EE488B Special Topics in EE <Deep Learning and AlphaGo>

Sae-Young Chung

Lecture 11

November 6, 2017

Contents

- Dynamic programming
  - Policy iteration
  - Value iteration
- Monte-Carlo methods
- Temporal-difference learning
  - Sarsa
  - Q-learning

Policy Evaluation

- Let's assume the model of the environment's dynamics, i.e., $p(s', r|s, a)$, is known.
  - It can be known for simple toy examples such as the maze game mentioned in the last lecture.
  - But, impossible to know when playing Go against a human opponent
- Policy evaluation: computing state-value function v_π given policy π
  - Recap: Bellman equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

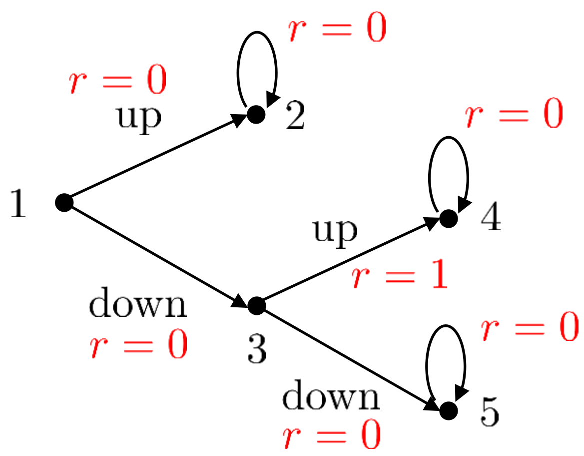
  - Iterative policy evaluation

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

    where $v_0(s)$'s are initialized arbitrarily. v_k can be shown to converge to v_π as $k \to \infty$ for finite MDP provided v_π exists.

Policy Evaluation Example

- Example)
  - $\mathcal{S} = \{1, 2, 3, 4, 5\}$
  - $\mathcal{A}(s) = \{\text{up}, \text{down}\}$ if $s = 1, 3$
  - State 1 is the starting state
  - States 2,4,5 are terminal (absorbing) states
  - Reward is 1 if $s = 3$ and $s' = 4$, otherwise reward is 0



$$p(s', r|s, a) = \begin{cases} 1 & \text{if } s' = 2,\ r = 0,\ s = 1,\ a = \text{up} \\ 1 & \text{if } s' = 3,\ r = 0,\ s = 1,\ a = \text{down} \\ 1 & \text{if } s' = 4,\ r = 1,\ s = 3,\ a = \text{up} \\ 1 & \text{if } s' = 5,\ r = 0,\ s = 3,\ a = \text{down} \\ 1 & \text{if } r = 0 \text{ and } s' = s = 2, 4, \text{ or } 5 \\ 0 & \text{otherwise} \end{cases}$$

Policy Evaluation Example

- Assume

$$\pi(a|s) = \begin{cases} 1 & \text{if } a = \text{down},\ s = 1 \\ 1 & \text{if } a = \text{up},\ s = 3 \\ 0 & \text{otherwise} \end{cases}$$

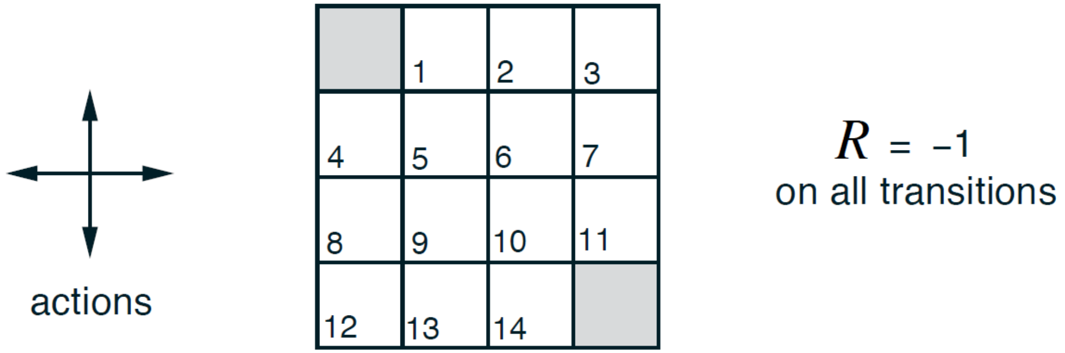
- Then, $v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$ becomes

$$\begin{pmatrix} v_\pi(1) \\ v_\pi(2) \\ v_\pi(3) \\ v_\pi(4) \\ v_\pi(5) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & \gamma & 0 & 0 \\ 0 & \gamma & 0 & 0 & 0 \\ 0 & 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 0 & \gamma \end{pmatrix} \begin{pmatrix} v_\pi(1) \\ v_\pi(2) \\ v_\pi(3) \\ v_\pi(4) \\ v_\pi(5) \end{pmatrix}$$

