Reinforcement Learning

Luca Citi
lciti@essex.ac.uk

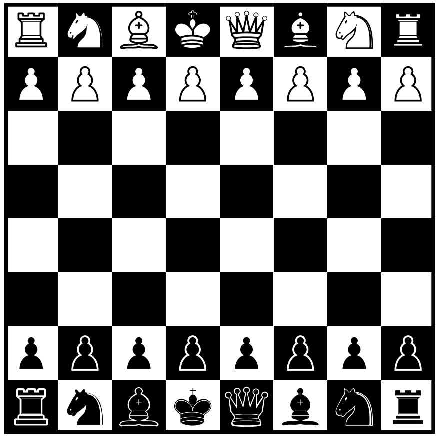
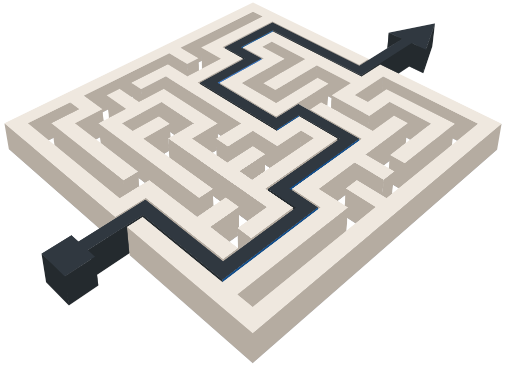
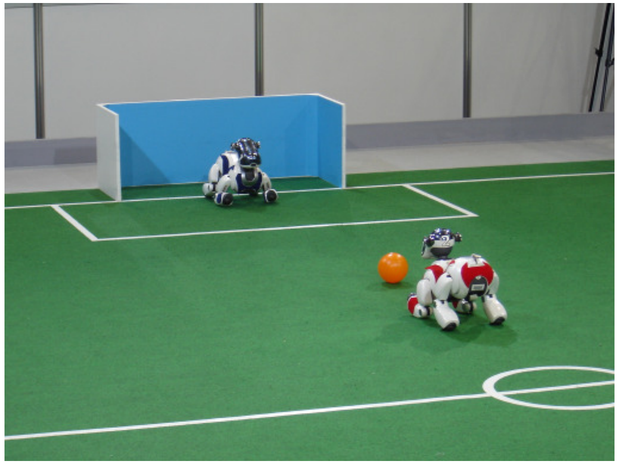
School of Computer Science and Electronic Engineering
University of Essex (UK)

CE802

Outline

Examples of Reinforcement Learning problems

- A child learning to ride a bicycle
- A rat learning to run a maze, poke its nose into a device and receive water
- A driver learning the best route between her home and her office in rush hour traffic
- A robot learning how to find the recharging unit in a laboratory
- An AI agent learning to play backgammon/chess/go/poker/any game (at human or superhuman level)

What is Reinforcement Learning?

- *Reinforcement learning is the study of how animals and artificial systems can learn to optimize their behaviour in the face of rewards and punishments* – Peter Dayan, Encyclopedia of Cognitive Science
- Not supervised learning - the animal/agent is not provided with examples of optimal behaviour, it has to be discovered!
- Not unsupervised learning either - we have more guidance than just observations
- Draws ideas from a wide range of contexts, including psychology, philosophy, neuroscience, operations research, cybernetics

