Introduction to intelligent systems

Reinforcement learning

Mikkel N. Schmidt

Technical University of Denmark,
DTU Compute, Department of Applied Mathematics and Computer Science.

Overview

1. Reinforcement learning

2. Python dictionaries

3. Tasks

Feedback group

- Karl Johan Murphy Mogensen
- Rasmus Grønnegaard Arnmark
- Mikel Taotao Yu
- Haaris Usman Syed

Learning objectives

- I Reinforcement learning: Markov decision process (state, action, reward).
- I Epsilon-greedy action selection and optimistic initialization.
- II RL algorithms: Value iteration and Q-learning.
- II Optimal action and optimal policy.
- II Discount factor.
- II Value (of a state) and quality (of a state-action pair).

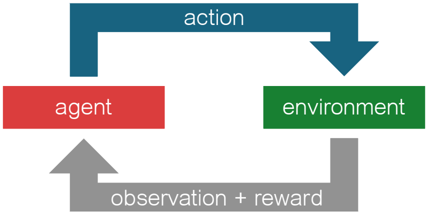
I Understand the concepts and definitions, and know their application. Reason about the concepts in the context of an example. Use correct technical terminology.

II As above plus: Read, manipulate, and work with technical definitions and expressions (mathematical and Python code). Carry out practical computations. Interpret and evaluate results.

Reinforcement learning

Reinforcement learning

Learn a function (policy) that maps inputs to actions to optimize cumulative reward



Markov decision process (MDP)

In the general setting where state transitions and rewards are stochastic, the MDP is defined by

$\mathcal{S}$	Set of states.
$\mathcal{A}$	Set of actions.
$p(s'\|s, a)$	Probability of next state s' given current state s and action a.
$p(r\|s', s, a)$	Distribution of reward for transitioning to state s' from state s using action a.

Markov decision process (deterministic setting)

In the deterministic setting, the MDP is defined by

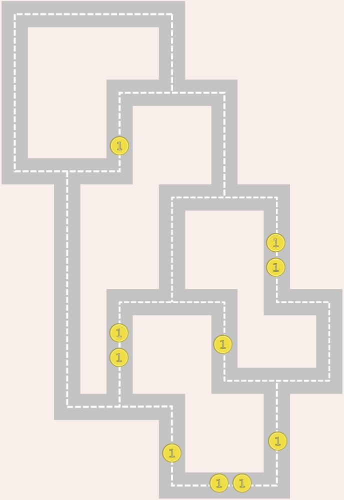
$\mathcal{S}$	Set of states.
$\mathcal{A}$	Set of actions.
$s' = f(s, a)$	Next state s' is determined from current state s and action a.
$r = r(s, a)$	Reward for taking action a in state s.

- The next state is deterministic, i.e. given as a function of the current state and action.
- Rewards depend only on current state and action.

Exercise: Collecting gold coins

Consider a game, where we drive around and collect gold coins (coins can be picked up multiple times.)
How could we meaningfully define:

- The set of states
- The set of actions
- The next-state function
- The reward

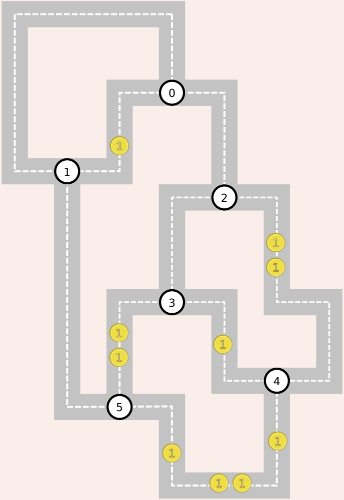


Discussion: Collecting gold coins

- The set of states could be defined as places where the road forks.
- The set of actions could be defined as north, south, east west. We need to consider what would happen if we take an "illegal" action.
- The next state is deterministic: Follow the road to the next fork.
- The reward could be the number of coins collected.

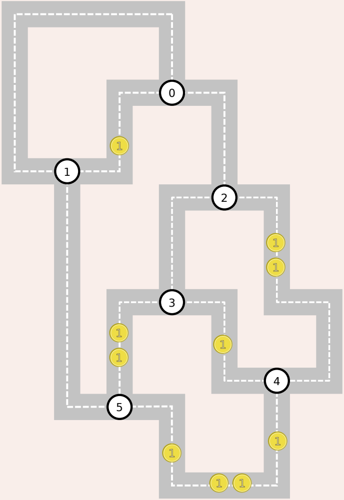
Exercise: Optimal policy

What is the *optimal policy*?
Hint: What should we end up doing, if we follow the optimal policy?



