Reinforcement Learning

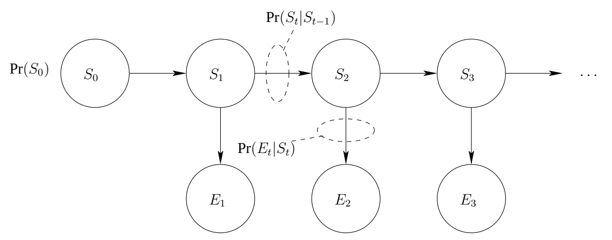
We now examine:

- some potential shortcomings of hidden Markov models, and of supervised learning;
- an extension know as the **Markov Decision Process (MDP)**;
- the way in which we might **learn from rewards** gained as a result of **acting within an environment**;
- specific, simple algorithms for performing such learning, and their convergence properties.

Reading: Russell and Norvig, chapter 21. Mitchell chapter 13.

Reinforcement learning and HMMs

Hidden Markov Models (HMMs) are appropriate when our agent models the world as follows

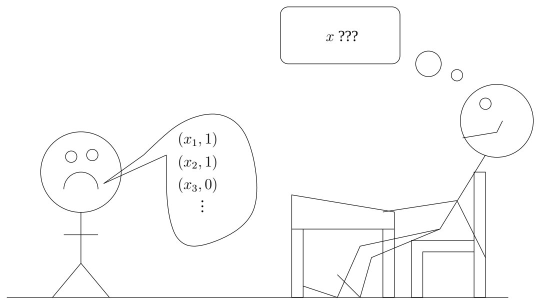


and only wants to infer information about the **state** of the world on the basis of observing the available **evidence**.

This might be criticised as un-necessarily restricted, although it is very effective for the right kind of problem.

Reinforcement learning and supervised learning

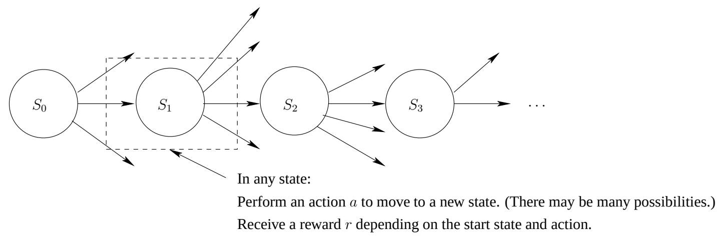
Supervised learners learn from **specifically labelled chunks of information**:



This might also be criticised as un-necessarily restricted: there are other ways to learn.

Reinforcement learning: the basic case

We now begin to model the world in a more realistic way as follows:


S_0
S_1
S_2
S_3
$\cdots$
In any state:
Perform an action a to move to a new state. (There may be many possibilities.)
Receive a reward r depending on the start state and action.

The agent can **perform actions** in order to **change the world's state**.

If the agent performs an action in a particular state, then it **gains a corresponding reward**.

Deterministic Markov Decision Processes

Formally, we have a set of states

$$S = \{s_1, s_2, \dots, s_n\}$$

and in each state we can perform one of a set of actions

$$A = \{a_1, a_2, \dots, a_m\}.$$

We also have a function

$$\mathcal{S} : S \times A \to S$$

such that $\mathcal{S}(s, a)$ is the new state resulting from performing action a in state s, and a function

$$\mathcal{R} : S \times A \to \mathbb{R}$$

such that $\mathcal{R}(s, a)$ is the **reward** obtained by executing action a in state s.

Deterministic Markov Decision Processes

From the point of view of the agent, there is a matter of considerable importance:

$\boxed{\text{The agent does not have access to the functions } \mathcal{S} \text{ and } \mathcal{R}}$.

It therefore has to **learn** a **policy**, which is a function

$$p : S \to A$$

such that $p(s)$ provides the action a that should be executed in state s.

What might the agent use as its criterion for learning a policy?

Measuring the quality of a policy

Say we start in a state at time t, denoted s_t, and we follow a policy p. At each future step in time we get a reward. Denote the rewards r_t, r_{t+1}, ... and so on.

A common measure of the quality of a policy p is the **discounted cumulative reward**

$$\begin{aligned} V^p(s_t) &= \sum_{i=0}^{\infty} \epsilon^i r_{t+i} \\ &= r_t + \epsilon r_{t+1} + \epsilon^2 r_{t+2} + \cdots \end{aligned}$$

where $0 \leq \epsilon \leq 1$ is a constant, which defines a trade-off for how much we value immediate rewards against future rewards.

The intuition for this measure is that, on the whole, we should like our agent to prefer rewards gained quickly.

Measuring the quality of a policy

Other common measures are the **average reward**

$$\lim_{T \to \infty} \frac{1}{T} \sum_{i=0}^{T} r_{t+i}$$

and the **finite horizon reward**

$$\sum_{i=0}^{T} r_{t+i}$$

In these notes we will only address the discounted cumulative reward.

