Ingeniería Informática y Administración y Dirección de empresas

2022-2023

Trabajo Fin de Grado

“Aplicación de técnicas de aprendizaje por refuerzo en el juego del Poker Texas Holdem”

Pablo Lago Sánchez-Martínez

NIA 100386262

Tutor/es

Miguel Angel Patricio Guisado

Colmenarejo 2023

Índice de contenidos

I. Introducción

A. Contexto y motivación

El póker Texas Hold'em es reconocido mundialmente como uno de los juegos de cartas más desafiantes y populares. A lo largo del tiempo, ha sido objeto de estudio y análisis por parte de jugadores profesionales, matemáticos y científicos de la computación, con el objetivo de comprender y mejorar las estrategias implicadas en el juego. Con el advenimiento de la inteligencia artificial y las técnicas de aprendizaje por refuerzo, se ha abierto una nueva puerta para explorar y perfeccionar aún más este apasionante juego.

La aplicación de técnicas de aprendizaje por refuerzo en el póker Texas Hold'em cuenta con fundamentos sólidos. En primer lugar, el Texas Hold'em es un juego complejo que requiere tomar decisiones estratégicas basadas en información imperfecta. Esto crea un desafío interesante para los algoritmos de inteligencia artificial, los cuales deben aprender a tomar decisiones óptimas en un entorno incierto y dinámico.

En segundo lugar, el póker Texas Hold'em ha experimentado un crecimiento exponencial en popularidad en los últimos años, tanto entre jugadores recreativos como profesionales. Esto ha llevado a un aumento en la cantidad de datos disponibles para su análisis, brindando una excelente oportunidad para aplicar técnicas de aprendizaje automático y descubrir patrones significativos que ayuden a mejorar el juego.

Además, el uso de inteligencia artificial y aprendizaje por refuerzo puede proporcionar una ventaja competitiva tanto a los jugadores profesionales como a los apasionados del póker.

Por último, como alguien apasionado por el póker, la aplicación de técnicas de aprendizaje por refuerzo en este juego permitiría explorar y comprender aún más las complejidades estratégicas que implica. Esto brindaría la oportunidad de analizar y mejorar su propio juego, así como de contribuir al campo de la inteligencia artificial y el aprendizaje automático mediante la creación de algoritmos y modelos capaces de superar a los jugadores humanos en el póker Texas Hold'em.

II. Objetivos y alcance

A. Objetivos

- Desarrollar un agente de inteligencia artificial utilizando RLCard y aprendizaje por refuerzo con el fin de mejorar su capacidad para jugar Texas Hold'em No Limit de manera efectiva.

- Mejorar las habilidades de toma de decisiones del agente en situaciones complejas y dinámicas inherentes al juego de póker Texas Hold'em.

- Investigar, evaluar y comparar diferentes algoritmos y enfoques de aprendizaje por refuerzo aplicados al póker para determinar la estrategia óptima para el agente.

- Optimizar progresivamente la estrategia del agente mediante iteraciones de entrenamiento y retroalimentación continua, con el objetivo de lograr un rendimiento competitivo.

- Evaluar y comparar el desempeño del agente desarrollado con jugadores humanos profesionales y no profesionales.

B. Alcance

- Implementar y personalizar el entorno de Texas Hold'em No Limit utilizando la biblioteca RLCard, que proporciona una plataforma de desarrollo para la investigación en aprendizaje por refuerzo en juegos de cartas.

- Realizar un estudio exhaustivo de los algoritmos y enfoques más relevantes de aprendizaje por refuerzo aplicados al póker, considerando tanto métodos clásicos como enfoques más recientes.

- Seleccionar y adaptar el algoritmo de aprendizaje por refuerzo más adecuado para entrenar al agente de póker, teniendo en cuenta las características específicas del juego y los objetivos del proyecto.

- Diseñar e implementar el proceso de entrenamiento del agente, estableciendo la metodología y los parámetros adecuados, y ajustándose en función de los resultados obtenidos durante la fase de evaluación.

- Realizar pruebas y evaluaciones periódicas para medir el desempeño del agente durante el entrenamiento, analizando métricas relevantes como la tasa de éxito, la estabilidad de la estrategia y el tiempo requerido para la toma de decisiones.

- Comparar y analizar de forma cuantitativa y cualitativa los resultados obtenidos por el agente desarrollado con respecto a estrategias previas y jugadores humanos, utilizando métricas de rendimiento establecidas y casos de prueba representativos.

- Documentar y presentar de manera formal los hallazgos y conclusiones del proyecto, destacando las lecciones aprendidas y las posibles áreas de mejora o investigación futura.

- Proporcionar recomendaciones claras y fundamentadas para la aplicación práctica del agente de póker desarrollado, considerando su potencial uso en entrenamiento de jugadores, asesoramiento estratégico o desarrollo de sistemas de juego autónomos.

III. Estado del arte

A. Juegos de cartas y su representación computacional

Los juegos de cartas, como el póker, han sido populares durante siglos y han evolucionado tanto en su forma física como en su representación computacional. En el caso del póker, se ha convertido en uno de los juegos de cartas más conocidos y jugados en todo el mundo.

La representación computacional del póker ha permitido que el juego se vuelva accesible en línea, lo que ha llevado a un gran auge de las plataformas de póker en internet. Estas plataformas permiten a los jugadores de todo el mundo competir entre sí sin importar su ubicación geográfica.

La representación computacional del póker se basa en algoritmos y sistemas que imitan las reglas y mecánicas del juego. Esto implica la creación de una interfaz gráfica donde los jugadores pueden ver sus cartas, realizar apuestas y tomar decisiones estratégicas. Además, se implementan algoritmos que generan manos de cartas aleatorias y calculan las probabilidades de victoria de cada jugador.

La representación computacional del póker también ha permitido el desarrollo de programas de inteligencia artificial (IA) capaces de competir y vencer a jugadores humanos en partidas de póker. Estos programas utilizan algoritmos sofisticados, como el aprendizaje automático y el procesamiento del lenguaje natural, para analizar y tomar decisiones estratégicas basadas en la información disponible. Ejemplos famosos incluyen "Libratus" y "Pluribus", que han demostrado un rendimiento sobresaliente en competiciones contra jugadores profesionales.

Además del póker tradicional, existen muchas variantes de póker que también han sido representadas computacionalmente. Estas variantes incluyen Texas Hold'em, Omaha, Seven-Card Stud y muchas más. Cada una de ellas tiene reglas y estrategias únicas, y su representación en entornos computacionales permite a los jugadores experimentar diferentes formatos de póker y mejorar sus habilidades.

B. Aprendizaje automático y sus aplicaciones

El aprendizaje automático, también conocido como machine learning en inglés, es una rama de la inteligencia artificial que se centra en desarrollar algoritmos y técnicas que permiten a las máquinas aprender y mejorar su rendimiento a través de la experiencia, sin necesidad de ser programadas explícitamente. Estos algoritmos son capaces de analizar grandes cantidades de datos, identificar patrones y tomar decisiones o hacer predicciones basadas en esa información.

En lo que respecta a los juegos de azar, el aprendizaje automático ha encontrado diversas aplicaciones. En particular, en el poker Texas Hold'em sin límite (no limit hold'em), el aprendizaje automático ha sido utilizado para desarrollar sistemas capaces de tomar decisiones estratégicas durante el juego.

Una de las aplicaciones más comunes del aprendizaje automático en juegos de azar es la creación de agentes inteligentes capaces de jugar contra humanos o incluso contra otros agentes. Estos agentes utilizan algoritmos de aprendizaje automático para analizar información sobre el estado actual del juego, como las cartas en la mano, las cartas comunitarias en la mesa y las acciones de los jugadores, y tomar decisiones óptimas en función de esa información.

El aprendizaje automático también se utiliza para la detección de patrones y comportamientos sospechosos en los juegos de azar, especialmente en los casinos en línea. Los algoritmos de aprendizaje automático pueden analizar datos en tiempo real de los jugadores y detectar anomalías o comportamientos que puedan indicar trampas o actividades fraudulentas.

En el caso específico del poker Texas Hold'em sin límite, el aprendizaje automático ha sido aplicado para desarrollar sistemas capaces de jugar a un nivel muy alto. Estos sistemas pueden aprender de forma autónoma a través de la experiencia y mejorar su estrategia a medida que juegan más partidas. Algunos de estos sistemas han demostrado un rendimiento sobresaliente, incluso superando a jugadores humanos profesionales.

El éxito del aprendizaje automático en el poker Texas Hold'em sin límite se debe en parte a la naturaleza del juego, que implica información parcial y toma de decisiones basada en probabilidades. Los algoritmos de aprendizaje automático pueden aprender a evaluar las probabilidades de ganar en diferentes situaciones y utilizar esta información para tomar decisiones óptimas, como apostar, subir o retirarse.

