REINFORCEMENT LEARNING

THE PROBLEM

Consider the following learning problems:

- A child learning to ride a bicycle.
- A person learning to play chess.
- A rat learning to run a maze.
- A driver learning the best route between his home and his office in rush hour traffic.
- A robot learning how to find the recharging unit in a laboratory.

Despite their different domains these learning tasks have much in common:

- 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 apparent until the end of a sequence of actions.
- When a reward is achieved, it may be due not to the last action performed but to one earlier in the sequence. This is known as the credit assignment problem.
- In contrast to classification learning, there is no pre-defined set of training examples. The experiences that form 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.

FORMALISING THE PROBLEM

A vast range of learning tasks resemble these examples.

To develop a method of addressing them, we must abstract the essential features of this class of learning problem.

One of the most productive is to regard the task as a Markov decision process.

Representation as a Markov Decision Process

- The agent is operating in a domain that can be represented as a set S of distinct states.
- The agent has a set A of actions that it can perform.
- Time advances in discrete steps.
- At time t the agent knows the current state $s_t \in S$ and must select an action $a_t \in A$ to perform.
- When the action a_t is carried out the agent will receive a reward r_t 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 $s_{t+1} = \delta(s_t,a_t)$, where δ is the transition function.

In a Markov decision process, the functions r and δ depend only on the current action and state.

Note: The agent may well have no knowledge of r and δ other than that obtained by interacting with the domain.

Defining a Control Policy

The agent must learn how to choose the best action for each state.

Thus it must acquire a control policy $\pi : S \rightarrow A$.

That is, $\pi (s_t,) = a_t$.

What is the "best action"?

Suppose 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.

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

We could define the payoff of a sequence of actions starting at initial state s_t as

$$V(s_t) = \Sigma r_t$$

where the sum is taken over all the states in the sequence.

But this makes a reward in the very distant future just as valuable as one received immediately. This is often unrealistic.

So we introduce a constant γ to indicate the relative value of immediate and delayed rewards thus:

$$V(s_t) \equiv \sum_{i=0}^{\infty} \gamma^i r_{i+t}$$

where $0 < \gamma \leq 1$.

This is called the discounted cumulative reward.

Alternatives to the Discounted Cumulative Reward

The discounted cumulative reward is only one of many possible cumulative reward functions. Other reasonable alternatives include:

- Finite horizon reward:

  Similar to cumulative discounted reward but only consider the first n terms of the series.

- Average reward:

  Mean reward per time step over a finite sequence:

$$V(s_t) \equiv 1/n \sum_{i=0}^{n} r_{i+t}$$

The Optimal Control Policy

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

$$\pi^* \equiv \text{argmax}_\pi \; V^\pi(s), \text{ for all } s.$$

where V^π denotes the cumulative reward function when the actions in the sequence are chosen using the policy π.

The cumulative reward given by the optimal policy π^* is denoted $V^*(s)$.

