CE213 Artificial Intelligence – Lectures 17&18

Reinforcement Learning

No training data available: Learning from experience/exploration rather than from sample data

To learn what to do next or find a sequence of actions

Applications: game playing and robotic control

Focus: evaluation of states and actions

[Warning: It may be difficulty to understand the Q learning concept and procedure]

Reinforcement Learning

WHY IS REINFORCEMENT LEARNING DIFFICULT?

Consider the following learning tasks:

- A child learning to ride a bicycle.
- A person/computer learning to play chess.
- A person learning how to find a way out of a complex maze.
- A robot learning how to navigate in a complicated environment.
- ……

Are these learning tasks more difficult than classification or prediction, e.g., face recognition, medical diagnosis?

\- Yes, due to the involvement of time/dynamics and no immediate teaching signal available. This would be the same for human learning.

Despite the different domains, these learning tasks have much in common.

Reinforcement Learning (2)

What do these learning tasks have in common?

- This type of learning is to choose a sequence of actions that will lead to a reward (usually long-term goal, similar to game playing).
- The ultimate consequence of an action may not be apparent until the end of a sequence of actions (no immediate teaching signal).
- When a reward is achieved, it may not be due to the last action performed, but due to one earlier in the sequence of actions.
- In contrast to classification or clustering, there is no pre-defined set of training samples. The experiences that form the basis of reinforcement learning are derived through the ***exploration in a learning environment***, instead of training data. So, ***reinforcement learning is not supervised learning or unsupervised learning*****.**

Markov Decision Process as Problem Representation

- An agent (e.g., a robot or an AI game player) is operating in a domain that can be represented as a set of distinct states, represented as a set $\boldsymbol{S}$.
- The agent has a set of actions that it can perform, represented as a set $\boldsymbol{A}$.
- Time advances are in discrete steps, with a fixed time step.
- At time t the agent knows the current state $s_t \in \boldsymbol{S}$ and must select an action $a_t \in \boldsymbol{A}$ to perform.
- When the action a_t is carried out, the agent will receive a reward r_t (it could be 0 if there is no reward) and the agent enters a new state s_{t+1}.
- The reward, which may be positive, negative or zero, depends on both the state and the action chosen, so $r_t \equiv r(s_t, a_t)$, where r is the **reward function**.
- The new state also depends on both the state and the action chosen, so $s_{t+1} = T(s_t, a_t)$, where T is the **transition function**.

In a ***Markov decision process***, the functions r and T depend only on the ***current*** action and state. (no memory of the past)

Control Policy
– what action to take at a given state?

The agent in reinforcement learning must learn how to choose the **best action at each state**. This is the key issue in reinforcement learning.

Therefore, it must acquire a ***control policy*** π, which is a mapping from $\boldsymbol{S} \rightarrow \boldsymbol{A}$.

That is, $a_t = \pi(s_t)$, for any state at time t.

So what do we mean by "best action"?

- This is related to the reward of an action or values of states as the consequence of a sequence of actions.

Learning to evaluate actions and states is a main task of reinforcement learning.

(It could help you understand better if you associate reinforcement learning with game playing or robot navigation.)

What is the "best action"?

If we define the best action as the one leading to the greatest **immediate reward**, this would produce a good short-term payoff, but might not be optimal in the long run. Also, immediate reward may not be clear or available.

It usually makes more sense to maximise the **total payoff** over time.

We could define the payoff of a sequence of actions starting from the current state s_0 to goal state s_N as the sum of all the rewards corresponding to these states:

$$V \equiv \sum_{i=0}^{N} r_i = r_0 + r_1 + \ldots + r_N$$

where the sum is taken over all the states involved in the sequence of actions.

However, this makes a reward in the very distant future just as valuable as one received immediately, which is often unrealistic.

Discounted Cumulative Reward

So, we need some way of weighting the rewards so that the more distant they are the less they are worth.

Quite naturally, we can introduce a constant γ, as a discount factor, to indicate the relative values of immediate and delayed rewards:

$$V \equiv \sum_{i=0}^{N} \gamma^i r_i = \gamma^0 r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^N r_N$$

where $0 < \gamma \leq 1$, r_i is the reward at state s_i, $i=0$ represents the current state or time (time as discrete steps). The total payoff calculate in this way is called the ***discounted cumulative reward***.

There may be more than one sequence of actions starting from a state to the goal state, corresponding to different routes. The V value should be the one calculated using the optimal route, i.e., the maximum value.

(see example later)

Optimal Control Policy

The optimal control policy, represented as π*, is clearly the one that **maximises the discounted cumulative reward for each state**, thus

$$\pi^* \equiv argmax_{\pi} V^{\pi}(s) \text{ for all } s$$

where V^{π} denotes the discounted cumulative reward function when the actions in the sequence are chosen using control policy π. Different control policies lead to different sequences of actions.

