Aprendizaje en MAS

Índice

Link vídeo: https://youtu.be/EslM5-R1I8U

Introducción

Un agente es un sistema informático que es capaz de actuar de forma autónoma en representación de su usuario o propietario. El agente tiene la capacidad de actuar, es decir, de modificar su entorno mediante acciones que tienen precondiciones. El agente ha de decidir entre sus posibles acciones cuál debería ejecutar para maximizar un resultado esperado. En la práctica, hay ciertas situaciones en las que varios usuarios interactúan entre ellos, por lo que no podemos representar estas situaciones con un único agente, ya que cada usuario puede tener objetivos y motivaciones independientes.

Los sistemas multiagente o MAS (por sus siglas en inglés: Multi-Agent System) surgen de la necesidad de solucionar problemas que no pueden ser representados o resueltos con un único agente. Un MAS es un sistema compuesto por múltiples agentes inteligentes que interactúan entre ellos. Para interactuar con éxito, los agentes deberán ser capaces de cooperar y negociar con el resto de agentes. Algunos campos de aplicación de este modelo incluyen el transporte, la logística, la gestión de recursos, las comunicaciones y la simulación.

Los agentes son entrenados a través de algoritmos que, a partir de la experimentación y del análisis de grandes cantidades de datos, les preparan para la toma de decisiones. Mediante el aprendizaje, un agente puede descubrir y explotar las dinámicas de su entorno y adaptarse a circunstancias no previstas durante la realización de su tarea. Este aprendizaje es más difícil en los MAS, ya que los agentes existen en un entorno dinámico formado por todos los agentes, cada uno con sus propios objetivos y supuestos.

El aprendizaje multiagente o MAL (Multi-Agent Learning en inglés), es la aplicación del aprendizaje automático (Machine Learning) en un sistema multiagente. En nuestro trabajo vamos a revisar la evolución de las técnicas y enfoques aplicados al aprendizaje en MAS, así como las nuevas corrientes de investigación. También estudiaremos algunas de las aplicaciones en el ámbito de los juegos y en el mundo real del MAL.

Primeros Enfoques en MAL

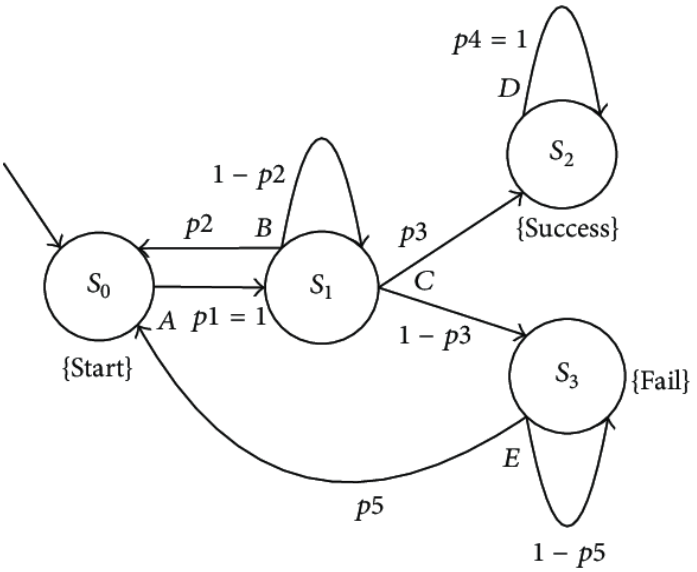
El aprendizaje en un entorno multiagente es esencialmente más complejo que en el caso de un único agente. Esto es compresible dado que un agente tiene que interactuar con su entorno y posiblemente otros agentes.

Inicialmente se intentó aplicar algoritmos diseñados para agentes independientes en los MAS a pesar de no cumplir algunas propiedades y asunciones fundamentales, dado que el entorno en el que interactúa el agente ya no es estático. Sorprendentemente, estos agentes conocidos como aprendices independientes se han utilizado con éxito en la práctica gracias a la buena escalabilidad que poseen. Después de esta primera aproximación se empezaron a desarrollar algoritmos más enfocados a sistemas multiagente para que un agente pudiese aprender e interactuar con otros agentes de su entorno. A continuación veremos Q-Learning, un algoritmo utilizado para agentes independientes, Minimax-Q y Opponent Modelling, algoritmos diseñados para su uso en MAS.

Antes de explicar estos algoritmos deberíamos introducir unos conceptos importantes para entender su funcionamiento.

MDP

Para empezar, los MDPs (Markov Decision Process) son un framework multiestado para agentes únicos utilizado para modelar la toma de decisiones en situaciones en las que los resultados son en parte aleatorios y en parte bajo el control del decisor. En cada paso de tiempo, el proceso está en un estado ***s*** y el decisor puede elegir cualquier acción ***a*** que está disponible en ese estado ***s***. El proceso responde al siguiente paso moviéndose a un estado aleatorio ***s'***, y dándole la recompensa correspondiente al decisor. La función de transición define una distribución probabilística sobre el siguiente estado ***s'*** que depende del estado actual y la acción del decisor. Toda transición de un proceso de decisión de Markov satisface la propiedad de Markov ya que cada transición solo toma en cuenta su estado actual y las acción que realiza, es decir, que no hay que guardar información de estados anteriores. [1]



Juegos Matriciales

También existen los juegos matriciales que son un framework de estado único multiagente. En estos juegos, los agentes eligen una de las acciones disponibles y reciben una recompensa que depende del resto de los agentes. Cada agente trata de buscar una estrategia que maximice su recompensa. A diferencia de los procesos de decisión de Markov, no se puede solucionar un juego matricial sin conocer previamente las estrategias de los otros agentes. [1]

		JUGADOR 2	
		Estrategia A	Estrategia B
JUGADOR 1	Estrategia A	p_{1A}, p_{2A}	p_{1A}, p_{2B}
	Estrategia B	p_{1B}, p_{2A}	p_{1B}, p_{2B}

Juegos Estocásticos

Ahora bien, los juegos estocásticos se pueden tomar como una combinación de los MDPs y los juegos matriciales de modo que es un framework multiestado multiagente. Un juego estocástico es una tupla ***(n,S,A_i,T,R_i)*** donde ***n*** es el número de jugadores, ***S*** es el conjunto

de estados, A_i es el conjunto de acciones disponibles para el i-ésimo jugador, T es la función de transición y R_i la función de recompensa para el i-ésimo jugador. El objetivo para cada agente es encontrar una política que maximice su futura recompensa. [1]

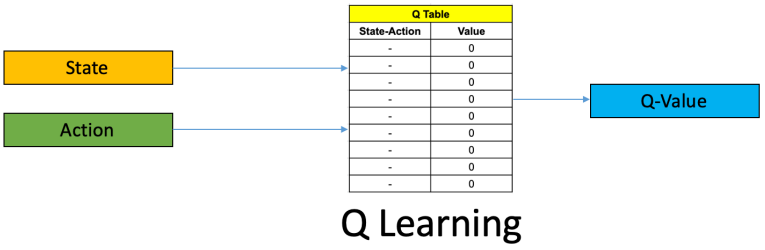
Propiedades

Hay dos propiedades que son deseables que tenga cualquier algoritmo de aprendizaje multiagente. En primer lugar, la convergencia que especifica que el agente llegará a converger en una política estacionaria, es decir, una política que decide la misma acción para cada estado independientemente del tiempo. Y la racionalidad que define que si las políticas de los otros agentes convergen en políticas estacionarias, el algoritmo de aprendizaje llegará a converger en una política de mejor respuesta a las de los otros agentes.

