Q-learning

Q-learning es una técnica de aprendizaje por refuerzo utilizada en aprendizaje automático. El objetivo del Q-learning es aprender una serie de normas que le diga a un agente qué acción tomar bajo qué circunstancias. No requiere un modelo del entorno y puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones.

Para cualquier proceso de decisión de Markov finito (PDMF) (finite Markov decision process en inglés), Q-learning encuentra una política óptima en el sentido de que maximiza el valor esperado de la recompensa total sobre todos los pasos sucesivos, empezando desde el estado actual.[1] Q-learning puede identificar una norma de acción-selección óptima para cualquier PDMF, dado un tiempo de exploración infinito y una norma parcialmente aleatoria[1] "Q" nombra la función que devuelve la recompensa que proporciona el refuerzo y representa la "calidad" de una acción tomada en un estado dado.[2]

Índice

Aprendizaje por refuerzo

El aprendizaje por refuerzo involucra a un agente, un conjunto de *estados* S, y un conjunto A de *acciones* por estado. Llevando a cabo una acción $a \in A$, el agente pasa de un estado a otro.$a \in A$ La ejecución de una acción en un estado concreto le proporciona una recompensa al agente (una puntuación numérica).

El objetivo del agente es maximizar su (futura) recompensa total. Lo hace sumando la máxima recompensa alcanzable, en estados futuros, a la recompensa por alcanzar su estado actual, influyendo eficazmente en la acción actual por la futura recompensa potencial. Esta recompensa potencial es una suma ponderada de la esperanza matemática de las recompensas de los futuros pasos empezando desde el estado actual.

Como ejemplo, considérese el proceso de subir a un tren, en el que la recompensa se mide por el tiempo total de embarque en negativo (en otras palabras, el coste de subir al tren es igual a la rapidez total del embarque). Una estrategia es entrar al tren tan pronto como se abran las puertas, minimizando tu tiempo inicial de espera. No obstante, si el tren está lleno, tendrás una entrada lenta tras entrar por la puerta, ya que la gente está luchando para salir del tren mientras intentas subir. El tiempo de embarque, o el coste, es:

- 0 segundos de espera + 15 segundos de lucha

Al día siguiente, por azar (exploración), decides esperar y dejar a otras subir primero. Esto provoca que el tiempo de espera sea mayor. Aun así, la entrada se retrasa ya que el tiempo que lucha contra otros pasajeros no se recompensa. En general, este camino tiene una recompensa más alta que la del día anterior puesto que el tiempo de embarque es ahora:

- 5 segundos de espera + 0 segundo de lucha

A través de exploración, a pesar de que la acción inicial (de espera) resultara en un coste mayor (o recompensa negativa) que en la estrategia enérgica, el coste global es menor, descubriendo, así, una estrategia más recompensada.

Algoritmo

El peso de un paso en un estado Δtpasos en el futuro se calcula como $\gamma^{\Delta t}$. γ (el *factor de descuento*) es un número entre 0 y 1 ($0 \leq \gamma \leq 1$) y evalúa las recompensas recibidas anteriormente con un valor mayor que las recibidas posteriormente (reflejando el valor de un "buen inicio").γ también puede ser interpretada como la probabilidad de tener éxito (o sobrevivir) en cada paso Δt.

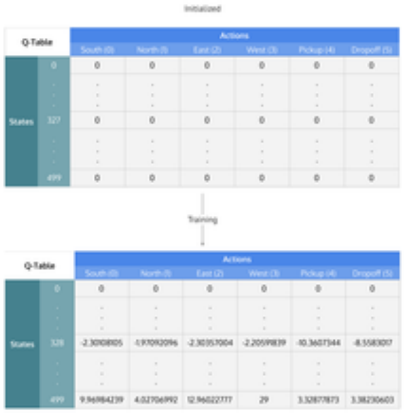

Tabla de estados de Q-learning por acciones inicializada a cero, entonces cada celda se actualiza con entrenamiento.

El algoritmo, por tanto, tiene una función que calcula la calidad de una combinación estado-acción:

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

Antes que comience el aprendizaje, Q se inicializa a un valor arbitrario constante (escogido por el programador). Después, en cada tiempo $t$el agente selecciona una acción a_t, observa una recompensa r_t, introduce un estado nuevo s_{t+1} (Que depende del estado anterior s_t y de la acción seleccionada), y Qse actualiza. El núcleo del algoritmo es una actualización del valor de la iteración simple, haciendo la media ponderada del valor antiguo y la información nueva:

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

Dónde r_t es la recompensa recibida al pasar del estado s_t al estado s_{t+1}, y α es el índice de aprendizaje ($0 < \alpha \leq 1$).

Un episodio del algoritmo termina cuando el estado s_{t+1}es un estado final o *terminal*. Aun así, Q-learning también puede aprender en tareas sin episodios. Si el factor de descuento es menor que 1, los valores de acción son finitos incluso si el problema puede contener bucles infinitos.