- Its solution is given by

$$\begin{pmatrix} v_\pi(1) \\ v_\pi(2) \\ v_\pi(3) \\ v_\pi(4) \\ v_\pi(5) \end{pmatrix} = \begin{pmatrix} \gamma \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Gridworld Example



Example 4.1 from Reinforcement Learning by Sutton & Barto

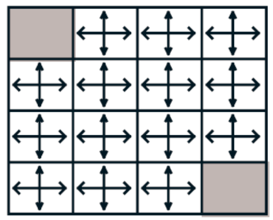
Gridworld Example

v_k for the Random Policy

Greedy Policy w.r.t. v_k

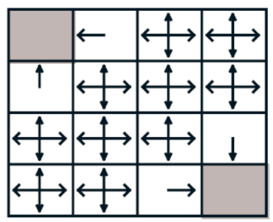
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



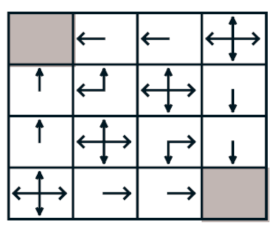
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



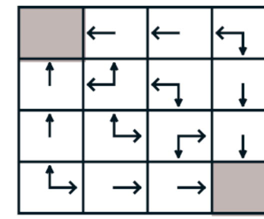
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



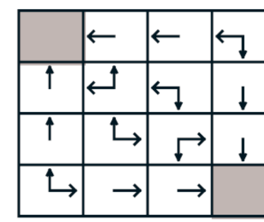
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

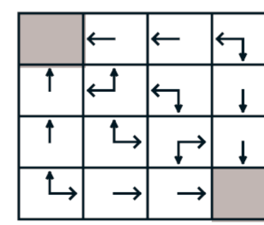


Fig. 4.1 from Reinforcement Learning by Sutton & Barto

Policy Improvement

- Policy improvement: contructing a new greedy policy π' based on old policy π as follows

$$\pi'(s) = \operatorname*{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

- Since the old policy satisfies the following Bellman equality, we can show the new policy is at least as good as the old one, i.e., $v_{\pi'}(s) \geq v_\pi(s)$, $\forall s \in \mathcal{S}$:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

Policy Iteration

- Policy iteration: combination of policy evaluation and policy improvement to find an optimal policy

  1. Initialize π, e.g., random policy
  2. Policy evaluation: calculate $v_\pi(s)$ based on π
  3. Policy improvement: obtain π' from v_π
  4. Replace π by π' and repeat the above two steps

- If $\pi' = \pi$, $v_{\pi'}$ satisfies the Bellman optimality equation.
- π converges to an optimal policy for finite MDP.

