Title: Q-learning

URL: https://en.wikipedia.org/wiki/Q-learning

PageID: 1281850

Categories: Category:1989 in artificial intelligence, Category:Machine learning algorithms, Category:Reinforcement learning

-----

Supervised learning

Unsupervised learning

Semi-supervised learning

Self-supervised learning

Reinforcement learning

Meta-learning

Online learning

Batch learning

Curriculum learning

Rule-based learning

Neuro-symbolic AI

Neuromorphic engineering

Quantum machine learning

Classification

Generative modeling

Regression

Clustering

Dimensionality reduction

Density estimation

Anomaly detection

Data cleaning

AutoML

Association rules

Semantic analysis

Structured prediction

Feature engineering

Feature learning

Learning to rank

Grammar induction

Ontology learning

Multimodal learning

Apprenticeship learning

Decision trees

Ensembles Bagging Boosting Random forest

Bagging

Boosting

Random forest

k -NN

Linear regression

Naive Bayes

Artificial neural networks

Logistic regression

Perceptron

Relevance vector machine (RVM)

Support vector machine (SVM)

BIRCH

CURE

Hierarchical

k -means

Fuzzy

Expectation–maximization (EM)

DBSCAN

OPTICS

Mean shift

Factor analysis

CCA

ICA

LDA

NMF

PCA

PGD

t-SNE

SDL

Graphical models Bayes net Conditional random field Hidden Markov

Bayes net

Conditional random field

Hidden Markov

RANSAC

k -NN

Local outlier factor

Isolation forest

Autoencoder

Deep learning

Feedforward neural network

Recurrent neural network LSTM GRU ESN reservoir computing

LSTM

GRU

ESN

reservoir computing

Boltzmann machine Restricted

Restricted

GAN

Diffusion model

SOM

Convolutional neural network U-Net LeNet AlexNet DeepDream

U-Net

LeNet

AlexNet

DeepDream

Neural field Neural radiance field Physics-informed neural networks

Neural radiance field

Physics-informed neural networks

Transformer Vision

Vision

Mamba

Spiking neural network

Memtransistor

Electrochemical RAM (ECRAM)

Q-learning

Policy gradient

SARSA

Temporal difference (TD)

Multi-agent Self-play

Self-play

Active learning

Crowdsourcing

Human-in-the-loop

Mechanistic interpretability

RLHF

Coefficient of determination

Confusion matrix

Learning curve

ROC curve

Kernel machines

Bias–variance tradeoff

Computational learning theory

Empirical risk minimization

Occam learning

PAC learning

Statistical learning

VC theory

Topological deep learning

AAAI

ECML PKDD

NeurIPS

ICML

ICLR

IJCAI

ML

JMLR

Glossary of artificial intelligence

List of datasets for machine-learning research List of datasets in computer vision and image processing

List of datasets in computer vision and image processing

Outline of machine learning

v

t

e

Q -learning is a reinforcement learning algorithm that trains an agent to assign values to its possible actions based on its current state , without requiring a model of the environment (model-free). It can handle problems with stochastic transitions and rewards without requiring adaptations. [1]

For example, in a grid maze, an agent learns to reach an exit worth 10 points. At a junction, Q-learning might assign a higher value to moving right than left if right gets to the exit faster, improving this choice by trying both directions over time.

For any finite Markov decision process , Q -learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. [2] Q -learning can identify an optimal action-selection policy for any given finite

Markov decision process, given infinite exploration time and a partly random policy. [2]

"Q" refers to the function that the algorithm computes: the expected reward—that is, the quality —of an action taken in a given state. [3]

Reinforcement learning

Reinforcement learning involves an agent , a set of states S {\displaystyle {\mathcal {S}}} , and a set A {\displaystyle {\mathcal {A}}} of actions per state. By performing an action a ∈ A {\displaystyle a\in {\mathcal {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 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 expected values of the rewards of all future steps starting from the current state. [1]

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, less time is spent fighting the departing passengers. 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

After Δ t {\displaystyle \Delta t} steps into the future the agent will decide some next step. The weight for this step is calculated as γ Δ t {\displaystyle \gamma ^{\Delta t}} , where γ {\displaystyle \gamma } (the discount factor) is a number between 0 and 1 (0 ≤ γ ≤ 1 {\displaystyle 0\leq \gamma \leq 1}). Assuming γ < 1 {\displaystyle \gamma <1} , it has the effect of valuing rewards received earlier higher than those received later (reflecting the value of a "good start"). γ {\displaystyle \gamma } may also be interpreted as the probability to succeed (or survive) at every step Δ t {\displaystyle \Delta t} .

