

08MIAR-Aprendizaje por refuerzo

Sesión 1 – Introducción

Curso Abril 23/24

The logo consists of the lowercase letters "viu" in white, sans-serif font, centered within a solid orange circle.

viu

Universidad
Internacional
de Valencia

De:

 Planeta Formación y Universidades

Sobre mí

Julio Silva

Postdoctoral Researcher (<https://liviamtl.ca>)

juliojose.silva@professor.universidadviu.com

<https://es.linkedin.com/in/julio-silva-rodriguez>

Sobre la asignatura

El contenido de la asignatura se organizará en tres bloques:

- En el **primer bloque** introduciremos el **contexto** en el que se desarrollan las soluciones de aprendizaje por refuerzo, así como los **conceptos** y **términos** más importantes.
- El **segundo bloque** estará compuesto del estudio de los **algoritmos** que forman el estado del arte actual. Principalmente, cuáles son las estrategias de **aprendizaje**.
- El **tercer bloque** cubrirá una serie de sesiones **prácticas** donde implementaremos soluciones de aprendizaje por refuerzo usando diferentes **frameworks** y **librerías**.

Para más información sobre las sesiones: **Anexo de organización de sesiones en la sección de guía didáctica de la asignatura.**

Sobre la asignatura

Respecto a los requisitos de la asignatura:

- Los conceptos teóricos que veremos se basarán en conceptos matemáticos y estadísticos. Hablaremos de distribuciones de probabilidad, de búsqueda óptima, de cadenas de Markov, etc. El conocer estos conceptos no es una restricción para el curso, pero sí que es recomendable.
- En la parte práctica trabajaremos con Python y algunas librerías típicas de entornos para trabajar con datos y modelado, como por ejemplo Keras, Tensorflow y Pytorch. Habrá disponible una sesión práctica grabada enfocada en la puesta a punto del entorno y la presentación de los frameworks que utilizaremos.
- En ambas partes será imprescindible conocer y tener cierta experiencia con modelos Deep Learning y, en concreto, con redes convolucionales.

Sobre la asignatura

El método de evaluación de la asignatura estará compuesto de:

- Participación en foros de debate (10%)
- Actividad sobre artículo científico (10%)
- Actividad sobre una serie de vídeos teóricos (10%)
- Proyecto práctico (30%)
- Examen final (40%)

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

¿Qué es el aprendizaje por refuerzo?

“Aprender a partir de la interacción es una idea fundamental y común a todas las teorías del aprendizaje y de la inteligencia”

“Aprendizaje por refuerzo [...] es a la vez un problema, un conjunto de soluciones que funcionan bien sobre un conjunto de problemas y el campo que estudia esos problemas y los métodos que pueden solucionarlo.”

Reinforcement learning: An Introduction, R. Sutton & A. Barto

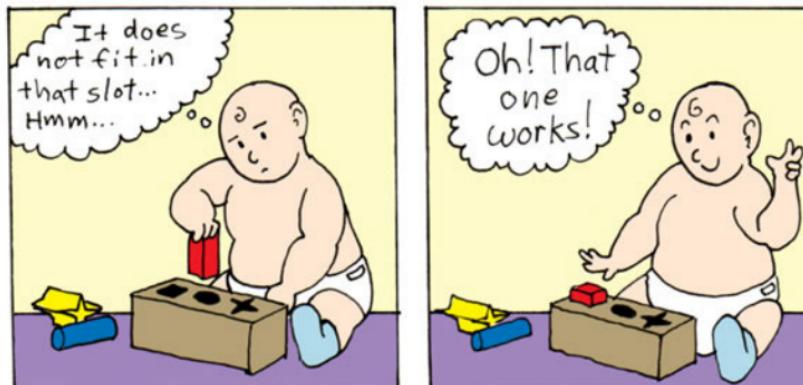
“El objetivo del aprendizaje por refuerzo es entrenar un agente inteligente que es capaz de interactuar con un entorno de manera inteligente.”

Deep Q Network vs Policy gradients, Felix Yu

El aprendizaje va a ser llevado a cabo únicamente a la interacción del agente con el entorno. No tenemos un conjunto fijo de datos etiquetados. El agente toma decisiones en base a dicha interacción.

¿Qué es el aprendizaje por refuerzo?

Podemos ver el aprendizaje por refuerzo como el estudio y diseño de **agentes** que aprenden por medio de **prueba-y-error**.



<https://rochemamabolo.files.wordpress.com/2018/07/trial02.jpg?w=4>

“Reinforcement learning is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.”

<https://huggingface.co/learn/deep-rl-course/unit1/what-is-rl>

¿Qué es el aprendizaje por refuerzo?

Si nos paramos y analizamos otras ramas dentro del aprendizaje basado en datos, encontramos dos grandes conjuntos: métodos supervisados y métodos no supervisados. Podríamos añadir aprendizaje por refuerzo como un tercer conjunto dentro de esta clasificación.

Desde un punto de vista de datos, podemos definir la relación de cada conjunto con el conocimiento que se puede obtener a partir de los datos:

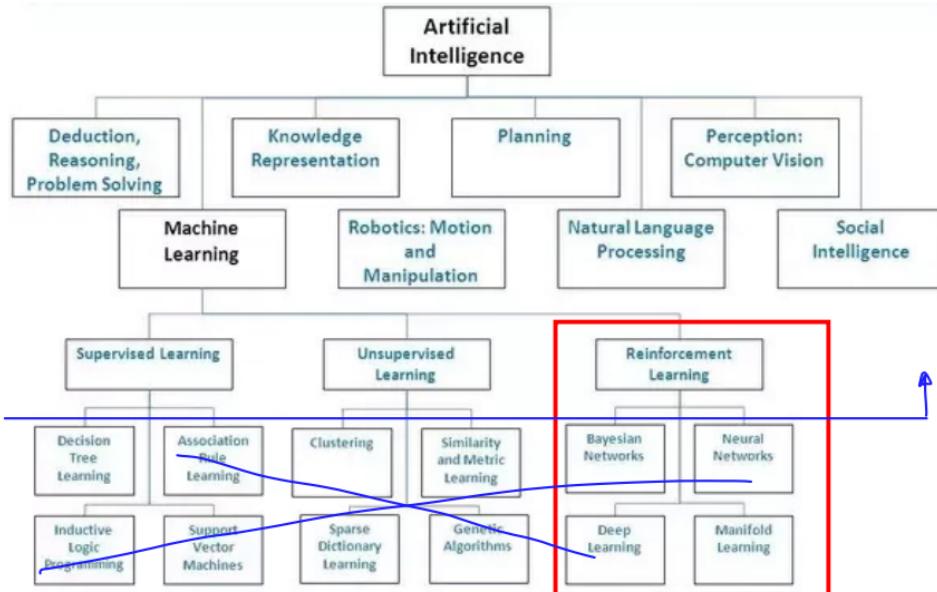
Análisis descriptivo → Métodos no supervisados

Análisis predictivo → Métodos supervisados

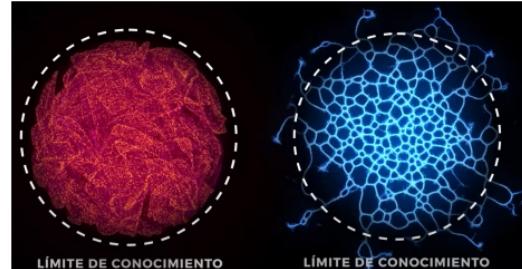
Análisis prescriptivo → Métodos aprendizaje por refuerzo

A.p.R. soluciones originales, más
alla del conocimiento humano.

¿Qué es el aprendizaje por refuerzo?



<https://qph.fs.quoracdn.net/main-qimg-541146e2d9d611a42ebb5074aa72fef4.webp>



<https://www.youtube.com/watch?v=yAF3bopYMGk>

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Estado del arte

Como en otras ramas de la inteligencia artificial, los primeros algoritmos y soluciones basadas en aprendizaje por refuerzo datan de hace algunas décadas. En nuestro caso, fue en los años **80/principios de los 90 cuando hubo una tendencia positiva** en el estudio y uso de algoritmos basados en **aprendizaje por refuerzo**.

Aunque los resultados fueron prometedores, debido a las capacidades computacionales y al desarrollo de otros algoritmos con enfoques similares en aquel momento (**algoritmos evolutivos**), las soluciones basadas en aprendizaje por refuerzo disminuyeron hasta hace unos años.

Como apunte, **la principal diferencia entre algoritmos de aprendizaje por refuerzo y algoritmos evolutivos es que los agentes de algoritmos evolutivos no aprenden de la interacción con su entorno**. Las reglas de su interacción son inamovibles, por lo que no pueden aprovechar toda la información de la que disponen para aprender y adaptarse al problema que intentan resolver.

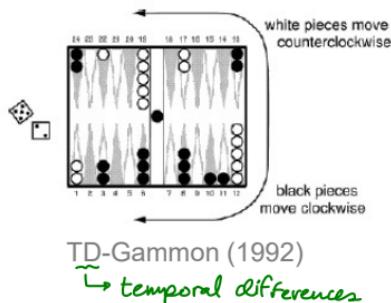
Estado del arte

Como decíamos, hasta hace algunos años no hubo una explosión en este tipo de algoritmos y del potencial que tienen. Una relación directa ha sido el magnífico desarrollo de las **técnicas de Deep Learning** así como la capacidad computacional que ahora tenemos a nuestra disposición.

Igualmente, si tuviéramos que elegir tres hitos que sentaron las bases para hablar de aprendizaje por refuerzo hoy serían:



Minsky PHD thesis (1954)



NATURE | LETTER

日本語要約

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Battail, Amir Sadik, Ioannis Antonoglou, Helen King, Charles Kumaran, Daan Wierwille, Shane Legg & Demis Hassabis

Affiliations · Contributions · Corresponding authors

Nature | \$18, 529–533 (26 February 2015) | doi:10.1038/nature14236
Received 11 July 2014 | Accepted 16 January 2015 | Published online 25 February 2015

DeepMind (2015)

Se ha desarrollado sobre todo en el campo de los juegos o videojuegos.

Estado del arte

Atari

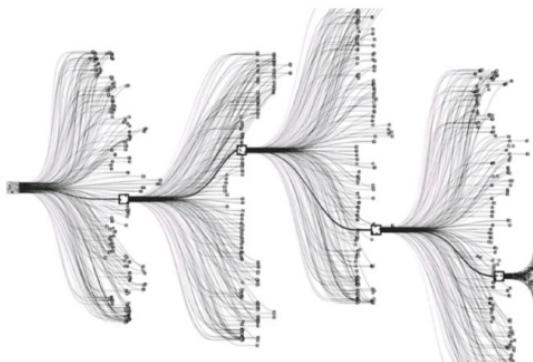
Vamos a centrarnos en Google Deepmind. El gran hito que consiguió Google Deepmind con su solución fue que combinó *Deep Learning* con Aprendizaje por refuerzo.

Hasta ese momento, las variables que se usaban en algoritmos por refuerzo eran variables *tabuladas* o definidas manualmente. Con la solución de **DeepMind, se usó una red neuronal como función aproximadora para que el agente tomara decisiones**, algo inaudito hasta el momento y con una base de inteligencia artificial formidable.

Este primer hito se realizó sobre un entorno de simulación de videojuegos de Atari.

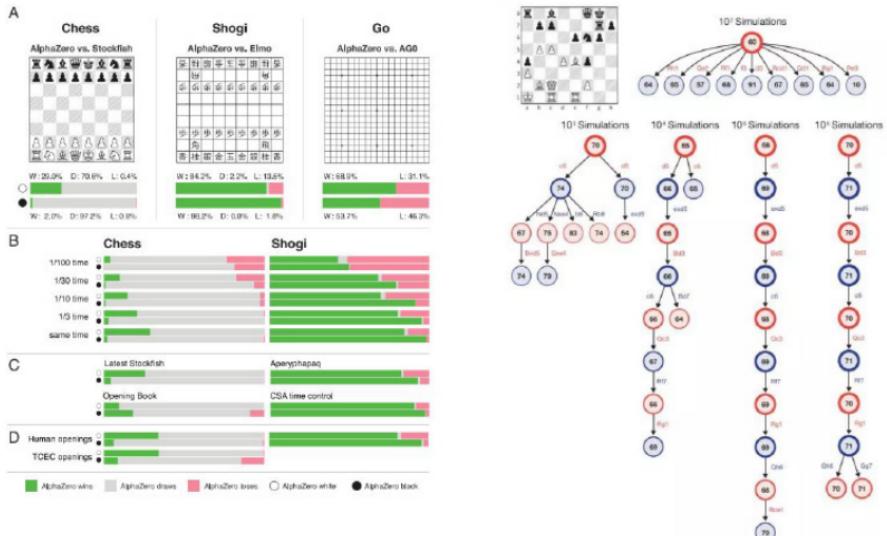


Estado del arte



Hitos de DeepMind: **AlphaGo (2016)**

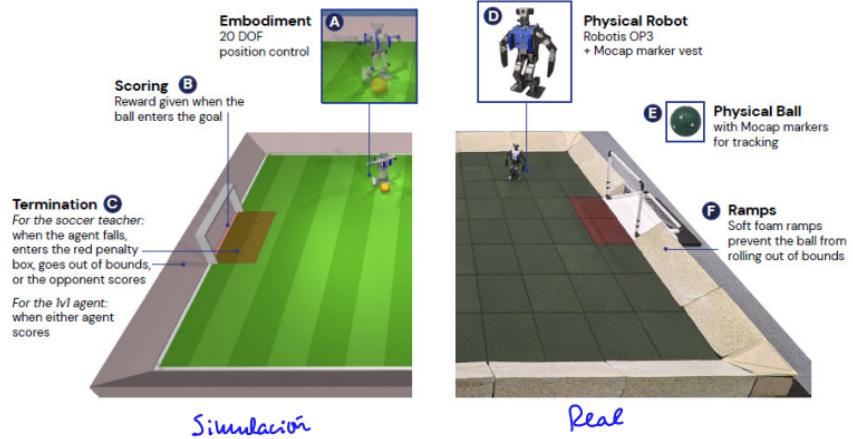
Estado del arte



Hitos de DeepMind: **AlphaZero (2017)**

Estado del arte

Robótica



Hitos de DeepMind: **OP3-Soccer (2023)**

Estado del arte

Otra compañía que ha sido un pulmón en el desarrollo del aprendizaje por refuerzo estos últimos años ha sido OpenAI.

La misión de OpenAI es la creación de una inteligencia artificial de carácter general con una fuerte base ética y de principios.

OpenAI han sido los responsables de algunos de los algoritmos que ahora mismo son estado del arte, así como de ser la primera compañía en ganar con inteligencia artificial en entornos de colaboración (y muy avanzados en cuanto a complejidad). Un ejemplo es el videojuego *DOTA2*.

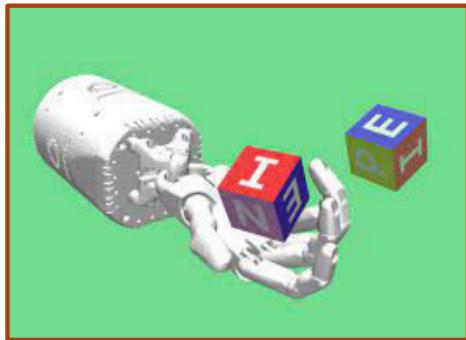


Estado del arte



Hitos de OpenAI: **Dota2 (2019)**

Estado del arte



Hitos de OpenAI: **Robótica (2019)**

Estado del arte: Tencencias Actuales



2023-4-27

Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning

Tuomas Haarnoja^{*,1}, Ben Moran^{*,1}, Guy Lever^{*,1}, Sandy H. Huang^{*,1}, Dhruva Tirumala¹, Markus Wulfmeier¹, Jan Humprik¹, Saran Tunyasuvunakool¹, Noah Y. Siegel¹, Roland Hafner¹, Michael Bloesch¹, Kristian Hartikainen^{2,4}, Arunkumar Byravan¹, Leonard Hasenclever¹, Yuval Tassa¹, Fereshteh Sadeghi^{3,4}, Nathan Bachelor¹, Federico Casarini¹, Stefano Saliceti¹, Charles Game¹, Neil Sreedhara, Kushal Patel, Marlon Gwira, Andrea Huber¹, Nicole Hurley¹, Francesco Nori¹, Raia Hadsell¹ and Nicolas Heess¹

*Equal contributions, ¹DeepMind, ²University of Oxford, ³Google, ⁴Work done at DeepMind

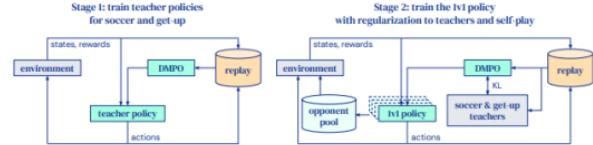
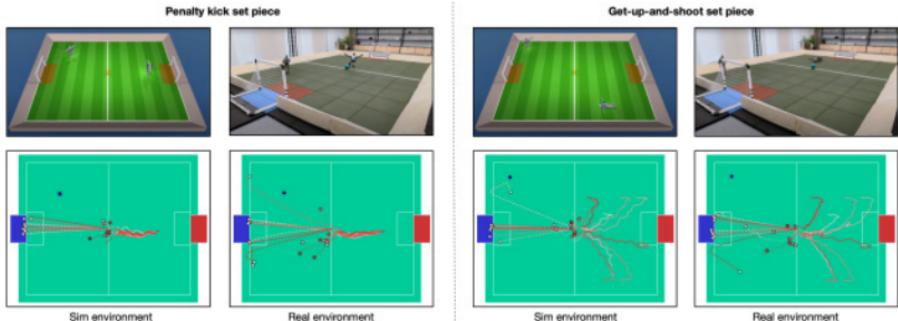


Figure 3 | We trained soccer agents in two stages. In the first stage (left), we train a separate soccer teacher and get-up teacher (Section 3.2.1). In the second stage (right), we distill these two teachers into a single agent that can both get up from the ground and play soccer (Section 3.2.2). The second stage also incorporates self-play: the opponent is uniformly randomly sampled from a pool that consists of policy snapshots from earlier in training. We found that this two-stage approach leads to qualitatively better behavior and improved sim-to-real transfer, compared to training an agent from scratch for the Ivt soccer task.

April23: <https://arxiv.org/pdf/2304.13653.pdf>



Figure 4 | The joint key poses used to train the get-up teacher, extracted from a scripted get-up controller (Robotis, 2023).



Estado del arte: Tencencias Actuales

[nature](#) > [articles](#) > [article](#)

Article | [Open access](#) | Published: 05 October 2022

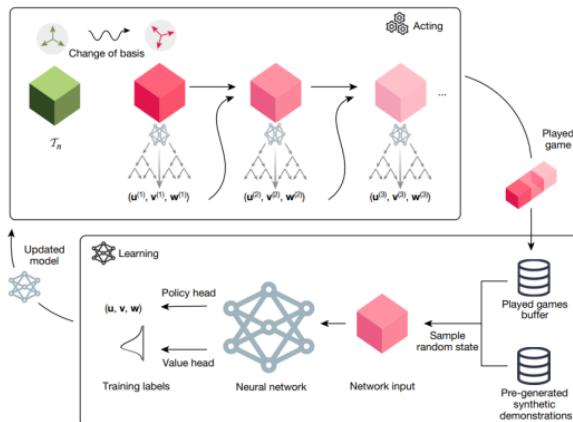
Discovering faster matrix multiplication algorithms with reinforcement learning

Alhussein Fawzi , Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes,

Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis & Pushmeet Kohli

Nature **610**, 47–53 (2022) | [Cite this article](#)

539k Accesses | **89** Citations | **3655** Altmetric | [Metrics](#)



Estado del arte: Tencencias Actuales

<https://huggingface.co/blog/rlhf>

Reinforcement Learning from Human Feedback (RLHF)

Can you explain to me how RL with human feedback works for language models like I am five?

Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

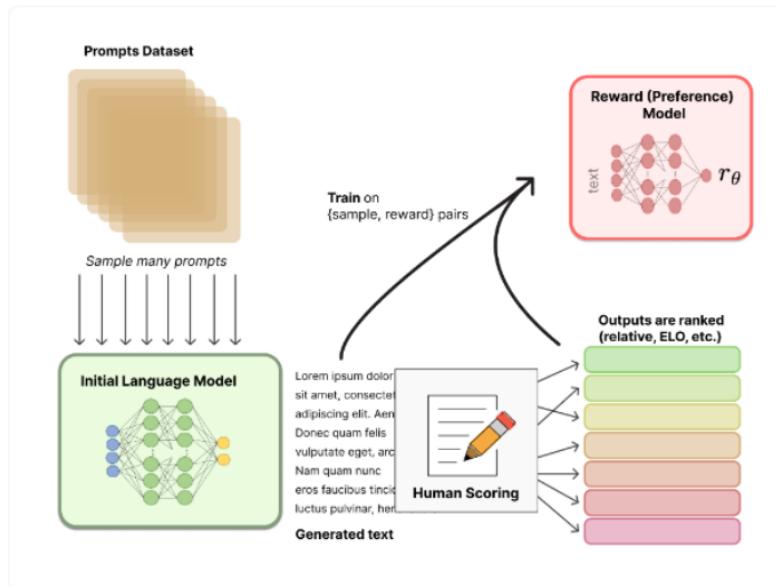
First, Rufus will say something using his language model. For example, he might say "I am a robot."

Then, a human will listen to what Rufus said and give him feedback on whether it sounded like a natural sentence a human would say. The human might say, "That's not quite right, Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I am a machine.'"

Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

The human will listen again and give Rufus more feedback. This process will continue until Rufus can say sentences that sound natural to a human.

Over time, Rufus will learn how to talk like a human thanks to the feedback he receives from humans. This is how language models can be improved using RL with human feedback.



