function is (typically) the expected return:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- **Objective:** learn V_π from episodes of experience under policy π

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- Estimate $V_\pi(s)$ as average total reward of epochs containing s (calculating from s to end of epoch).

Monte-Carlo Policy Evaluation

- **Reminder**:
  - Return is (typically) the total discounted reward:

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

  - Value function is (typically) the expected return:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- **Objective:** learn V_π from episodes of experience under policy π

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

- Estimate $V_\pi(s)$ as average total reward of epochs containing s (calculating from s to end of epoch).

Monte-Carlo policy evaluation uses empirical mean return instead of expected return. Why ?

Monte-Carlo Policy Evaluation

Key Idea

Use observed reward-to-go of the state as the direct evidence of the actual expected utility of that state.

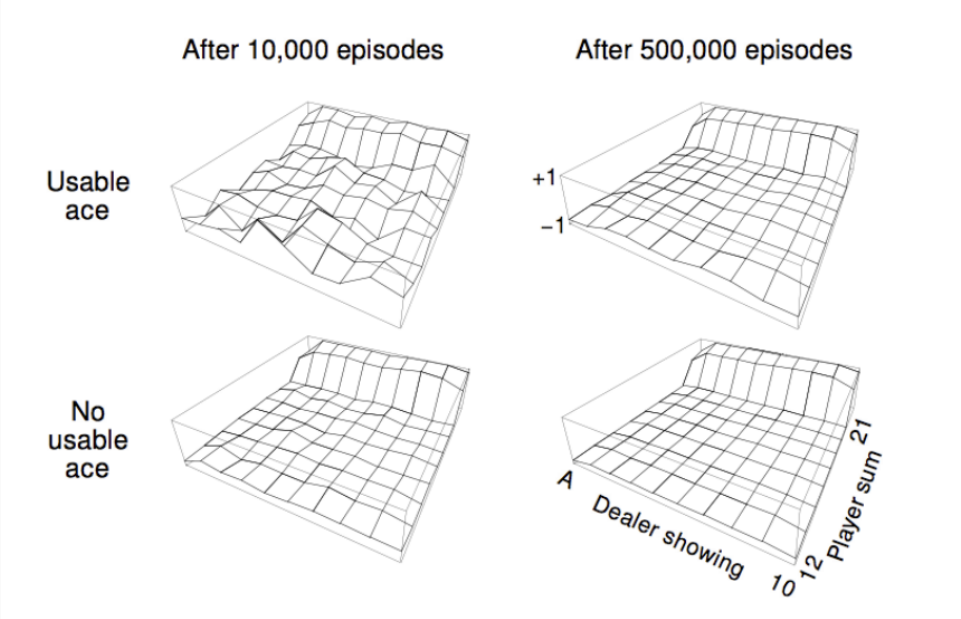
- Reward-to-go of a state s = the sum of the (discounted) rewards from that state until a terminal state is reached
- Two versions to evaluate state s:
  - The first time-step t that state s is visited in an episode
  - Every time-step t that state s is visited in an episode
- Increment visit counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$ (averaging the reward-to-go samples will converge to true value at state)

Monte-Carlo Policy Evaluation - Blackjack

- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action stick: Stop receiving cards (and terminate)
- Action twist: Take another card (no replacement)
- Reward for stick:
  - +1 if sum of cards > sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards < sum of dealer cards
- Reward for twist:
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
- Transitions: automatically twist if sum of cards < 12

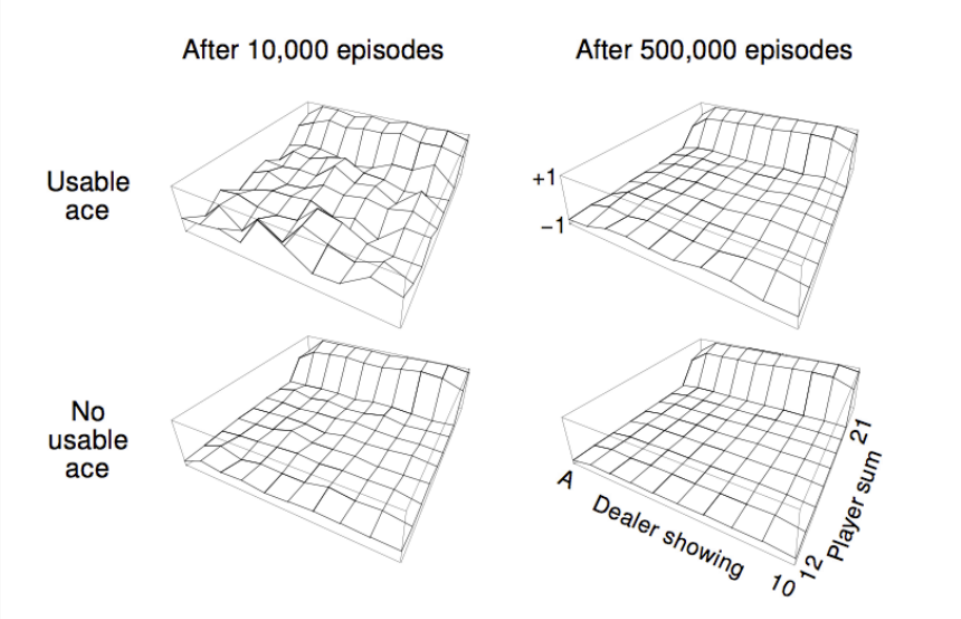
Monte-Carlo Policy Evaluation

Blackjack Value Function after Monte-Carlo Learning



Monte-Carlo Policy Evaluation

Blackjack Value Function after Monte-Carlo Learning



Policy - stick if sum of cards ≥ 20, otherwise twist.

Monte-Carlo: Prediction (Evaluation)

Prediction:

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π

For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

Monte-Carlo: Prediction (Evaluation)

Prediction:

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π

For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

How can we use this for control ?

Monte-Carlo: From Prediction (Evaluation) to Control

Control:

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
- $Q(s,a) \leftarrow$ arbitrary
- $\pi(s) \leftarrow$ arbitrary
- $Returns(s,a) \leftarrow$ empty list

Repeat forever:
- Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
- Generate an episode starting from S_0, A_0, following π
- For each pair s, a appearing in the episode:
  - $G \leftarrow$ return following the first occurrence of s, a
  - Append G to $Returns(s,a)$
  - $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
- For each s in the episode:
  - $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Monte-Carlo: From Prediction (Evaluation) to Control

Control:

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
- $Q(s,a) \leftarrow$ arbitrary
- $\pi(s) \leftarrow$ arbitrary
- $Returns(s,a) \leftarrow$ empty list

Repeat forever:
- Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
- Generate an episode starting from S_0, A_0, following π
- For each pair s, a appearing in the episode:
  - $G \leftarrow$ return following the first occurrence of s, a
  - Append G to $Returns(s,a)$
  - $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
- For each s in the episode:
  - $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Problem with this approach?

Monte-Carlo Policy Contorl - Taxi

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | :|
|Y| : |B: |
+---------+
  (North)
num episodes completed:   1
total rewards:            -133
mean rewards per episode: -133.00
```

```python
def build_decision_dict(raw_data):
    # Nested dictionary for: State -> Action -> Reward List
    state_action_scores = defaultdict(lambda: defaultdict(lambda: []))
    for trajectory in raw_data:
        reward_sum = 0
        # iterate backwards to calculate the return G of each observed state action pair
        for state, action, reward in reversed(list(zip(trajectory.observations, trajectory.actions, trajectory.rewards))):
            reward_sum += reward
            state_action_scores[state][action].append(reward_sum)

    for state, action_values in state_action_scores.items():
        for action, values_list in action_values.items():
            # Calculate the mean of all returns for a state action pair
            state_action_scores[state][action] = np.mean(values_list)
        # For each state choose the action with the highest mean return
        state_action_scores[state] = max(state_action_scores[state], key=state_action_scores[state].get)
    return state_action_scores
```

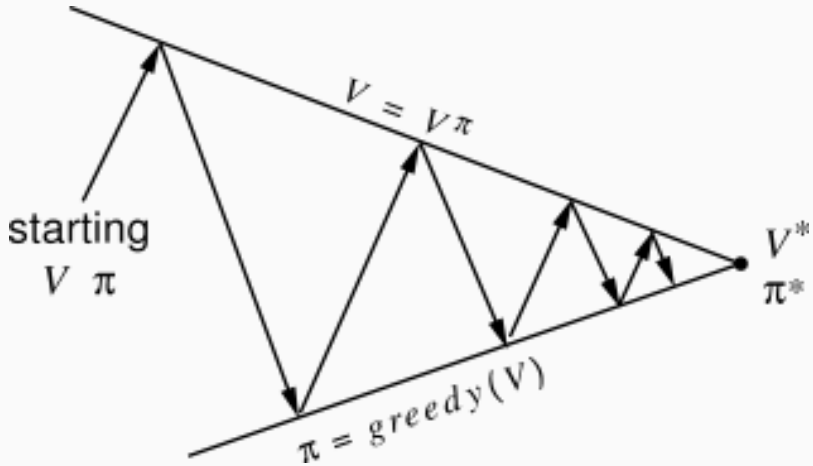
Monte-Carlo Control - Taxi

```
env.seed(seed)

policy = calc_final_policy(RandomTraveler, 100, "mcc_100")
evaluate_and_print(policy)
policy = calc_final_policy(RandomTraveler, 1000, "mcc_1000")
evaluate_and_print(policy)
policy = calc_final_policy(RandomTraveler, 10000, "mcc_10000")
evaluate_and_print(policy)
policy = calc_final_policy(RandomTraveler, 100000, "mcc_100000")
evaluate_and_print(policy)
```

