SARSA vs Q-learning

SARSA Algorithm

ϵ-greedy policy improvement done for TD methods

CS234
lecture 4

Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$

Take $a_t \sim \pi(s_t)$ // Sample action from policy

Observe (r_t, s_{t+1})

loop

  Take action $a_{t+1} \sim \pi(s_{t+1})$

  Observe (r_{t+1}, s_{t+2})

  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

  $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random

E-greedy policy update?

  $t = t + 1$

end loop

Loop Q value update for every time-step : based on TD methods

Sutton
and
Barto

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Action chosen by e-greedy

Action taken for next state already decided

Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
Take $a_t \sim \pi(s_t)$ // Sample action from policy
Observe (r_t, s_{t+1})
loop
  Take action $a_{t+1} \sim \pi(s_{t+1})$
  Observe (r_{t+1}, s_{t+2})
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
  $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
  $t = t + 1$
end loop

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

CS234
lecture 4

Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
Take $a_t \sim \pi(s_t)$ // Sample action from policy
Observe (r_t, s_{t+1})
loop
  Take action $a_{t+1} \sim \pi(s_{t+1})$
  Observe (r_{t+1}, s_{t+2})
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
  $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
  $t = t + 1$
end loop

E-greedy policy update?

Loop Q value update for every time-step : based on TD methods

Sutton
and
Barto

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Action chosen by e-greedy

Action taken for next state already decided

Difference?

CS234
lecture 4

Seems to indicate that actions are taken greedily

Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
Take $a_t \sim \pi(s_t)$ // Sample action from policy
Observe (r_t, s_{t+1})
loop
    Take action $a_{t+1} \sim \pi(s_{t+1})$
    Observe (r_{t+1}, s_{t+2})
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
    $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
    $t = t + 1$
end loop

No indication of how next action is chosen

But can assume greedy action choosing

E-greedy policy update?

Loop Q value update for every time-step : based on TD methods

But this would mean q-value has been calculated for all actions at state st

Sutton and Barto

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

Action chosen by e-greedy

Action taken for next state already decided

lecture 4

stop

Simple difference between SARSA and Q-learning

SARSA = on-policy
Q-learning = off-policy

SARSA = on-policy

on-policy = on e-greedy policy

on-policy = learn to estimate and evaluate policy from experience obtained from following that policy (purely from experience)

CS234 lecture 4

1: Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
2: Take $a_t \sim \pi(s_t)$ // Sample action from policy
3: Observe (r_t, s_{t+1})
4: **loop**
5: Take action $a_{t+1} \sim \pi(s_{t+1})$
6: Observe (r_{t+1}, s_{t+2})
7: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
8: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
9: $t = t + 1$
10: **end loop**

E-greedy policy update?

Loop Q value update for every time-step : based on TD methods

Sutton and Barto

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

Action chosen by e-greedy

Action taken for next state already decided

SARSA = on-policy

on-policy = on e-greedy policy

on-policy = learn to estimate and evaluate policy from experience obtained from following that policy (purely from experience)

CS234
lecture 4

Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
Take $a_t \sim \pi(s_t)$ // Sample action from policy
Observe (r_t, s_{t+1})
loop
  Take action $a_{t+1} \sim \pi(s_{t+1})$
  Observe (r_{t+1}, s_{t+2})
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
  $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
  $t = t + 1$
end loop

E-greedy policy update?

Loop Q value update for every time-step : based on TD methods

Seems to be experience following a different policy (trying out all other actions)

Sutton
and
Barto

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Action chosen by e-greedy

Action taken for next state already decided

The policy on the lefthandside seems stochastic?

8: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1-\epsilon$, else random

When choosing action from this policy

If pi takes a single action = either argmax or random by epsilon

Then

pi = Deterministic policy

Choose A from S using policy derived from Q (e.g., ε-greedy)

Action already pre-determined

Conditions of SARSA:

GLIE
Robbin-Munro

lecture 4

Conditions of SARSA:

GLIE
Robbin-Munro

Definition of GLIE

- All state-action pairs are visited an infinite number of times

$$\lim_{i \to \infty} N_i(s, a) \to \infty$$

- Behavior policy converges to greedy policy
  $\lim_{i \to \infty} \pi(a|s) \to \arg\max_a Q(s, a)$ with probability 1

Basically being able to explore all actions

= use method of e-greedy where e starts big and then diminishes

All s, a are visited infinitely often

Conditions of SARSA:

GLIE
Robbin-Munro

2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$
$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Learning rate not too big..?

Conditions of SARSA:

GLIE
Robbin-Munro

2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Learning rate not too big..?

