

Reinforcement Learning

Wprowadzenie

Jeremi Kaczmarczyk

Rafał Nowak

Piotr Semberecki

by Tooploox AI

Agenda:

1. Program / Agenda
2. Wprowadzenie do OpenAI
3. Gra ?????
4. Metody Monte Carlo
5. Temporal Difference Learning

Cele tego szkolenia

- » Zapoznanie z tematem Reinforcement Learningu
- » Zapoznanie z technikami
- » Stworzyć okazje do zaprogramowania podstawowych algorytmów/technik RL

Zasady:

- » Prezentacja (hangout – view) + ludzie zdalni będą zmutowani podczas rozwiązywania zadań a w przerwach pomiędzy zadaniami mamy czas na dyskusję i wtedy będziemy ich odblokowywali
- » Colab – jupyter Google:
- » Slack kanał #rl-workshop

Reinforcement Learning

“Reinforcement Learning – jest to uczenie co zrobić – jak dopasować sytuacje do akcji aby zmaksymalizować numeryczny sygnał nagrody”

» Reinforcement Learning: An Introduction 2nd ed.

State - Stan

S, S', S_t

Stan określamy symbolem s i jest to obecna sytuacja w jakiej znajduje się środowisko. Jako s' oznaczamy stan będący rezultatem stanu s , natomiast s_t jest to stan dla danego kroku.

Action – Akcja

a, a_t

Znajdując się w stanie s możemy wykonać akcję a . Akcja powoduje zmianę stanu z s do s' .

Reward – Nagroda

r, r_t

Po wykonaniu akcji otrzymujemy nagrodę r od środowiska. Nagrodę otrzymujemy po każdym kroku i niekoniecznie jest pozytywna.

Projektowanie sygnału nagrody ma kluczowe znaczenie przy rozwiązywaniu problemów Reinforcement Learningiem.

Policy – Polityka

$$\pi, \pi(s), \pi(a|s)$$

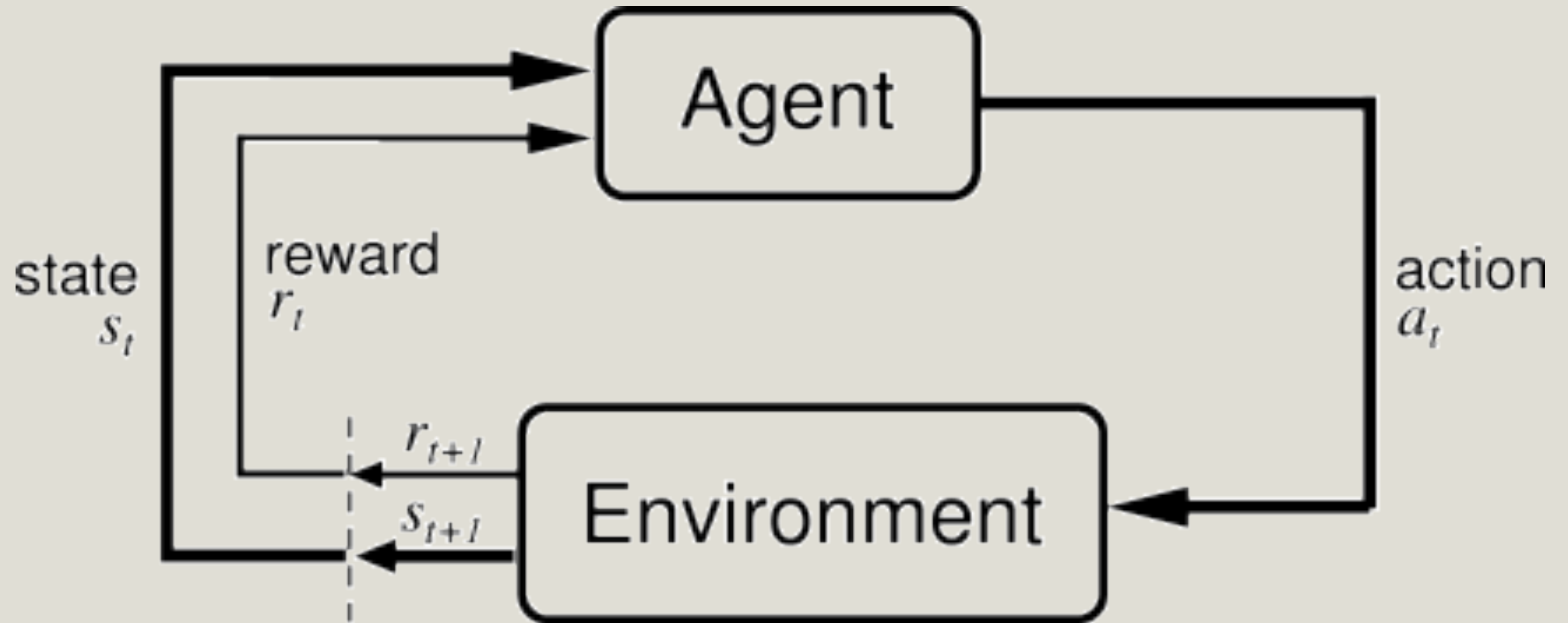
Polityka definiuje zachowanie w danym momencie. Jest to funkcja parametryzowana zwykle stanem lub parą stan-akcja. W przypadku prostych problemów jest to zwykle słownik. Zwraca akcję którą agent powinien wykonać w danym stanie albo prawdopodobieństwa wykonania każdej z akcji.

