Q-learning

***Q*-learning** is a model-free reinforcement learning algorithm. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model (hence the connotation "model-free") of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.

For any finite Markov decision process (FMDP), *Q*-learning finds a policy that is optimal in the sense that it maximizes the expected value of the total reward over any and all successive steps, starting from the current state.[1] *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy.[1] "Q" names the function that returns the reward used to provide the reinforcement and can be said to stand for the "quality" of an action taken in a given state.[2]

Contents

Reinforcement learning

Reinforcement learning involves an agent, a set of *states* $\boldsymbol{S}$, and a set $\boldsymbol{A}$ of *actions* per state. By performing an action $\boldsymbol{a} \in \boldsymbol{A}$, the agent transitions 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 (future) reward. It does this by adding the maximum reward attainable from future states to the reward for achieving its current state, effectively influencing the current action by the potential future reward. This potential reward is a weighted sum of the expected values of the rewards of all future steps starting from the current state.

As an example, consider the process of boarding a train, in which the reward is measured by the negative of the total time spent boarding (alternatively, the cost of boarding the train is equal to the boarding time). One strategy is to enter the train door as soon as they open, minimizing the initial wait time for yourself. If the train is crowded, however, then you will have a slow entry after the initial action of entering the door as people are fighting you to depart the train as you attempt to board. The total boarding time, or cost, is then:

- 0 seconds wait time + 15 seconds fight time

On the next day, by random chance (exploration), you decide to wait and let other people depart first. This initially results in a longer wait time. However, time fighting other passengers is less. Overall, this path has a higher reward than that of the previous day, since the total boarding time is now:

- 5 second wait time + 0 second fight time.

Through exploration, despite the initial (patient) action resulting in a larger cost (or negative reward) than in the forceful strategy, the overall cost is lower, thus revealing a more rewarding strategy.

Algorithm

The weight for a step from a state Δt steps into the future is calculated as $\gamma^{\Delta t}$, where γ (the *discount factor*) is a number between 0 and 1 ($0 \leq \gamma \leq 1$) and has the effect of valuing rewards received earlier higher than those received later (reflecting the value of a "good start"). γ may also be interpreted as the probability to succeed (or survive) at every step Δt.

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

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

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

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	327	0	0	0	0	0	0
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Q-Learning table of states by actions that is initialized to zero, then each cell is updated through training.

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

where r_t is the reward received when moving from the state s_t to the state s_{t+1}, and α is the learning rate ($0 < \alpha \leq 1$).

An episode of the algorithm ends when state s_{t+1} is a final or *terminal 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.

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 equal zero.

Influence of variables

Learning Rate

The learning rate or *step size* determines to what extent newly acquired information overrides old information. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities). In fully deterministic environments, a learning rate of $\alpha_t = 1$ is optimal. When the problem is stochastic, the algorithm 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.[3]

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, i.e. r_t (in the update rule above), 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 become infinitely long, and utilities with additive, undiscounted rewards generally become infinite.[4] Even with a discount factor only slightly lower than 1, *Q*-function learning leads to propagation of errors and instabilities when the value function is approximated with an artificial neural network.[5] In that case, starting with a lower discount factor and increasing it towards its final value accelerates learning.[6]

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",[7] 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. The first reward r can be used to reset the initial conditions.[8] According to this idea, the first time an action is taken the reward is used to set the value of Q. This allows immediate learning in case of fixed deterministic rewards. A model that incorporates *reset of initial conditions* (RIC) is expected to predict participants' behavior better than a model that assumes any *arbitrary initial condition* (AIC).[8] RIC seems to be consistent with human behaviour in repeated binary choice experiments.[8]

Implementation

Q-learning at its simplest stores data in tables. This approach falters with increasing numbers of states/actions since the likelihood of the agent visiting a particular state and performing a particular action is increasingly small.

Function approximation

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.

One solution is to use an (adapted) artificial neural network as a function approximator.[10] Function approximation may speed up learning in finite problems, due to the fact that the algorithm can generalize earlier experiences to previously unseen states.

Quantization

Another technique to decrease the state/action space quantizes possible values. Consider the example of learning to balance a stick on a finger. To describe a state at a certain point in time involves the position of the finger in space, its velocity, the angle of the stick and the angular velocity of the stick. This yields a four-element vector that describes one state, i.e. a snapshot of one state encoded into four values. The problem is that infinitely many possible states are present. To shrink the possible space of valid actions multiple values can be assigned to a bucket. The exact distance of the finger from its starting position (-Infinity to Infinity) is not known, but rather whether it is far away or not (Near, Far).

History

Q-learning was introduced by Chris Watkins[11] in 1989. A convergence proof was presented by Watkins and Dayan[12] in 1992.