Estado del arte: Tencencias Actuales

Open X-Embodiment: Robotic Learning Datasets and RT-X Models

Open X-Embodiment Collaboration⁰

Absheek Adelkar⁶, Acorn Pooley⁷, Ajinkya Jain¹⁰, Alex Bentley⁷, Alex Hong⁷, Alex Iordan⁷, Alexander Khazatsky¹⁸, Anant Bal¹⁴, Anilika Singh^{7,27}, Anirudh Kandasamy¹, Ankit Patel¹, Ankit Singh¹², Ankit Singh¹², Ankit Singh¹², Bernhard Schölkopf¹³, Brian Ichter¹, Cewu Lu^{1,2}, Charles Xu^{1,2}, Chetlur Jain¹⁸, Cheung Xu^{2,3}, Chengyu Chi¹⁸, Chengzhang Huang¹³, Christine Chan¹, Chuan Pan¹, Chuyun Fu¹, Coline Devri¹, Danny Dries¹, Deepak Pathak², Dhruv Huang¹³, Deter Böhme¹³, Dmitry Kalashnikov²⁰, Dorus Sadigh⁹, Edward Johns⁹, Federico Coletti¹¹, Fei Xia¹, Freck Stulp⁶, Guoyao Zhou¹, Gaurav S. Sukhatme^{20,10}, Gautam Salhotra¹, Ge Yan²³, Giulio Schiavi¹, Gregory Kahn²², Hao Su¹², Hao-Shu Fang¹⁷, Haochen Shi¹⁸, Heng Ben Amor¹, Henrik I Christensen²³, Hiroki Furuta¹, Homer Walk²², Hongjie Fang¹⁷, Igor Mordatch⁷, Ilijas Radosavovic²², Isabel Leal¹, Jacky Liang⁷, Jad Abou-Chakra¹⁵, Jaehyung Kim¹², Jan Peters¹⁹, Jan Schneider¹³, Jasmine Hu¹, Jeannette Bohg¹⁸, Jeffrey Bingham⁷, Jiajun Wu¹⁸, Julian Wu⁸, Juanan Luo²², Jayuan Gu²³, Jie Tan⁷, Jihoo Oh²¹, Jitendra Malik²², Jonathan Tompson¹, Jonathan Yang¹⁸, Joseph J. Lim¹², João Silvério⁶, Junhyeok Han¹², Kanishka Rao⁹, Karl Persch^{22,18}, Karol Myslinski¹, Keegan Go¹⁰, Keerthana Gopalakrishnan¹, Ken Goldberg², Kendra Byrne¹, Kenneth Oslund¹, Kento Kawaharazuka²¹, Kevin Furukawa¹, Krishna Rana¹⁷, Krishnamoorthy Srinivasan¹⁸, Lawrence Yunliang Chen², Lerrel Pinto¹, Liam Tan²², Lionel Ott¹, Lisa Lee¹, Masayoshi Tomizuka¹, Matthew Du¹⁸, Michael Ahn¹, Mingtong Zhang²², Mingyu Ding¹, Murali Kumar Venkata¹, Mohit Sharma³, Moo Jin Kim¹⁸, Naresh Kannan¹, Nicholas Hwang¹, Niharika Deshpande¹, Nishanth Subramanian¹, Noam Di Palo¹, Nur Muhammad Mah Shafullah¹, Oliver Mett²⁴, Oliver Kroemer², Parag R. Singla², Paul Wohlhart¹, Peng Xu¹, Pierre Serenelli¹, Priya Sundaresan¹⁰, Quan Yang⁹, Rafael Rafalov^{7,10}, Ran Tian⁹, Ria Doshi²², Roberto Martín-Martín²⁰, Russell Mendonça², Rutar Shah²⁰, Ryan Hoque²², Ryan Julian¹, Samvel Bystamyan⁶, Sean Kirmani⁷, Sergey Levine²², Sherry Moore¹, Shikhar Bahl¹, Shivina Shaik¹⁰, Stefan Schaal¹⁰, Stefan Welker¹, Stephen Tian¹⁸, Sudheep Dasari¹, Simeon Adeboju²², Simon Guist¹³, Soroush Nasrani²⁰, Tatsuya Harada^{21,16}, Tatsuya Matsushima²¹, Ted Xiao¹, Timbe Yu¹, Tianli Ding⁷, Timur Belkhaija¹⁸, Takanuki Osa²¹, Tatsuya Harada²¹, Tatsuya Matsushima²¹, Ted Xiao¹, Vincent Vanhoucke⁷, Wei Zhan²², Todor Davchev¹², Tony Z. Zhao¹⁸, Travis Armstrong⁷, Trevor Darrell²², Vidal Jain^{7,2}, Vincent Vanhoucke⁷, Wei Zhan²², Wenxian Zhou¹², Wolfram Burgard²², Xi Chen¹, Xiaolong Wang²³, Xinghang Zhu²³, Xuanlin Li²³, Yao Lu¹, Yevgen Chebotar², Yifan Zhou¹, Yifeng Zhou²⁰, Ying Xu¹, Yixuan Wang²³, Younyoung Cho¹², Youngwoon Lee²², Yuchen Cui¹⁸, Yuezhu Wu²³, Yujin Tang^{1,21}, Yuke Zhu²⁰, Yunzhu Iwasa²¹, Yutaka Matsuo²¹, Zhiuo Xu¹, Zichen Jeff Cui¹⁴



Fig. 1: We propose an open, large-scale dataset for robot learning curated from 21 institutions across the globe. The dataset represents diverse behaviors, robot embodiments and environments, and enables learning generalized robotic policies.

Oct23: <https://robotics-transformer-x.github.io/paper.pdf>

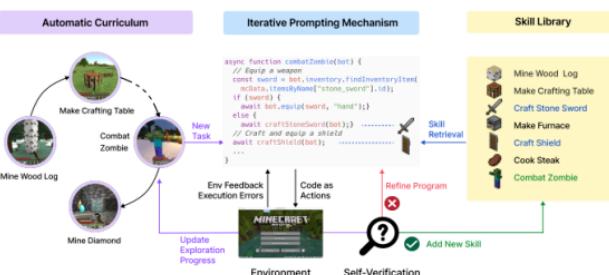
VOYAGER: An Open-Ended Embodied Agent with Large Language Models

Guanzhi Wang^{1,2}, Yuqi Xu³, Yunfan Jiang^{4*}, Ajay Mandelkar^{1*}, Chaowei Xiao^{1,5}, Yuke Zhu^{1,3}, Linsi "Jin" Fan^{1,17}, Anima Anandkumar^{1,2†}

¹NVIDIA, ²Caltech, ³UT Austin, ⁴Stanford, ⁵UW Madison

*Equal contribution [†]Equal advising [‡]Corresponding authors

<https://voyager.minedojø.org>



May23: <https://arxiv.org/pdf/2305.16291.pdf>

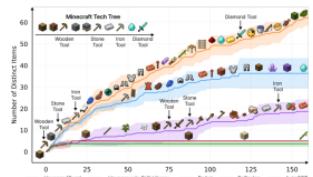


Figure 1: VOYAGER discovers new Minecraft items and skills continually by self-driven exploration, significantly outperforming the baselines. X-axis denotes the number of prompting iterations.

Estado del arte: Tencencias Actuales

SayTap: Language to Quadrupedal Locomotion

Yujin Tang
yujintang@google.com
Google DeepMind

Wenhuo Yu
magicmelon@google.com
Google DeepMind

Jie Tan
jietan@google.com
Google DeepMind

Heiga Zen
heigazen@google.com
Google DeepMind

Aleksandra Faust
sandrafaust@google.com
Google DeepMind

Tatsuya Harada
harada@mi.t.u-tokyo.ac.jp
The University of Tokyo

Abstract: Large language models (LLMs) have demonstrated the potential to perform high-level planning. Yet, it remains a challenge for LLMs to comprehend low-level commands, such as joint angle targets or motor torques. This paper proposes an approach to use foot contact patterns as an interface that bridges human commands in natural language and a locomotion controller that outputs these low-level commands. This results in an interactive system for quadrupedal robots that allows the users to craft diverse locomotion behaviors flexibly. We contribute an LLM prompt design, a reward function, and a method to expose the controller to the feasible distribution of contact patterns. The results are a controller capable of achieving diverse locomotion patterns that can be transferred to real robot hardware. Compared with other design choices, the proposed approach enjoys more than 50% success rate in predicting the correct contact patterns and can solve 10 more tasks out of a total of 30 tasks. (<https://saytap.github.io>)

Keywords: Large language model (LLM), Quadrupedal robots, Locomotion



Jun23: <https://arxiv.org/abs/2306.07580>

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Distintos enfoques del aprendizaje por refuerzo

Una posible clasificación que podemos hacer cuando hablamos de aprendizaje por refuerzo está relacionado con el entorno donde se ejecutará la solución desarrollada.

En este sentido tenemos dos grupos principales:

Entorno puramente digital - Ámbito de investigación

Entorno Digital-a-Real - Ámbito de investigación/aplicación

Vamos a entrar en detalle con cada enfoque.

Distintos enfoques del aprendizaje por refuerzo



Entorno puramente digital - Ámbito de Investigación

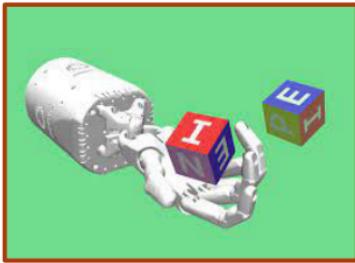
Este es el entorno típico que nos encontraremos en la mayoría de ejemplos a día de hoy.

Trabajaremos en un entorno totalmente simulado y controlado para poder probar nuevos algoritmos, nuevos modelos, etc.

Este enfoque es fundamental para contrastar hipótesis teóricas y comprobar su posible aplicación en el mundo real.



Distintos enfoques del aprendizaje por refuerzo



Entorno Digital-a-Real - Ámbito de Investigación

En este caso tenemos una situación parecida al enfoque anterior aunque ahora usaremos el conocimiento extraído en un entorno real.



Un caso común es la robótica, en el que se diseña toda la simulación en un entorno digital para usar la solución obtenida en brazos robóticos reales.

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Retos actuales y de futuro

El principal reto en el que el aprendizaje por refuerzo se centra actualmente es encontrar **casos de uso** en los que se pueda aplicar de una manera beneficiosa.

Es verdad que en los últimos años se han ido desarrollando soluciones con aplicaciones reales en ámbitos como la robótica, control automático, etc., pero **todavía no tienen un impacto de negocio como para tomar a estas soluciones más en “serio”**.

Para la mayoría de estos **casos hay soluciones tradicionales que funcionan muy bien** y no necesitan de la complejidad que necesita el aprendizaje por refuerzo.

Retos actuales y de futuro

Una de las complejidades que presentan las soluciones basadas en aprendizaje por refuerzo, y que veremos en las siguientes sesiones, es **la cantidad de hiperparámetros** que se necesitan configurar para que una solución funcione.

Y no sólo eso, **¿qué significa que una solución funcione?**

Veremos en siguientes sesiones conceptos para poder discutir sobre esta pregunta, pero lo que está claro es que el concepto de que algo sea útil o no sigue estando muy sesgado a los ojos de quien lo mire (como es normal por otra parte).

Retos actuales y de futuro

A pesar de tener en la actualidad mucho dominio de la materia y conociendo muy bien todos los conceptos, **el control sobre el proceso de aprendizaje es vago.**

En otras disciplinas de la inteligencia artificial, como por ejemplo en la aplicación de técnicas de Deep Learning, tenemos una metodología que, aunque sea empírica, podemos controlar y medir cómo el sistema evoluciona y si lo está haciendo de forma favorable o no.

En el caso del aprendizaje por refuerzo no se dan las mismas circunstancias. **Dependiendo del algoritmo, del reto y de la información disponible, la forma de medir la bondad de una solución difiere** y por tanto le añade una complejidad extra a su diseño.

Retos actuales y de futuro

Aun así, todos los grupos de investigación comparten la idea de que el **potencial que tiene este enfoque de la inteligencia artificial** es *infinito*.

La mayoría de los problemas que ahora mismo podemos solucionar son problemas más cercanos a la automatización que al razonamiento y adaptación que una inteligencia real permitiría.

Es aquí donde también se presenta un gran reto: ser capaz de encontrar la madurez suficiente para afrontar verdaderos problemas de Inteligencia Artificial siguiendo esta metodología.

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

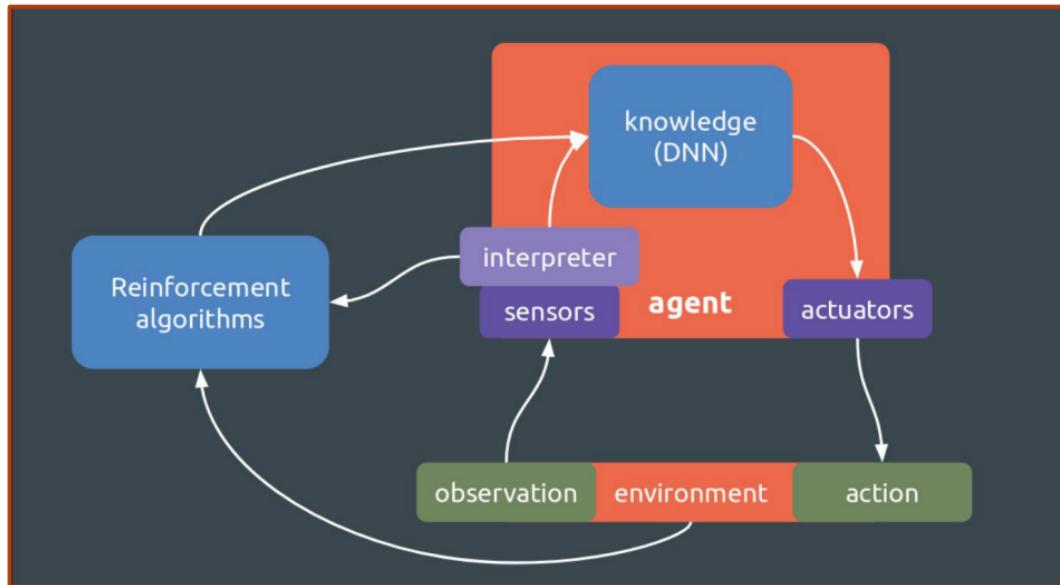
Retos actuales y de futuro

Vista general

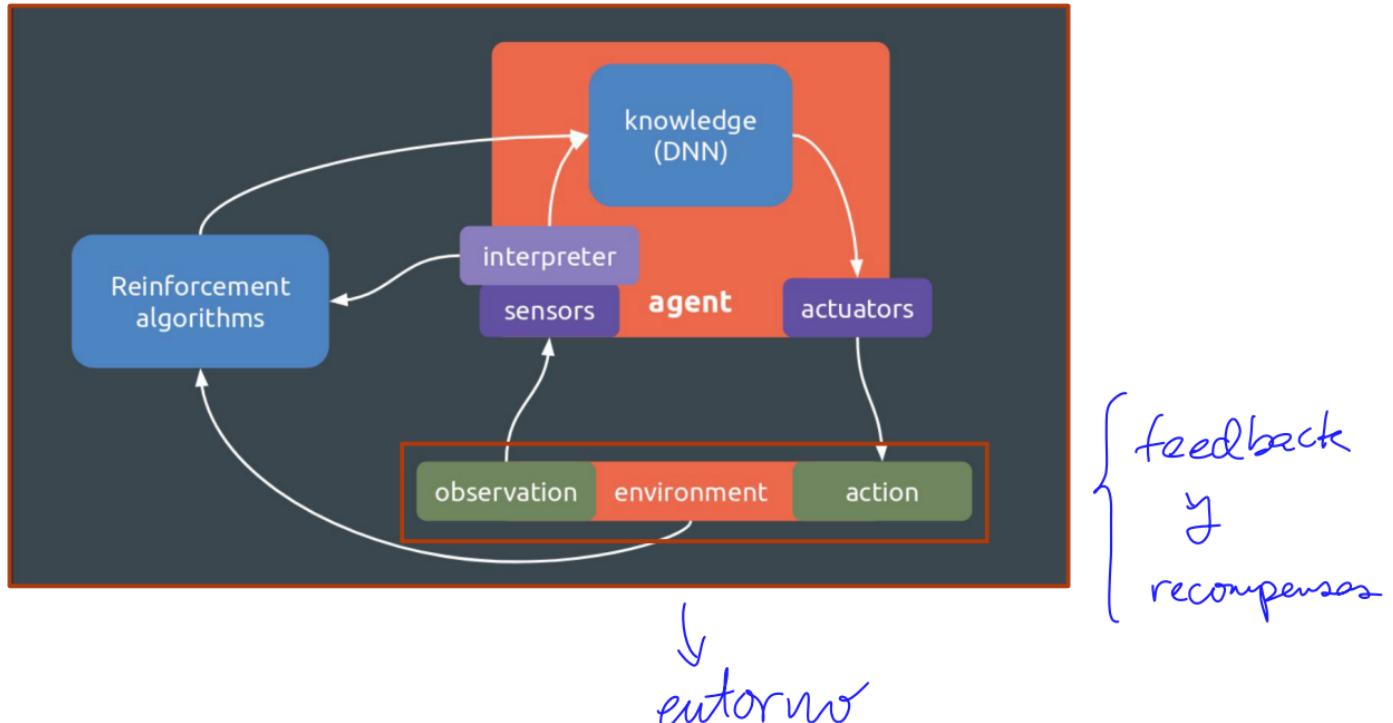
Conclusiones

Bibliografía recomendada

Vista general

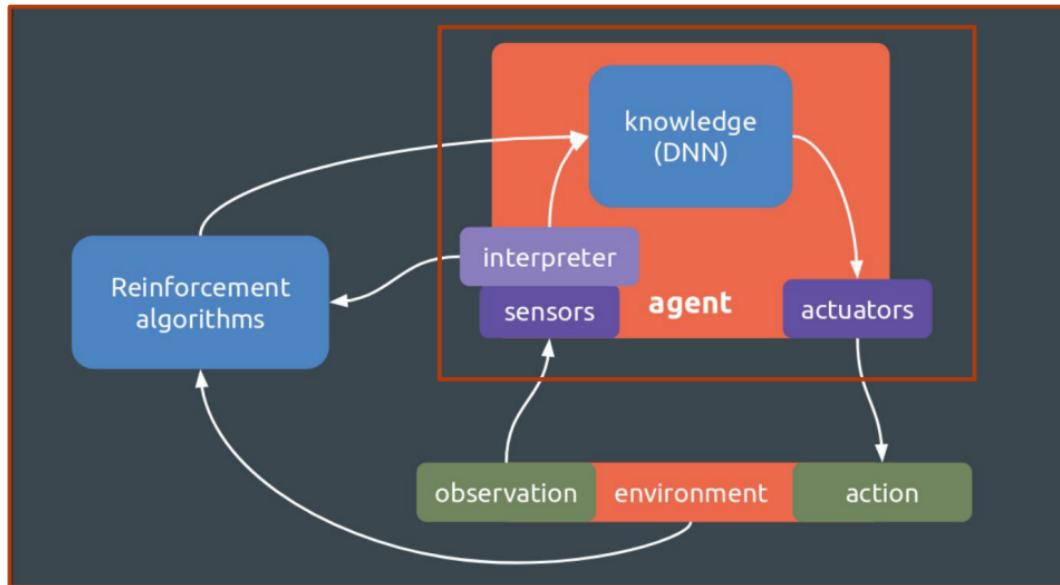


Vista general

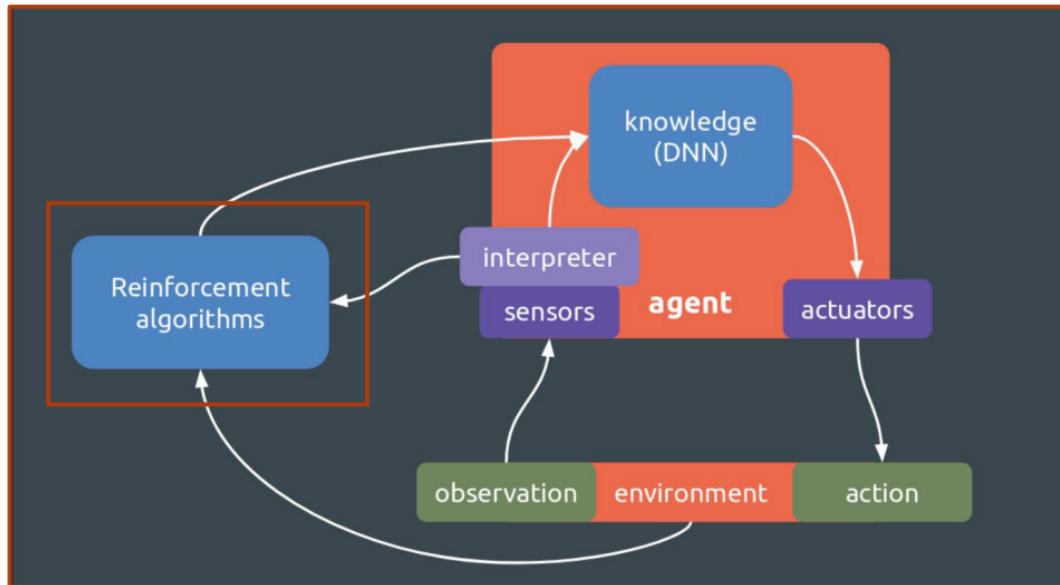


Vista general

↑ agente



Vista general



Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Conclusiones

- El aprendizaje por refuerzo es una de las **ramas más prometedoras dentro de la inteligencia artificial**.
- El proceso de aprendizaje **se asemeja a la idea de aprendizaje que tenemos los seres humanos**.
- Aunque es una rama muy prometedora, todavía **están en sus primeros pasos**. No está totalmente asentada y **sus soluciones no se pueden aplicar en todos los campos y/o dominios**.
- La unión de modelos de **Deep Learning con algoritmos de aprendizaje por refuerzo** ha sido la mezcla perfecta para abrir todo el conjunto de posibilidades con tanto potencial que tenemos a nuestro alcance actualmente.

Índice

¿Qué es el aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

“Reinforcement Learning: An introduction”, Sutton y Barto:
<http://incompleteideas.net/book/bookdraft2017nov5.pdf>

Contenidos y documentación de OpenAI SpinningupRL:
<https://spinningup.openai.com/en/latest/>

→ casi todo lo que se va a ver en el curso

HuggingFace Courses:

<https://huggingface.co/learn/deep-rl-course/>

→ cosas interesantes, robótica y demás

Modelos fundamentales: dado un ámbito entrena en muchas tareas.

08MIAR-Aprendizaje por refuerzo

Sesión 2 – Conceptos y Terminología

Curso Abril 23/24

The logo consists of the lowercase letters "viu" in white, bold, sans-serif font, centered within a solid orange circle.

viu

Universidad
Internacional
de Valencia

De:

 Planeta Formación y Universidades

Índice

Vista general

Conceptos básicos

Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Índice

Vista general

Conceptos básicos

Conceptos avanzados

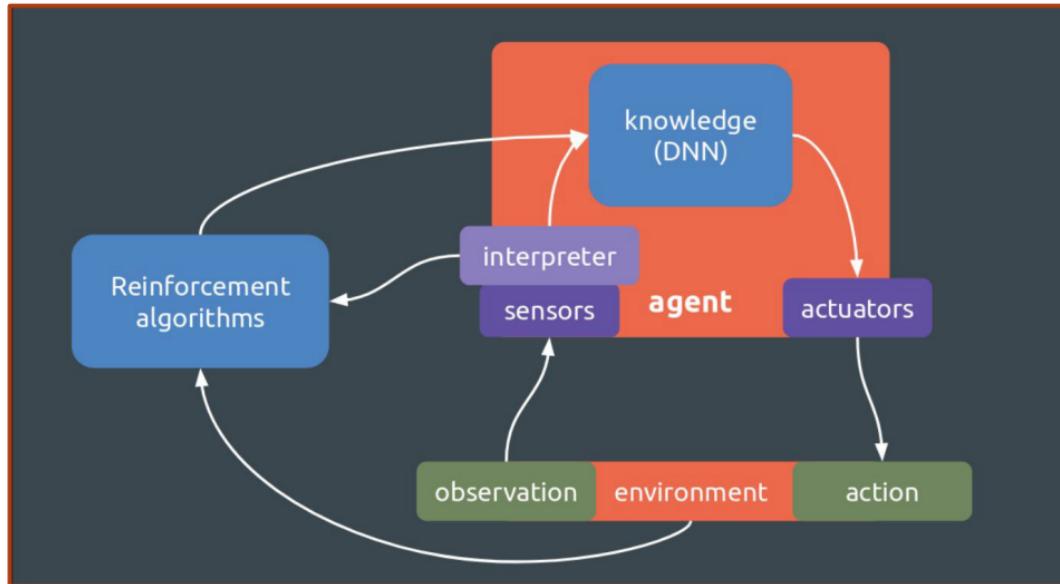
Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Vista general



Índice

Vista general

Conceptos básicos

Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Conceptos básicos

Para entender bien los algoritmos que veremos y cómo podríamos aplicarlos a distintos problemas, necesitamos conocer previamente los conceptos y la terminología que usaremos durante el desarrollo.

En el caso del aprendizaje por refuerzo, la terminología es muy característica y sus conceptos muy cercanos a las matemáticas y estadística que subyace en este tipo de problemas.

En esta sesión entenderemos los conceptos de una manera intuitiva y a alto nivel, así como la representación de cada uno para explotar su uso más adelante.

En las sesiones sobre los algoritmos será donde entraremos más en detalle, desde un punto de vista formal y analítico, sobre los conceptos que veremos a continuación.

