CS4618: Artificial Intelligence I

Reinforcement Learning

Derek Bridge
School of Computer Science and Information Technology
University College Cork

Initialization

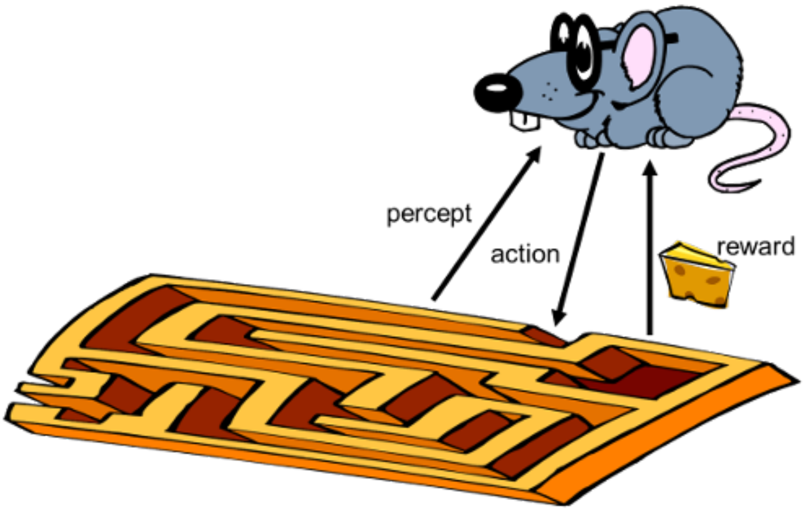
In [1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Reinforcement learning

- The agent carries out an action
- A teacher or the environment provides a **reward** (or punishment), often delayed, that acts as positive (or negative) reinforcement
  - making it more (or less) likely that the agent will execute that action if it find itself in the same or similar situation in the future
- For simplicity, in this lecture, we assume a fully-observable, deterministic environment

Reward



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \cdots$$

Cumulative reward

- Cumulative reward:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

  or

$$\sum_{t=0}^{t=\infty} \gamma^t r_t$$

  where γ is the **discount rate** $(0 \leq \gamma \leq 1)$
- The task of the agent is to learn an action function that maximises cumulative reward

Action-value function

- Assume 2 Boolean sensors and 3 actions
- Compare

Percept	Action
00	MOVE
01	TURN(RIGHT, 2)
10	MOVE
11	TURN(LEFT, 2)

Percept	Action	Q
00	MOVE	…
00	TURN(RIGHT, 2)	…
00	TURN(LEFT, 2)	…
01	MOVE	…
01	TURN(RIGHT, 2)	…
01	TURN(LEFT, 2)	…
10	MOVE	…
10	TURN(RIGHT, 2)	…
10	TURN(LEFT, 2)	…
11	MOVE	…
11	TURN(RIGHT, 2)	…
11	TURN(LEFT, 2)	…

- Class exercise: Suppose the agent has m touch sensors (returning 0 or 1) and n different actions. How many rows will the table contain?

What is Q?

- $Q(\boldsymbol{s}, a)$ is an *estimate* of the cumulative reward the agent will receive if, having sensed $\boldsymbol{s}$, it chooses to execute action a
- Hence, having sensed $\boldsymbol{s}$, choose action a for which $Q(\boldsymbol{s}, a)$ is highest:

$$\arg\max_a Q(\boldsymbol{s}, a)$$

Class exercise

- Given this table

Percept	Action	Q
⋮	⋮	⋮
01	MOVE	0.2
01	TURN(RIGHT, 2)	0.1
01	TURN(LEFT, 2)	0.7
⋮	⋮	⋮

- Suppose $\boldsymbol{s}$ is 01
- What is $\arg\max_a Q(\boldsymbol{s}, a)$?

Q-learning

- Start wih random Q-values (or all zero)
- Improve by trial-and-error: choose actions, get rewards, update Q-values

QLearning(ϵ)

• $\boldsymbol{s} = SENSE()$;
• do forever
  ■ $rand =$ a randomly-generated number in $[0, 1)$;
  ■ if $rand < \epsilon$
    ○ Choose action a randomly;
  else
    ○ $a = \arg\max_a Q(\boldsymbol{s}, a)$;
  ■ $r = EXECUTE(a)$;
  ■ $\boldsymbol{s}' = SENSE()$;
  ■ $Q(\boldsymbol{s}, a) = r + \gamma \times \max_{a'} Q(\boldsymbol{s}', a')$;
  ■ $\boldsymbol{s} = \boldsymbol{s}'$;

Exploration vs. Exploitation

- Exploration:
  - Choose an action which may not be the best action according to the current Q values. But it may gain you new experience and improve the Q values
- Exploitation:
  - Choose the action which is best according to the current Q values. It may gain you reward
- The so-called ϵ**-greedy policy** (where $0 \leq \epsilon \leq 1$) is the simplest way to balance exploration and exploitation

Updating Q-values

- From the algorithm:

$$Q(\boldsymbol{s}, a) = r + \gamma \times \max_{a'} Q(\boldsymbol{s}', a')$$

- The new value is the reward for the latest action r plus our highest current estimate of the cumulative reward it can receive
- Over the course of repeated actions, the Q-values will get better and better:
  - When one Q-value improves then the Q-values of its immediate predecessors will also improve next time they get updated

Class exercise

- Given this table

Percept	**Action**	Q
$\vdots$	$\vdots$	$\vdots$
10	MOVE	5
10	TURN(RIGHT, 2)	4
10	TURN(LEFT, 2)	1
11	MOVE	0
11	TURN(RIGHT, 2)	4
11	TURN(LEFT, 2)	6

- Suppose current percept is 10
  Assuming exploitation, which action will be chosen?
- Suppose reward is 3, next percept is 11 and γ is 1
  Using $Q(\boldsymbol{s}, a) = r + \gamma \times \max_{a'} Q(\boldsymbol{s}', a')$, update the table

Concluding remarks

- Reinforcement Learning underpins a lot of current success in game playing
- But it is seeing real use in other areas, e.g. robot motion control
- For real use, you need more sophisticated algorithms:
  - To handle non-deterministic environments
  - To improve convergence
  - To build a model of the environment
  - To scale up, and
  - To represent the policy in a way that allows the agent to generalise from what it learns

```
In [ ]:
```