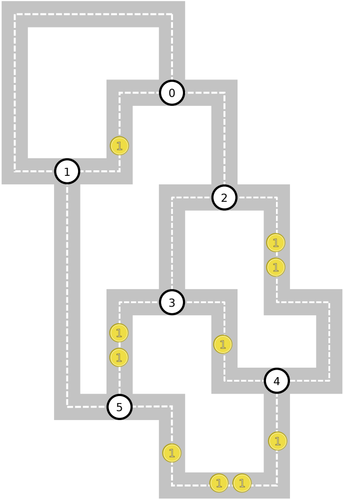
Solution: Optimal policy

- The optimal policy should end up going back and forth between state 4 and 5, collecting 4 gold coins in each step.
- From state 3, the optimal policy would be to go to state 5, since this gives us 2 gold coins.
- From state 2 go to state 4.
- From state 1 go to state 5.
- From state 0 go to state 1 or 2. This depends on whether we prioritize getting 1 coin immediately or 2 coins a bit later.



Reward

Reward $r(s, a)$		Action, a N	S	E	W
State, s	0	0	0	0	1
	1	0	0	1	0
	2	0	0	2	0
	3	0	0	1	2
	4	0	4	2	1
	5	2	0	4	0



Value function

In the deterministic setting, the value function is defined recursively as

Value function (deterministic setting)

$$v(s) = \max_a \big(r(s, a) + \gamma v(s')\big)$$

Value of state 5

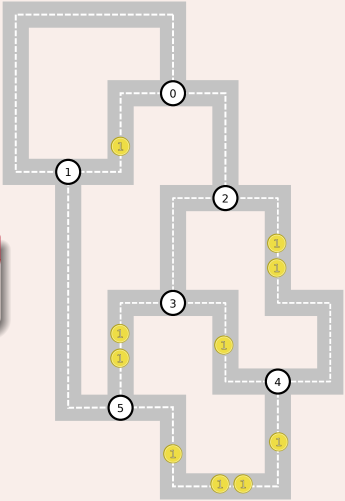
- The optimal policy will end up going back and forth between state 4 and 5.
- We will use $\gamma = 0.9$

What is the value of state 5?

Hint: We can deduce that $v(4) = v(5)$ from the optimal policy.

Value function (deterministic setting)

$$v(s) = \max_a \left(r(s, a) + \gamma v(s') \right)$$



Value of state 5

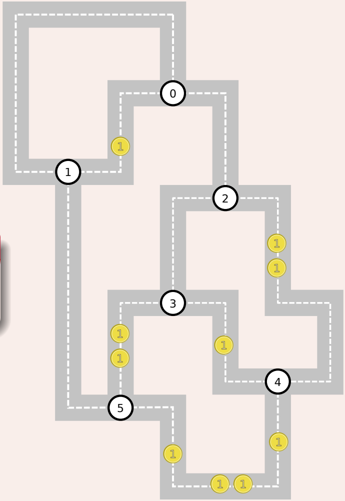
- The optimal policy will end up going back and forth between state 4 and 5.
- We will use $\gamma = 0.9$

What is the value of state 5?

Hint: We can deduce that $v(4) = v(5)$ from the optimal policy.

Value function (deterministic setting)

$$v(s) = \max_a \left(r(s, a) + \gamma v(s')\right)$$



Solution

$$v(5) = r(5, \text{East}) + \gamma v(4) = 4 + \gamma v(5)$$
$$= \frac{4}{1-\gamma} = \underline{40}$$

Value iteration

- Loop through all states and update according to

$$v(s) = \max_a \left(r(s, a) + \gamma v(s') \right)$$

- Repeat until convergence

Value iteration

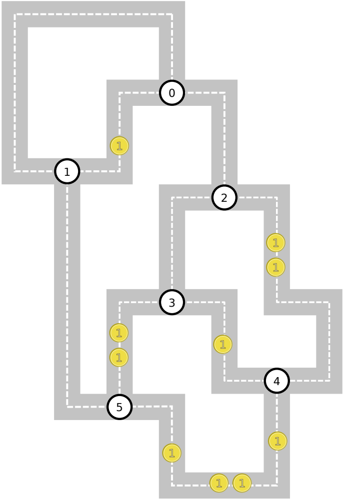
```python
# Initial values
V = [0,0,0,0,0,0]
# Discount
gamma = 0.9
# Actions: 0=North,
# 1=South, 2=East, 3=West
actions = [0, 1, 2, 3]

# Next state table
F =  [[1, 0, 2, 1],
      [1, 5, 0, 0],
      [0, 2, 4, 3],
      [2, 3, 4, 5],
      [4, 5, 2, 3],
      [3, 5, 4, 1]]

# Reward table
R = [[0, 0, 0, 1],
      [0, 0, 1, 0],
      [0, 0, 2, 0],
      [0, 0, 1, 2],
      [0, 4, 2, 1],
      [2, 0, 4, 0]]
```

```python
# 1000 value iterations
for t in range(1000):
  # Loop over all states
  for s in range(6):
    # Update value
    V[s] = max([r+gamma*V[sp] for r,sp in zip(R[s], F[s])])
```