Value - Wartość

$$v_{\pi}(s), q_{\pi}(s, a)$$

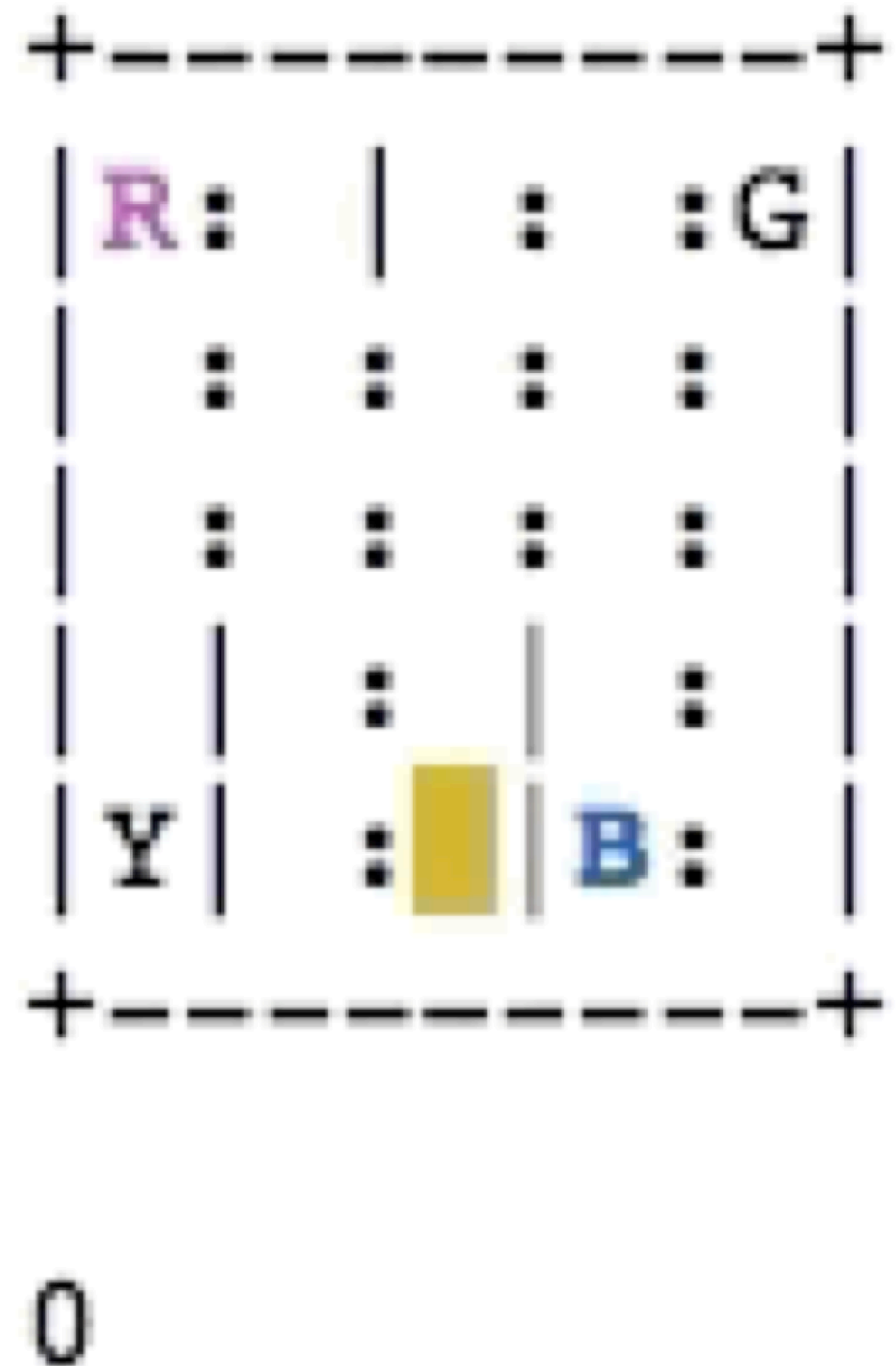
Wartość określana jest w stosunku do danej polityki π . Określana jako v jeśli jest to wartość dla stanu, albo q jeśli dla pary stan-akcja. Jest to numeryczna wartość określająca jak dobrze jest być w danym stanie albo inaczej jaka jest średnia skumulowana nagroda możliwa w danym stanie lub dla danej pary stan-akcja.