```
100% ████████████████████████████████ 10000/10000 [00:01<00:00, 5139.08it/s]
Monte Carlo Control Policy
-----------------
total reward over all episodes: -1306708
mean reward per episode:        -130.6708
100% ████████████████████████████████ 10000/10000 [00:01<00:00, 5334.19it/s]
Monte Carlo Control Policy
-----------------
total reward over all episodes: -1242335
mean reward per episode:        -124.2335
100% ████████████████████████████████ 10000/10000 [00:01<00:00, 6415.93it/s]
Monte Carlo Control Policy
-----------------
total reward over all episodes: -729155
mean reward per episode:        -72.9155
100% ████████████████████████████████ 10000/10000 [00:01<00:00, 7840.29it/s]
Monte Carlo Control Policy
-----------------
total reward over all episodes: -154428
mean reward per episode:        -15.4428
```

Generalised Policy Iteration With Monte-Carlo Evaluation



Policy evaluation: Monte-Carlo policy evaluation, $V = \mathcal{V}_\pi$?
Policy improvement: Greedy policy improvement?

Model-Free Policy Iteration Using Action-Value Function

$$\pi'(s) = \arg\max_{a \in A} R_s^a + \mathcal{P}_{ss'}^a V(s')$$

$$\pi'(s) = \arg\max_{a \in A} Q(s, a)$$

Model-Free Policy Iteration Using Action-Value Function

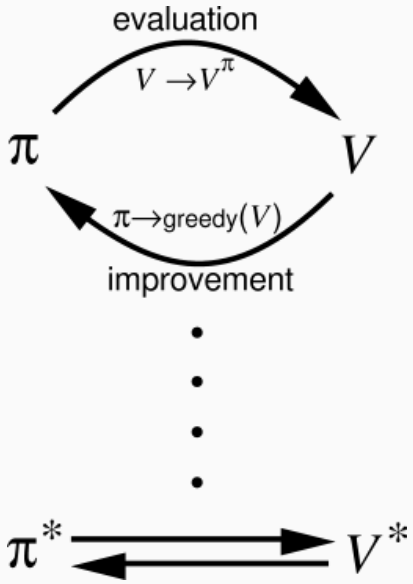
Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \arg\max_{a \in A} R_s^a + \mathcal{P}_{ss'}^a V(s')$$

Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \arg\max_{a \in A} Q(s, a)$$

Generalised Policy Iteration With Monte-Carlo Evaluation



- **Policy evaluation:** Monte-Carlo policy evaluation. $Q_\pi(s, a)$
- **Policy improvement:** Greedy policy improvement. How?

Example of Greedy Action Selection

There are two doors in front of you:
You open the left door and get reward 0
$V(left) = 0$
You open the right door and get reward +1
$V(right) = +1$
You open the right door and get reward +2
$V(right) = +2$
You open the right door and get reward +2
$V(right) = +2$
.
.
.
Are you sure you've chosen the best door?

ϵ-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

Policy Improvement Theorem*

Theorem

Let π and π' be any pair of deterministic policies such that for all $s \in \mathcal{S}$ $Q_\pi(s, \pi'(s)) \geq \mathcal{V}_\pi(s)$ then for all $s \in \mathcal{S}$, $\mathcal{V}_{\pi'}(s) \geq \mathcal{V}_\pi(s)$.

π', that can either exploit or explore, must be at least as good as π

ϵ-Greedy Policy Improvement*

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q_π is an improvement, $V_{\pi'}(s) \geq V_\pi(s)$

$$Q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) Q_\pi(s, a)$$

ϵ-Greedy Policy Improvement*

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q_π is an improvement, $V_{\pi'}(s) \geq V_\pi(s)$

$$Q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) Q_\pi(s, a)$$

$$= \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q_\pi(s, a)$$

ϵ-Greedy Policy Improvement*

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q_π is an improvement, $V_{\pi'}(s) \geq V_\pi(s)$

$$Q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) Q_\pi(s, a)$$

$$= \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q_\pi(s, a)$$

$$\geq \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1 - \epsilon} Q_\pi(s, a)$$

ϵ-Greedy Policy Improvement*

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q_π is an improvement, $V_{\pi'}(s) \geq V_\pi(s)$

$$Q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s) Q_\pi(s, a)$$

$$= \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q_\pi(s, a)$$

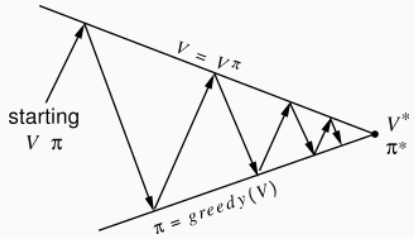
$$\geq \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1 - \epsilon} Q_\pi(s, a)$$

$$= \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a) = V_\pi(s)$$

Where $m = |\mathcal{A}|$.

Therefore from policy improvement theorem, $V_{\pi'}(s) \geq V_\pi(s)$

Monte-Carlo Policy Iteration



- **Policy evaluation:** Monte-Carlo policy evaluation. $Q_\pi(s, a)$
- Policy improvement: ϵ-greedy improvement

Is it necessary to wait until the end of execution of all trajectories ?

Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally,

$$\mu_k = \frac{1}{k} \sum_{j=1}^{k} x_j$$

$$= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right)$$

$$= \frac{1}{k} \left(x_k + (k-1)\mu_{k-1} \right)$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t:

$$N(S_t) \leftarrow N(S_t) + 1$$

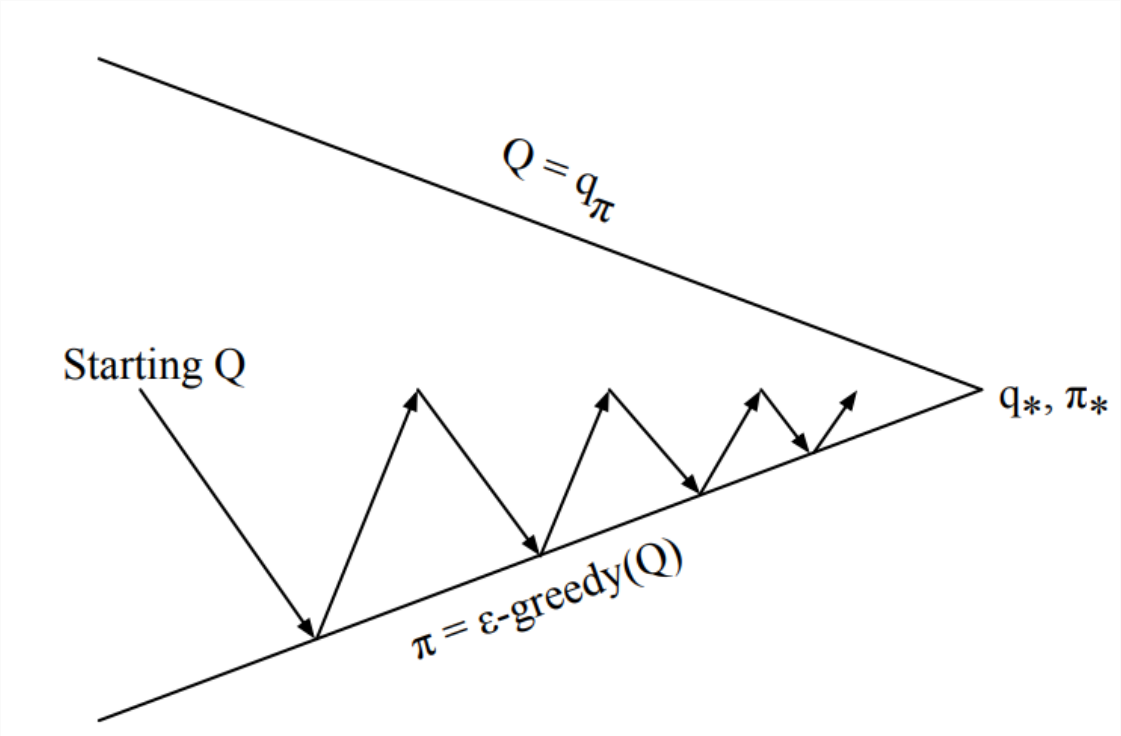
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

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

How do we set α a.k.a **the learning rate ?**

Monte-Carlo Control



Every Iteration:

- **Policy evaluation:** Monte-Carlo policy evaluation. $Q \approx q_\pi$
- Policy improvement: ϵ-greedy improvement

Monte-Carlo Control

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Will this converge to an optimal policy?

How do we make sure exploration eventually stops?

How do we make sure exploration eventually stops?

- Decay ϵ Over Time: e.g., exponential decay $\epsilon_t = \gamma \cdot \epsilon_{t-1}$ with $\gamma \in (0, 1)$.
- After a sufficient number of iterations (when $Q(s, a)$ has stabilized), switch to a purely greedy policy.

GLIE

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 on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \arg\max_{a' \in \mathcal{A}} Q_k(s, a'))$$

For example, ϵ-greedy is GLIE if α reduces to zero at $\epsilon_k = \frac{1}{k}$.

GLIE Monte-Carlo Control

- Sample kth episode using $\pi : \{S_1, A_1, R_2, ..., S_T\} \sim \pi$
- For each state S_t and action A_t in the episode,