Estimated value function

After running the code, we arrive at the following value function

State, s	0	1	2	3	4	5
Value, $v(s)$	34.2	36	38	38	40	40



Model-based and model-free

Model based
: We know the state transition function.
Example: Value iteration

Model free
: We can only learn about state transitions by interacting with the environment
Example: Q-learning

Quality function

In the deterministic setting, the quality function is defined recursively as

Quality function (deterministic setting)

$$q(s, a) = r(s, a) + \gamma \max_{a'} q(s', a')$$

The quality of taking action a in state s is

- The immediate associated reward +
- The discounted quality of the best action in the next state.

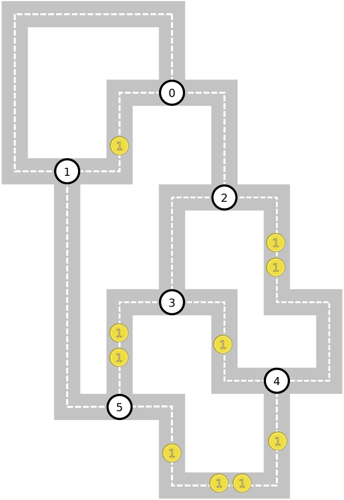
Q-learning

- Explore the environment according to some policy that ensures visiting all state-action pair
- At each step, update the quality function according to

$$q(s, a) = r(s, a) + \gamma \max_{a'} q(s', a')$$

Q-learning example

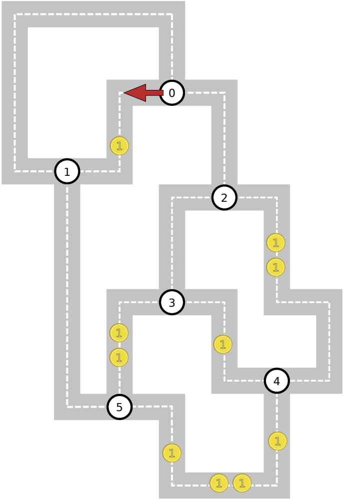
Quality $q(s, a)$		Action, a N	S	E	W
State, s	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0



Q-learning example

Quality $q(s, a)$		Action, a N	S	E	W
State, s	0	0	0	0	***1***
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0

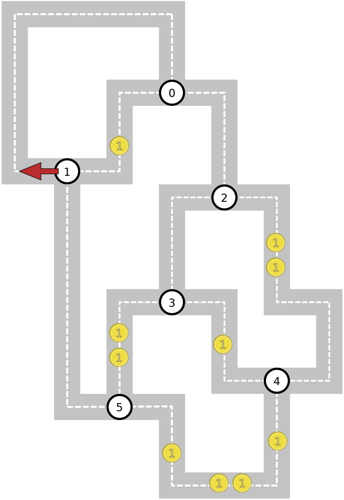
$$q(0, \mathrm{W}) = r(0, \mathrm{W}) + \gamma \max_{a'} q(1, a')$$
$$= 1 + 0 = 1$$



Q-learning example

Quality $q(s, a)$		Action, a N	S	E	W
State, s	0	0	0	0	1
	1	0	0	0	***0.9***
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0

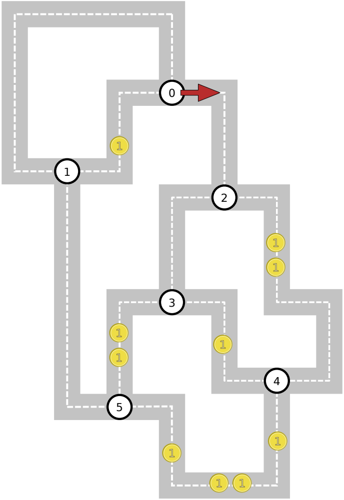
$$
\begin{aligned}
q(1, \mathrm{W}) &= r(1, \mathrm{W}) + \gamma \max_{a'} q(0, a') \\
&= 0 + 0.9 \cdot 1 = 0.9
\end{aligned}
$$



Q-learning example

Quality $q(s, a)$		Action, a N	S	E	W
State, s	0	0	0	***0***	1
	1	0	0	0	0.9
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0

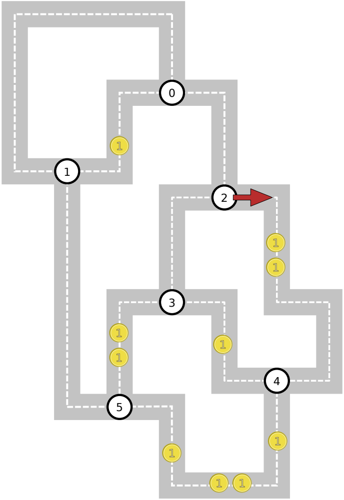
$$q(0, \mathrm{E}) = r(0, \mathrm{E}) + \gamma \max_{a'} q(2, a')$$
$$= 0 + 0.9 \cdot 0 = 0$$



