Sequential Decision Making and Reinforcement Learning

(SDMRL)

Model-free RL

Sarah Keren

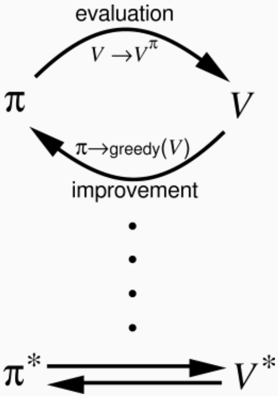
The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Acknowledgments

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

Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Recap



Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Model-Free vs. Model-Based RL

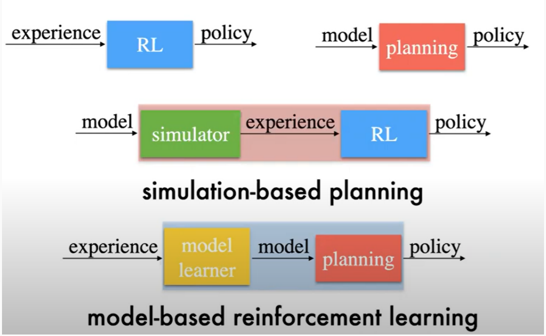


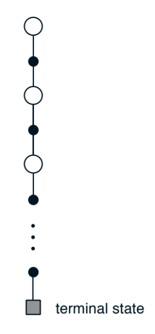
Figure 1: By Michael Littman

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

Model-Free vs. Model-Based RL

Model Free RL: Monte Carlo

- 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



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

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Monte Carlo Simulation-Rollout

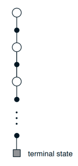
- A Monte Carlo simulation is a statistical technique used to analyze the behavior of complex systems and processes that are influenced by random variables.
- Each rollout involves simulating a sequence of actions, typically by selecting actions according to some policy (e.g., uniformly at random, or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



Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

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).

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

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

  - 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 ?

Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Key Idea

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

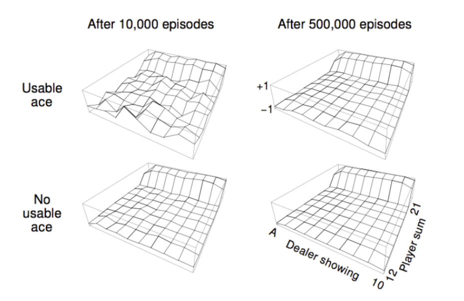
- Reward-to-go of a state s = the sum of the (discounted) rewards from that state until a terminal state is reached
- Two versions to evaluate state s:
  - The first time-step t that state s is visited in an episode
  - Every time-step t that state s is visited in an episode
- Increment visit counter $N(s) \leftarrow N(s) + 1$
- 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)

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

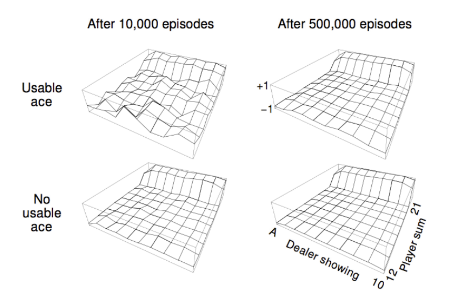
Monte-Carlo Policy Evaluation - Blackjack

- 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

Blackjack Value Function after Monte-Carlo Learning



Blackjack Value Function after Monte-Carlo Learning



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

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

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 ?

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:
        $\pi(s) \leftarrow \arg\max_a Q(s,a)$

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?

- Use a random policy to simulate many (!) trajectories
- Compute the q-value of each state-action pair
- Update π by taking the max action.

Problem with this approach?

