Q-learning

Q-learning is a model-free reinforcement learning technique. Specifically, *Q*-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP). It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. A policy is a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. One of the strengths of *Q*-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment. Additionally, *Q*-learning can handle problems with stochastic transitions and rewards, without requiring any adaptations. It has been proven that for any finite MDP, *Q*-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable.[1]

Contents

Algorithm

The problem model consists of an agent, states S and a set of actions per state A. By performing an action $a \in A$, the agent can move from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score). The goal of the agent is to maximize its total reward. It does this by learning which action is optimal for each state. The action that is optimal for each state is the action that has the highest long-term reward. This reward is a weighted sum of the expected values of the rewards of all future steps starting from the current state, where the weight for a step from a state Δt steps into the future is calculated as $\gamma^{\Delta t}$. Here, γ is a number between 0 and 1 ($0 \leq \gamma \leq 1$) called the discount factor and trades off the importance of sooner versus later rewards. γ may also be interpreted as the likelihood to succeed (or survive) at every step Δt.

The algorithm therefore has a function that calculates the Quantity of a state-action combination:

$$Q : S \times A \to \mathbb{R}\,.$$

Before learning has started, Q returns an (arbitrary) fixed value, chosen by the designer. Then, at each time t the agent selects an action a_t and observes a reward r_t and a new state s_{t+1} that may depend on both the previous state s_t and the selected action, Q is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right),$$

where r_t is the reward observed for the current state s_t, and α is the learning rate ($0 < \alpha \leq 1$).

An episode of the algorithm ends when state s_{t+1} is a final state (or, "absorbing state"). However, *Q*-learning can also learn in non-episodic tasks. If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loops.

Note that for all final states s_f, $Q(s_f, a)$ is never updated but is set to the reward value r observed for state s_f. In most cases, $Q(s_f, a)$ can be taken to be equal to zero.

Influence of variables on the algorithm

Learning rate

The learning rate or *step size* determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. In fully deterministic environments, a learning rate of $\alpha_t = 1$ is optimal. When the problem is stochastic, the algorithm still converges under some technical conditions on the learning rate that require it to decrease to zero. In practice, often a constant learning rate is used, such as $\alpha_t = 0.1$ for all t.[2]

Discount factor

The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the action values may diverge. For $\gamma = 1$, without a terminal state, or if the agent never reaches one, all environment histories will be infinitely long, and utilities with additive, undiscounted rewards

will generally be infinite.[3] Even with a discount factor only slightly lower than 1, the *Q*-function learning leads to propagation of errors and instabilities when the value function is approximated with an artificial neural network.[4] In that case, it is known that starting with a lower discount factor and increasing it towards its final value yields accelerated learning.[5]

Initial conditions (Q_0)

Since *Q*-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values, also known as "optimistic initial conditions",[6] can encourage exploration: no matter what action is selected, the update rule will cause it to have lower values than the other alternative, thus increasing their choice probability. Recently, it was suggested that the first reward r could be used to reset the initial conditions. According to this idea, the first time an action is taken the reward is used to set the value of Q. This will allow immediate learning in case of fixed deterministic rewards. Surprisingly, this resetting-of-initial-conditions (RIC) approach seems to be consistent with human behaviour in repeated binary choice experiments.[7]

Implementation

Q-learning at its simplest uses tables to store data. This very quickly loses viability with increasing sizes of state/action space of the system it is monitoring/controlling. One answer to this problem is to use an (adapted) artificial neural network as a function approximator, as demonstrated by Tesauro in his Backgammon playing temporal difference learning research.[8]

More generally, *Q*-learning can be combined with function approximation.[9] This makes it possible to apply the algorithm to larger problems, even when the state space is continuous, and therefore infinitely large. Additionally, it may speed up learning in finite problems, due to the fact that the algorithm can generalize earlier experiences to previously unseen states.

Early study

Q-learning was first introduced by Watkins[10] in 1989. The convergence proof was presented later by Watkins and Dayan[11] in 1992.