Agent – Środowisko

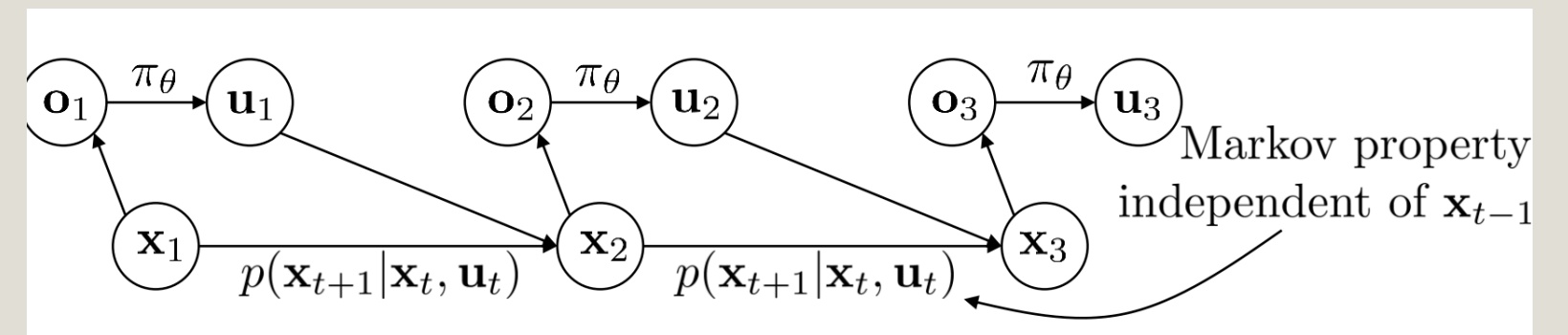
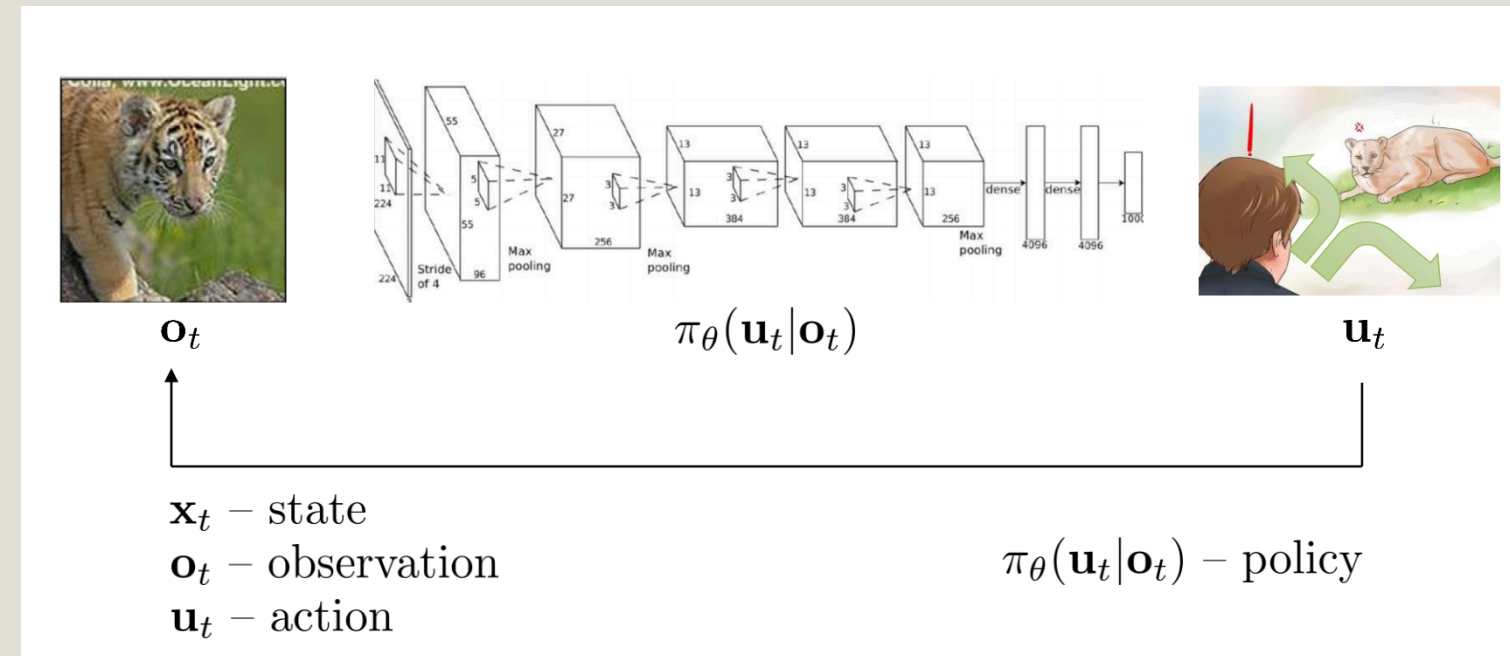