Para todos los estados finales s_f, $Q(s_f, a)$ nunca se actualiza, pero se fija al valor de recompensa r observado para el estado s_f. En la mayoría de los casos, $Q(s_f, a)Q(s_f, a)$puede ser igualado a 0.

Influencia de variables

Exploración vs beneficio

El índice de aprendizaje o *tamaño* del *paso* determina hasta qué punto la información adquirida sobrescribe la información vieja. Un factor de 0 hace que el agente no aprenda (únicamente aprovechando el conocimiento previo), mientras un factor de 1 hace que el agente considere solo la información más reciente (ignorando el conocimiento previo para explorar posibilidades).

En entornos totalmente deterministas, un índice de aprendizaje de $\alpha_t = 1$ es óptimo.$\alpha_t = 1$ Cuándo el problema es estocástico, el algoritmo converge bajo determinadas condiciones técnicas en el índice de aprendizaje que requiere un descenso hasta cero. En la práctica, a menudo se utiliza un índice de aprendizaje constante, como $\alpha_t = 0.1$para todo $t.\alpha_t = 0.1t$[3]

Factor de descuento

El factor de descuento γ determina la importancia de recompensas futuras. Un factor de 0 hará que el agente sea "miope" (o corto de miras) por considerar únicamente las recompensas actuales, i.e. r_t (en la regla de actualización anterior), mientras que un factor que se acerca a 1 hará que luche por una recompensa alta a largo plazo.r_t

Si el factor de descuento es mayor o igual a 1, los valores de acción pueden divergir. Para $\gamma = 1$, sin un estado terminal, o si el agente nunca llega a uno, todos los historiales se vuelven infinitamente largos, y funciones aditivas, recompensas discontinuas generalmente tienden a infinito. [4] Incluso con un factor de descuento ligeramente menor que 1, el aprendizaje de Q-función lleva a la propagación de errores e inestabilidades cuando la función de valor está aproximada con una red neuronal artificial.[5] En ese caso, empezando con un factor de descuento menor y aumentándolo hacia su valor final se acelera el aprendizaje.[6]

Condiciones iniciales (Q0)

Al ser Q-learning un algoritmo iterativo, implícitamente supone una condición inicial antes de que se de la primera actualización. Valores iniciales altos, también conocidos como "condiciones iniciales optimistas",[7] puede promover la exploración: da igual la acción seleccionada, la regla de actualización provocará que tenga valores menores que la otra alternativa, por tanto aumenta su probabilidad de elección. La prime$r$ra recompensa r puede ser utilizada para reiniciar las condiciones iniciales.r[8] Según esta idea, la primera vez que se elige una acción, se fija el valor de la recompensa a $Q.Q$ Esto permite el aprendizaje inmediato en caso de recompensas deterministas fijas. Un modelo que incorpora *reinicialización de condiciones iniciales* (RCI), se espera que pronostique el comportamiento de los participantes mejor que un modelo que suponga cualquier *condición inicial arbitraria* (CIA).[8] RCI, sin embargo, parece para ser consistente del comportamiento humano en repetidos experimentos de elección binaria.[8]

Implementación

Q-learning simplemente guarda información en tablas. Esta aproximación flaquea con números crecientes de acciones/estados.

Aproximación de funciones

Q-learning se puede combinar con aproximación de funciones.[9] Esto hace posible la aplicación del algoritmo a problemas más largos, incluso cuando el espacio de estados es continuo.

Una solución es utilizar una red neuronal artificial (adaptado) como aproximador de funciones.[10] La aproximación de funciones puede acelerar el aprendizaje en problemas finitos, debido a que el algoritmo puede generalizar experiencias previas a estados no vistos anteriormente.

Cuantificación

Esta técnica para disminuir el espacio de acciones/estados cuantifica los posibles valores. Considérese el ejemplo de aprender a equilibrar un palo en un dedo. Para describir un estado en un punto concreto en el tiempo involucra la posición del dedo en el espacio, su velocidad, el ángulo del palo y la velocidad angular del palo. Esto produce un vector de cuatro elementos que describe un estado, i.e. una captura de un estado codificado con cuatro valores. El problema es que están presentes infinitos estados posibles. Para encoger el espacio posible de las acciones válidas se pueden descartar muchos valores. La distancia exacta del dedo desde su posición de inicio (-infinito a infinito) no se conoce, sino si está lejos (cerca, lejos).