Watkins was addressing “Learning from delayed rewards”, the title of his PhD Thesis. Eight years earlier in 1981 the same problem, under the name of “Delayed reinforcement learning”, was solved by Bozinovski's Crossbar Adaptive Array (CAA).[13][14] The memory matrix W =||w(a,s)|| was the same as the eight years later Q-table of Q-learning. The architecture introduced the term “state evaluation” in reinforcement learning. The crossbar learning algorithm, written in mathematical pseudocode in the paper, in each iteration performs the following computation:

- In state s perform action a;
- Receive consequence state s’;
- Compute state evaluation v(s’);
- 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 via backpropagation: the state value v(s’) of the consequence situation is backpropagated to the previously encountered situations. CAA computes state values vertically and actions horizontally (the "crossbar"). Demonstration graphs showing delayed reinforcement learning contained states (desirable, undesirable, and neutral states), which were computed by the state evaluation function. This learning system was a forerunner of the Q-learning algorithm.[15]

In 2014 Google DeepMind patented[16] an application of Q-learning to deep learning, titled "deep reinforcement learning" or "deep Q-learning" that can play Atari 2600 games at expert human levels.

Variants

Deep Q-learning

The DeepMind system used a deep convolutional neural network, with layers of tiled convolutional filters to mimic the effects of receptive fields. Reinforcement learning is unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and the data distribution, and the correlations between Q and the target values.

The technique used *experience replay,* a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent action to proceed.[2] This removes correlations in the observation sequence and smooths changes in the data distribution. Iterative updates adjust Q towards target values that are only periodically updated, further reducing correlations with the target.[17]

Double Q-learning

Because the future maximum approximated action value in Q-learning is evaluated using the same Q function as in current action selection policy, in noisy environments Q-learning can sometimes overestimate the action values, slowing the learning. A variant called Double Q-learning was proposed to correct this. Double Q-learning[18] is an off-policy reinforcement learning algorithm, where a different policy is used for value evaluation than what is used to select the next action.

In practice, two separate value functions are trained in a mutually symmetric fashion using separate experiences, Q^A and Q^B. The double Q-learning update step is then as follows:

$$Q_{t+1}^A(s_t, a_t) = Q_t^A(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q_t^B\left(s_{t+1}, \arg\max_a Q_t^A(s_{t+1}, a)\right) - Q_t^A(s_t, a_t)\right),$$

and

$$Q_{t+1}^B(s_t, a_t) = Q_t^B(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q_t^A\left(s_{t+1}, \arg\max_a Q_t^B(s_{t+1}, a)\right) - Q_t^B(s_t, a_t)\right).$$

Now the estimated value of the discounted future is evaluated using a different policy, which solves the overestimation issue.

This algorithm was later modified in 2015 and combined with deep learning, as in the DQN algorithm, resulting in Double DQN, which outperforms the original DQN algorithm.[19]

Others

Delayed Q-learning is an alternative implementation of the online *Q*-learning algorithm, with probably approximately correct (PAC) learning.[20]

Greedy GQ is a variant of *Q*-learning to use in combination with (linear) function approximation.[21] The advantage of Greedy GQ is that convergence is guaranteed even when function approximation is used to estimate the action values.

See also

- Reinforcement learning
- Temporal difference learning
- SARSA
- Iterated prisoner's dilemma
- Game theory

References

1. Melo, Francisco S. "Convergence of Q-learning: a simple proof" (http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf) (PDF).
2. Matiisen, Tambet (December 19, 2015). "Demystifying Deep Reinforcement Learning" (http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/). *neuro.cs.ut.ee*. Computational Neuroscience Lab. Retrieved 2018-04-06.