Conceptos básicos

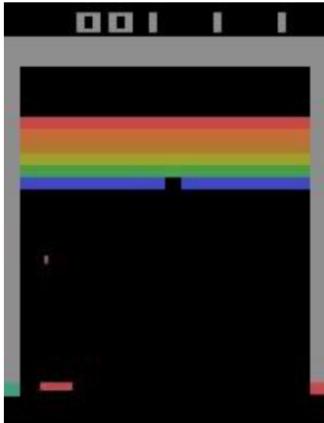
En nuestro caso, los retos en los que trabajaremos estarán basados en videojuegos ya que conectan muy bien con el poder aplicar y probar nuestros algoritmos sobre ellos. Además, los videojuegos están definidos por reglas que rigen cómo se pueden comportar nuestros agentes en los mismos.

Por otro lado, hay que tener en cuenta que para poder demostrar empíricamente que nuestras soluciones *funcionan*, necesitamos controlar en todo momento la simulación, de ahí que los videojuegos sean una buena elección.

Para entender más fácilmente cada uno de estos conceptos vamos a usar como base un entorno digital común para todos, el videojuego de ***Breakout de Atari***.

Entorno

esta imagen del entorno en sí misma no es el entorno completo además de esto hay cierta lógica que subyace. La gravedad de cómo cae la bola, como se puntuá, los tiempos disponibles, cuándo acaba la simulación, ... Es algo que va mucho más allá.



El entorno es el mundo donde el agente vive e interactúa con los elementos de su alrededor.

El entorno no sólo se compone de los elementos visuales de la simulación, también de la lógica que subyace y que define cómo se puntuá, tiempos disponibles, etc.

El entorno es uno de los actores principales en nuestros problemas ya que es la simulación sobre la que nuestro agente aprenderá.

Entorno

Todos los conceptos que abordaremos a alto nivel tienen una relación directa con una parte que también nos interesa, la analítica/tecnológica.

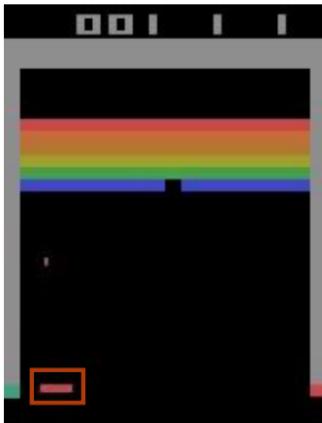
Iremos viendo cómo se representa cada uno de estos conceptos desde un punto de vista más cercano al (*pseudo*)código para poder trabajar con ellos.

En el caso del entorno, normalmente encontraremos un wrapper de la simulación. Al ejecutarlo, o al interactuar con el agente, nos irá devolviendo toda la información disponible.

```
# Python code  
  
env = wrapper.open("Atari.rom")  
  
env.action_space  
  
env.step(action)  
  
env.internal_variables
```

Agente

El agente NO es la barra, es el jugador.
La barra es parte del entorno, es un actuador.



El agente es la otra pieza fundamental de nuestra solución.

Un agente es una entidad que interactúa con elementos del entorno (que estén disponibles en el momento de la interacción) para llegar a un objetivo.

Generalmente, este objetivo es maximizar una recompensa (como en el caso de la mayoría de juegos).

El agente será la entidad inteligente que irá aprendiendo durante las ejecuciones del experimento.

Agente

El agente será la pieza de nuestras soluciones en la que recaerá más responsabilidad por nuestra parte.

Comúnmente nos encontraremos una clase que sea la que englobe todas las funciones y comportamientos que el agente puede realizar durante su aprendizaje.

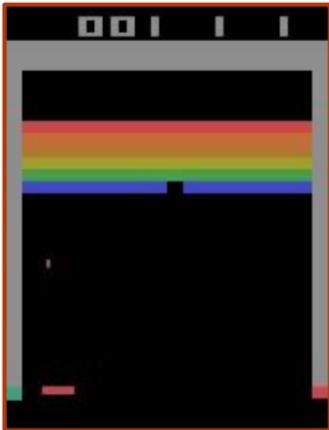
Será en el agente donde se relaciona la información que proviene del entorno con el módulo de aprendizaje que hayamos definido (en este curso, una red neuronal).

*La parte de actualización
de la red neuronal no está
aquí, se encuentra en otro
bloque bien diferenciado.*

Python code

```
class OurAgent:  
    def __init__(...):  
        ...  
    def select_action(...):  
        ...  
    def preprocess_obs(...):  
        ...  
    def update_states(...):  
        ...
```

Observación



Una vez tenemos el *feedback-loop* entre nuestro entorno y nuestro agente, ¿cómo se realiza el aprendizaje?

En primer lugar tendremos observaciones. Cada vez que el agente toma una decisión, esa decisión afecta al entorno. Cada fotografía del entorno tras una decisión del agente es lo que podemos entender como observación.

Cada observación contiene toda la información disponible en el entorno en ese momento.

Observación

Normalmente, sobre todo en el ámbito de los videojuegos, las observaciones van a tener la forma de un *array* o lista de elementos.

En nuestro reto, cada uno de estos elementos será un píxel de la pantalla.

Dependiendo de la información disponible, podemos encontrar información extra en estructuras de datos o en otros campos que el entorno nos devuelve.

[Examen]

- ¿Está bien esta línea de código?
`state, info, reward, done = env.step(...)`

- R) No, lo que devuelve un step sobre el entorno es una observación, no el estado

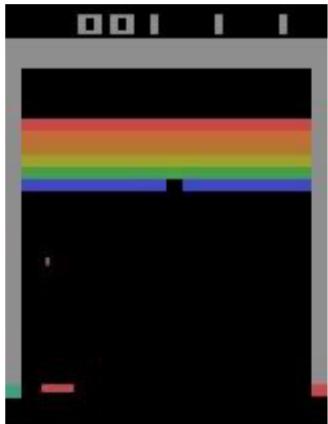
Python code

obs, info, reward, done = env.step(...)

```
# obs
[[[255, 255, 255, 255],
 [120, 123, 122, 123],
 ...,
 [110, 110, 115, 115],
 [255, 255, 255, 255]]]
```

aquí se
le pasa una
acción

Estado (Observación pre-procesada y lista para ser consumida por el agente, o sea, para ser procesada por la red)



Cuando el **entorno nos devuelve la observación** en un momento determinado de la ejecución, la información que el agente obtiene es *raw*, es decir, es tal y como el entorno la genera.

Dependiendo de cómo se defina el módulo de aprendizaje del agente, y de la disponibilidad computacional que tengamos, esta observación necesita un tratamiento.

Esto es lo que se conoce como **estado**. **El estado es la observación lista para ser consumida por nuestro agente y poder tomar decisiones.**

Otra posible interpretación del concepto de **estado** es la relación directa con la definición formal mediante cadenas de Markov.

Estado

En el caso del estado, partimos de **la información que tenemos a partir de la observación**.

A esta observación le aplicamos todo lo necesario en cuanto a **pre-procesamiento, redimensionado** de los datos, etc., *RGB a Gris (para reducir la carga comput.)*

Una vez aplicado este pre-procesamiento, la nueva estructura de datos estaría lista para que el agente la pudiera usar adecuadamente.



```
# Python code
state = resize(obs)
state = preprocess_and_rgb(State)

# state
[[[1.0, 1.0, 1.0, 1.0],
  [0.5, 0.5, 0.5, 0.5],
  ...
  [0.3, 0.4, 0.3, 0.3],
  [1.0, 1.0, 1.0, 1.0]]]
```

Acciones



La activación de la capa de salida de la red puede ser **Softmax** (la mejor en este caso) o **Sigmoid** (si regularizas muy bien la red)

Proceso de exploración: podríamos tomar varias estrategias para seleccionar la acción. Ejemplo:

- > proba = [0, 0, 0.55, 0.45]
- { selection = argmax (proba)
- selection = sampling (proba) → podría salir la de 0.45

Toda **decisión** que hemos ido comentando que el **agente** va a **realizar** en el **entorno** es lo que se conoce como **acción**.

El agente siempre tiene a su disposición una lista de acciones disponibles, que puede ir cambiando conforme la simulación vaya evolucionando.

Python code

action = agent.select_action(state)

action
[0, 0, 0, 1] # Move left

env.step(action)

previamente
pre-procesado
(estado)

1 para acciones que se quieren realizar
0 para las acciones que no quiero realizar

Acciones

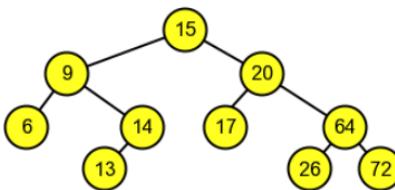
En el caso del estado, **partimos de la información que tenemos a partir de la observación.**

A esta observación le aplicamos todo lo necesario en cuanto a **pre-procesamiento, redimensionado** de los datos, etc.

Una vez aplicado este pre-procesamiento, la nueva estructura de datos estaría lista para que el agente la pudiera usar adecuadamente.

```
# Python code  
  
action = agent.select_action(state)  
  
# action  
[0, 0, 0, 1] # Move left  
  
env.step(action)
```

Espacio de estados y acciones



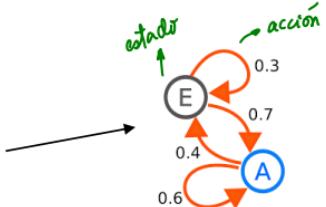
Tanto las acciones como los estados tienen características matemáticas muy interesantes. **Al conjunto de todos los estados y acciones disponibles se le conoce como espacio.**

Son estos espacios los que producen que el aprendizaje por refuerzo tenga un fuerte componente de **optimización** y que, dependiendo del problema, una búsqueda óptima en estos espacios es la parte crucial de nuestras soluciones.

Matemáticamente, el espacio de estados se puede ver como una **cadena de Markov (MDP)**, en el que cada estado va cambiando en el tiempo.

En el caso de las acciones, la búsqueda óptima está más relacionada con técnicas meta-heurísticas, *back-tracking* o búsqueda en árbol. Estas decisiones también son muy dependientes del problema que estemos modelando.

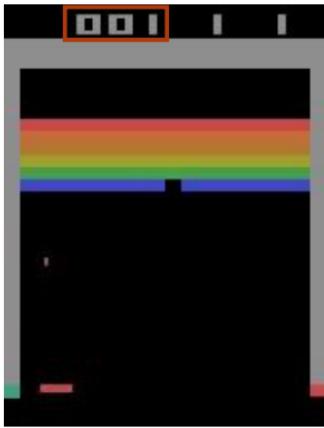
estocasticidad
aleatoriedad



Asumimos:

MDP: en t tienes toda la información necesaria para pasar estimar la acción en $t+1$. (el paso al siguiente estado únicamente depende de la información del estado actual, no de donde hayas venido previamente)

Recompensa



Una vez que el agente toma una acción, ¿cómo sabe si la acción que ha tomado es buena o mala?

Para guiar el aprendizaje y analizar qué acciones son buenas o malas usamos la recompensa. La recompensa es el feedback que nos devuelve el entorno para evaluar cómo lo está haciendo el agente.

Por ejemplo, si jugamos a algún videojuego, la recompensa sería la puntuación del mismo. Pero también tenemos otras alternativas como: medir la barra de vida, cuántos continúes nos quedan, velocidad a la que jugamos, etc.

Recompensa

La recompensa es un valor que nos devuelve el entorno. Dependiendo del entorno en el que estemos trabajando, este valor puede variar aunque siempre intenta ser fiel al comportamiento básico de la simulación.

El punto de complejidad respecto a la recompensa se produce cuando definimos nuestras propias funciones. Por ejemplo, dependiendo del experimento nos puede interesar potenciar ciertas acciones o incluso usar también un castigo para puntuar situaciones adversas.

```
# Python code

obs, info, reward, done =
env.step(...)

# reward
0.8

# Define a reward taking into
account
# the game counter
def my_reward(...):
    ...
    ...
```

Iteración (Step)

Congelación de todos los componentes de nuestra ejecución en un instante de tiempo.
↳ Observación, estado, acción tomada, ... en total.
Una vez tomada la acción se pasa al siguiente step.

Cuando decíamos que la observación era una *fotografía* del entorno en un momento determinado, esa definición de momento es lo que se conoce como iteración o *step*.

La iteración se produce cuando congelamos todos los componentes de nuestra ejecución en un instante de tiempo.

Este componente también es muy útil para estimar la ejecución y bondad de ciertos algoritmos ya que será el valor con el que mediremos tiempos y velocidad de convergencia.

Episodio

Cuántas simulaciones se van de llevar a cabo.

Por último, tenemos el concepto de episodio. El **episodio se desarrolla desde el comienzo de una ejecución hasta que llegamos a game over.**

- Se acaba la simulación
- Se fija un numero T máximo de iteraciones en el episodio. → sobre todo en entrenamiento

Dependiendo del entorno podemos encontrarnos con que el episodio se corresponda con una fase o reto, o que se desarrolle en el tiempo hasta que termine de alguna manera.

El episodio está compuesto de un conjunto de iteraciones.

Episodio e iteración

Si queremos interpretar las iteraciones en nuestro código, nos encontramos con una variable para controlar el número de ejecuciones de nuestra simulación.

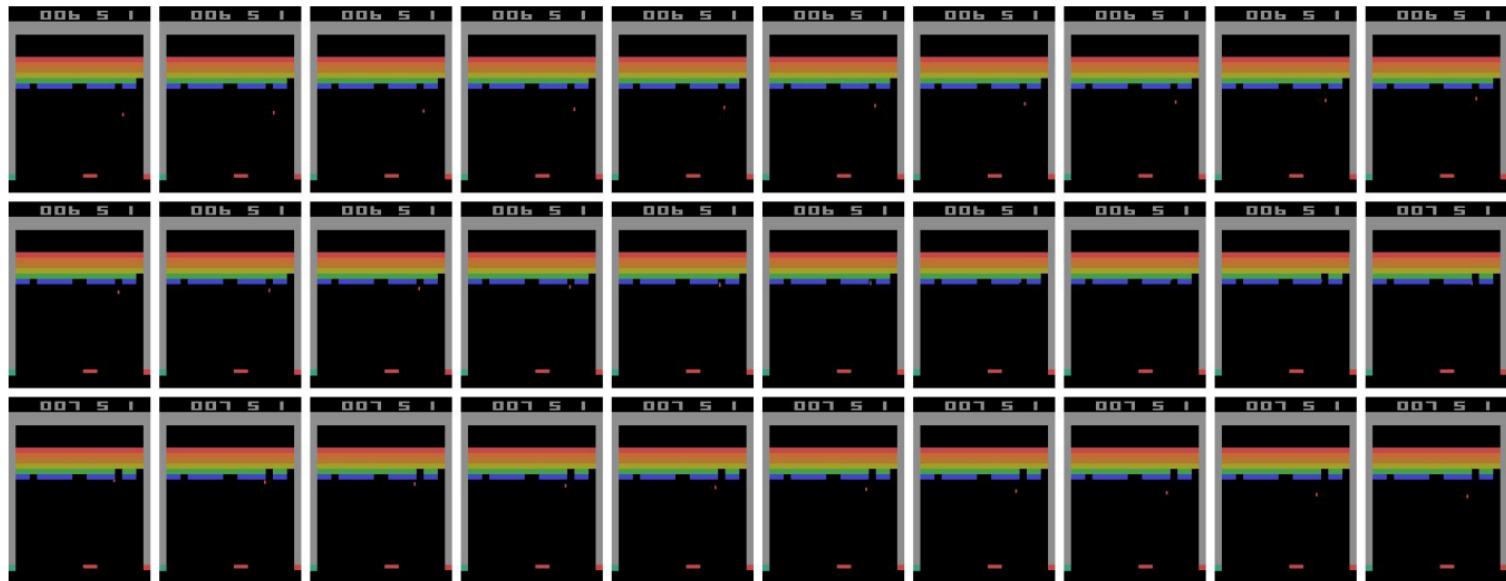
Los **episodios** están un nivel por encima para definir el **número de pasadas globales (o partidas) que nuestro agente va a realizar**.

Ambos conceptos los podemos ver como **dos bucles anidados** para controlar desde lo más atómico hasta el nivel superior. Normalmente, el bucle que controlamos es el de los episodios, siendo el bucle de las iteraciones interno en la propia simulación.

2 bucles.

```
# Bucle de episodios
for i_episode in episodes:
    step=0
    # Bucle de iteraciones
    while run:
        step += 1
        ...
        ...
        if game_over:
            run=False
        if step <=T:
            run=False
```

Episodio e iteración



Índice

Vista general

Conceptos básicos

Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

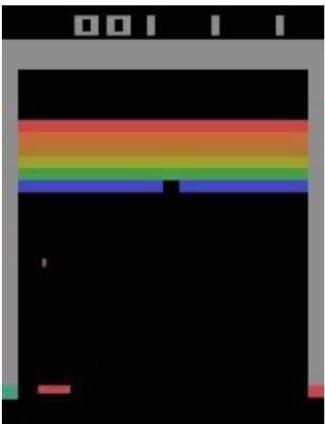
Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Estrategia (Policy)

Ej: Si vamos por un camino, pero al final llegamos a un precipicio y caemos y perdemos todo.
No estamos siguiendo una buena estrategia, tendremos que tener en cuenta la recompensa esperada a futuro. Cuando hablamos de maximizar la recompensa, hablamos de la recompensa esperada a futuro.



Cuando hablamos de que el agente aprende sobre un entorno, en realidad nos referimos a que el agente encuentra una estrategia para maximizar un objetivo.

Esta **estrategia** será la que nos diga **qué acción tomar en un estado**. Nuestra **meta** es que nuestro **agente aprenda la estrategia óptima**.

La búsqueda de la mejor estrategia posible es la base del aprendizaje por refuerzo.

En inglés se usa el término *policy* por lo que iremos intercambiando ambos términos durante nuestro curso.

Estrategia (Policy)

Desde un punto de vista programático, la **estrategia** es el concepto ligado a nuestro modelo de Deep Learning.

Va a ser nuestro modelo el que se encargue de **modelar las acciones que se toman dependiendo de los estados en los que el agente se encuentra**.

Dependiendo del algoritmo podemos tener distintos comportamientos en la estrategia.

Modelos que se usan:

- Basados en Q
- Policy Gradients.

Python code

```
action = agent_model(state)
```

Q models → **política voraz** → determinista.
`action_selected = np.argmax(action)`

PG models → **Sampling** → estocástico.
`action_selected = distribution(1, prob=action)`

estrategia en 2 pasos

- 1.- output de la red
prob. de acciones.
- 2.- selección de acción

Transiciones

En cada **paso de un estado a otro** se produce lo que llamamos transición. Las transiciones están compuestas de una tupla de elementos:

[Estado, acción, recompensa, Siguiente estado]

(s, a, r, s')



Con esta **tupla mínima** podemos modelar el **paso de un estado a otro con información suficiente como para que el agente vaya aprendiendo**. Dependiendo del algoritmo usado, esta tupla puede contener otros elementos que añadan información útil al agente.

→ podría contener más información, pero como mínimo tiene que contener esta

Transiciones

Lo más importante para las transiciones es seleccionar una estructura de datos adecuada.

Podemos seleccionar desde la más simple, una **lista de elementos**, a usar **colas de mensajería** que se encarguen de procesar y gestionar todas las transiciones.

Lo más importante es mantener la misma estructura durante todas las ejecuciones, para que no afecte desde un punto de vista computacional.

```
# Python code

transition = (state, next_state,
               reward, action)

# Lists
transitions.append(transition)

# Queue
transitions = Queue()
transitions.push(transition)
```

Experiencia

(lista o cola de transiciones)

Dependiendo del algoritmo que usemos nos interesa extraer un tipo de información u otra.

Esto es lo que se conoce como experiencia, porque va directamente relacionado con la información pasada que el agente tiene a su disposición para ir aprendiendo.

En su forma más básica se almacena la información de las transiciones, pero dependiendo del algoritmo esa información puede variar.

Experiencia = [Transiciones + Extras] en el intervalo $[t_0 - T \text{ a } t_0]$

(s, a, r, s)

$(a(t+1), Q\text{-values}, \dots,)$

Experiencia

Al igual que con las **transiciones**, la experiencia también necesita de una estructura de datos adecuada.

En este caso, es común usar **colas de un tamaño fijo** ya que, como veremos en los algoritmos, la experiencia está relacionada con un conjunto fijo de número de transiciones para ir entrenando a nuestro agente.

Cola tipo FIFO normalmente.

```
# Python code

class Experience():
    def __init__():
        # Queue with 120 elements
        self.data = Queue(120)

    def add(elem):
        self.data.push(elem)

    def get():
        self.data.pop()

transition = (state, next_state,
              reward, action)
experience = Experience()
Experience.data.add(transition)
```

Exploración



EXPLORATION

Playing the other machines to see if any pay out more.

Proceso por el que el agente va adquiriendo experiencia a partir de prueba-y-error en el entorno.

Se puede explorar de forma aleatoria al comienzo de una simulación, para empezar el almacenamiento de las transiciones y así comenzar el aprendizaje.

Durante entrenamiento, la exploración está controlada por una variable aleatoria que va disminuyendo en el tiempo. Dependiendo del valor de esta variable vamos decidiendo si la acción que toma el agente es aleatoria o no.

Exploración

Como hemos indicado, el proceso de **exploración** está controlado por una **variable externa** para tomar una acción de una manera **aleatoria** o siguiendo la estrategia que se está aprendiendo.

Los valores típicos para esta variable van **desde 0.99** (acción siempre aleatoria) **hasta 0.05** (acción la mayoría de veces usando la *policy* aprendida).

Justo durante **el intervalo de tiempo entre estos dos valores será nuestro tiempo de exploración.**

Se dejará un valor > 0 incluso en deployment
(*exploración*)

```
# Python code  
  
epsilon = 0.99  
  
# During the simulations  
(...)  
  
random_number = np.random(...)  
  
if random_number > epsilon:  
    # policy action  
else:  
    # random action  
  
epsilon -= 0.01
```

Explotación



EXPLOITATION

Playing the machine that (currently) pays out the most.

Una vez que el agente ha **explorado un tiempo suficiente** (esto es totalmente dependiente del problema), comienza el proceso de **explotación**.

En esta parte, **el agente ha aprendido cómo tomar decisiones** a partir del proceso de **aprendizaje**, sin necesidad de explorar situaciones nuevas.

Es común dejar siempre un grado de **aleatoriedad** para la decisión de qué acción tomar, aunque esa **probabilidad** es muy pequeña.

Explotación

Como hemos visto en la diapositiva anterior, en **explotación consideramos que el agente ha aprendido lo suficiente**, por lo que las **acciones** que se seleccionan las obtenemos **directamente de nuestra policy**.

Para ello obtenemos las acciones directamente usando nuestro modelo, que es el encargado de aproximar la **mejor policy aprendida**.

Este estado es la situación típica cuando desplegamos nuestra solución.



<https://huggingface.co/learn/deep-rl-course/unit1/exp-exp-tradeoff>

Índice

Vista general

Conceptos básicos

Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Clasificación de problemas de aprendizaje por refuerzo

En la primera sesión vimos distintos enfoques a la hora de clasificar los problemas y retos que nos podemos encontrar cuando queremos aplicar técnicas de aprendizaje por refuerzo. La mayoría de estos enfoques estaban definidos a alto nivel.

Si bajamos a detalle, podemos analizar los problemas de aprendizaje por refuerzo desde el punto de vista del comportamiento del agente, de la información que tenemos a nuestra disposición, etc. Este análisis nos permite saber qué algoritmo es el más adecuado acorde al reto que queramos solucionar.

En las siguientes diapositivas nos centraremos en dos **clasificaciones básicas**:

- **Basada en modelo** - ¿Tenemos un modelo de nuestro entorno?
- **Basada en estrategia** - ¿Cómo gestionamos la memoria?

Hay más, pero para nuestro objetivo estas dos son las más importantes.

Basados en modelo

model free: no conocemos por qué nos vienen las recompensas ni cómo nos vienen, vamos un poco a ciegas

1) based: ej. juegos de tablero; podemos añadir conocimiento de búsqueda en árbol y heurísticas para ayudarnos. Conocemos las dinámicas del entorno.

