Introduction to Reinforcement Learning
Session 2

Jean Martinet, based on the course of Diane Lingrand

Polytech SI4 / EIT Digital DSC

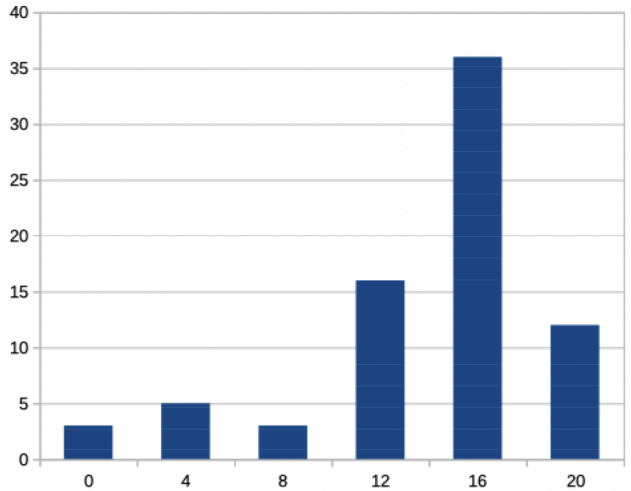
2020 - 2021

Today's menu

- Feedback after first submission
- About optimal policy
- Back to time difference
  - Q-learning and SARSA

Feedback after first submission

- Went globally well
- Grades range from 0/20 to 20/20



- Was meant to be done individually, not in groups
  - But was not clear enough, so okay (more strict grading though)
- BEWARE ! Strictly avoid plagiarism
  - = copy something without citing the source [www.plagiarism.org]
- More technical
  - If you use ipynb, always re-run the whole notebook before submitting
  - Do not put the move cost in the reward model
  - Episodes should always start with same initial state
  - Episodes should normally terminate in terminal states
  - 2D got you a bonus only (optional)

Mini-quizz (lecture)

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}[G_t|s_t] = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t]$$

- What does the value $V^{\pi}(s_t)$ tell us ?
  -
- How many policies are there ?
  -
- Is there a best policy ?
  -
- Is it unique ?
  -
  -

Mini-quizz (lecture)

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}[G_t|s_t] = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t]$$

- What does the value $V^{\pi}(s_t)$ tell us ?
  - Expected disc. sum of rewards when starting from s_t and following π
- How many policies are there ?
  - There are $|A|^{|S|}$ possible policies
- Is there a best policy ?
  - Yes, it is $\pi^* = argmax_{\pi} V^{\pi}(s)$
- Is it unique ?
  - The value V^{π} is unique, but the optimal policy is not necessarily unique
  - (there can be several actions or policies yielding same value)

Mini-quizz (lab)

- What happens when *alpha* varies ?
  -
- What happens when $\gamma = 0$? $\gamma = 1$?
  -
- What happens when $\epsilon = 0$? $\epsilon = 1$?
  -

Mini-quizz (lab)

- What happens when *alpha* varies ?
  - Updates to the Q tables are slower / faster
- What happens when $\gamma = 0$? $\gamma = 1$?
  - $\gamma = 0$: only cares about immediate reward, $\gamma = 1$: can see farther
- What happens when $\epsilon = 0$? $\epsilon = 1$?
  - $\epsilon = 1$: exploration, $\epsilon = 0$: exploitation

Back to state-value function $V^{\pi}(s)$

- We define the Bellman equation for V^{π} as follows :

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

  - (averages all possibilities, weighted by probability of occurrence)
- **Bellman optimality equations for V^{π^*} and Q^{π^*}**

$$V^{\pi^*}(s_t) = \max_a \sum_{s',r} p(s', r|s_t, a)[r + \gamma V^{\pi^*}(s')]$$

$$Q^{\pi^*}(s_t, at) = \sum_{s',r} p(s', r|s_t, a_t)[r + \gamma \max_{a'} Q^{\pi^*}(s', a')]$$

MDP control and optimal policy π^*

- The optimal policy for a MDP in an infinite horizon is :
  - deterministic (always take the same action in a given state),
  - stationary (does not change with timestep),
  - not necessarily unique
- How to determine the best policy ?
  - Enumeration : evaluate each policy separately (finite), and pick the best
  - Policy iteration (computes optimal policy and value)
  - Value iteration – what we did last week
- Remember : Temporal Difference error δ_t is the difference between
  - the *estimated* value of s_t and
  - the *better estimate* $r_{t+1} + \gamma V^{\pi}(s_{t+1})$

$$\delta_t = r_{t+1} + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

  - Q-learning is a TD algorithm, so is SARSA

SARSA

- SARSA = State Action Reward State Action
- similar to Q-learning
- on-policy algorithm (same algorithm for choosing action and updates)

```
def learn(state, state2, reward, action):#Qlearning
    predict = Q[state, action]
    target = reward + gamma * np.max(Q[state2, :])
    Q[state, action] = Q[state, action] + lr_rate * (target - predict)

def learn(state, state2, reward, action, action2):#SARSA
    predict = Q[state, action]
    target = reward + gamma * Q[state2, action2]
    Q[state, action] = Q[state, action] + lr_rate * (target - predict)
```

SARSA versus Qlearning

```
#Qlearning
for episode in range(total_episodes):
    state = env.reset()
    t = 0
    while t < max_steps:
        env.render()
        action = choose_action(state)
        state2, reward, done, info = env.step(action)
        learn(state, state2, reward, action)
        state = state2
#SARSA
for episode in range(total_episodes):
    t = 0
    state = env.reset()
    action = choose_action(state)
    while t < max_steps:
       env.render()
       state2, reward, done, info = env.step(action)
       action2 = choose_action(state2)
       learn(state, state2, reward, action, action2)
       state = state2
       action = action2
```

Today's lab

SFFF (S : starting point, safe)
FHFH (F : frozen surface, safe)
FFFH (H : hole, fall to your doom)
HFFG (G : goal, where the frisbee is located)

- 1. Use your code and play with the parameters α, γ, ϵ
  - Plot the time needed for convergence, when varying parameters
- 2. Try gym with the FrozenLake
  - 16 states, 4 actions, a pinch of stochasticity :-)
  - score of $+1$ if the agent achieves the goal (end of game)
  - score of 0 if the agent falls into a hole (end of game)
  - have a try using code from Adesh Gautam
  - if you want to play using the Q-table learned :

```
with open("frozenLake_qTable.pkl", 'rb') as f:
    Q = pickle.load(f)

def choose_action(state):
    action = np.argmax(Q[state, :])
    return action
```