WORKSHEET

8.5 Learning by reward

Learning · Robot club · about 30 min

Name _____ Class _____ Date _____

What this lesson is about

Q-learning: a table of how good each action is, filled in by trying.

- States and actions
- The table
- The reward, and the rule
- What it learns

Questions

8 marks in all

1. What makes learning by reward different from the classifiers earlier in the module? [1 mark]

- ☐ **A** There are no labels: the robot tries actions and is told afterwards how well they went
- ☐ **B** It needs more labelled samples
- ☐ **C** It uses the camera instead of the depth grid
- ☐ **D** It never makes mistakes while learning

2. The states are near (under 20 cm), mid (under 40 cm) and far, and the actions are forward and turn. How many numbers does the Q table hold? [1 mark]

3. What does this program print? [1 mark]

```
def state(ahead):
    return "near" if ahead < 20 else ("mid" if ahead < 40 else "far")

print(state(12))
print(state(20))
print(state(40))
```

4. What does this program print? [1 mark]

```
Q = {"near": {"forward": 0.0, "left": 0.0}, "far": {"forward": 2.0, "left": 0.5}}
s, a, s2 = "near", "forward", "far"
reward = 1.0
Q[s][a] += 0.3 * (reward + 0.8 * max(Q[s2].values()) - Q[s][a])
print(round(Q[s][a], 2))
```

5. Why does the robot sometimes pick an action at chance instead of the best one it knows? [1 mark]

- ☐ **A** Those tries are how it discovers anything it does not already know
- ☐ **B** To save battery
- ☐ **C** Because the table is full
- ☐ **D** So it bumps into things on purpose

6. What is `was_bumped` for in the reward loop? [1 mark]

- ☐ **A** `bumped()` stays true for a third of a second, so it makes one bump cost -20 only once
- ☐ **B** It counts how many times the robot has turned
- ☐ **C** It stops the program after the first bump
- ☐ **D** It remembers which side the bump was on

7. After learning, what does the table usually say in the `near` state? [1 mark]

- ☐ **A** Turn is worth more than forward
- ☐ **B** Forward is worth more than turn
- ☐ **C** Both actions are worth zero
- ☐ **D** Both actions are strongly negative

8. Why does a turn keep going the same way until the robot drives forward again? [1 mark]

- ☐ **A** A robot that picks left or right afresh every step can swap between them in a corner for ever
- ☐ **B** Turning left and right costs different amounts
- ☐ **C** The depth grid can only see one side at a time
- ☐ **D** It saves battery

The task: learn to wander

Let the robot learn for 88 seconds. It may bump early on. After 60 seconds it must not bump at all, and it must keep getting about the mat: rocking back and forth or spinning in one spot does not count. It must drive at least 250 cm in total. Print `bumps: <n>` at the end.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()
bumps = 0
# do this 880 times (tick counts from 0)
for tick in range(880):
    # drive forward at 60 (keeps going until the next command)
    forward(60)
    # pause 0.1 s (the robot keeps doing what it was told)
    wait(0.1)
    if bumped():
        bumps += 1
        # spin clockwise on the spot at 60
        turn_right(60)
        # pause 0.5 s (the robot keeps doing what it was told)
        wait(0.5)
# all motors off
stop()
print('bumps:', bumps)
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/8-5-learning-by-reward/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Keep epsilon at 0.3 for the whole run. How many bumps now?
2. Give it three actions, `forward`, `left` and `right`, each choosing afresh. Watch what it does the first time it meets a corner.
3. Make a bump cost 0 instead of -20. What does the table say about `forward` in `near` now, and where does the robot end up?