3. Sutton, Richard; Barto, Andrew (1998). *Reinforcement Learning: An Introduction* (http://incompleteideas.net/sutton/book/ebook/the-book.html). MIT Press.
4. Russell, Stuart J.; Norvig, Peter (2010). *Artificial Intelligence: A Modern Approach* (Third ed.). Prentice Hall. p. 649. ISBN 978-0136042594.
5. Baird, Leemon (1995). "Residual algorithms: Reinforcement learning with function approximation" (http://www.leemon.com/papers/1995b.pdf) (PDF). *ICML*: 30–37.
6. François-Lavet, Vincent; Fonteneau, Raphael; Ernst, Damien (2015-12-07). "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies". arXiv:1512.02011 (https://arxiv.org/abs/1512.02011) [cs.LG (https://arxiv.org/archive/cs.LG)].
7. Sutton, Richard S.; Barto, Andrew G. "2.7 Optimistic Initial Values" (https://web.archive.org/web/20130908031737/http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node21.html). *Reinforcement Learning: An Introduction*. Archived from the original (http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node21.html) on 2013-09-08. Retrieved 2013-07-18.
8. Shteingart, Hanan; Neiman, Tal; Loewenstein, Yonatan (May 2013). "The role of first impression in operant learning" (http://ratio.huji.ac.il/sites/default/files/publications/dp626.pdf) (PDF). *Journal of Experimental Psychology: General*. **142** (2): 476–488. doi:10.1037/a0029550 (https://doi.org/10.1037%2Fa0029550). ISSN 1939-2222 (https://www.worldcat.org/issn/1939-2222). PMID 22924882 (https://pubmed.ncbi.nlm.nih.gov/22924882).
9. Hasselt, Hado van (5 March 2012). "Reinforcement Learning in Continuous State and Action Spaces" (https://books.google.com/books?id=YPjNuvrJR0MC). In Wiering, Marco; Otterlo, Martijn van (eds.). *Reinforcement Learning: State-of-the-Art*. Springer Science & Business Media. pp. 207–251. ISBN 978-3-642-27645-3.
10. Tesauro, Gerald (March 1995). "Temporal Difference Learning and TD-Gammon" (http://www.bkgm.com/articles/tesauro/tdl.html). *Communications of the ACM*. **38** (3): 58–68. doi:10.1145/203330.203343 (https://doi.org/10.1145%2F203330.203343). Retrieved 2010-02-08.
11. Watkins, C.J.C.H. (1989), *Learning from Delayed Rewards* (http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf) (PDF) (Ph.D. thesis), Cambridge University
12. Watkins and Dayan, C.J.C.H., (1992), 'Q-learning.Machine Learning'
13. Bozinovski, S. (15 July 1999). "Crossbar Adaptive Array: The first connectionist network that solved the delayed reinforcement learning problem" (https://books.google.com/books?id=clKwynlfZYkC&pg=PA320-325). In Dobnikar, Andrej; Steele, Nigel C.; Pearson, David W.; Albrecht, Rudolf F. (eds.). *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Portorož, Slovenia, 1999*. Springer Science & Business Media. pp. 320–325. ISBN 978-3-211-83364-3.
14. Bozinovski, S. (1982). "A self learning system using secondary reinforcement" (https://books.google.com/books?id=mGtQAAAAMAAJ&pg=PA397). In Trappl, Robert (ed.). *Cybernetics and Systems Research: Proceedings of the Sixth European Meeting on Cybernetics and Systems Research*. North Holland. pp. 397–402. ISBN 978-0-444-86488-8.
15. Barto, A. (24 February 1997). "Reinforcement learning" (https://books.google.com/books?id=oLcAiySCow0C). In Omidvar, Omid; Elliott, David L. (eds.). *Neural Systems for Control*. Elsevier. ISBN 978-0-08-053739-9.
16. "Methods and Apparatus for Reinforcement Learning, US Patent #20150100530A1" (https://patentimages.storage.googleapis.com/71/91/4a/c5cf4ffa56f705/US20150100530A1.pdf) (PDF). US Patent Office. 9 April 2015. Retrieved 28 July 2018.
17. Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Rusu, Andrei A.; Veness, Joel; Bellemare, Marc G.; Graves, Alex; Riedmiller, Martin; Fidjeland, Andreas K. (Feb 2015). "Human-level control through deep reinforcement learning". *Nature*. **518** (7540): 529–533. doi:10.1038/nature14236 (https://doi.org/10.1038%2Fnature14236). ISSN 0028-0836 (https://www.worldcat.org/issn/0028-0836). PMID 25719670 (https://pubmed.ncbi.nlm.nih.gov/25719670).
18. van Hasselt, Hado (2011). "Double Q-learning" (http://papers.nips.cc/paper/3964-double-q-learning) (PDF). *Advances in Neural Information Processing Systems*. **23**: 2613–2622.
19. van Hasselt, Hado; Guez, Arthur; Silver, David (2015). "Deep reinforcement learning with double Q-learning" (https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847) (PDF). *AAAI Conference on Artificial Intelligence*: 2094–2100.

20. Strehl, Alexander L.; Li, Lihong; Wiewiora, Eric; Langford, John; Littman, Michael L. (2006). "Pac model-free reinforcement learning" (https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/published-14.pdf) (PDF). *Proc. 22nd ICML*: 881–888.
21. Maei, Hamid; Szepesvári, Csaba; Bhatnagar, Shalabh; Sutton, Richard (2010). "Toward off-policy learning control with function approximation in Proceedings of the 27th International Conference on Machine Learning" (https://webdocs.cs.ualberta.ca/~sutton/papers/MSBS-10.pdf) (PDF). pp. 719–726.

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" (https://web.archive.org/web/20081202105235/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)
- JavaScript Example with Reward Driven RNN learning (https://gammastorm.github.io/SinglePages/Brain.html)
- A Brain Library (https://github.com/gammastorm/gammastorm.github.io/blob/master/myjs/Brain.js)
- A Genetics Library used by the Brain (https://github.com/gammastorm/gammastorm.github.io/blob/master/myjs/SelfGenetics.js)