The discounted cumulative reward given by the optimal control policy π* is denoted as *V*(s)*.

The Learning Task

We can now define what we want to learn in reinforcement learning:

The learning task is to discover the optimal control policy π*,
i.e., the best action at each state.

This is similar to find the optimal route or optimal sequence of operations /moves in state space search. However, the focus in reinforcement learning is evaluation of states and actions, rather than search strategies.

Indirectly, the learning task is **to find out the maximum discounted cumulative reward values of all states**. As a matter of fact, this is the focus of reinforcement learning (most of the remaining lecture on reinforcement learning is about how to calculate or estimate the discounted cumulative reward values).

An Example Domain/Scenario

Suppose we want to write a program to learn to play an extremely simple "adventure" game (Similarly, we can handle a complex one for practical applications if we have time).

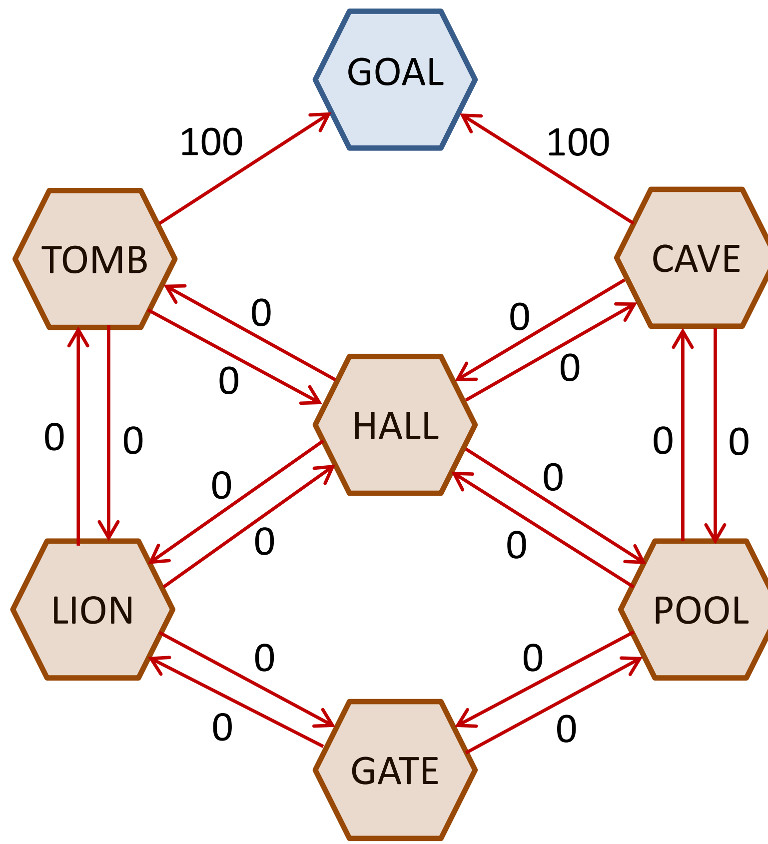
The domain may be represented as follows:

Hexagons denote **states**.

Arrows indicate the possible **actions** for each state.

The game finishes when the player reaches the goal state and receives a reward of £100.

Numbers adjacent to arrows indicate values of the reward function *r*, i.e., the **immediate reward** associated with the state transition. Only two non-zero reward values are shown in the figure. The other reward values are not known, and we assume they are all zero.



Discounted Cumulative Rewards

For this simple game it is easy to determine the optimal control policy π*, or the best actions at each state.

Suppose that the discount factor γ=0.8, then we can easily calculate the **discounted cumulative reward** *V*(s)* for every state using

$$V \equiv \sum_{i=0}^{N} \gamma^i r_i$$

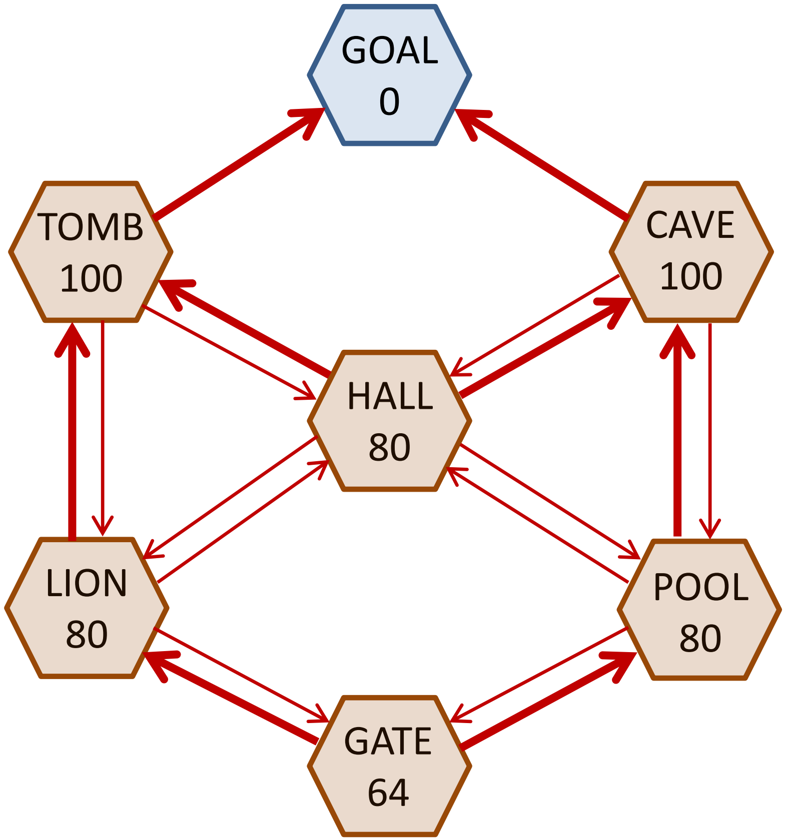
For example:

$$\underbrace{\gamma^0 r_0}_{\text{GATE to POOL}} + \underbrace{\gamma^1 r_1}_{\text{POOL to CAVE}} + \underbrace{\gamma^2 r_2}_{\text{CAVE to GOAL}}$$

$$\begin{aligned} V^*(GATE) &= \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 \\ &= 0.8^0 \times 0 + 0.8^1 \times 0 + \ 0.8^2 \times 100 \\ &= 64 \ \ (optimal) \end{aligned}$$

(there are other routes with smaller values)

The best actions at each state are indicated by thick arrows.



(Where to go at POOL?)

Optimal Control Policy and Q Function

Suppose the agent in reinforcement learning knows:

- the transition function $T(s, a)$
- the reward function $r(s, a)$
- the discounted cumulative reward of each state $V^*(s)$

Then the optimal control policy for state s would be

$$\pi^*(s) = argmax_{\pi} V^{\pi}(s) = argmax_a[r(s, a) + \gamma V^*(T(s, a))]$$

Therefore, V^* can be used as an evaluation function for actions or states.

However, for many applications we do not know the immediate reward values $r_0, r_1, r_2, \ldots, r_N$ for calculating discounted cumulative reward. Even for the simple game just considered earlier, we assume most of the r values are zero. From the point of view of machine learning, it could be a good idea for the agent to try to learn V^* if there is insufficient knowledge about it.

Optimal Control Policy and Q Function (2)

Suppose an agent is in state s and is trying to select its next action.

If the agent does not know the transition function T, then no form of **action evaluation** that requires looking ahead is possible.

What information does such an agent need to know?

It is the total payoff that can be expected for each possible choice of action a in the current state s, i.e., $V^*(s)$, but this may not be available.

So, let's *define* such an **evaluation function** as follows, which is called Q function and related to V^*:

$$Q(s, a) \equiv r(s, a) + \gamma V^*(s')$$

where s' is the state resulted from action a in the current state s.

Thus, in place of V^*, which is **a function of state only and may not be available**, we now have Q, which is **a function of both state and action**.

Why is Q function useful?

If we know Q, we no longer need to know transition function T and reward function r to determine the best actions because

$$\pi^*(s) = argmax_a Q(s,a)$$

The information, which could be obtained by looking ahead using T and r , has been absorbed into the Q function.

However, in many applications we do not have $V^*(s)$ and do not have an analytical Q function either.

Is it possible to learn Q? Yes, that's why the Q function is useful!

- Chris Watkins (http://www.cs.rhul.ac.uk/~chrisw/) proposed a Q learning algorithm in 1989 when he was a PhD student at Cambridge University. He proposed a formula to express the Q function and an iterative learning algorithm to estimate the Q values through iterative updating.

Learning Q

The Problem

The agent requires a procedure that will be able to form a good estimate of Q as a result of its experiences obtained from exploration in the domain.

Such experiences only provide direct information about **immediate rewards**.

But the true value of Q depends on **a sequence of immediate rewards**.

The Solution

We can exploit the fact that the value of Q for a state s and an action a depends on the Q values of the neighbours of state s, because

$$Q(s,a) \equiv r(s,a) + \gamma V^*(s') \coloneqq r(s,a) + \gamma max_{a'} Q(s',a')$$

where $\equiv$ means “is equivalent to …” and $\coloneqq$ means ”is defined by …”

The trick: Making Q iteratively updatable,
so that it is possible to estimate Q through machine learning.

The Q Learning Algorithm (pseudo code)

No need of functions r and T

Suppose there are:

m states $s_1 \ldots s_m$ and n actions $a_1 \ldots a_n$ *//the domain*

Create an $m \times n$ array QE to hold estimates of $Q(s,a)$ and an m x n array rE to hold estimates of $r(s,a)$.

Initialise all entries in QE to zero, *i.e.*, $QE(s,a)=0$.