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

Before learning begins, ■ Q {\displaystyle Q} ■ is initialized to a possibly arbitrary fixed value (chosen by the programmer). Then, at each time t {\displaystyle t} the agent selects an action A t {\displaystyle A_{t}} , observes a reward R t + 1 {\displaystyle R_{t+1}} , enters a new state S t + 1 {\displaystyle S_{t+1}} (that may depend on both the previous state S t {\displaystyle S_{t}} and the selected action), and Q {\displaystyle Q} is updated. The core of the algorithm is a Bellman equation as a simple value iteration update , using the weighted average of the current value and the new information: [4]

where R t + 1 {\displaystyle R_{t+1}} is the reward received when moving from the state S t {\displaystyle S_{t}} to the state S t + 1 {\displaystyle S_{t+1}} , and α {\displaystyle \alpha } is the learning rate (0 < α ≤ 1) {\displaystyle (0<\alpha \leq 1)} .

Note that Q n e w (S t , A t) {\displaystyle Q^{new}(S_{t},A_{t})} is the sum of three terms:

(1 − α) Q (S t , A t) {\displaystyle (1-\alpha)Q(S_{t},A_{t})} : the current value (weighted by one minus the learning rate) α R t + 1 {\displaystyle \alpha \,R_{t+1}} : the reward R t + 1 {\displaystyle R_{t+1}} to obtain if action A t {\displaystyle A_{t}} is taken when in state S t {\displaystyle S_{t}} (weighted by learning rate)

α γ max a Q (S t + 1 , a) {\displaystyle \alpha \gamma \max _{a}Q(S_{t+1},a)} : the maximum reward that can be obtained from state S t + 1 {\displaystyle S_{t+1}} (weighted by learning rate and discount factor)

An episode of the algorithm ends when state S t + 1 {\displaystyle S_{t+1}} is a final or terminal state . However, Q -learning can also learn in non-episodic tasks (as a result of the property of convergent infinite series). If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loops or paths.

For all final states s f {\displaystyle s_{f}} , Q (s f , a) {\displaystyle Q(s_{f},a)} is never updated, but is set to the reward value r {\displaystyle r} observed for state s f {\displaystyle s_{f}} . In most cases, Q (s f , a) {\displaystyle 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 α t = 1 {\displaystyle \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 α t = 0.1 {\displaystyle \alpha _{t}=0.1} for all t {\displaystyle t} . [5]

Discount factor

The discount factor ■ γ {\displaystyle \gamma } ■ 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 {\displaystyle 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 ■ γ = 1 {\displaystyle \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. [6] 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 . [7] In that case, starting with a lower discount factor and increasing it towards its final value accelerates learning. [8]

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", [9] 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 {\displaystyle r} can be used to reset the initial conditions. [10] According to this idea, the first time an action is taken the reward is used to set the value of Q {\displaystyle 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). [10] RIC seems to be consistent with human behaviour in repeated binary choice experiments. [10]

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 . [11] 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. [12] Another possibility is to integrate Fuzzy Rule Interpolation (FRI) and use sparse fuzzy rule-bases [13] instead of discrete Q-tables or ANNs, which has the advantage of being a human-readable knowledge representation form. 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). [14]

History

Q -learning was introduced by Chris Watkins in 1989. [15] A convergence proof was presented by Watkins and Peter Dayan in 1992, [16] building on Watkins' doctoral dissertation, Learning from Delayed Rewards . Eight years earlier in 1981, the same problem, under the name of "Delayed reinforcement learning," was solved by Bozinovski's Crossbar Adaptive Array (CAA). [17] [18] The memory matrix W = ■ w (a , s) ■ {\displaystyle 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 ′) {\displaystyle v(s')} ■ ;

Update crossbar value w ′ (a , s) = w (a , s) + v (s ′) {\displaystyle 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 ′) {\displaystyle 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. [19]

In 2014, Google DeepMind patented [20] an application of Q-learning to deep learning , entitled "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 of the agent and the data distribution, and the correlations between Q and the target values. The method can be used for stochastic search in various domains and applications. [1] [21]

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. [3] 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. [22]

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 [23] 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 Q A {\displaystyle Q^{A}} and Q B {\displaystyle Q^{B}} are trained in a mutually symmetric fashion using separate experiences. The double Q-learning update step is then as follows:

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 , [24] as in the DQN algorithm, resulting in Double DQN, which outperforms the original DQN algorithm. [25]

Others

Delayed Q-learning is an alternative implementation of the online Q -learning algorithm, with probably approximately correct (PAC) learning . [26]

Greedy GQ is a variant of Q -learning to use in combination with (linear) function approximation. [27] The advantage of Greedy GQ is that convergence is guaranteed even when function approximation is used to estimate the action values.

Distributional Q-learning is a variant of Q -learning which seeks to model the distribution of returns rather than the expected return of each action. It has been observed to facilitate estimate by deep neural networks and can enable alternative control methods, such as risk-sensitive control. [28]

Multi-agent learning

Q-learning has been proposed in the multi-agent setting (see Section 4.1.2 in [29]). One approach consists in pretending the environment is passive. [30] Littman proposes the minimax Q learning algorithm. [31]

Limitations

The standard Q-learning algorithm (using a Q {\displaystyle Q} table) applies only to discrete action and state spaces. Discretization of these values leads to inefficient learning, largely due to the curse of dimensionality . However, there are adaptations of Q-learning that attempt to solve this problem such as Wire-fitted Neural Network Q-Learning. [32]

See also

Reinforcement learning

Temporal difference learning

SARSA

Iterated prisoner's dilemma

Game theory

References

External links

Watkins, C.J.C.H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England.