Una vez ya establecidos estos conceptos podemos introducir las primeras aproximaciones de aprendizaje en sistemas multiagente. [1]

Q-Learning

Como ya hemos mencionado, Q-Learning fue diseñado para agentes únicos con el fin de encontrar políticas óptimas en MDPs. Pero cuando los otros agentes utilizan una estrategia estacionaria, el juego estocástico se parece a un MDP y, por tanto, Q-Learning encontrará la respuesta óptima a las acciones de los otros jugadores, haciéndolo racional. En cambio, Q-Learning no es convergente porque no emplea políticas estocásticas. A pesar de esto, Q-Learning fue utilizado como una base para muchos algoritmos utilizados en entornos multiagente como Minimax-Q, Nash-Q, Friend-or-Foe Q-Learning… [1]



Q Learning

Minimax-Q

Minimax-Q es una extensión de Q-Learning que utiliza el concepto de equilibrio en teoría de juegos con el fin de estimar el valor de un estado. Este valor se utiliza para actualizar el valor de los estados al que llega mediante una transición y así sucesivamente. Empleando esta metodología, Minimax-Q consigue aprender el equilibrio Nash de la parte del agente. Con la condición de que todo los jugadores lleguen a explorar todas las acciones y los estados, Minimax-Q es un algoritmo independiente de los otros jugadores para aprender la solución de equilibrio. Por tanto, este algoritmo es convergente. Sin embargo, no es racional porque hay situaciones en el que no produce la mejor respuesta al oponente. [1]

Opponent Modeling

Opponent Modeling, también conocido como Joint-Action Learners (JALs), es un algoritmo que se basa en la idea de aprenderse los modelos explícitos de los otros agentes o jugadores bajo la suposición de que aplican una política estacionaria. Basándose en jugadas pasadas, el agente estimará la siguiente acción del oponente y responderá de forma óptima en la siguiente jugada.

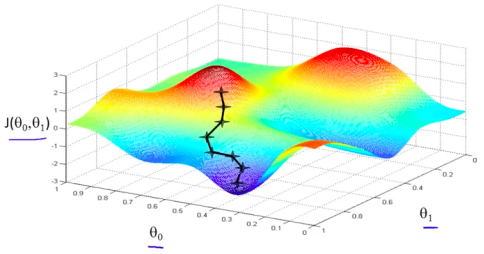
Opponent modeling es racional porque llegará un momento en el que las estimaciones del agente sobre la política del oponente convergen en una política verdadera. Y dado que las políticas que encuentra son de mejor respuesta, llegará un momento en el que el algoritmo converge en una política de mejor respuesta a la política verdadera del oponente. En cambio, este algoritmo no es convergente ya que solo aplica políticas puras y no puede converger en juegos con equilibrio mixto. [1]

Principio “Win or Learn Fast” (WoLF)

El principio “Win or Learn Fast” o WoLF, es una técnica de aprendizaje para agentes en sistemas multiagente que utiliza un ritmo de aprendizaje variable. Esta técnica fue desarrollada en el año 2002 por M. Bowling y M. Veloso tras observar que los algoritmos previamente definidos para el aprendizaje no eran capaces de converger en una política de actuación óptima en sistemas multiagente.

Ascenso del gradiente

El ascenso del gradiente es un algoritmo iterativo de optimización para encontrar el máximo de una función. Es el algoritmo opuesto al descenso del gradiente, conocido por ser el algoritmo de retropropagación en el entrenamiento de redes neuronales artificiales.



A grandes rasgos, el algoritmo del ascenso del gradiente realiza los siguientes pasos de forma iterativa: calcula las derivadas parciales de la función para cada una de las dimensiones, dando como resultado un vector de dirección, el gradiente. El gradiente nos indica la dirección hacia la que la pendiente asciende. Conociendo este vector, actualizaremos los parámetros de la función en base al gradiente multiplicado por una constante, el ratio de aprendizaje. Esta constante, la magnitud en la que se actualizan los parámetros de la función en cada iteración del algoritmo. El algoritmo terminará cuando hayamos encontrado el máximo de la función (convergencia) o cuando la mejora a través del gradiente sea menor que cierto umbral.

Aplicado al entrenamiento de agentes, la función sobre la que aplicaremos el algoritmo del ascenso del gradiente es la función que dada una acción en una situación determinada nos devuelve el beneficio esperado para el agente. El algoritmo del ascenso del gradiente en este caso actualizará los parámetros que ponderan la frecuencia de la realización de las

distintas acciones para maximizar el beneficio. En el marco de los sistemas multiagente tendremos que tener en cuenta las acciones de todos los agentes involucrados a la hora de elaborar nuestra función. La función tomará como entrada una estrategia, es decir, el conjunto de acciones tomadas por todos los agentes para calcular el beneficio esperado para cada uno de los agentes. [1]

Ritmo de aprendizaje variable

Como hemos determinado en el apartado anterior, el gradiente indica la dirección del espacio hacia la que orientamos los parámetros del modelo para encontrar el máximo local, pero el ratio de aprendizaje determina la magnitud en la que variamos estos parámetros.

En el principio WoLF el ratio de aprendizaje no es constante, si no que cambia según el resultado de la iteración actual en el proceso de entrenamiento. El resultado de la iteración se puede definir en los términos de “victoria” y “derrota” en el campo de los juegos o las competiciones entre varios agentes, pero puede ser extrapolado para otro tipo de aplicación del agente que estamos entrenando según su desempeño. Si consideramos que el resultado de la iteración ha sido una victoria, el agente aprenderá de forma cautelosa, es decir el valor del ratio de aprendizaje será bajo. Esto se debe a que los otros agentes serán más propensos a cambiar sus políticas de actuación para intentar contrarrestar nuestra estrategia. De otro modo, si el agente tiene un comportamiento que no conduce a un resultado favorable, sus parámetros se actualizarán con un ratio de aprendizaje más elevado, es decir, aprenderá más rápido.