En la actualidad ya existen algunos grandes agentes que ganan al poker Texas Hold'em No Limit sin embargo, no son agentes de código abierto por lo que no permiten su utilización de cara al público.

Uno de esos programas se llama **Libratus**:

Libratus, desarrollado por investigadores de la Universidad Carnegie Mellon, es un algoritmo basado en técnicas de aprendizaje automático que utiliza una estrategia computacionalmente intensiva para tomar decisiones óptimas en cada etapa del juego.
A diferencia de los enfoques tradicionales que se basaban en reglas fijas o heurísticas, Libratus aprende de manera autónoma a través de la experiencia y mejora su estrategia a medida que juega más partidas.

Uno de los aspectos destacados de Libratus es su capacidad para enfrentar la incertidumbre inherente al póker. El algoritmo no tiene información completa sobre las cartas de sus oponentes, lo que significa que debe evaluar las probabilidades y tomar decisiones basadas en una estimación de las manos de los demás jugadores. Utiliza técnicas sofisticadas de aprendizaje automático, como árboles de búsqueda Monte Carlo y algoritmos de optimización, para calcular las mejores jugadas en función de las probabilidades y maximizar sus ganancias a largo plazo.

El éxito de Libratus se demostró en el desafío "Brains vs. Artificial Intelligence" (Cerebros vs. Inteligencia Artificial) que tuvo lugar en 2017. En este evento, el algoritmo se enfrentó a cuatro de los mejores jugadores de póker sin límite Texas Hold'em del mundo durante un período de 20 días y 120,000 manos. Al finalizar el desafío, Libratus logró una victoria aplastante, superando a los jugadores humanos con una ventaja de más de $1.7 millones de dólares en fichas virtuales.

La fortaleza de Libratus radica en su capacidad para adaptarse y ajustar su estrategia en tiempo real. El algoritmo no solo es capaz de analizar las jugadas de sus oponentes y ajustar su enfoque, sino que también puede detectar patrones de juego y explotar las debilidades de sus adversarios. Además, Libratus tiene la capacidad de mezclar su estilo de juego y sorprender a sus oponentes, lo que dificulta su lectura y toma de decisiones.

Libratus representa un hito importante en el desarrollo de algoritmos de aprendizaje automático aplicados al póker. Su capacidad para superar a los mejores jugadores humanos en una de las variantes más complejas y desafiantes del juego demuestra el potencial del aprendizaje automático en la toma de decisiones estratégicas bajo condiciones de incertidumbre.

Aunque Libratus es un ejemplo impresionante de la capacidad de las máquinas para dominar el póker, también plantea preguntas fascinantes sobre el futuro de este juego y la interacción entre humanos y algoritmos. A medida que los avances en el aprendizaje automático continúan, es posible que veamos nuevas generaciones de algoritmos aún más sofisticados que desafíen aún más la supremacía humana en el póker y otros juegos estratégicos.

Otro de esos programas se llama **Pluribus**:

Pluribus es un programa de inteligencia artificial desarrollado por Facebook y la Universidad Carnegie Mellon. Es un bot de poker que ha ganado reconocimiento por su habilidad para jugar al poker de Texas Hold'em de seis jugadores.

En 2019, Pluribus hizo historia al derrotar a varios jugadores de poker profesionales en un desafío de 12 días de duración, conocido como el "Man vs. Machine Poker Challenge". Este evento fue organizado por Facebook y CMU, y Pluribus demostró su capacidad para superar a jugadores expertos en esta variante de poker.

Lo que hace que Pluribus sea particularmente notable es su capacidad para tomar decisiones estratégicas y tácticas complejas en un juego de información imperfecta, como el poker. El poker es un juego desafiante debido a la incertidumbre inherente a las cartas ocultas y la falta de información completa sobre las acciones de los oponentes. Pluribus utiliza algoritmos avanzados de aprendizaje por refuerzo y búsqueda de árboles para desarrollar estrategias ganadoras.

Un aspecto impresionante de Pluribus es su capacidad para jugar en un nivel sobrehumano sin tener acceso a grandes cantidades de potencia informática. A diferencia de los programas de poker anteriores, que se basaban en supercomputadoras, Pluribus pudo ejecutarse en hardware estándar, lo que demuestra una eficiencia sorprendente.

El desarrollo de Pluribus tiene implicaciones más allá del poker. Su capacidad para tomar decisiones en condiciones de información imperfecta podría aplicarse a otros problemas del mundo real, como la toma de decisiones empresariales, la planificación estratégica y las negociaciones.

C. Texas Hold'em y sus reglas

El Texas Hold'em No Limit es una variante del póker que se juega con una baraja estándar de 52 cartas. Es una de las formas más populares de póker y se utiliza comúnmente en torneos y juegos en efectivo tanto en casinos físicos como en plataformas de póker en línea.

El objetivo del Texas Hold'em No Limit es ganar el bote, que es la suma de todas las apuestas realizadas por los jugadores durante una mano. El bote se gana al tener la mejor combinación de cinco cartas posible o mediante otras estrategias, como hacer que los oponentes se retiren sin mostrar sus cartas.

El juego comienza con la asignación de dos cartas privadas a cada jugador, que solo ellos pueden ver. Estas cartas se llaman "cartas de bolsillo" o "hole cards". A continuación, se reparten cinco cartas comunitarias boca arriba en el centro de la mesa, que todos los jugadores pueden utilizar para formar su mano de cinco cartas. Estas cartas comunitarias se distribuyen en tres etapas: el flop (tres cartas), el turn (una carta adicional) y el river (otra carta adicional).

Durante cada etapa de distribución de cartas comunitarias, los jugadores tienen la oportunidad de realizar apuestas. En el Texas Hold'em No Limit, los jugadores pueden apostar cualquier cantidad de fichas en cualquier momento. Esto significa que no hay límites establecidos para las apuestas y los jugadores pueden apostar todas sus fichas en una sola mano, lo que agrega un nivel de emoción e incertidumbre al juego.

A lo largo de la mano, los jugadores pueden tomar diferentes decisiones estratégicas, como igualar (poner la misma cantidad de fichas que la apuesta anterior), subir (aumentar la apuesta), retirarse (abandonar la mano y renunciar a cualquier posibilidad de ganar el bote) o hacer all-in (apostar todas las fichas restantes).
Estas decisiones se toman teniendo en cuenta las cartas de bolsillo y las cartas comunitarias, así como la fuerza percibida de la mano en comparación con las manos de los oponentes.

El Texas Hold'em No Limit es un juego complejo que implica habilidades matemáticas, análisis estratégico y capacidad para leer a los oponentes. Los jugadores deben evaluar constantemente las probabilidades de obtener diferentes combinaciones de manos, así como anticipar las jugadas de los oponentes.

IV. RICard

A. Reinforcement Learning (Aprendizaje por refuerzo)

El aprendizaje por refuerzo es una rama del aprendizaje automático que se centra en cómo un agente o sistema puede aprender a tomar decisiones óptimas a través de la interacción con su entorno. En lugar de tener un conjunto de datos de entrenamiento etiquetados, el agente aprende a través de la retroalimentación continua que recibe del entorno en forma de recompensas o castigos.

En el aprendizaje por refuerzo, el agente se enfrenta a un problema de toma de decisiones secuenciales, donde debe aprender a seleccionar las acciones adecuadas en cada estado para maximizar una señal de recompensa a largo plazo. El agente toma una acción en un estado determinado, luego el entorno responde con una recompensa y transita a un nuevo estado, y así sucesivamente. El objetivo del agente es aprender una política de toma de decisiones que maximice la recompensa acumulada a lo largo del tiempo.

El aprendizaje por refuerzo se basa en la idea de prueba y error. El agente explora diferentes acciones en su entorno y aprende de las consecuencias de esas acciones para mejorar su desempeño en el tiempo. Esto se logra utilizando un algoritmo de aprendizaje que actualiza la política del agente con base en las recompensas obtenidas y la información del estado-acción.

El enfoque principal en el aprendizaje por refuerzo es el equilibrio entre la exploración y la explotación. La exploración implica probar nuevas acciones para descubrir mejores estrategias, mientras que la explotación implica aprovechar las acciones conocidas para maximizar la recompensa a corto plazo. Es necesario encontrar un equilibrio adecuado entre ambos para obtener el mejor rendimiento.

Un concepto clave en el aprendizaje por refuerzo es la noción de "función de valor". La función de valor estima la utilidad o el valor esperado de un estado o una acción en función de las recompensas futuras que se pueden obtener. Hay dos tipos principales de funciones de valor: la función de valor de estado (que estima el valor de un estado dado) y la función de valor de acción (que estima el valor de una acción en un estado dado).

