Master's degree in Control System Engineering

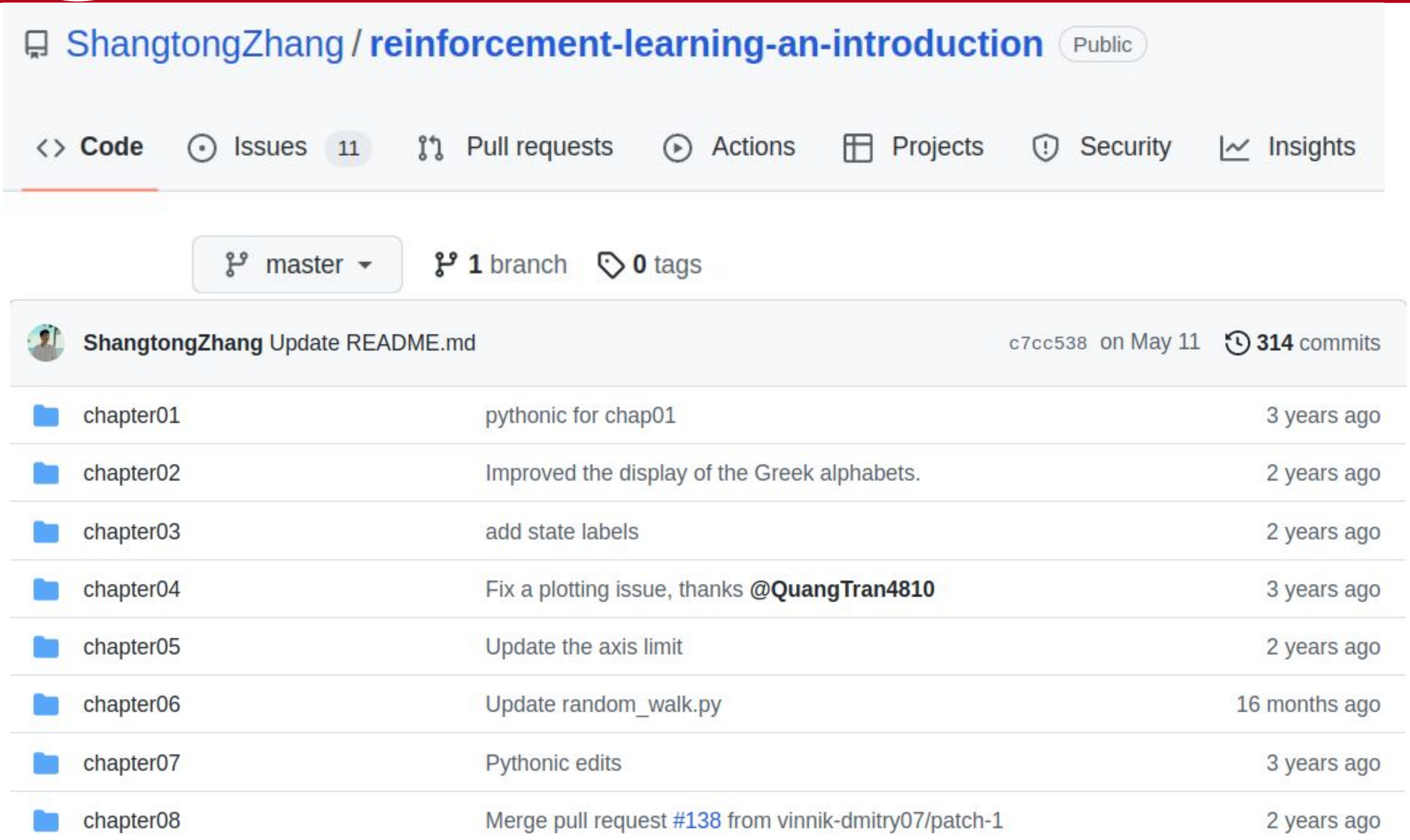
Reinforcement Learning
LAB 1

k-armed Bandits

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Niccolò Turcato
niccolo.turcato@phd.unipd.it