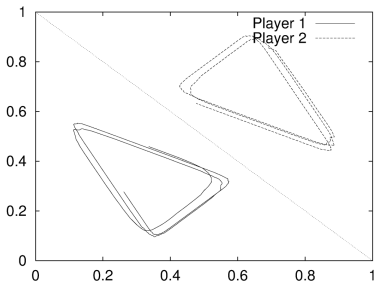
Aunque este algoritmo pueda parecer bastante simple al leer una definición a grandes rasgos, tiene varios requerimientos estrictos que dificultan su funcionamiento. En primer lugar, hemos de definir qué constituye una victoria. Podemos considerar que nuestro agente gana si su estrategia actual obtiene mejores resultados contra la estrategia actual de su oponente que con una estrategia que conduce al equilibrio. Para poder determinar qué estrategia conduce al equilibrio el agente debe conocer la matriz de recompensas, es decir, para cada acción de nuestro agente que recompensa obtendrá según las acciones llevadas a cabo por sus oponentes. Otro requerimiento es conocer la distribución de las acciones de los oponentes. Ambos requerimientos exigen tener una información que normalmente no es conocida y a menudo deben ser obtenida mediante el aprendizaje. Sin embargo, en muchos casos, sólo podemos conocer la acción tomada por el oponente, y a veces, ni siquiera disponemos de esta información. [1]

Política de escalada (PHC)

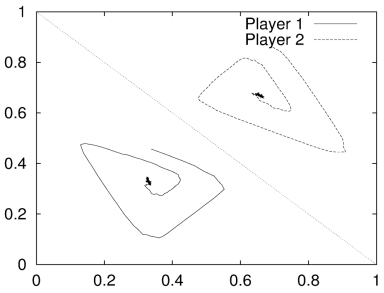
Para evitar estos requerimientos tan estrictos, existe un algoritmo alternativo que usa la política de escalada en combinación con el ratio de aprendizaje variable del principio WoLF para converger en una política de actuación ganadora. Este algoritmo parte de una versión del Q-Learning modificado, ya que tiene ratio de aprendizaje variable, pero conserva las mismas propiedades que las definidas en la sección de Q-Learning de la sección 2.

Con esta base definiremos el algoritmo WoLF-PHC, que es capaz de converger sin sacrificar la propiedad de racionalidad introducida por el algoritmo Q-Learning. Para esto necesitamos concretar la manera en la que determinaremos si el agente está ganando, para

poder modificar el ratio de aprendizaje acorde al principio WoLF. Consideraremos que el agente gana cuando el valor esperado de su política actual es mayor que el de la política media (calculada iterativamente durante la ejecución del algoritmo). [1]



Policy Hill-Climbing

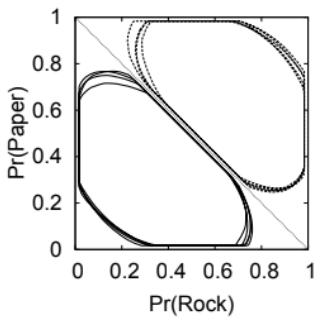


WoLF-PHC

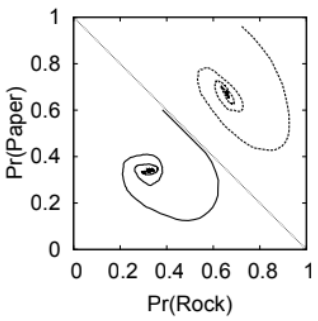
GIGA-WoLF

GIGA-WoLF es un algoritmo basado en el aprendizaje mediante el ascenso del gradiente. Este algoritmo tiene el objetivo de combinar la convergencia, que ya proporciona el algoritmo WoLF-PHC, con la propiedad de "no arrepentimiento". El arrepentimiento se define como la diferencia de rendimiento entre nuestra política actual y la mejor estrategia estática. Esta propiedad es deseable para evitar caer en trampas de algoritmos específicamente diseñados para ser engañosos.

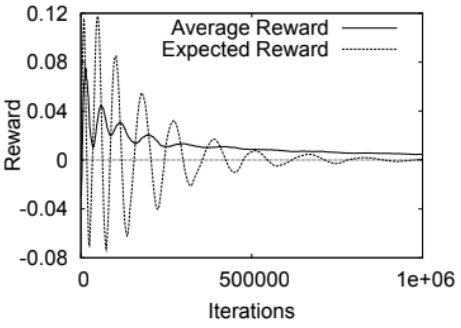
Internamente, el algoritmo almacena dos estrategias ("x" y "z") que son modificadas en cada iteración usando el ascenso de gradiente del algoritmo GIGA (Generalized Infinitesimal Gradient Ascent). Ambas son actualizadas en la misma dirección, pero "z" tiene un ratio de aprendizaje menor. Después, "x" será modificada hacia "z". Siempre modificaremos nuestra estrategia hacia la dirección del ascenso del gradiente, pero el tamaño del cambio dependerá de cuál de las dos estrategias es más favorable. Si lo definimos en los términos del principio WoLF, "x" recibirá una mayor modificación si pierde contra "z", y menor si gana. De esta manera, el algoritmo GIGA-WoLF sustituye el concepto de victoria definido con respecto a una estrategia de equilibrio. Este cambio reduce los requerimientos estrictos sobre el conocimiento de la matriz de recompensas y la distribución de las acciones de los oponentes que demandaba el algoritmo WoLF-PHC sin perder la propiedad de convergencia y garantizando el no-arrepentimiento. [2]



GIGA



GIGA-WoLF



GIGA vs GIGA-WoLF

Deep Learning

Deep Reinforcement Learning

A pesar de que los métodos de aprendizaje reforzado como Q-Learning tuvieron gran éxito, tienen muchas limitaciones. Entre ellas están que los métodos no se generalizan en el espacio de estados, las representaciones de estados tiene que ser especificados a mano y aprender en un espacio de estados grande puede ser muy lento.

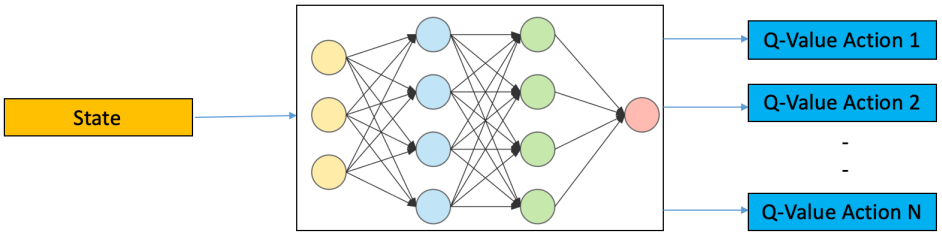
Con la idea de solucionar estas dificultades, se introdujo deep learning al aprendizaje reforzado ya que deep learning ayuda a generalizar los métodos entre estados y reduce la necesidad de representar los estados manualmente.

Sin embargo, extendiendo deep learning al aprendizaje reforzado trae sus propios problemas. Uno de ellos es que muchos métodos de aprendizaje supervisado asumen que los datos de entrenamiento son independientemente e idénticamente distribuidos. En cambio, en RL (aprendizaje reforzado), los datos de entrenamiento consisten en interacciones secuenciales altamente correlacionadas entre agente y entorno que rompe la condición de independencia. Además, la distribución de datos en RL no es estacionaria ya que el agente aprende de forma activa mientras explora el espacio de los estados. Esto rompe con la condición de datos idénticamente distribuidos. [3]

A continuación, introduciremos Deep Q-Networks y Federated deep Reinforcement Learning que intentan afrontar estas dificultades.