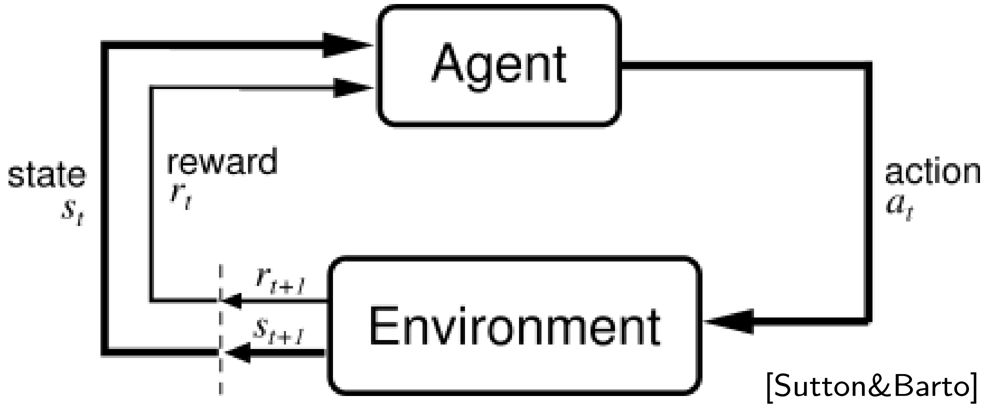
Common aspects of learning problems

- An agent is learning to choose a sequence of actions that will lead to a reward
- The ultimate consequences of an action may not be immediately apparent; when a reward is achieved, it is often not (not just) because of the last action performed (credit assignment problem)
- It is often hard to assess the effects of an action in isolation but rather one needs to consider it as part of an overall policy
- There is no pre-defined set of training examples: experiences forming the basis of learning are derived through some form of exploration
- The learning is expected to be permanent; that is, it will determine the agents behaviour for the indefinite future

Problem abstraction

Like with many complex problems, to better understand and address learning tasks we must abstract the essential features.
Most learning tasks can be modelled using the following abstraction:

- There is a single learner, called the agent
- Everything it interacts with is called the environment
- The agent and the environment interact at discrete steps t:
  - the agent receives some representation of the environment's state s_t and on that basis selects an action a_t
  - one time step later, the environment responds presenting new situations to the agent, in the form of a new state s_{t+1}, and providing a reward r_{t+1}



[Sutton&Barto]

The Markov Decision Process

It is often useful to represent a learning task mathematically as a discrete Markov Decision Process (MDP).

A discrete MDP is defined as a 4-tuple $< S, A, T, R >$ where:

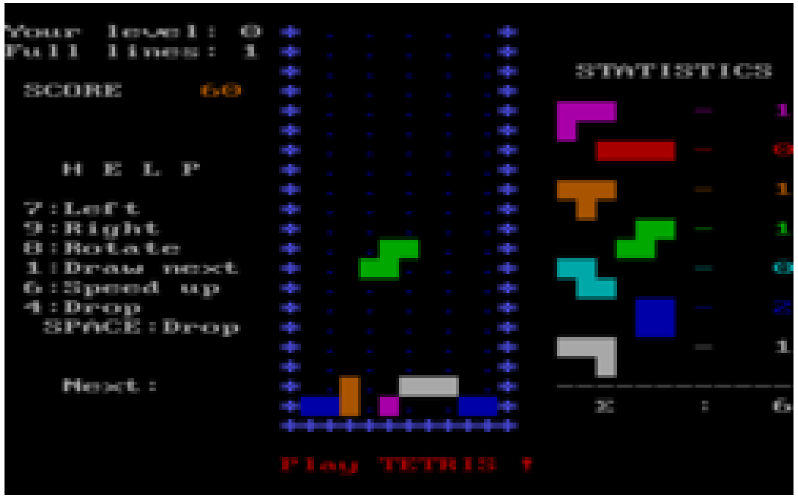
- S, $s \in S$ is a discrete set of states
- A, $a \in A$ is a discrete set of actions
- $T : S \times S \times A \to [0, 1]$ represents the state transition probability: $T(s'|s, a) \triangleq \Pr(s_{t+1}{=}s' \,|\, s_t{=}s, a_t{=}a)$ is the probability of an agent transitioning from state s to state s' after taking action a
- $R : S \times A \to \mathbb{R}$ is the expected reward obtained at the next time step in response to taking action a in state s: $R(s, a) \triangleq \mathbb{E}\left[r_{t+1} \,|\, s_t{=}s, a_t{=}a\right]^{\star}$

Important: the "Markov property" requires that the probability of the next state (T) and reward (R) only depends on the current state and action.

$^{\star}\mathbb{E}\left[\right]$ is the "Expected value" operator: the average value that one would expect after carrying out an infinite number of independent repetitions of an experiment (e.g. game).