Cuando nos referimos a "algoritmos de aprendizaje por refuerzo **basados en modelo**" podemos encontrar dos posibilidades: **model free** y **model based**.

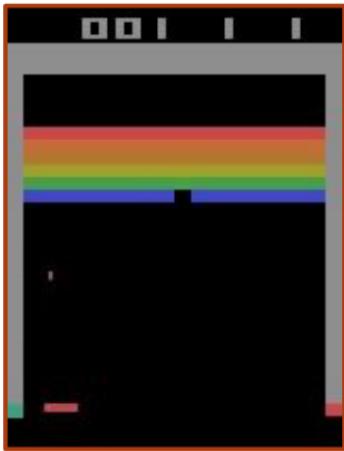
→ más determinista

En **model free** la simulación no conoce un modelo para "predecir" siguientes estados en el entorno. Es el caso típico que encontramos en la mayoría de los ejemplos hoy en día, como es el caso en casi todos los videojuegos. El comportamiento de los entornos en este tipo de simulaciones no es determinista, de ahí que no se pueda conocer un modelo.

Al no poder anticipar siguientes estados, la estrategia que busca el agente es dependiente sólo del estado actual en el que se encuentra.

Podría haber un modelo auxiliar que sirve de model free a model based.

Basados en modelo



model free

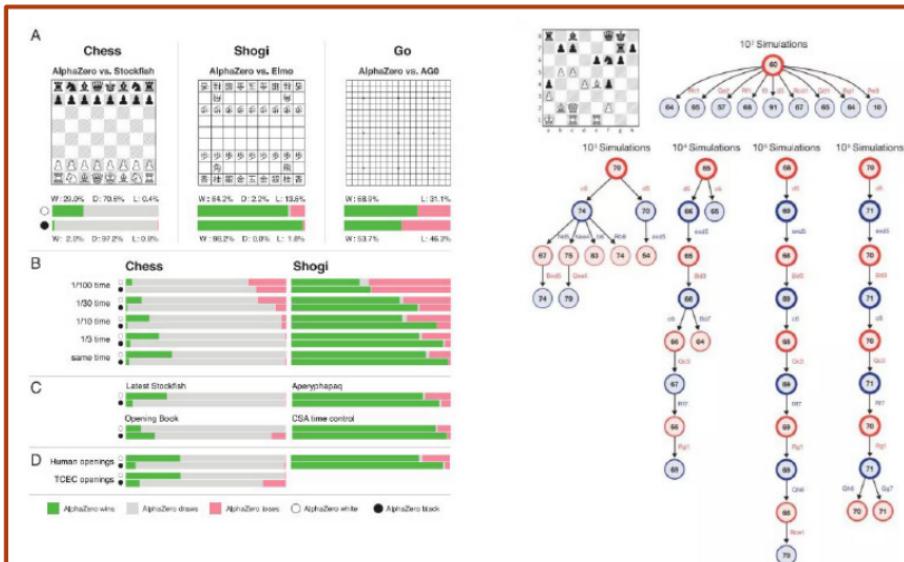
Basados en modelo

En el caso de *model based* sí que conocemos un modelo de cómo se comporta el entorno en relación con las posibles acciones que el agente tome durante la ejecución.

Al tener el conocimiento de este modelo, es típico combinar los algoritmos de aprendizaje con refuerzo con otras soluciones de inteligencia artificial como técnicas de planificación y búsquedas óptimas con objetivo de mejorar el proceso de aprendizaje.

Por ello, en este tipo de enfoque el agente no sólo se encarga de aprender la estrategia óptima sino que también esta estrategia va acompañada de algoritmos de optimización para evitar los costes computacionales de hacer búsquedas *por fuerza bruta*.

Basados en modelo



model based

Basados en estrategia

(La estrategia hace referencia a cómo gestionar la memoria.)

On policy: sólo guardan en la memoria iteraciones con la versión actual del modelo. Hacer una tirada entera de un episodio y optimizar el modelo únicamente con la información de ese episodio. Se resetea la memoria tras cada episodio. (más populares a día de hoy)

Una vez vista la primera clasificación, ahora es el turno de la otra opción: por **estrategia**. Evidentemente, cuando hablamos de estrategia nos referimos a nuestra definición de estrategia o **policy**. Al igual que con la clasificación basada en modelos, tendremos dos posibles conjuntos para identificar a los algoritmos de aprendizaje por refuerzo.

El primero de ellos es el que se conoce como **on policy**. Durante el proceso de aprendizaje, la estrategia que nuestros agentes siguen puede ir cambiando en el tiempo. Con **on policy** nos referimos a **que el agente sólo puede usar la experiencia pasada de una estrategia específica**.

Normalmente este tipo de aprendizaje se caracteriza por **más tiempo de entrenamiento** para **encontrar una solución óptima** así como de **más varianza en los datos** a la hora de entrenar a nuestro agente. En este tipo de aprendizaje encontramos muchos de los algoritmos de la familia de **Policy gradients**.

Basados en estrategia

Off policy: realiza el entrenamiento con una memoria enorme.
La memoria contiene versiones de la policy antiguas.

Q-networks

- Ventaja: Más experiencia

- Desventaja: Tiene mucho ruido (decisiones que fueron poco óptimas)

Por otro lado encontramos la otra opción para clasificar nuestros algoritmos, una estrategia **off policy**.
Este es el caso contrario, **podemos usar experiencia adquirida con estrategias distintas cada vez que queramos hacer una actualización del aprendizaje de nuestro agente**.

Esta situación está muy relacionada con conceptos vistos durante la sesión, por ejemplo, la experiencia. Si ejecutamos nuestra simulación y vamos almacenando la experiencia que el agente va adquiriendo a la vez que el agente va aprendiendo, es normal **que encontremos transiciones que pertenecen a estrategias distintas**.

Este enfoque, como todos, tiene sus pros y sus contras. El punto a resaltar es que aunque podemos usar **más información (heterogénea) pasada** para el aprendizaje, **puede conllevar a más tiempo de convergencia dependiendo de la información almacenada**. Un ejemplo de familia de algoritmos que siguen este tipo de comportamiento son las **Q-networks**.

* Hay algoritmos híbridos

Índice

Vista general

Conceptos básicos

Conceptos avanzados

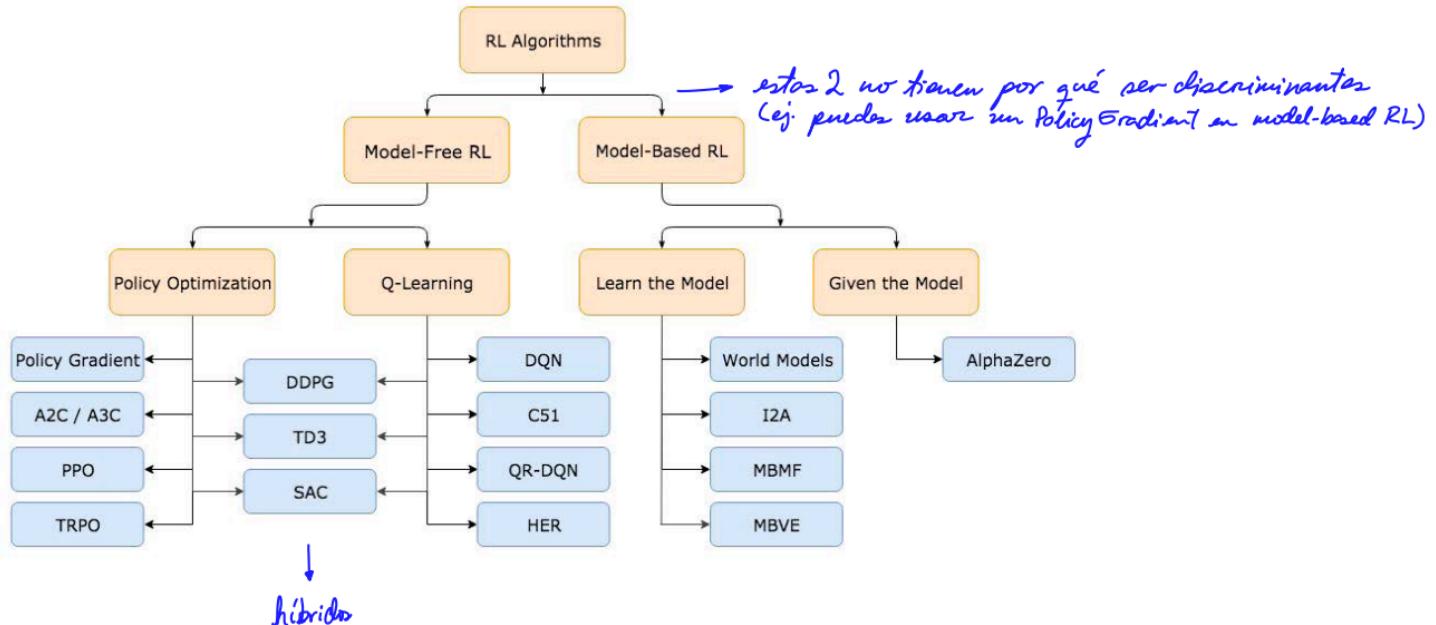
Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

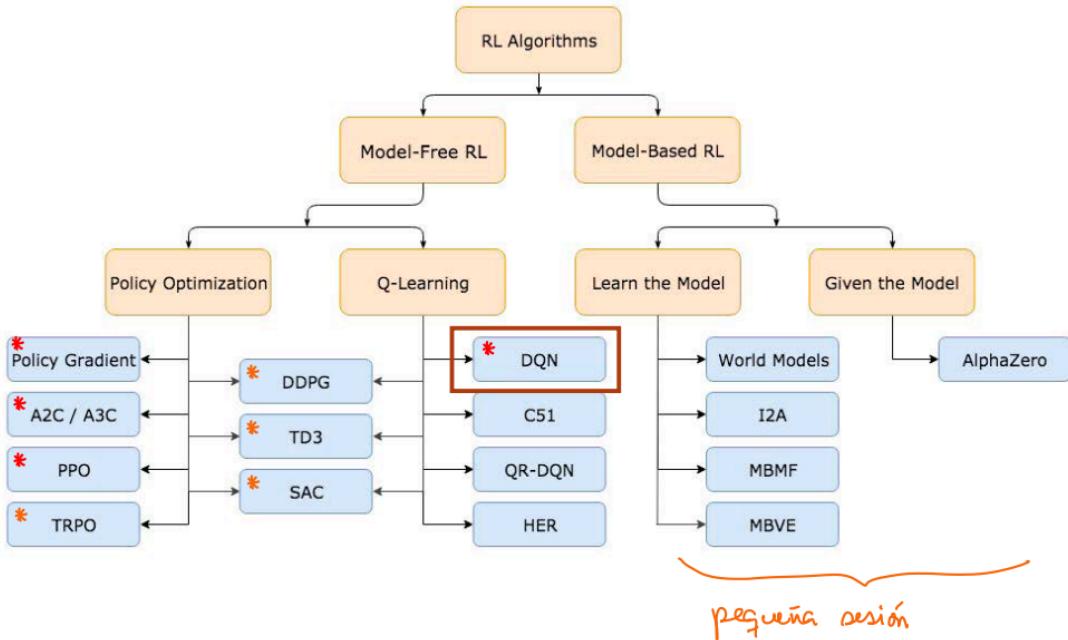
Conclusiones

Bibliografía recomendada

Jerarquía de algoritmos



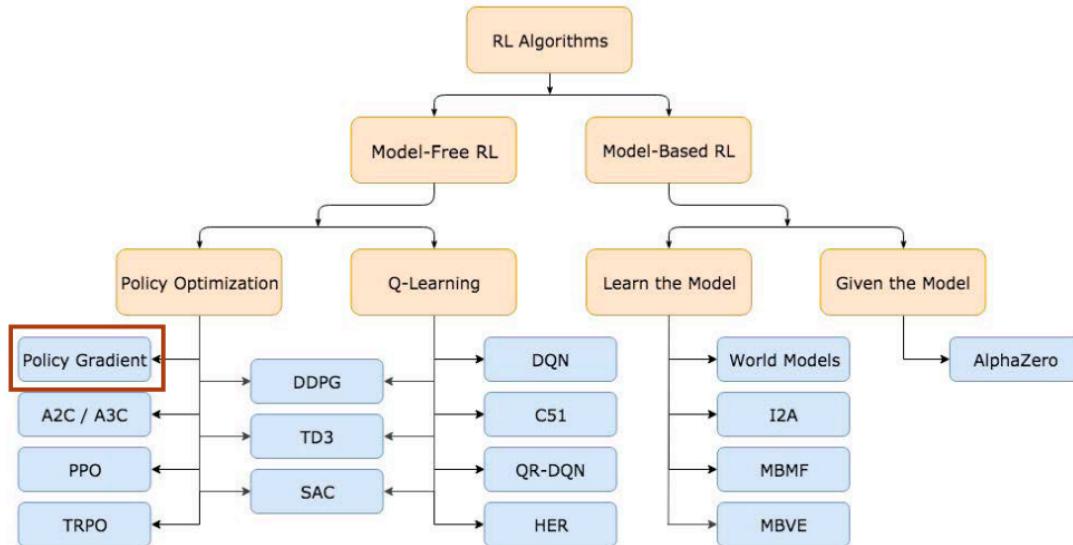
Jerarquía de algoritmos



pequeña sesión

+
actividad de world models?

Jerarquía de algoritmos



Índice

Vista general

Conceptos básicos

Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Conclusiones

- La **terminología** y los conceptos vistos son muy específicos de aprendizaje por refuerzo. Hay que saber diferenciarlos de otras definiciones dentro de la inteligencia artificial o el mundo tecnológico.
- Todos los conceptos están muy relacionados entre sí como por ejemplo los conceptos de Observación y Estado, así como Experiencia y Transición.
- La aplicación de los conceptos vistos es muy flexible dependiendo del algoritmo usado y del reto que se quiera resolver.
- **Las clasificaciones presentadas son fundamentales** para entender qué nos ofrece cada enfoque dentro de todas las posibilidades de las soluciones de aprendizaje por refuerzo.

Índice

Vista general

Conceptos básicos

Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

“Reinforcement Learning: An introduction”, Sutton y Barto:

<http://incompleteideas.net/book/bookdraft2017nov5.pdf>

(Capítulo 2, *Multi Armed Bandits*, donde se trata el dilema de exploración-explotación)

Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems

Levine S. et al, <https://arxiv.org/pdf/2005.01643.pdf>

08MIAR-Aprendizaje por refuerzo

Sesión 3 – Algoritmos base: Deep Q-Network

Curso Octubre 22/23

The logo consists of the lowercase letters "viu" in white, sans-serif font, centered within a solid orange circle.

viu

Universidad
Internacional
de Valencia

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

Conclusiones

Bibliografía recomendada

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

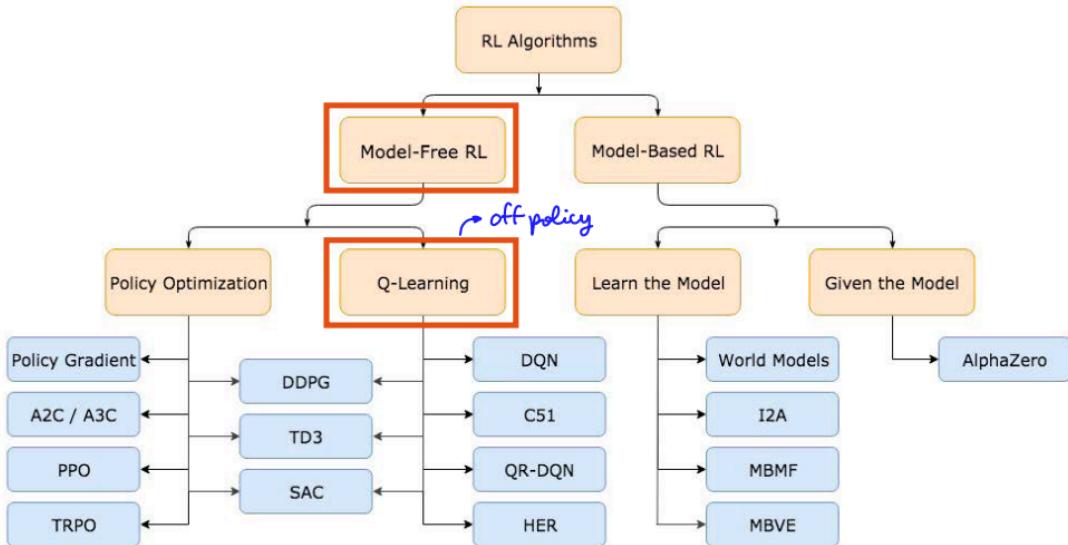
Proceso de aprendizaje

Algoritmo DQN

Conclusiones

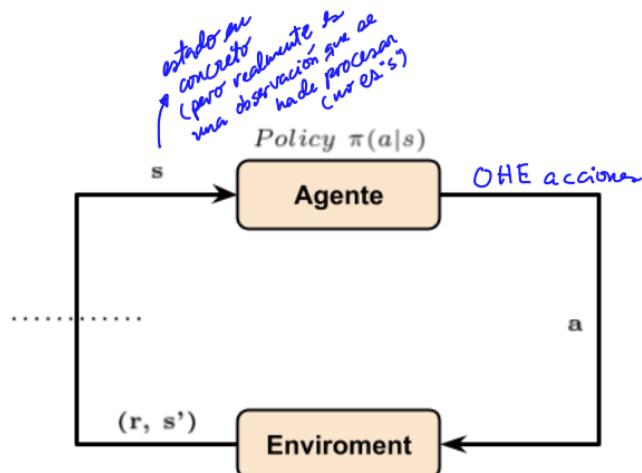
Bibliografía recomendada

¿Dónde estamos?



espacio de estados y acciones finito

Finite Markov Decision Process (aprendizaje)

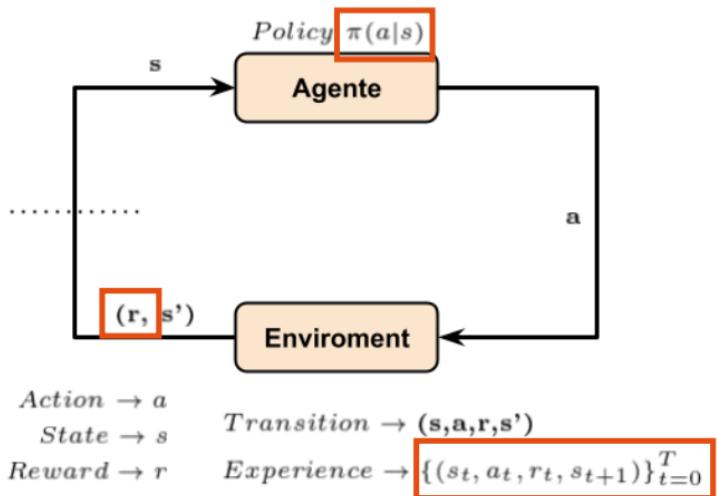


$$\begin{aligned}
 \text{Action} &\rightarrow a \\
 \text{State} &\rightarrow s \\
 \text{Reward} &\rightarrow r
 \end{aligned}
 \quad
 \begin{aligned}
 \text{Transition} &\rightarrow (s, a, r, s') \\
 \text{Experience} &\rightarrow \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^T
 \end{aligned}$$

- **Finite MDPs:** (S, A) son espacios finitos.
- **Markov property:** La probabilidad de cada valor posible para s' depende del estado y la acción inmediatamente anteriores.
- El **estado** debe incluir información sobre todos los **aspectos relevantes** de la interacción agente-entorno en el pasado.

experiencia acumulada de $t=t_0=0$ a $t=T$

¿Cómo de buena es una policy?



según el algoritmo que utilicemos, la función de activación de la policy será distinta.

- Una **policy**, $\pi(a|s)$, es una función de **mapeo de estados a probabilidades sobre acciones**.
- El **objetivo** del agente es **maximizar la recompensa acumulada a futuro**.
(esperada)

estamos mirando al futuro, puede no ser una estimación muy precisa entonces introducimos
- Discounted reward**: discount factor $0 \leq \gamma \leq 1$.

$$\text{Goal } \leftarrow G_t = r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

factor de reducción

- Propiedad de **recursividad**: $G_t = r_{t+1} + \gamma G_{t+1}$

recompensa
inmediata

recompensa
a
futuro

Valor y calidad de un estado

evaluamos el
sitio a nivel general → depende de la policy dada

- **Función value**, $v_\pi(s)$. La función estado-valor de un estado, s , **bajo una policy** π , es la **recompensa esperada a futuro** comenzando en s , y siguiendo la policy desde entonces.

evaluamos
 una única acción
 ↑

$$v_\pi(s) = E_\pi[G_t|s] = \sum_a \pi(a|s) \cdot r_{t+1} + v_\pi(s')$$

media

- **Quality**, $q_\pi(s, a)$. La función calidad de tomar la acción a , en estado s , bajo una , **bajo una policy** π , **es la recompensa esperada a futuro**, comenzando en s , tomando la acción a , y a continuación siguiendo la policy.

$$q_\pi(s, a) = E_\pi[G_t|(s, a)] = \{r_{t+1} + v_\pi(s')\}_a$$

Q-learning
 " "
 Quality - learning

Q-learning → políticas voraces

- **Política óptima, π^* .** Aquella estrategia que, siguiéndola, nos devuelve la máxima recompensa esperada a futuro. Se suele seguir una **política voraz (greedy)**, respecto a $q_\pi(s, a)$.

$$\pi^*(s) = \operatorname{argmax}_a (q^{\pi^*}(s, a))$$

- **Ecuación de Bellman.** El objetivo entonces es **encontrar la función q óptima** para tu problema, y luego seguir una política voraz sobre dicha función. La ecuación de Bellman proporciona una **definición recursiva** con la que podremos encontrar la función óptima q .

$$q^{\pi^*}(s, a) = r + \gamma \max_{a'} q^{\pi^*}(s', a')$$

policy: greedy (max)

definición un tanto contradictoria,
para optimizar q^{π^*} necesitamos conocer q^{π^*}

Este es el problema de Bellman

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

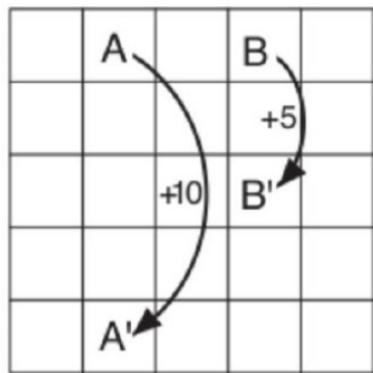
Conclusiones

Bibliografía recomendada

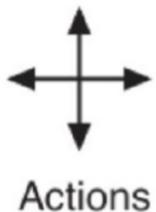
Gridworld

Ejemplo; Objetivo en Q-learning \Rightarrow estimar q

- * 25 estados • 4 acciones posibles \rightarrow recompensa. \Rightarrow tabla de 25×5
- * en cada paso buscamos la fila que maximice la columna de "recompensa"



Gridworld de Sutton & Barto, Capítulo 3



Gridworld de Sutton & Barto, Capítulo 3

- Las **celdas** corresponden con los **estados del entorno**. \rightarrow 25 estados
- La recompensa para cada acción-estado es:
 - **+10** para la transición de **A a A'** (que es la única acción que se puede ejecutar en A).
 - **+5** para la transición de **B a B'** (que es la única acción que se puede ejecutar en B).
 - **-1** si el movimiento se sale del *grid*.
 - **0** en cualquier otro caso.