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

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

- Improve policy based on new action-value function

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

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

Theorem

GLIE Monte-Carlo control converges to the optimal action-value function, $Q(s, a) \rightarrow Q^*(s, a)$

Monte-Carlo Learning

- Converge very slowly to correct utilities values (requires a lot of sequences)

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Doesn't exploit Bellman constraints on policy values

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s')$$

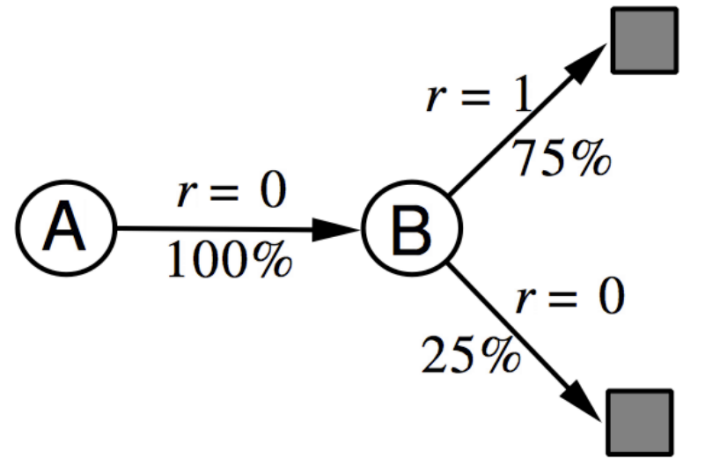
- Can consider estimates that violate this property badly
- How can we incorporate such constraints ?

Temporal-Difference Learning

AB Example

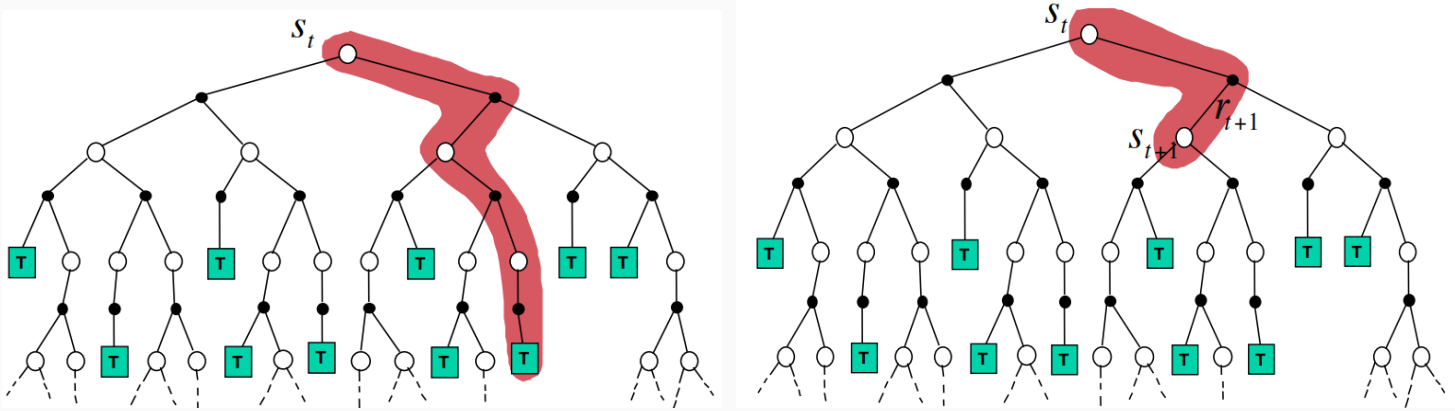
Two states A, B; no discounting; 8 episodes of experience

- A, 0, B, 0
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 0



What is $V(A)$, $V(B)$?

From Monte Carlo to Temporal-Difference (TD) Learning



Temporal-Difference Learning

Do local updates of utility/value function on a per-action basis.

- TD methods learn directly from episodes of experience
- TD is model-free: no learning of MDP transitions / rewards (doesn't try to estimate entire transition function).
- TD learns from incomplete episodes, by **bootstrapping** - updates a guess towards a guess.

From MC to TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step.

Temporal-Difference Learning

- For each transition from s to s', we perform the following update

$$V_\pi(s) := V_\pi(s) + \alpha(R(s) + \gamma V_\pi(s') - V_\pi(s))$$

with α as the learning rate

How does this move us closer to satisfying the Bellman constraint ?

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s')$$

Temporal-Difference Learning

- For each transition from s to s', we perform the following update

$$V_\pi(s) := V_\pi(s) + \alpha(R(s) + \gamma V_\pi(s') - V_\pi(s))$$

with α as the learning rate

How does this move us closer to satisfying the Bellman constraint ?

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s')$$

- $R(s) + \gamma V_\pi(s')$ is a (noisy) sample of utility based on next state.
- The update is maintaining a "mean" of (noisy) utility sample
- How do we guarantee convergence to true value ?

Temporal-Difference Learning

- For each transition from s to s', we perform the following update

$$V_\pi(s) := V_\pi(s) + \alpha(R(s) + \gamma V_\pi(s') - V_\pi(s))$$

with α as the learning rate

How does this move us closer to satisfying the Bellman constraint ?

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s')$$

- $R(s) + \gamma V_\pi(s')$ is a (noisy) sample of utility based on next state.
- The update is maintaining a "mean" of (noisy) utility sample
- How do we guarantee convergence to true value ?
  - If the learning rate decreases appropriately with the number of samples (e.g. $\frac{1}{n}$), then the utility estimates will converge to true values (non-trivial).

MC and TD

- Goal: learn and optimize value function online from experience
- Incremental every-visit Monte-Carlo
  - Update value $V(s_t)$ toward actual return G_t

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

- Simplest temporal-difference learning algorithm: $TD(0)$
  - Update value $V(s_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

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

  - $R + \gamma V(S_{t+1})$ is called the **TD target**
  - $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

MC and TD

- Simplest temporal-difference learning algorithm: $TD(0)$
  - Update value $V(s_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

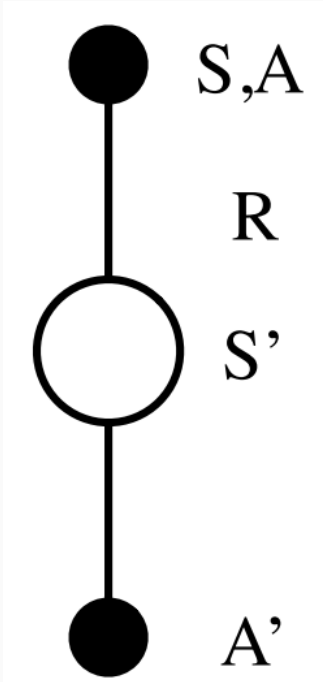
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- If we are using Q

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

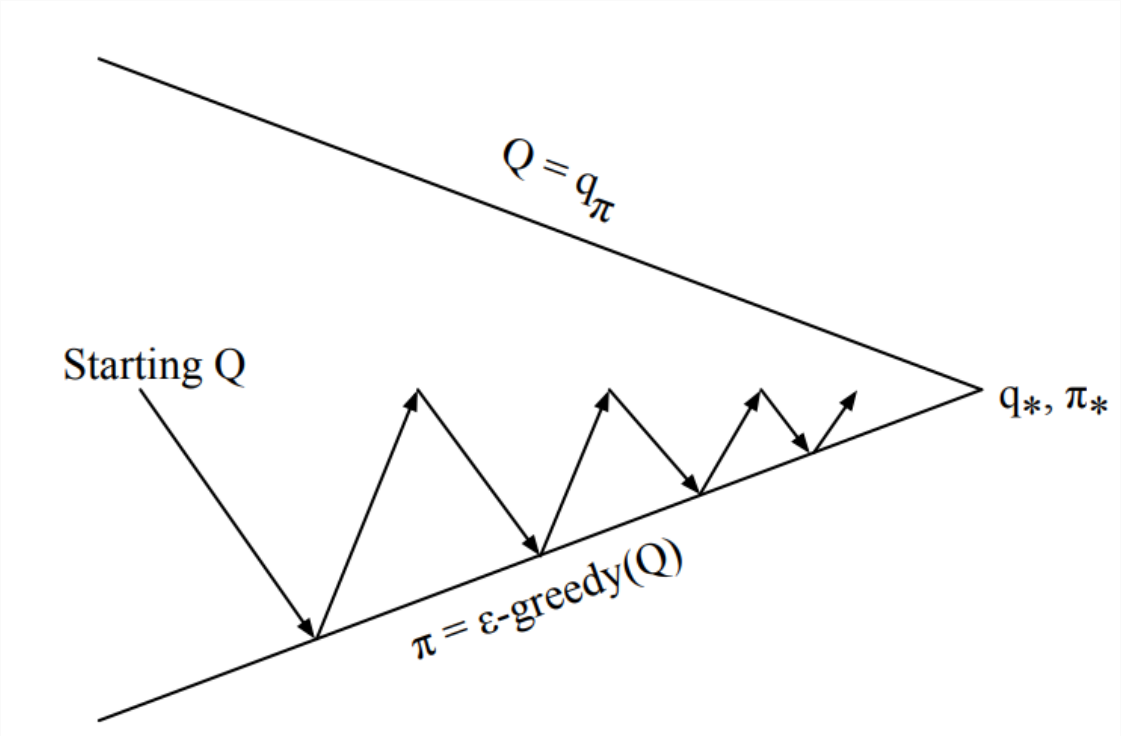
TD methods: SARSA

Updating Action-Value Functions with



$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

On-Policy Control With SARSA



Every time-step:

- Policy evaluation: Sarsa $Q \approx q_\pi$
- Policy improvement: ϵ-greedy improvement

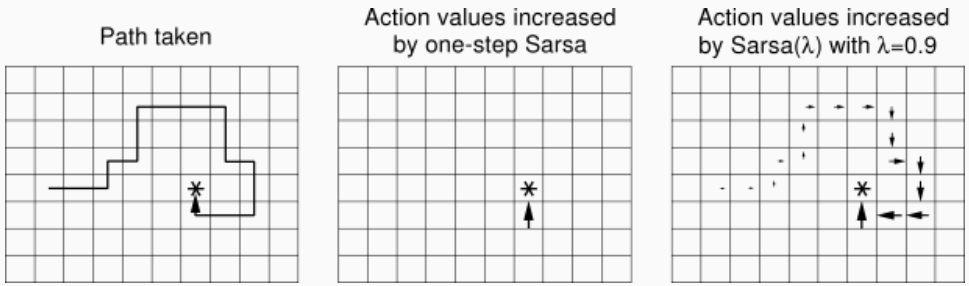
SARSA

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

Additional topics:

- N-Step Sarsa
- Sarsa(λ)
- Forward and backward view Sarsa
- Convergence proof



Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $\mathcal{V}_\pi(s)$ or $Q_\pi(s,a)$ while following behaviour policy $\mu(a|s)$

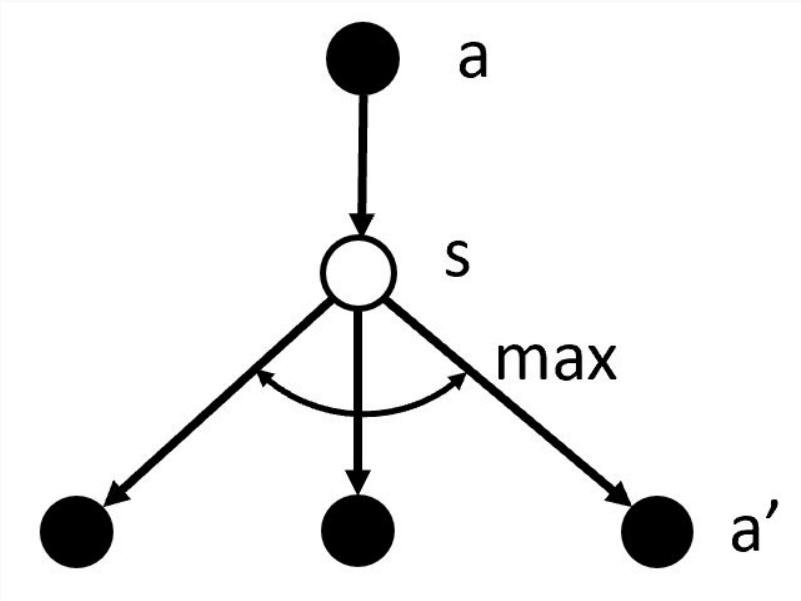
$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

- Why is this important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t1}$
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(|S_t)$
- But we consider alternative successor action $A' \sim \pi(|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S', A') - Q(S_t, A_t))$$



Off-Policy Control with Q-Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \arg\max_{a'} Q(St + 1, a')$$

- The behaviour policy μ is (e.g.) ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$R_{t+1} + \gamma Q(S_{t+1}, A')$$

$$= R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a'))$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

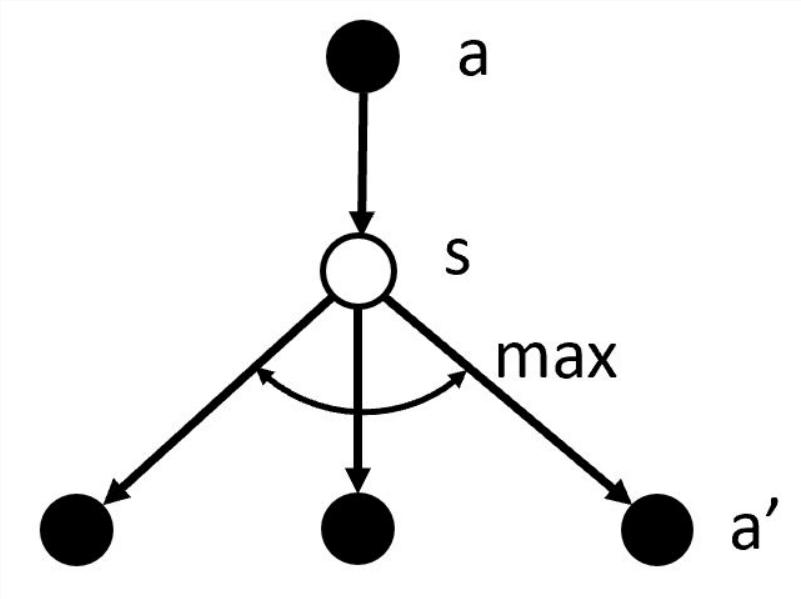
Q-Learning

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

Q-Learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S', A') - Q(S_t, A_t))$$



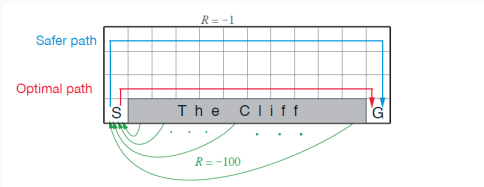
