Alina Vereshchaka

University at Buffalo

avereshc@buffalo.edu

June 6, 2019

*Slides are based on Deep Reinforcement Learning and Control, CMU 10703, Carnegie-Mellon University
Overview

1. Monte Carlo (MC) Methods
2. Temporal Difference
3. SARSA
4. Q-Learning
## Table of Contents

1. Monte Carlo (MC) Methods
2. Temporal Difference
3. SARSA
4. Q-Learning
Monte Carlo

- Monte Carlo methods are learning methods
  - Experience $\rightarrow$ values, policy

- Monte Carlo uses the simplest possible idea: $\text{value} = \text{mean return}$

- Monte Carlo methods can be used in two ways:
  - Model-free: No model necessary and still attains optimality
  - Simulated: Needs only a simulation, not a full model

- Monte Carlo methods learn from complete sample returns
  - Only defined for episodic tasks (this class)
  - All episodes must terminate (no bootstrapping)
Monte-Carlo Policy Evaluation

- **Goal:** learn $v_{\pi}(s)$ from episodes of experience under policy $\pi$
  
  $$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Remember that the **return** is the total discounted reward:
  
  $$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Remember that the **value function** is the expected return:
  
  $$v_{\pi}(s) = \mathbb{E}_{\pi} [G_t \mid S_t = s]$$

- Monte-Carlo policy evaluation uses **empirical mean return** instead of expected return
Monte-Carlo Policy Evaluation

- **Goal**: learn $v_\pi(s)$ from episodes of experience under policy $\pi$
- **Idea**: Average returns observed after visits to $s$:

  ![Monte-Carlo Graph](image)

- **Every-Visit MC**: average returns for every time $s$ is visited in an episode
- **First-visit MC**: average returns only for first time $s$ is visited in an episode
- Both converge asymptotically
First-Visit MC Policy Evaluation

- To evaluate state $s$
  - The **first** time-step $t$ that state $s$ is visited in an episode,
  - Increment 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 $N(s) \rightarrow \infty$
Every-Visit MC Policy Evaluation

- To evaluate state $s$

- *Every* time-step $t$ that state $s$ is visited in an episode,

- **Increment 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 $V(s) \to v_\pi(s)$ as $N(s) \to \infty$
Backup Diagram for Monte Carlo

- Entire rest of episode included
- Only one choice considered at each state (unlike DP)
  - thus, there will be an explore/exploit dilemma
- Does not bootstrap from successor state's values (unlike DP)
- Value is estimated by mean return
- Time required to estimate one state does not depend on the total number of states
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 $G_t$

  $\begin{align*}
  N(S_t) &\leftarrow N(S_t) + 1 \\
  V(S_t) &\leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))
  \end{align*}$

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

  $V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$
Monte-Carlo Control

- **MC policy iteration step**: Policy evaluation using MC methods followed by policy improvement
- **Policy improvement step**: greedify with respect to value (or action-value) function
Monte-Carlo Algorithm

Initialize, for all \( s \in S, \ a \in \mathcal{A}(s) \):

\[
Q(s,a) \leftarrow \text{arbitrary} \\
\pi(s) \leftarrow \text{arbitrary} \\
\text{Returns}(s,a) \leftarrow \text{empty list}
\]

Fixed point is optimal policy \( \pi^* \)

Repeat forever:

Choose \( S_0 \in S \) 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 \( \pi \)
For each pair \( s, a \) appearing in the episode:

\[
G \leftarrow \text{return following the first occurrence of } s, a \\
\text{Append } G \text{ to } \text{Returns}(s,a) \\
Q(s,a) \leftarrow \text{average}(\text{Returns}(s,a))
\]

For each \( s \) in the episode:

\[
\pi(s) \leftarrow \arg \max_a Q(s,a)
\]
Monte Carlo

Advantages

- MC can be used to learn optimal behavior directly from interaction with the environment. It does not require a model of the environment’s dynamics.

- MC can be used with simulation or sample models.

- MC can be used to focus on one region of special interest and be accurately evaluated without having to evaluate the rest of the state set.

Disadvantages

- MC only works for episodic (terminating) environments. It does not work with environment with no terminating states.

- MC must have a complete episodes, it does not have bootstrapping, meaning it does not give an estimates of the other states.

- MC must wait until the end of an episode before return is known. For problems with very
# Table of Contents

1. Monte Carlo (MC) Methods
2. Temporal Difference
3. SARSA
4. Q-Learning
Online decision-making involves a fundamental choice:

- **Exploitation**: Make the best decision given current information (greedy)
- **Exploration**: Gather more information

The greedy algorithm selects action with highest value:

\[ a_t^* = \arg \max_{a} Q_t(s, a) \]
$\epsilon$ – greedy algorithm:

- With probability $\epsilon$ choose a random action $a$
- With probability $1 - \epsilon$ choose “greedy” action $a$ with the highest Q-value.
In $\epsilon$-greedy action selection, for the case of two actions $[a_1, a_2]$ and $\epsilon = 0.5$, what is the probability that the greedy action is selected?
Monte Carlo (MC) and Temporal Difference (TD) Learning

- **Goal:** learn $v_\pi(s)$ from episodes of experience under policy $\pi$

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

- $R_{t+1} + \gamma V(S_{t+1})$ is called the **TD target**
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the **TD error**.
DP vs. MC vs TD Learning