Monte-Carlo Policy Control - 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
```

https://github.com/CLAIR-LAB-TECHNION/FSTMA-course/blob/main/tutorials/tut03/Monte_Carlo.ipynb

Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

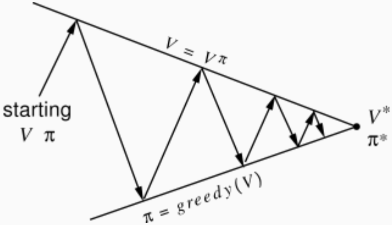
Monte-Carlo Control - Taxi

```python
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

$$\pi'(s) = \arg\max_{a \in A} \sum_{s' \in \mathcal{S}} P(s'|s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right],$$

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

Model-Free Policy Iteration Using Action-Value Function

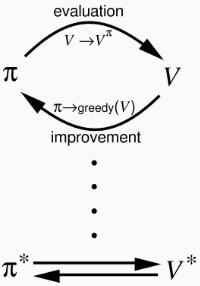
Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \arg\max_{a \in A} \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a,s') + \gamma V^{\pi}(s') \right],$$

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. If for all $s \in \mathcal{S}$ $Q_\pi(s, \pi'(s)) \geq \mathcal{V}_\pi(s)$ then for all $s \in \mathcal{S}$, $\mathcal{V}_{\pi'}(s) \geq \mathcal{V}_\pi(s)$. i.e., π' is an improvement over π.

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

Theorem

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

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

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

ϵ-Greedy Policy Improvement*

Theorem

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

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

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

Theorem

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

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

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

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

Theorem

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

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

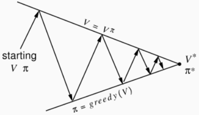
$$= \frac{\epsilon}{m} \sum_{a \in \mathcal{A}} Q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} Q_\pi(s, a)$$

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

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

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

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



- **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,

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})
\end{aligned}$$

Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

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:

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

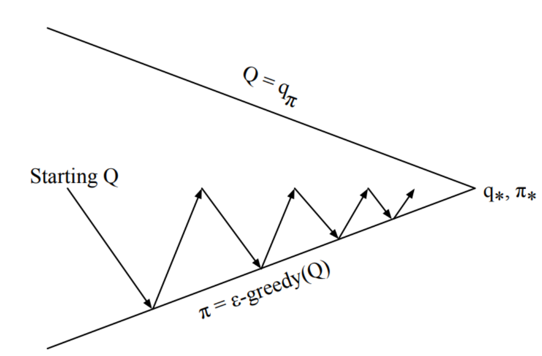
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- 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

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

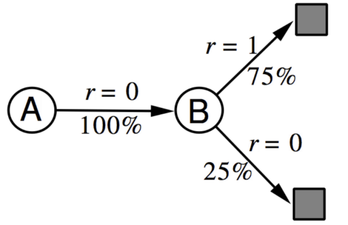
$$V_\pi(s) = R(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s) \cdot V_\pi(s'))$$

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

Temporal-Difference Learning

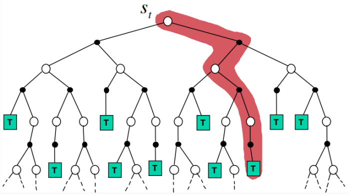
Two states A, B; no discounting; 8 episodes of experience

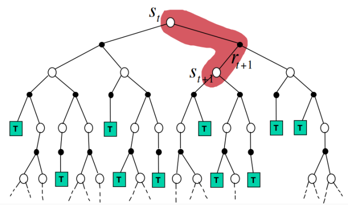
- A, 0, B, 0
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 0



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

From Monte Carlo to Temporal-Difference (TD) Learning





Bellman equation (for a given deterministic policy):

Multiple variations, e.g.,

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

Bellman equation (for a given stochastic policy):

$$V_\pi(s) = \sum_a \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a) \, V_\pi(s') \right]$$

Using Q

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s, a) \sum_{a'} \pi(a' \mid s') Q_\pi(s', a')$$

Can we use these in RL?

Bellman Optimality Equations

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) \max_{a'} Q^*(s',a')$$

$$V^*(s) = \max_a Q^*(s,a)$$

Can we use these in RL?