Theorem

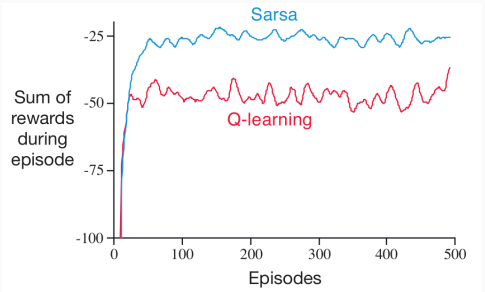
Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow Q*(s, a)$

SARSA vs. Q-Learning

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal





Importance Sampling

Estimate the expectation of a different distribution:

$$\mathbb{E}_{X\sim P(X)}[f(x)] = \sum P(X)f(x)$$

$$= \sum Q(X)\frac{P(X)}{Q(X)}f(x)$$

$$= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(x)\right]$$

Importance Sampling for Off-Policy Monte-Carlo

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

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)\pi(A_{t+1}|S_{t+1})}{\mu(A_t|S_t)\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards corrected return

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

- Cannot use if μ is zero when π is non-zero
- Importance sampling can dramatically increase variance

Importance Sampling for Off-Policy TD

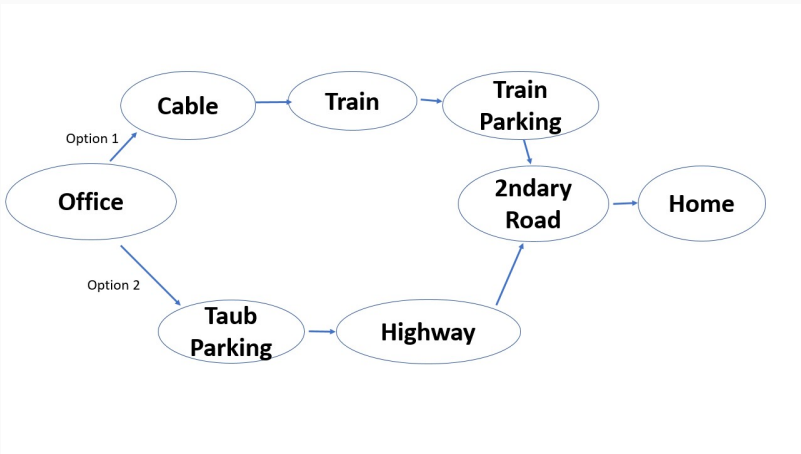
- Use TD targets generated from μ to evaluate π
- Weight TD target $R + \gamma \mathcal{V}(S')$ by importance sampling
- Only need a single importance sampling correction

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

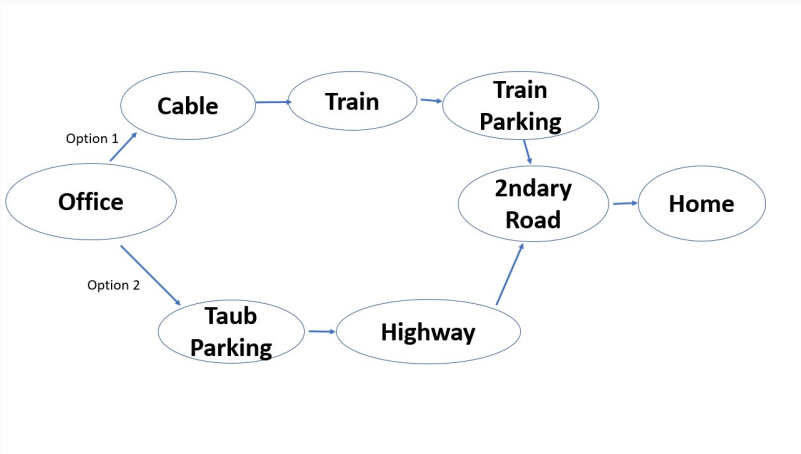
- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Comparing Monte Carlo and TD methods

Going Home Example



Going Home Example

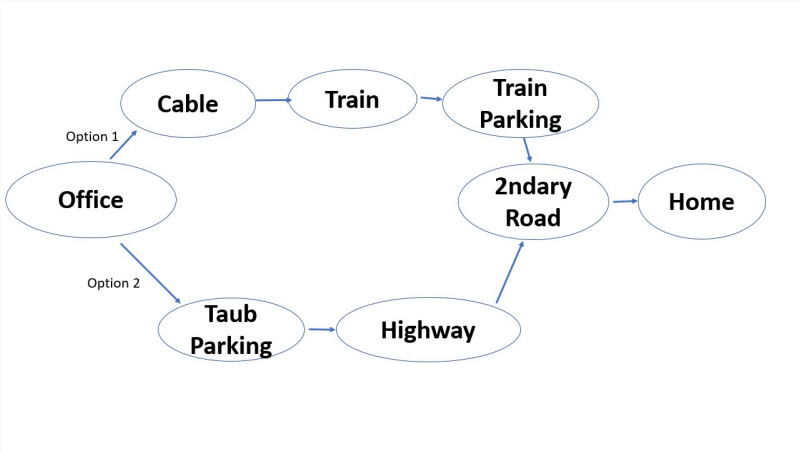


- τ_1 = office, 20, cable, 15, train, 30, train-park, 5, 2ndary, 15, home
- τ_2 = office, 20, cable, 15, train, 60, train-park, 5, 2ndary, 15, home
- τ_3 = office, 5, taub-park, 10, highway, 60, 2ndary, 15, home
- τ_4 = office, 5, taub-park, 10, highway, 60, 2ndary, 35, home
- τ_5 = office, 5, taub-park, 10, highway, 60

Going Home Example

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(G_t - V(S_t))$$
$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(R(S_t) + \gamma V_\pi(S_{t+1}) - V_\pi(S))$$
$$\gamma = 1$$



- τ_1 = office, 20, cable, 15, train, 30, train-park, 5, 2ndary, 15, home
- τ_2 = office, 20, cable, 15, train, 60, train-park, 5, 2ndary, 15, home
- τ_3 = office, 5, taub-park, 10, highway, 60, 2ndary, 15, home
- τ_4 = office, 5, taub-park, 10, highway, 60, 2ndary, 35, home
- τ_5 = office, 5, taub-park, 10, highway, 60

MC vs. TD

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is an **unbiased** estimate of $V_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is **unbiased estimate** of $\pi(S_t)$.
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate $\pi(S_t)$. why?
- TD target has much lower variance than the return - why?

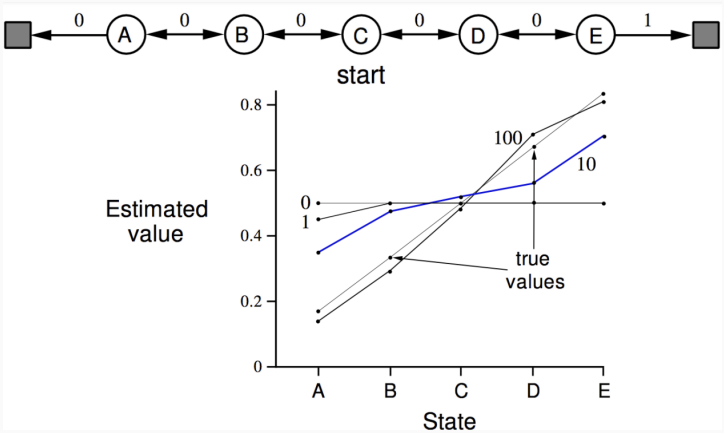
Bias/Variance Trade-Off

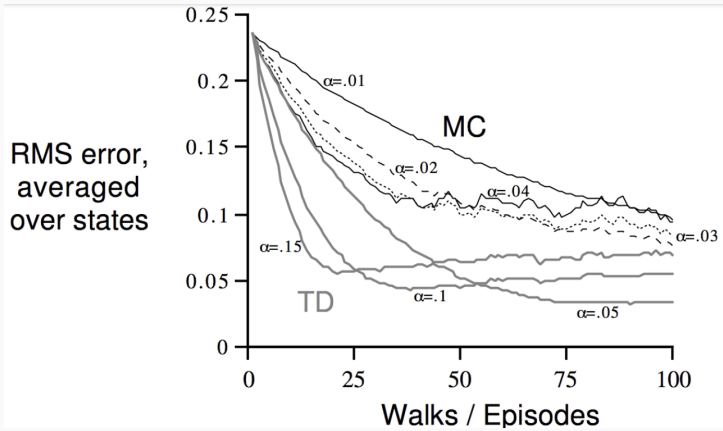
- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is an **unbiased** estimate of $V_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is **unbiased estimate** of $\pi(S_t)$.
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate $\pi(S_t)$. why?
- TD target has much lower variance than the return - why?
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward

MC vs. TD (continued)

- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $\pi(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

Random Walk Example





Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}(G_t^k - \mathcal{V}(s_t^k))^2$$

  - In the AB example, $\mathcal{V}(A) = 0$
- $TD(0)$ converges to solution of max likelihood Markov model
  - Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ that best fits the data

$$\mathcal{P}_{s,s'}^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k} 1(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\mathcal{R}_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k} 1(s_t^k, a_t^k = s, a) r_t^k$$

In the AB example, $\mathcal{V}(A) = 0.7$

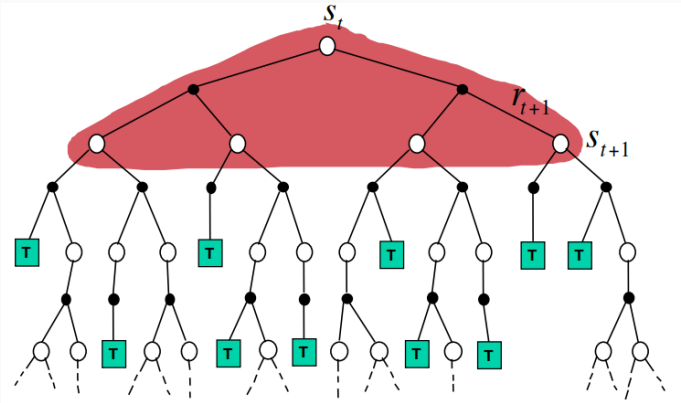
Advantages and Disadvantages of MC vs. TD (continued)

- TD exploits Markov property
  - Usually more efficient in Markov environments
- MC does not exploit Markov property
  - Usually more effective in non-Markov environments

Monte-Carlo vs. Temporal-Difference vs. Dynamic Programming Backup

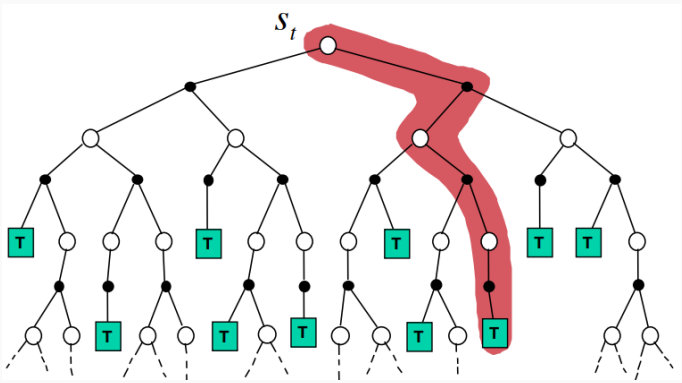
Which is which ?

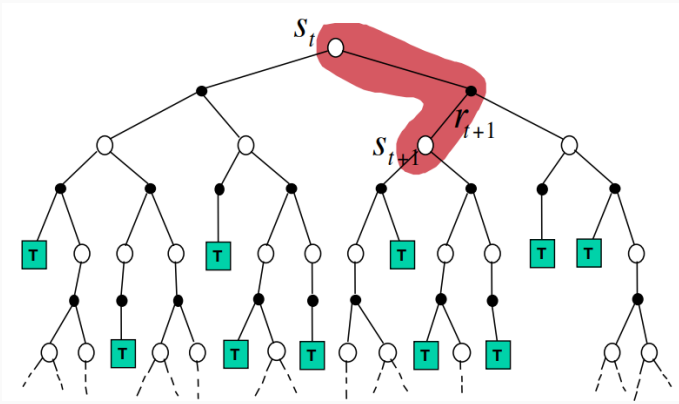
$$\mathcal{V}(s_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma \mathcal{V}(S_{t+1})]$$



$$\mathcal{V}(S_t) \leftarrow \mathcal{V}(S_t) + \alpha(G_t \mathcal{V}(S_t))$$

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





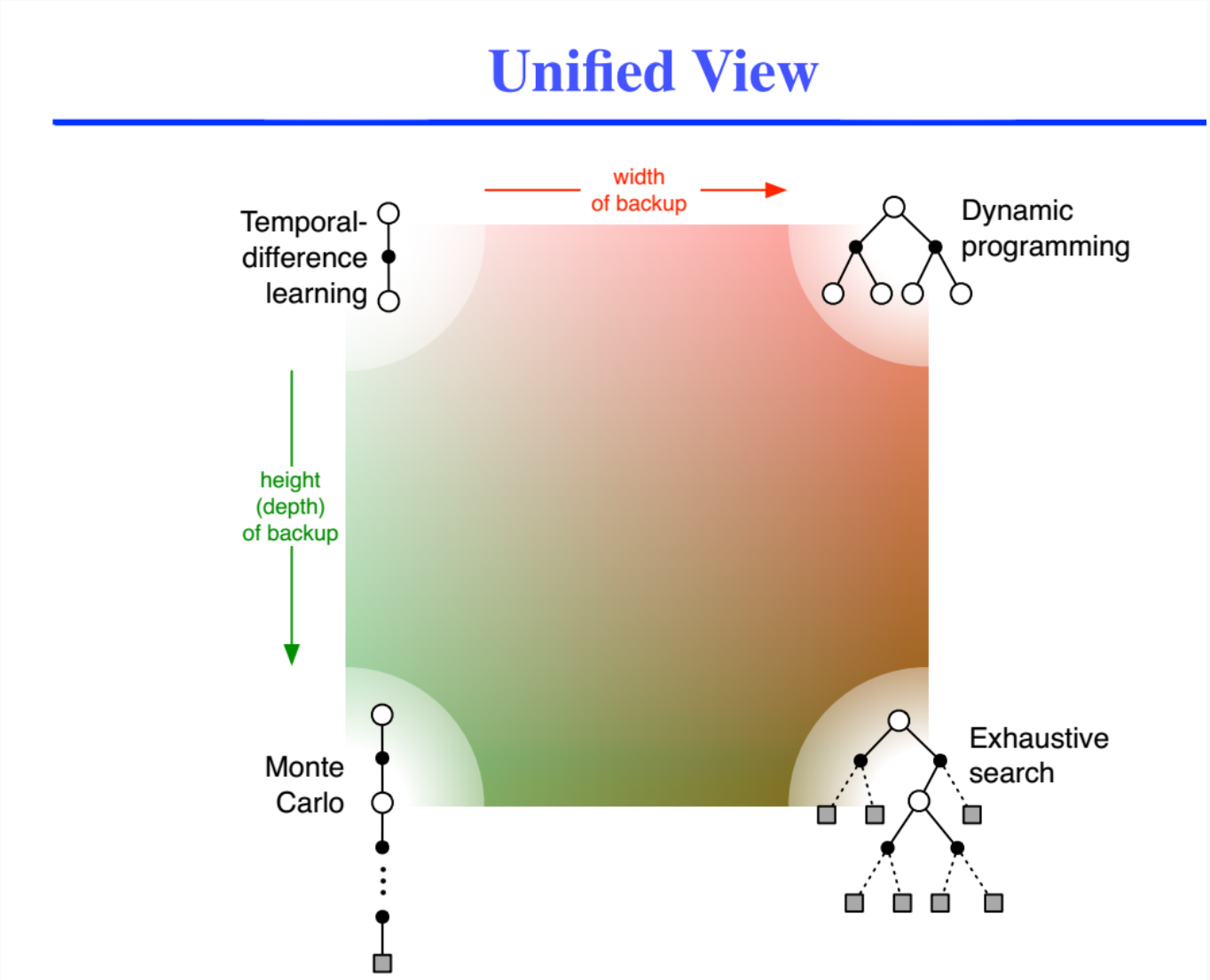
Bootstrapping and Sampling

- **Bootstrapping:** update involves an estimate
  - MC
  - DP
  - TD
- **Sampling:** update samples an expectation
  - MC
  - DP
  - TD

Bootstrapping and Sampling

- **Bootstrapping:** update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- **Sampling:** update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

Unified View of Reinforcement Learning



Extensions and Additional Topics (which we won't cover)

- n-Step Prediction
- Averaging n-Step Returns ($TD(\lambda)$)
- Eligibility Traces and credit assignment:
  - Frequency heuristic: assign credit to most frequent states
  - Recency heuristic: assign credit to most recent states
- Forward vs. backward view

Model-Based RL

Model-Based RL

- We have so far explored two **model-free** approaches:
  - Monte-Carlo methods:

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(G_t - V(S_t))$$

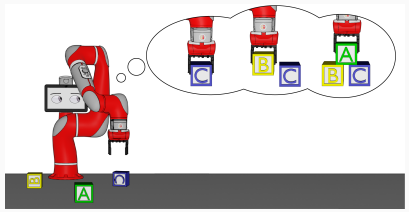
  - Temporal Difference methods:

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(R(S_t) + \gamma V_\pi(S_{t+1}) - V_\pi(S))$$