Existen varios algoritmos utilizados en el aprendizaje por refuerzo, entre los que se incluyen la programación dinámica, el aprendizaje basado en tablas (Q-learning), los métodos de Monte Carlo y los algoritmos de aprendizaje profundo, como el Deep Q-Network (DQN) y el Proximal Policy Optimization (PPO). Estos algoritmos han demostrado su eficacia en una amplia gama de aplicaciones, desde juegos de mesa como el ajedrez y el Go, hasta robótica, control de sistemas y mucho más.

B. DQN(Deep Q-Network)

El aprendizaje por refuerzo es una rama del aprendizaje automático que se basa en un agente que aprende a tomar decisiones óptimas en un entorno dinámico para maximizar una recompensa acumulativa a largo plazo. Uno de los algoritmos más populares de aprendizaje por refuerzo es el Deep Q-Network (DQN), que utiliza redes neuronales profundas para aproximar la función de valor Q.

El DQN combina el aprendizaje profundo con el aprendizaje por refuerzo, lo que permite abordar problemas complejos y de alta dimensionalidad. A continuación, se explica brevemente cómo funciona el DQN:

Representación del estado: El DQN utiliza una red neuronal profunda para aproximar la función de valor Q, que estima la recompensa esperada para cada posible acción en un estado dado.

Almacenamiento de la memoria de repetición: El DQN utiliza una memoria de repetición para almacenar las experiencias pasadas del agente. Cada experiencia está compuesta por el estado actual, la acción tomada, la recompensa obtenida, el estado siguiente y una bandera que indica si el episodio ha terminado. Esta memoria permite al agente aprender de forma más eficiente al romper la correlación temporal de las experiencias.

Selección de acciones epsilon-greedy: Para explorar y explotar el entorno, el DQN utiliza una estrategia epsilon-greedy para seleccionar acciones. Con una probabilidad epsilon, el agente selecciona una acción aleatoria para explorar el entorno, y con una probabilidad 1 - epsilon, selecciona la acción con el valor Q más alto estimado por la red neuronal.

Entrenamiento de la red neuronal: El DQN entrena la red neuronal utilizando el algoritmo de descenso de gradiente estocástico (SGD) para minimizar la diferencia cuadrada entre los valores Q predichos y los valores Q esperados. Los valores Q esperados se calculan utilizando la ecuación de Bellman, que incorpora la recompensa inmediata y el valor Q del próximo estado.

Actualización de la red objetivo: Para hacer que el entrenamiento del DQN sea más estable, se utiliza una red neuronal adicional llamada red objetivo. La red objetivo se actualiza periódicamente copiando los pesos de la red principal. Esto evita la sobreestimación de los valores Q y hace que la convergencia sea más suave.

Iteración de los pasos anteriores: El agente repite los pasos anteriores durante varias iteraciones, interactuando con el entorno, almacenando experiencias, seleccionando acciones, entrenando la red neuronal y actualizando la red objetivo. Con el tiempo, el agente aprende a tomar decisiones óptimas para maximizar la recompensa acumulativa.

C. Presentación y características de RLCard

RLCard es una biblioteca de código abierto desarrollada por el Laboratorio de Investigación en Inteligencia Artificial de la Universidad de Alberta. Está diseñada específicamente para la investigación y desarrollo de algoritmos de aprendizaje por refuerzo (RL, por sus siglas en inglés) en el campo de los juegos de cartas.

Una de las principales características de RLCard es su enfoque en los juegos de cartas, que son un dominio desafiante y rico para el aprendizaje por refuerzo. La biblioteca proporciona una amplia gama de juegos de cartas populares, como el póker de Texas Hold'em, el póker de Omaha, el póker de Leduc Hold'em, el póker de Dou Dizhu y el póker de Mahjong, entre otros. Esto permite a los investigadores y desarrolladores trabajar con una variedad de juegos de cartas y evaluar sus algoritmos RL en diferentes contextos.

Algunas de las características clave de RLCard son:

Interfaz de programación fácil de usar: RLCard proporciona una interfaz de programación intuitiva y sencilla, lo que facilita a los usuarios definir y personalizar sus propios agentes RL y entornos de juego.

Juegos de cartas predefinidos: La biblioteca ofrece una colección de juegos de cartas predefinidos, con implementaciones completas de las reglas del juego, la generación de barajas y las interacciones entre los jugadores.

Múltiples modos de juego: RLCard permite a los usuarios seleccionar entre diferentes modos de juego, como el juego de un solo jugador (contra oponentes controlados por la computadora) y el juego multijugador (con múltiples agentes RL interactuando entre sí).

Componentes de evaluación: La biblioteca incluye componentes de evaluación estándar que permiten a los usuarios medir y comparar el rendimiento de diferentes agentes RL en los juegos de cartas.

Extensibilidad: RLCard está diseñado para ser fácilmente extensible, lo que significa que los usuarios pueden agregar nuevos juegos de cartas y personalizar los aspectos del entorno y del agente según sus necesidades.

D. Arquitectura y componentes

El entorno de RLCard se basa en 4 componentes principalmente: agents, environments, games y utils. Cada componente posee diferentes características como funciones que permiten modelar el problema según se necesite.
A continuación presentaremos todos los componentes anteriormente mencionados.

Agents

Random Agent (Agente Aleatorio): Este agente toma decisiones completamente al azar, sin considerar ninguna estrategia o conocimiento sobre el juego. Es útil como punto de referencia para comparar el rendimiento de otros agentes más sofisticados.

DQN Agent (Agente DQN): Este agente utiliza el algoritmo Deep Q-Network (DQN) para aprender una función de valor Q que estima la calidad de las acciones en cada estado.
El agente elige la acción con la mayor estimación de valor Q para tomar decisiones. DQN es un algoritmo de aprendizaje por refuerzo profundo que ha demostrado buenos resultados en varios dominios.

CFR Agent (Agente CFR): Este agente utiliza el algoritmo Counterfactual Regret Minimization (CFR) para aprender una estrategia óptima en juegos de suma cero de información imperfecta. CFR busca iterativamente una estrategia que minimice el arrepentimiento promedio a lo largo de múltiples iteraciones. CFR ha sido utilizado con éxito en el póker y ha superado a jugadores humanos de alto nivel.

NFSP Agent (Agente NFSP): Este agente utiliza el algoritmo Neural Fictitious Self-Play (NFSP), que combina aprendizaje supervisado y aprendizaje por refuerzo. El agente comienza jugando contra una versión anterior de sí mismo y utiliza un modelo de red neuronal para aprender una estrategia. A medida que avanza el entrenamiento, el agente utiliza cada vez menos la estrategia aprendida y juega más basado en la búsqueda adversaria. NFSP ha demostrado un rendimiento sobresaliente en juegos de póker.

A2C Agent (Agente A2C): Este agente utiliza el algoritmo Advantage Actor-Critic (A2C) para aprender una política y una función de valor de estado. A2C combina métodos de actor-critic y aprendizaje por gradiente de políticas para mejorar la estabilidad y eficiencia del entrenamiento. Los agentes A2C pueden ser entrenados utilizando tanto enfoques basados en muestras como basados en modelos.

PPO Agent (Agente PPO): Este agente utiliza el algoritmo Proximal Policy Optimization (PPO) para aprender una política óptima. PPO busca actualizar la política en pasos pequeños y asegura que la actualización no se aleje demasiado de la política actual para mantener la estabilidad del entrenamiento. PPO ha demostrado buenos resultados en juegos de cartas y otros entornos de aprendizaje por refuerzo.

Human Agents (Agente humano): Este agente permite controlar las decisiones a realizar por teclado y jugar contra otro agente y contra modelos ya entrenados, se suele emplear para evaluar modelos ya entrenados y descubrir patrones de los agentes más experimentados.

Para nuestro objeto de estudio vamos a emplear el agente DQN ya que está basado en aprendizaje por refuerzo, está constituido por dos clases: **DQNAgent** y **Estimator**.

La clase DQNAgent implementa un agente de aprendizaje por refuerzo basado en Q-Learning con aproximación de funciones utilizando redes neuronales. Aquí está la descripción de las funciones en la clase DQNAgent:

- __init__(...): Este es el método de inicialización del agente. Configura los parámetros del agente, como el tamaño de la memoria de reproducción, el tamaño de lote, el factor de descuento, la tasa de aprendizaje, etc. También crea los estimadores Q y objetivo, y la memoria de reproducción.

- feed(...): Este método almacena la transición en la memoria de reproducción y entrena al agente. Tiene dos etapas: en la primera etapa, se llena la memoria sin entrenar, y en la segunda etapa, se entrena al agente cada ciertos pasos de tiempo.

- step(...): Este método predice la acción para generar datos de entrenamiento, pero las predicciones están desconectadas del grafo de cómputo. Utiliza un enfoque epsilon-greedy para elegir una acción basada en los valores Q estimados.