Deep Q-Networks

Deep Q-Networks (DQN) es una estructura enfocada para agentes únicos que combina Deep Learning y RL para enfrentarse a entornos de alta dimensionalidad. Esta estructura aprovecha CNN para interpretar la representación gráfica del estado de entrada del entorno. DQN produce los Q-values de todas las posibles acciones tomadas desde el estado de entrada. Por tanto, DQN se puede interpretar como una red de políticas con entrenamiento continuo con el fin de aproximar una política óptima. Cuando fue anunciado en 2015, se creó un agente autónomo que usando DQN consiguió jugar de forma competente a 49 juegos de Atari.



DQN solucionó la maldición de la dimensionalidad en RL, dando el primer paso en resolver aplicaciones en el mundo real. A continuación, detallaremos las variantes de DQN que han sido propuestas en los últimos años.

La primera variante es Double Deep Q-Networks (DDQN) y también la más simple. DDQN se basa en la idea de separar la selección de acciones codiciosas de la evaluación de las mismas reduciendo la sobrestimación de los Q-values. En segundo lugar, se introdujo la repetición prioritaria basada en la experiencia con el objetivo de romper correlaciones entre muestras. Esto lo consigue recordando muestras poco comunes que pueden ser olvidadas por la red de políticas ya que es más interesante tener más muestras poco comunes que redundantes. La combinación de DDQN con la repetición prioritaria basada en la experiencia tiene un rendimiento 5 veces mayor a DQN en cuanto a la puntuación media normalizada de 57 juegos de Atari.

Otro inconveniente de DQN es que utiliza una memoria de 4 frames como entrada a la red de políticas. Por tanto, no es eficiente para resolver problemas donde el estado actual depende de una gran cantidad de frames de memoria como ocurre con Double Dunk y Frostbite, juegos conocidos como problemas parcialmente observables de MDP. La solución es sustituir la capa totalmente conectada justo después de la última capa de convolución de la red política por una larga memoria recurrente a corto plazo. Esta variante de DQN, denominada Deep Recurrent Q-Network (DRQN), supera a la DQN estándar hasta en un 700% en los juegos Double Dunk y Frostbite. Además, se creó con éxito un agente que supera a un jugador medio en Doom, un entorno FPS (shooter en primera persona) en 3D, añadiendo una capa de características del juego en DRQN.

Otra variante interesante de DRQN es Deep Attention Recurrent Q-Network (DARQN) donde se añade un mecanismo de atención a DRQN para que la red pueda centrarse sólo en las regiones importantes del juego, lo que permite reducir los parámetros de la red y, por tanto, acelerar el proceso de entrenamiento. Como resultado, DARQN alcanza una puntuación de 7263 en comparación con los 1284 y 1421 de DQN y DRQN en el juego Seaquest, respectivamente. [4]

Federated deep Reinforcement Learning (FedRL)

El uso de aprendizaje en sistemas multiagente en aplicaciones reales se ha visto dificultado por una limitación de datos de entrenamiento, puesto que los conjuntos de datos de clientes suelen ser privados y también es difícil que un centro de datos pueda garantizar la construcción de modelos de alta calidad. Además si los datos que utiliza cada agente son privados, no pueden ser compartidos con otros agentes.

Para combatir con estas dificultades, se propuso Federated deep Reinforcement Learning (FedRL) cuyo objetivo es aprender un Q-network privado para cada agente compartiendo información limitada entre agentes. Dicha información es cifrada, mediante diferenciales gaussianas, y luego enviada a otros agentes donde se descifra. Se asume que existen agentes que tienen recompensas asociadas a estados y acciones. Adicionalmente, existen agentes que solo han observado estados sin recompensa, por tanto, estos agentes no pueden construir buenas políticas de decisión con su propia información, es decir, necesitan que los otros agentes compartan información con ellos. La idea es que todos los agentes se beneficien al unirse a la federación para construir políticas de decisión.

FedRL permite el uso de aprendizaje en MAS en situaciones donde antes no era posible como, por ejemplo, en el mundo de la sanidad. Al construir políticas de tratamiento médico a

los pacientes, es posible que un paciente reciba un tratamiento por parte del hospital pero no reciben feedback de dicho paciente. Esto significa que los hospitales no pueden recaudar recompensas sobre los tratamientos y, como consiguiente, no pueden construir buenas políticas de decisión. Además, como los datos de los pacientes son privados, la única forma de aprender políticas de tratamiento es mediante una federación. [5]

Aplicaciones

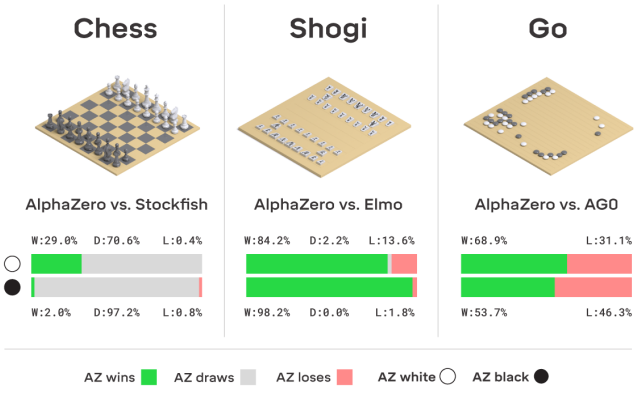
Juegos Estáticos

Ajedrez y Go

El juego del ajedrez es el ámbito más estudiado en la historia de la inteligencia artificial. Como consecuencia se han desarrollado programas de ajedrez que juegan a nivel sobrehumano. Sin embargo, estos programas están altamente ajustados a su dominio, y no pueden generalizarse a otros juegos como Go o shogi sin un notable esfuerzo humano.

En 2017 DeepMind introdujo AlphaZero, una versión más genérica de AlphaGo, como un sistema que mediante auto-playing dominaba juegos como ajedrez, Go y shogi. A diferencia de los programas de ajedrez que dependen de miles de reglas y heurísticas, AlphaZero toma un enfoque distinto cambiando las reglas por una deep neural network y algoritmos que solo conocen las reglas básicas del juego.
AlphaZero fue entrenado mediante RL, jugando millones de partidas contra sí mismo. Al principio, jugaba de forma aleatoria pero con cada partida iba ajustando los parámetros de la red neuronal para asegurar que escogiese acciones ventajosas en el futuro.

Con el sistema totalmente entrenado se puso a prueba contra los programas más fuertes de ajedrez (Stockfish) y shogi (Elmo). Además se comparó con AlphaGo Zero, el mejor jugador conocido de Go. Después de mil partidas, ganó 155 y perdió tan solo 5 contra Stockfish, venció en 91,2% de las partidas que jugó contra Elmo en shogi y, finalmente, en Go ganó en 61% de la partidas que jugó contra AlphaGo Zero.



La capacidad que tiene AlphaZero para dominar tres juegos complejos distintos y, potencialmente, cualquier juego de información perfecta demuestra que un solo algoritmo puede aprender cómo descubrir nuevos conocimientos en diversos ámbitos. DeepMind cree que esto es un paso importante para llegar a crear sistemas de aprendizaje de propósito

general que algún día ayuden a solucionar los problemas más importantes y complejos de la ciencia. [6]

Juegos Dinámicos

Starcraft

StarCraft II, creado por Blizzard Entertainment, está ambientado en un universo de ciencia ficción y cuenta con una jugabilidad rica de múltiples capas diseñada para desafiar el intelecto humano. Junto con el título original, Starcraft, se encuentra entre los juegos más grandes y exitosos de la historia, con jugadores compitiendo en torneos de esports durante más de 20 años.

