

#### Master's degree in Control System Engineering

# Reinforcement Learning LAB 1

#### k-armed Bandits



Niccolò Turcato niccolo.turcato@phd.unipd.it



# Github repo

☐ ShangtongZhang / reinforcement-learning-an-introduction Public						
<> Code	O Issues 11	ຸງ Pull requests	Actions	Projects	! Security	<u> ✓</u> Insights
	₽ master <del>-</del>	₽1 branch 🛇 (	<b>)</b> tags			
ShangtongZhang Update README.md c7cc538 on May 11						314 commits
chapter01 pythonic for chap01						3 years ago
chapter02 Improved the display of the Greek alphabets.						2 years ago
chapter(	03	add state labels	ld state labels			2 years ago
chapter(	chapter04 Fix a plotting issue, thanks @QuangTran4810					3 years ago
chapter(	05	Update the axis	Update the axis limit			2 years ago
chapter	06	Update random	_walk.py			16 months ago
chapter	07	Pythonic edits				3 years ago
chapter	08	Merge pull requ	est #138 from vinn	ik-dmitry07/patch-1		2 years ago

https://github.com/ShangtongZhang/reinforcement-learning-an-introduction

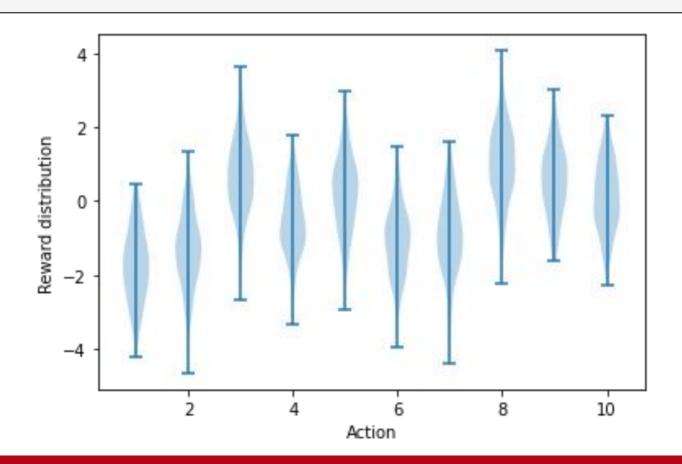




class Bandit:

# @k\_arm: # of arms

#### 10 arms in our simulations!





class Bandit:

# @k arm: # of arms

- Allows to simulate interaction with the environment
- Implements:

$$A_t = \begin{cases} \text{best action with } P = 1 - \epsilon \\ \text{uniform random } P = \epsilon \end{cases}$$

$$A_{t} = argmax_{a} \left[ Q_{t}(a) + c\sqrt{\frac{\ln t}{N_{t}(a)}} \right]$$

Gradient bandit 
$$Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}}$$



```
class Bandit:
```

# @k\_arm: # of arms

# @epsilon: probability for exploration in epsilon-greedy algorithm

$$A_t = \begin{cases} \text{best action with } P = 1 - \epsilon \\ \text{uniform random } P = \epsilon \end{cases}$$



```
class Bandit:
# @k_arm: # of arms
# @epsilon: probability for exploration in epsilon-greedy algorithm
# @initial: initial estimation for each action
```

$$Q_{t_0}(a) \ \forall a \in A$$



```
class Bandit:
```

```
# @k_arm: # of arms
```

# @epsilon: probability for exploration in epsilon-greedy algorithm

# @initial: initial estimation for each action

# @step size: constant step size for updating estimations

$$\begin{cases} H_{t+1}(A_t) \leftarrow H_t(A_t) + \alpha \left( R_t - \bar{R}_t \right) (1 - \pi_t(A_t)) \\ H_{t+1}(a) \leftarrow H_t(a) - \alpha \left( R_t - \bar{R}_t \right) \pi_t(a) & \text{for all } a \neq A_t \end{cases}$$

$$Q_{n+1} \leftarrow Q_n + \alpha \left( R_n - Q_n \right)$$



```
class Bandit:
# @k_arm: # of arms
# @epsilon: probability for exploration in epsilon-greedy algorithm
# @initial: initial estimation for each action
# @step_size: constant step size for updating estimations
# @sample_averages: if True, use sample averages to update estimations
```

$$Q_{n+1} \leftarrow Q_n + \alpha \left( R_n - Q_n \right)$$



```
class Bandit:
# @k arm: # of arms
# @epsilon: probability for exploration in epsilon-greedy algorithm
# @initial: initial estimation for each action
# @step size: constant step size for updating estimations
# @sample_averages: if True, use sample averages to update estimations
# @UCB_param: if not None, use UCB algorithm to select action
                                                                 Algorithm
                                                                 Selection
# @gradient: if True, use gradient based bandit algorithm
```