Alpha = 0 : no learning

Alpha = 1 : new experience complete overtakes

SARSA → Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

Bootstrapping

lecture 4

Initialize $Q(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$

Set π_b to be ϵ-greedy w.r.t. Q

loop

  Take $a_t \sim \pi_b(s_t)$ // Sample action from policy

  Observe (r_t, s_{t+1})

  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \arg\max_a Q(s_{t_1}, a) - Q(s_t, a_t))$

  $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random

  $t = t + 1$

end loop

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 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, 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., 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-learning = off policy

Off-policy = estimate and evaluate policy from following a different policy

Initialize $Q(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
Set π_b to be ϵ-greedy w.r.t. Q
loop
    Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
    Observe (r_t, s_{t+1})
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \arg\max_a Q(s_{t_1}, a) - Q(s_t, a_t))$
    $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
    $t = t + 1$
end loop

Not necessarily the same policy

lecture 4

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 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, 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., 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

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Not necessarily the same policy

Difference = random by epsilon?

lecture 4

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 $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, 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., ε-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

Not necessarily the same policy

Difference =

No matter what next action chosen for evaluating current state,

Next action will be decided with the updated policy (new and different from that used to evaluate)

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

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

When evaluated with next action chosen, the chosen next action will be used on next step

Q-learning tends to have better results

For most cases...

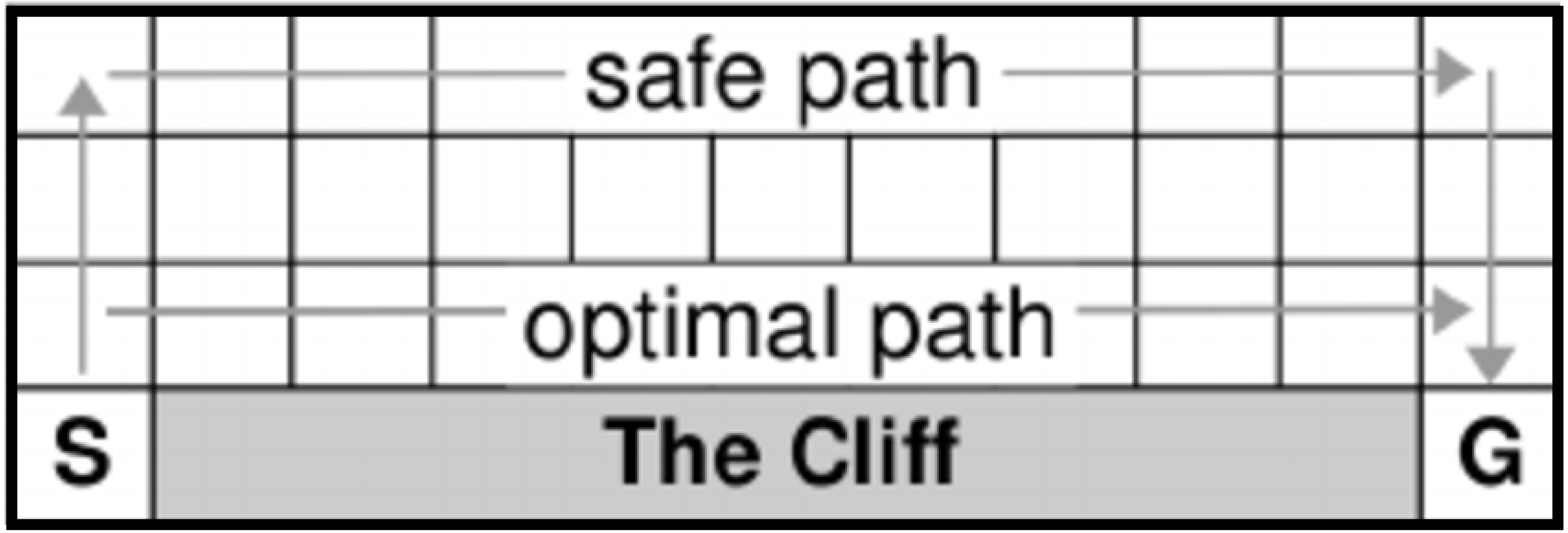
But for some cases…

Why SARSA rather than Q-learning?

Sutton and Barto example of cliff-walking

SARSA may do better on early stages

SARSA is realistic
Q-learning is optimistic



Case where there could be a lot of negative outcomes from optimism of using max

Optimism would mean being only interested in the best outcome

Optimism on Cliff for example:

Near cliff is the shortest route to finish = highest possible reward & but forgets that it also has high chance of falling

SARSA Code breakdown

```
import gym
import gym_gridworlds
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register
import random as pr


env = gym.make('Cliff-v0')
```

Import package that makes & registers Cliff environment

Create element

SARSA Code breakdown

Q value (state-action value) is defined by value that comes from action taken at current state

= preparing Table for all possible states and actions

```
# Q-table
# note that:
# action = [up, right, down, left]
Q = np.zeros([env.height , env.width, env.action_space.n])
```

SARSA Code breakdown

```
# Q-table
# note that:
# action = [up, right, down, left]
Q = np.zeros([env.height , env.width, env.action_space.n])
```

Current state can be defined by current position (x, y)

