Two Cognitive Descriptions of Q-Learning[1]

Terrence C. Stewart, Sanjay Chandrasekharan
Institute of Cognitive Science
Carleton University,
Ottawa, Canada, K1S 2S3
tcstewar@connect.carleton.ca, schandra@sce.carleton.ca

We provide two descriptions of the Q-Learning algorithm (Watkins, 1989), one high-level and the other at the mechanism-level, to support the use of the algorithm within cognitive modeling frameworks.

High-level Description

The Q-Learning algorithm is a probabilistic learning rule that maps states in the world [s] to possible actions [a]. For an upcoming action, the agent starts with an expected reward R, but since it doesn't know what that reward is, every encounter with the world [s, a] is given a quality value Q, which is some function of previous rewards. For a given state, the agent decides an action based on this Q value, which is an approximated 'projection' of future reward, based on previous values from experience. What makes Q-learning different is that this projection is not calculated by explicitly running possible action chains for every state, and compiling their rewards. It is estimated using a function (the Q function) learned in real-time, derived from previously executed actions, where every action in the world is considered a 'test' action. Once derived, the use of this function can be considered as *implicitly running* possible future actions, across time.

The algorithm works by learning, in real-time (i.e. while negotiating the world), a function that provides a projected estimate of the eventual outcome of performing an action. It does not develop this estimate by looking ahead and calculating possible actions, states and rewards, but by learning a function that approximates this.

1 Carleton University Cognitive Science Technical Report 2005-03. http://www.carleton.ca/iis/TechReports

One way to think of Q-Learning is to think of ‘pretend play’ by chess novices, where they ‘try out’ moves in the world. Such actions are termed epistemic actions (Kirsh, 1994). The organism ‘tests’ the environment with individual actions to see what reward that particular environment provides for that particular action. An example of this from the animal world is predators who make a “test attack” on herds to identify animals whose ability to run away is insufficient to protect them (Curio, 1976). In such cases, the actions in the world are not ‘real’, but ‘tests’, or ‘simulated’ actions. And the organism uses itself and the environment as a ‘test-bed’ or ‘simulation environment’ to judge the quality of its own actions.

Like the chess player, the Q-learning algorithm only 'simulates’ one step ahead, but as it simulates one step ahead, its evaluation of how good that step is includes the whole future set of actions, because the Q-function approximates the possible outcome of an entire range of state-action combinations. Think of a chess player who tries out the move of taking a knight with his queen, and then looks at the new board position and gets the feeling of 'that looks dangerous – I better not do that'.

The Q function can be thought of as developing an estimate of the reward structure of ‘perturbations’ in the agent-environment system, instead of developing an estimate of rewards for a single action. This means it can look ahead (i.e. test run) only one step, but the output of that test-run provides an estimate of how the system as a whole would evolve many steps into the future, and the reward structure then. Once the Q function is developed, it looks ahead only one step, but it can be considered to implicitly run many states ahead. This implicit running process can be considered similar to simulating the evolution of the system across time. So Q-Learning can be considered to implement a form of ‘simulation’.

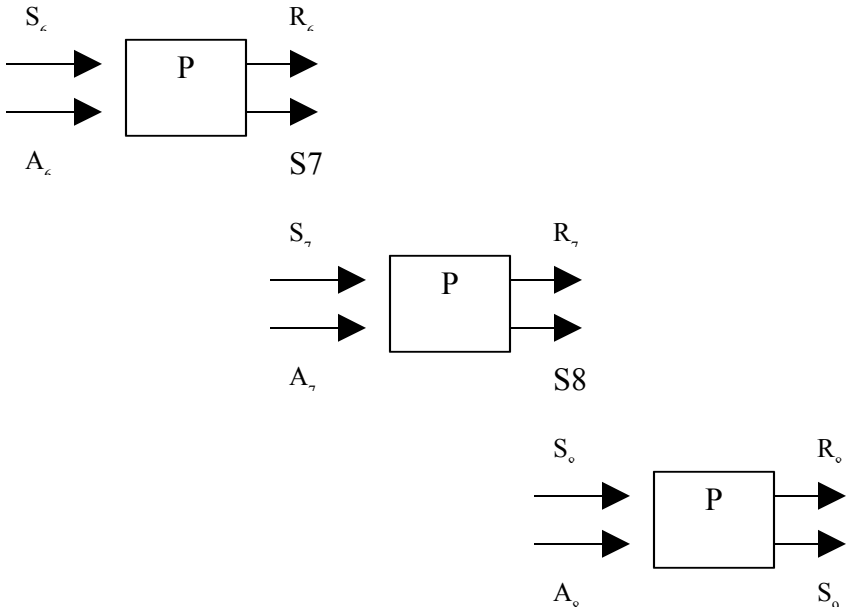
Mechanism-level Description

Consider the following matrix of world states (S), actions (A) and rewards (R). For world state S, the agent takes action A, and gets a reward R.

States	S1	S2	S3	S4	S5	…	…	…
Actions	A1	A2	A3	A4	A5	…	…	…
Rewards	R1	R2	R3	R4	R5	…	…	…

Now, given this experience set, the agent could develop a function P during the training phase, such that P provides an estimate of reward R for a given set [S, A].