Strehl, Li, Wiewiora, Langford, Littman (2006). PAC model-free reinforcement learning

Reinforcement Learning: An Introduction by Richard Sutton and Andrew S. Barto, an online textbook. See "6.5 Q-Learning: Off-Policy TD Control" .

Piqle: a Generic Java Platform for Reinforcement Learning

Reinforcement Learning Maze , a demonstration of guiding an ant through a maze using Q -learning

Q -learning work by Gerald Tesauro

v

t

e

History timeline

timeline

Companies

Projects

Parameter Hyperparameter

Hyperparameter

Loss functions

Regression Bias–variance tradeoff Double descent Overfitting

Bias–variance tradeoff

Double descent

Overfitting

Clustering

Gradient descent SGD Quasi-Newton method Conjugate gradient method

SGD

Quasi-Newton method

Conjugate gradient method

Backpropagation

Attention

Convolution

Normalization Batchnorm

Batchnorm

Activation Softmax Sigmoid Rectifier

Softmax

Sigmoid

Rectifier

Gating

Weight initialization

Regularization

Datasets Augmentation

Augmentation

Prompt engineering

Reinforcement learning Q-learning SARSA Imitation Policy gradient

Q-learning

SARSA

Imitation

Policy gradient

Diffusion

Latent diffusion model

Autoregression

Adversary

RAG

Uncanny valley

RLHF

Self-supervised learning

Reflection

Recursive self-improvement

Hallucination

Word embedding

Vibe coding

Machine learning In-context learning

In-context learning

Artificial neural network Deep learning

Deep learning

Language model Large language model NMT

Large language model

NMT

Reasoning language model

Model Context Protocol

Intelligent agent

Artificial human companion

Humanity's Last Exam

Artificial general intelligence (AGI)

AlexNet

WaveNet

Human image synthesis

HWR

OCR

Computer vision

Speech synthesis 15.ai ElevenLabs

15.ai

ElevenLabs

Speech recognition Whisper

Whisper

Facial recognition

AlphaFold

Text-to-image models Aurora DALL-E Firefly Flux Ideogram Imagen Midjourney Recraft Stable Diffusion

Aurora

DALL-E

Firefly

Flux

Ideogram

Imagen

Midjourney

Recraft

Stable Diffusion

Text-to-video models Dream Machine Runway Gen Hailuo AI Kling Sora Veo

Dream Machine

Runway Gen

Hailuo AI

Kling

Sora

Veo

Music generation Riffusion Suno AI Udio

Riffusion

Suno AI

Udio

Word2vec

Seq2seq

GloVe

BERT

T5

Llama

Chinchilla AI

PaLM

GPT 1 2 3 J ChatGPT 4 4o o1 o3 4.5 4.1 o4-mini 5

1

2

3

J

ChatGPT

4

4o

o1

o3

4.5

4.1

o4-mini

5

Claude

Gemini Gemini (language model) Gemma

Gemini (language model)

Gemma

Grok

LaMDA

BLOOM

DBRX

Project Debater

IBM Watson

IBM Watsonx

Granite

PanGu-Σ

DeepSeek

Qwen

AlphaGo

AlphaZero

OpenAI Five

Self-driving car

MuZero

Action selection AutoGPT

AutoGPT

Robot control

Alan Turing

Warren Sturgis McCulloch

Walter Pitts

John von Neumann

Claude Shannon

Shun'ichi Amari

Kunihiko Fukushima

Takeo Kanade

Marvin Minsky

John McCarthy

Nathaniel Rochester

Allen Newell

Cliff Shaw

Herbert A. Simon

Oliver Selfridge

Frank Rosenblatt

Bernard Widrow

Joseph Weizenbaum

Seymour Papert

Seppo Linnainmaa

Paul Werbos

Geoffrey Hinton

John Hopfield

Jürgen Schmidhuber

Yann LeCun

Yoshua Bengio

Lotfi A. Zadeh

Stephen Grossberg

Alex Graves

James Goodnight

Andrew Ng

Fei-Fei Li

Alex Krizhevsky

Ilya Sutskever

Oriol Vinyals

Quoc V. Le

Ian Goodfellow

Demis Hassabis

David Silver

Andrej Karpathy

Ashish Vaswani

Noam Shazeer

Aidan Gomez

John Schulman

Mustafa Suleyman

Jan Leike

Daniel Kokotajlo

François Chollet

Neural Turing machine

Differentiable neural computer

Transformer Vision transformer (ViT)

Vision transformer (ViT)

Recurrent neural network (RNN)

Long short-term memory (LSTM)

Gated recurrent unit (GRU)

Echo state network

Multilayer perceptron (MLP)

Convolutional neural network (CNN)

Residual neural network (RNN)

Highway network

Mamba

Autoencoder

Variational autoencoder (VAE)

Generative adversarial network (GAN)

Graph neural network (GNN)

Category