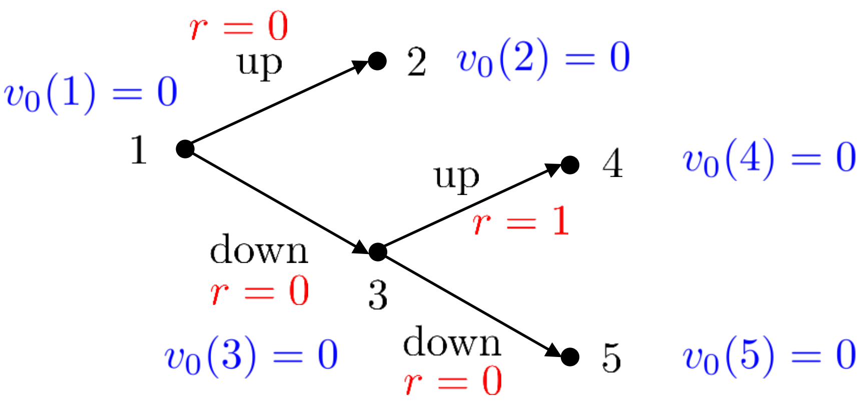
Value Iteration

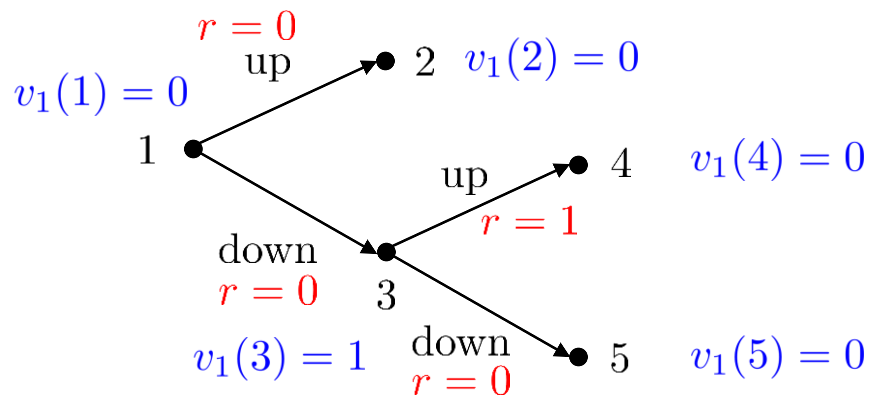
- One problem with policy iteration: policy evaluation step requires many sub-steps if it uses iterations
- Value iteration: a simpler iterative algorithm that directly updates the state-value function

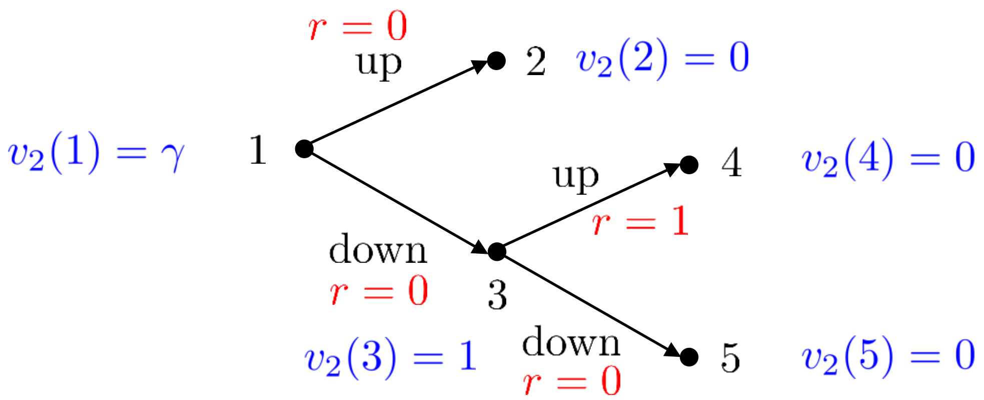
$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_k(s')] \end{aligned}$$

  – It can be considered as combining policy improvement and truncated policy evaluation (i.e., only one iteration).
  – It can also be considered as solving the Bellman optimality equation iteratively.
  – For arbitrary v_0, v_k can be shown to converge to v_* for finite MDP provided v_* exists.
- Policy iteration and value iteration are examples of dynamic programming.