- eval_step(...): Este método predice la acción para fines de evaluación. Devuelve la mejor acción según los valores Q estimados.

- predict(...): Este método predice los valores Q enmascarados. Dado un estado, calcula los valores Q para todas las acciones posibles, pero solo devuelve los valores Q para las acciones legales.

- train(...): Este método entrena a la red neuronal. Realiza una actualización de descenso de gradiente utilizando muestras de la memoria de reproducción.

- feed_memory(...): Este método almacena una transición en la memoria de reproducción.

- set_device(...): Este método establece el dispositivo (CPU o GPU) en el que se ejecuta el agente y los estimadores.

- checkpoint_attributes(...): Este método devuelve los atributos actuales del punto de control del agente en forma de diccionario. Estos atributos se utilizan para guardar y restaurar el modelo en medio del entrenamiento.

- from_checkpoint(...): Este método restaura el modelo a partir de un punto de control.

- save_checkpoint(...): Este método guarda el punto de control del modelo.

La clase Estimator implementa un estimador de valores Q utilizando una red neuronal.

- __init__(...): Este es el método de inicialización del estimador. Configura los parámetros del estimador, como el número de acciones, la tasa de aprendizaje, la forma del estado, las capas MLP, etc. Crea una red neuronal para estimar los valores Q.

- predict_nograd(...): Este método predice los valores de acción, pero la predicción no está incluida en el grafo de cálculo. Se utiliza para predecir las acciones óptimas en el algoritmo Double-DQN.

- update(...): Este método actualiza el estimador hacia los objetivos dados. Se utiliza para entrenar la red neuronal utilizando el algoritmo de descenso de gradiente. Calcula la pérdida en el lote y realiza una actualización de los pesos de la red neuronal.

Environments

RLCard proporciona una variedad de entornos de aprendizaje por refuerzo para juegos de cartas. Estos entornos están diseñados para permitir a los investigadores y desarrolladores experimentar con diferentes juegos de cartas y aplicar algoritmos de aprendizaje por refuerzo.

A continuación, se presentan algunos de los entornos de aprendizaje por refuerzo que se incluyen en RLCard:

Texas Hold'em Limit y No Limit: Estos entornos implementan la variante popular del póker llamada Texas Hold'em. Puedes elegir entre la versión de límite (limit) o sin límite (no limit) para jugar. El objetivo es ganar el bote mediante una combinación de cartas y estrategia de apuestas.

Leduc Hold'em: Este entorno implementa una versión simplificada del póker llamada Leduc Hold'em. El juego se juega con un mazo de seis cartas y tiene reglas más sencillas que el Texas Hold'em. El objetivo sigue siendo ganar el bote a través de la estrategia de apuestas.

Dou Dizhu: Este entorno implementa el juego de cartas chino Dou Dizhu. Es un juego de cartas de estrategia en el que tres jugadores compiten para formar combinaciones de cartas y superar a los otros jugadores.

Blackjack: Este entorno implementa el popular juego de cartas Blackjack. El objetivo es obtener una mano con un valor total lo más cercano posible a 21 sin pasarse, mientras se compite contra el crupier.

Cada ecosistema está constituido por diferentes funciones, en el caso de texas holdem no-limit, contiene las explicadas a continuación:
extract_state(state): Esta función toma un estado original del juego como argumento y devuelve una observación combinada del puntaje del jugador y el puntaje observable del crupier. Primero, se extraen las acciones legales del estado y se almacenan en un diccionario ordenado llamado legal_actions. Luego, se obtienen las cartas públicas y las cartas en la mano del jugador del estado. La función crea un vector de ceros de longitud 54 y establece los índices correspondientes a las cartas presentes en el estado en 1. También asigna el puntaje de fichas del jugador y el puntaje máximo de fichas entre todos los jugadores en las dos últimas posiciones del vector. Finalmente, se devuelven el estado extraído y otros detalles relacionados.

get_payoffs(): Esta función devuelve los pagos (payoffs) del juego en forma de una lista.

_decode_action(action_id): Esta función decodifica la acción para aplicarla al juego. Toma el identificador de la acción como argumento y devuelve la acción correspondiente en forma de cadena. Primero, obtiene las acciones legales del juego y verifica si la acción decodificada no está entre las acciones legales. Si no es una acción legal, se comprueba si la acción "CHECK" es legal. Si es legal, se devuelve "CHECK"; de lo contrario, se imprime un mensaje de advertencia y se devuelve "FOLD". Si la acción decodificada es legal, se devuelve la acción correspondiente.

get_perfect_information(): Esta función devuelve la información perfecta del estado actual del juego en forma de un diccionario. La información incluye las fichas de cada jugador, las cartas públicas, las cartas en la mano de cada jugador, el jugador actual y las acciones legales disponibles.

Games

RICard proporciona una variedad de juegos de cartas para aplicar aprendizaje por refuerzo.
Estos juegos permiten a los investigadores y desarrolladores aplicar algoritmos de aprendizaje por refuerzo para crear estrategias y desarrollar agentes que sean ganadores.

Cada partida de texas holdem no-limit incluye las siguientes funciones:

configure(self, game_config): Esta función se utiliza para especificar algunos parámetros específicos del juego, como el número de jugadores, las fichas iniciales y el identificador del crupier. Si el dealer_id es None, se selecciona aleatoriamente.

init_game(self): Esta función inicializa el juego de póker sin límite. Crea un crupier, jugadores y un juez para el juego. Luego, se reparten cartas a los jugadores y se establece la ciega grande y la ciega pequeña. Se inicializa una ronda de apuestas y se devuelve el primer estado del juego y el identificador del jugador actual.

get_legal_actions(self): Esta función devuelve las acciones legales para el jugador actual en forma de lista. Las acciones legales se obtienen de la ronda de apuestas actual.

step(self, action): Esta función avanza al siguiente estado del juego después de que el jugador realice una acción específica. Verifica si la acción es válida y, si se permite, guarda una instantánea del estado actual en el historial. Luego, la ronda de apuestas avanza y se actualiza el jugador actual. Si una ronda ha terminado, se reparten más cartas públicas según corresponda. Finalmente, se devuelve el siguiente estado y el identificador del siguiente jugador.

get_state(self, player_id): Esta función devuelve el estado de un jugador específico en forma de diccionario. Incluye información como las fichas del jugador, las acciones legales, el pozo actual y la etapa actual del juego.

step_back(self): Esta función permite retroceder al estado anterior del juego si es posible. Recupera el estado anterior del historial y lo restaura como el estado actual del juego.

get_num_players(self): Esta función devuelve el número de jugadores en el juego de póker sin límite.

get_payoffs(self): Esta función devuelve los pagos (payoffs) del juego en forma de lista. Utiliza un juez para determinar los pagos basados en las manos de los jugadores y las fichas que poseen.

get_num_actions(): Esta función devuelve el número de acciones posibles en el juego. En este caso, hay 6 acciones: call, raise_half_pot, raise_pot, all_in, check y fold.

A su vez, en cada ronda, se llevan a cabo las acciones a través de las siguientes funciones:

start_new_round(self, game_pointer, raised=None): Esta función inicia una nueva ronda de apuestas. Toma como argumentos el game_pointer que indica el próximo jugador y una lista raised (opcional) para inicializar las fichas apostadas por cada jugador. Establece el game_pointer al valor proporcionado, inicializa el contador not_raise_num a 0 y, si se proporciona la lista raised, la asigna a la variable self.raised. De lo contrario, inicializa self.raised como una lista de ceros del tamaño del número de jugadores.

proceed_round(self, players, action): Esta función avanza una ronda del juego. Toma como argumentos una lista de jugadores players y la acción action realizada por el jugador actual. Según la acción tomada, actualiza el estado de las apuestas, las fichas de los jugadores y el game_pointer para indicar el próximo jugador. Si un jugador realiza una acción de "fold" (abandonar), se establece su estado como "folded". La función también verifica si un jugador ha quedado sin fichas o ha apostado todas sus fichas, estableciendo su estado correspondientemente. Luego, pasa al siguiente jugador que no haya abandonado el juego. Finalmente, devuelve el game_pointer actualizado.

get_nolimit_legal_actions(self, players): Esta función obtiene las acciones legales para el jugador actual en el juego sin límite. Toma como argumento la lista de jugadores players. La función determina las acciones legales según el estado actual de las apuestas y las fichas de los jugadores. Si un jugador no tiene suficientes fichas para igualar la apuesta máxima, no se le permite subir la apuesta. Si el pozo total de fichas es mayor que las fichas restantes de un jugador, no se le permite igualar la apuesta del pozo.
Si el jugador ya ha alcanzado la apuesta máxima de la ronda, no se le permite subir la apuesta. La función devuelve una lista de las acciones legales.

is_over(self): Esta función verifica si la ronda actual ha terminado. Devuelve True si el número de jugadores que no han subido la apuesta más el número de jugadores que ya no están jugando (han abandonado) es mayor o igual al número total de jugadores en el juego. En ese caso, la ronda se considera terminada.