- In **model-based RL** the agent uses a transition and reward model of the environment to make decisions about how to act.
  - The model may be initially known (e.g., chess) or unknown (e.g., robot manipulator).

Model-Based RL

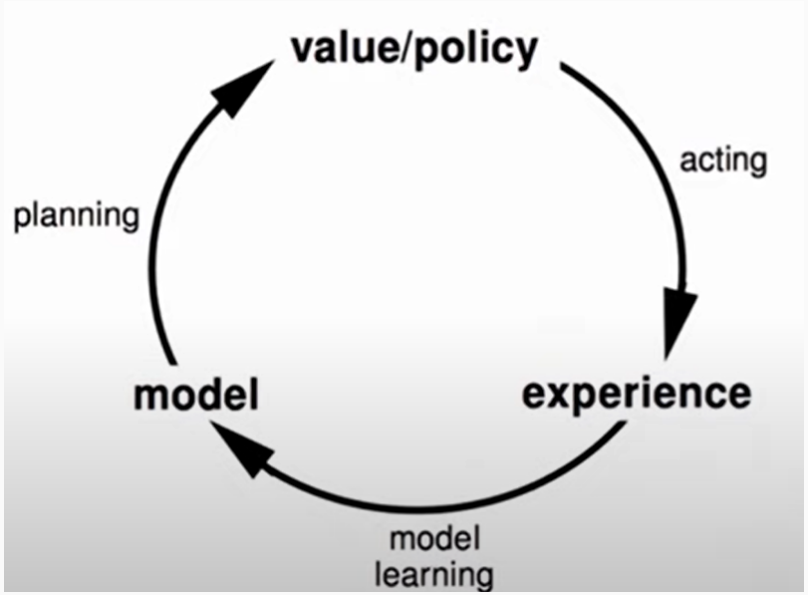
- Reinforcement learning systems can make decisions in one of two ways.
  - In the model-based approach, a system uses a predictive model of the world to ask questions of the form **"what will happen if I do x?"** to choose the best action.
  - In the alternative model-free approach, the modeling step is bypassed altogether in favor of **learning a control policy directly**. **"how much reward will I get if I do x?"**
- Although in practice the line between these two techniques can become blurred, as a coarse guide it is useful for dividing up the space of algorithmic possibilities.



BAIR blog by Michael Janner

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

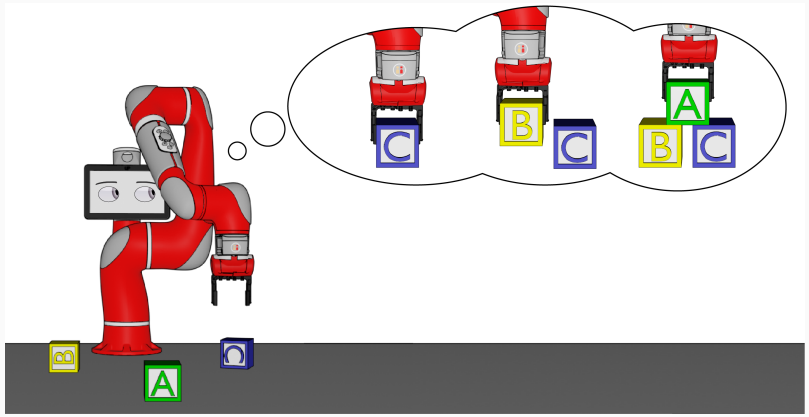
Model-Based RL



By Emma Brunskill

https://www.youtube.com/watch?v=vDF1BYWhqL8

Model-Based RL



Ideas for model-based evaluation ?

Ideas for model-based control ?

Model-based Estimation: Adaptive Dynamic Programming (ADP)

Adaptive Dynamic Programming (ADP)

- Follow the policy for a while
- Estimate transition model based on observations
- Learn reward function
- Use estimated model to compute utility of policy

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s')$$

How can we estimate transition model $\mathcal{P}$ and R?

Model-based Estimation: Adaptive Dynamic Programming (ADP)

Adaptive Dynamic Programming (ADP)

- Follow the policy for a while
- Estimate transition model based on observations
- Learn reward function
- Use estimated model to compute utility of policy

$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s')$$

How can we estimate transition model $\mathcal{P}$ and R?

Compute the fraction of times we see s' after taking a in state s (similarly for the reward function).

(*) Can bound error with Chernoff bound - that bounds the total amount of probability of some random variable Y that is in the "tail", i.e. far from the mean.

Naïve Approach To Model-Based Control

Model-based approach to RL

1. Act randomly for a (long) time
2. Learn transition function and reward function
3. Use value iteration, policy iteration, LAO*,
4. Follow resulting policy thereafter.

Will this work ?

Naïve Approach To Model-Based Control

Model-based approach to RL

1. Act randomly for a (long) time
2. Learn transition function and reward function
3. Use value iteration, policy iteration, LAO*,
4. Follow resulting policy thereafter.

Will this work ?

Yes, if we do step 1 long enough and there are no “dead-ends”.

Any problems?

Naïve Approach To Model-Based Control

Model-based approach to RL

1. Act randomly for a (long) time
2. Learn transition function and reward function
3. Use value iteration, policy iteration, LAO*,
4. Follow resulting policy thereafter.

Will this work ?

Yes, if we do step 1 long enough and there are no “dead-ends”.

Any problems?

We will act randomly for a long time before exploiting what we know

Revision of Naïve Approach

Model-based approach to RL

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

Revision of Naïve Approach

Model-based approach to RL

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work ?

Revision of Naïve Approach

Model-based approach to RL

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work ? No. Can get stuck in local optimum

What can be done ?

Reminder: Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation:** To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration:** Get more information about the world. How do we know if there is not a pot of gold around the corner?
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic **intuition** behind most approaches:

Reminder: Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation:** To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration:** Get more information about the world. How do we know if there is not a pot of gold around the corner?
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic **intuition** behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

ADP-based RL

1. Start with initial model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by an explore/exploit policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work?

ADP-based RL

1. Start with initial model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by an explore/exploit policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy

Any ideas?

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s')$$

  - where V is current optimal value function estimate (based on current model), and $\mathcal{R}$ and $\mathcal{P}$ are current estimates of the model
  - $Q(s,a)$ is the expected value of taking action a in state s and then getting the estimated value $\mathcal{V}(s')$ of the next state s'.
- We want an exploration policy that is GLIE (greedy in the limit of infinite exploration).

Explore/Exploit Policies

- GLIE policy 1:
  - On time step t select random action with probability $p(t)$ (i.e., ϵ) and greedy action with probability $1 - p(t)$
  - $p(t) = \frac{1}{t}$ (will lead to convergence, but is slow.)
  - Greedy action is the one that maximizes the Q value.
- GLIE policy 2: Boltzmann Exploration [1]
  - Select action a with probability $\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$