Initialise all entries in rE with given initial values or zero if not given.

Select an initial state s_i *//could be randomly chosen from* $s_1 \ldots s_m$

$s := s_i$

REPEAT

Select and execute an action a to reach a new state s' *//could be randomly*

$QE(s,a) := rE(s,a) + \gamma \times \max_{a'} QE(s',a')$ *//estimate Q function value*

$rE(s,a) := QE(s,a)$ *//update immediate reward*

If a' is not empty (action available at s'), then $s := s'$.

Otherwise, set s to another initial state *//could be randomly*

UNTIL Termination Condition Satisfied

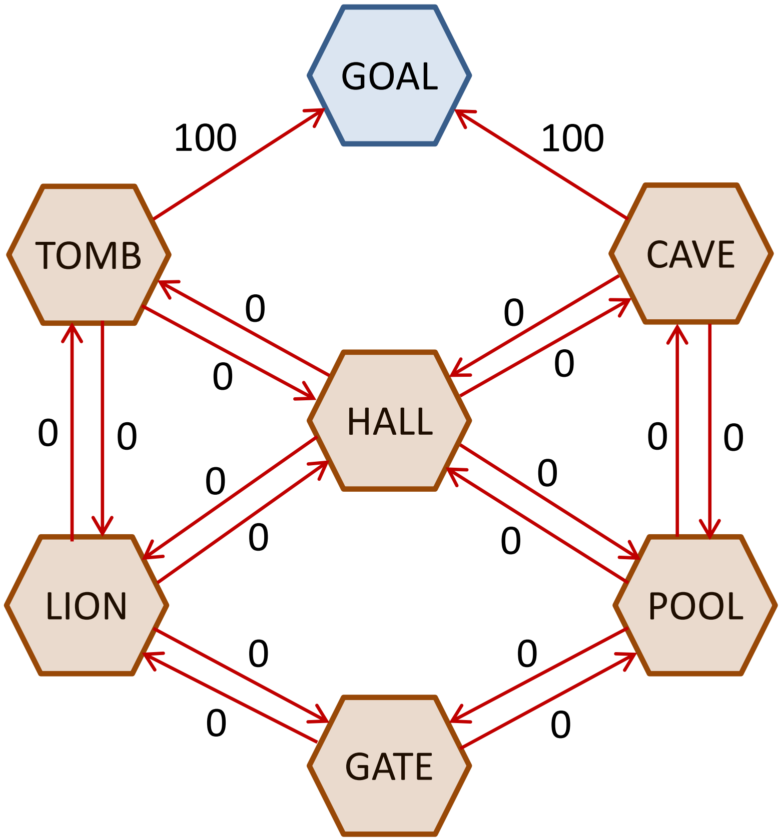
//pre-set number of iterations reached or QE and rE are stable.

Q Learning in Action

Consider learning the Q values of states and actions in the same 'adventure' game domain as shown in the diagram.

Numbers adjacent to arrows indicate initial values of the estimates of immediate rewards (i.e., rE).

Initially all the estimates of Q (i.e., QE) will be zero.



Q Learning in Action (2)

To start iterative updating of Q values, **suppose** the agent begins at state HALL and selects the action To-CAVE (**kind of random**). Therefore, we have s=HALL, a=To-CAVE, s'=CAVE.

Applying Q-learning update procedure …

Before updating:

rE(HALL, To-CAVE) = 0

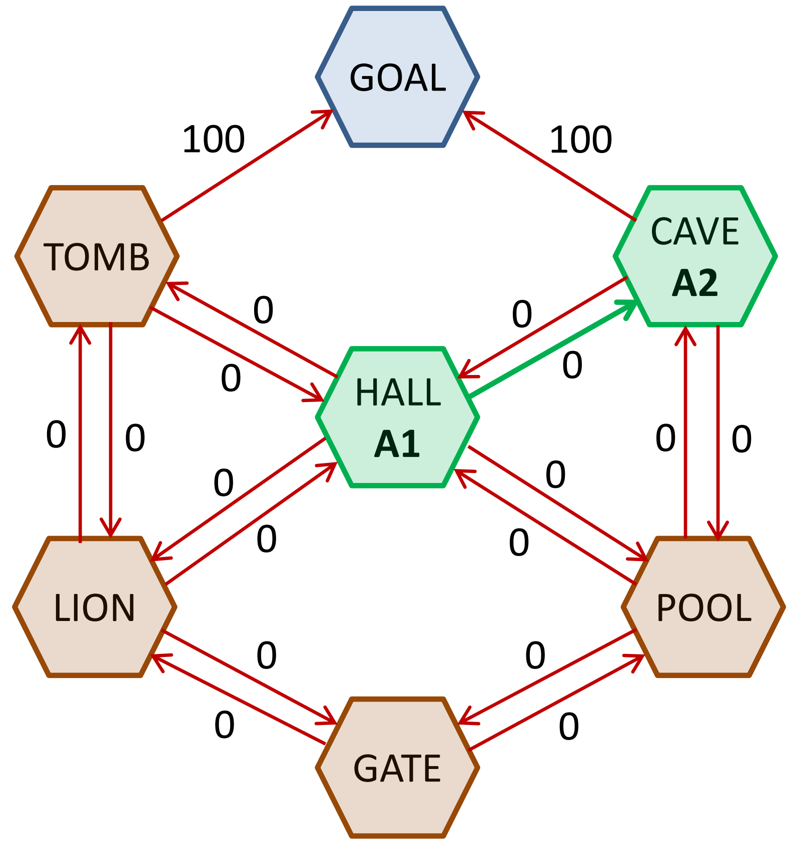
QE(CAVE, a') = 0 for all actions a'