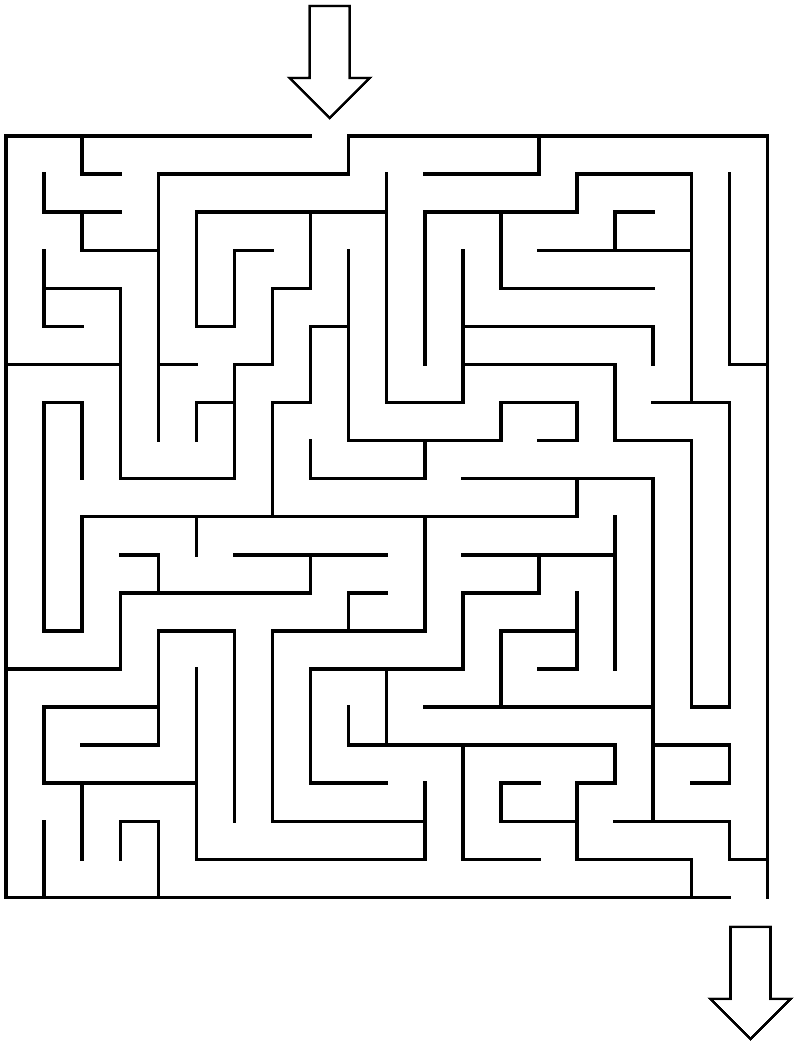
Value Iteration Example







Recap – Maze Game



Random Walk

- Random walk example: $S_n = \sum_{i=1}^n X_i$, $X_i \sim$ i.i.d. ± 1 w.p. $\frac{1}{2}$ each

$$\mathbb{E}[S_n] = \sum_{i=1}^n \mathbb{E}[X_i] = 0$$

$$\mathbb{E}[S_n^2] = \mathbb{E}\left[\sum_{i=1}^n X_i^2\right] = \sum_{i=1}^n \sum_{j=1}^n \mathbb{E}[X_i X_j]$$

$$= \sum_{i=1}^n \mathbb{E}[X_i^2] = n$$

$$\lim_{n\to\infty} \frac{\mathbb{E}[|S_n|]}{\sqrt{n}} = \sqrt{\frac{2}{\pi}}$$

Monte-Carlo (MC) Methods

- From now on, let's assume $p(s', r|s, a)$ is not available.
- Can we still estimate the value function and find near-optimal policy?
- Yes, but not based on $p(s', r|s, a)$ but based on experience (actual or simulated)
- Experience: sample sequences of states, actions, and rewards
- Monte-Carlo method: method based on taking averages of such random samples
- Higher accuracy if more samples are available for averaging
- For simplicity, let's assume episodic tasks

Monte-Carlo Policy Evaluation

Algorithm 1 First-visit MC prediction for estimating $V \sim v_\pi$

```
function FirstVisitMCPrediction(π (policy to be evaluated), NumEpisodes)
    initialize G(s) ← ∅, ∀s ∈ S
    for i = 1, 2, ..., NumEpisodes do
        Generate an episode using π
        for each state s appearing in the episode do
            G ← return following the first occurrence of s
            Append G to G(s)
    V(s) ← average of G(s), ∀s ∈ S
```

- Guaranteed to converge to v_π as $N \to \infty$ since sample returns are i.i.d.
- Every-visit MC policy evaluation: averages returns following all occurrences of s
- MC policy evaluation can be combined with policy improvement to give a MC version of policy iteration.

Temporal-difference Learning

- Temporal-difference learning
  - Iterative learning method combining ideas of MC methods and dynamic programming
  - Learns directly from experience without a model of the environment's dynamics
- Instead of averaging returns following every occurrence of s (as in every-visit MC policy evaluation), let's try exponentially weighted averaging using a one-tap IIR low pass filter, i.e.,

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t$$

- This can be written as the following temporal-difference form (α[new - old])

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

- To obtain G_t, we need wait until the end of the episode, which is costly. We can try to use $R_{t+1} + \gamma V(S_{t+1})$ instead of G_t, i.e.,

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

Temporal-difference Learning

- This is the simplest TD method known as TD(0).
- It is also called a bootstrapping method since it relies on existing estimate $V(S_{t+1})$. For the same reason, DP is also called a bootstrapping method.
- $\delta_t := R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error.
- Since it does not require G_t, it can operate in an on-line fashion, i.e., we don't need to wait until the end of an episode.
- TD(0) converges to v_π in the mean if α is small enough and with probability 1 if α is decreased to zero appropriately.
- But, convergence can be slower compared to MC if the reward is delayed a lot, e.g., as in maze game. Because G_t can contain information on the delayed reward while δ_t cannot contain information on the delay reward not yet captured in $V(S_{t+1})$.

Sarsa: On-policy TD Control

Algorithm 2 SARSA (State-Action-Reward-State-Action) algorithm

function SARSA
  Set $Q(s,a)$, $\forall s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily
  $Q(\text{terminal state}, \cdot) \leftarrow 0$
  for each episode **do**
    $S \leftarrow$ start state
    Choose A at S using policy based on Q (e.g., ϵ-greedy)
    for each step of the episode **do**
      Take action A, get R and S'
      Choose A' at S' using policy based on Q (e.g., ϵ-greedy)
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
      $S \leftarrow S'$, $A \leftarrow A'$

- Based on TD(0) for estimating action-value function as follows:

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

- On-policy method evaluates or improves the policy that is being used to make decisions.