Hay distintas formas de jugar, pero el formato más común es el torneo de uno contra uno jugado a cinco partidas. Para empezar, un jugador tiene que elegir entre 3 razas alienígenas diferentes (Zerg, Protoss o Terran), cada uno con características y habilidades distintivas. Cada jugador comienza con un número de unidades de trabajadores, que recogen recursos básicos para construir más unidades y estructuras y para crear nuevas tecnologías. Éstas, a su vez, permiten al jugador cosechar otros recursos, construir bases y estructuras más sofisticadas y desarrollar nuevas capacidades que pueden utilizarse para superar al oponente. Para ganar, el jugador debe equilibrar cuidadosamente la gestión a gran escala de su economía, conocida como macro, junto con el control a bajo nivel de sus unidades individuales, conocido como micro.

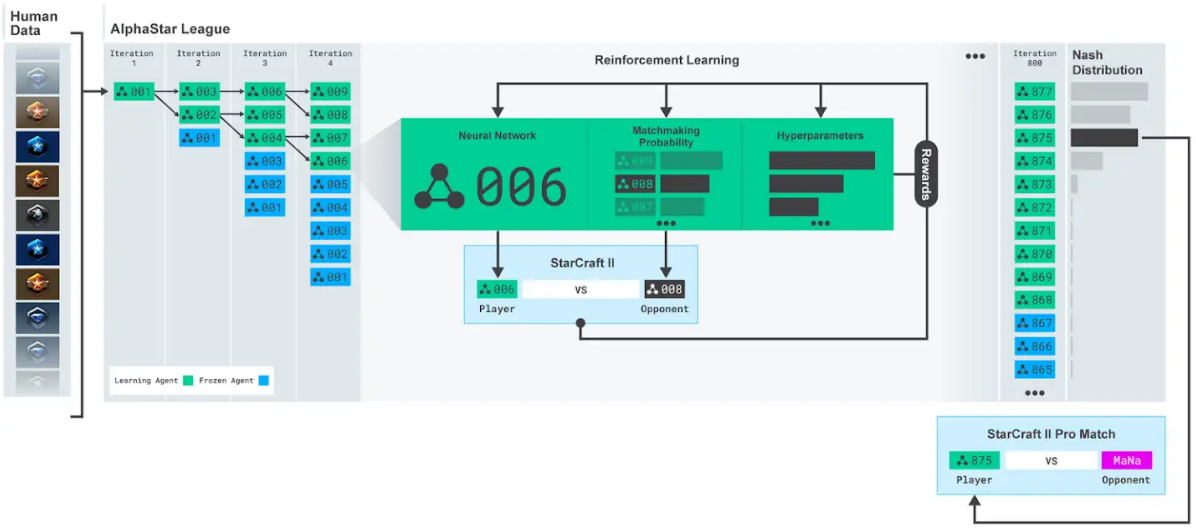
La necesidad de balancear entre metas a corto y largo plazo y adaptarse a situaciones inesperadas, posee un gran reto. Además, existen otros retos pertenecientes a la inteligencia artificial como:

- Teoría de juegos: StarCraft es un juego en el que no hay una sola estrategia óptima, es decir, en el proceso de entrenamiento será necesario explorar continuamente nuevas estrategias.
- Información imperfecta: Información esencial es escondida y solo puede ser descubierta mediante la exploración.
- Planificación a largo plazo: Dado que ciertas partidas pueden durar hasta una hora, las jugadas empleadas al principio pueden tardar en dar los resultados esperados.
- Tiempo real: A diferencia del Go o Ajedrez alternan turnos, en StarCraft las acciones se emplean continuamente a medida que la partida progresa.
- Gran espacio de acciones: En cada paso del tiempo se pueden realizar hasta 26 acciones resultando en un espacio combinatorio de posibilidades.

AlphaStar es una inteligencia artificial que mediante deep neural networks, aprendizaje supervisado y multiagente RL. Es la primera AI que consiguió derrotar a un jugador profesional StarCraft II.

El comportamiento de AlphaStar es generado por una deep neural network que recibe como entrada la interfaz del juego. Inicialmente, esta red neuronal se entrenaba mediante aprendizaje supervisado de partidas jugadas por humanos. Esto le permitió a AlphaStar aprender, por medio de la imitación, las micro y macro estrategias básicas. Con tan solo esto, AlphaStar derrotaba al 95% de sus oponentes humanos de nivel oro.

Con el fin de mejorar AlphaStar, se implementó un proceso de RL multiagente que consiste en crear una liga de agentes donde cada agente juega partidas contra otros. A medida que progresa la liga, los agentes aprenden de las partidas y descubren nuevas y mejores estrategias. Esta liga se puede considerar un método de auto-playing que no se olvida de estrategias pasadas. La liga AlphaStar fue ejecutado durante 14 días en el que cada agente jugó hasta 200 años en partidas en tiempo real. El agente final consiste en la combinación más efectiva de las estrategias descubiertas.



Tras la implementación de la liga multiagente, AlphaStar se puso a prueba contra dos de los mejores profesionales de StarCraft II. A pesar de que los profesionales eran capaces de realizar casi el triple de acciones por minuto más que AlphaStar, la precisión y optimización de las acciones de la AI eran mucho más superior a aquella de los profesionales. Además, eso se demostró en resultado con AlphaStar ganando las dos partidas 5 a 0. Ambos profesionales comentaron que estaban sorprendidos por la ausencia de errores en las estrategias de AlphaStar y por el uso de tácticas desconocidas por ellos. Eso significa que AlphaStar ha descubierto estrategias que la enorme comunidad de jugadores no había hecho aún en 20 años.

Alcanzar los niveles más altos en StarCraft representa un gran avance en uno de los videojuegos más complejos jamás creados. Se espera que el progreso de la arquitectura de las redes neuronales y los métodos de entrenamiento sean un gran paso hacia diseñar sistemas inteligentes que un día nos ayuden a resolver algunos de los problemas científicos más importantes y fundamentales. [7]

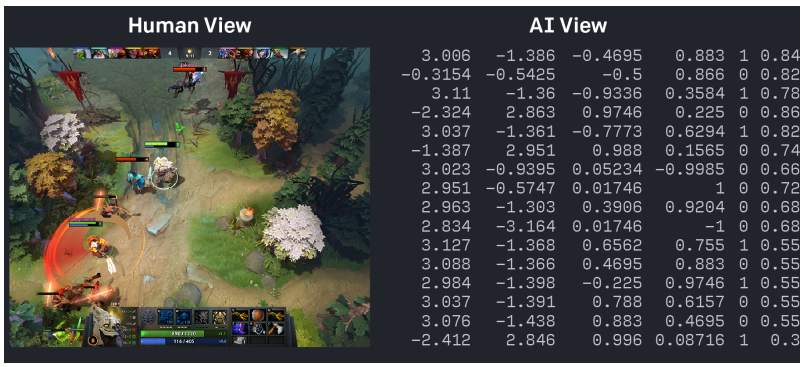
Dota 2

Dota 2 es un videojuego del género MOBA (Multiplayer Online Battle Arena) lanzado el 9 de Julio de 2013 por la empresa estadounidense Valve Corporation. Actualmente Dota 2 es uno de los videojuegos más jugados, con más de 400000 jugadores concurrentes de media en el último mes y un pico de jugadores concurrentes de casi 1.3 millones. Es uno de los videojuegos más importantes dentro de los esports, The International 2019 el torneo más importante de Dota 2 hasta la fecha, tuvo casi 2 millones de espectadores concurrentes de pico y un total de más de 34 millones de dólares en premios. [8] [9]