Historia

Q-learning fue introducido por Watkins[11] en 1989. Watkins presentó una prueba de convergencia a Dayan[12] en 1992. Una prueba matemática más detallada fue expuesta por Tsitsiklis[13] en 1994, y por Bertsekas y Tsitsiklis en su libro de 1996 *Neuro-Dynamic Programming.*[14]

Watkins lo señalaba ya el título de la tesis de su doctorado “Learning from delayed rewards” (Aprendiendo de las recompensas tardías). Ocho años antes en 1981 el mismo problema, bajo el nombre de “aprendizaje por refuerzo tardío”, que fue solucionado por Bozinovski's Crossbar Adaptive Array (CAA).[15] [16] La matriz de memoria W(a,s) es exactamente igual que la Q-table de Q-learning de ocho años más tarde. La arquitectura introdujo el término “evaluación de estado” en aprendizaje por refuerzo. El algoritmo de aprendizaje *crossbar (*escrito en pseudocódigo matemático en papel) realiza los siguientes cálculos en cada iteración:

- En el estado s efectúa la acción a;
- En consecuencia recibe el estado s';
- Realiza la valuación del estado v(s');
- Actualiza el valor *crossbar* W'(a,s) = W(a,s) + v(s').

El término “refuerzo secundario” viene de la teoría de aprendizaje animal, utilizada para modelar los valores de estado a través de propagación hacia atrás: el valor de estado v(s') de la situación consecuente es propagado hacia atrás hacia las situaciones encontradas previamente. CAA calcula los valores de estados verticalmente y las acciones horizontalmente (el "*crossbar*"). Grafos de demostración nos muestran los estados contenidos en el aprendizaje por refuerzo tardío (deseables, indeseables, y estados neutros), que han sido calculados por la función de evaluación de estados. Este sistema de aprendizaje fue el precursor del algoritmo Q-learning.[17]

En 2014 Google DeepMind patentó una aplicación de Q-learning de cara al aprendizaje profundo, llamado "aprendizaje de refuerzo profundo" o "Q-learning profundo" que puede jugar a los juegos Atari 2600 en niveles humanos expertos.[18]

Variantes

Q-learning profundo

El sistema DeepMind utilizó redes neuronales convolucionales profundas, con capas de filtros convolucionados enladrillados para asemejar los efectos de los campos receptivos. El aprendizaje por refuerzo es inestable o divergente cuando un aproximador de funciones no lineales se utiliza para representar Q. Esta inestabilidad proviene de la correlación presente en las secuencias de observación, el hecho de que actualizaciones pequeñas de Q puedan cambiar significativamente la política y la distribución de los datos, y las correlaciones entre Q y los valores objetivo.

La técnica utilizó *repetición de experiencia*, un mecanismo biológicamente inspirado que usa una muestra aleatoria de acciones previas en vez de la acción más reciente para continuar.[2] Esto elimina las correlaciones en la secuencia de observación y suaviza los cambios en la distribución de los datos. La actualización iterativa ajusta Q de cara a los valores objetivo que solo se actualizan periódicamente, reduciendo más adelante las correlaciones con el objetivo.[19]

Q-learning doble

Puesto que el valor futuro máximo aproximado de una acción en Q-learning se evalúa utilizando la misma función Q que en la política de acción actualmente seleccionada, en entornos ruidosos Q-learning a veces puede sobrestimar los valores de la acción, retrasando el aprendizaje. Se propuso una variante llamada Q-learning doble para corregir esto. Q-learning doble[20] es un algoritmo de aprendizaje por refuerzo sin política, donde una política diferente se utiliza para la evaluación del valor que se usa para seleccionar próxima acción.

En la práctica, dos funciones de valor diferentes se entrenan en Q^Auna moda simétrica mutua usando experiencias separadas, Q^Ay Q^B.$Q^A Q^B$ El paso de actualización de Q-learning doble es, por tanto, así:

$$Q^A_{t+1}(s_t, a_t) = Q^A_t(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q^B_t\left(s_{t+1}, \arg_a \max Q^A_t(s_{t+a}, a) - Q^A_t(s_t, a_t)\right)\right.$$, y

Ahora el valor estimado del futuro descontado se evalúa utilizando una política diferente, lo cual soluciona el problema de la sobrestimación.

Este algoritmo se combinó más tarde con aprendizaje profundo, como el algoritmo DQN, resultando en doble DQN, el cual supera en rendimiento el algoritmo DQN original.[21]

Otros

Q-learning tardío es una implementación alternativa del algoritmo Q-learning online, con aprendizaje correcto probablemente aproximado (PAC).[22]

El "Avaro GQ" es una variante de Q-learning para combinarlo con aproximación de funciones (lineales).[23] La ventaja del "Avaro GQ" es que la convergencia es garantizada incluso cuando la aproximación de funciones se usa para estimar los valores de acción.