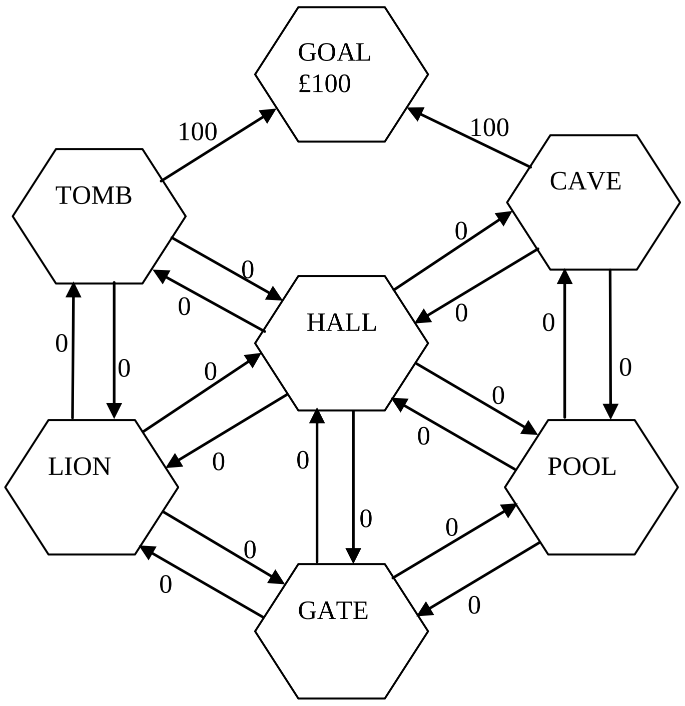
Definition of the Learning Task

The learning task is thus to discover the optimal policy π^*.

AN EXAMPLE DOMAIN

Suppose we want to write a program to learn to play a very simple “adventure” type game.

The domain may be represented thus:



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 that state transition.

Discounted Cumulative Values

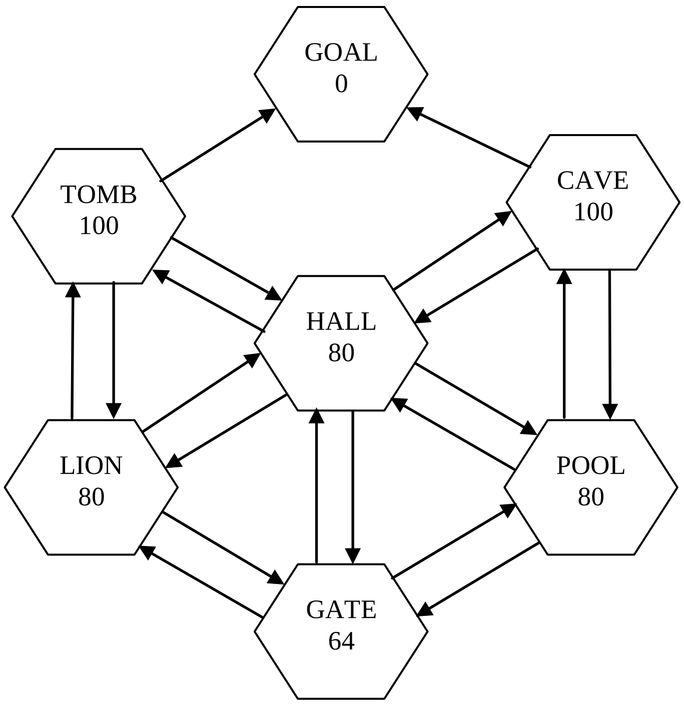
In this simple game it is easy to determine the optimal control policy π^*.

Suppose that we use 0.8 for the discount factor γ in the expression for discounted cumulative value.

Then we can easily calculate V*(s) for every state using

$$V(s_t) \equiv \sum_{i=0}^{\infty} \gamma^i r_{i+t}$$

thus:



e.g.

$$V^*(\mathrm{GATE}) = \gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3$$

$$= 0.8^0 \times 0 + 0.8^1 \times 0 + 0.8^2 \times 100$$

$$= 64$$

Optimal Policy with Complete Domain Theory

Suppose the agent knows:

- the transition function, $\delta(s,a)$
- the reward function, $r(s,a)$
- the discounted value of each state, $V^*(s)$.

Then the optimal policy for state s would clearly be

$$\pi^*(s) \equiv \operatorname{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

Thus V^* can be used as an evaluation function for actions.

This suggests that it would be a good idea for an agent to try to learn V^*.

BUT

- We are concerned with learning when δ and r are unknown.
- Without this knowledge, V^* cannot be used to evaluate actions.

So there are two alternatives:

1. The agent first learns δ and r and then proceeds to learn V^*.
2. The agent uses a different evaluation function that can be used to evaluate actions without knowledge of δ and r.

If it is possible, the second alternative would be easier.

THE Q FUNCTION

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

If the agent does not know δ or r then no form of action evaluation that requires lookahead is possible.

What information does such an agent need to know?

- The total payoff it can expect for each possible choice of action a in the current state s.