Dota 2 es un videojuego en el que dos equipos, cada uno formado por 5 jugadores, se enfrentan con el objetivo de destruir la base enemiga para ganar la partida. Todas las partidas se juegan en el mismo mapa pero cada jugador puede elegir un héroe distinto en cada partida. El mapa se divide en dos por el río y está compuesto por tres líneas, top, mid y bot, separadas por la jungla. Cada jugador se sitúa en una posición y lleva a cabo una función para su equipo según el personaje que haya elegido.

En 2016, OpenAI, una compañía de investigación de inteligencia artificial sin ánimo de lucro, comenzó un proyecto llamado OpenAI Five, con el objetivo de desarrollar un equipo de bots capaces de dominar el Dota 2. Para ello, en su modelo, se plantea una partida del videojuego como un entorno multiagente donde cada héroe, normalmente controlado por un jugador, representa un agente. Los agentes de un mismo equipo deben cooperar para ganar la partida y enfrentarse a los agentes enemigos, que también trabajan juntos para vencer.

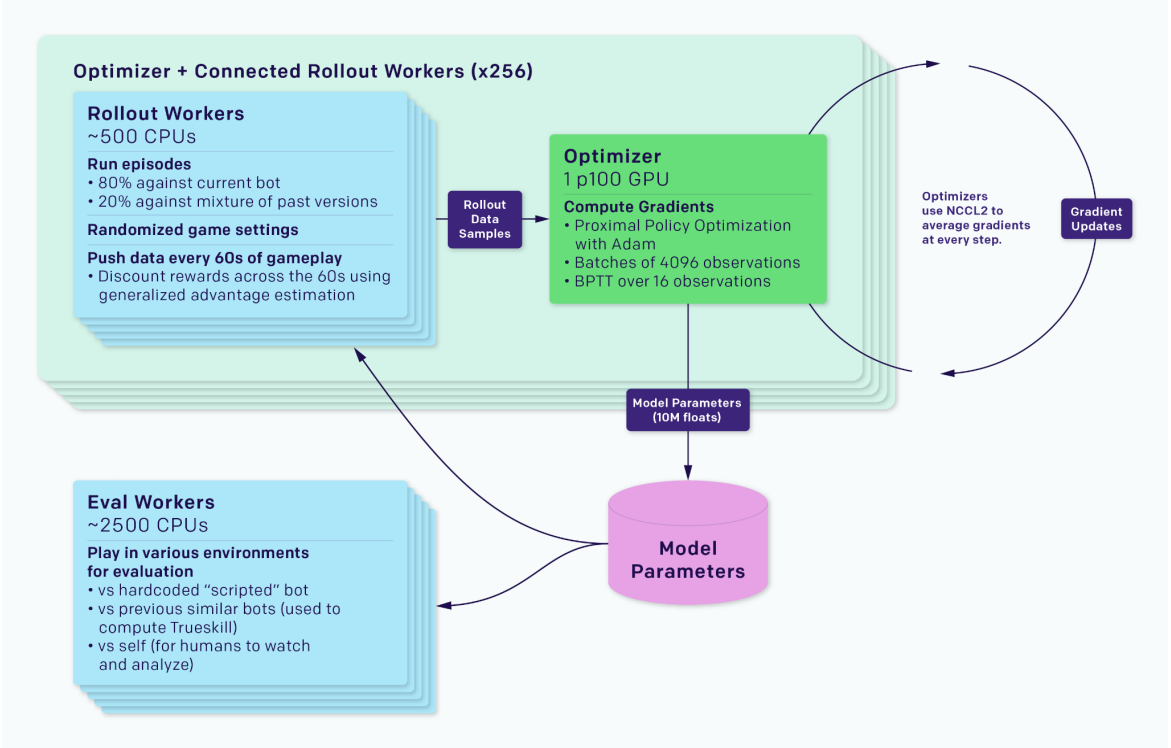
OpenAI identificó una serie de desafíos en Dota 2 que su modelo debía superar para ser capaz de dominar el juego. En primer lugar, el número de acciones por partida. Dota 2 se ejecuta a 30 fotogramas por segundo, y sus partidas duran una media de 45 minutos, resultando en 80.000 fotogramas por partida en las que se puede actuar. En comparación, el ajedrez termina antes de 40 movimientos y el Go en menos de 150. Además, cada héroe tiene alrededor de 1000 acciones posibles en cada uno de estos fotogramas (de un conjunto de 170.000 acciones totales) al mismo tiempo que ha de observar al resto de héroes (agentes), decenas de unidades, estructuras y características del juego, todo representado por unos 20.000 números. Por otra parte, Dota 2 es un juego de información parcial, en la que cada equipo solo puede ver en un área alrededor de sus unidades, el resto del mapa está dentro de la “niebla de guerra”, por lo que el agente debe ser capaz de inferir el estado de la partida a partir de información incompleta. [10]



Visión Humana *Visión del Agente*

OpenAI Five fue entrenado únicamente mediante autoaprendizaje (partidas contra sí mismo) y sin observación de partidas humanas. Este aprendizaje utiliza un algoritmo de Reinforced Learning de la clase de algoritmos conocida como Proximal Policy Optimization (PPO). Este tipo de algoritmos son una subclase de los Policy Gradient Methods, basados en seleccionar la política que maximiza las recompensas futuras de nuestras acciones. Los algoritmos PPO adoptan el concepto de la región de confianza de los algoritmos TRPO (Trust Region Policy Optimization) para mejorar su funcionamiento, asegurando que la nueva política no sale de la región de confianza de la política anterior, es decir, no se

modifica demasiado. Mediante PPO, OpenAI Five recibió 180 años de entrenamiento diarios para cada héroe. [10] [11]



Arquitectura del modelo

En 2017, OpenAI desarrolló un bot capaz de derrotar a los mejores jugadores profesionales en partidas de 1 contra 1. El agente aprendió habilidades como predecir dónde se moverán sus oponentes, esquivar habilidades, improvisar en respuesta a situaciones nuevas y distintas interacciones con los elementos aliados y enemigos de la partida. Desde 2018, OpenAI empezó a entrenar a los agentes para partidas de 5 contra 5 contra humanos. En cada experimento de OpenAI Five se enfrentaba contra mejores equipos y reducía las restricciones impuestas sobre las variables involucradas (número de héroes, objetos, habilidades…). En Abril de 2019, tras numerosas iteraciones del modelo, OpenAI Five fue capaz de derrotar 2-0 al campeón del mundo. Este mismo modelo estuvo disponible temporalmente en internet jugando más de 7000 partidas con un porcentaje de victorias superior al 99%. Finalmente, OpenAI creó un último agente llamado Rerun. Este agente fue desarrollado en 2 meses invirtiendo para entrenarlo únicamente el 20% de los recursos usados en el modelo previo y ganó el 98% de las partidas en las que se enfrentó contra su predecesor. [10] [12] [13]