The Markov Property

- By "state" we mean whatever information is available to the agent
- A state signal should summarize the current knowledge compactly, yet in such a way that all relevant information is retained
- In a MDP, the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories
- Examples:
  - Chess: current configuration of all the pieces summarizes everything important about the complete sequence of moves that led to it
  - Cannonball: current position and velocity is all that matters for its future flight
  - Tetris: all information captured by a single screen-shot



Policy

- The type of actions the agent takes when in a given state is called "the policy"
- While the MDP tuple represents a model of the environment, the policy describes the behaviour of the agent
- We can say that learning boils down to finding the optimal policy
- We can have a stochastic policy defined as $\Pr(a|s)$, i.e. the probability of taking a given action when in a given state
- We will only consider deterministic policies (in a possibly stochastic environment):

  $$\pi : S \to A,\ \pi(s_t) \triangleq a_t$$

  i.e. the agent will take a given action whenever in a given state
- Policies are Markov and stationary: they depend only on the current state (not on the history and not on the time: $s_t = s_{t'} \Rightarrow \pi(s_t) = \pi(s_{t'})$)

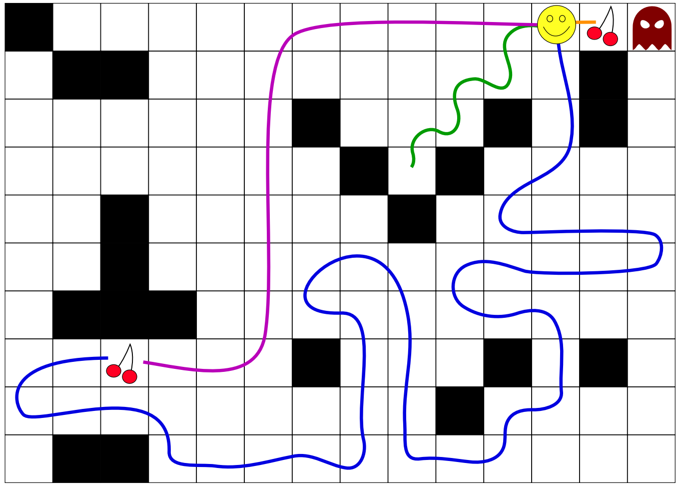
Rewards and discount factor

- Rewards allow the agent to improve its policy, i.e. to learn
- Way of telling the agent what to achieve, not how to do it (e.g., in chess reward for winning, not for taking opponent's pieces)
- This means rewards can be distant in the future: a greedy approach (choosing the action maximising r_{t+1}) would not work
- It usually makes more sense to maximise the total payoff over time
- On the other hand, rewards in the very distant future are often not as valuable as one received immediately

We define the discounted cumulative reward from time step t as:

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $\gamma \in [0, 1]$ is called the discount factor



Value function

We define the following:

- State-value function (or discounted cumulative value) $V^\pi(s)$ is the expected discounted cumulative reward starting from state s, and then following the policy π:

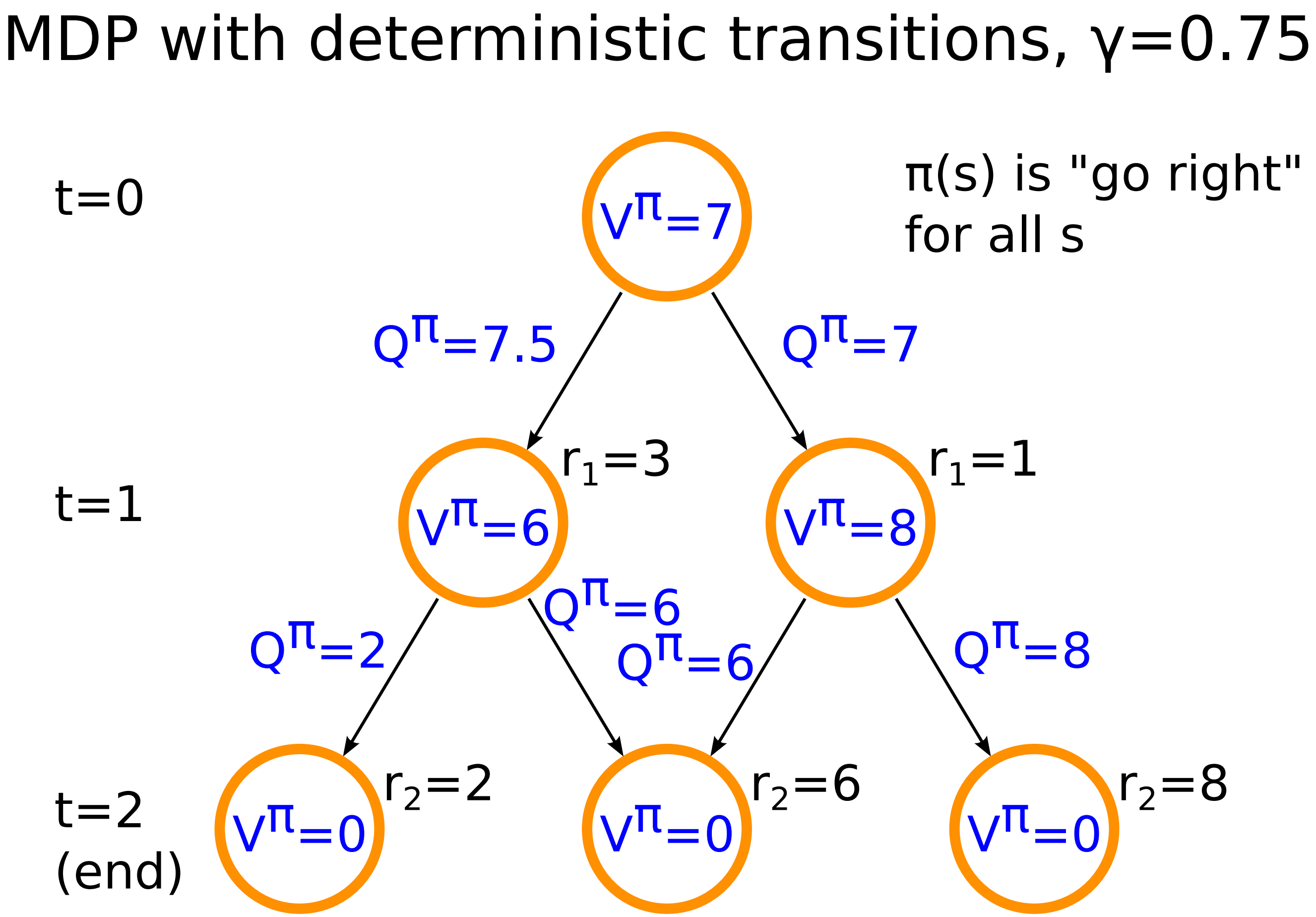
$$V^\pi(s) \triangleq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right]$$

- Action-value function (or Q-function) $Q^\pi(s, a)$ is the expected discounted cumulative reward starting from state s, taking action a, and then following the policy π afterwards:

$$Q^\pi(s, a) \triangleq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right]$$

Value function (example)

Example:



Bellman's Expectation Equation (maths for reference)

Remembering the definitions:

- $T(s'|s,a) \triangleq \Pr(s_{t+1}{=}s' \mid s_t{=}s, a_t{=}a)$ (state transition probability)
- $R(s,a) \triangleq \mathbb{E}\left[r_{t+1} \mid s_t{=}s, a_t{=}a\right]$ (expected reward)
- $\pi(s_t) \triangleq a_t$ (policy)

we can rewrite the state-value function as:

$$V^\pi(s) \triangleq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,\middle|\, s_t{=}s\right] = \mathbb{E}_\pi\left[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+1+k+1} \,\middle|\, s_t{=}s\right]$$
$$= R(s,\pi(s)) + \gamma\sum_{s'\in S} T(s'|s,\pi(s))\, V^\pi(s')$$

Similarly, we can rewrite the action-value function as:

$$Q^\pi(s,a) = R(s,a) + \gamma\sum_{s'\in S} T(s'|s,a)\, Q^\pi(s',\pi(s'))$$

Optimal policy

The best control policy, indicated with π^*, is clearly that which maximises the expected cumulative reward, thus

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \text{ for all } s.$$

The state-value and action-value functions given by the optimal policy π^* are denoted $V^*(s)$ and $Q^*(s,a)$, respectively.

If we have a way to find $Q^*(s,a)$, an optimal policy can be determined as

$$\pi^* : \pi^*(s) = \arg\max_{a \in A} Q^*(s,a).$$

This means that if we know the values of $Q^*(s,a)$, then by using a greedy search at each local step we get the optimal sequence of steps maximising the cumulative reward. Finally, also observe that:

$$V^*(s) = \max_{a \in A} Q^*(s,a).$$

Bellman's Optimality Equation

Following the optimal policy $\pi^*(s) = \arg\max_{a \in A} Q^*(s,a)$, Bellman's eqs

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s'|s, \pi(s))\, V^\pi(s') \qquad \text{and}$$

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)\, Q^\pi(s', \pi(s'))$$

become

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)\, V^*(s') \right] \qquad \text{and}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a \in A} Q^*(s', a)$$

These are recursive equations; no closed form solution (in general).
There are iterative solution methods, we'll now see a couple of them.