So lets define such an evaluation function $Q(s,a)$:

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

Thus in place of V^*, a function only of the state, we now have Q that is a function of both state and action.

Why is Q so useful?

If we know Q, we no longer need to know δ and r to determine the best action since:

$$\pi^*(s) = \mathrm{argmax}_a\, Q(s,a)$$

In effect, the information that could be obtained by looking ahead using δ and r has been absorbed into Q.

Is it possible to learn Q?

LEARNING Q

The Problem

- The agent requires a procedure that will enable it to form a good estimate of Q as a result of its experiences in the domain.
- Such experiences only provide direct information about immediate rewards.
- The true value of Q depends on a sequence of rewards.

The Solution

- Exploit the fact that the values of Q for a state s depends upon the Q values of the neighbours of s since:

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

Suppose the agent has reached state s_i and performs an action a_k and that $Q(s_i, a_k)$ is unknown.

It will reach state $s_j = \delta(s_i, a_k)$ and receive reward $r(s_i, a_k)$.

Suppose also that the agent already has estimates of the Q values for all the actions at state s_j.

Then the equation can be used to estimate $Q(s_i, a_k)$.

How good this estimate is will depend on how good the estimates of Q for s_j are.

If we can arrange that the estimates are continually improved as the agent explores the domain, then we have a way of learning Q.

The Q Learning Algorithm

Suppose there are:

m states, $s_1 \ldots s_m$

n actions $a_1 \ldots a_n$

```
Create the $m \times n$ array QE to hold estimates of $Q(s,a)$.
Initialize all entries in QE to zero.
Select an initial state $s_i$
$s := s_i$
REPEAT
        Select and execute an action a
        $s' := \delta(s,a)$
        $r := r(s,a)$
        $QE[s,a] := r + \gamma \times \max_{a'} QE[s',a']$
        $s := s'$
UNTIL Termination Condition Satisfied
```

This procedure will converge so that $QE[s,a] \rightarrow Q(s,a)$, provided:

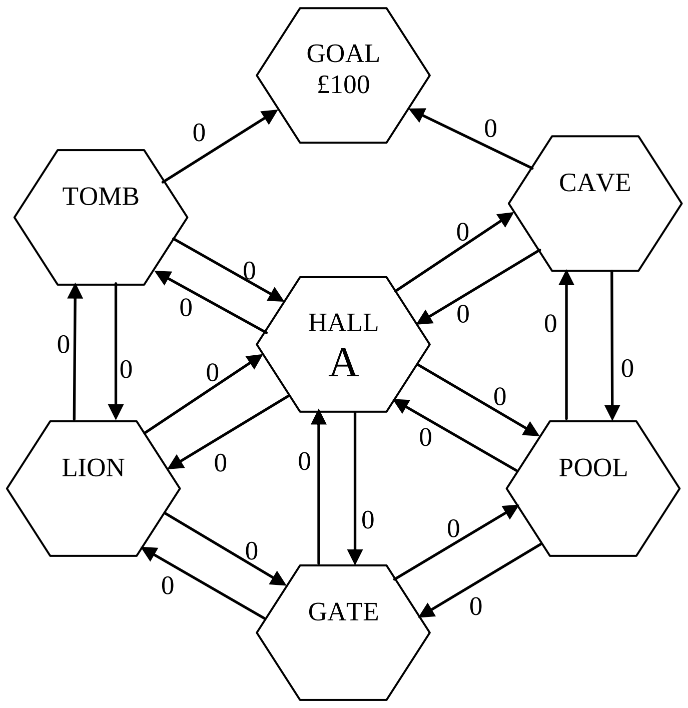
- The domain behaves as a deterministic Markov process.
- The reward function r is bounded.
- Every state-action pair is visited infinitely often

(For proof, see Mitchell pp 377-379).

Q Learning in Action

Consider learning the adventure game domain.

Initially all the QE estimates will be zero thus:



Now suppose the agent begins at state HALL and selects the action to To-CAVE.

r(HALL, To-CAVE) = 0

QE[CAVE,a] = 0 for all actions a.