Es importante destacar que OpenAI Five tiene un tiempo de reacción de 80 ms, superior a los humanos (alrededor de 200 ms de media, menor en los jugadores profesionales) pero un número de acciones por minuto (120 APM) menor que el de un jugador de Dota 2 de alto nivel (entre 200 y 300 acciones por minuto). [10]

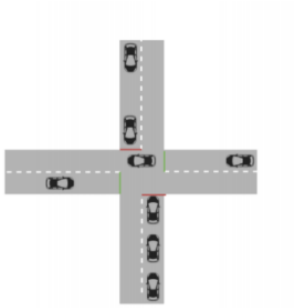
En la práctica

Control de semáforos

La mayor parte del tráfico en las ciudades se debe al funcionamiento ineficiente de los semáforos. Durante años se ha intentado mejorar el comportamiento de estos mediante el uso de Inteligencia Artificial, obteniendo buenos resultados a pequeña escala pero con problemas de escalabilidad debido a la maldición de la dimensionalidad. Como hemos visto en el apartado de Deep Learning, podemos evitar este obstáculo mediante el uso de Deep Reinforcement Learning.

Jeancarlo Arguello Calvo e Ivana Dusparic plantearon un modelo para entrenar a varios agentes heterogéneos para el control de los semáforos mediante el uso de Independent Q-Learning (IQL). Para poder entrenar a los agentes de forma simultánea, se usa Dueling Double Deep Q-Networks (DDDQN) en combinación con un algoritmo de huella digital (para compatibilizar Prioritized Experience Replay con IQL).

El modelo representa el estado de un cruce usando dos matrices de 64x64, una para la posición de los vehículos y otra para la velocidad. El espacio de acciones es discreto y depende del número de carreteras que haya en la intersección. En general, cada calle tendrá una acción para poner el semáforo en verde y otra para ponerlo en rojo (con un periodo intermedio en ámbar). La función de recompensas toma valores entre 0 y 1 según el tiempo de espera acumulado de los vehículos en la intersección mediante la fórmula:
$W_t = \Sigma w_{i,t}$ dónde $w_{i,t}$ es el tiempo de espera del vehículo i en el tiempo t. [14]



Estado del cruce

$$\begin{bmatrix} 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0 \end{bmatrix}$$

Matriz de posiciones

$$\begin{bmatrix} 0.0&0.0&0.0&0.5&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.4&0.0&0.0&0.0&1.0 \\ 0.0&0.8&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \end{bmatrix}$$

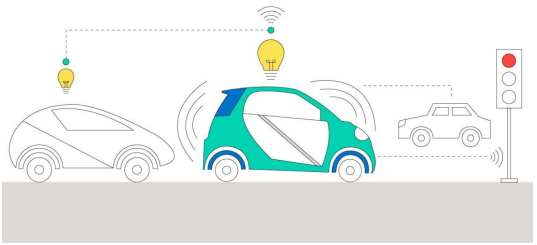
Matriz de velocidades

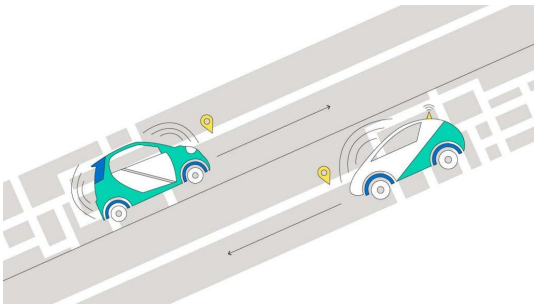
Conducción Autónoma

La conducción autónoma puede ser modelada como un entorno multiagente en el que el vehículo anfitrión debe aplicar sofisticadas habilidades de negociación con otros usuarios de la carretera al adelantar, ceder el paso, girar a la izquierda y a la derecha y al avanzar en vías urbanas no estructuradas. Dado que hay muchos escenarios posibles, abordando manualmente todos los casos obtendremos una política demasiado simplista. Además, hay que equilibrar entre el comportamiento inesperado de otros conductores o peatones y, al mismo tiempo, no ser demasiado defensivo para que se mantenga el flujo normal del tráfico. La capacidad de aprender y adaptarse a los cambios en el entorno de conducción es crucial para desarrollar sistemas de conducción autónoma que sean escalables más allá de los

dominios de diseño operacional delimitados. Deep Reinforcement Learning proporciona un marco prometedor y escalable para desarrollar soluciones basadas en el aprendizaje adaptativo. [15] [16]

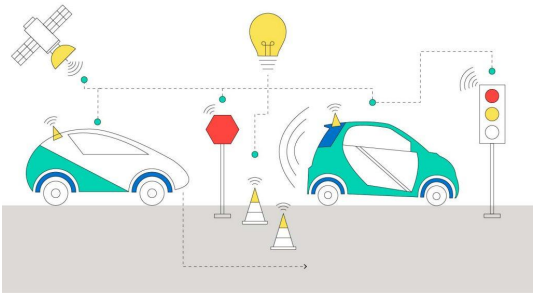
Diferentes compañías automovilísticas están enfocando el problema de la conducción autónoma de distintas maneras. Tesla sigue un modelo de conocimiento en tiempo real mediante análisis y reconocimiento de sus alrededores. Cada vehículo aprende de forma autónoma y comparte su conocimiento con el resto. Este modelo desprecia los mapas y otros conocimientos previos lo que lo hace más complejo pero adaptable al entorno y sus cambios.





Por otro lado, General Motors, Mercedes Benz y Ford apuestan por el uso del Lidar. Los vehículos dependen de mapas 3D de alta resolución captados previamente por vehículos no autónomos equipados con un Lidar. Después, los vehículos usan su propio Lidar para determinar si su entorno se ha modificado con respecto al mapa y actuar en consecuencia. Este enfoque es muy predecible y confiable pero tiene un coste elevado.

Por su parte, Volkswagen está desarrollando un modelo en el que el entorno es inteligente, de manera que, en vez de que el vehículo se adapte a su entorno es el entorno el que le comunica al vehículo los cambios. Este modelo también implementa comunicación entre vehículos (V2V) para comunicar información sobre accidentes o objetos en la carretera. Este enfoque requiere una gran inversión para construir carreteras inteligentes pero puede llegar a ser el más seguro. [17]



ZeroEnergy Communities

Debido al preocupante avance de la crisis climática, se le ha dado importancia a los ZEBs (Zero Energy Buildings). Los ZEBs son edificios cuya energía neta es 0, es decir, edificios que producen la misma energía de la que consumen. Sin embargo, es muy difícil conseguir que la energía neta sea exactamente cero, por eso se introdujeron los nZEBs (nearly Zero Energy Buildings). Durante un periodo de tiempo la energía neta de los nZEBs debe ser cero o positiva. También se ha introducido la idea de las nZECs (nearly Zero Energy Communities) definido como un conjunto de edificios cuya energía neta total es casi 0, es decir, que la suma de la energía neta de todos los edificios de la comunidad es casi 0. A

continuación estudiaremos la propuesta de Amit Prasad e Ivana Dusparic sobre como modelar un nZEC.