Q-learning example

Quality $q(s, a)$		Action, a N	S	E	W
State, s	0	0	0	0	1
	1	0	0	0	0.9
	2	0	0	2	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	0	0	0

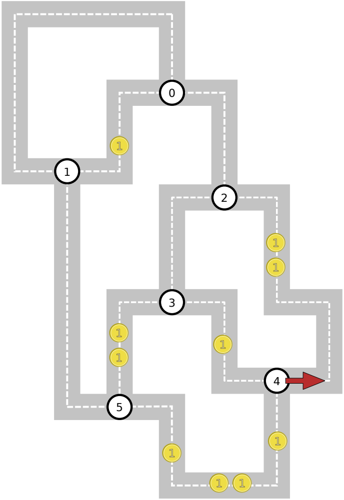
$$
\begin{aligned}
q(2, \mathrm{E}) &= r(2, \mathrm{E}) + \gamma \max_{a'} q(4, a') \\
&= 2 + 0.9 \cdot 0 = 2
\end{aligned}
$$



Q-learning example

Quality $q(s, a)$		N	S	E	W
		Action, a			
State, s	0	0	0	0	1
	1	0	0	0	0.9
	2	0	0	2	0
	3	0	0	0	0
	4	0	0	***3.8***	0
	5	0	0	0	0

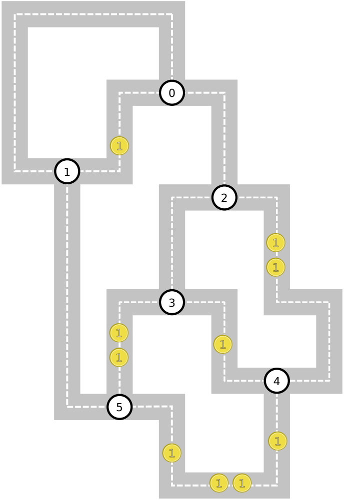
$$q(4, \mathrm{E}) = r(4, \mathrm{E}) + \gamma \max_{a'} q(2, a')$$
$$= 2 + 0.9 \cdot 2 = 3.8$$



Q-learning example

Final q-table

Quality $q(s, a)$		N	S	E	W
State, s	0	32.4	30.8	*34.2*	33.4
	1	32.4	*36.0*	31.8	30.8
	2	30.8	34.2	*38.0*	34.2
	3	34.2	34.2	37.0	*38.0*
	4	36.0	*40.0*	36.2	35.2
	5	36.2	36.0	*40.0*	32.4



Epsilon-greedy exploration

The epsilon-greedy policy is one way to explore the environment, that mixes *exploration* and *exploitation*.
With probability

- ϵ Take a random action.
- $1-\epsilon$ Take the best action according to the current estimate of the quality function.

The best action is simply given as

$$a^* = \max_a q(s, a)$$

Exploration by optmistic initialization

Another way to ensure exploration is to use *optimistic initialization*.
Here, we always take the best action according to the current estimate of the quality function.

- The quality of all state-action pairs are initialized with a (relatively) high value.
- When the agent receives its reward, it will be lower than the initial values.
- The agent then avoids actions that lead to this low reward.
- After a while, all actions have been explored, and the quality function converges.

Python dictionaries

Dictionaries

A Python dictionary is a data structure that associates *keys* with *values*.

```
>>> my_dict = {'a':[0,1,2], 'b':[3,4,5]}
>>> my_dict
{'a': [0, 1, 2], 'b': [3, 4, 5]}

>>> my_dict['a']
[0, 1, 2]

>>> my_dict['b'][2]
5

>>> my_dict['b'][2] = 10
>>> my_dict
{'a': [0, 1, 2], 'b': [3, 4, 10]}

>>> my_dict['c']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'c'
```

Default-dictionaries

In a default-dictionary, we have a function that specifies the default value for undefined keys.

```
>>> from collections import defaultdict
>>> my_defaultdict = defaultdict(lambda: [0, 0, 0])
>>> my_defaultdict
defaultdict(<function <lambda> at 0x7f6836046200>, {})

>>> my_defaultdict['a'] = [1,2,3]
>>> my_defaultdict['a']
[1, 2, 3]

>>> my_defaultdict['c']
[0, 0, 0]
```

Tasks

Tasks for today

1. Today's feedback group
   - Karl Johan Murphy Mogensen
   - Rasmus Grønnegaard Arnmark
   - Mikel Taotao Yu
   - Haaris Usman Syed

Lab report

- Lab 5: Reinforcement learning (Deadline: Thursday 23 November 20:00)