After updating:

QE(HALL, To-CAVE) = 0+ γ x 0 =0

rE(HALL, To-CAVE) = QE(HALL, To-CAVE) = 0

Nothing changed by this updating.



$QE(s,a) := rE(s,a) + \gamma \times max_{a'} QE(s',a')$

$rE(s,a) := QE(s,a)$

Q Learning in Action (3)

Now the agent is in state CAVE, **suppose** it selects the action To-GOAL. Therefore we have
s=CAVE, a=To-GOAL, s'=GOAL.

Applying the update procedure again …

Before updating:

rE(CAVE, To-GOAL) = 100

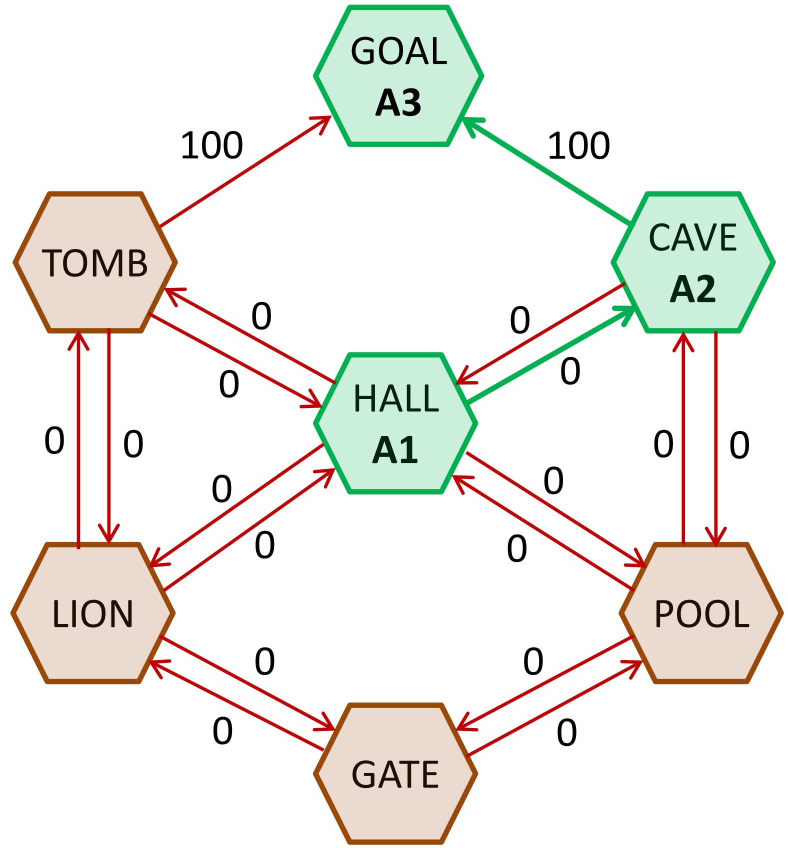
QE(GOAL, a') = 0

After updating:

QE(CAVE, To-GOAL) = 100 + γ x 0 = 100

rE(CAVE, To-GOAL) = QE(CAVE, To-Goal) =100

The estimated *Q* value for the CAVE-GOAL transition has been improved.



$QE(s,a) := rE(s,a) + \gamma \times max_{a'} QE(s',a')$
$rE(s,a) := QE(s,a)$

Q Learning in Action (4)

Since GOAL is a sink state in this domain, the agent must start again with a new initial state . **Suppose** it begins once more at HALL, and once again selects the action To-CAVE. Therefore we have s=HALL, a=To-CAVE, s'=CAVE.