  - T is the **"temperature"**: Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically: start with large T and decrease with time.

[1] according to wikipedia - a Boltzmann distribution (also called Gibbs distribution) is a probability distribution or probability measure that gives the probability that a system will be in a certain state as a function of that state's energy and the temperature of the system.

Impact of Temperature

$$\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$$

Suppose we have just two actions and that $Q(s, a_1) = 1$, and $Q(s, a_2) = 2$.

1. $T = 10$ gives $\mathcal{P}(a_1|s) = 0.48$, $\mathcal{P}(a_1|s) = 0.52$
   Almost equal probability, and so explore
2. $T = 1$ gives $\mathcal{P}(a_1|s) = 0.27$, $\mathcal{P}(a_1|s) = 0.73$
   Probabilities more skewed, so explore a_1 less
3. $T = .25$ gives $\mathcal{P}(a_1|s) = 0.02$, $\mathcal{P}(a_1|s) = 0.98$
   Almost always exploit a_2

Alternative Approach: Optimistic Exploration

Model-based approach to RL

1. Start with initial model
2. Solve for optimistic policy given current model using optimistic value iteration - that inflates value of actions leading to unexplored regions.
3. Execute greedy action suggested by the optimistic policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

Basically act as if all "unexplored" state-action pairs are maximally rewarding

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) := \mathcal{R}(s) + \gamma \max_a \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s')$$

  - Optimistic variant -

What do we mean by “explored enough”?

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) := \mathcal{R}(s) + \gamma \max_a \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s')$$

  - Optimistic variant - adjusts update to make actions that lead to unexplored regions look promising
- Implement variant of VI that assigns the highest possible value V^{max} to any state-action pair that has not been explored enough
  - Maximum value is when we get maximum reward forever

$$V^{max} = \sum_{t=0}^{\infty} \gamma^t \cdot R^{max} = \frac{R^{max}}{1-\gamma}$$

What do we mean by “explored enough”?

Optimistic Exploration

- What do we mean by "explored enough"?
  - $N(s,a) > N_e$, where $N(s,a)$ is number of times action a has been tried in state s and N_e is a user selected parameter.
  - While the standard update rule is:

$$V^{max} := \sum_{t=0}^{\infty} \gamma^t \cdot R^{max} = \frac{R^{max}}{1-\gamma}$$

  - Optimistic value iteration computes an optimistic value function V^+ using updates:

$$V^+ := \mathcal{R}(s) + \gamma \max_a \begin{cases} V^{max}, & N(s,a) < N_e \\ \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s'), & \text{otherwise} \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around (-> optimistic)
- But after actions are tried enough times, we will perform standard "non-optimistic" value updates

For more details: R-max – A General Polynomial Time Algorithm for Near-Optimal

Model-Based RL

https://www.natolambert.com/writing/debugging-mbrl

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

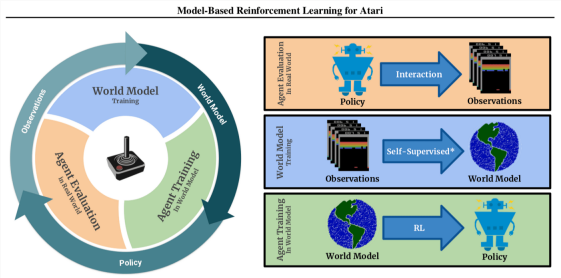
Approaches to Models-Based RL

- Analytic gradient computation:
  - Based on assumptions about the form of the dynamics and cost function (.e.g., Gaussian processes)
  - Can yield locally optimal control (e.g.Linear–quadratic regulator).
  - Even when these assumptions are not valid, can account for small errors introduced by approximated dynamics.
  - Linear (simplified) models, can also be used to provide guiding samples for training more complex nonlinear policies.
- Sampling-based planning:
  - Typically, for nonlinear dynamics models we resort to sampling action sequences (e.g., via random shooting). More sophisticated variants iteratively adjust the sampling distribution.
  - In discrete-action settings, however we can search over tree structures than to iteratively refine a single trajectory of waypoints.
    - **Monte-Carlo Tree Search (MCTS)**- has underpinned recent impressive results in games playing, and iterated width search.
  - In both continuous and discrete domains, can be combined with structured physics-based, object-centric priors.
- Model-based data generation
  - Many machine learning success stories rely on artificially increasing the size of a training set.
  - It is difficult to define a manual data augmentation procedure for policy optimization, but we can view a predictive model analogously as a learned

Model-Based RL: State-of-the-Art

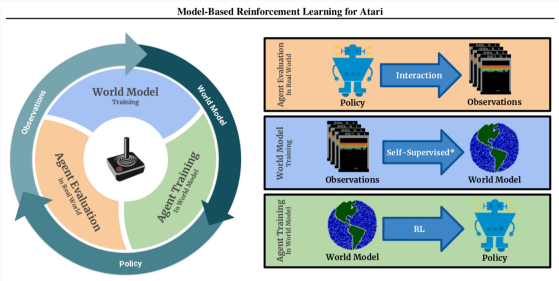
> *"In this paper, we explore how video prediction models can similarly enable agents to solve Atari games with orders of magnitude fewer interactions than model-free methods. We describe Simulated Policy Learning (SimPLe), a complete model-based deep RL algorithm based on video prediction models and present a comparison of several model architectures, including a novel architecture that yields the best results in our setting." (arXiv)*

Paper by Kaiser et al 2020 *https://arxiv.org/abs/1903.00374*
https://medium.com/syncedreview/google-brain-simple-complete-model-based-reinforcement-learning-for-atari



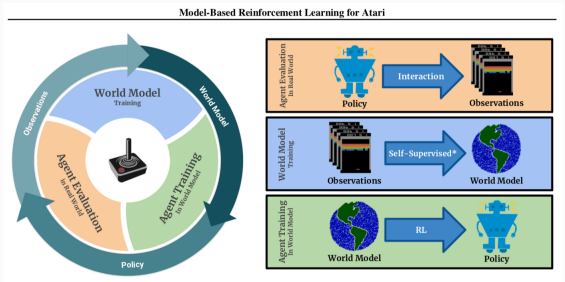
Model-Based RL: Example

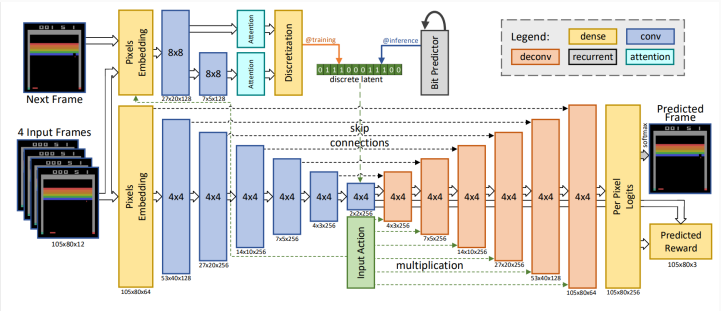
- SimPLe is a complete model-based deep RL algorithm that utilizes video prediction techniques and can train a policy to play a game within the learned model.
- SimPLe outperforms model-free algorithms in terms of learning speed on nearly all of the games, and in the case of a few games, does so by over an order of magnitude.