Remember:

\[ v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \]
\[ = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \]
\[ = \mathbb{E}_\pi \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s \right] \]
\[ = \mathbb{E}_\pi \left[ R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]. \]

MC: sample average return approximates expectation

TD: combine both: Sample expected values and use a current estimate \( V(S_{t+1}) \) of the true \( v_\pi(S_{t+1}) \)

DP: the expected values are provided by a model. But we use a current estimate \( V(S_{t+1}) \) of the true \( v_\pi(S_{t+1}) \)
\[ V(S_t) \leftarrow E_\pi \left[ R_{t+1} + \gamma V(S_{t+1}) \right] = \sum_a \pi(a|S_t) \sum_{s', r} p(s', r|S_t, a) [r + \gamma V(s')] \]
Monte Carlo

\[ V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t)) \]
TD(0) Method

\[ V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \]
Bootstrapping: update involves an estimate
- MC does not bootstrap
- DP bootstrap
- TD bootstrap

Sampling: update does not involve an expected value
- MC samples
- DP does not sample
- TD samples
TD(0) Method

- **Policy Evaluation** (the prediction problem):
  - for a given policy $\pi$, compute the state-value function $v_\pi$

- **Remember**: Simple every-visit Monte Carlo method:
  \[ V(S_t) \leftarrow V(S_t) + \alpha \left[ G_t - V(S_t) \right] \]
  \(\text{target}:\) the actual return after time $t$

- The simplest **Temporal-Difference** method TD(0):
  \[ V(S_t) \leftarrow V(S_t) + \alpha \left[ R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \]
  \(\text{target}:\) an estimate of the return
### Example: Driving Home

<table>
<thead>
<tr>
<th>State</th>
<th>Elapsed Time (minutes)</th>
<th>Predicted Time to Go</th>
<th>Predicted Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaving office, friday at 6</td>
<td>0</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>reach car, raining</td>
<td>5</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>exiting highway</td>
<td>20</td>
<td>15</td>
<td>35</td>
</tr>
<tr>
<td>2ndary road, behind truck</td>
<td>30</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>entering home street</td>
<td>40</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>arrive home</td>
<td>43</td>
<td>0</td>
<td>43</td>
</tr>
</tbody>
</table>
Example: Driving Home

Changes recommended by Monte Carlo methods ($\alpha=1$)

Changes recommended by TD methods ($\alpha=1$)
Advantage of TD Learning

- TD methods do not require a model of the environment, only experience.
- You can learn before knowing the final outcome:
  - Less memory
  - Less computation
- You can learn without the final outcome: from incomplete sequences
- Both MC and TD converge (under certain assumptions)
Unified View

- Temporal-difference learning
- Dynamic programming
- Exhaustive search
- Monte Carlo

width of backup

height (depth) of backup
Table of Contents

1. Monte Carlo (MC) Methods
2. Temporal Difference
3. SARSA
4. Q-Learning
- Estimate $q_{\pi}$ for the current policy $\pi$

After every transition from a nonterminal state, $S_t$, do this:

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

If $S_{t+1}$ is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$
SARSA

Turn this into a control method by always updating the policy to be \textbf{greedy} with respect to the current estimate.

**Sarsa (on-policy TD control) for estimating \( Q \approx q_* \)**

<table>
<thead>
<tr>
<th>Algorithm parameters: step size ( \alpha \in (0, 1] ), small ( \varepsilon &gt; 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize ( Q(s, a) ), for all ( s \in S^+, a \in A(s) ), arbitrarily except that ( Q(\text{terminal}, \cdot) = 0 )</td>
</tr>
<tr>
<td>Loop for each episode:</td>
</tr>
<tr>
<td>Initialize ( S )</td>
</tr>
<tr>
<td>Choose ( A ) from ( S ) using policy derived from ( Q ) (e.g., ( \varepsilon )-greedy)</td>
</tr>
<tr>
<td>Loop for each step of episode:</td>
</tr>
<tr>
<td>Take action ( A ), observe ( R, S' )</td>
</tr>
<tr>
<td>Choose ( A' ) from ( S' ) using policy derived from ( Q ) (e.g., ( \varepsilon )-greedy)</td>
</tr>
<tr>
<td>( Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma Q(S', A') - Q(S, A) \right] )</td>
</tr>
<tr>
<td>( S \leftarrow S' ); ( A \leftarrow A' );</td>
</tr>
<tr>
<td>until ( S ) is terminal</td>
</tr>
</tbody>
</table>
Instead of the sample value-of-next-state, use the expectation!

\[
Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right]
\]

\[
\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum \pi(a \mid S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]
\]

Expected Sarsa performs better than Sarsa (but costs more)
SARSA

- SARSA is an **on-policy** algorithm which means that while learning the optimal policy it uses the current estimate of the optimal policy to generate the behaviour.

- SARSA converges to an **optimal policy** as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy ($\epsilon = \frac{1}{t}$).
In Q-learning the learned action-value function, $Q$, directly approximates the optimal action-value function, independent of the policy being followed.

\[
Q(s_t, a_t) \leftarrow (s_t, a_t) + \alpha \left( r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)
\]
Q-Learning Algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi^*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma \max_a Q(S', a) - Q(S, A) \right]$
    $S \leftarrow S'$
  until $S$ is terminal