Before updating:

rE(HALL, To-CAVE) = 0

QE(CAVE, To-GOAL) = 100

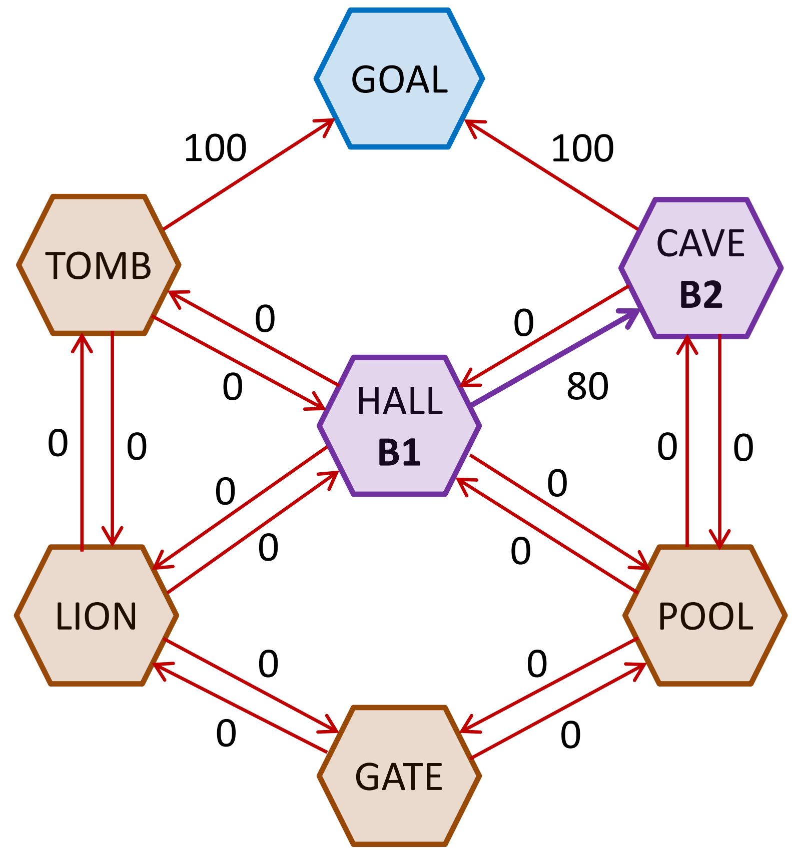
QE(CAVE, a') = 0 for all other actions

$\text{Max}_{a'}$QE(CAVE, a') = 100

After updating (γ = 0.8):

QE(HALL, To-CAVE) = 0 + γ * 100 = 80

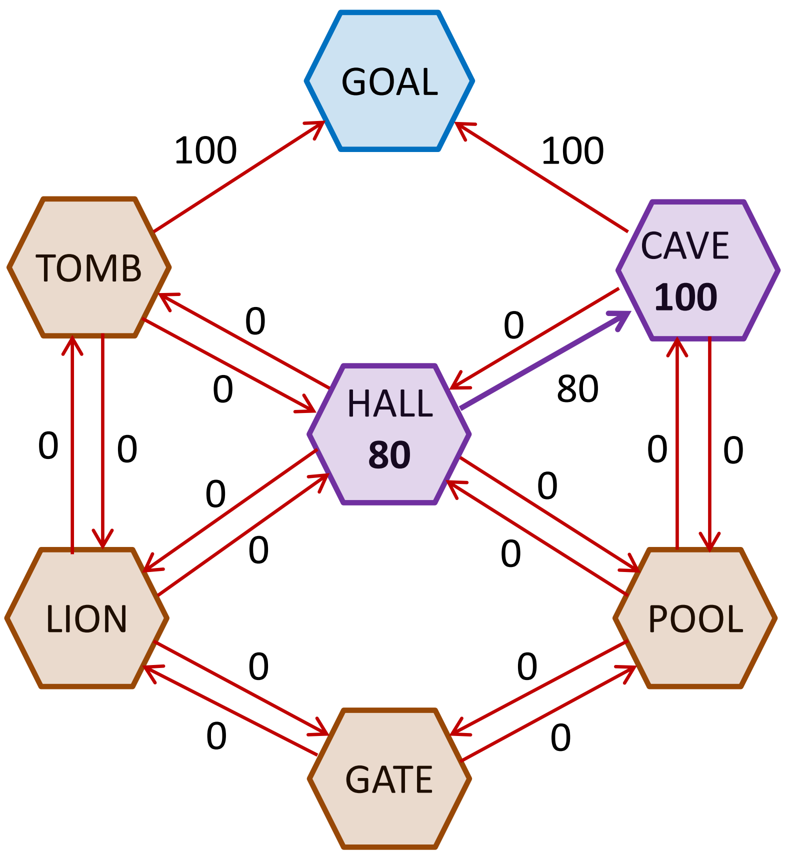
rE(HALL, To-CAVE) = QE(HALL, To-CAVE) = 80