La función bet(self, chips) se utiliza para realizar una apuesta por parte de un jugador en el juego de póker. Toma como argumento la cantidad de fichas chips que el jugador desea apostar.
La función comienza verificando si la cantidad de fichas especificada (chips) es menor o igual a las fichas restantes del jugador (self.remained_chips). Si es así, se asigna la cantidad de fichas especificada a la variable quantity. De lo contrario, si el jugador no tiene suficientes fichas para realizar la apuesta completa, se asigna el valor de self.remained_chips a quantity, lo que significa que el jugador apostará todas sus fichas restantes.

Luego, la cantidad de fichas apostadas (quantity) se agrega al total de fichas apostadas por el jugador (self.in_chips). Esto permite realizar un seguimiento de las fichas que el jugador ha apostado durante el juego.

Finalmente, la cantidad de fichas apostadas (quantity) se resta de las fichas restantes del jugador (self.remained_chips), actualizando así el número de fichas que le quedan al jugador después de hacer la apuesta.

Utils

Por último, tenemos una serie de funciones que permiten el desarrollo del juego:

set_seed(seed): Esta función establece la semilla para generar números aleatorios. Si se proporciona un valor seed, la función realiza lo siguiente:
Comprueba los paquetes instalados en el entorno utilizando el comando pip freeze.
Si el paquete "torch" está instalado, establece la semilla para generar números aleatorios en la biblioteca PyTorch utilizando torch.manual_seed(seed) y configura torch.backends.cudnn.deterministic en True.
Establece la semilla para generar números aleatorios en la biblioteca NumPy utilizando np.random.seed(seed).
Establece la semilla para generar números aleatorios en la biblioteca random utilizando random.seed(seed).

get_device(): Esta función devuelve el dispositivo de ejecución disponible, ya sea "cuda:0" si la GPU está disponible o "cpu" si no lo está.

init_standard_deck(): Esta función inicializa una baraja estándar de 52 cartas y devuelve una lista de objetos de tipo "Card" que representan cada carta.

init_54_deck(): Esta función inicializa una baraja de 54 cartas, que incluye las 52 cartas estándar y dos cartas adicionales ("BJ" y "RJ") utilizadas en un juego específico. Devuelve una lista de objetos de tipo "Card" que representan cada carta.

rank2int(rank): Esta función toma una cadena de texto rank que representa el rango de una carta y devuelve el número correspondiente. Por ejemplo, si rank es "A", devuelve 14, si es "T", devuelve 10, y si es "J", devuelve 11. Si rank es una cadena vacía, devuelve -1. Si el rank no es válido, devuelve None.

elegent_form(card): Esta función toma una cadena de texto card que representa una carta y devuelve una representación elegante de la carta en forma de cadena. Por ejemplo, si card es "H2" (el dos de corazones), devuelve "♥2". Utiliza caracteres unicode para los símbolos de los palos de la baraja.

print_card(cards): Esta función imprime de manera legible una o varias cartas. Puede tomar como argumento una cadena de texto que representa una carta o una lista de cadenas de texto que representan varias cartas. Utiliza caracteres unicode para imprimir las cartas de manera visualmente atractiva.

reorganize(trajectories, payoffs): Esta función reorganiza las trayectorias de juego para que sean compatibles con algoritmos de aprendizaje por refuerzo (RL). Toma una lista de trayectorias y una lista de pagos para los jugadores. Devuelve una nueva lista de trayectorias que se pueden utilizar para entrenar algoritmos RL.

remove_illegal(action_probs, legal_actions): Esta función elimina las acciones ilegales de un vector de probabilidades y normaliza el vector resultante. Toma como argumentos un vector de probabilidades de acciones y una lista de índices de acciones legales. Devuelve un nuevo vector de probabilidades sin las acciones ilegales, normalizado de manera que la suma de las probabilidades sea 1.

tournament(env, num): Esta función evalúa el rendimiento de los agentes en un entorno de juego. Toma como argumentos el entorno (env) y el número de juegos (num) que se jugarán. Devuelve una lista de pagos promedio para cada jugador.

plot_curve(csv_path, save_path, algorithm): Esta función lee datos de un archivo CSV y traza los resultados. Toma como argumentos la ruta al archivo CSV, la ruta donde se guardará la gráfica y el algoritmo utilizado. Lee los datos del archivo CSV, traza una curva de recompensas en función del número de episodios y guarda la gráfica en la ubicación especificada.

E. Preparación del entorno de trabajo y adaptaciones

Justificación

Es importante comprender y planificar el proceso antes de comenzar a escribir código por varias razones:

- ➔ Claridad y organización: Al tener una comprensión clara de los pasos y las tareas necesarias, se puede estructurar el código de manera más organizada y coherente. Esto ayudará a mantener un flujo de trabajo eficiente y facilitará la comprensión y el mantenimiento del código en el futuro.

- ➔ Identificar dependencias y requisitos: Al comprender los componentes y las dependencias necesarias, se asegura de tener todas las bibliotecas y herramientas adecuadas instaladas. Esto evitará problemas y retrasos innecesarios.

- ➔ Configuración adecuada del entorno: Al comprender los pasos necesarios para configurar correctamente el entorno de trabajo, como importar los módulos necesarios y configurar la semilla aleatoria, se podrá establecer las bases sólidas para el desarrollo del proyecto.

- ➔ Planificación y diseño del algoritmo: Al tener una visión general de los pasos requeridos, se podrá planificar y diseñar mejor el algoritmo que se utilizará. Esto implica decidir qué técnicas y enfoques específicos implementar, cómo se estructurará el entrenamiento y cómo se evaluará el rendimiento del agente.

- ➔ Evaluación de viabilidad y alcance: Antes de comenzar a escribir código, es importante evaluar la viabilidad y el alcance de tu proyecto. Esto implica determinar si los recursos y el tiempo disponibles son adecuados para el objetivo que te has propuesto. También permitirá realizar ajustes y considerar posibles limitaciones antes de invertir demasiado tiempo en el desarrollo.

Instalación

En primer lugar, se han instalado todas las dependencias necesarias, incluyendo RLCard y sus dependencias.
Se ha de utilizar un entorno de desarrollo como Visual Studio Code para crear un nuevo proyecto.
Configurar el proyecto para utilizar Python y asegurarse de tener un archivo principal para el código.

Importar los módulos necesarios

Importar los módulos necesarios para el proyecto, como os, argparse, torch y rlcard.
Asegurarse de importar los componentes relevantes, como los agentes, el juego de póker No-Limit Texas Hold'em y las utilidades necesarias para la configuración del entorno.

Configurar el entorno de juego y el agente

Configurar el entorno del juego para jugar al póker No-Limit Texas Hold'em en modo heads-up (uno contra uno).
Inicializar el agente DQN, que utilizará una red neuronal profunda para aprender a jugar al póker.
Opcional: Configurar la semilla aleatoria:

Para que los experimentos sean reproducibles, se ha de configurar una semilla aleatoria utilizando la función set_seed(seed). Esto asegurará que los resultados sean consistentes en diferentes ejecuciones.

Entrenar el agente

- ➔ Definir los parámetros de entrenamiento, como el número total de episodios y la frecuencia de evaluación del rendimiento del agente.
- ➔ Utilizar un bucle para iterar a través de los episodios de entrenamiento.
- ➔ Jugar un episodio utilizando el agente y el entorno de juego, y obtener las trayectorias y los resultados del episodio.
- ➔ Reorganizar las trayectorias para que sean compatibles con el algoritmo DQN y entrenar el agente utilizando esas trayectorias.
- ➔ Registrar los resultados del episodio, como la recompensa obtenida por el agente.
- ➔ Evaluar el rendimiento del agente periódicamente utilizando la función de torneo.

Probar el agente contra un agente aleatorio y humano

- ➔ Crear un agente aleatorio para competir contra el agente DQN entrenado.
- ➔ Utilizar la función de torneo para evaluar el rendimiento del agente DQN frente al agente aleatorio.
- ➔ Observar los resultados y comparar el rendimiento del agente DQN con el agente aleatorio.
- ➔ Siguiendo estos pasos, se prepara el entorno de trabajo, entrenado un agente DQN para jugar al póker en modo heads-up y luego probado su rendimiento contra un jugador humano y contra un agente aleatorio. Esto permitirá evaluar la capacidad de aprendizaje del agente y su capacidad para enfrentarse a diferentes oponentes.