The problem Watkins was solving was named “Learning from delayed rewards” which was the title of his PhD Thesis.. Eight years earlier in 1981 the same problem under the name of “Delayed reinforcement learning” was solved by a system named Crossbar Adaptive Array (CAA)[12] Initial published report was given in 1982[13] The memory matrix W(a,s) of the presented CAA architecture is exactly the same as the Q-table of Q-learning. The architecture shown in a Figure introduced the term “state evaluation” in reinforcement learning research. The crossbar learning algorithm, written in mathematical pseudocode in the paper, in each iteration performs the following computation: 1) in state s perform action a; 2) receive consequence state s’; 3) compute state evaluation v(s’); 4) update crossbar value W’(a,s) = W(a,s) + v(s’). The term “secondary reinforcement” is borrowed from animal learning theory, to model state values backpropagation: the state value v(s’) of the consequence situation is backpropagated to the previously encountered situation s. In a crossbar fashion, CAA computes state values vertically and actions horizontally. Demonstration graphs showing delayed reinforcement learning contained states represented by emoticons (desirable, undesirable, and neutral states), which were computed by state evaluation function. This learning system in 1997 was recognized as a forerunner of the Q-learning algorithm[14].

Variants

A recent application of *Q*-learning to deep learning, by Google DeepMind, titled "deep reinforcement learning" or "deep *Q*-networks", has been successful at playing some Atari 2600 games at expert human levels. Preliminary results were presented in 2014, with a paper published in February 2015 in Nature.[15]

Because the maximum approximated action value is used in the *Q*-learning update, in noisy environments *Q*-learning can sometimes overestimate the actions values, slowing the learning. A recent variant called Double *Q*-learning was proposed to correct this. [16] This algorithm was later combined with deep learning, as in the DQN algorithm (see above), resulting in Double DQN which was shown to outperform the original DQN algorithm. [17]

Delayed *Q*-learning is an alternative implementation of the online *Q*-learning algorithm, with Probably approximately correct learning (PAC).[18]

Greedy GQ is a variant of *Q*-learning to use in combination with (linear) function approximation.[19] The advantage of Greedy GQ is that convergence guarantees can be given even when function approximation is used to estimate the action values.

Q-learning may suffer from slow rate of convergence, especially when the discount factor γ is close to one.[20] Speedy *Q*-learning, a new variant of *Q*-learning algorithm, deals with this problem and achieves a slightly better rate of convergence than model-based methods such as value iteration.[21]

See also

- Reinforcement learning
- Temporal difference learning
- SARSA
- Iterated prisoner's dilemma
- Game theory
- Fitted Q iteration algorithm

References

1. Francisco S. Melo, "Convergence of Q-learning: a simple proof" (http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf)
2. Reinforcement Learning: An Introduction (http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html). Richard Sutton and Andrew Barto. MIT Press, 1998.
3. Stuart J. Russell; Peter Norvig (2010). *Artificial Intelligence: A Modern Approach* (Third ed.). Prentice Hall. p. 649. ISBN 978-0136042594.
4. Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. ICML, pages 30–37, 1995
5. François-Lavet Vincent, Raphael Fonteneau, Damien Ernst. "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies" (http://arxiv.org/abs/1512.02011). NIPS, Deep RL workshop 2015.
6. http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node21.html
7. Shteingart, H; Neiman, T; Loewenstein, Y (May 2013). "The Role of First Impression in Operant Learning". *J Exp Psychol Gen*. **142** (2): 476–88. PMID 22924882 (https://www.ncbi.nlm.nih.gov/pubmed/22924882). doi:10.1037/a0029550 (https://doi.org/10.1037%2Fa0029550).