Reminder: during the learning process we see (possibly partial) trajectories of the form:

$$\tau = s_0, a_1, r_1, s_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$$

Temporal-Difference (TD) Learning: Evaluation

- 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 ?

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

- 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 the next state.
- The update maintains a "mean" of (noisy) utility samples
- How do we guarantee convergence to the true values ?

Reinforcement Learning
(SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Temporal-Difference (TD) Learning: Evaluation

- 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 the next state.
- The update maintains a “mean” of (noisy) utility samples
- How do we guarantee convergence to the true values ?
  - 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).

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

- 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

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

Which policy do we use here?

Temporal-Difference (TD) Learning

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

Temporal-Difference (TD) Learning

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

- TD methods learn directly from episodes of experience

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).

Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

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.

Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

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

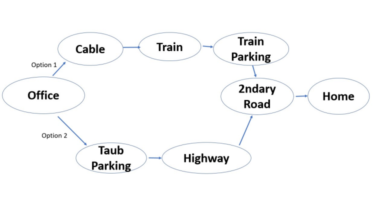
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

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

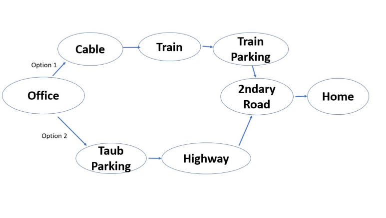
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

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

Going Home Example



Going Home Example

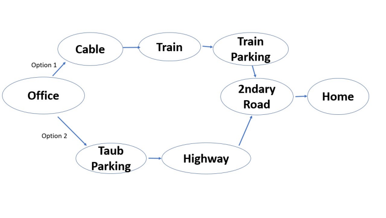


- τ_1 = office, 20, cable, 15, train, 30, train-park, 5, 2ndary, 15, home
- τ_2 = office, 20, cable, 15, train, 60, train-park, 5, 2ndary, 15, home
- τ_3 = office, 5, taub-park, 10, highway, 60, 2ndary, 15, home
- τ_4 = office, 5, taub-park, 10, highway, 60, 2ndary, 35, home
- τ_5 = office, 5, taub-park,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

Going Home Example

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

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



- 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

Going Home Example

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

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

Exit office
Enter Cable

- 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

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

From MC to TD Control

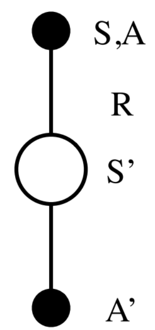
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)

From MC to TD Control

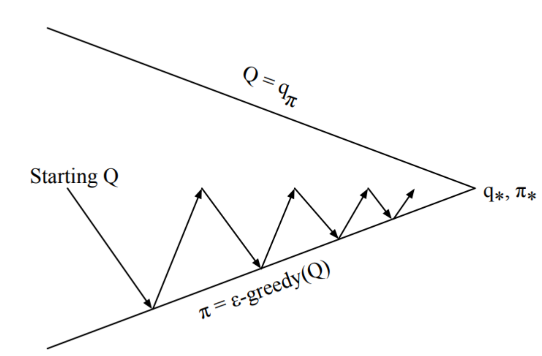
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Works with incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step.

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Updating Action-Value Functions with



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



Every time-step:

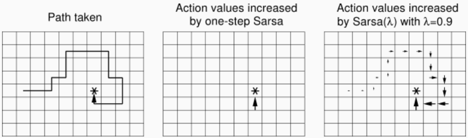
- **Policy evaluation:** Sarsa $Q \approx q_\pi$
- **Policy improvement:** ϵ-greedy improvement

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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  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 $\mu(a|s)$

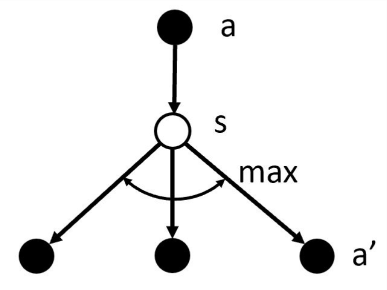
$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

- 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 $Q(s, a)$
- Next action is chosen using the behaviour policy $A_{t+1} \sim \mu(|S_t)$ but we update the target policy by considering the possible actions that could be applied to the next state and choose one with maximal value.