Outline

Model-Based Learning

In Model-Based Learning:

- The agent has access to model, i.e., has a copy of the MDP (the outside world) in its mind
- Using that copy, it tries to “think” what is the best route of action (mentally simulating possible scenarios and oucomes)
- It then executes this policy on the real world MDP
- In this case, we do not need any exploration and can directly solve for the optimal value function and policy

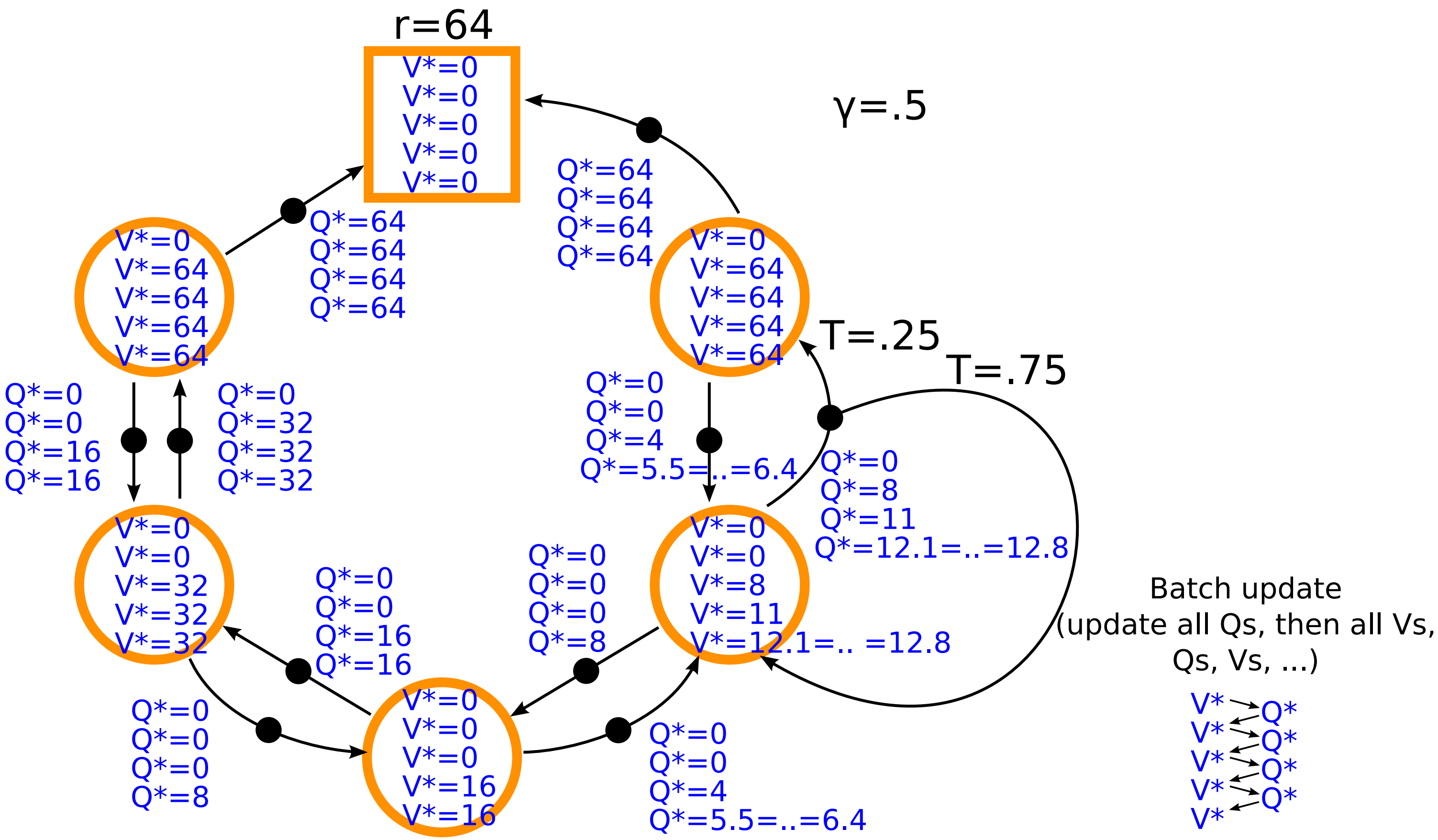
Model-Based Learning

Value iteration algorithm for model-based learning:

INITIALIZE $V(s)$ to random values
REPEAT UNTIL CONVERGENCE
    FOR all $s \in S$
        FOR all $a \in A$
            $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)V(s')$
        $V(s) \leftarrow \max_{a \in A} Q(s,a)$

If the values of all states s' out of a given state s are known exactly, we obtain the final $V^*(s)$ in one iteration. In general, the algorithm will iteratively converge to the correct $V^*(s)$ values.