Github repo

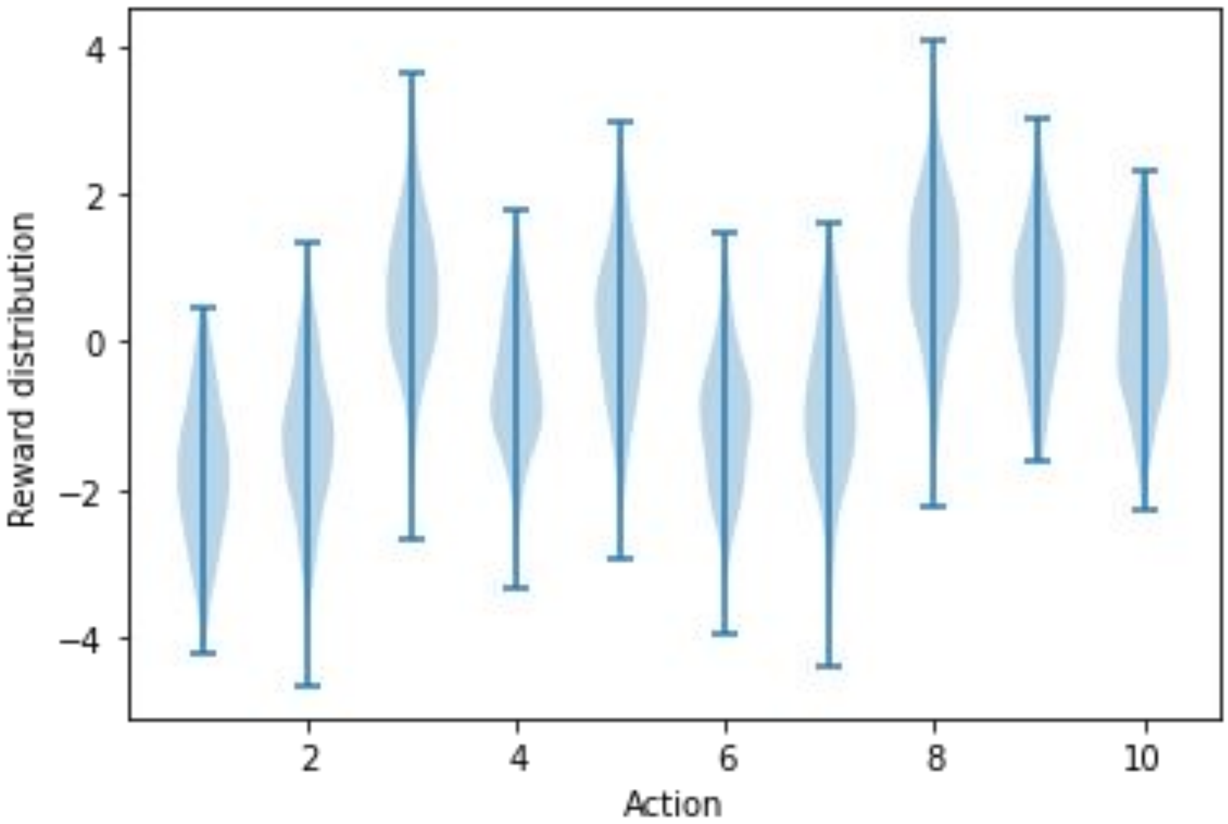


https://github.com/ShangtongZhang/reinforcement-learning-an-introduction

Bandit Class

```
class Bandit:

# @k_arm: # of arms
```

10 arms in our simulations!



Bandit Class

```
class Bandit:

# @k_arm: # of arms
```

- Allows to simulate interaction with the environment
- Implements:
    - ε-greedy policy $$A_t = \begin{cases} \text{best action with } P = 1 - \epsilon \\ \text{uniform random } P = \epsilon \end{cases}$$
    - UCB $$A_t = argmax_a \left[Q_t(a) + c\sqrt{\frac{ln\, t}{N_t(a))}} \right]$$
    - Gradient bandit $$Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}}$$

Bandit Class

```
class Bandit:

# @k_arm: # of arms

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

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

Bandit Class

```
class Bandit:

# @k_arm: # of arms

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

# @initial: initial estimation for each action
```

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

Bandit Class

```
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
```

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

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

Bandit Class

```python
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
```

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

Bandit Class