Q-learning: Off-policy TD Control

Algorithm 3 Q-learning algorithm

function QLEARNING
    Set $Q(s,a)$, $\forall s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, arbitrarily
    $Q(\text{terminal state}, \cdot) \leftarrow 0$
    for each episode **do**
        $S \leftarrow$ start state
        for each step of the episode **do**
            Choose A at S using policy based on Q (e.g., ϵ-greedy)
            Take action A, get R and S'
            $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
            $S \leftarrow S'$

- Q-learning uses

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

- Can show Q converges to q_* with probability 1 under some conditions.
- Off-policy method evaluates or improves a policy different from that used to generate data.

Cliff Walk

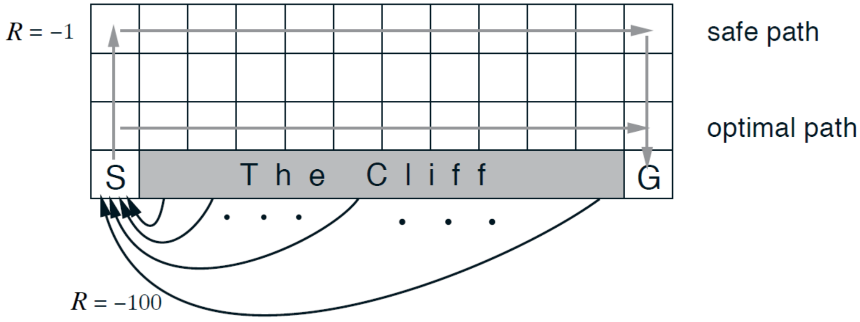


Fig. 6.5 from https://webdocs.cs.ualberta.ca/~sutton/book/the-book-2nd.html

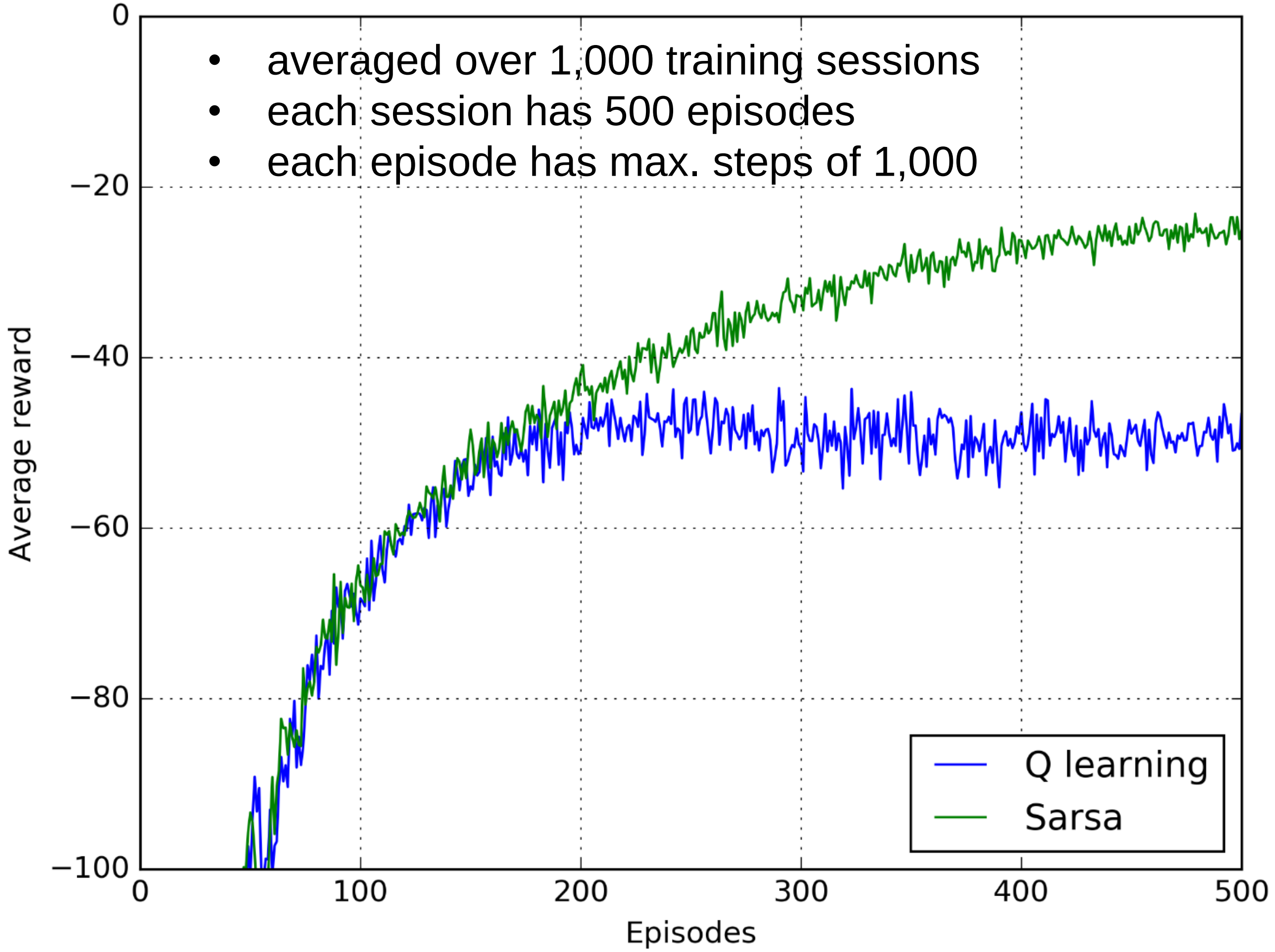
cliff_walk.py (download from KLMS)