Value iteration (example)



Value iteration (batch version) alternates between:

UPDATE ALL Qs: $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)V(s')$

UPDATE ALL Vs: $V(s) \leftarrow \max_{a \in A} Q(s,a)$

Model-Based VS Model-Free Learning

In "Model-Based Learning", once the agent has "internally" computed the value of each state (or state-action) using the internal model, it will act on the world using the optimal policy $\pi^*(s) = \arg\max_{a \in A} Q^*(s, a)$.

But, in general, we are concerned with learning when T and R are not known in advance. This is called "Model-Free Learning".
There are two alternatives:

- The agent explores the world to learn T and R first, then it proceeds to learn V^*
- The agent explores the world and tries to learn Q^* directly

This second approach is more efficient in practice.

Deterministic Model-Free Learning

We start assuming deterministic transitions and rewards, i.e. taking an action from a given state will always lead to the same new state and always result in the same reward.
As a result, Bellman's optimality equation simplifies to:

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \max_{a \in A} Q^*(s_{t+1}, a)$$

We can use the following algorithm implementing Deterministic Q-learning:

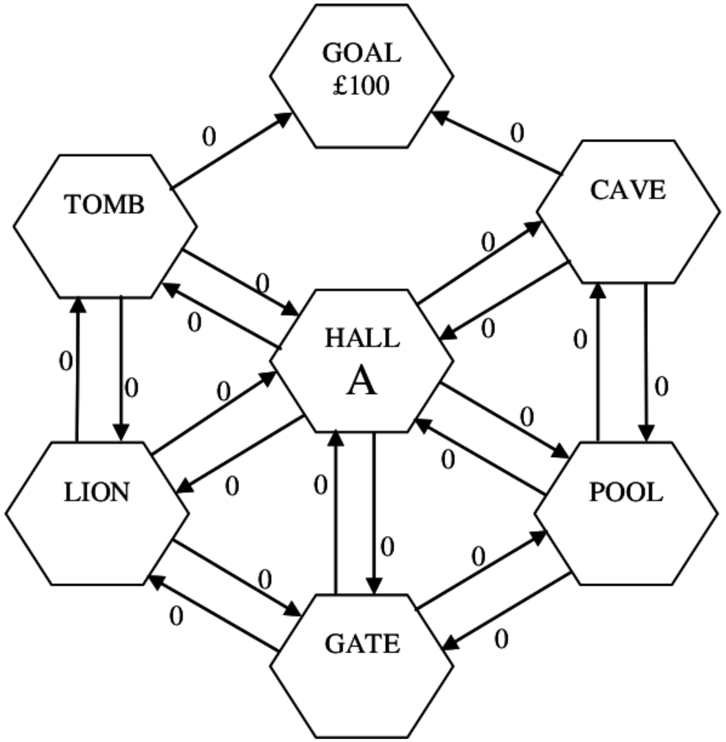
Initialise all $\hat{Q}(s,a)$ to zero (or small random values)
REPEAT UNTIL CONVERGENCE
    From state s_t, select an action a_t using an exploration policy
    The world responds moving the agent into state s_{t+1} and giving reward r_{t+1}
    Update the estimate of Q for the last action:

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a \in A} \hat{Q}(s_{t+1}, a)$$

    Increment t

An Example (1)