```
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
```

Algorithm Selection

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

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

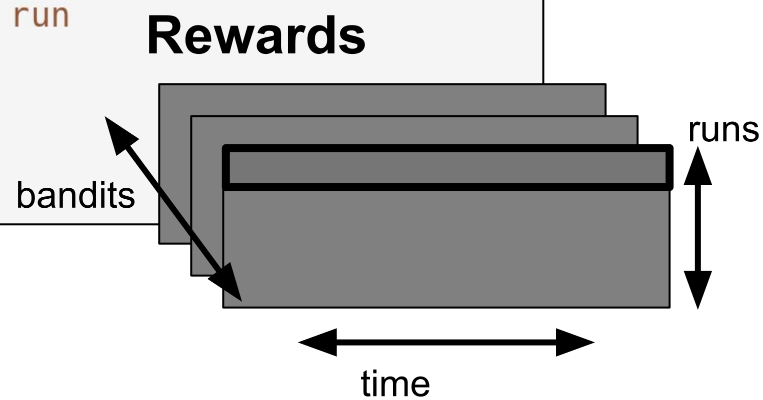
$$\begin{cases} H_{t+1}(A_t) \leftarrow H_t(A_t) + \alpha\left(R_t - \boxed{\bar{R}_t}\right)\left(1 - \pi_t(A_t)\right) \\ H_{t+1}(a) \leftarrow H_t(a) - \alpha\left(R_t - \boxed{\bar{R}_t}\right)\pi_t(a) \quad \text{for all } a \neq A_t \end{cases}$$

Simulation of Bandits

```python
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
```

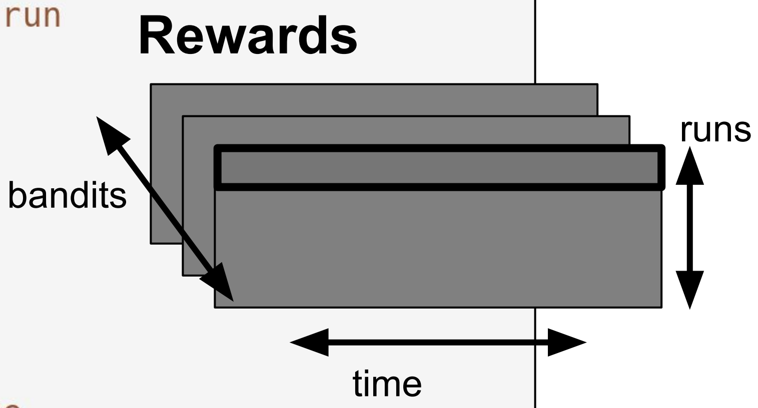
Simulation of Bandits

```
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)
```



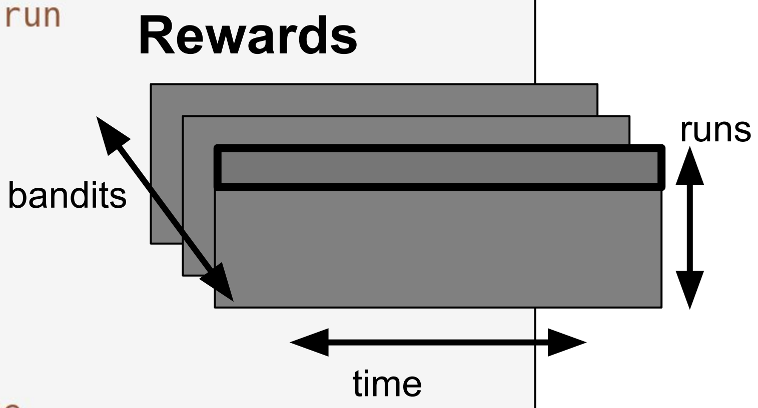
Simulation of Bandits

```
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
```



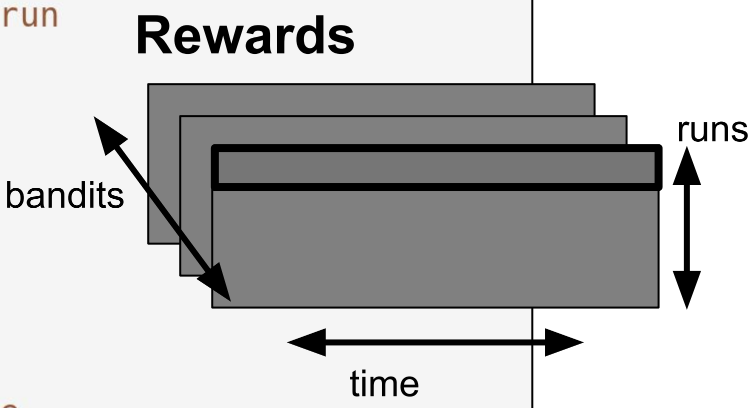
Simulation of Bandits

```
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

            for t in range(time):

                action = bandit.act()           # decide the action
                reward = bandit.step(action)    # get reward
                rewards[i, r, t] = reward       # save reward
```



Simulation of Bandits

```
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

            for t in range(time):

                action = bandit.act()            # decide the action
                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
```



Simulation of Bandits

```
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

            for t in range(time):

                action = bandit.act()              # decide the action
                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
```