SARSA Code breakdown

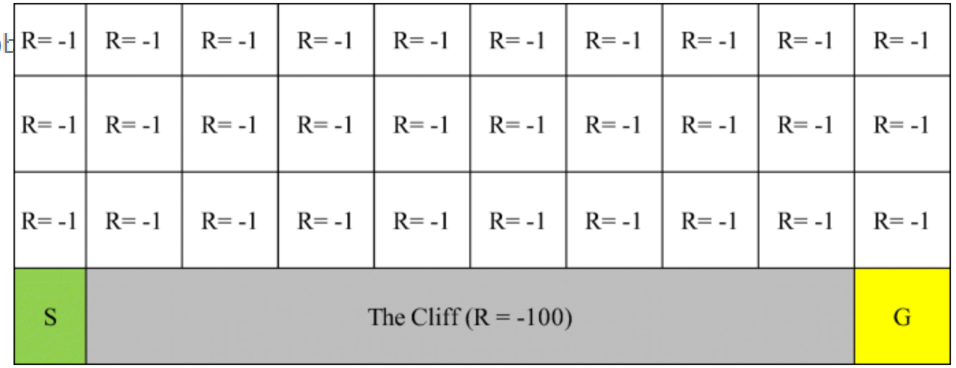
```
# define hyperparameters (epsilon defined as function later)
dis = .91
num_episodes = 1000
lr = 0.01
success = 0

# create lists to contain total rewards and steps per episode
rList = []
```

= to track how total reward changes

SARSA Code breakdown

```python
def reset(self):
    self.S = (3, 0)
    return self.S
```

```python
for i in range(num_episodes):
    # Reset environment and get first new ob
    (y, x) = env.reset()
    # state = y , x
    rAll = 0
    done = False

    e = 1. / ((i // 100) + 1)
    # e-greedy where epsilon starts big
    # and becomes small
    # = more exploration in the beginning and less later on
    # with larger num_episodes, this might come into better effect

    # note that this is a method to satisfy condition of GLIE
```



SARSA Code breakdown

```
for i in range(num_episodes):
    # Reset environment and get first new observation
    (y, x) = env.reset()
    # state = y , x
    rAll = 0
    done = False

    e = 1. / ((i // 100) + 1)
    # e-greedy where epsilon starts big
    # and becomes small
    # = more exploration in the beginning and less later on
    # with larger num_episodes, this might come into better effect

    # note that this is a method to satisfy condition of GLIE
```

Done = reached destination

When done = True, episode is terminated

As #episode trained increases,

Epsilon decreases

SARSA Code breakdown

```python
# e-greedy
if np.random.rand(1) < e:
    action = env.action_space.sample()
else:
    action = np.argmax(Q[y, x, :])
```

SARSA Code breakdown

```
    # The Q-Table learning algorithm
    while not done:
        # Get new state and reward from environment
        (new_y, new_x), reward, done, _ = env.step(action)
        # new_state = new_x , new_y
        # Choose the next action by e greedy
        if np.random.rand(1) < e:
            next_action = env.action_space.sample()
        else:
            next_action = np.argmax(Q[new_y, new_x, :])
```

SARSA Code breakdown

```
    # The Q-Table learning algorithm
    while not done:
        # Get new state and reward from environment
        (new_y, new_x), reward, done, _ = env.step(action)
```

```
    def step(self, action):
        x, y = self.moves[action]
        self.S = self.S[0] + x, self.S[1] + y

        self.S = max(0, self.S[0]), max(0, self.S[1])
        self.S = (min(self.S[0], self.height - 1),
                  min(self.S[1], self.width - 1))

        if self.S == (self.height - 1, self.width - 1):
            return self.S, -1, True, {}
        elif self.S[1] != 0 and self.S[0] == self.height - 1:
            # the cliff
            return self.S, -100, True, {}
        return self.S, -1, False, {}
```

```
        self.moves = {
                0: (-1, 0),   # up
                1: (0, 1),   # right
                2: (1, 0),  # down
                3: (0, -1),  # left
                }
```

SARSA Code breakdown

```
        # Update Q-Table with new knowledge using learning rate
        Q[y, x, action] = (Q[y, x, action] + lr * (reward + dis * Q[new_y, new_x, next_action] - Q[y, x, action]))

        rAll += reward
        y, x = new_y, new_x

        # difference with Q-learning
        # action is already chosen for updating current state-action value
        action = next_action
        # therefore the loop continues by choosing only the next action

    rList.append(rAll)
```

Evaluate using
next_action determined
by the current policy

SARSA Code breakdown

```
        # Update Q-Table with new knowledge using learning rate
        Q[y, x, action] = (Q[y, x, action] + lr * (reward + dis * Q[new_y, new_x, next_action] - Q[y, x, action]))

        rAll += reward
        y, x = new_y, new_x

        # difference with Q-learning
        # action is already chosen for updating current state-action value
        action = next_action
        # therefore the loop continues by choosing only the next action

    rList.append(rAll)
```

Then use the next_action for next_state in next timestep

SARSA Code breakdown

```
        # Update Q-Table with new knowledge using learning rate
        Q[y, x, action] = (Q[y, x, action] + lr * (reward + dis * Q[new_y, new_x, next_action] - Q[y, x, action]))

        rAll += reward
        y, x = new_y, new_x

        # difference with Q-learning
        # action is already chosen for updating current state-action value
        action = next_action
        # therefore the loop continues by choosing only the next action

    rList.append(rAll)
```