Random policy evaluation

- Originalmente, la estimación de q se realiza por medio de **programación dinámica**, de forma **iterativa**. De esta forma, se define una **política aleatoria**, y se realiza una exploración del entorno actualizando la función v en base a la experiencia.
- Política aleatoria, $\pi^r(s|a) = 1/A$.

i) Inicialización $v_\pi(s)$

```
array([
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.]])
```

ii) Exploración iterativa

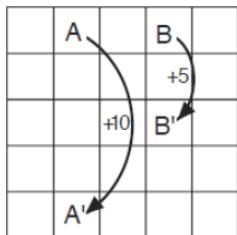
- Comenzamos en $s=[0,0]$. Para cada acción:

- Arriba: $0.25 * (-1 + 0.9 * 0) = -0.25$
- Abajo: $0.25 * (0 + 0.9 * 0) = 0$
- Derecha: $0.25 * (0 + 0.9 * 0) = 0$
- Izquierda: $0.25 * (-1 + 0.9 * 0) = -0.25$

$$v_\pi(s) = E_\pi[G_t|s] = \sum_a \pi(a|s) \cdot r_{t+1} + v_\pi(s') = -0.5$$

- Repetir para cada **iteración k** , para todo s , y actualizar la función value tras cada iteración $v_\pi(s)$

Exploración iterativa



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

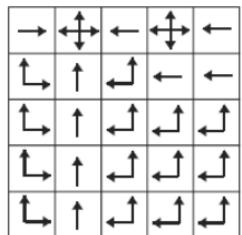
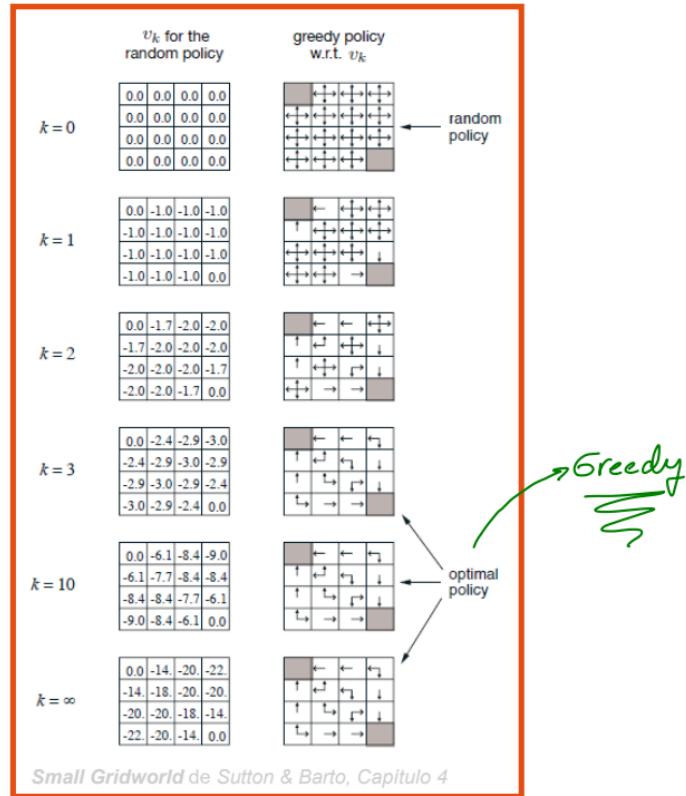
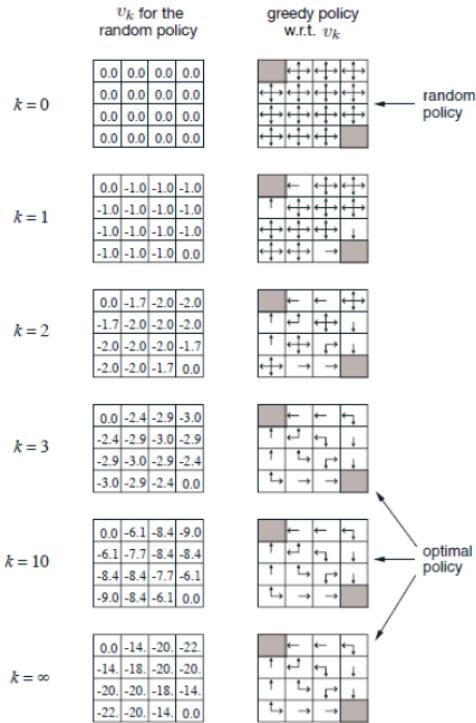
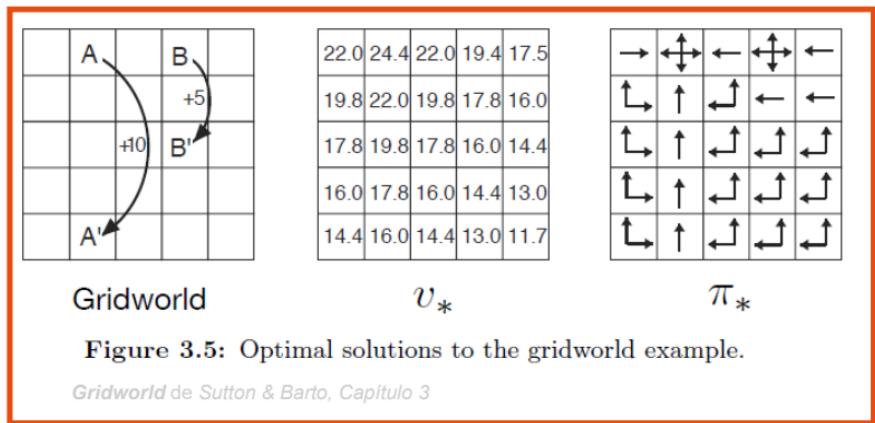
 v_*  π_*

Figure 3.5: Optimal solutions to the gridworld example.

Gridworld de Sutton & Barto, Capítulo 3



Exploración iterativa



Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

Conclusiones

Bibliografía recomendada

Deep Q-network

- Asunciones en **Finite MDPs** : $(\mathcal{S}, \mathcal{A})$ son espacios finitos.
- En escenarios prácticos, la **dimensionalidad** del espacio de **estados y acciones** hace **intratable** esta evaluación iterativa.
- Uso de una **DNN**, $f_\theta(\cdot)$, como función **aproximadora** de $q^{\pi^*}(s, a)$.

$$\pi^*(s) = \operatorname{argmax}_a \left(q^{\pi^*}(s, a) \right) = \operatorname{argmax}_a \left(f_\theta(s) \right)$$

- Debido a que la mayoría de las simulaciones trabajan con la pantalla directamente, el tipo de red neuronal que usaremos serán **redes convolucionales**.

Deep Q-network

La red estima, dado un estado, la función "q" para cada posible acción. Es decir, recompensa esperada a futuro para cada una de las acciones posibles en ese estado.

A partir de ahí, seleccionamos la que nos maximice la recompensa esperada a futuro.



<https://cdn.analyticsvidhya.com/wp-content/uploads/2019/04/Screenshot-2019-04-16-at-5.46.01-PM.png>

- ¿Función de activación?
- ~~softmax~~ → desaparece independencia de acciones
 - sigmoid } puede ser, si esto todo
 - tanh } muy bien escalado
 - linear → mejor para la recompensa en un escalar entre -∞ e ∞

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

Conclusiones

Bibliografía recomendada

Proceso de aprendizaje

- Uso de **Bellman** como función aproximadora en nuestro proceso de aprendizaje.

$$q(s, a) = r + \gamma \max_a q(s', a)$$

vora

MSE, RMSE, ...

- Considerando que $q(s, a)$ como una red neuronal, $f_\theta(\cdot)$, se puede definir una **función de coste** en base a las **diferencias temporales**, y optimizar la red neuronal en base a la tupla (s, a, r, s') de cada iteración por **descenso de gradiente**. Como función de pérdidas, el error cuadrático suele ser la solución base.

$$q(s, a) \rightarrow r + \gamma \max_a q(s', a)$$

$$L = (r + \gamma \max_a q(s', a)) - q(s, a)^2$$

* teacher-network

este no se tiene en cuenta para el gradiente

valor de una versión estática de nuestra red (target network)

dependencia con el "futuro"

error cuadrático

de este salen los gradientes para el back-propagation

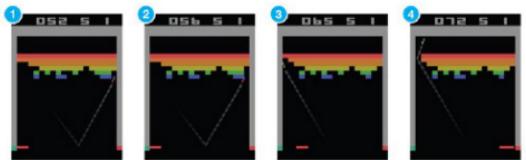
Este tipo de entrenamiento tiene problemas de convergencia, por ello utilizamos lo siguiente.

Proceso de aprendizaje

- El uso de una **red neuronal para estimar la función $q(s, a)$** no **asegura convergencia a la $q^*(s, a)$** . Es un **proceso inestable**, con tendencia a llegar a **mínimos locales**. Se han desarrollado una serie de **estrategias** para aliviar esto:

- Series temporales.**

Convergencia: La fase de exploración cogiendo una policy aleatoria, no actualizamos ninguna red, y durante 1000 iteraciones almacenamos experiencia (tuplas (s_t, a_t, r_t, s_{t+1})) sin usar. Llegamos un buffer de experiencia y luego empezamos a entrenar la red; ahora usa el ϵ -greedy. A la hora de seleccionar la acción se gasta un número determinado y si ésta es mejor que un E dado se elige la acción que indica la policy, si no se elige una acción aleatoria. E decrece con el tiempo.



- Exploración.** Se sigue una política ϵ -greedy para evitar sobreajuste.

- Experience replay.** En la experiencia se almacenan tuplas (s, a, r, s') obtenidas con versiones antiguas de la red $q(s, a)$.

- Target network.** En la función de coste se sustituye la $q(s, a)$ del siguiente estado por una versión anterior de la red neuronal, llamada target network, $\hat{q}(s', a)$. That is: $q(s, a) \rightarrow r + \gamma \max_a \hat{q}(s', a)$.

- Error clipping.** Evitar actualizaciones demasiado altas. Se suele utilizar la función de Huber.

$$\text{Huber}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq 1, \\ (|a| - \frac{1}{2}), & \text{otherwise.} \end{cases}$$

Por qué DQN es off-policy?

Según el profesor de la memoria.
o sólo por la gestión de la memoria.

- DQN es el principal exponente de la familia de métodos conocidos como **off-policy**. Pero, ¿a qué se refiere este tipo de aprendizaje? Os encontraréis dos interpretaciones en la literatura:
 - ✗ • Basada en **Exploración**. Al seguir una **política ϵ -greedy**, la política que tratamos de predecir como **política objetivo (voraz, greedy)** no se corresponde con la que nuestro agente está siguiendo.
 - ✓ • Basada en **Experience replay**. Al acumular en la **experiencia** tuplas (s, a, r, s') de **antiguas iteraciones**, la red es actualizada con políticas que no se corresponden con la actual.
- La opción más aceptada en la literatura es la segunda, basada en la **clasificación de estrategias de aprendizaje centrada en el uso de la memoria**.

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

Conclusiones

Bibliografía recomendada

Algoritmo DQN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N

Initialize action-value function Q with random weights

for episode = 1, M **do**

 Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

puede hacerse cada
step o cada N steps,
para hacer el proceso
más eficiente y acumular
memoria

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

Playing Atari with Deep Reinforcement Learning (<https://arxiv.org/abs/1312.5602>)

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

Conclusiones

Bibliografía recomendada

Conclusiones

- Hemos estudiado el primero de los algoritmos base que veremos en la asignatura, **DQN**. Este algoritmo pertenece a la familia de **métodos off-policy**.
- Está basado en **Q-learning**, donde el objetivo es **estimar la recompensa** esperada para cada par **estado/acción**.
- **DQN combina Q-learning con arquitecturas de Deep Learning**, para poder abarcar problemas con **espacios de dimensiones muy grandes**
- Algunas variaciones, necesarias para la convergencia de la solución, en la versión final del algoritmo de DQN son el uso de una **target network** y de **secuencias de frames** como datos de entrada del modelo del agente.

* Bellman no asegura la convergencia por el tema de la recursividad

Índice

Definición Q-learning

Ejemplo Q-learning: gridworld

Deep Q-network

Proceso de aprendizaje

Algoritmo DQN

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

- Human level control through Deep Reinforcement learning, Google Deepmind
[https://deepmind.com/research/publications/2019/human-level-control-through-deep-reinforcement learning](https://deepmind.com/research/publications/2019/human-level-control-through-deep-reinforcement-learning)
- An Introduction to Q-learning: Reinforcement Learning, Sayak Paul, Floydhub
<https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/>

08MIAR-Aprendizaje por refuerzo

Sesión 4 – Algoritmos base: Policy Gradient

Curso Abril 23/24

The logo consists of the lowercase letters "viu" in white, sans-serif font, centered within a solid orange circle.

viu

Universidad
Internacional
de Valencia

De:

 Planeta Formación y Universidades

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

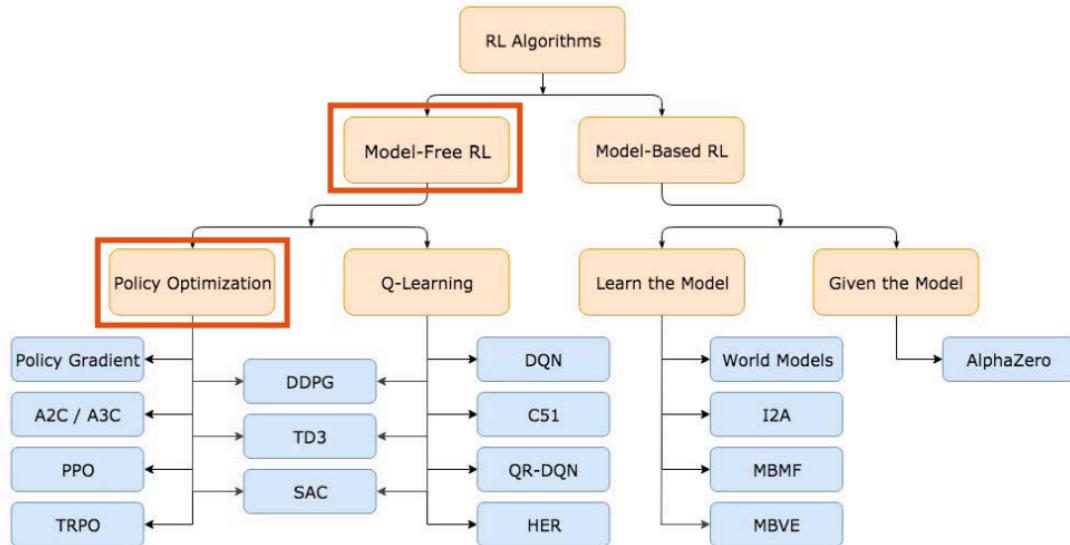
Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

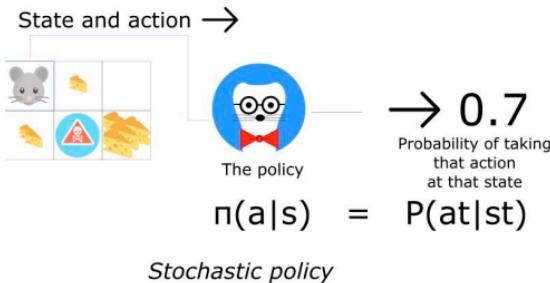
Bibliografía recomendada

¿Dónde estamos?



Motivación

Policy Gradients no se basa en estimar la recompensa esperada a futuro, sino que se basa en devolver directamente una distribución de probabilidad de la policy.



https://cdn-images-1.medium.com/max/1600/1*YCAjP7x1wZZZKqz2CoyQ.png

- **Policy, π .** Estrategia que sigue nuestro agente. Probabilidad de realizar una acción en un estado.

$$p(a|s) = \pi(s): \pi(s) \in [0, 1]$$

- **Q-learning.** Se sigue una **política voraz** respecto a la función Q, la cual debe estimarse previamente.

$$\pi(s) = \operatorname{argmax}_a (q^\pi(s, a))$$

en vez de dar recompensas más da probabilidades que para todo el conjunto de acciones sumarán 1.

- **Policy gradient (PG).** Tiene como objetivo modelar directamente una distribución de probabilidad policy, $\pi(s)$.

Motivación

PG vs. Q-learning

- PG permite obtener directamente **policies estocásticas** - e.g., $\pi(s) = [0, 0.2, 0.6, 0.2]$ -, mientras que Q-learning ofrece únicamente policies deterministas - e.g., $\pi(s) = [0, 0, 1, 0]$.
- PG permite modelar un espacio con **acciones continuas**. Funciona especialmente bien en espacios de acciones muy grandes.
- Q-learning busca obtener la mejor estimación de la función Q, para mejorar de forma indirecta la policy. **PG actúa directamente sobre la policy**, lo cual mejora la **eficiencia y convergencia**.

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

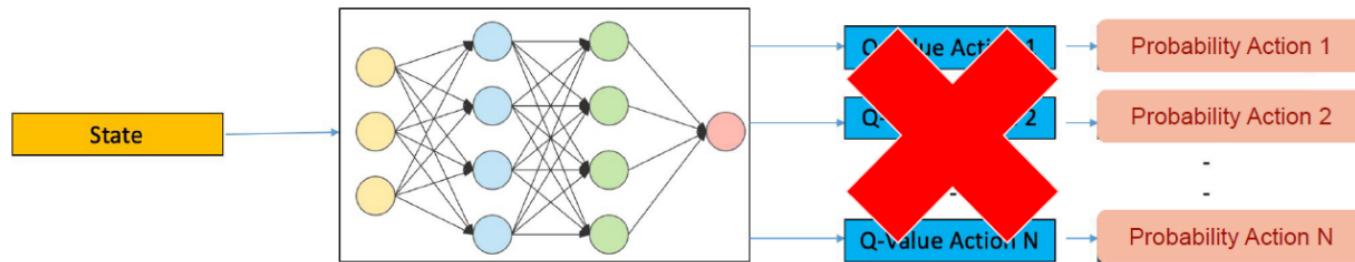
Deep Policy Gradient (DPG)

- Asunciones en **DQN**: mejorando la estimación de la q , *indirectamente* mejoraremos la **policy**. Muchas ocasiones (espacio de acciones muy grande) dicha convergencia no sucede.
- DPG: optimizamos la **policy** – distribución de probabilidad de acciones - directamente, $\pi(s)$.
- Uso de una **DNN**, $f_\theta(\cdot)$, como función **aproximadora** de $\pi^*(s)$.

$$\pi^*(s) = (f_\theta(s))$$

- Debido a que la mayoría de simulaciones trabajan con la pantalla directamente, el tipo de red neuronal que usaremos serán **redes convolucionales**.

Conceptos importantes



Deep Policy Gradient

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

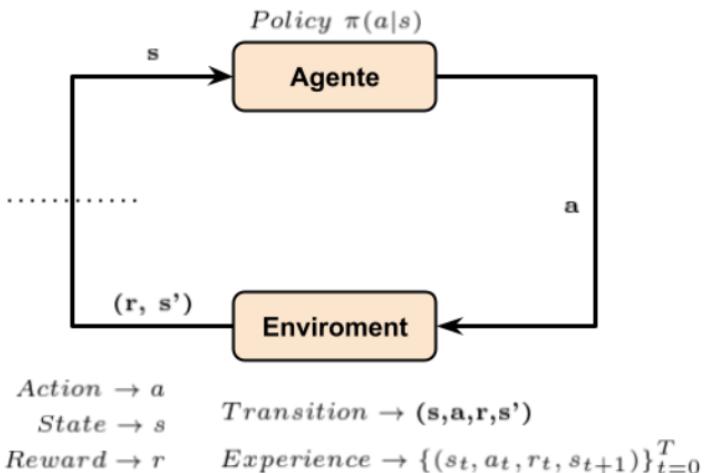
Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

Trayectorias entorno-agente en PG



- En **PG**, trabajamos **on-policy**. Es decir, la memoria únicamente contiene **transiciones de una policy dada**. En este caso, se le llama **Trayectoria (T)**.

↳ Experiencia recopilada en un episodio

- Se define una **policy**, $\pi(a|s)$, parametrizada por una DNN, que toma como entrada el estado preprocesado, y devuelve una **distribución de probabilidad sobre las acciones, con activación softmax**. Por ejemplo $\pi(a|s^*) = [0.2, 0.2, 0.6, 0]$.

$$\pi(a|s) = f_\theta(s)$$

- Seleccionamos la acción a a partir de un muestreo de $\pi(a|s)$, lo cual permite la **exploración**.

$$a = \text{sample } (\pi(s))$$

Aprendizaje supervisado

Una función de optimización típica de otra rama de *deep learning*, el **aprendizaje supervisado**, es la **entropía cruzada**, utilizando las probabilidades predichas, y un *one-hot-encoding* de la distribución de probabilidad predicha para el conjunto de etiquetas futuras.

$$L = - \sum y_i \log (\hat{y}_i)$$

En **reinforcement learning**, no existe un **ground truth** para una acción dada, pero nos sirve para establecer ciertas **analogías**.

$$L = - \sum y_a \log (\pi(s)_a)$$

podríamos usar la recompensa que nos da el entorno para la acción como algo similar al ground truth

Policy gradients – Naive solution

Se define un entorno con una **policy**, $\pi_\theta(a|s)$, que produce una distribución de probabilidad sobre el conjunto de acciones en una interacción agente-entorno. Esta interacción se realiza durante una trayectoria, de duración T , y en cada iteración, t_i , se produce una transición $(s_{t_i}, a_{t_i}, r_{t_i}, s_{t_i+1})$.

Se quiere optimizar la policy para acercarla en la mayor medida a una solución óptima. Policy Gradients (PG) se centra en actualizar π_θ en dirección del gradiente de la misma. En el caso ideal, **si la acción seleccionada es la óptima, se quiere que su probabilidad sea lo más cercana a 1**, lo cual se conseguiría por descenso de gradiente.

$$\theta_{t_i+1} = \theta_{t_i} + \alpha \nabla \pi_{\theta_{t_i}}(a_{t_i}^* | s_{t_i}) \quad (i)$$

Sin embargo, **la acción óptima no es conocida a priori**. Inicializando en una **policy aleatoria**, si optimizamos sobre una acción genérica (ii), estamos **optimizando soluciones sub-óptimas**.

$$\theta_{t_i+1} = \theta_{t_i} + \alpha \nabla \pi_{\theta_{t_i}}(a_{t_i} | s_{t_i}) \quad (ii)$$

Policy gradients – Ponderando acciones

Una solución para **ponderar los gradientes** es dar más importancia a aquellas acciones que den una mejor policy, es utilizar la información que proporciona el entorno como *feedback*: las **recompensas esperadas a futuro** dada la trayectoria realizada. Para ello, retomamos el concepto de función **quality** $q_\pi(s, a)$:

$$q_\pi(s, a) = E_\pi[G_t | (s, a)]$$

Por lo tanto, la actualización de la policy en un step dado quedaría:

$$\theta_{t_i+1} = \theta_{t_i} + \alpha q_\pi(s_{t_i}, a_{t_i}) \nabla \pi_{\theta_{t_i}}(a_{t_i} | s_{t_i}) \quad (iii)$$

On-policy correction: actualizamos la policy en base a las acciones que toma la propia policy en una trayectoria. **Se debe compensar que acciones más probables serán llevadas a cabo más a menudo.** Por ello, se obtiene el *ratio* entre la actualización de la policy y la probabilidad de la acción tomada.

$$\theta_{t_i+1} = \theta_{t_i} + \alpha \frac{q_\pi(s_{t_i}, a_{t_i}) \nabla \pi_{\theta_{t_i}}(a_{t_i} | s_{t_i})}{\pi_{\theta_{t_i}}(a_{t_i} | s_{t_i})} \quad (iv)$$

¿Qué hay del logaritmo?