F. Pruebas de concepto

Para comprobar el correcto funcionamiento de RLCard, vamos a proceder a realizar varias pruebas con diferentes juegos y características, probaremos tanto el blackjack como el poker limit holdem ya que más adelante estudiaremos a fondo el texas no-limit holdem que supone el juego más interesante de estudiar.

BlackJackHuman

Esta primera prueba permite probar el agente humano, es decir, controlar las decisiones que toma un jugador por teclado.

Para comprobar qué datos nos permite conocer RLCard sobre nuestra ejecución vamos a extraer algunos campos:

Observation: [20 10]
Legal Actions: OrderedDict([(0, None), (1, None)])
Raw Observation: {'actions': ('hit', 'stand'), 'player0 hand': ['HK', 'SK'], 'dealer hand': ['DT'], 'state': (['HK', 'SK'], ['DT'])}
Raw Legal Actions: ['hit', 'stand']
Action Record: [(0, 'stand')]

A través de *trajectories* podemos obtener, las acciones posibles a realizar, siendo 0, no pedir carta y 1 pedirla, sabemos las cartas que tenemos, dos reyes, uno de corazones (hearts) y otro de picas (spades) así como la acción realizada, en este caso, stand (no pedir carta).

También podemos obtener los payoffs o recompensas recibidas, en este caso 0:

Reward for Player 0: -1
Siendo el player 0, el jugador es controlado por teclado.
La recompensa puede ser 0 o 1 en función de si se gana o no la mano, solo se puede apostar 1 ficha por cada jugador.

BlackJackRandom

Esta segunda prueba consiste en entrenar un agente DQN y comprobar que resultados arroja, es imposible hacerlo ganador ya que depende mucho del azar, y al ser un juego de casino, su reward medio a largo plazo siempre será negativo y coincidirá con el RTP(Return to player) inverso, siempre perderemos un poco al largo plazo.

Esta segunda prueba permite entrenar al jugador mediante un agente DQN para que mejore su juego por nosotros, si se realizan 20.000 manos de 25 juegos diferentes, obtenemos estos resultados:

Media de recompensas: -0.059800000000000006
Desviacion estándar de recompensas: 0.19893171980789953

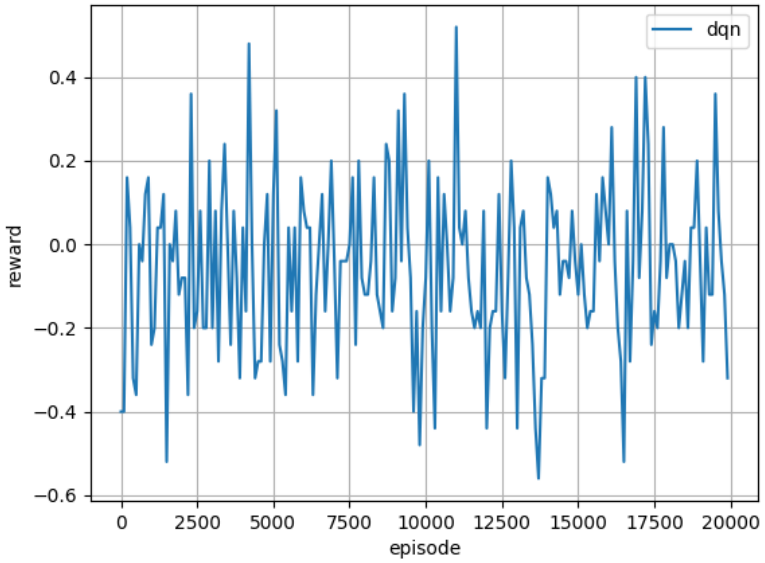

Imagen 1: Entrenamiento de 20.000 manos de BlackJack de 25 partidas distintas evaluándose cada 100 manos.

Ahora bien, si se realizan 50.000 manos de BlackJack de 25 partidas distintas, se reduce la pérdida de recompensas y la desviación estándar disminuye un poco, los resultados son los siguientes:

Media de recompensas: -0.05512
Desviacion estándar de recompensas: 0.1946256506418811

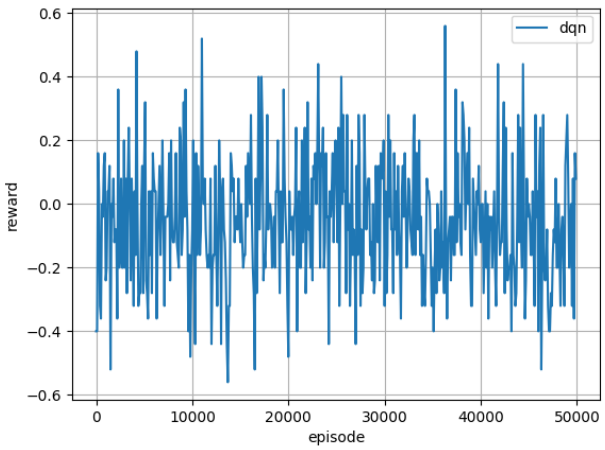

Imagen 2: Entrenamiento de 50.000 manos de BlackJack de 25 partidas distintas evaluándose cada 100 manos.

Si usamos ese modelo para evaluar los resultados en 10.000 manos en 25 partidas una vez entrenado, los resultados son los siguientes:

Media de recompensas: -0.15439999999999998
Desviación estándar de recompensas: 0.1891625349969297

Son resultados mucho mejores, en parte debido a que son menos número de manos por lo que la varianza afecta mucho más.

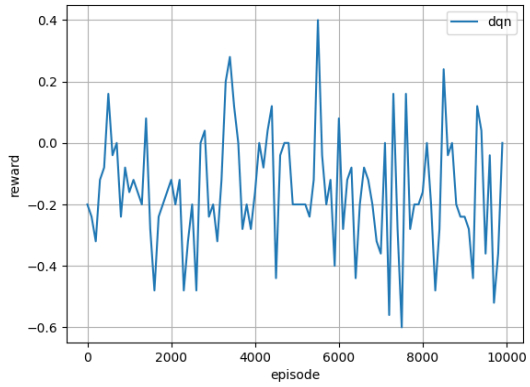

Imagen 3: Conjunto de test 10.000 manos de BlackJack de 25 partidas distintas evaluándose cada 100 manos.

LimitHoldemHuman

Esta primera prueba permite probar el agente humano, es decir, controlar las decisiones que toma un jugador por teclado.

Para comprobar qué datos nos permite conocer RLCard sobre nuestra ejecución vamos a extraer algunos campos:

'raw_obs': {'hand': ['HJ', 'D3'], 'public_cards': ['SA', 'CQ', 'DK', 'S8'], 'all_chips': [10, 6], 'my_chips': 6, 'legal_actions': ['raise', 'fold', 'check'], 'raise_nums': [1, 1, 1,
0]}, 'raw_legal_actions': ['raise', 'fold', 'check'], 'action_record': [(1, 'call'), (0, 'raise'), (1, 'call'), (0, 'raise'), (1, 'call'), (0, 'raise'), (1, 'fold')]}]]
[3. -3.]

A través de *trajectories* podemos obtener, las cartas que tenemos, en este ejemplo una J de corazones y un 3 de diamantes, las cartas comunitarias, As de picas, Dama de corazones rey de Diamantes y 8 de picas, podemos ver las fichas maximas a poder apostar, en este caso 10 para el rival y 6 para mi, podemos ver las acciones a realizar siendo 0 raise, 1 fold y 2 check.
Y por último, también podemos ver las acciones realizadas por parte de ambos jugadores.

También podemos obtener los payoffs o recompensas recibidas, en este caso 3:

Reward for Player 0: 3
Siendo el player 0, el jugador es controlado por teclado.
La recompensa equivale al número de fichas que se gana en una ronda.

LimitHoldemRandom

Esta cuarta prueba permite entrenar al jugador mediante un agente DQN para que mejore su juego, a diferencia del BlackJack, este juego se practica contra otros jugadores, no contra el casino por lo que los resultados son muy diferentes, a continuación, vamos a realizar numerosas pruebas para comprobar su éxito contra un oponente aleatorio.

Esta prueba permite entrenar al jugador mediante un agente DQN para que mejore su juego por nosotros, si se realizan 20.000 manos de 25 juegos diferentes, obtenemos estos resultados:

Media de recompensas: 2.5202
Desviación estándar de recompensas: 0.9417955480205652

Son resultados mucho mejores, en parte debido a que son menos número de manos por lo que la varianza afecta mucho más.