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



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

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

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

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

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

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

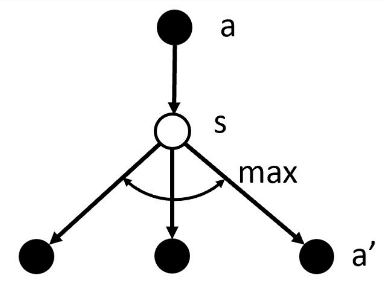
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

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

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

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, $Q(s, a) \to Q^*(s, a)$

SARSA vs. Q-Learning

SARSA:

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

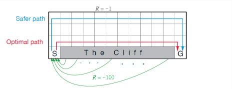
Q-Learning:

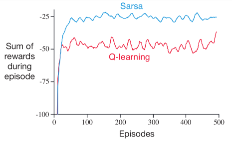
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S', A') - Q(S_t, A_t))$$

SARSA vs. 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
  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 $Q(\textit{terminal-state}, \cdot) = 0$
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





Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Comparing Monte Carlo and TD methods

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?

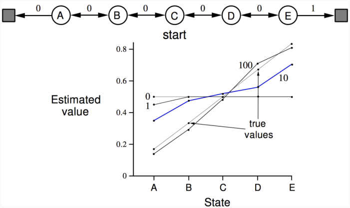
Bias/Variance Trade-Off

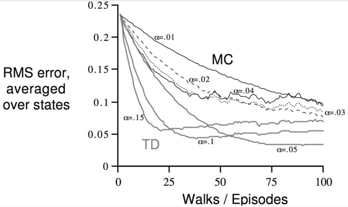
- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is an **unbiased** estimate of $V_\pi(S_t)$
- True TD target $R_{t+1} + \gamma v_\pi(S_{t+1})$ is **unbiased estimate** of $\pi(S_t)$.
- TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate $\pi(S_t)$. why?
- TD target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





MC vs. TD (continued)

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

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

- $TD(0)$ 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

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

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

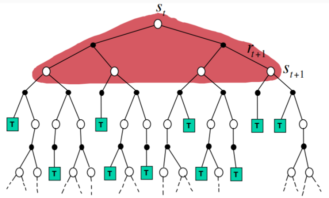
MC vs. TD (continued)

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

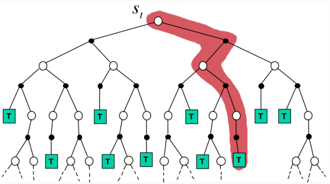
Monte-Carlo vs. Temporal-Difference vs. Dynamic Programming Backup

Which is which ?

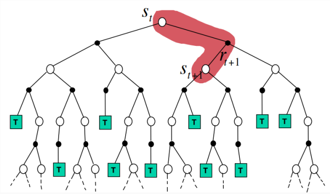
$$\mathcal{V}(s_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma \mathcal{V}(S_{t+1})]$$