- The best model-free reinforcement learning algorithms require tens or hundreds of millions of time steps — equivalent to several weeks. SimPLe has obtained competitive results with only 100K interactions between the agent and the environment on Atari games, which corresponds to about two hours of real-time play.



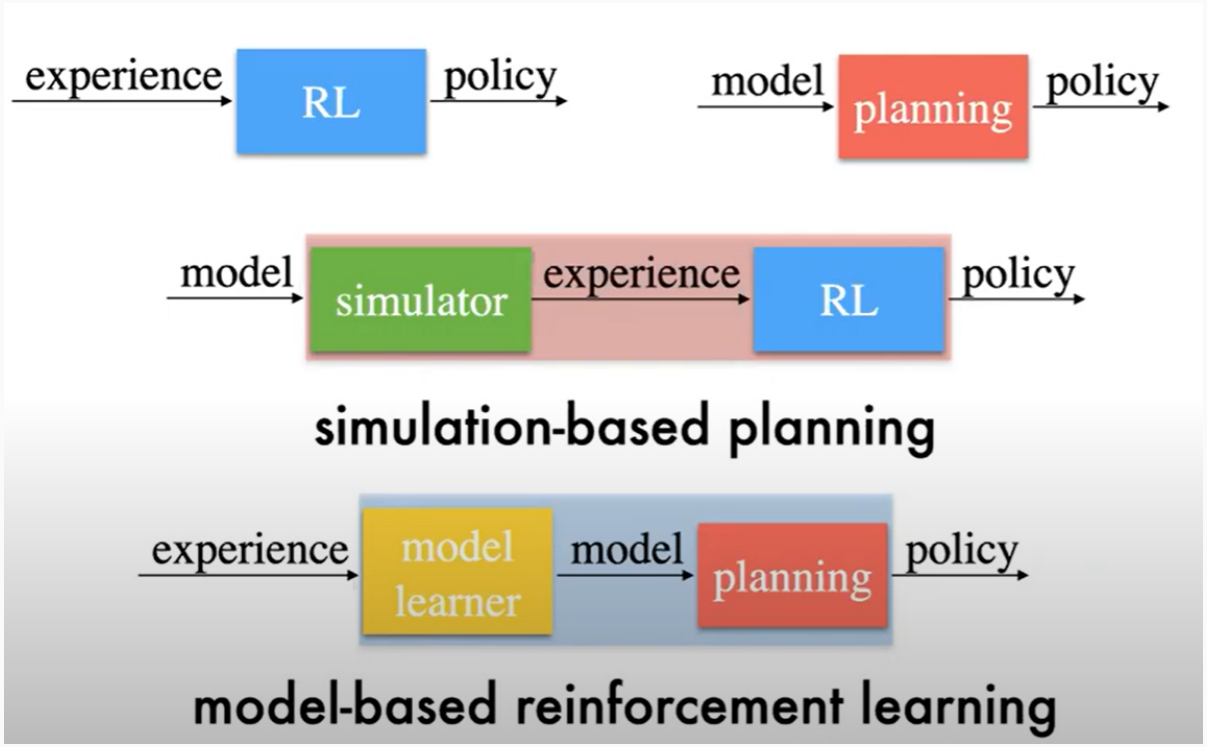
Model-Based RL: Example

- Agent starts interacting with the real environment following the latest policy (initialized to random).
- Collected observations used to train (update) current world model.
- Agent updates the policy by acting inside the world model. The new policy will be evaluated to measure performance and collect more data (back to 1).





Models in RL



Talk by Michael Littman (the funniest AI researcher in the world!)

https://youtu.be/45FKxa3qPHo?t=265

Models-Based RL - Links

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

Benchmarking Model-Based Reinforcement Learning by Wang et al. 2019 *https://arxiv.org/abs/1907.02057* When to Trust Your Model: Model-Based Policy Optimization Janner et al. 2021 *https://arxiv.org/abs/1906.08253*
https://www.natolambert.com/writing/debugging-mbrl

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

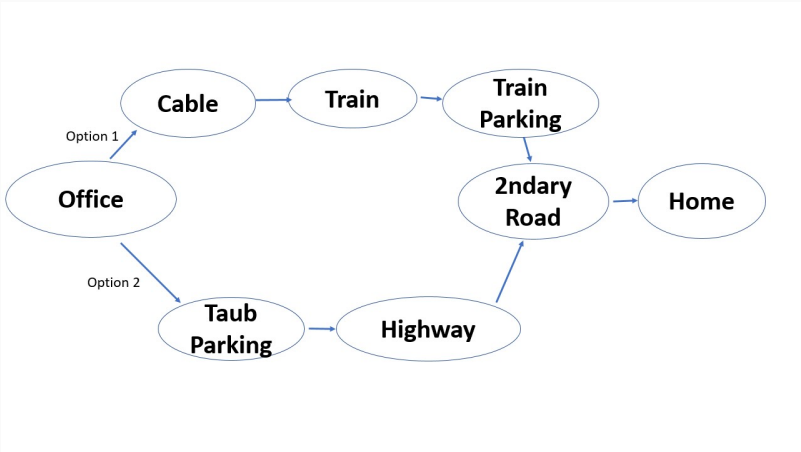
Model-Free vs. Model-Based RL

- Adaptive Dynamic Programming (model based)
- Monte-Carlo Direct Estimation (model free)
- Temporal Difference Learning (model free)

Model-Free vs. Model-Based RL

- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Temporal Difference Learning (model free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints—adjusts state to "agree" with observed successor (not all possible successors)
  - Convergence in between direct estimation and ADP

Model-Based RL for the Going Home Example ?



Policy Search Methods

Policy Search

- **Key Idea:** Keep Twiddling the policy as long as its performance improves, then stop.
- In some ways, policy search is the simplest of all methods

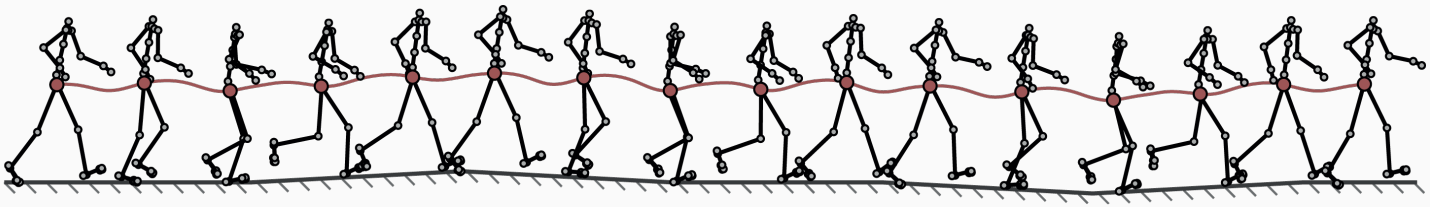
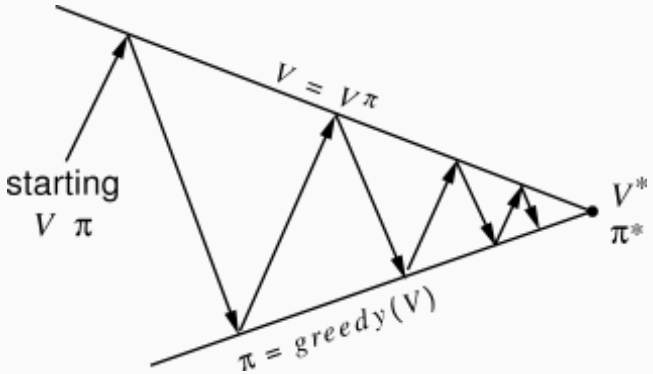


Image from Guided Policy Search by Levine and Koltun, 2013

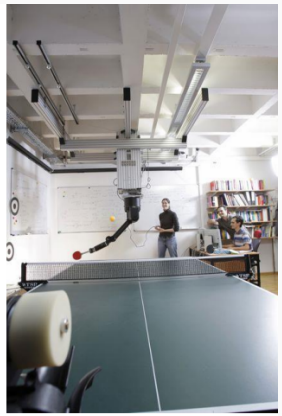
How does this fit within our general structure ?

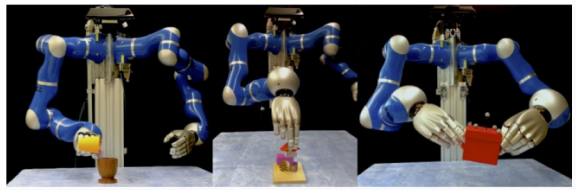


Motivation for Policy-based Reinforcement Learning

Challenges:

- Dimensionality:
  - High-dimensional continuous state and action space
  - Huge variety of tasks
- Real world environments:
  - High-costs of generating data
  - Noisy measurements
- Exploration:
  - Do not damage the robot
  - Need to generate smooth trajectories





From *https://icml.cc/2015/tutorials/PolicySearch.pdf*

Policy Search

Which policy search methods have we already seen ?

Policy Search

Which policy search methods have we already seen ?

- **Policy Iteration:** for known MDPs, we start with a fixed policy and iteratively improve until we reach convergence.
- **Hill Climbing:** maximize (or minimize) a target function $f(\mathbf{x})$. At each iteration, adjust a single element in $\mathbf{x}$ and determine whether the change improves the value of $f(\mathbf{x})$.

Reminder: Value-based Reinforcement Learning:

- Estimate value function: e.g. $Q(s,a) \leftarrow Q(s,a) + \alpha(R + \gamma Q(s',a') - Q(s,a))$
  - Global estimate for all reachable states
  - Hard to scale to high-dimensional space
  - Approximations might compromise policy quality.
- Estimate global policy: e.g. $\pi'(s) = \arg\max_{a \in A} Q(s,a)$
  - Greedy policy update for all states
  - Policy update might get unstable
- Explore the whole state space: e.g. $\pi(a|s) = \frac{\exp(Q(s,a)}{\sum'_a \exp(Q(s,a')}$
  - Uncorrelated exploration in each step
  - (Might damage the robot)

Parameterized Policies

- A policy π is a function that maps states to actions (or state-action pairs to probabilities).
- We approximate the value or action-value function using parameters Θ,

$$\mathcal{V}_\theta(s) \approx \mathcal{V}_\pi(s)$$

$$Q_\theta(s,a) \approx Q_\pi(s,a)$$

- A policy was generated directly from the value function e.g. using ϵ-greedy.
- **Policy search (gradient)** methods directly parametrise the policy $\pi_\theta(s,a) = \mathcal{P}[a|s,\theta]$
- Both model-free and model-based versions.

Policy Search Methods [2]

- Use parametrized policy $a \sim \pi(a|s;\theta)$, θ - parameter vector.
  - Compact parametrizations for high-dimensional spaces
  - Encode prior knowledge
- Locally optimal solutions e.g., $\theta_{new} = \theta_{old} + \alpha \frac{\partial J_\theta}{\partial \theta}$, where J is the loss function.
  - Safe policy updates
  - No global value function estimation
- Correlated local exploration
  e.g.:$\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$
  - Explore in parameter space
  - Generates smooth trajectories

https://icml.cc/2015/tutorials/PolicySearch.pdf

[2] see Deisenroth, Neumann and Peters for A Survey of Policy Search for Robotics, FNT 2013

Model-Free vs. Model-Based Policy Search Methods

- Policy Gradients
  - Likelihood Gradients: REINFORCE [Williams, 1992], PGPE [Rückstiess et al, 2009]
  - Natural Gradients: episodic Natural Actor Critic (eNAC), [Peters & Schaal, 2006]
- Weighted Maximum Likelihood Approaches
  - Success-Matching Principle [Kober & Peters, 2006]
  - Information Theoretic Methods [Daniel, Neumann & Peters, 2012]
  - Extensions: Contextual
- Greedy Updates: PILCO [Deisenroth & Rasmussen, 2011]