Simulations

$\alpha = 0.1$

$\gamma = 0.99$

$\epsilon = 0.1$

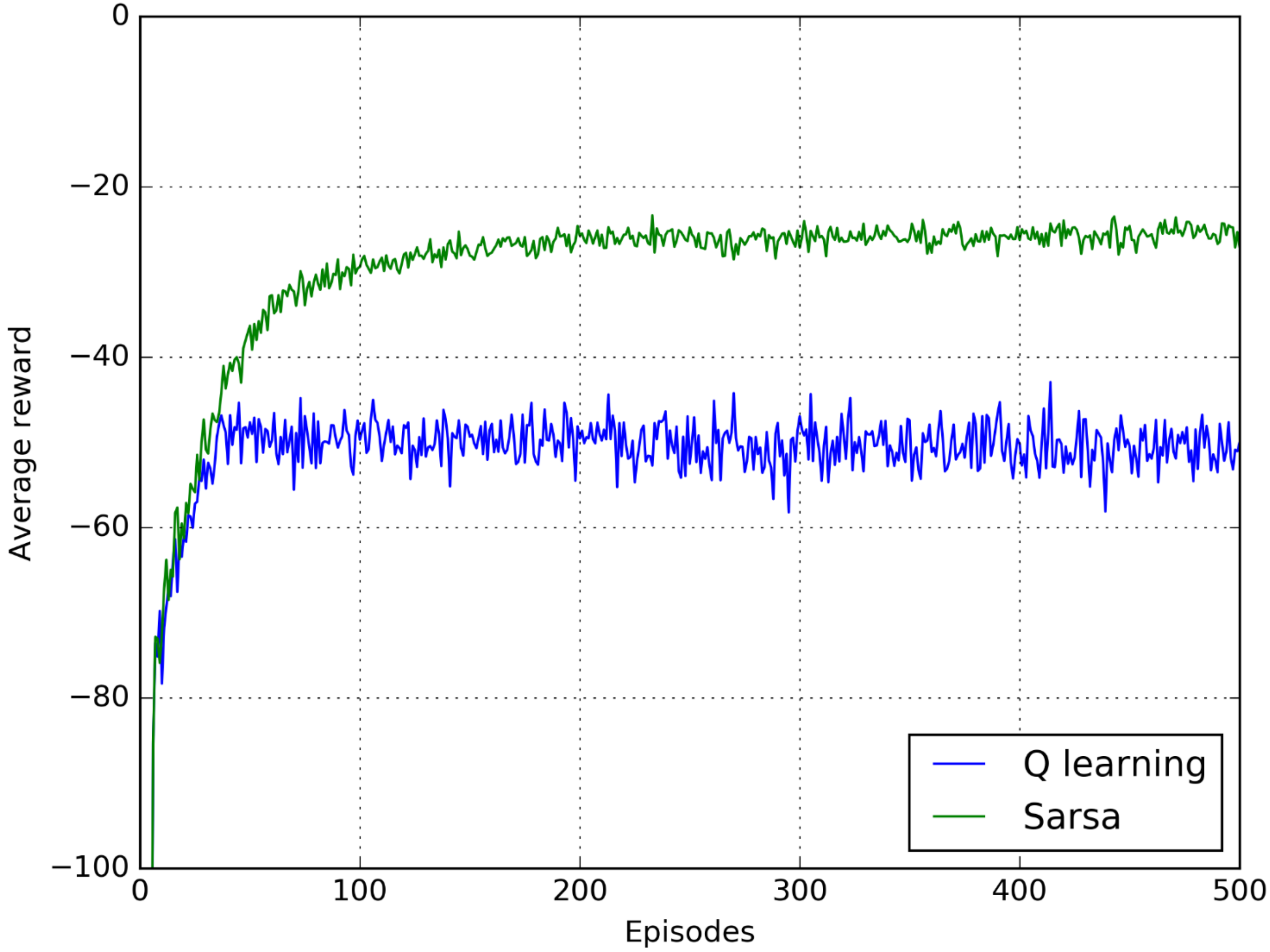


Simulation Results

$\alpha = 0.5$

$\gamma = 0.99$

$\epsilon = 0.1$



$\alpha = 0.1$

$\gamma = 0.99$

$\epsilon = 0.1$

Q Table (Q-Learning)

```
up
[[-12.2478977  -11.53619229 -10.89856116 -10.51137989]
 [-83.03446861 -10.82507719 -10.35882892 -10.10020035]
 [-51.22711726 -10.05752168  -9.71255028  -9.53598014]
 [-48.93482366  -9.26149224  -9.0062303   -8.89981986]
 [-47.26030014  -8.42641001  -8.25404562  -8.21773143]
 [-45.747969    -7.59647031  -7.47014522  -7.49867428]
 [-44.32843495  -6.7343663   -6.65173075  -6.76091935]
 [-43.04965278  -5.86077544  -5.8145235   -6.00402193]
 [-41.93487206  -5.00768858  -4.95237398  -5.24208759]
 [-40.94196848  -4.13404626  -4.07541085  -4.47149503]
 [-40.94725418  -3.25837305  -3.17376592  -3.69883474]
 [  0.          -2.44495307  -2.31986294  -2.95909303]]
```

```
left
[[-12.7373751  -11.86456393 -10.91861283 -10.51442829]
 [-82.89771476 -11.46842168 -10.41579913 -10.12919026]
 [-51.44695548 -10.58480177  -9.76857217  -9.57803561]
 [-48.96251803  -9.67241687  -9.06561643  -8.94840403]
 [-47.39127355  -8.77173546  -8.32239409  -8.26588887]
 [-45.64000226  -7.86006923  -7.54658956  -7.55798698]
 [-44.41106341  -6.95154566  -6.7297379   -6.82668666]
 [-42.93527063  -6.03205165  -5.88922236  -6.07128866]
 [-41.94174358  -5.11172989  -5.0345131   -5.31317117]
 [-40.97572413  -4.2062433   -4.14201397  -4.52186237]