Loop until termination

SARSA Code breakdown

```
print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print(Q)
print("Q Values near starting position")
print("START\n" + str(Q[3,0:1]))
print("Above START\n" + str(Q[2,0:2]))
print("Above above START\n" + str(Q[1,0:2]))
plt.bar(range(len(rList)), rList, color="blue")
plt.show()
```

Q-learning Code breakdown

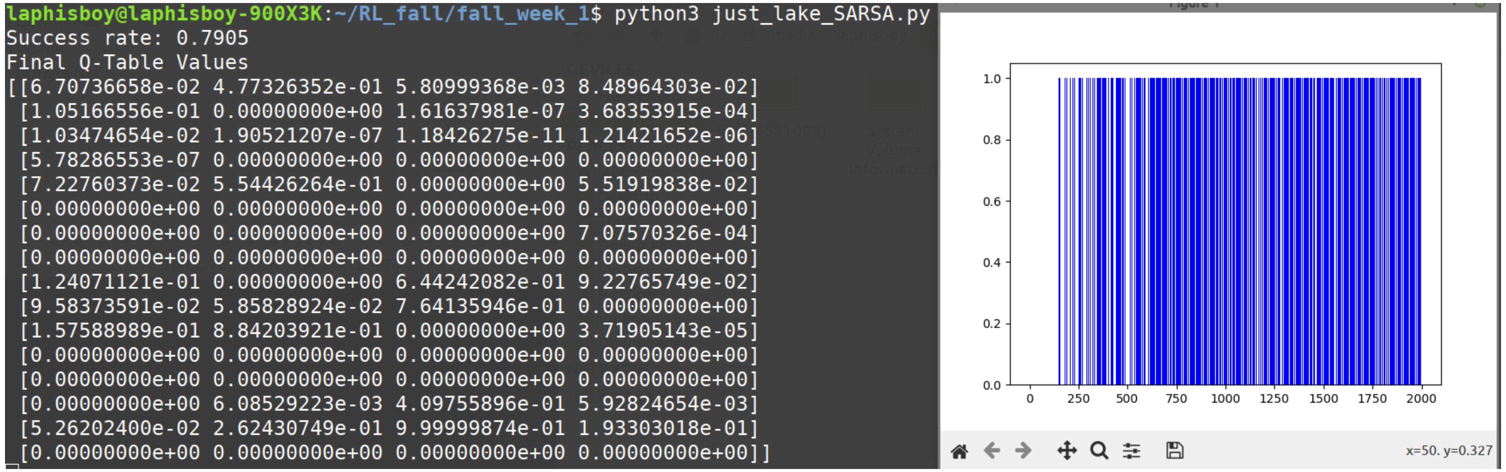
```
# The Q-Table learning algorithm
while not done:

    if np.random.rand(1) < e:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q[y, x, :])
    # Get new state and reward from environment
    (new_y, new_x), reward, done, _ = env.step(action)
    # new_state = new_x , new_y

    # Update Q-Table with new knowledge using learning rate
    Q[y, x, action] = (Q[y, x, action] + lr * (reward + dis * np.max(Q[new_y, new_x, :]) - Q[y, x, action]))

    rAll += reward
    y, x = new_y, new_x

rList.append(rAll)
```