Aprendizaje supervisado: $L = - \sum y_i \log (\hat{y}_i)$

RL, Policy Gradient: Podemos realizar el *log-derivative trick*, para obtener una expresión logarítmica de la actualización de los pesos.

$$\nabla \ln f(x) = \frac{\nabla f(x)}{f(x)} \rightarrow \frac{q_{\pi}(s_{t_i}, a_{t_i}) \nabla \pi_{\theta_{t_i}}(a_{t_i} | s_{t_i})}{\pi_{\theta_{t_i}}(a_{t_i} | s_{t_i})} = q_{\pi}(s_{t_i}, a_{t_i}) \nabla_{\theta_{t_i}} \log \pi_{\theta_{t_i}}(a_{t_i} | s_{t_i})$$

→ similar a la entropía cruzada.

Por tanto, el criterio de optimización de *vanilla policy gradient* consiste en:

$$\theta_{t_i+1} = \theta_{t_i} + \alpha \underbrace{q_{\pi}(s_{t_i}, a_{t_i}) \nabla_{\theta_{t_i}} \log \pi_{\theta_{t_i}}(a_{t_i} | s_{t_i})}_{(v)}$$

mejoramos
la \hat{f}

||| Close 14/11/2023, Sesión 4 → min 01:40:00 |||

Suavizando recompensas

$$q_{\pi}(s, a) = [1.0, 1.1, 1.0, 1.0]$$

*esta no tiene tanta
relevancia frente
al resto*

$$q_{\pi}(s, a) = [\underline{1.0}, 0.0, 0.0, 0.0]$$

*↳ es muy importante
esta acción en este
caso*

¿A qué acción se le debe dar
mayor importancia al
actualizar?

- Para suavizar el problema con el uso de la recompensa como factor, se proponen reemplazos de la función de calidad.

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_{\theta} \mathbb{E} [\sum_{t=0}^{\infty} r_t]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E} \left[\sum_{t=0}^{\infty} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \quad (1)$$

where Ψ_t may be one of the following:

- $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
- $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t .
- $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
- $Q^{\pi}(s_t, a_t)$: state-action value function.
- $A^{\pi}(s_t, a_t)$: advantage function.
- $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual.

The latter formulas use the definitions

$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty} \\ a_{t:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty} \\ a_{t+1:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad (2)$$

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad (\text{Advantage function}). \quad (3)$$

Estrategias de entrenamiento

Durante el proceso de aprendizaje debemos tener algunos conceptos y situaciones presentes, para entender qué está ocurriendo:

- Una de las primeras decisiones que debemos tomar es “**¿Cuántos steps vamos a usar para ir modificando la estrategia?**”. El proceso de aprendizaje se puede ver más o menos impactado dependiendo del número de iteraciones que realicemos para ir almacenando nuestra experiencia.
- Además, al trabajar con la trayectoria, **las recompensas obtenidas se procesarán siguiendo un enfoque conocido como discounted rewards**. Al tener una trayectoria finita, utilizaremos las recompensas en sentido inverso para ir estimando la recompensa esperada a futuro (G) en los siguientes estados y de esta forma poder ponderar las acciones de manera adecuada.
→ Se usa el baseline.
- Por otro lado, **usar la recompensa como factor de las probabilidades de las acciones produce una varianza en los datos muy grande**. Tened en cuenta que con esta definición la probabilidad de una acción en un estado puede cambiar dependiendo de si la recompensa cambia también. Esto **dificulta el aprendizaje** ya que **no se encuentra una correlación entre estado y probabilidad de acción fácilmente**. Esta situación se puede dar en muchos escenarios, sobre todo en las simulaciones basadas en videojuegos.

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

REINFORCE

```
function REINFORCE
    Initialise  $\theta$  arbitrarily
    for each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  do
        for  $t = 1$  to  $T - 1$  do
             $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$ 
        end for
    end for
    return  $\theta$ 
end function
```

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

Vanilla Policy Gradient

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ , baseline b

for iteration=1, 2, ... **do**

 Collect a set of trajectories by executing the current policy

 At each timestep in each trajectory, compute

 the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

 the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

Baseline, reducir
la varianza!

 Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
 summed over all trajectories and timesteps.

 Update the policy, using a policy gradient estimate \hat{g} ,
 which is a sum of terms $\nabla_\theta \log \pi(a_t | s_t, \theta) \hat{A}_t$

end for

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

Conclusiones

- (+) - **Policy Gradient** es uno de los algoritmos base dentro del aprendizaje por refuerzo. Este algoritmo es el **origen de algunos de los algoritmos más potentes actualmente**. Es un algoritmo de tipo **on-policy**.
- (+) - En comparación con DQN, algunas de las características a destacar son una **mayor capacidad de convergencia**, apto para trabajar con **espacios de acciones grandes (y continuos)** y posibilidad de aprender **policies estocásticas**.
- (+) - La **función de coste** en Policy Gradient se centra en ir **optimizando la propia policy aplicando gradient ascent sobre la probabilidad de la acción seleccionada y su recompensa obtenida**.
- Al ser un algoritmo base, veremos en las siguientes sesiones de la asignatura las evoluciones que se han ido produciendo para estabilizar y mejorar el proceso de aprendizaje del agente.
- (-) *Es muy ineficiente en el uso de la memoria. Se descarta mucha.*

Índice

Definición Policy Gradient

Deep Policy Gradient

Proceso de aprendizaje

Algoritmo: REINFORCE

Algoritmo: Vanilla Policy Gradient

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

- “Reinforcement Learning: An Introduction”, Sutton y Barto:
<http://incompleteideas.net/book/bookdraft2017nov5.pdf>
(Capítulo 13, Policy Gradient Methods)
- An Intuitive explanation on Policy Gradients, Adrien Lucas, Towards data science / Medium
<https://towardsdatascience.com/an-intuitive-explanation-of-policy-gradient-part-1-reinforce-aa4392cbfd3c>

08MIAR-Aprendizaje por refuerzo

**Sesión 6 – Evolución del algoritmo Policy Gradient:
Actor-Critic y A2C/A3C**

Curso Abril 23/24



**Universidad
Internacional
de Valencia**

De:

Planeta Formación y Universidades

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

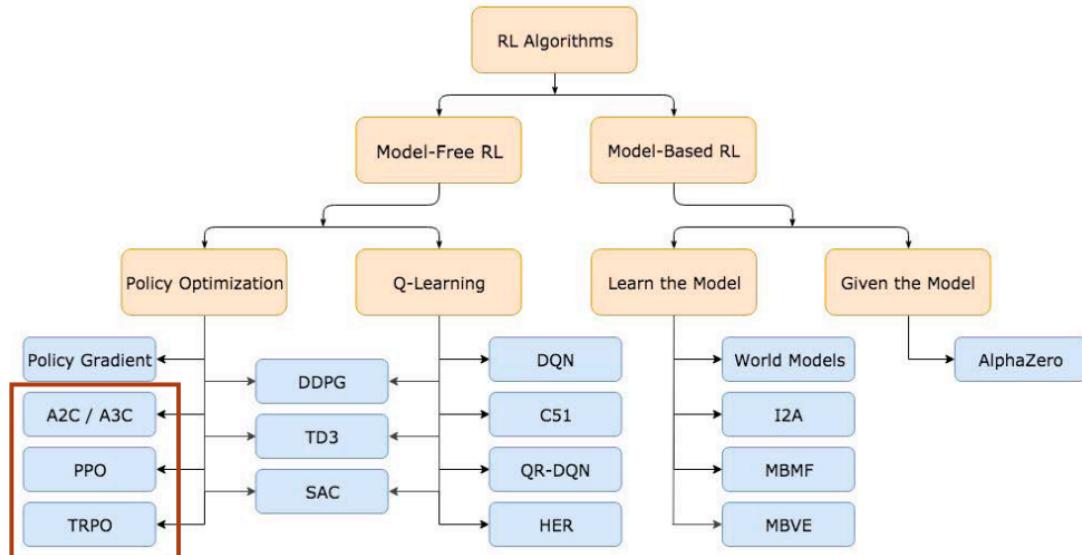
Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Introducción



Introducción

Antes de continuar con los algoritmos, recordemos justo uno de los puntos más relevantes del algoritmo de Policy Gradient, el uso de la **recompensa como factor de relevancia**. Este hecho provocaba que en algunos casos las soluciones tuvieran **problemas de convergencia y de demasiada varianza** a la hora de entrenar el modelo.

Para solventar esta situación y como punto de mejora introducida por estos algoritmos, vamos a presentar el concepto conocido como **Value**. De manera informal, podemos definir el *Value* como:

- **El valor medio de recompensa esperada en el estado actual**
- Cómo de bueno es estar en el estado actual de la ejecución
- Cómo de bien estoy ejecutando la *policy* actual

El *value* es un concepto fundamental en aprendizaje por refuerzo para poder derivar funciones más complejas a la hora de ponderar las acciones seleccionadas por el agente.

Introducción

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_{\theta} \mathbb{E} [\sum_{t=0}^{\infty} r_t]$. There are several different related expressions for the policy gradient, which have the form

[Examen]

$$g = \mathbb{E} \left[\sum_{t=0}^{\infty} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \quad (1)$$

where Ψ_t may be one of the following:

- 1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
- 2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t .
- 3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
- 4. $Q^{\pi}(s_t, a_t)$: state-action value function.
- 5. $A^{\pi}(s_t, a_t)$: advantage function.
- 6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual.

↳ pelear varianza o grandes valores del reward

The latter formulas use the definitions

$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty} \\ a_{t:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty} \\ a_{t+1:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad (2)$$

más mala. → $A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$, (Advantage function). (3)

↳ normaliza respecto a escala y horario — cuenta no solo la trayectoria actual, sino las pasadas también (ya realizadas)

(Recordatorio)

Es común confundir o utilizar en el mismo contexto los conceptos de Value y Q_value. La diferencia principal es el parámetro de entrada en cada caso, ya que el Value se calcula a partir del estado donde se encuentra el agente mientras que el Q_value se obtiene relacionando el estado y la acción tomada.

[Examen]

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

$$Q^{\pi}(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Actor-Critic: Definición & Conceptos importantes

El primero de los algoritmos que estudiaremos en la sesión de hoy es el conocido como algoritmo de **Actor-Critic**. La principal característica de este enfoque es que el modelo del agente se divide en dos componentes, el *Actor* y el *Critic*:

actor

- El **Actor** será el responsable de devolver la **distribución de probabilidades asociada a las acciones que el agente** puede tomar, similar a Policy Gradient.

crítico
de
teatro

- El **Critic** se encargará de **estimar el Value** en el estado en el que se encuentre el agente en la ejecución. ↗ *cuán buenas son las estados en los que se mueve.*

El resto de la lógica de este algoritmo es similar a la implementación estudiada de Policy Gradient.

Actor-Critic: Definición & Conceptos importantes

Como este algoritmo sigue comportándose siguiendo una estrategia ***on-policy***, el ***sampling*** de los datos es igual que en el caso de Policy Gradient.

Definimos un **número finito de steps**, que van a definir la trayectoria que vamos a ejecutar, y **vamos almacenando toda las transiciones** que se van obteniendo a partir de la interacción agente-entorno.

(s, a, r, s' , 

Como novedad, tenemos que tener en cuenta que **necesitamos estimar el Value en cada estado que hemos almacenado.**

$$R_t = \sum_{i=t}^T \gamma^{i-t} r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_T \quad v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

Actor-Critic: Definición & Conceptos importantes

Respecto a la función de coste, al dividir el objetivo en **dos estimaciones** diferentes, ésta estará compuesta de **dos componentes**, una parte enfocada en el *Actor* y otra en el *Critic*.

Para el *Actor*, la fórmula que empleamos es la misma que definimos en **Policy Gradient**, con la variación de que **en vez de la recompensa utilizamos el Value estimado**. El *Critic* estimará el **error cometido entre el Value recolectado y el Value estimado**.

El hecho de utilizar el Value en la función de coste durante el proceso de aprendizaje va a dotar al entrenamiento de **más robustez a la hora de relacionar recompensas obtenidas con las acciones ejecutadas**, ya que el Value es un estimador más “general” en comparación con la recompensa directa de la acción.

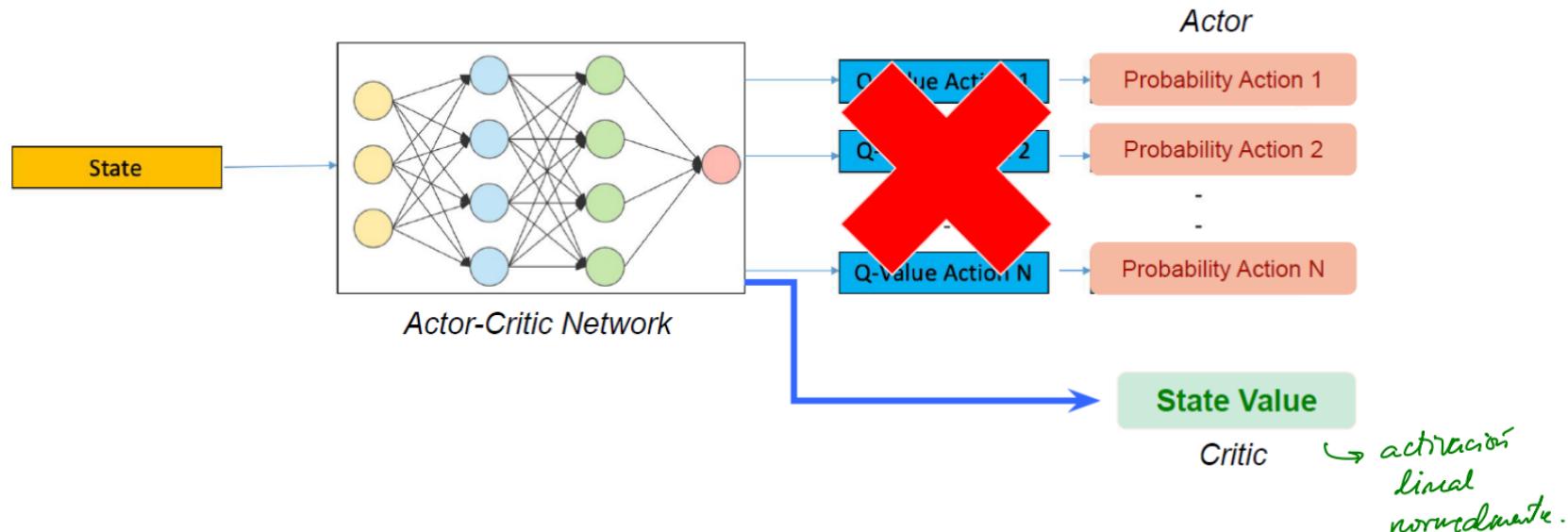
[Examum]

$$\nabla_{\theta} \log \pi_{\theta}(s, a) (R - V_{\varphi}(s))$$

$(R - V_{\varphi}(s))^2$ → ecuación de Bellman.
 función de pérdida optimizar el Critic

discount rewards obtenidos en los steps

Actor-Critic: Definición & Conceptos importantes



Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

[Examen]

Algoritmo Actor-Critic

Algorithm 1 Monte Carlo on policy actor-critic.

Require: Initialize policy π with parameters θ_π and value critic v_π with parameters θ_v

- 1: **for** each episode **do**
- 2: Get initial state s
- 3: Initialize storage buffer S, A, R, S'
- 4: **for** $i = 1, 2, 3 \dots N$ steps **do**
- 5: Sample action with policy: $a \sim \pi_\theta(s)$
- 6: Run action through environment, obtain reward and post state: $r, s' \leftarrow ENV(s, a)$
- 7: Collect and store: $S, A, R, S' \leftarrow s, a, r, s'$
- 8: $s \leftarrow s'$
- 9: **end for**
- 10: Compute discount returns: $\hat{V} = \sum_{l=0}^{N-1} \gamma^l r_{t+l}$
- 11: Update θ_v to minimize $\sum_{n=1}^N \|v_\pi(s_n) - \hat{V}_n\|^2$ "baseline"
- 12: With learning rate α , update policy: $\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_\theta \log \pi(A|S)v_\pi(S)$
- 13: **end for**

↓ Aquí se está ponderando con la función Advantage?
 → NO, advantage es $Q - V$

ponderar con el value,
 pero lo que se suele hacer es ponderar con la función Advantage

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

[Examens]

Problema del policy gradient.

↳ memoria muy pobre
↓
mejora

↳ aumentar la cantidad de
datos que se tienen al actualizar
la policy (varias trayectorias).

Pero varias trayectorias en serie
es poco eficiente.

Introducción

No es un sistema multi-agente, es un agente con distintas versiones del entorno (distintas semillas que generan distintos caminos)

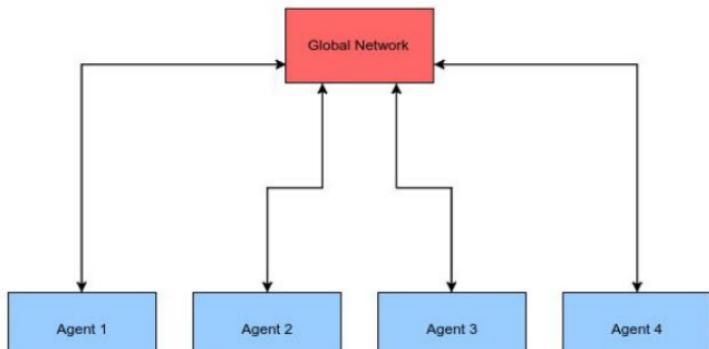
El siguiente de los algoritmos que veremos es el conocido como **(Asynchronous) Advantage Actor Critic**, del que podemos encontrar dos versiones: **una síncrona y otra asíncrona**.

El componente de la **sincronicidad** está relacionado con el hecho de que esta nueva versión es un algoritmo multi-proceso, en el que **una batería de agentes se ejecutan en paralelo durante el entrenamiento** en un entorno determinado. En esa ejecución, dependiendo de la comunicación que haya entre ellos, hablaremos de **A2C (sincronía)** o **A3C (asincronía)**.

Por **comunicación** nos referimos a **cómo se actualiza el modelo global**. Hay que tener en cuenta que aunque cada agente tenga su propia versión del modelo para la toma de acciones, **todos comparten ese conocimiento global por medio de un modelo general**. Cómo se actualice este modelo determina la versión del algoritmo con la que trabajamos.

Vista general

Mismo agente, distintas versiones del entorno: **diversidad en las trayectorias**



A3C



A2C

<https://external-preview.redd.it/uwqehzHvsm6y82P2d1jReEkSINvhqUv6EPITNZI-aSk.png?auto=webp&s=a68dcc5f54e690d3a2acb2b7510848bcb8726ae8>

→ no es ruido
como tal

Características principales

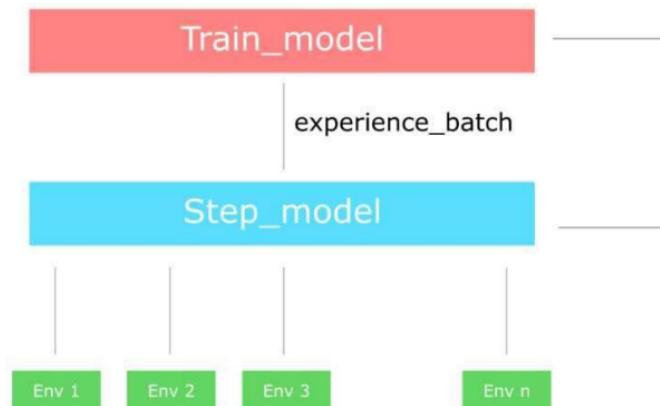
Algunas de las **características** que podemos destacar para **A2C/A3C** son:

*Heredado de
PG*

- Posibilidad de modelar **espacios de acciones continuos y entornos estocásticos.**
- Tener una **mayor diversidad y volumen de datos** disponibles para el proceso de entrenamiento.
- **Cada proceso se ejecuta en CPU**, por lo que permite trabajar con problemas **de visión por computador sin tener que utilizar GPU**
- Al utilizar un entrenamiento multi-proceso, los **tiempos de entrenamiento se reducen** de manera notoria.

Algoritmo multi-proceso

Esquema (interpretación) en el caso de A2C sería:



<https://medium.freecodecamp.org/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d>

Función de ventaja

El otro elemento que es nuevo en este enfoque es **el reemplazo del Value como factor de las acciones tomadas por el agente**. En su lugar, **usaremos la conocida como función de ventaja**:

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad (\text{Advantage function}).$$

Esta función es una medida más precisa para decidir si el agente va por el buen camino o no siguiendo la estrategia actual. Es una medida **que compara el Value del estado con la recompensa esperada obtenida con la acción tomada en el estado actual**, de tal forma que define un umbral de bondad de la acción seleccionada con respecto al resto de opciones.

Generalized Advantage estimation

Actualmente, las funciones de relevancia que más se utilizan en Aprendizaje por refuerzo son funciones derivadas de esta función de Ventaja. Por ejemplo, otra opción que encontraremos ampliamente en la literatura es la conocida como **Generalized Advantage estimation**:

$$\text{GAE}(\gamma, \lambda) : \quad \hat{A}_t := \delta_t + \underbrace{\gamma V(s_{t+1}) - V(s_t)}_{\text{para eliminar el ruido?}}$$

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Algoritmo: A2C

Algorithm 1 Parallel advantage actor-critic

```

1: Initialize timestep counter  $N = 0$  and network weights  $\theta, \theta_v$ 
2: Instantiate set  $e$  of  $n_e$  environments
3: repeat
4:   for  $t = 1$  to  $t_{max}$  do
5:     Sample  $a_t$  from  $\pi(a_t|s_t; \theta)$ 
6:     Calculate  $v_t$  from  $V(s_t; \theta_v)$ 
7:     parallel for  $i = 1$  to  $n_e$  do
8:       Perform action  $a_{t,i}$  in environment  $e_i$ 
9:       Observe new state  $s_{t+1,i}$  and reward  $r_{t+1,i}$ 
10:      end parallel for
11:    end for
12:     $R_{t_{max}+1} = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_{t_{max}+1}; \theta) & \text{for non-terminal } s_t \end{cases}$ 
13:    for  $t = t_{max}$  down to 1 do
14:       $R_t = r_t + \gamma R_{t+1}$ 
15:    end for postprocesar contenido actualizar Advantage
16:     $d\theta = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} (R_{t,i} - v_{t,i}) \nabla_\theta \log \pi(a_{t,i}|s_{t,i}; \theta) + \beta \nabla_\theta H(\pi(s_{e,t}; \theta))$ 
17:     $d\theta_v = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} \nabla_{\theta_v} (R_{t,i} - V(s_{t,i}; \theta_v))^2$ 
18:    Update  $\theta$  using  $d\theta$  and  $\theta_v$  using  $d\theta_v$ .
19:     $N \leftarrow N + n_e \cdot t_{max}$ 
20: until  $N \geq N_{max}$ 

```

recopilar
trayectorias

postprocesar
contenido
y
actualizar

trayectorias.

calculamos recompensa
esperada a futuro

actualización
red.