 [-41.03027797  -3.29513622  -3.25622667  -3.74346426]
 [  0.          -2.44507262  -2.36505572  -2.99368731]]
```

```
down
[[-12.7270939  -12.39311949 -10.96678582 -10.55231411]
 [-82.97176677 -57.1586645  -10.3909073  -10.12672612]
 [-51.28484127 -31.56487786  -9.7180721   -9.55730846]
 [-48.8532225  -28.8558715   -8.991867    -8.91519367]
 [-47.32226778 -27.1013116   -8.22399046  -8.22694901]
 [-45.54067771 -25.31194722  -7.42149493  -7.50610672]
 [-44.45091848 -24.01381889  -6.58636891  -6.76078078]
 [-42.99984005 -22.54653601  -5.72135777  -5.99743281]
 [-42.19037053 -21.53167158  -4.82769025  -5.22196701]
 [-40.94160955 -20.55190228  -3.90623497  -4.43728185]
 [-41.05193686 -20.56442299  -2.95949277  -3.65531384]
 [  0.          -1.          -1.98991461  -2.89675371]]
```

```
right
[[-61.27629319 -11.36151283 -10.89347402 -10.50641643]
 [-82.9027089  -10.46617457 -10.33765519 -10.07788311]
 [-51.4151433   -9.5617925   -9.67996873  -9.50930842]
 [-48.93112945  -8.64827525  -8.96384417  -8.86819074]
 [-47.22809101  -7.72553056  -8.20427812  -8.18166874]
 [-45.66765008  -6.79346521  -7.40852959  -7.4643217 ]
 [-44.43544856  -5.85198506  -6.57872664  -6.72368209]
 [-43.01493522  -4.90099501  -5.71697878  -5.96682276]
 [-41.91834883  -3.940399    -4.82550058  -5.19674462]
 [-41.2167373   -2.9701      -3.90559227  -4.42287167]
 [-40.95307932  -1.99        -2.95951336  -3.65228002]
 [  0.          -1.73282528  -2.19130084  -2.95741521]]
```