Second difference

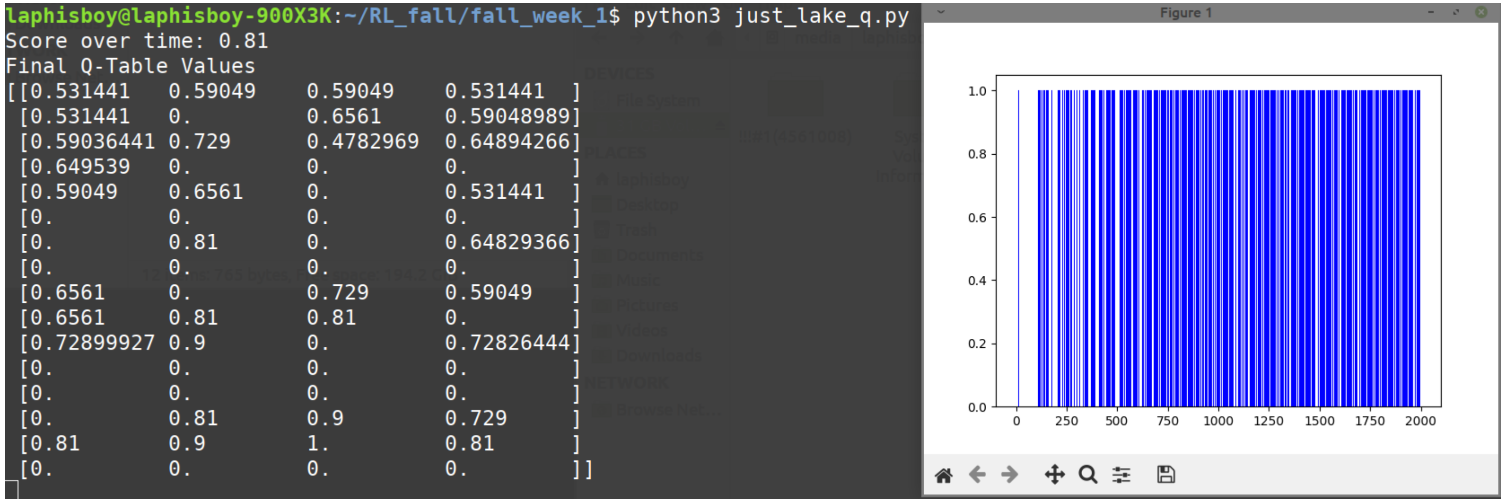
Main difference

Q-learning tends to have better results

SARSA



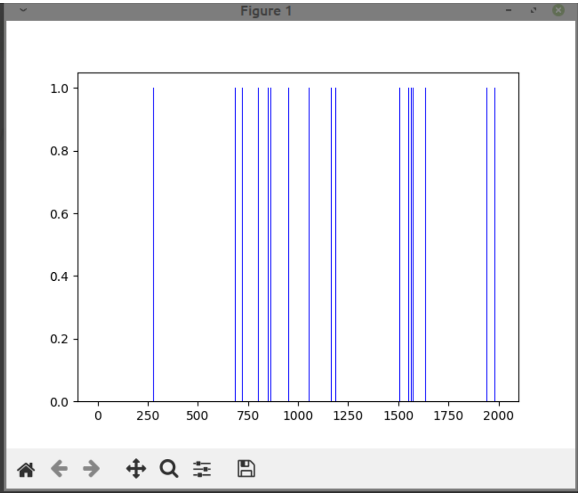
Q-Learning



Q-learning tends to have better results

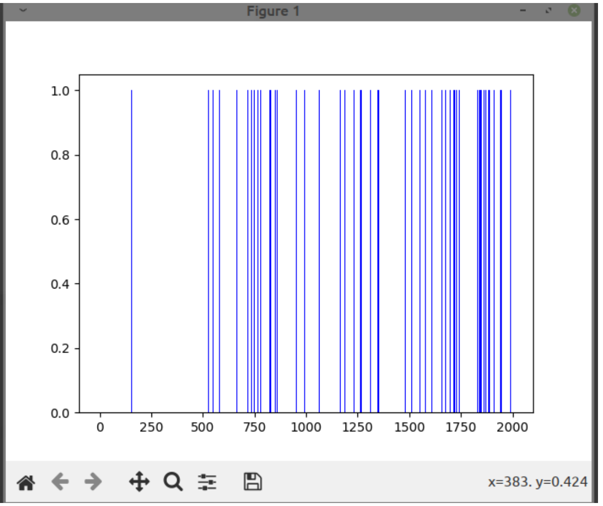
SARSA

```
laphisboy@laphisboy-900X3K:~/RL_fall/fall_week_1$ python3 frozen_lake_SARSA.py
Success rate: 0.0545
Final Q-Table Values
[[1.55087990e-03 6.78733437e-03 1.30792462e-03 9.31839567e-04]
 [1.04827592e-04 3.51441482e-03 2.05297738e-04 4.02499257e-04]
 [3.87858024e-04 7.99769647e-03 5.85235056e-04 5.91762768e-05]
 [1.26320704e-04 1.84382247e-03 5.84039809e-06 3.39594415e-05]
 [1.38777383e-02 1.73185685e-03 2.00942524e-03 6.09587098e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [2.71560631e-02 1.22320626e-10 1.33757456e-03 1.99071938e-05]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.78146584e-03 2.79410650e-03 2.46060202e-03 2.91133741e-02]
 [3.38610920e-03 5.83086773e-03 6.36602543e-02 3.06886333e-03]
 [6.92414686e-03 1.09936351e-01 3.59391950e-03 2.88692781e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [3.52018257e-03 1.28705679e-01 1.44792346e-02 1.61855613e-02]
 [3.28335009e-03 3.20059102e-02 4.79574003e-03 3.53480997e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```



Q-Learning

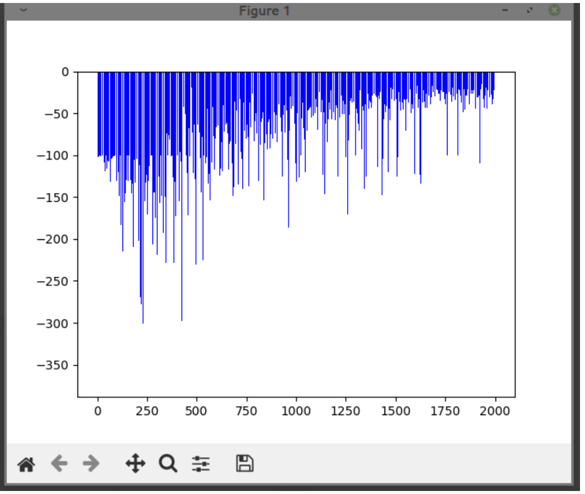
```
laphisboy@laphisboy-900X3K:~/RL_fall/fall_week_1$ python3 frozen_lake_q.py
Score over time: 0.1205
Final Q-Table Values
[[4.30972397e-02 8.68329874e-02 5.16130150e-03 1.39912677e-02]
 [8.41989861e-04 1.11620981e-03 2.78075826e-04 4.70342591e-02]
 [1.04278780e-03 6.54248387e-03 1.70384432e-03 1.56023476e-03]
 [1.34461696e-03 3.82147058e-05 1.81494164e-04 1.63379968e-03]
 [4.92277658e-02 1.55019952e-03 1.01424568e-02 3.88767743e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [5.30290511e-07 6.58845984e-07 8.85605526e-04 2.62413302e-06]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.11079066e-03 1.84759192e-02 3.67341763e-02 3.01863580e-01]
 [4.05955922e-04 5.21394469e-01 3.49797214e-02 5.06608931e-02]
 [1.12056118e-01 2.71397368e-06 8.23377276e-06 2.83359708e-02]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [2.11084497e-02 3.40265637e-01 8.54451439e-01 2.59187179e-03]
 [1.73027131e-01 9.89435350e-01 1.48444394e-01 4.63286997e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```