Two important issues

Note that in this kind of problem we need to address two particularly relevant issues:

- The **temporal credit assignment** problem: that is, how do we decide which specific actions are important in obtaining a reward?
- The **exploration/exploitation** problem. How do we decide between **exploiting** the knowledge we already have, and **exploring** the environment in order to possibly obtain new (and more useful) knowledge?

We will see later how to deal with these.

The optimal policy

Ultimately, our learner's aim is to learn the **optimal policy**

$$p_{\text{opt}} = \operatorname*{argmax}_p V^p(s)$$

for all s. We will denote the optimal discounted cumulative reward as

$$V_{\text{opt}}(s) = V^{p_{\text{opt}}}(s).$$

How might we go about learning the optimal policy?

Learning the optimal policy

The only information we have during learning is the individual rewards obtained from the environment.

We could try to learn $V_{\text{opt}}(s)$ directly, so that states can be compared:

Consider s as better than s' if $V_{\text{opt}}(s) > V_{\text{opt}}(s')$.

However we actually want to compare **actions**, not **states**. Learning $V_{\text{opt}}(s)$ might help as

$$p_{\text{opt}}(s) = \operatorname*{argmax}_a \left[\mathcal{R}(s,a) + \epsilon V_{\text{opt}}(\mathcal{S}(s,a))\right]$$

but **only if we know** $\mathcal{S}$ and $\mathcal{R}$.

As we are interested in the case where these functions are **not** known, we need something slightly different.

The $\mathcal{Q}$ function

The trick is to define the following function:

$$\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \epsilon V_{\text{opt}}(\mathcal{S}(s,a))$$

This function specifies the discounted cumulative reward obtained if you do action a in state s **and then follow the optimal policy**.

As

$$p_{\text{opt}}(s) = \operatorname*{argmax}_a \mathcal{Q}(s,a)$$

then provided one can learn $\mathcal{Q}$ **it is not necessary to have knowledge of $\mathcal{S}$ and $\mathcal{R}$ to obtain the optimal policy**.

The $\mathcal{Q}$ function

Note also that
$$V_{\text{opt}}(s) = \max_{\alpha} \mathcal{Q}(s,\alpha)$$
and so
$$\boxed{\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \epsilon \max_{\alpha} \mathcal{Q}(\mathcal{S}(s,a),\alpha)}$$
which suggests a simple learning algorithm.

Let Q' be our learner's estimate of what the exact $\mathcal{Q}$ function is.

That is, in the current scenario Q' is a table containing the estimated values of $\mathcal{Q}(s,a)$ for all pairs (s,a).

$\mathcal{Q}$-learning

Start with all entries in Q' set to 0. (In fact we will see in a moment that random entries will do.)

Repeat the following:

1. Look at the current state s and choose an action a. (We will see how to do this in a moment.)
2. Do the action a and obtain some reward $\mathcal{R}(s,a)$.
3. Observe the new state $\mathcal{S}(s,a)$.
4. Perform the update
$$Q'(s,a) = \mathcal{R}(s,a) + \epsilon \max_{\alpha} Q'(\mathcal{S}(s,a),\alpha)$$

Note that this can be done in **episodes**. For example, in learning to play games, we can play multiple games, each being a single episode.

Convergence of $\mathcal{Q}$-learning

This looks as though it might converge!

Note that, if the rewards are at least 0 and we initialise Q' to 0 then,
$$\forall n, s, a \; Q'_{n+1}(s,a) \geq Q'_n(s,a)$$
and
$$\forall n, s, a \; Q(s,a) \geq Q'_n(s,a) \geq 0$$
However, we need to be a bit more rigorous than this...

Convergence of $\mathcal{Q}$-learning

If:

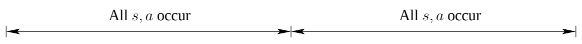
1. the agent is operating in an environment that is a deterministic MDP;
2. rewards are bounded in the sense that there is a constant $\delta > 0$ such that
$$\forall s, a \; |\mathcal{R}(s,a)| < \delta$$
3. all possible pairs s and a are visited infinitely often;

then the $\mathcal{Q}$-learning algorithm converges, in the sense that
$$\forall a, s \; Q'_n(s,a) \to \mathcal{Q}(s,a)$$
as $n \to \infty$.

Convergence of $\mathcal{Q}$-learning

This is straightforward to demonstrate.

Using condition 3, take two stretches of time in which all s and a pairs occur:



Define

$$\xi(n) = \max_{s,a} |Q'_n(s,a) - \mathcal{Q}(s,a)|$$

the maximum error in Q' at n.

What happens when $Q'_n(s,a)$ is updated to $Q'_{n+1}(s,a)$?