Q values at the end of 500 episodes averaged over 1,000 training sessions

$\alpha = 0.1$

$\gamma = 0.99$

$\epsilon = 0.1$

Q Table (Sarsa)

```
up
[[-14.29842916 -13.45793271 -12.6489765  -12.10805576]
 [-76.09854901 -11.96286173 -11.78556147 -11.47821076]
 [-38.90795933 -10.98348689 -10.93959991 -10.7551744 ]
 [-37.14696958 -10.02456552 -10.07832299  -9.99534053]
 [-35.52656218  -9.07099829  -9.20913419  -9.20710235]
 [-34.36235558  -8.07646928  -8.3406688   -8.40039606]
 [-32.56418718  -7.12246688  -7.46546674  -7.57772865]
 [-31.54159392  -6.14421987  -6.57574708  -6.73396452]
 [-30.91216477  -5.14739953  -5.68882978  -5.87403894]
 [-30.23970385  -4.12865181  -4.78720509  -5.00362312]
 [-30.68514024  -2.93190393  -3.87790856  -4.11542683]
 [  0.          -2.5613995   -3.04934975  -3.24069655]]
```

```
left
[[-15.40202072 -13.64200354 -12.78371393 -12.11068571]
 [-76.12487689 -12.10306394 -12.1225347  -11.49301342]
 [-38.62677753 -11.36597698 -11.25517833 -10.78160218]
 [-37.25266427 -10.15296017 -10.39376904 -10.03088494]
 [-35.33801312  -9.19244443  -9.50781416  -9.24812508]
 [-34.40675903  -8.20741936  -8.60417222  -8.44653775]
 [-32.50682905  -7.25949856  -7.72556244  -7.63125245]
 [-31.61926379  -6.24972408  -6.81740988  -6.79024394]
 [-30.56648222  -5.25468209  -5.88791396  -5.93965297]
 [-29.97211834  -4.27749923  -4.95089508  -5.06786687]
 [-30.94473557  -3.09383494  -4.02495947  -4.17472476]
 [  0.          -2.64643005  -3.10955788  -3.3058261 ]]
```

```
down
[[-15.4764039  -14.66853417 -13.02619095 -12.12640457]
 [-76.09968731 -36.93681087 -12.35083729 -11.47292531]
 [-38.78964713 -18.73786216 -11.08315656 -10.73794804]
 [-37.17362566 -17.19050437 -10.17606518  -9.96020168]
 [-35.51496178 -15.61593104  -9.24903908  -9.16356647]
 [-34.14312518 -14.60710104  -8.31594309  -8.3440964 ]
 [-32.55298986 -13.10858704  -7.36307662  -7.50938886]
 [-31.46509799 -12.28713745  -6.4057857   -6.66049663]
 [-30.9105218  -11.63446439  -5.44970554  -5.79349034]
 [-30.05595622 -11.06640804  -4.48456363  -4.91152419]
 [-30.86689022 -11.53035368  -3.51727924  -4.02012952]
 [  0.          -1.          -2.08717924  -3.10074873]]
```

```
right
[[-56.69802733 -14.28135727 -12.59052267 -12.04416026]
 [-76.13351646 -11.96777098 -11.69317051 -11.40126721]
 [-38.86869433 -10.92094611 -10.79794863 -10.67452114]
 [-37.09809202  -9.94023898  -9.88613027  -9.90864439]
 [-35.33715031  -8.94923332  -8.96166791  -9.11718068]
 [-34.12627782  -7.90326284  -8.02557734  -8.30700232]
 [-32.43120475  -6.8904646   -7.07575683  -7.47962031]
 [-31.50876236  -5.82834617  -6.11227261  -6.63541911]
 [-30.69472763  -4.68316638  -5.13154978  -5.7751099 ]
 [-30.07981419  -3.46877018  -4.13420863  -4.89797122]
 [-30.65804689  -2.08424001  -3.1346809   -4.00612304]
 [  0.          -1.78501234  -2.61823246  -3.24841893]]
```

Q values at the end of 500 episodes averaged over 1,000 training sessions

Assignment

- Download problem set #3 from KLMS
  - Due: 11/15 (Wed) 1:00pm
- Future schedule
  - Project #2 (Reinforcement learning)
    - Due: November 29 (Wed) 1:00pm
  - Project #3 (AlphaGo)
    - Due: December 17 (Sun) 11:59pm