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



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

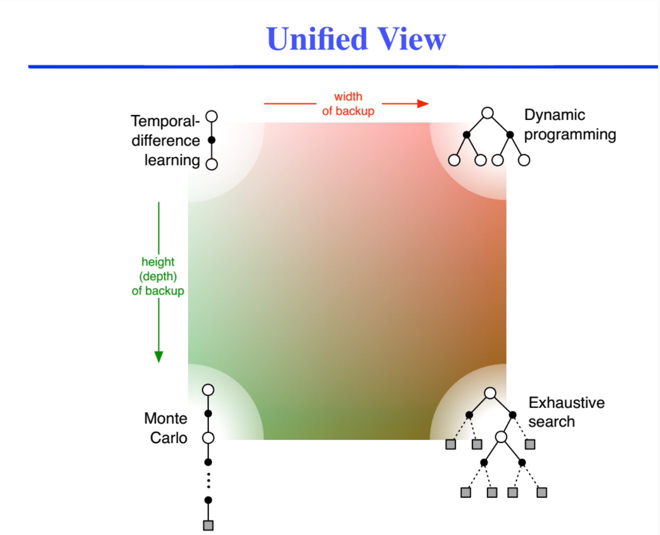


- Bootstrapping:
    - MC
    - DP
    - TD
- Sampling:
    - MC
    - DP
    - TD

Bootstrapping and Sampling

- Bootstrapping:
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling:
  - 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

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

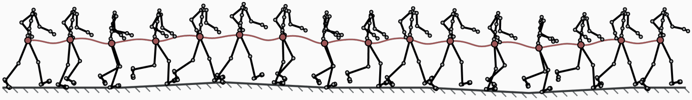
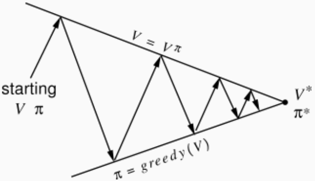


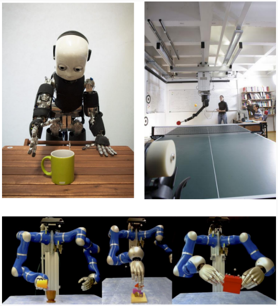
Image from Guided Policy Search by Levine and Koltun, 2013

How does this fit within our general structure ?



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*

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

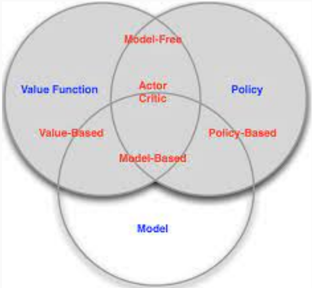
Reinforcement Learning
(SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

Which policy search methods have we already seen ?

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 $f(\mathbf{x})$.

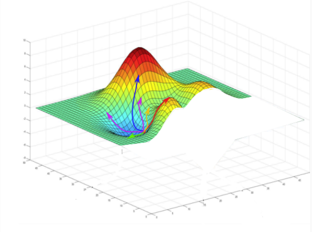
Reminder: RL Approaches



Reminder: Value-based Reinforcement Learning:

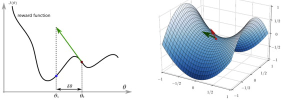
- Estimate value function: e.g.
  $Q(s,a) \leftarrow Q(s,a) + \alpha(R + \gamma Q(s',a') - Q(s,a))$
  - Global estimate for all reachable states
  - Hard to scale to high-dimensional space
  - Approximations might compromise policy quality.
- Estimate global policy: e.g. $\pi'(s) = \arg\max_{a \in A} Q(s,a)$
  - 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 a robot)

Policy Search



Policy Search

A gradient provides a local direction and needs a step size



If the step size is too large, may result in loss

If the direction is slightly wrong, may result in loss

- Policy search methods directly optimize policy parameters without learning a value function
- Key advantage: Can handle continuous action spaces naturally
- Different from value-based methods like Q-learning

Policy Representation

- Policies can be represented as:
  - Neural networks: $\pi_\theta(a|s)$
  - Linear functions: $\pi_\theta(a|s) = \sigma(\theta^T \phi(s))$
  - Gaussian policies: $\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$

- 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.

Reinforcement Learning (SDMRL)
Sarah Keren
Recap
Model Free RL: Monte Carlo
Temporal-Difference Learning
Comparing Monte Carlo and TD methods
Policy Search Methods

- 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

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

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Figure 2: Taken from Sutton and Barto

- Spectrum of approaches to model-free RL
- Next: Dealing with large states space and model-based RL