Temporal Difference (TD)  
Double Q-Learning

Alina Vereshchaka

CSE4/510 Reinforcement Learning  
Spring 2020

avereshc@buffalo.edu

February 24, 2020
Overview

1 Recap

2 Double Q-learning

3 n-step Bootstrapping
<table>
<thead>
<tr>
<th>#</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Recap</td>
</tr>
<tr>
<td>2</td>
<td>Double Q-learning</td>
</tr>
<tr>
<td>3</td>
<td>n-step Bootstrapping</td>
</tr>
</tbody>
</table>
Recap: 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.**
\[ V(S_t) \leftarrow E_{\pi} \left[ R_{t+1} + \gamma V(S_{t+1}) \right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')] \]
Monte Carlo

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

\[ V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \]
Update Functions

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

2. $V(s_t) \leftarrow V(s_t) + \alpha \left[ r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]

3. $V(s_t) \leftarrow V(s_t) + \alpha \left[ G_t - V(s_t) \right]$

4. $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$
Update functions

Monte-Carlo:

\[ V(s_t) \leftarrow V(s_t) + \alpha [G_t - V(s_t)] \]

TD(0):

\[ V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \]

SARSA:

\[ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \]

Q-learning:

\[ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \]
TD(0) Algorithm

\[ \text{Tabular TD(0) for estimating } v_\pi \]

Input: the policy \( \pi \) to be evaluated
Algorithm parameter: step size \( \alpha \in (0, 1] \)
Initialize \( V(s) \), for all \( s \in S^+ \), arbitrarily except that \( V(\text{terminal}) = 0 \)

Loop for each episode:
- Initialize \( S \)
  - Loop for each step of episode:
    - \( A \leftarrow \text{action given by } \pi \text{ for } S \)
    - Take action \( A \), observe \( R, S' \)
    - \( V(S) \leftarrow V(S) + \alpha \left[ R + \gamma V(S') - V(S) \right] \)
    - \( S \leftarrow S' \)
  until \( S \) is terminal
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_\ast**

- 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 $S$
  - Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  - Loop for each step of episode:
    - Take action $A$, observe $R, S'$
    - Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-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
\[ Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \right] \]

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

- Instead of the sample value-of-next-state, use the expectation!
- Expected Sarsa performs better than Sarsa (but costs more)
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 $Q(terminal, \cdot) = 0$

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 $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
    until $S$ is terminal
Table of Contents

1 Recap

2 Double Q-learning

3 n-step Bootstrapping
One (Estimator) Isn’t Good Enough?
Double Q-learning

One (Estimator) Isn’t Good Enough?

https://pbs.twimg.com/media/C5ymV2iVMAYtAev.jpg
Double Q-learning

Two estimators:

- **Estimator** $Q_1$: Obtain best actions
- **Estimator** $Q_2$: Evaluate $Q$ for the above action
Double Q-learning

Two estimators:

- **Estimator $Q_1$**: Obtain best actions
- **Estimator $Q_2$**: Evaluate $Q$ for the above action

What is the main motivation?
Double Q-learning

Two estimators:

- **Estimator** $Q_1$: Obtain best actions
- **Estimator** $Q_2$: Evaluate $Q$ for the above action

Chances of both estimators overestimating at same action is lesser
Double Q-learning

Two estimators:

- **Estimator $Q_1$:** Obtain best actions
- **Estimator $Q_2$:** Evaluate $Q$ for the above action

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha (\text{Target} - Q_1(s, a))$$

**Q Target:** \(r(s, a) + \gamma \max_{a'} Q_1(s', a')\)
Double Q-learning

Two estimators:

- **Estimator** $Q_1$: Obtain best actions
- **Estimator** $Q_2$: Evaluate $Q$ for the above action

\[
Q_1(s, a) \leftarrow Q_1(s, a) + \alpha (\text{Target} - Q_1(s, a))
\]

**Q Target:** \[ r(s, a) + \gamma \max_{a'} Q_1(s', a') \]

**Double Q Target:** \[ r(s, a) + \gamma Q_2(s', \arg \max_{a'} (Q_1(s', a')))) \]
Double Q-learning

\[ r(s_t = B, \cdot) \text{ after 2 trials} \]

<table>
<thead>
<tr>
<th>Action 1</th>
<th>Action 2</th>
<th>Action 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.9</td>
<td>-0.5</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

\[ N(-0.1, 1) \]
Double Q-learning

% left actions from A

Q-learning

Double Q-learning

Episodes
Algorithm 1 Double Q-learning

1: Initialize $Q^A, Q^B, s$

2: repeat

3: Choose $a$, based on $Q^A(s, \cdot)$ and $Q^B(s, \cdot)$, observe $r, s'$

4: Choose (e.g. random) either UPDATE(A) or UPDATE(B)

5: if UPDATE(A) then

6: Define $a^* = \text{arg max}_a Q^A(s', a)$

7: $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a) \left( r + \gamma Q^B(s', a^*) - Q^A(s, a) \right)$

8: else if UPDATE(B) then

9: Define $b^* = \text{arg max}_a Q^B(s', a)$

10: $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a) \left( r + \gamma Q^A(s', b^*) - Q^B(s, a) \right)$

11: end if

12: $s \leftarrow s'$

13: until end
Double Q-learning

- Two approximate value functions are treated completely symmetrically.
- The policy can use both action-value estimates (e.g. $\epsilon$-greedy policy for Double Q-learning could be based on the average of the two action-value estimates).
- Double Q-learning might underestimate the action values at times, but avoids the flaw of the overestimation bias that Q-learning does.
- In most type of problems Double Q-learning reaches good performance faster comparing to Q-learning
- We will be using Double Q-learning in state-of-art algorithm later in the course
1 Recap

2 Double Q-learning

3 n-step Bootstrapping
Neither MC methods nor one-step TD methods are always the best

n-step TD methods generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task
TD Bootstrapping
TD Bootstrapping

1-step TD and TD(0)
TD Bootstrapping

1-step TD and TD(0)  2-step TD

Diagram showing the flow of TD Bootstrapping.
TD Bootstrapping

1-step TD
and TD(0)  2-step TD  3-step TD
TD Bootstrapping

1-step TD and TD(0)  2-step TD  3-step TD  n-step TD

Diagram showing the progression from 1-step TD to n-step TD.
**TD Bootstrapping**

### n-step TD for estimating $V \approx v_\pi$

Input: a policy $\pi$
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer $n$
Initialize $V(s)$ arbitrarily, for all $s \in S$
All store and access operations (for $S_t$ and $R_t$) can take their index mod $n + 1$

Loop for each episode:
  - Initialize and store $S_0 \neq$ terminal
  - $T \leftarrow \infty$
  - Loop for $t = 0, 1, 2, \ldots$:
    - If $t < T$, then:
      - Take an action according to $\pi(\cdot|S_t)$
      - Observe and store the next reward as $R_{t+1}$ and the next state as $S_{t+1}$
      - If $S_{t+1}$ is terminal, then $T \leftarrow t + 1$
    - $\tau \leftarrow t - n + 1$ (\( \tau \) is the time whose state's estimate is being updated)
    - If $\tau \geq 0$:
      - $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1}R_i$
      - If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$
      - $V(S_{\tau}) \leftarrow V(S_{\tau}) + \alpha [G - V(S_{\tau})]$  
  - Until $\tau = T - 1$

**Compute n-step return** $(G_{\tau;\tau+n})$

**Update V**