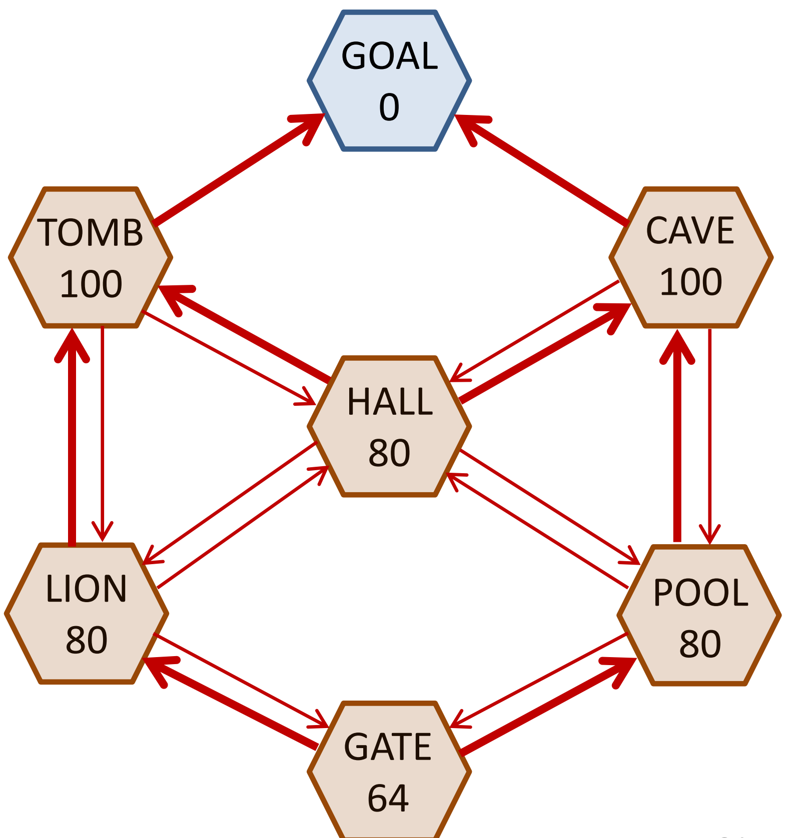


QE(s,a) := rE(s,a) + $\gamma \times \max_{a'}$ QE(s',a')
rE(s,a) := QE(s,a)

Q Learning in Action (5)

After a few iterations, Q-learning has obtained estimated Q values for CAVE and HALL, which are the same as the previously calculated discounted cumulative reward values. Choosing different initial states and repeating Q-learning update procedure we can obtain estimated Q values for other states.





The Q Learning Algorithm (pseudo code)

No need of functions r and T

Suppose there are:

m states $s_1 \ldots s_m$ and n actions $a_1 \ldots a_n$ *//the domain*

Create an $m \times n$ array QE to hold estimates of $Q(s,a)$ and an m x n array rE to hold estimates of $r(s,a)$.

Initialise all entries in QE to zero, *i.e.*, $QE(s,a)=0$.

Initialise all entries in rE with given initial values or zero if not given.

Select an initial state s_i *//could be randomly*

$s := s_i$

REPEAT

Select and execute an action a to reach a new state s' *//could be randomly*

$QE(s,a) := rE(s,a) + \gamma \times \max_{a'} QE(s',a')$ *//estimate Q function value*

$rE(s,a) := QE(s,a)$ *//update immediate reward*

If a' is not empty (action available at s'), then $s := s'$.

Otherwise, set s to another initial state *//could be randomly*

UNTIL Termination Condition Satisfied

//pre-set number of iterations reached or QE and rE are stable.

Some Issues in Q Learning

As indicated in the Q learning algorithm (pseudo code), the steps for updating QE(s, a) and rE(s, a) can be done iteratively, with different initial states and different choices of actions at each state involved in the agent's learning trajectory. **How to choose initial states and actions at a specific state? Any better ways than random choice?**

In general, each state should be visited many times during the Q learning process. This means the number of iterations could be very large, especially when there are many states and actions. Therefore, **Q learning could be very slow. How to improve Q learning?**

When should the Q learning stop? - There could be different stop criteria. For example, when the pre-set maximum number of iterations have been conducted or when there has been no improvement/change of QE values by the Q learning iterations.

Strategies for Selecting Initial States and Actions

The choice of initial states and actions at a given state determines the agent's learning trajectory through state space and hence its learning experience and outcome.

How should the agent choose initial states and actions during Q learning?

- Uniform random selection

  Advantage: Will explore the whole state space with equal opportunity.

  Disadvantage: May spend a great deal of time learning the values of transitions that are not on any optimal path.

- Select actions with highest expected cumulative reward (exploitation)

  Advantage: Concentrates resources on apparently useful transitions.

  Disadvantage: May ignore even better pathways whose values have not been explored.

  How about trade-off between exploitation and exploration, as what is done for node selection in Monte-Carlo tree search?