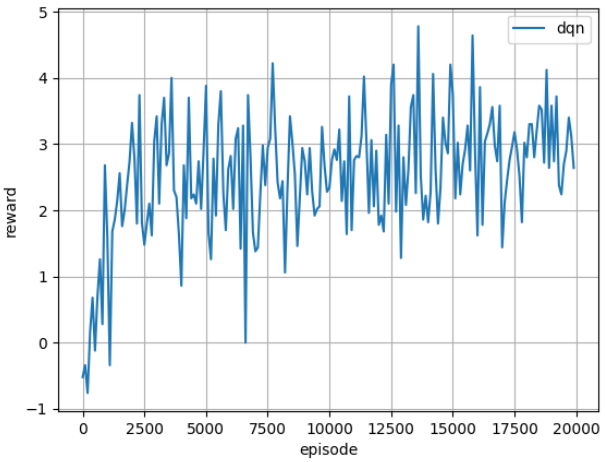

Imagen x: Entrenamiento de 20.000 manos de Limit Holdem de 25 partidas distintas evaluándose cada 100 manos.

Ahora bien, si se realizan 50.000 manos de Limit Holdem de 25 partidas distintas, se reduce la pérdida de recompensas y la desviación estándar disminuye un poco, los resultados son los siguientes:

Media de recompensas: 2.5927200000000004
Desviacion estándar de recompensas: 0.8267597383338522

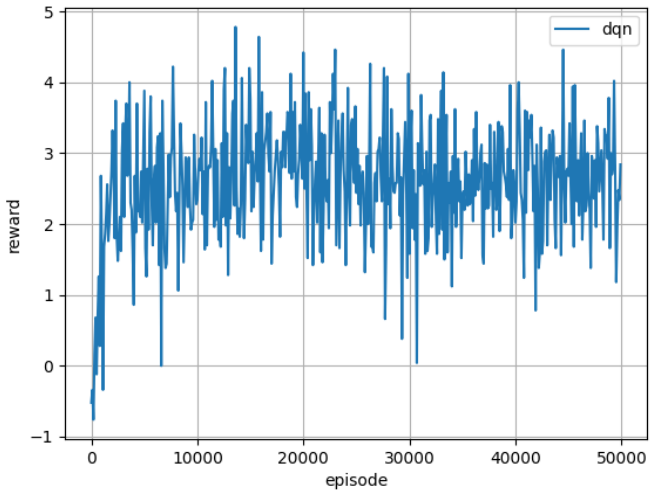

Imagen x: Entrenamiento de 50.000 manos de Limit Holdem de 25 partidas distintas evaluándose cada 100 manos.

Si usamos ese modelo para evaluar los resultados en 10.000 manos en 25 partidas una vez entrenado, los resultados son los siguientes:

Media de recompensas: 0.7324
Desviacion estándar de recompensas: 0.6301116142911877

Son resultados mucho peores, en parte debido a que son menor número de manos por lo que la varianza afecta mucho más.

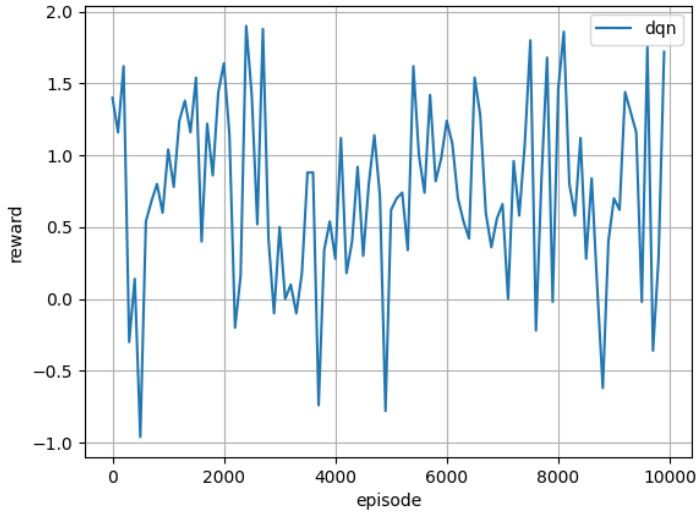

Imagen 3: Conjunto de test 10.000 manos de Limit Holdem de 25 partidas distintas evaluándose cada 100 manos.

V. Modelado del juego Texas Hold'em en RLCard

A. Definición de funciones y simplificaciones

Se han realizado numerosas adaptaciones en este proyecto.
En primer lugar se ha de considerar que el poker texas Holdem no limit es muy complicado de representar por lo que en primer lugar se ha decidido generar un entorno heads up esto quiere decir que se enfrentarán dos jugadores comúnmente el heads up es un modo de juego muy competitivo.

Es el entorno más duro del póker. esta modalidad permite simplificar mucho el estudio del juego debido a que sólo intervendrán dos agentes, se realizarán dos modelos principales uno para entrenar el

agente enfrentándose a un jugador aleatorio, es lo que se entenderá por un jugador recreacional, un jugador de recreacional es aquel que no tiene conocimientos sobre estadística y estrategia sobre el juego por lo que sus acciones se realizan en torno a sus sensaciones y percepciones sobre el mismo.

También se ha decidido simplificar las acciones que puede llevar un jugador a cabo, estas son: fold, descartarse de las cartas y comenzar una nueva ronda, call, pagar la apuesta y raise, aumentar el tamaño de la puesta. En un entorno real existen diferentes tipos de raise, el jugador puede subir al tamaño que quiera, sin límites de ahí proviene el nombre de esta modalidad sin embargo solo se permitirá aumentar la apuesta a un tamaño determinado.

También se ha decidido iniciar las partidas con 100 ciegas grandes, en un entorno real las fichas de los jugadores se mantendrían durante todas las partidas sin embargo hemos decidido reiniciarlas cada partida.

Se ha entrenado el agente durante cien mil manos y 25 torneos o partidas diferentes esto permite evaluar el agente durante un largo periodo de tiempo. Al no disponer de la potencia suficiente no se ha podido evaluar el agente durante un mayor número de manos por lo que cabe esperar que los resultados no sean muy precisos ya que se necesitarán muchas más manos, en torno a 5 millones para poder estimar valores cercanos a los reales.

En cuanto a las funciones, Se ha empleado la función train para poder entrenar el modelo, se han llevado a cabo diferentes pruebas con diferentes opciones de número de manos y número de torneos para poder evaluar el desempeño de la gente DQN, También se ha ido evaluando este resultado por medio del reward obtenido cada X número de manos, de esta forma podemos tener una idea de cuantas ciegas grandes estaría ganando nuestro agente cada X número de manos dado.

Los jugadores profesionales evalúan su desempeño mediante el número de ciegas grandes ganado cada cien manos, esto es lo que se conoce por ganancia BB/cien.

Por último se ha guardado ese modelo para su posterior evaluación de test empleando otro agente aleatorio y también un jugador de poker profesional, este jugador es Enrique Iago un jugador profesional que actualmente se encuentra jugando No Limit 200 y torneos de buy-in medio mil dólares.

B. Espacio de estados

Cartas del jugador: Se refiere a las cartas privadas que cada jugador tiene en su mano. Estas cartas son visibles solo para el jugador correspondiente y pueden variar en su combinación y valor. Por ejemplo, un jugador podría tener un par de ases (A-A), una combinación más baja como (2-3) o incluso una mano vacía si aún no ha recibido cartas.

Cartas comunitarias: Son las cartas reveladas en la mesa que son visibles para todos los jugadores. Estas cartas se colocan gradualmente en la mesa durante las distintas rondas de apuestas. Por ejemplo, en un momento dado puede haber tres cartas comunitarias visibles en la mesa, como (A-10-6).

Acciones de los jugadores: Se refiere a las decisiones tomadas por los jugadores hasta el momento en la ronda actual. Estas acciones pueden incluir apostar, igualar, subir o retirarse, entre otras posibilidades. El historial de acciones refleja las elecciones individuales de cada jugador a medida que avanza el juego.

Posición del jugador: Se refiere a la ubicación relativa de cada jugador en la mesa en relación con el botón del crupier. Esto puede determinar el orden de juego y las estrategias empleadas por los jugadores. Por ejemplo, un jugador puede ocupar la posición de "early position" (primeros jugadores después del botón), "middle position" (jugadores en medio) o "late position" (últimos jugadores antes del botón).

Historial de apuestas: Representa la secuencia de apuestas realizadas durante la mano actual. Esto incluye información sobre las apuestas realizadas por cada jugador, los montos apostados y cualquier aumento o re-raise efectuado. El historial de apuestas permite tener un registro de las acciones y movimientos financieros de los jugadores a lo largo de la partida.