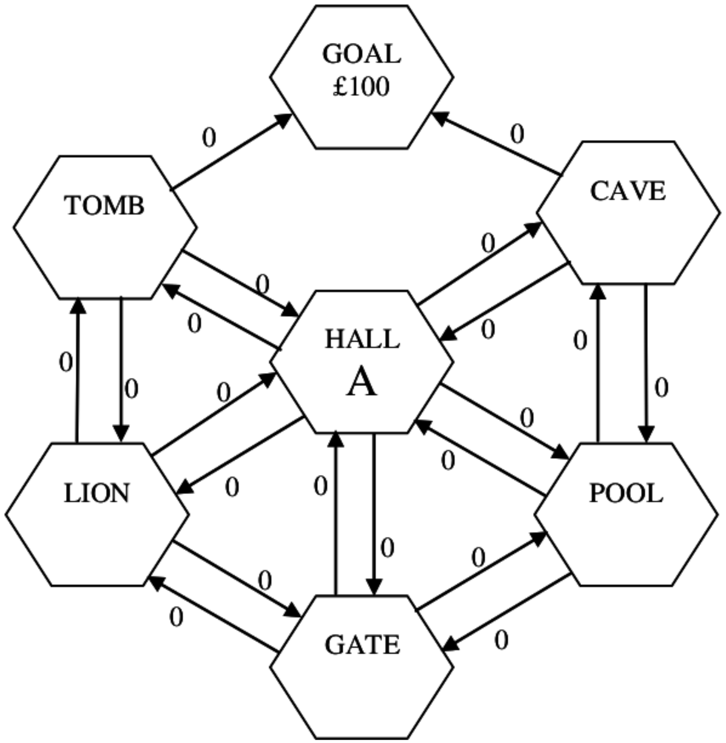
Suppose we want to write a program to learn to play a simple "adventure" type game. The game finishes when the player reaches the goal state and receives a reward of 100.



Following the Q-learning algorithm, we initialise all estimates of Q to zero: $\hat{Q}(s, a) = 0$ for all states s and actions a.

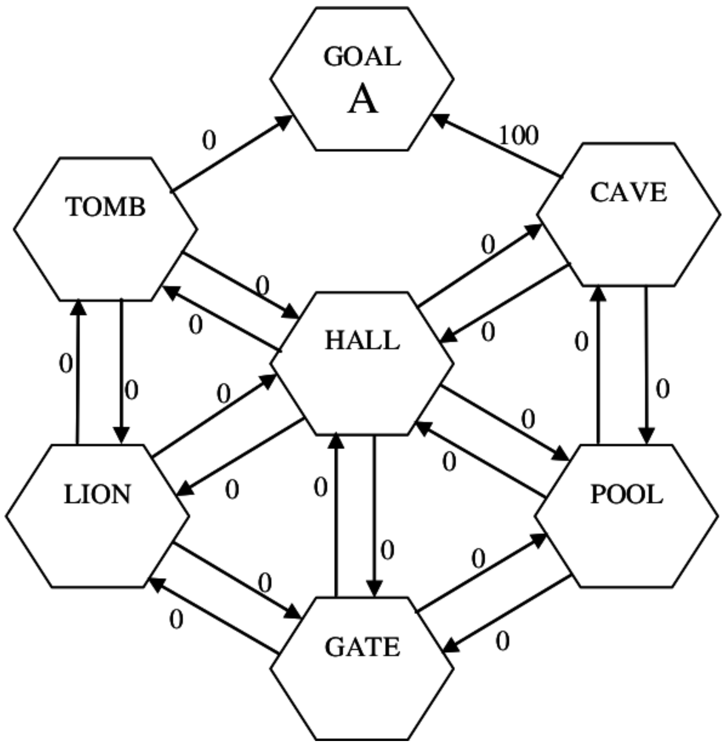
An Example (2)

Now suppose the agent begins at state HALL and selects the action ToCAVE.
The reward is zero and $\hat{Q}(\texttt{CAVE}, a) = 0$ for all actions a.
Hence, applying the Q-learning update procedure
$\hat{Q}(\texttt{HALL}, \texttt{ToCAVE}) \leftarrow r(\texttt{CAVE}) + \gamma \max_a \hat{Q}(\texttt{CAVE}, a)$ produces no change.