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Algoritmo: A3C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```

// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$ 
// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$ 
Initialize thread step counter  $t \leftarrow 1$ 
repeat
    Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ . Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$  → resetar modelo y cargar el último modelo guardado
     $t_{start} = t$ 
    Get state  $s_t$ 
    repeat
        Perform  $a_t$  according to policy  $\pi(a_t|s_t; \theta')$ 
        Receive reward  $r_t$  and new state  $s_{t+1}$ 
         $t \leftarrow t + 1$ 
         $T \leftarrow T + 1$ 
    until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
     $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{Bootstrap from last state} \end{cases}$ 
    for  $i \in \{t - 1, \dots, t_{start}\}$  do
         $R \leftarrow r_i + \gamma R$ 
        Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$ 
        Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ 
    end for
    Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ . → única diferencia es asíncrona
until  $T > T_{max}$ 

```

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Conclusiones

- A partir el algoritmo de Policy Gradient, la primera variante que hemos analizado es el algoritmo de Actor-Critic, donde el modelo del agente predice las probabilidades de las acciones (Actor) y el Value del estado (Critic).
- El uso del **Value** como factor de relevancia de las acciones tomadas proporciona **más estabilidad durante el aprendizaje**. Además, hemos introducido una **versión mejorada** a partir del Value y de las recompensas estimadas para una acción específica, como es la función de Ventaja.
- Al incluir multiproceso y el uso de la función de Ventaja en el algoritmo de Actor-Critic, hemos desarrollado los algoritmos de **A2C/A3C para disminuir los tiempos de entrenamiento y facilitar la convergencia de la solución**.
- La principal diferencia entre **A2C y A3C** es que el primero hace actualizaciones del modelo de manera **síncrona** mientras que el segundo las realiza **asíncronamente**.

No son multijugador

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: Actor-Critic

Evolución de Actor-Critic: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

- *Asynchronous methods for deep reinforcement learning*, Mnih V. et al, Arxiv
<https://arxiv.org/abs/1602.01783>
- *OpenAI Baselines: ACKTR & A2C*, OpenAI
<https://openai.com/blog/baselines-acktr-a2c/>

08MIAR-Aprendizaje por refuerzo

**Sesión 7 – Espacios de acciones continuos:
DDPG & PPO**

Curso Abril 23/24

The logo consists of a solid orange circle containing the lowercase letters "viu" in a white, sans-serif font.

viu

**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Introducción

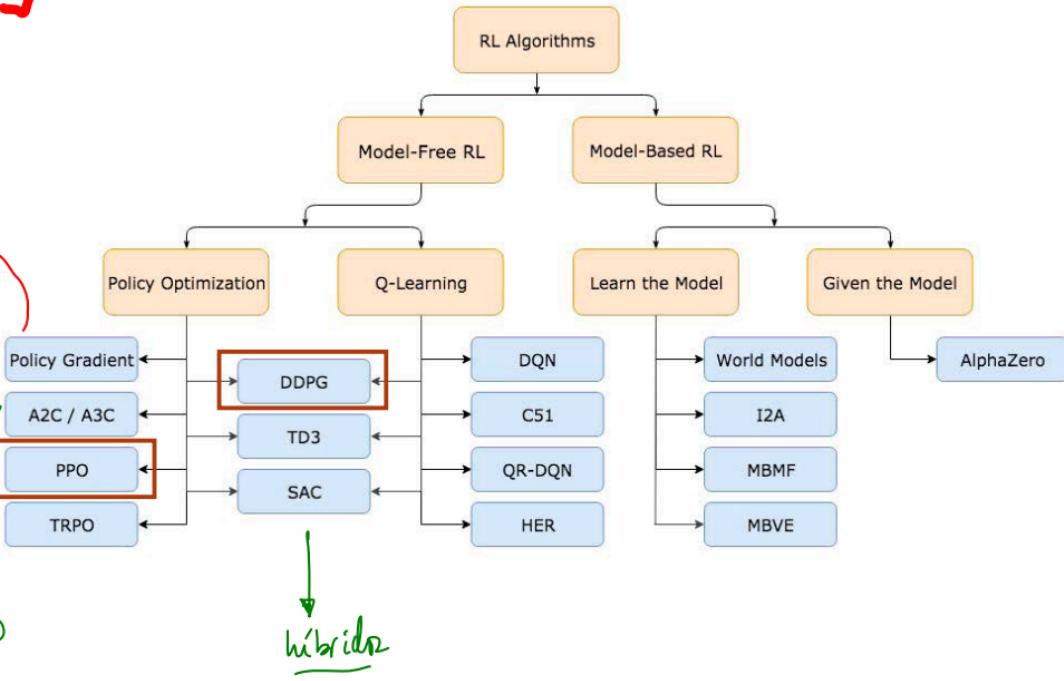
(Examen)

Problemas de PG

- 1.- uso de la recompensa como factor de ponderación
 - gran varianza y mala convergencia
 - " magnitud altas gradientes
 - recompensa irrelevante

- 2.- insuficiente respecto a la memoria

- 3.- lo resuelve el PPO



Introducción

Una vez hemos estudiado las diferentes versiones de algoritmos de la familia de Policy gradients vamos a entrar en detalle en dos soluciones muy extendidas dentro del mundo del aprendizaje por refuerzo, **DDPG y PPO**.

De hecho, PPO se puede interpretar como un siguiente nivel dentro de la evolución natural de la familia de Policy Gradient.

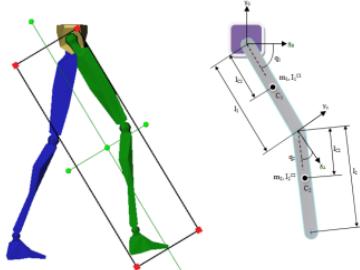
Por su parte, DDPG es un algoritmo que combina características de DQN y de Policy Gradient, sacando partido de ambos enfoques.

Es importante resaltar cómo estos dos algoritmos (al igual que la familia de Policy Gradients) **pueden trabajar en espacios de acciones continuos**. Sin embargo, **DDPG está especialmente diseñado para ello, y PPO es una solución más general**.

Espacio de acciones continuo

Además de desarrollarse en **espacios de acciones continuos**, los algoritmos de DDPG y PPO incluyen **novedades** también desde el punto de vista del **aprendizaje**, del uso que pueden hacer de una ejecución multiproceso o de la combinación de diferentes arquitecturas de modelos que pertenecen a diferentes definiciones.

Estas características hacen que estos dos algoritmos sean muy utilizados en entornos que, además de la necesidad de trabajar en espacios continuos y complejos, necesiten de una **alta carga computacional** en la simulación del entorno, como pueden ser entornos de **robótica** o de **navegación autónoma**.



Movimientos: izq-derecha-arriba-abajo

Discreto – $a = [0 \ 0 \ 1 \ 0]$

Continuo – $a = [-1 \ -0.45 \ 0.8 \ 0.3]$

función de activación
 \tanh

↑
↑
↑
↑
Grados, normalizados

<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQfI5N1crUOkbXLbj4IP5dkr5dCNC9dKzzE2NSVmqlA&s>

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Introducción

Deterministic Policy Gradient Algorithms

David Silver

DeepMind Technologies, London, UK

Guy Lever

University College London, UK

Nicolas Heess, Thomas Degrif, Daan Wierstra, Martin Riedmiller

DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM

GUY.LEVER@UCL.AC.UK

*@DEEPMIND.COM

Abstract

In this paper we consider *deterministic* policy gradient algorithms for reinforcement learning with continuous actions. The deterministic policy gradient has a particularly appealing form: it is the expected gradient of the action-value function. This simple form means that the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, we introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. We demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

case, as policy variance tends to zero, of the stochastic policy gradient.

From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.

In order to explore the full state and action space, a stochastic policy is often necessary. To ensure that our deterministic policy gradient algorithm continues to explore satisfactorily, we introduce an off-policy learning algorithm. The basic idea is to choose actions according to a stochastic behaviour policy (to ensure adequate exploration), but to learn about a deterministic target policy (exploiting the ef-

Deep Deterministic Policy Gradient (DDPG) es un algoritmo que aprende de manera simultánea una función Q y una policy (Actor-Critic por tanto).

Usa una estrategia off-policy y la ecuación de Bellman para modelar la función Q. Para entrenar la policy, aprovecha la función Q modelada como factor de relevancia en la misma.

- DDPG es un algoritmo off-policy (Por lo tanto, no tendremos los discounted rewards calculados)
- Algoritmo híbrido.
- DDPG sólo se puede utilizar en entornos con espacios de acciones continuos
- DDPG se puede ver como la “forma de aplicar Q-learning en espacios de acciones continuos”.

Introducción

Deterministic Policy Gradient Algorithms

David Silver
 DeepMind Technologies, London, UK
Guy Lever
 University College London, UK
Nicolas Heess, Thomas Degrif, Daan Wierstra, Martin Riedmiller
 DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM
 GUY.LEVER@UCL.AC.UK
 *@DEEPMIND.COM

Abstract

In this paper we consider *deterministic* policy gradient algorithms for reinforcement learning with continuous actions. The deterministic policy gradient has a particularly appealing form: it is the expected gradient of the action-value function. This simple form means that the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, we introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. We demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

case, as policy variance tends to zero, of the stochastic policy gradient.

From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.

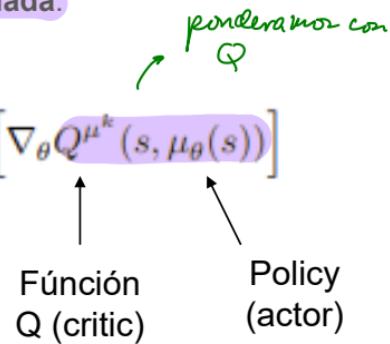
In order to explore the full state and action space, a stochastic policy is often necessary. To ensure that our deterministic policy gradient algorithms continue to explore satisfactorily, we introduce an off-policy learning algorithm. The basic idea is to choose actions according to a stochastic behaviour policy (to ensure adequate exploration), but to learn about a deterministic target policy (exploiting the ef-

En espacio de acciones continuas, utilizar una política voraz para la exploración y maximización de la función Q resulta problemático.

Por el contrario, una alternativa computacionalmente abordable consiste en actualizar la policy en dirección al gradiente de la función Q estimada.

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s \sim \rho^{\mu^k}} [\nabla_{\theta} Q^{\mu^k}(s, \mu_{\theta}(s))]$$

En DQN $Q = f(s)$,
 en DDQN $Q = f(s, \text{acciones})$



Introducción

[Examen]

* Experience replay \Rightarrow off-policy.

* No se sigue una política voraz.

usando
Bellman

Error mediante Bellman para entrenar el critic.

Recompensa esperada a futuro. Utilizamos Q en el estado siguiente. Realmente para esto utilizaremos una target network.



$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t) |_{a=\mu_\theta(s)}$$

(1) Actualizamos Fución Q (critic)

(2) Actualizamos la Policy (actor)

Gradientes de la policy

Fíjate! Obtenemos la Q para las acciones dadas por el actor. Queremos maximizar la Q, por ello ponderamos con sus gradientes, que nos dirigen a maximizarla.

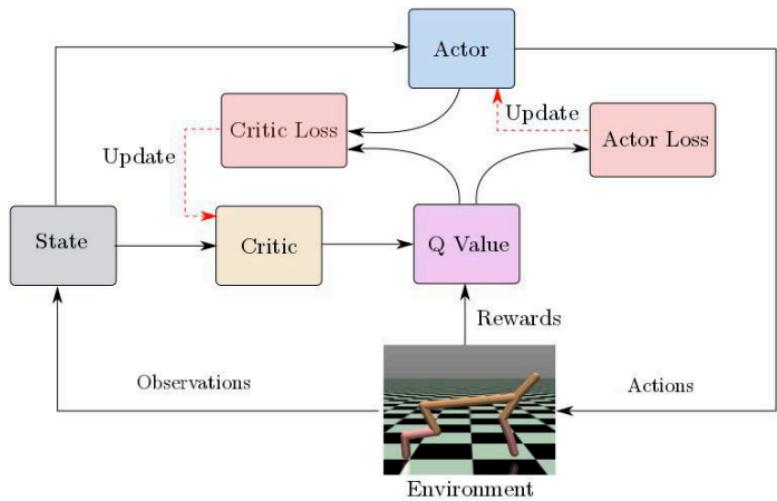
Características principales

Algunas de las características (y decisiones) que se nos presentan con DDPG son:

- El **espacio de acciones es continuo**. Cómo podemos aplicar una estrategia seleccionando la acción que maximice la recompensa esperada en un espacio continuo?
- Al ser **off-policy**, se utilizará un **replay buffer** para almacenar la experiencia que se va acumulando.
- Se definirán **cuatro redes neuronales diferentes**: Dos redes para predicción (**actor y critic networks**) y dos redes target (**actor y critic target networks**).
- El proceso de **exploración es diferente** a los estudiados hasta ahora. Se llevará a cabo incluyendo una **función de ruido en el momento de la selección de la acción**.

Ejemplo: mover el brazo de abajo a arriba, en vez de hacer un movimiento instantáneo de 30° lo hace de 2°, manteniendo así la consistencia temporal

Características principales



```

class Actor(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, learning_rate = 3e-4):
        super(Actor, self).__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.linear3 = nn.Linear(hidden_size, output_size)

    def forward(self, state):
        """
        Param state is a torch tensor
        """
        x = F.relu(self.linear1(state))
        x = F.relu(self.linear2(x))
        x = torch.tanh(self.linear3(x))
        return x
  
```

salida, número de acciones posibles

→ rango entre -1 y 1 o grados por ejemplo

```

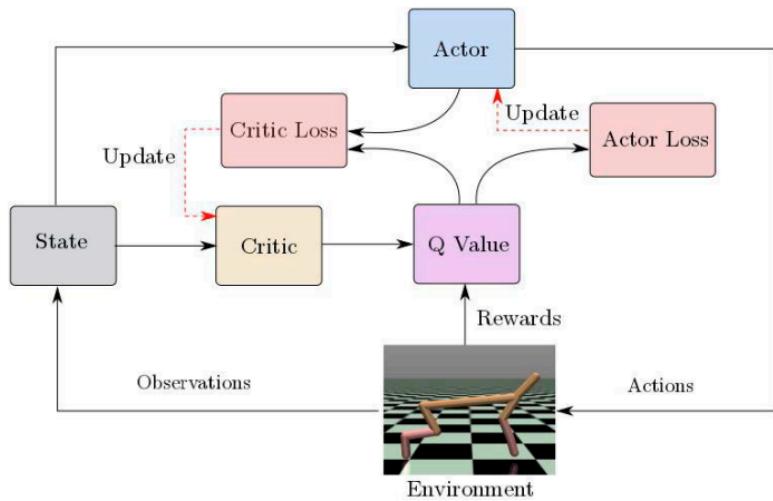
class Critic(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(Critic, self).__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)
        self.linear3 = nn.Linear(hidden_size, output_size)

    def forward(self, state, action):
        """
        Params state and actions are torch tensors
        """
        x = torch.cat([state, action], 1)
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = self.linear3(x)
        return x
  
```

para que state son datos tabulares

→ recompensa => act. lineal

Características principales



```

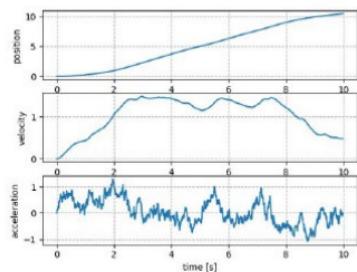
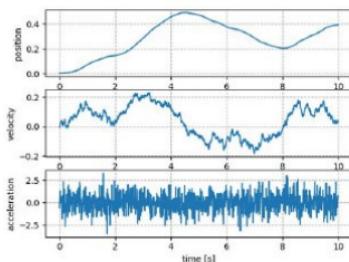
# Critic loss
Qvals = self.critic.forward(states, actions)
next_actions = self.actor_target.forward(next_states)
next_Q = self.critic_target.forward(next_states, next_actions.detach())
Qprime = rewards + self.gamma * next_Q
critic_loss = self.critic_criterion(Qvals, Qprime) ← se computan gradientes aquí