At state S6, the agent has to estimate which action to take, so that its overall reward is optimal. One way to do this would be to look ahead by "testing out" each possible action, the state that results from that action, the best possible action for that state etc. Graphically,



Once this lookup is done, the agent has to collate all these expected rewards. We can define the total expected reward for A6 at S6 as:

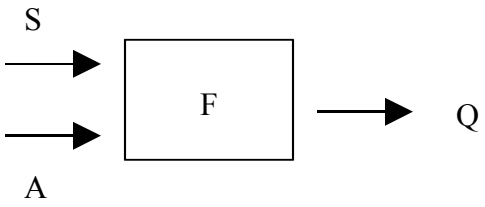
$$Q = R_5 + 0.9R_6 + 0.9^2R_7 + 0.9^3R_8 + \ldots\ldots \qquad Eq1$$

(Note that the value 0.9 can be changed to adjust how quickly the algorithm discounts future rewards.)

But this process leads to a combinatorial explosion very quickly, as there are many possible actions that can be taken, and so cannot be used at run-time to develop estimates. Q-Learning works by developing an optimal function that provides an estimate of the output of this extended lookup. This is done using an expected reward value Q, where Q is some function of S and A.

F (S, A) → Q

Graphically:



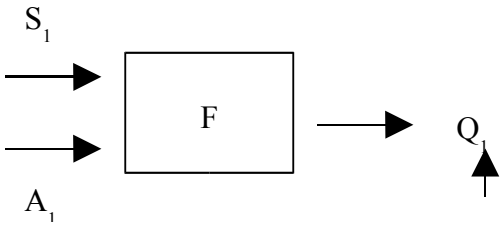
Generalising from *Eq1*,

$Q_1 = R_1 + 0.9R_2 + 0.9^2R_3 + 0.9^3R_4 + \ldots$ *Eq2*

$Q_2 = R_2 + 0.9R_3 + 0.9^2R_4 + \ldots$ *Eq3*

From this, $Q_1 = R_1 + 0.9Q_2$ *Eq4*

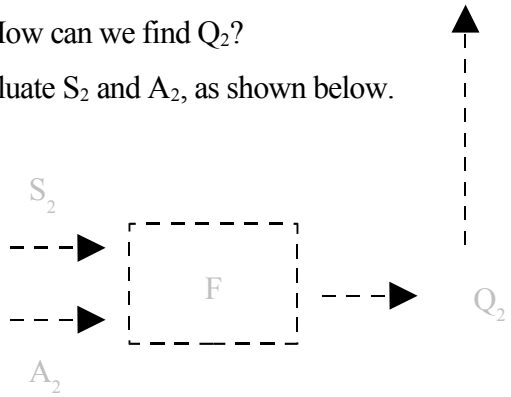
Putting this in our graphic above:



$Q_1 = R_1 + 0.9Q_2$

But to do this substitution, we need to know Q_2. How can we find Q_2?

This can be done by using the same system to evaluate S_2 and A_2, as shown below.



Essentially, during the training phase, the Q function calls itself recursively to estimate Q_2. Initially, this estimate would not be optimal, but it converges to the optimal as training progresses. Even though this estimation of Q_2 is based on looking up one step ahead, it involves an *implicit running* of many actions ahead.

One way to think of the optimal Q function is to think of it as developing an estimate of the reward structure of 'perturbations' in the agent-environment system and how they propagate, instead of developing an estimate of rewards for a single action. This means it can look ahead (i.e. test run) only one step (like in the chess-player example), but the output of that test-run provides an estimate of how the system as a whole would evolve many steps into the future, and the reward structure then.

In most implementations, there is no specific training phase, the algorithm learns from its initial exposure to the environment, the initial encounters with the environment are treated as the training phase. Once the Q function is developed, it looks ahead only one step, but it can be considered to *implicitly run* many states ahead. This implicit running process can be considered similar to simulating the evolution of the system across time. This is why we argue that Q-Learning can be considered a form of simulation.

It should also be noted that the algorithm does not depend on how the function F is implemented. It may be a simple memorization-based look-up-table (implemented by remembering the results of all situations the agent has been in). For more flexibility, however, it is often implemented using a neural network. However, any algorithm capable of learning from example to map one set of values to another could be used.

Furthermore, while the above discussion deals specifically with Q-Learning, a similar argument can be made for any form of TD-Learning. This is a relatively large set of Reinforcement Learning models, which have been well studied be computer scientists. For an excellent overview, see (Sutton & Barto, 1998).

References

Curio, E. (1976). *The ethology of predation*. Springer Verlag, New York.

Kirsh, D., & Maglio, P. (1994). On distinguishing epistemic from pragmatic action. *Cognitive Science*. 18, 513-549.

Sutton, R.S. & Barto, A. (1996). Reinforcement Learning. Cambridge, MA: MIT Press.

Watkins, C. (1989). *Learning From Delayed Rewards*, Doctoral dissertation, Department of Psychology, University of Cambridge, Cambridge, UK.