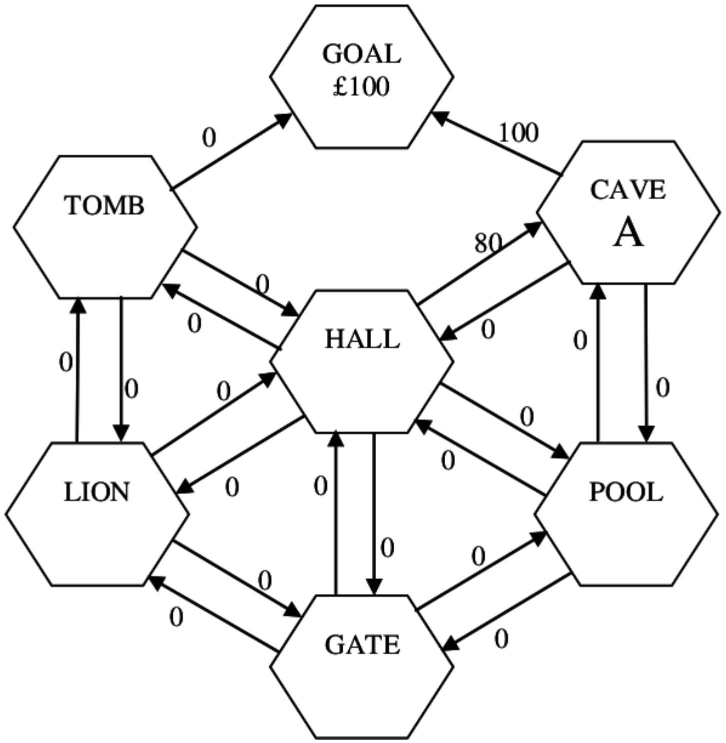
An Example (3)

Next suppose the agent, now in state CAVE, selects action ToGOAL.
The reward is 100 and $\hat{Q}(\texttt{GOAL}, a) = 0$ for all actions (there are no actions).
Hence $\hat{Q}(\texttt{CAVE}, \texttt{ToGOAL}) \leftarrow r(\texttt{GOAL}) + \gamma \max_a \hat{Q}(\texttt{GOAL}, a) = 100$



An Example (4)

Let's start at hall again and select the same action $\texttt{ToCAVE}$.
The reward is zero and $\hat{Q}(\texttt{CAVE}, \texttt{GOAL}) = 100$ while $\hat{Q}(\texttt{CAVE}, a) = 0$ for all other actions a
Hence $\max_{a \in A} Q(\texttt{CAVE}, a) = 100$. Assuming $\gamma = 0.8$, we have
$\hat{Q}(\texttt{HALL}, \texttt{ToCAVE}) = r(\texttt{CAVE}) + \gamma \max_{a} \hat{Q}(\texttt{CAVE}, a) = 0 + 0.8 \cdot 100 = 80$



Exploration / Exploitation

The choice of actions determines the agent's trajectory through state space and hence its learning experience.

How should the agent choose?

U) Uniform random selection (explore)
- Advantage: Will explore the whole space and thus satisfy criteria of convergence theorem.
- Disadvantage: May spend a great deal of time learning the value of transitions that are not on any optimal path.

G) Select action with highest expected cumulative reward (exploit)
- Advantage: Concentrates resources on apparently useful transitions.
- Disadvantage: May ignore even better pathways whose value has not been explored, and does not satisfy convergence criterion.

One of the most common methods:

ϵ-greedy: act greedily (G) with probability $1 - \epsilon$, random (U) otherwise

Algorithms for non-deterministic settings

What can we do if the MDP is not deterministic?

- Q-learning:
  $$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \eta \left[R(s,a) + \gamma \max_{a' \in A} \hat{Q}(s',a') - \hat{Q}(s,a) \right]$$
  where η is a small learning rate, e.g., $\eta = 0.001$.
- SARSA(0)
- SARSA(1)/MC

(beyond the scope of this course)

Outline

- Modelling the problem
  - Introduction
  - Markov Decision Process
  - Policy, rewards, discount factor
  - Bellman's equations
- Learning
  - Model-Based Learning
  - Model-Free Learning
  - Deterministic Q-Learning
  - Exploration / Exploitation
  - Non-deterministic settings

References

Required course material reading:
Alpaydin 2010/2014
18.1, 18.3, 18.4, 18.4.1, 18.5.1 (epsilon-greedy,~~softmax~~), 18.5.2, 18.5.3
(~~on-policy~~, ~~off-policy~~, ~~sarsa~~, ~~TD learning~~)

Reinforcement Learning: An Introduction, by RS Sutton and AG Barto
(http://incompleteideas.net/sutton/book/the-book.html)
3.1, 3.2, 3.5

Further reading:
Mitchell 1997
Chapter 13

Credits:
Partly based on previous slides by Paul Scott and Spyros Samothrakis, and on the book by Sutton&Barto