# Actor loss
policy_loss = -self.critic.forward(states, self.actor.forward(states)).mean()
  
```

teachers

NO se computan gradientes aquí

Características principales



El proceso de exploración tipo que encontraremos en el algoritmo será llevado a cabo utilizando el **proceso de Ornstein-Uhlenbeck**.

La idea de este proceso es utilizar un proceso que mantenga una **correlación temporal** para que la exploración sea más suave que con otras posibles distribuciones.

$$\mu'(s_t) = \mu(s_t | \theta_t^\mu) + \mathcal{N}$$

https://en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck_process
<https://www.quora.com/Why-do-we-use-the-Ornstein-Uhlenbeck-Process-in-the-exploration-of-DDPG>

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Algoritmo: DDPG

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
 Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
 Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process \mathcal{N} for action exploration
 Receive initial observation state s_1
 for t = 1, T **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \quad \text{→ exponential moving average} \\ (\text{media móvil exponencial})$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for
end for

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Introducción

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
`{joschu, filip, prafulla, alec, oleg}@openai.com`

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

<https://spinningup.openai.com/en/latest/algorithms/ppo.html>

Sesión 7 – Evolución del algoritmo de Policy Gradient

La idea clave en la que se centra el algoritmo de PPO es:

Cómo podemos mejorar lo máximo posible la policy actual, usando la trayectoria recolectada, controlando que el aprendizaje no colapse?

→ evitar el overfitting.

- Algoritmo basado en otro conocido como TRPO → policy optimization
- PPO es un algoritmo on-policy. Pertenece a la familia de Policy Gradient.
- PPO puede ser utilizado para entornos con espacios de acciones tanto discretos como continuos.
- Normalmente, las implementaciones que encontraremos de PPO utilizan una ejecución multiproceso para tareas de recolección de datos.

Características principales

Algunas de las características (y decisiones) que se nos presentan con PPO son:

- El espacio de **acciones** puede ser **discreto o continuo**.
- Al igual que los otros algoritmos de su familia, sigue una filosofía **on-policy**. Nuevamente, tenemos que decidir el tamaño de la **trayectoria** para almacenar los datos que se utilicen para el entrenamiento. En este caso, tenemos que tener en cuenta que la ejecución **multiproceso** aumenta el volumen recolectado.
- La arquitectura de modelo utilizada es similar a **Actor-Critic**.
- La **exploración** se realiza de la misma manera que los algoritmos Actor-Critic, utilizando una **distribución de probabilidad aleatoria** ponderada.

Objetivo de PPO

Algunas de las **limitaciones que presenta policy gradients** (de algunas ya hemos hablado):

- **Ineficacia de las muestras:** Las muestras sólo se utilizan una vez. Después, la política se actualiza y la nueva política se utiliza para muestrear otra trayectoria. Como el muestreo suele ser caro, esto puede resultar prohibitivo. → A2C/A3C multiproceso
- **Alta varianza de la recompensa:** El gradiente de la política es un enfoque de aprendizaje Monte Carlo, que tiene en cuenta la trayectoria completa de la recompensa (es decir, un episodio completo). Estas trayectorias suelen tener una gran varianza, lo que dificulta la convergencia. → Actor-Critic , función *baseline*
- **Actualizaciones de políticas incoherentes:** las actualizaciones de políticas tienden a sobrepasar y no alcanzar el pico de recompensa, o a detenerse prematuramente. Especialmente en las arquitecturas de redes neuronales, los gradientes desvanecientes y explosivos constituyen un grave problema. **El algoritmo puede no recuperarse de una mala actualización.** --> PPO

Características principales

La mayor diferencia aparece en la función de coste, PPO ofrece dos versiones diferentes:

- Una función de coste basada en **Kullback-Leibler divergence** (*intenta matchear una distribución con otra*)
- Una función de coste basada en **Clipped surrogate objective**

Será la **segunda opción** la que trataremos en la asignatura.

Características principales

La primera diferencia que introduce esta función es la **sustitución del logaritmo de la probabilidad de la acción por el ratio de la probabilidad de la acción con la policy actual en comparación con la versión de la policy anterior.**

Este **ratio** será mayor que **1** cuando la **acción es más probable con la nueva policy** y estará entre **0 y 1** cuando la **acción sea menos probable**.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

función advantage.

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_t \left[\underbrace{\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta old}(a_t | s_t)}}_{\text{ratio de la policy}} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

policy base, el valor de probabilidad base

valor nuevo (última época)

<https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl>

Características principales

Para **estabilizar** el proceso de aprendizaje (con probabilidades que sean muy grandes), PPO introduce **dos límites** para los casos en los que este ratio se dispare y, por tanto, los cambios en la policy sean muy drásticos.

Básicamente, junto al ratio calculado se define un rango (a partir de un **nuevo hiperparámetro**, con **valor** por defecto **0.1/0.2**) para que la actualización de los pesos del modelo se lleve a cabo con el valor mínimo de este rango. **Así aseguramos que la actualización se va haciendo de forma “suave” y en una buena dirección.**

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\overline{r_t(\theta) \hat{A}_t}, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right]$$

same objective same, but $r(\theta)$ is clipped
from before between $(1 - \epsilon, 1 + \epsilon)$

min of the same objective from before and the clipped one

Características principales

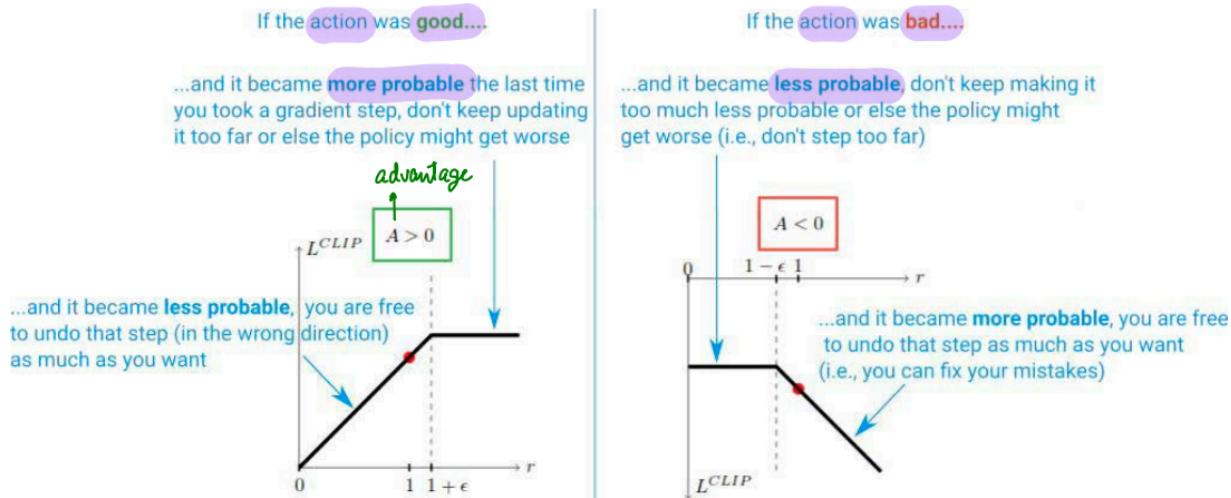


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r , for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

<https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl>

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Algoritmo: PPO

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do** *episode*
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\overbrace{\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}}^{\text{polic. adv.}}, \overbrace{g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))}^{\text{CLIP}} \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**

actualizar
la red

CLIP

se actualiza
el Critic

[Examen]

¿Todas las algoritmos basados en Policy Gradients utilizan el logaritmo de la policy para actualizar los pesos?
No, hay algunos que utilizar un ratio (PPO)

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Alternativas a DDPG y PPO

trata de regularizar la policy forzando a que la distribución de probabilidades sea homogénea

<https://arxiv.org/abs/1801.01290>

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja[†] Aurick Zhou[†] Pieter Abbeel[†] Sergey Levine[†]

Abstract

Model-free deep reinforcement learning (RL) algorithms have been demonstrated on a range of challenging decision making and control tasks. However, these methods typically suffer from two major challenges: very high sample complexity and brittle convergence properties, which necessitate meticulous hyperparameter tuning. Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this paper, we propose soft actor-critic, an off-policy actor-critic deep RL algorithm based on the

of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. Second, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

$$J(\pi) = \sum_{t=0}^T \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))]$$

↳ integró de la propia policy, para regular las probabilidades

en off-policy

mejora la gestión de la memoria, de tal forma que guarda algunos casos en los que la policy se ha equivocado generando una especie de data-augmentation para combinarla con otras experiencia donde la policy ha ido bien ⇒ factor de regularización (valor error de la policy)

<https://arxiv.org/abs/1707.01495>

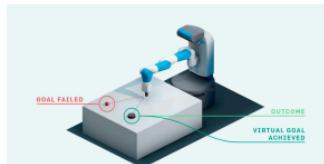
Hindsight Experience Replay (HER)

Marcin Andrychowicz*, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong,
Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel†, Wojciech Zaremba†
OpenAI

Abstract

Dealing with sparse rewards is one of the biggest challenges in Reinforcement Learning (RL). We present a novel technique called *Hindsight Experience Replay* which allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering. It can be combined with an arbitrary off-policy RL algorithm and may be seen as a form of implicit curriculum.

We demonstrate our approach on the task of manipulating objects with a robotic arm. In particular, we run experiments on three different tasks: pushing, sliding, and pick-and-place, in each case using only binary rewards indicating whether or not the task is completed. Our ablation studies show that Hindsight Experience Replay is a crucial ingredient which makes training possible in these challenging environments. We show that our policies trained on a physics simulation can be deployed on a physical robot and successfully complete the task. The video presenting our experiments is available at <https://goo.gl/SMrQnI>.



Sample a set of additional goals for replay $G := S(\text{current episode})$ for $g' \in G$ do
 $r' := r(s_t, a_t, g')$
 Store the transition $(s_t || g', a_t, r', s_{t+1} || g')$ in R
 end for

Se hace experience replay.

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Conclusiones

- Siempre que trabajemos en espacios de acciones continuos tendremos que utilizar algoritmos de la **familia de Policy Gradients** (como PPO) o utilizar **enfoques híbridos** que nos permitan utilizar funciones Q (como DDPG).
- Los **espacios de acciones continuos** son muy comunes en entornos **complejos** y que necesitan de **muchas capacidades computacionales**, como por ejemplo **robótica** o **conducción autónoma**.
- Además de funcionar muy bien en espacios de acciones continuos, **PPO es uno de los algoritmos que forman parte del estado del arte actual en multitud de retos**. Además, principalmente por su implementación intuitiva, hace que sea uno de los algoritmos por defecto cuando implementamos una solución de aprendizaje por refuerzo.

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: DDPG

Proximal Policy Optimization (PPO)

Algoritmo: PPO

Alternativas a DDPG y PPO

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

- *Learning Dexterity*, OpenAI

<https://openai.com/blog/learning-dexterity/>

- *What is the way to understand Proximal Policy Optimization Algorithm in RL?*, Cyberman, A.

Stackoverflow

<https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl>

08MIAR-Aprendizaje por refuerzo

Sesión 10 – Algoritmos basados en modelo

Curso Abril 23/24

The logo consists of the lowercase letters "viu" in white, sans-serif font, centered within a solid orange circle.

viu

**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

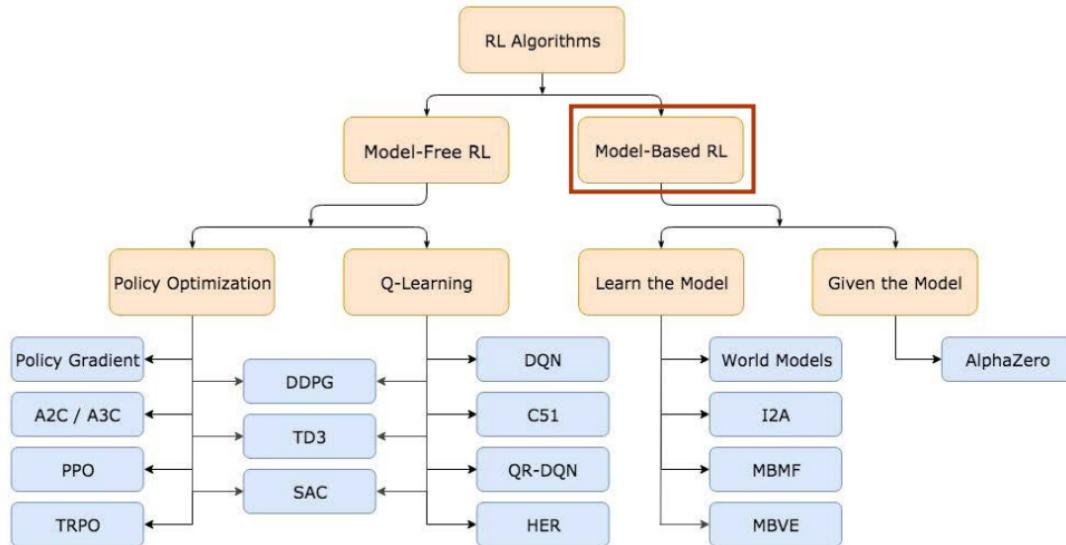
Introducción

*The next big step forward in AI will be systems that actually understand their worlds. The world is only accessed through the lens of experience, so to understand the world means to be able to **predict and control your experience**, your sense data, with some accuracy and flexibility. In other words, understanding means forming a predictive model of the world and using it to get what you want. **This is model-based reinforcement learning.***

Richard Sutton

<https://medium.com/the-official-integrate-ai-blog/understanding-reinforcement-learning-93d4e34e5698>

Introducción



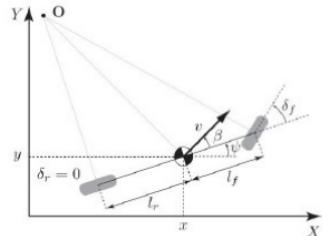
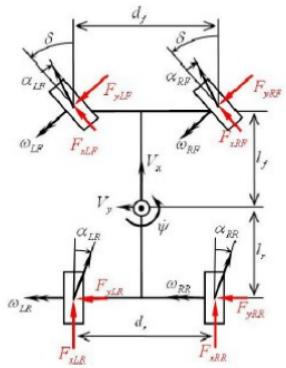
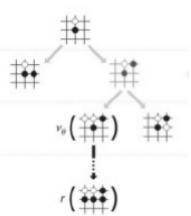
Introducción

La principal diferencia entre soluciones *model free* y *model based* es el **conocimiento a priori de las dinámicas del entorno**.

Si se conocen las dinámicas del entorno, se pueden **estimar las transiciones que el agente puede ejecutar** desde el estado actual para **valorar cuál es la mejor acción a tomar**.

Actualmente, uno de los **retos** principales aparece con entornos en los que el **espacio de acciones es muy elevado** y, por tanto, esta estimación es muy demandante en cuanto a **recursos computacionales** necesarios.

Introducción



<https://jonathan-hui.medium.com/rl-model-based-reinforcement-learning-3c2b6f0aa323>

Sesión 10 – Algoritmos basados en modelo

Introducción

En esta asignatura nos centraremos en un tipo específico de implementación: **acompañar el aprendizaje del modelo del agente con metaheurísticas u otras técnicas de optimización**, que sirvan de apoyo al proceso de aprendizaje.

Dentro de las opciones que podemos encontrar, las más utilizadas son la **búsqueda en árbol o algoritmos basados en población (genéticos)**. Todos ellos basados a su vez en simulaciones de **Montecarlo** para poder analizar los distintos caminos disponibles.

Introducción

Además, otro punto de vista de **Model based** es su conexión directa con **problemas de control** por computador. En este sentido, es común ver el **modelo como un controlador** y que el problema se transforme en “**minimizar el error cometido**” en vez de “maximizar la recompensa esperada”.

Por último, hay un aspecto muy importante a tener en cuenta en relación con la recompensa. Hasta ahora, la recompensa nos ha venido dada como valores que el entorno nos devuelve de manera arbitraria. **En soluciones basadas en modelo es común conocer la estructura o función de recompensa**, lo que complementa al uso de técnicas de optimización a la hora de estimar las acciones.

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

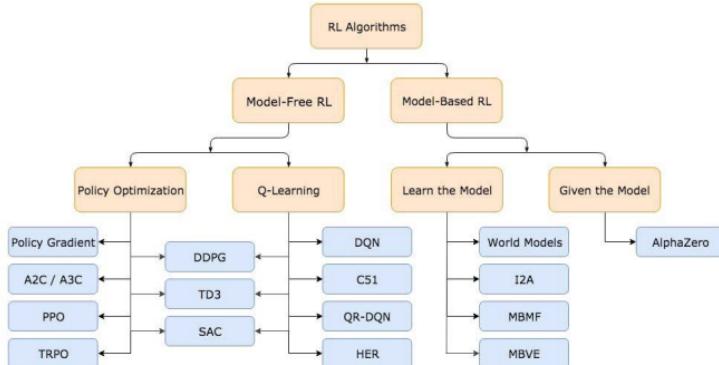
Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Model based: Modelo conocido vs. Modelo desconocido



Cuando hablamos de soluciones basadas en modelo podemos encontrarnos en dos situaciones diferentes: que el **modelo sea conocido o no**.

Si el modelo es **conocido**, podremos aplicar directamente las técnicas de **metaheurísticas** vistas anteriormente.

Si el modelo es **desconocido**, tendremos que llevar a cabo un paso previo en la ejecución para aprender o **aproximar** de la mejor manera posible el modelo del **entorno**.

Model based: Modelo conocido vs. Modelo desconocido

Como hemos comentado, si el **modelo es conocido** entonces podemos utilizar ese conocimiento para enriquecer el proceso de entrenamiento. Esta es la situación típica con **metaheurísticas, planificación y otras técnicas de optimización**. Este es también el tipo de ejecución en la que se basan soluciones como **Alphago o Alphazero**.

El elemento principal a tener en cuenta en este tipo de soluciones es la **capacidad computacional** necesaria para la implementación. Los ejemplos de Alphago y Alphazero muestran esta problemática, ya que aunque podamos estimar las transiciones del entorno de una manera precisa, es imposible poder llevar todas las estimaciones posibles en cada momento de la ejecución.

Model based: Modelo conocido vs. Modelo desconocido

Si el **modelo es desconocido**, tendremos que llevar a cabo una **fase previa de aprendizaje o aproximación**. Lo común es utilizar un **enfoque de aprendizaje supervisado/no supervisado** para crear el modelo del entorno **utilizando datos recolectados**. Una vez tengamos esta aproximación, podemos utilizarla en el algoritmo de aprendizaje por refuerzo que deseemos.

A diferencia del uso de metaheurísticas, este tipo de solución Model-based tiene su base en **algoritmos/procesos basados en gradientes**, debido a la necesidad de optimizar la representación que se va obteniendo del entorno.

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Model learning: Aspectos principales

Model-based Reinforcement Learning: A Survey.

Thomas M. Moerland^{1,2}, Joost Broekens², and Catholijn M. Jonker^{1,2}

¹ Interactive Intelligence, TU Delft, The Netherlands

² LIACS, Leiden University, The Netherlands

Model learning: Aspectos principales

El primer aspecto a analizar es el tipo de **modelo que se va a estimar a partir del entorno**. Podemos encontrar tres tipos principales:

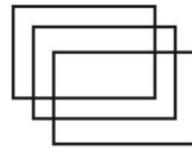
- 1) Modelo forward $(s_t, a_t) \rightarrow s_{t+1}$
- 2) Modelo backward $s_{t+1} \rightarrow (s_t, a_t)$
- 3) Modelo inverso $(s_t, s_{t+1}) \rightarrow a_t$

Model learning: Aspectos principales

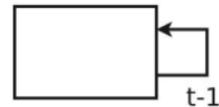
Las **observaciones parciales** se refieren a cuando la **información** disponible en una observación **no es completa**. Es una situación muy típica que puede ocurrir por la propia naturaleza del problema. Algunas formas de suavizar esta situación son:



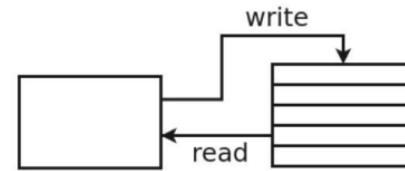
Single state



Window



Recurrency



Neural Turing Machine

Model learning: Aspectos principales

Uno de los **puntos más críticos** cuando se aproxima el modelo de un entorno es la **abstracción de los estados**. Debido a la alta carga de información de este tipo de problemas, **el modelo del entorno siempre va a ser una representación del mismo**.

Normalmente, esta representación va asociada a algún método o proceso de abstracción, desde una reducción de dimensionalidades hasta el uso de alguna **arquitectura de Deep Learning**.

Model learning: Aspectos principales

Otros aspectos analizados en el artículo de referencia son la **incertidumbre, la aleatoriedad o la estacionalidad de los datos**.

Incertidumbre y aleatoriedad son conceptos que están muy relacionados entre sí, ya que la incertidumbre trata la falta de información en los datos para aproximar un modelo preciso, mientras que la aleatoriedad trata la propia naturaleza estocástica del proceso que se está modelando. La aleatoriedad siempre va a estar presente mientras que la incertidumbre se podría disminuir teniendo más datos disponibles.

La estacionalidad se refiere a si cambian las funciones de transición o recompensa a lo largo del tiempo. Estos cambios, si no son percibidos por el agente, pueden conllevar a que la solución se desgaste con el paso del tiempo.

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Planning & Learning

Model-based Reinforcement Learning: A Survey.

Thomas M. Moerland^{1,2}, Joost Broekens², and Catholijn M. Jonker^{1,2}

¹ Interactive Intelligence, TU Delft, The Netherlands

² LIACS, Leiden University, The Netherlands

Planning & Learning

A la hora de aplicar **planificación** junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) ¿En qué estado comenzamos la planificación?
- 2) ¿Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) ¿Cómo llevamos a cabo la planificación?
- 4) ¿Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) ¿En qué estado comenzamos la planificación?
- 2) ¿Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) ¿Cómo llevamos a cabo la planificación?
- 4) ¿Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- **Estado aleatorio**
- **Estado visitado**
- **Estado prioritario**
- **Estado actual**

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) ¿En qué estado comenzamos la planificación?
- 2) **¿Al ejecutar simulaciones, cuánta performance reservamos para la planificación?**
- 3) ¿Cómo llevamos a cabo la planificación?
- 4) ¿Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- **Después de cuántos steps comenzamos?**
 - **Cuánto esfuerzo ejecutamos por iteración?**
 - (AlphaGo Zero, 1MCTS, 1600 trazas de profundidad 200)**

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) ¿En qué estado comenzamos la planificación?
- 2) ¿Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) ¿Cómo llevamos a cabo la planificación?**
- 4) ¿Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- **De tipo discreto (árbol, tabla) o diferencial (modelos de transición y recompensa diferenciables)**
- **La dirección puede ser forward o backward**
- **Tenemos que definir los niveles de profundidad y anchura en la simulación**

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) ¿En qué estado comenzamos la planificación?
- 2) ¿Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) ¿Cómo llevamos a cabo la planificación?
- 4) **¿Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?**

- **Usando la policy y el value para llevar a cabo la planificación.**
- **Usando el resultado de la planificación para actualizar la policy y el value.**
- **Usando directamente la planificación para seleccionar una acción.**

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

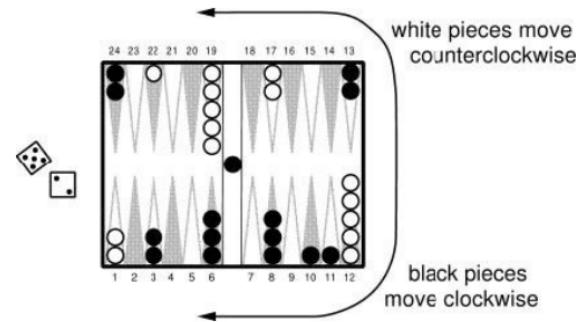
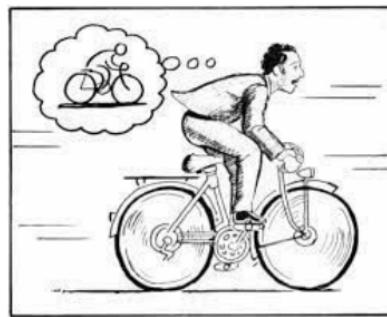
Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Ejemplos de soluciones Model based

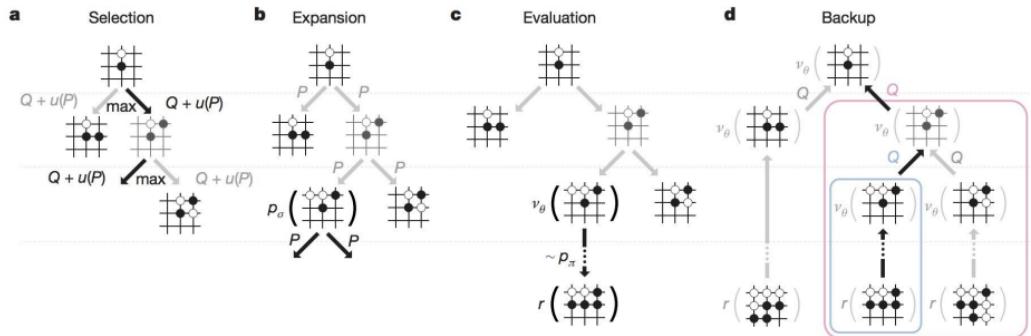


https://es.m.wikipedia.org/wiki/Archivo:Alphago_logo_Reversed.svg
<http://incompleteideas.net/book/ebook/node108.html>
<https://arxiv.org/pdf/1803.10122.pdf>

Ejemplos de soluciones Model based

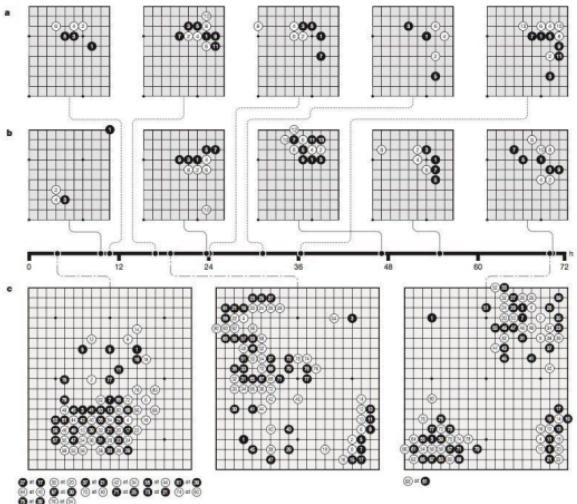


- Solución entrenada con **aprendizaje supervisado y aprendizaje por refuerzo**
- Utiliza **MCTS** para la **estimación de qué acción escoger** en cada estado



https://es.m.wikipedia.org/wiki/Archivo:Alphago_logo_Reversed.svg
<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

Ejemplos de soluciones Model based



En su siguiente versión, Alphago Zero, el entrenamiento de la solución se produce mediante ***self-play***, sin incluir la primera fase de aprendizaje supervisado.

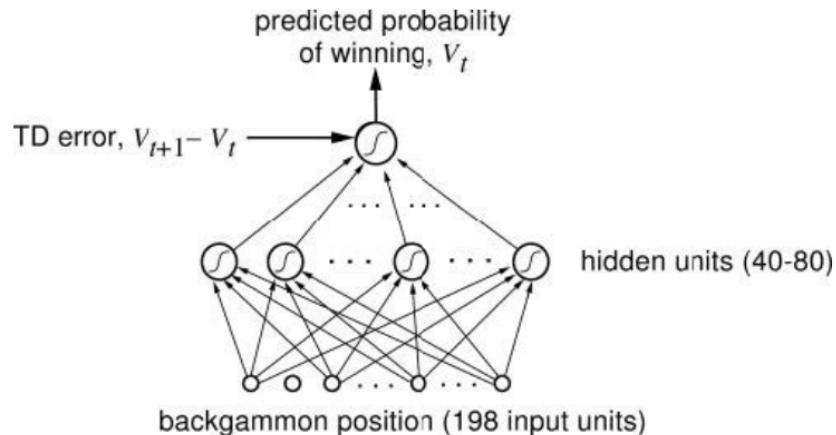
Como versión siguiente, Deepmind desarrolla AlphaZero, para generalizar a los juegos del ajedrez y Shogi además del Go.

Como curiosidad, se estima que **Alphago Zero costó unos 35M\$ en recursos computacionales***

<https://www.xataka.com/robotica-e-ia/la-nueva-alphago-esta-un-paso-mas-cerca-de-la-singularidad-aprende-de-si-misma-y-deja-en-ridiculo-a-la-anterior>

*<https://www.yuzeh.com/data/agz-cost.html>

Ejemplos de soluciones Model based

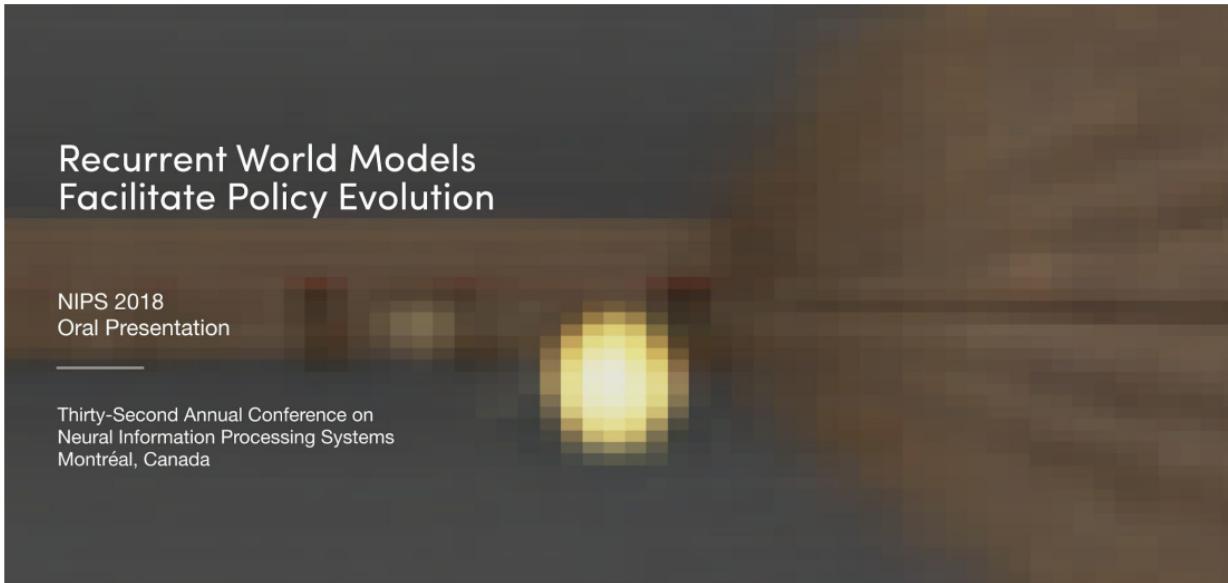


- **TD-Gammon**, desarrollado por Gerry Tesauro (Tesauro, 1992, 1994, 1995).
- A diferencia de otras soluciones de la época, **utiliza como datos de entrada los datos en crudo del tablero**.
- Otro elemento diferencial es la **función de evaluación** utilizada, **basada en diferencias temporales** para optimizar la red neuronal con la que se approxima la probabilidad de ganar.

<https://en.wikipedia.org/wiki/TD-Gammon>

<http://incompleteideas.net/book/ebook/node108.html>

Ejemplos de soluciones Model based



<https://worldmodels.github.io/>

Sesión 10 – Algoritmos basados en modelo

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Conclusiones

- 1) La principal diferencia entre model-free y model-based es que con **model-based conocemos las dinámicas del entorno**, por ejemplo en cuanto a transiciones y funciones de recompensa.
- 2) Dentro de model-based, podemos encontrarnos **con dos situaciones diferentes: que el modelo del entorno sea conocido o que sea desconocido** y, por tanto, tengamos que aproximarla.
- 3) Dentro de las metaheurísticas usadas en estas soluciones, las implementaciones más comunes combinan técnicas de **planificación junto con simulaciones de Montecarlo**.

Índice

Introducción

Model based: Modelo conocido vs. Modelo desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones Model based

Conclusiones

Bibliografía recomendada

Bibliografía recomendada

- “Model-Based Reinforcement Learning: A survey”, Moerland, T. et al
<https://arxiv.org/pdf/2006.16712.pdf>

- “AlphaGo”, Google Deepmind
<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

- “World Models”, David Ha, Jürgen Schmidhuber
<https://arxiv.org/abs/1803.10122>
<https://worldmodels.github.io/>



viu

Universidad
Internacional
de Valencia