Ver también

- Aprendizaje por refuerzo
- Aprendizaje de diferencia temporal
- SARSA
- El dilema del prisionero iterado
- Teoría de juego

Referencias

1. Francisco S. Melo, "Convergencia de Q-aprendiendo: una prueba sencilla"
2. Matiisen, Tambet (19 de diciembre de 2015). «Demystifying Deep Reinforcement Learning | Computational Neuroscience Lab» (http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/). *neuro.cs.ut.ee* (en inglés estadounidense). Consultado el 6 de abril de 2018.
3. Aprendizaje de refuerzo: Una Introducción. Richard Sutton y Andrew Barto. MIT Prensa, 1998.
4. Stuart J. Russell; Peter Norvig (2010). *Artificial Intelligence: A Modern Approach* (Third edición). 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). *ICML*.
6. **MISSING LINK.**.
7. «Archived copy» (https://web.archive.org/web/20130908031737/http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node21.html). Archivado desde el original (http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node21.html) el 8 de septiembre de 2013. Consultado el 18 de julio de 2013.
8. Shteingart, H; Neiman, T; Loewenstein, Y (May 2013). «The Role of First Impression in Operant Learning» (http://ratio.huji.ac.il/sites/default/files/publications/dp626.pdf). *J Exp Psychol Gen* **142** (2): 476-88. PMID 22924882 (https://www.ncbi.nlm.nih.gov/pubmed/22924882). doi:10.1037/a0029550 (https://dx.doi.org/10.1037%2Fa0029550).
9. Hasselt, Hado van (5 de marzo de 2012). «Reinforcement Learning in Continuous State and Action Spaces». Wiering, ed. *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» (https://web.archive.org/web/20100209103427/http://www.bkgm.com/articles/tesauro/tdl.html). *Communications of the ACM* **38** (3): 58-68. doi:10.1145/203330.203343 (https://dx.doi.org/10.1145%2F203330.203343). Archivado desde el original (http://www.bkgm.com/articles/tesauro/tdl.html) el 9 de febrero de 2010. Consultado el 8 de febrero de 2010.
11. Watkins, C.J.C.H. (1989), *Learning from Delayed Rewards* (http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf), Cambridge University
12. Watkins Y Dayan, C.J.C.H., (1992), 'Q-aprendiendo.Aprendizaje de máquina'
13. Tsitsiklis, J., (1994), 'Aproximación Estocástica Asíncrona y Q-aprendiendo. Aprendizaje de máquina'
14. Bertsekas Y Tsitsiklis, (1996), 'Neuro-Programación Dinámica. Athena Científica'
15. Bozinovski, S. (15 de julio de 1999). «Crossbar Adaptive Array: The first connectionist network that solved the delayed reinforcement learning problem». editor-Dobnikar, ed. *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.
16. Bozinovski, S. (1982). «A self learning system using secondary reinforcement». 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.
17. Barto, A. (24 de febrero de 1997). «Reinforcement learning». Omidvar, ed. *Neural Systems for Control*. Elsevier. ISBN 978-0-08-053739-9.
18. «Methods and Apparatus for Reinforcement Learning, US Patent #20150100530A1» (https://patentimages.storage.googleapis.com/71/91/4a/c5cf4ffa56f705/US20150100530A1.pdf). US Patent Office. 9 de abril de 2015. Consultado el 28 de julio de 2018.
19. Mnih, Volodymyr (2015). *Human-level control through deep reinforcement learning* (http://www.nature.com/nature/journal/v518/n7540/pdf/nature14236.pdf) **518**. pp. 529-533.
20. 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.
21. 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.
22. Strehl, Alexander L.; Li, Lihong; Wiewiora, Eric; Langford, John; Littman, Michael L. (2006). «=Pac model-free reinforcement learning» (http://research.microsoft.com/pubs/178886/published.pdf). *Proc. 22nd ICML*.

23. Maei, Hamid (2010). «Toward off-policy learning control with function approximation in Proceedings of the 27th International Conference on Machine Learning» (https://web.archive.org/web/20120908050052/http://webdocs.cs.ualberta.ca/~sutton/papers/MSBS-10.pdf). Archivado desde el original (https://webdocs.cs.ualberta.ca/~sutton/papers/MSBS-10.pdf) el 8 de septiembre de 2012. Consultado el 5 de enero de 2019.

Enlaces externos

- 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://web.archive.org/web/20180420073911/https://gammastorm.github.io/SinglePages/Brain.html)
- A Brain Library (https://github.com/gammastorm/gammastorm.github.io/blob/master/myjs/Brain.js) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/https://github.com/gammastorm/gammastorm.github.io/blob/master/myjs/Brain.js) y la última versión (https://web.archive.org/web/2/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) (enlace roto disponible en Internet Archive; véase el historial (https://web.archive.org/web/*/https://github.com/gammastorm/gammastorm.github.io/blob/master/myjs/SelfGenetics.js) y la última versión (https://web.archive.org/web/2/https://github.com/gammastorm/gammastorm.github.io/blob/master/myjs/SelfGenetics.js)).