Why SARSA rather than Q-learning?

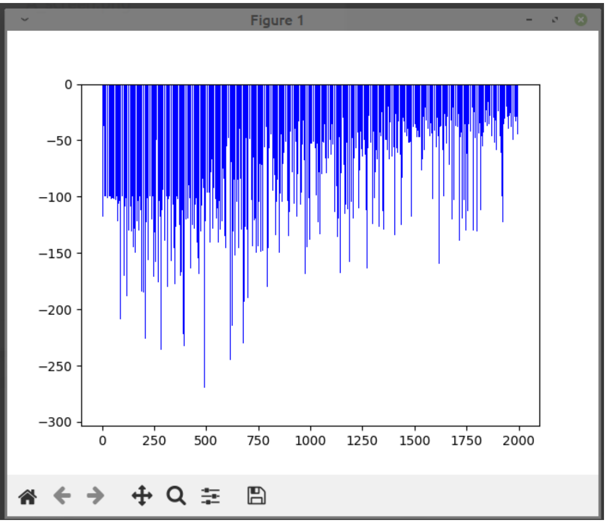
SARSA

```
laphisboy@laphisboy-900X3K:~/RL_fall/fall_week_1$ python3 cliff_SARSA.py
Success rate: -78.4395
Final Q-Table Values
[[[ -6.57224026  -6.56897259  -6.57255013  -6.57016348]
  [ -6.40792615  -6.40794207  -6.41262707  -6.40722306]
  [ -6.19563068  -6.19150736  -6.19055137  -6.19008469]
  [ -5.94812063  -5.9433751   -5.94397375  -5.94298732]
  [ -5.66411456  -5.65894466  -5.65843276  -5.66155133]
  [ -5.35078124  -5.34704226  -5.34938056  -5.35235109]
  [ -4.99885228  -4.99516469  -4.99846572  -5.00306954]
  [ -4.600921    -4.59841477  -4.59874203  -4.5972695 ]
  [ -4.152033    -4.15241057  -4.154525    -4.15715972]
  [ -3.66646721  -3.66500498  -3.66553048  -3.66373331]
  [ -3.15142676  -3.14236339  -3.14232914  -3.14794916]
  [ -2.62328346  -2.62218288  -2.6148902   -2.62865687]]

 [[ -6.7159586   -6.71529783  -6.71919877  -6.71900845]
  [ -6.47374734  -6.47359043  -6.84136547  -6.47542736]
  [ -6.21253041  -6.21002613  -6.21058457  -6.21021332]
  [ -5.9561723   -5.95173097  -6.02177763  -5.95312993]
```



Q-Learning

```
laphisboy@laphisboy-900X3K:~/RL_fall/fall_week_1$ python3 cliff_q.py
Success rate: -90.979
Final Q-Table Values
[[[ -5.65282542  -5.65361114  -5.65574407  -5.65471412]
  [ -5.53672858  -5.5375399   -5.54061073  -5.53960431]
  [ -5.35812053  -5.35881434  -5.3613277   -5.36680103]
  [ -5.14629029  -5.14456665  -5.14268356  -5.14482495]
  [ -4.89721249  -4.89824816  -4.8961145   -4.89803612]
  [ -4.62018105  -4.61822491  -4.61625788  -4.62235642]
  [ -4.30511984  -4.30426263  -4.30310824  -4.30619001]
  [ -3.95317341  -3.95352794  -3.95746675  -3.95710484]
  [ -3.56627702  -3.56746228  -3.56728727  -3.57233242]
  [ -3.15481121  -3.15248086  -3.15664835  -3.15322673]
  [ -2.73066757  -2.72614154  -2.72950243  -2.7296559 ]
  [ -2.35850521  -2.35794923  -2.35953948  -2.3586992 ]]

 [[ -5.76232745  -5.76456868  -5.76388702  -5.76406465]
  [ -5.60076494  -5.60036745  -5.60415485  -5.60290239]
  [ -5.39606477  -5.39335139  -5.39800854  -5.39444687]
  [ -5.16188336  -5.15905796  -5.16290388  -5.16118663]
```