Hence apply the Q learning update procedure produces no change.

Next suppose the agent, now in state CAVE, selects the action To-GOAL.

Applying the update procedure again:

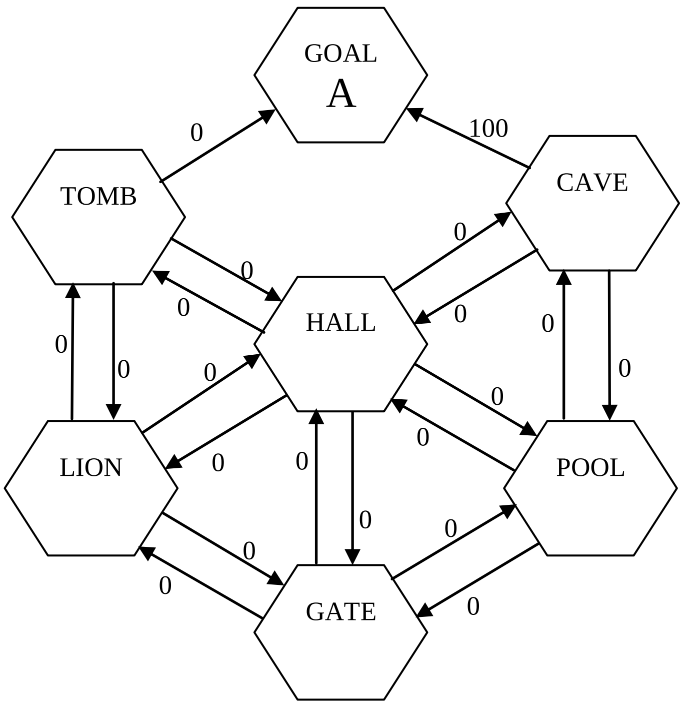
r(CAVE, To-GOAL) = 100

QE[GOAL,a] = 0 for all actions (there are no actions).

Hence

QE[CAVE, To-GOAL] = 100 + γ * 0 = 100

Thus the situation is now:



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

Since GOAL is a sink state in this domain, the agent must start again.

Suppose it begins once more at HALL, and once again selects the action To-CAVE.

$$r(\text{HALL, To-CAVE}) = 0$$

$$QE[\text{CAVE, GOAL}] = 100.$$

$QE[\text{CAVE},a] = 0$ for all other actions a.

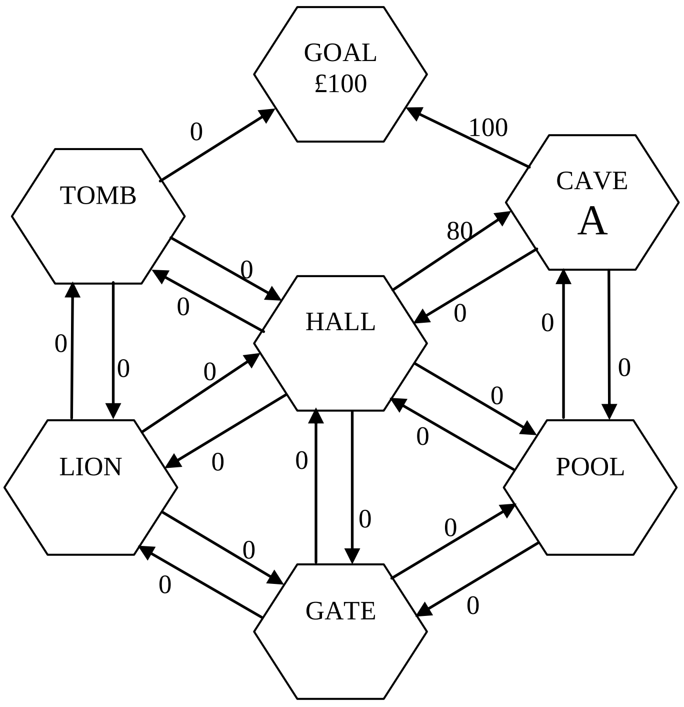
Hence

$$\max_a QE[\text{CAVE},a] = 100.$$

Applying the update procedure, using $\gamma = 0.8$:

$$QE[\text{HALL, To-CAVE}] = 0 + \gamma * 100 = 80$$

Thus the situation is now:



Action Selection Strategies

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

How should the agent choose?