- Bounded Updates: Model-Based REPS [Peters at al., 2010], Guided Policy Search by Trajectory Optimization [Levine & Koltun, 2010]

Model-Free and Model-Based Policy Search

$$\text{Samples } \mathcal{D} = \{(s^{[i]}_{1:T}, a^{[i]}_{1:T-1}, r^{[i]}_{1:T})\}$$

Model-Free Policy Search

- Use samples to directly update the policy
- Properties:
  - No model approximations required
  - Applicable in many situations
  - Requires a lot of samples

Model-Based Policy Search

- Use samples to estimate a model
- Properties:
  - Sample efficient
  - Only works if a good model can be learned
  - Optimization of inaccurate models might lead to disaster

Model-Free and Model-Based Policy Search

Model-Free Policy Search Optimization methods:

- Policy Gradients [Williams et al. 992, Peters & Schaal 2006, Rückstiess et al 2008]
- Natural Gradients [[Peters & Schaal 2006, Peters & Schaal 2008, Su, Wiestra & Peters 2009]
- Expectation Maximization [[Kober & Peters 2008, Vlassis & Toussaint 2009]
- Information-Theoretic Policy Search [[Daniel, Neumann & Peters 2012, Daniel, Neumann & Peters, 2013]
- Path Integral Control [[Theoudorou, Buchli & Schaal 2010, Stulp & Sigaud 2012] ‘
- Stochastic Search Methods [[Hansen 2012, Mannor 2004]

Optimization methods for Model-Based Policy Search

- Any model-free method with artificial samples [[Kupscik , Deisenroth, Peters & Neumann, 2013]
- Analytic Policy Gradients [[Deisenroth & Rasmussen 2011]
- Trajectory Optimization [Levine & Koltun 2014]

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy

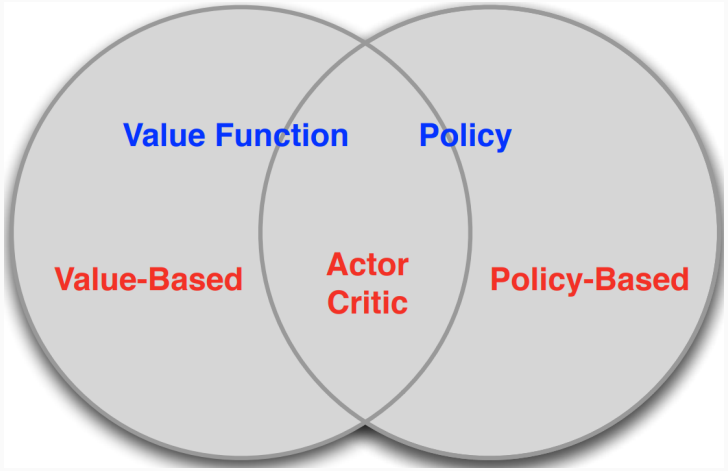


Image by David Silver

Links

- Slides by Jan Peters and Gerhard Neumann *https://icml.cc/2015/tutorials/PolicySearch.pdf*

What Next

What Next ?

Large State Spaces

- When a problem has a large state space we can not longer represent the V or Q functions as explicit tables
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long

What to do??

What Next ?

Large State Spaces

- When a problem has a large state space we can not longer represent the V or Q functions as explicit tables
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long

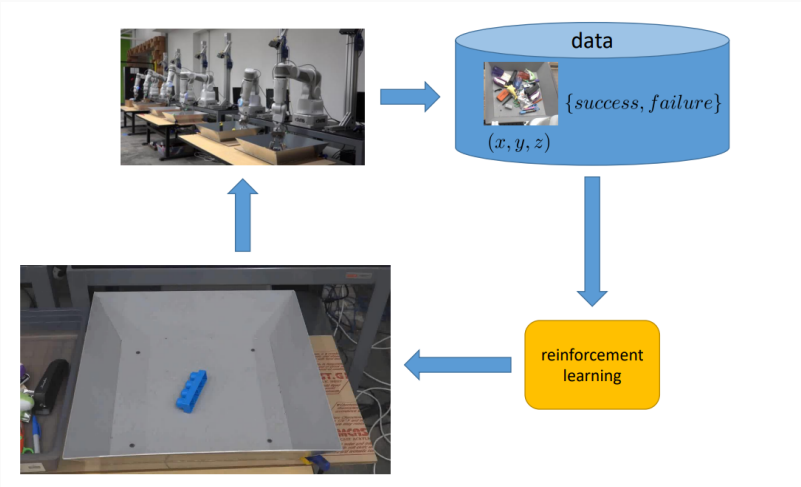
What to do??

- Value function and policy approximators.
- Policy gradient and actor-critic.
- Monte-Carlo Tree Search
- and then...

Supervised Learning

Learning a policy π:

- A sample set: $S = \{(s_i, a_i)_{i=1}^m\}$ or $S = \{(\beta_i, a_i)_{i=1}^m\}$
- Typically, a parameterized policy over a factored state-space representation is used.
- Parameterized policy as a conditional probability $\pi_\theta(a_t|\beta_t)$ or $\pi_\theta(a_t|s_t)$ for fully observed settings

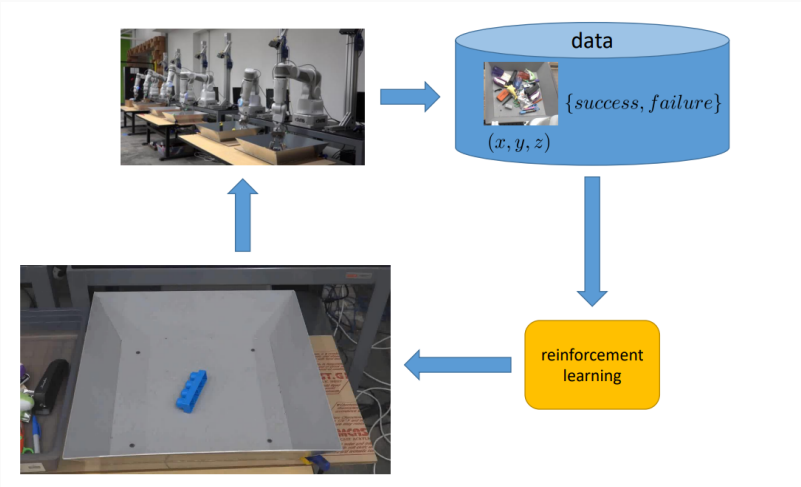


https://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-1.pdf

Supervised Learning

Learning a policy π:

- A sample set: $S = \{(s_i, a_i)_{i=1}^m\}$ or $S = \{(\beta_i, a_i)_{i=1}^m\}$
- Typically, a parameterized policy over a factored state-space representation is used.
- Parameterized policy as a conditional probability $\pi_\theta(a_t|\beta_t)$ or $\pi_\theta(a_t|s_t)$ for fully observed settings



https://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-1.pdf

Limitations?

Monte Carlo Methods

Wiki: Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.

Monte Carlo Methods

- Monte Carlo methods only require experience—sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.
- Requires no prior knowledge of the environment's dynamics to learn efficient behaviors.
- To ensure that well-defined returns are available, we use Monte Carlo methods only for episodic tasks.

Monte Carlo Prediction / Evaluation

- Estimate $\mathcal{V}_\pi(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)
- Reward-to-go of a state s = the sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed reward-to-go of the state as the direct evidence of the actual expected utility of that state
- Averaging the reward-to-go samples will converge to true value at state (under which condition)?

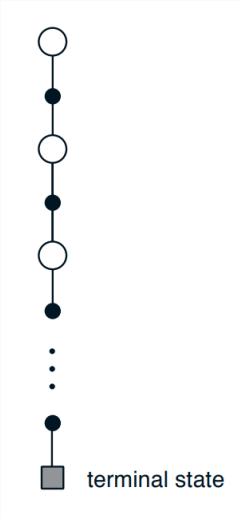


Figure 2: Backup for Monte-Carlo

First-Visit Monte Carlo Prediction

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

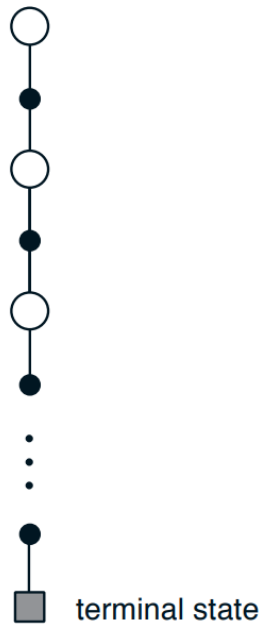
Generate an episode using π

For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$



https://people.cs.umass.edu/~barto/courses/CS383-Fall11/383-Fall11-Lec7.pdf

First-Visit Monte Carlo Prediction

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

$$= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid S_t = s \right]$$

$$= \mathbb{E}_\pi \left[r_{t+1} + \gamma G_{t+1} \mid S_t = s \right]$$

$$= \mathbb{E}_\pi \left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid S_t = s \right]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s' \right] \right]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \forall s \in \mathcal{S}$$

Monte Carlo Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$\pi(s) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list

Repeat forever:
Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
Generate an episode starting from S_0, A_0, following π
For each pair s, a appearing in the episode:
$G \leftarrow$ return following the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
For each s in the episode:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Figure 3: Monte Carlo Exploring Starts (Algorithm

Characteristics ? Model Free or Model Based ? On-Policy or Off-Policy ? Value based or policy based ? Limitations?

Recap and what next

- Spectrum of approaches to RL
- next: deeper dive into RL methods ?