Why SARSA rather than Q-learning?

```
# action = [up, right, down, left]
```

https://github.com/laphisboy/RL_fall/blob/master/fall_week_1/cliff_SARSA_text.txt

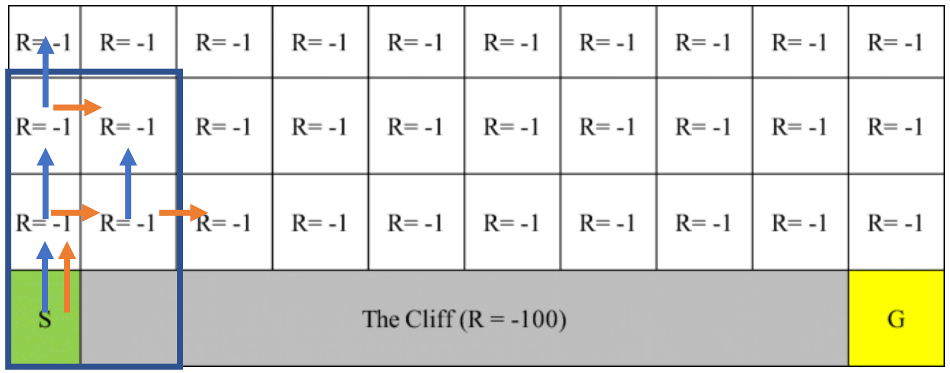
https://github.com/laphisboy/RL_fall/blob/master/fall_week_1/cliff_q_text.txt

SARSA

```
Q Values near starting position
START
[[ -5.65773872 -73.19532831  -6.58676967  -6.09652006]]
Above START
[[ -5.4619545   -5.60901178  -5.85142828  -5.46444349]
 [ -4.68087526  -4.90508453 -44.73165228  -4.67900451]]
Above above START
[[-5.18131607 -5.18191879 -5.17902335 -5.18519691]
 [-4.94702408 -4.94652235 -5.08051739 -4.95036558]]
```

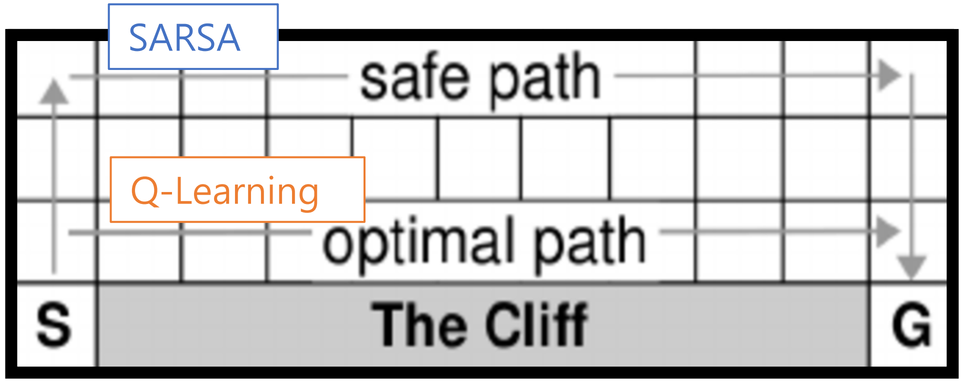
Q-Learning

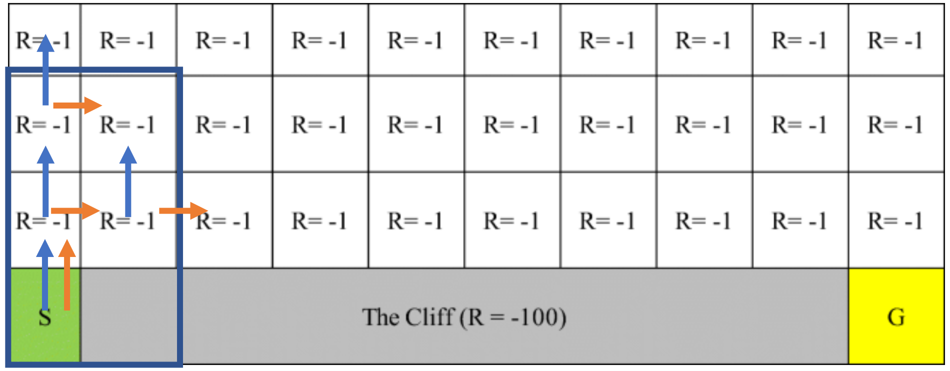
```
Q Values near starting position
START
[[ -4.47977623 -87.76072562  -4.48266823  -4.47962649]]
Above START
[[ -4.12097592  -4.12070405  -4.11925462  -4.11693268]
 [ -3.8334134   -3.83093316 -57.86657778  -3.83874233]]
Above above START
[[-3.90748219 -3.90242961 -3.90697297 -3.90700077]
 [-3.75335178 -3.75719326 -3.75394856 -3.75815594]]
```