Two possibilities:

1. Uniform random selection
   - 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.
2. Select action with highest expected cumulative reward
   - 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.

A Practical Compromise:

Choose action randomly but bias choice in favour of those with a higher expected cumulative value.

For example, choose actions with probabilities given by:

$$P(a_i \mid s) = \frac{k^{Q'(s,a_i)}}{\sum_j k^{Q'(s,a_j)}}$$

where $k > 0$ determines the bias to higher value paths.

Improved Updating Methods

Using the updating method described, a path leading to a rewarding transition must be repeatedly traversed to enable the improved Q estimate to move back to the starting of the path.

The improvement can only move back one step for each full traversal of the path.

Solution 1: Remember the whole path

- Remember all transitions until a non-zero reward is achieved.
- Then update the Q estimates for all transitions on the path in reverse order.

This makes more effective use of training thus producing faster convergence on good Q estimates.

It requires additional memory: how much depends on path length.

Solution 2: Remember rewards for all transitions:

Whenever a Q estimate for an action a at state s is changed and a is or becomes the best choice, then the Q estimates of all transitions leading to s need to be revised.

If the agent has remembered all previous transitions and their associated immediate rewards, then the update procedure can be applied at once to all affected transitions.

This saves much exploration but also has a high memory cost.

NONDETERMINISTIC DOMAINS

So far we have only considered domains in which the transition function δ and the reward function r were deterministic.

What happens if they have probabilistic outcomes?

Why can't we just re-write the preceding arguments in terms of expected rather than absolute values and use the same procedure for obtaining Q estimates?

The Problem

- The procedure for updating Q is based on individual traversals of a path.
- These will not produce consistent results even though their expectation values are constant.
- There is no longer any guarantee that all changes will be improvements and the system may not converge.

The Solution

The original update Q procedure replaces the existing value.

Make the updated value depend on both the existing value and the new estimate. This introduces an element of averaging and hence convergence can take place.

$$QE[s,a] := (1 - \alpha)QE[s,a] + \alpha\{r + \gamma \times \max_{a'} QE[s',a']\}$$

α may be a constant or vary as a function of how often the value has been updated.

For more details, see Mitchell pp 381-383.

GENERALIZING

The Problem

All of these variants of Q learning share one core feature: an array storing the Q estimates for every state-action pair in the domain.

This is acceptable if the total number of state-action pairs is reasonably small.

Unfortunately in many real domains both the number of states and the number of actions can be large.

This leads to problems:

- The increased memory load.
- The increased time necessary to explore the whole domain and achieve convergence.

The Solution

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 incorporating some kind of function approximation technique.

An Example:

- Replace the state-action array with a neural network.
- The inputs encode the state and action.
- Train output using backpropagation to the values produce by the Q update procedure.

Suggested Readings:

*Mitchell, T. M., (1997), "*Machine Learning*", McGraw*-Hill. Chapter 13.

*Samuel, A. L., (1959), "*Some studies in machine learning using the game of checkers*", IBM Journal of Research and Development,*3, pp 211-229. (This has been anthologised several times; see for example Readings in Machine Learning, (J. Shavlik & T. Dietterich eds.) Morgan Kaufmann, 1990)

*Watkins, C. & Dayna, P. (1992), "*Q learning*", Machine Learning,* 8, pp 279-292.

*Holland, J. H., (1986), "*Escaping Brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems*". In "Machine Learning: An Artificial Intelligence Approach II", (R. Michalski, J.* Carbonell & T. Mitchell. eds.) Morgan Kaufmann, pp 593-623.

Sutton, R., (1988), "Learning to predict by the methods of temporal differences*". Machine Learning,* 22, pp 9-44.