Taxi

- » : możemy przejść, przez | nie
- » R, G, Y, B miejsca podnoszenia / zostawienia pasażerów
- » +20 nagrody za sukces
- » -10 nagrody za nielegalne podniesienie / zostawienie
- » -1 nagrody za każdy ruch



Markov Decision Process – MDP



OpenAI Gym

“Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.”

» Gym website <https://gym.openai.com>

Installing

```
pip install gym
```

OpenAI Gym

```
done = False
while not done:
    env.render()
    time.sleep(.5)
    clear_output(True)
    observation, reward, done, info = env.step(env.action_space.sample())
```

```
+-----+
|R:  | :  :G|
| :  :  :  |
| :  :  :  |
| |  :  |  |
|Y|  :  |B:  |
+-----+
(South)
```

First Visit Monte Carlo Predicion

First Visit Monte Carlo Predicion

Input: `policy`

Output: `state-action dictionary Q`

Initialize **N** - dictionary where keys are states and values are number of visits to states

Initialize **returns_sum** - dictionary where keys are states and values are total rewards gained for that state

for **given number of episodes**:

 Generate whole **episode** ($S_0, A_0, R_1, S_2, A_2, \dots S_T$) using **policy**

 for every step **t** (S_t, A_t, R_t) in **episode**:

 if (S_t, A_t) is first visit in **current episode**:

$N(S_t, A_t) += 1$

$\text{returns_sum}(S_t, A_t) += R_t + \text{sum of discounted future rewards}$

$Q = \text{returns_sum}(s, a) / N(s, a)$

return Q

First Visit Monte Carlo Control

First Visit Monte Carlo Control

Output: state-action dictionary Q

Initialize **N** - dictionary where keys are states and values are number of visits to states

Initialize **returns_sum** - dictionary where keys are states and values are total rewards gained for that state

for **given number of episodes**:

$\epsilon = \epsilon(i)$

$\text{policy} = \text{epsilon greedy policy}(Q)$

Generate whole **episode** ($S_0, A_0, R_1, S_2, A_2, \dots, S_T$) using **policy**

for every step **t** (S_t, A_t, R_t) in **episode**:

if (S_t, A_t) is first visit in **current episode**:

$Q(S_t, A_t) += \alpha * (\text{sum of discounted future rewards} - Q(S_t, A_t))$

return Q

Exploration vs Exploitation

TD-Learning (SARSA)

SARSA

Input: policy, number of episodes, alpha, GLIE

Output: state-action dictionary Q

Initialize Q - with 0

for **given number of episodes:**

epsilon = **epsilon**(i)

 Observe **S0**

 Choose **action A0** using epsilon greedy policy based on Q

t = 0

repeat until terminal state:

 Take action **At** and observe **Rt+1** and **St+1**

 Choose **action At+1** using epsilon greedy policy based on Q

$Q(\text{St}, \text{At}) \mathrel{+}= \text{alpha} (\text{Rt+1} + \text{gamma } Q(\text{St+1}, \text{At+1}) - Q(\text{St}, \text{At}))$

t += 1

return Q

TD-Learning (Q-Learning)

Q-Learning

Input: policy, number of episodes, alpha, GLIE

Output: state-action dictionary Q

Initialize Q - with 0

for **given number of episodes:**

epsilon = **epsilon**(i)

 Observe **S**0

t = 0

repeat until terminal state:

 Choose **action At** using epsilon greedy policy based on Q

 Take action **At** and observe **Rt+1** and **St+1**

$Q(\text{St}, \text{At}) += \text{alpha} (\text{Rt}+1 + \text{gamma} \text{argmax}(Q(\text{St}, \text{a})) - Q(\text{St}, \text{At}))$

t += 1

return Q