Why SARSA rather than Q-learning?

```
# action = [up, right, down, left]
```





SARSA → Q-Learning

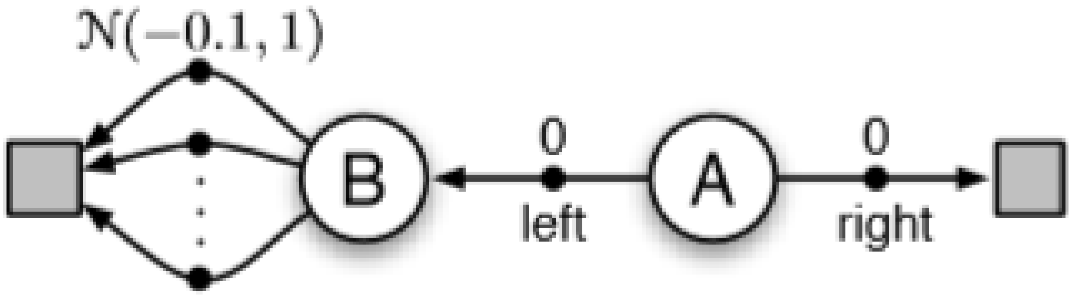
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \boxed{\max_{a'}} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

Leads to positive bias

= Maximization Bias

Maximization Bias

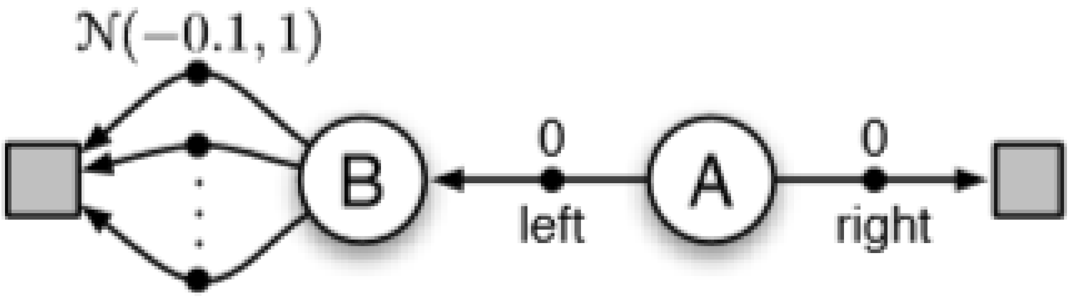


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

Q. Both have maximization bias right?

Maximization Bias



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

Q. Both have maximization bias right?

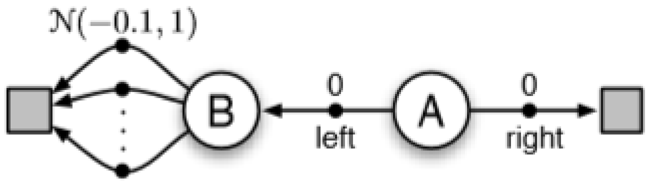
A. Yes because policy is updated with action with maximum Q-value = optimism in e-greedy policy

B. But SARSA is less optimistic while evaluation of given action

Q-Learning → Double Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big]$$



Q7. What is double Q-learning? And how does it overcome maximization bias?

Double Q learning?

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$

loop

  Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t,a) + Q_2(s_t,a)$

  Observe (r_t, s_{t+1})

  if (with 0.5 probability) **then**

    $Q_1(s_t,a_t) \leftarrow Q_1(s_t,a_t) + \alpha(r_t + \gamma \max_a Q_2(s_{t+1},a)$

  else

    $Q_2(s_t,a_t) \leftarrow Q_2(s_t,a_t) + \alpha(r_t + \gamma \max_a Q_1(s_{t+1},a)$

  end if

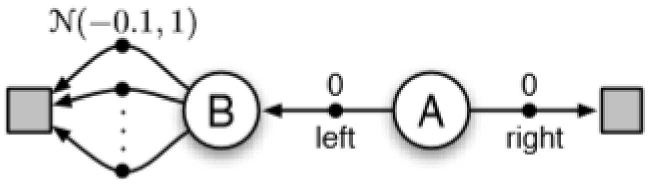
  $t = t + 1$

end loop

Q-Learning → Double Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$



Q7. What is double Q-learning? And how does it overcome maximization bias?

한 번 왼쪽 action 에서 0보다 큰 값이 나오면 지속적으로 argmax가 left action이 되는 문제가 생김 (true action(?) = right)

Double Q-learning 은 해결이 아닌 완화 method으로 이해…

참고

Base code https://github.com/hunkim/ReinforcementZeroToAll

See how SARSA is implemented https://github.com/rlcode/reinforcement-learning/tree/master/1-grid-world

Cliff environment https://github.com/podondra/gym-gridworlds

Difference between those two https://tcnguyen.github.io/reinforcement_learning/sarsa_vs_q_learning.html

코드

https://github.com/laphisboy/RL_fall/tree/master/fall_week_1