8. Tesauro, Gerald (March 1995). "Temporal Difference Learning and TD-Gammon" (http://www.bkgm.com/articles/tesauro/tdl.html). *Communications of the ACM*. **38** (3). doi:10.1145/203330.203343 (https://doi.org/10.1145%2F203330.203343). Retrieved 2010-02-08.
9. Hado van Hasselt. Reinforcement Learning in Continuous State and Action Spaces. In: Reinforcement Learning: State of the Art, Springer, pages 207-251, 2012
10. Watkins, C.J.C.H., (1989), Learning from Delayed Rewards. Ph.D. thesis, Cambridge University.
11. Watkins and Dayan, C.J.C.H., (1992), 'Q-learning.Machine Learning'
12. S. Bozinovski, Crossbar Adaptive Array: The first connectionist network that solved the delayed reinforcement learning problem, In A. Dobnikar, N. Steele, D. Pearson, R. Albert (Eds.) Artificial Neural Networks and Genetic Algorithms, 320-325, Springer Verlag, 1999
13. S. Bozinovski, A self learning system using secondary reinforcement, In R. Trappl (ed.) Cybernetics and Systems Research, p. 397-402, North Holland,1982
14. A. Barto, Reinforcement learning, in O. Omidvar, D. Elliot (eds.) Neural Systems for Control, p.7-30, Academic Press, 1997
15. Mnih, Volodymyr; et al. (2015). "Human-level control through deep reinforcement learning" (http://www.nature.com/nature/journal/v518/n7540/pdf/nature14236.pdf) (PDF). **518**: 529–533.
16. van Hasselt, Hado (2011). "Double Q-learning" (http://books.nips.cc/papers/files/nips23/NIPS2010_0208.pdf) (PDF). *Advances in Neural Information Processing Systems*. **23**: 2613–2622.
17. van Hasselt, Hado; Guez, Arthur; Silver, David (2015). "Deep reinforcement learning with double Q-learning" (http://www.aaai.org/Conferences/AAAI/2016/Papers/12vanHasselt12389.pdf) (PDF). *AAAI Conference on Artificial Intelligence*: 2094–2100.
18. Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. Pac model-free reinforcement learning (http://research.microsoft.com/pubs/178886/published.pdf). In Proc. 22nd ICML 2006, pages 881–888, 2006.
19. Hamid Maei, and Csaba Szepesvári, Shalabh Bhatnagar and Richard Sutton. Toward off-policy learning control with function approximation (https://webdocs.cs.ualberta.ca/~sutton/papers/MSBS-10.pdf). In proceedings of the 27th International Conference on Machine Learning, pages 719-726, 2010.
20. Csaba Szepesva ́ri. The asymptotic convergence-rate of *Q*-learning. *Advances in Neural Information Processing Systems* 10, Denver, Colorado, USA, 1997.
21. Gheshlaghi Azar, Mohammad; Munos, Remi; Ghavamzadeh, Mohammad; Kappen, Hilbert J. (2011). "Speedy Q-Learning" (http://books.nips.cc/papers/files/nips24/NIPS2011_1278.pdf) (PDF). *Advances in Neural Information Processing Systems*. **24**: 2411–2419.

External links

- Watkins, C.J.C.H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England. (http://www.cs.rhul.ac.uk/~chrisw/thesis.html)
- Strehl, Li, Wiewiora, Langford, Littman (2006). PAC model-free reinforcement learning (http://portal.acm.org/citation.cfm?id=1143955)
- *Reinforcement Learning: An Introduction* (https://web.archive.org/web/20050806080008/http://www.cs.ualberta.ca/~sutton/book/the-book.html) by Richard Sutton and Andrew S. Barto, an online textbook. See "6.5 Q-Learning: Off-Policy TD Control" (http://www.cs.ualberta.ca/~sutton/book/ebook/node65.html).
- Piqle: a Generic Java Platform for Reinforcement Learning (http://sourceforge.net/projects/piqle/)
- Reinforcement Learning Maze (http://ccl.northwestern.edu/netlogo/models/community/Reinforcement%20Learning%20Maze), a demonstration of guiding an ant through a maze using *Q*-learning.
- *Q*-learning work by Gerald Tesauro (http://www.research.ibm.com/infoecon/paps/html/ijcai99_qnn/node4.html)
- *Q*-learning work by Tesauro Citeseer Link (https://web.archive.org/web/20080529074412/http://citeseer.comp.nus.edu.sg/352693.html)
- *Q*-learning algorithm implemented in processing.org language (https://github.com/sandropaganotti/processing.org-q-learning-td-lambda-/tree/master)

- Solution for the pole balancing problem with Q(lambda) / SARSA(lambda) and the fourier basis in javascript (https://web.archive.org/web/20150131172946/http://toki78.github.io/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Q-learning&oldid=804687897"