 $argmax_a[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}]$   $Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}}$ 



```
\begin{cases} H_{t+1}(A_t) \leftarrow H_t(A_t) + \alpha \left( R_t - \bar{R}_t \right) \left( 1 - \pi_t(A_t) \right) \\ H_{t+1}(a) \leftarrow H_t(a) - \alpha \left( R_t - \bar{R}_t \right) \pi_t(a) & \text{for all } a \neq A_t \end{cases}
class Bandit:
# @k arm: # of arms
# @epsilon: probability for exploration in epsilon-greedy algorithm
# @initial: initial estimation for each action
# @step_size: constant step size for updating estimations
# @sample averages: if True, use sample averages to update estimations
# @UCB_param: if not None, use UCB algorithm to select action
# @gradient: if True, use gradient based bandit algorithm
# @gradient baseline: if True, use average reward baseline for gradient bandit
```



```
def simulate(runs, time, bandits):
    # @runs: number of times each badit is simulated
    # @time: interaction time (number of steps) for each run
    # @bandits: list of Bandit objects
```



```
def simulate(runs, time, bandits):
    # @runs: number of times each badit is simulated
    # @time: interaction time (number of steps) for each run
    # @bandits: list of Bandit objects

    rewards = np.zeros((len(bandits), runs, time))
    best_action_counts = np.zeros(rewards.shape)

    runs
    time
```



```
def simulate(runs, time, bandits):
    # @runs: number of times each badit is simulated
    # @time: interaction time (number of steps) for each run
    # @bandits: list of Bandit objects

rewards = np.zeros((len(bandits), runs, time))
    best_action_counts = np.zeros(rewards.shape)

for i, bandit in enumerate(bandits):
    for r in range(runs):
    bandit.reset() # start from initial state
```



```
def simulate(runs, time, bandits):
# @runs: number of times each badit is simulated
# @time: interaction time (number of steps) for each run
                                                           Rewards
# @bandits: list of Bandit objects
                                                                                     runs
   rewards = np.zeros((len(bandits), runs, time))
   best_action_counts = np.zeros(rewards.shape)
                                                    bandits
   for i, bandit in enumerate(bandits):
       for r in range(runs):
                                                                     time
           bandit.reset() # start from initial state
           for t in range(time):
                                       # decide the action
               action = bandit.act()
               reward = bandit.step(action) # get reward
               rewards[i, r, t] = reward
                                         # save reward
```



```
def simulate(runs, time, bandits):
# @runs: number of times each badit is simulated
# @time: interaction time (number of steps) for each run
                                                          Rewards
# @bandits: list of Bandit objects
                                                                                     runs
   rewards = np.zeros((len(bandits), runs, time))
   best action counts = np.zeros(rewards.shape)
                                                    bandits
   for i, bandit in enumerate(bandits):
       for r in range(runs):
                                                                     time
           bandit.reset() # start from initial state
           for t in range(time):
                                       # decide the action
               action = bandit.act()
               reward = bandit.step(action) # get reward
               rewards[i, r, t] = reward  # save reward
               if action == bandit.best action: # Count best action
                   best action counts[i, r, t] = 1  # for stats
```



```
def simulate(runs, time, bandits):
# @runs: number of times each badit is simulated
# @time: interaction time (number of steps) for each run
                                                           Rewards
# @bandits: list of Bandit objects
                                                                                     runs
   rewards = np.zeros((len(bandits), runs, time))
   best action counts = np.zeros(rewards.shape)
                                                    bandits
   for i, bandit in enumerate(bandits):
       for r in range(runs):
                                                                     time
           bandit.reset() # start from initial state
           for t in range(time):
                                       # decide the action
               action = bandit.act()
               reward = bandit.step(action) # get reward
               rewards[i, r, t] = reward  # save reward
               if action == bandit.best action: # Count best action
                   best_action_counts[i, r, t] = 1  # for stats
   # Averages over runs
   mean best action counts = best action counts.mean(axis=1)
   mean rewards = rewards.mean(axis=1)
   return mean_best_action_counts, mean_rewards
```