Learning Q Values by Neural Networks
(if training data is available, e.g. from previous Q learning)

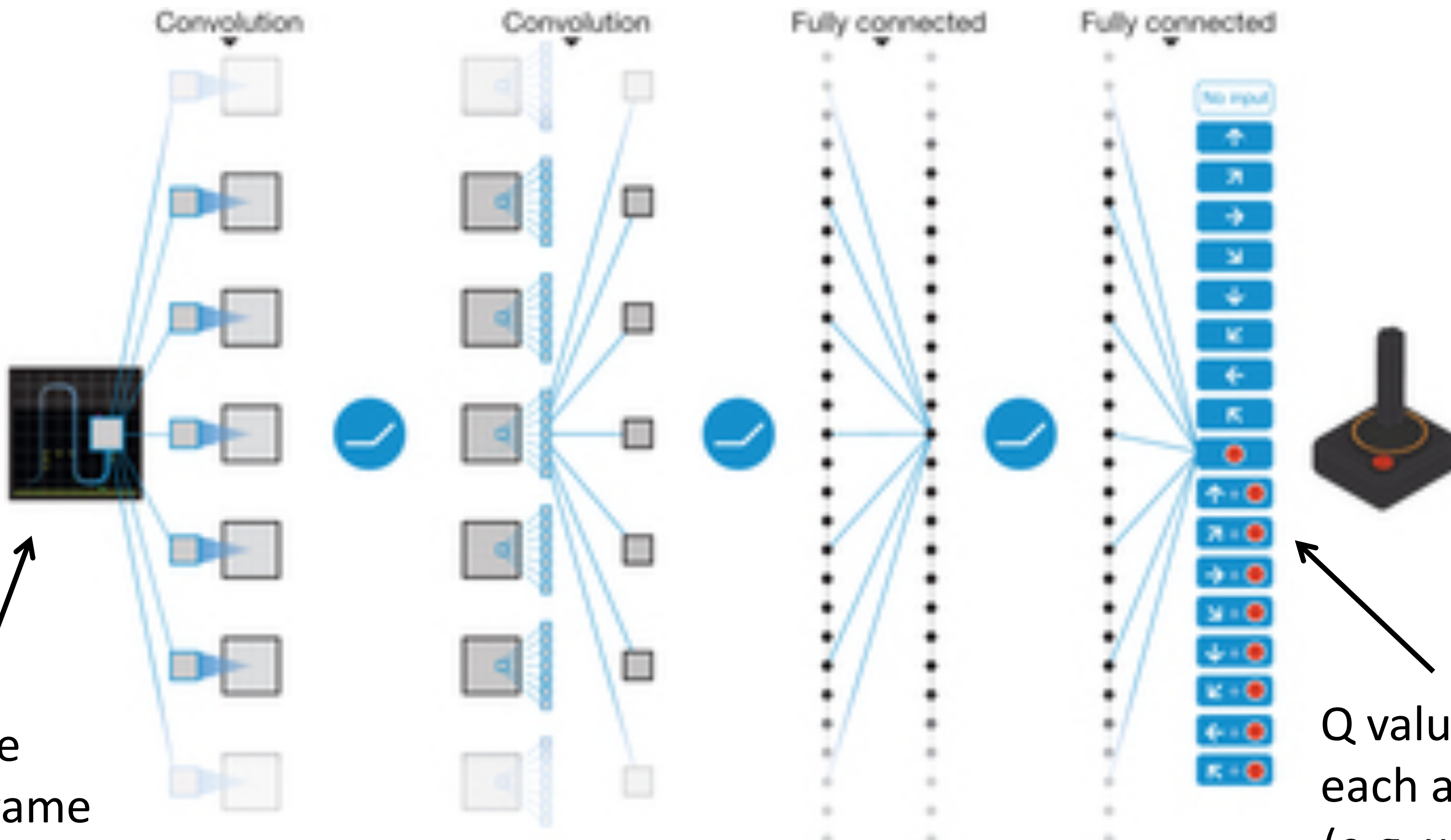
In many applications (e.g., game Go), the total number of state-action pairs can be very large. **Both time complexity and space complexity of Q learning could be too high.**

However, in many domains, **similar actions in similar states usually have similar consequences**. This fact can be exploited to generalize the results of Q learning by using multilayer neural networks to carry out function approximation.

Two approaches:
- Using one neural network: input is (***s, a***), output is *Q*.
- Using *n* neural networks, where *n* is the number of actions: input is ***s*** for all the neural networks, output of each neural network is *Q* corresponding to a specific action. This approach is better in general.

Problem: It requires labelled training data, which could be from the interim results of Q learning, leading to the combination of Q learning and learning in neural networks.

Learning Q Values by a Deep Neural Network



State
(e.g. game
position)

Q value for
each action
(e.g. winning
probability)

AlphaGo’s Value Network

[This is equivalent to *n* neural networks, each with one output unit.]

Summary

Reinforcement Learning:

Tasks of reinforcement learning

Markov Decision Process as Problem Representation:

It is similar to state space representation
with extra reward function

Control policy

Discounted cumulative reward

Q Learning:

The Q function (replacing V*(s))

The Q learning algorithm (pseudo code)

Q learning in action – an example

Strategies for selecting initial states and actions

Learning Q values by neural networks