La propuesta se basa en modelar una nZEC como un entorno multiagente donde cada edificio representa un agente. Cada agente aprende una política de actuación óptima de forma independiente y se encarga de realizar transacciones de energía. Para poder llevar a cabo esta propuesta se han identificado dos componentes principales: un agente DRL, que aprende el comportamiento de un edificio, y un CMS (Community Service Management) para habilitar la colaboración entre agentes.

Agente DRL

Para conseguir que el agente DRL aprenda el comportamiento de un edificio se ha empleado el algoritmo DQN explicado previamente. El proceso de aprendizaje de un agente DRL se resume en que un agente extrae datos del entorno como la consumición y generación de energía, traduce los datos a un vector de estados. A continuación, este vector es procesado y se elige la acción percibida como la más correcta o adecuada.
El conjunto de acciones está formado por:

- Consumir y almacenar el exceso de energía
- Solicitar al vecino energía adicional
- Solicitar a la red de alimentación energía adicional
- Aceptar la solicitud de energía de un vecino
- Rechazar la solicitud de energía de un vecino

Dependiendo del estado en el que se encuentra el agente y la acción que realiza, recibirá una recompensa u otra.

Community Monitoring Service

El CMS actúa como un servicio de gestión de grupos de agentes que se encarga de qué hacer cuando un agente entra o sale del grupo y de mantener una lista de los agentes activos. Además, el CMS recopila los estados de energía de cada agente a intervalos regulares y calcula el estado de energía de la comunidad. Los otros agentes pueden acceder a esta información para calcular sus propias recompensas basadas en la acción tomada.

Durante la evaluación de la propuesta de Prasad y Dusparic se ha observado que con el uso de sistemas multiagente una comunidad de 3 edificios en invierno mejoró su energía neta 40 kWh comparado con una comunidad que no emplea una estrategia de compartir energía. Además, en verano el nZEC con 4 edificios detectó que su energía neta mejoró en 60 kWh comparado con la otra comunidad. Sin embargo, este modelo se basa en que los agentes estén motivados a compartir energía sin un régimen de precios y esto en sistemas del mundo real no es factible. [18]

Conclusión

El aprendizaje en sistemas multiagente es un campo de investigación muy interesante con muchas aplicaciones en el mundo real. La posibilidad de modelar problemas mediante agentes que interactúan entre ellos y con el entorno resulta una herramienta muy potente a la vez que intuitiva.

A lo largo de los años se han ido haciendo avances en la materia pero ninguno tan importante como la implementación del Deep Reinforcement Learning. DRL permite afrontar problemas mucho más complejos y sofisticados, imposibles de resolver con otros algoritmos. Además, mediante la aplicación de técnicas de machine learning, podemos condensar la dimensionalidad de los datos sin perder información, evitando así la maldición de la dimensionalidad, que hace que la cantidad de datos requerida aumente exponencialmente con el número de dimensiones de los datos.

Los juegos siempre han sido un indicador de referencia en la historia de la inteligencia artificial. Los juegos se plantean como un problema que tratar de resolver con todos los medios y herramientas disponibles y suponen un desafío que lleva a mejorar las técnicas existentes y desarrollar otras nuevas para tratar de superar los retos impuestos por el juego. La misma tendencia se ha seguido en el aprendizaje en MAS, primero con juegos muy simples como el 3 en raya, después en juegos más complejos como ajedrez y go y en los últimos años se han desarrollado agentes capaces de vencer a los mejores jugadores de videojuegos mundialmente conocidos como Starcraft II y Dota 2. Todas las herramientas y algoritmos desarrollados durante el estudio de los juegos son trasladadas a problemas del mundo real.

Los sistemas multiagente se han aplicado a diferentes ámbitos a lo largo de los años. Entre estas aplicaciones encontramos algunas de las que hemos tratado como: el transporte (conducción autónoma, control del tráfico mediante semáforos…), la logística y gestión de recursos tanto del día a día (electricidad, agua…) como en situaciones extremas (catástrofes, pandemias...), la comunicación (sistemas de comunicación distribuidos, tecnologías móviles, comercio electrónico...), la simulación (simulación de sociedades, mercado bursátil...) y los videojuegos.

En general, podemos concluir que el aprendizaje en sistemas multiagente es un ámbito de la inteligencia artificial muy importante debido a la utilidad y las aplicaciones de los sistemas multiagente. El estudio y la mejora de los algoritmos de aprendizaje es clave para el desarrollo de sistemas capaces de resolver problemas más complejos y de mayor relevancia.

Bibliografía

- Michael Bowling and Manuela Veloso, Multiagent learning using a variable learning rate, 2001 [1]
- Michael Bowling, Convergence and No-Regret in Multiagent Learning, 2004 [2]
- Pablo Hernandez-Leal, Bilal Kartal and Matthew E. Taylor, A Survey and Critique of Multiagent Deep Reinforcement Learning, 2019 [3]
- Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi, Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications, 2019 [4]
- Hankz Hankui Zhuo, Wenfeng Feng, Yufeng Lin, Qian Xu and Qiang Yang, Federated Deep Reinforcement Learning, 2020 [5]
- Deep Mind, A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, 2018 [6]
- Deep Mind, AlphaStar: Mastering the Real-Time Strategy Game StarCraft II (https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii), 2019 [7]
- Steamcharts (https://steamcharts.com/app/570) [8]
- Esports Charts (https://escharts.com/tournaments/dota2) [9]
- OpenAI, OpenAI Five (https://openai.com/blog/openai-five/), 2018 [10]
- OpenAI, Proximal Policy Optimization Algorithms (https://openai.com/blog/openai-baselines-ppo), 2017 [11]
- OpenAI, More on Dota 2 (https://openai.com/blog/more-on-dota-2/), 2017 [12]
- OpenAI, OpenAI Five Defeats Dota 2 World Champions (https://openai.com/blog/openai-five-defeats-dota-2-world-champions/), 2019 [13]
- Jeancarlo Arguello Calvo and Ivana Dusparic, Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control, 2018 [14]
- Praveen Palanisamy, Multi-Agent Connected Autonomous Driving using Deep Reinforcement Learning, 2019 [15]
- Shai Shalev-Shwartz, Shaked Shammah and Amnon Shashua, Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving, 2016 [16]
- Katie Luke, Three Approaches to Solving the Autonomous Vehicle Orientation Problem (https://www.connected.io/post/three-approaches-to-solving-the-autonomous-vehicle-orientation-problem), 2019 [17]
- Amit Prasad and Ivana Dusparic, Multi-agent Deep Reinforcement Learning for Zero Energy Communities, 2019 [18]