Convergence of $\mathcal{Q}$-learning

We have,

$$\begin{aligned}
&|Q'_{n+1}(s,a) - \mathcal{Q}(s,a)| \\
&\quad = |(\mathcal{R}(s,a) - \epsilon \max_{\alpha} Q'_n(\mathcal{S}(s,a),\alpha)) - (\mathcal{R}(s,a) - \epsilon \max_{\alpha} \mathcal{Q}(\mathcal{S}(s,a),\alpha))| \\
&\quad = \epsilon | \max_{\alpha} Q'_n(\mathcal{S}(s,a),\alpha) - \max_{\alpha} \mathcal{Q}(\mathcal{S}(s,a),\alpha)| \\
&\quad \leq \epsilon \max_{\alpha} |Q'_n(\mathcal{S}(s,a),\alpha) - \mathcal{Q}(\mathcal{S}(s,a),\alpha)| \\
&\quad \leq \epsilon \max_{s,a} |Q'_n(s,a) - \mathcal{Q}(s,a)| \\
&\quad = \epsilon \xi(n).
\end{aligned}$$

Convergence as described follows.

Choosing actions to perform

We have not yet answered the question of how to choose actions to perform during learning.

One approach is to choose actions based on our current estimate Q'. For instance

$$\text{action chosen in current state } s = \operatorname*{argmax}_a Q'(s,a).$$

However we have already noted the trade-off between exploration and exploitation. It makes more sense to:

- **explore** during the early stages of training;
- **exploit** during the later stages of training.

This seems particularly important in the light of condition 3 of the convergence proof.

Choosing actions to perform

One way in which to choose actions that incorporates these requirements is to introduce a constant λ and choose actions **probabilistically** according to

$$\text{Pr}(\text{action } a | \text{state } s) = \frac{\lambda^{Q'(s,a)}}{\sum_a \lambda^{Q'(s,a)}}$$

Note that:

- if λ is **small** this promotes **exploration**;
- if λ is **large** this promotes **exploitation**.

We can vary λ as training progresses.

Improving the training process

There are two simple ways in which the process can be improved:

1. If training is episodic, we can store the rewards obtained during an episode and update **backwards** at the end.

   This allows better updating at the expense of requiring more memory.

2. We can remember information about rewards and occasionally **re-use** it by re-training.

Nondeterministic MDPs

The $\mathcal{Q}$-learning algorithm generalises easily to a more realistic situation, where the outcomes of actions are **probabilistic**.

Instead of the functions $\mathcal{S}$ and $\mathcal{R}$ we have **probability distributions**

$$\text{Pr}(\text{new state}|\text{current state}, \text{action})$$

and

$$\text{Pr}(\text{reward}|\text{current state}, \text{action}).$$

and we now use $\mathcal{S}(s,a)$ and $\mathcal{R}(s,a)$ to denote the corresponding random variables.

We now have

$$V^p = \mathbb{E}\left(\sum_{i=0}^{\infty} \epsilon^i r_{t+i}\right)$$

and the best policy p_{opt} maximises V^p.

$\mathcal{Q}$-learning for nondeterministic MDPs

We now have

$$\begin{aligned}\mathcal{Q}(s,a) &= \mathbb{E}(\mathcal{R}(s,a)) + \epsilon \sum_{\sigma} \text{Pr}(\sigma|s,a) V^{\text{opt}}(\sigma) \\ &= \mathbb{E}(\mathcal{R}(s,a)) + \epsilon \sum_{\sigma} \text{Pr}(\sigma|s,a) \max_{\alpha} \mathcal{Q}(\sigma,\alpha)\end{aligned}$$

and the rule for learning becomes

$$\boxed{Q'_{n+1} = (1-\theta_{n+1})Q'_n(s,a) + \theta_{n+1}\left[\mathcal{R}(s,a) + \max_{\alpha} Q'_n(\mathcal{S}(s,a),\alpha)\right]}$$

with

$$\boxed{\theta_{n+1} = \frac{1}{1+v_{n+1}(s,a)}}$$

where $v_{n+1}(s,a)$ is the number of times the pair s and a has been visited so far.

Convergence of $\mathcal{Q}$-learning for nondeterministic MDPs

If:

1. the agent is operating in an environment that is a nondeterministic MDP;
2. rewards are bounded in the sense that there is a constant $\delta > 0$ such that

   $$\forall s, a\ |\mathcal{R}(s,a)| < \delta$$

3. all possible pairs s and a are visited infinitely often;
4. $n_i(s,a)$ is the ith time that we do action a in state s;

and also...

Convergence of $\mathcal{Q}$-learning for nondeterministic MDPs

...we have

$$0 \leq \theta_n < 1$$
$$\sum_{i=1}^{\infty} \theta_{n_i(s,a)} = \infty$$
$$\sum_{i=1}^{\infty} \theta^2_{n_i(s,a)} < \infty$$

then with probability 1 the $\mathcal{Q}$-learning algorithm converges, in the sense that

$$\forall a, s \; Q'_n(s,a) \rightarrow \mathcal{Q}(s,a)$$

as $n \rightarrow \infty$.

Alternative representation for the Q' table

But there's always a catch...

We have to store the table for Q':

- even for quite straightforward problems it is HUGE!!! - certainly big enough that it can't be stored;
- a standard approach to this problem is, for example, to represent it as a **neural network**;
- one way might be to make s and a the inputs to the network and train it to produce $Q'(s,a)$ as its output.

This, of course, introduces its own problems, although it has been used very successfully in practice.

It might be covered in **Artificial Intelligence III**, which unfortunately does not yet exist!