C. Proceso de aprendizaje de las políticas óptimas (parámetros del modelo)

Para llevar a cabo nuestro modelo hemos comenzado realizando un agente DQN que juega contra un oponente aleatorio durante cien mil manos. Los valores iniciales de nuestro agente DQN son los siguientes:

discount_factor=0.99: El factor de descuento, también conocido como gamma, es un valor que determina la importancia relativa de las recompensas futuras en comparación con las recompensas inmediatas. Un valor de 0.99 indica que se da un peso alto a las recompensas futuras, lo que significa que el agente considerará las recompensas a largo plazo al tomar decisiones.

epsilon_start=1.0: Epsilon es el parámetro de exploración utilizado en el algoritmo de aprendizaje por refuerzo. Un valor de 1.0 indica que al comienzo del entrenamiento, el agente elige acciones de manera completamente aleatoria (exploración máxima) para descubrir y aprender sobre el entorno.

epsilon_end=0.1: Epsilon también se decae gradualmente a lo largo del entrenamiento para pasar de una fase de exploración a una fase de explotación. Un valor de 0.1 indica que al final del entrenamiento, el agente elegirá la acción óptima la mayoría de las veces (explotación máxima), utilizando el conocimiento adquirido hasta ese momento.

epsilon_decay_steps=20000: Este parámetro indica el número de pasos de tiempo necesarios para que epsilon se reduzca desde su valor inicial hasta el valor final. A medida que el número de pasos aumenta, epsilon disminuye gradualmente, lo que permite que el agente se vuelva más selectivo y confiable en sus acciones a medida que avanza el entrenamiento.

train_every=1: Indica cada cuántos pasos de tiempo el agente debe entrenarse en un lote de muestras de experiencia. En este caso, el agente se entrena después de cada paso de tiempo, lo que significa que se actualiza y mejora constantemente.

learning_rate=0.00005: La tasa de aprendizaje determina el tamaño de los ajustes realizados en los pesos de la red neuronal durante el entrenamiento. Un valor de 0.00005 indica que los ajustes en los pesos son pequeños, lo que puede ser beneficioso para una convergencia estable y suave del modelo.

D. Pruebas y evaluación de resultados (comparativas)

VI. Conclusiones (limitaciones también y si he cumplido el objetivo)

A modo de conclusión, se lograron los objetivos establecidos en este proyecto al desarrollar un agente DQN capaz de ganar en partidas de poker Texas No Limit Hold'em. Las ganancias del agente se muestran en la gráfica X y se incrementan a medida que se juegan más manos.

Se realizaron pruebas del modelo tanto con jugadores profesionales como no profesionales, así como con agentes aleatorios, con el fin de evaluar su desempeño. Sin embargo, es importante mencionar que

existen algunas limitaciones en este estudio. En primer lugar, no se cuenta con suficiente potencia de cómputo para entrenar al agente de manera adecuada, lo cual requeriría un entrenamiento más extenso y una evaluación a lo largo de un mayor número de manos. Además, el modelo implementado a través de RLCard no es óptimo para jugar mediante entrada de teclado, ya que las decisiones se vuelven lentas. Aunque hay una interfaz disponible, no está diseñada para este modo de juego.

Es importante destacar que el objetivo principal del proyecto fue construir un agente capaz de ganar en el poker, pero no se tiene pleno conocimiento de cómo lo logra. Como suele ocurrir en proyectos de Inteligencia Artificial, a menudo se trata de una "caja negra", donde conocemos los resultados y las entradas, pero desconocemos las acciones específicas que lleva a cabo para obtener esos resultados.

A través de este proyecto, se pudieron extraer algunas conclusiones durante las partidas. Se observó que los jugadores tienden a realizar acciones basadas en experiencias pasadas, son conscientes de qué manos son buenas y cuáles no, y no toman decisiones precipitadas, sino que aprenden sobre la marcha. Por lo general, el agente logra ganar frente a agentes aleatorios y jugadores no profesionales, ya que la agresividad en el juego, es decir, realizar numerosas subidas, suele ser crucial al enfrentarse a este tipo de jugadores.

Es importante tener en cuenta que este estudio se enfocó más en evaluar la capacidad de aprendizaje del agente en el contexto del poker actual, en lugar de la estrategia de poker convencional. Actualmente, muchos jugadores profesionales estudian con programas que se basan en una estrategia común, es decir, acciones que la mayoría de los jugadores realizan. Sin embargo, nuestro agente parte desde cero, lo que complica el estudio.

Finalmente, se agradece a los jugadores que participaron en este estudio por su tiempo y dedicación, ya que su contribución fue fundamental para el desarrollo y la evaluación del agente.

VII. Bibliografía

[1] Hwang, D., & Bourgeois, B. (2005). Advanced Pot-Limit Omaha: Small Ball

[2] Short-Handed Play. Two Plus Two Publishing.Chen, B., & Ankenman, J. (2006). The Mathematics of Poker. Conjelco.

[3] Sklansky, D., & Malmuth, M. (1999). Hold'em Poker for Advanced Players. Two Plus Two Publishing.

[4] https://pluribusai.com/

[5] https://www.muycomputer.com/2017/02/01/libratus-ia-poker/

[6] Richard S. Sutton y Andrew G. Barto. "Reinforcement Learning: An Introduction" (Aprendizaje por refuerzo: una introducción)

[7] David Silver, et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search" (Dominando el juego del Go con redes neuronales profundas y búsqueda en árbol).

[8] Volodymyr Mnih, et al. "Human-level control through deep reinforcement learning" (Control a nivel humano a través del aprendizaje por refuerzo profundo).

[9] John Schulman, et al. "Proximal Policy Optimization Algorithms" (Algoritmos de optimización de políticas proximales).

[10] Pieter Abbeel y John Schulman. "Deep Reinforcement Learning" (Profundo aprendizaje por refuerzo).
[11] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
[12] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. The MIT Press.
[13] Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al. (2015). Continuous control with deep reinforcement learning
[14] Hessel, M., Modayil, J., van Hasselt, H., et al. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning

VIII. Anexos

A.Pruebas realizadas

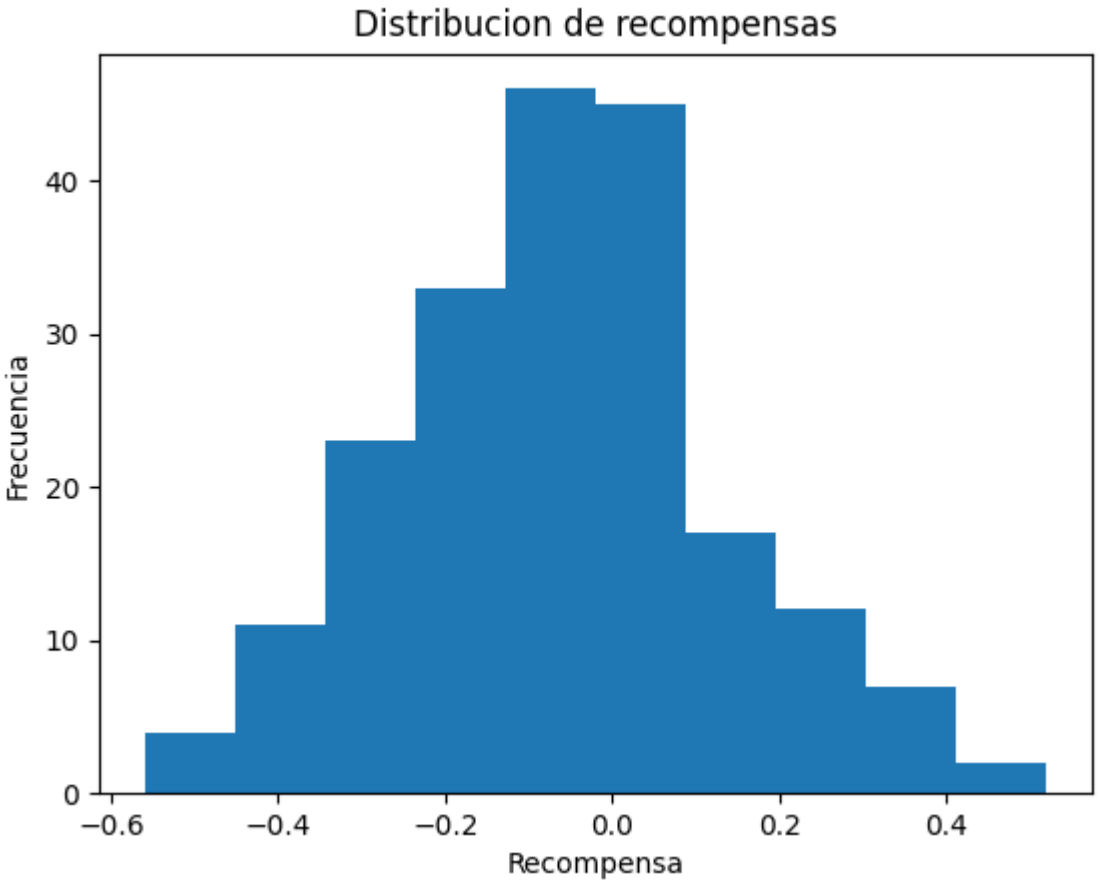

Imagen x: BlackJack entrenamiento con agente DQN: 20.000 manos en 25 agentes.

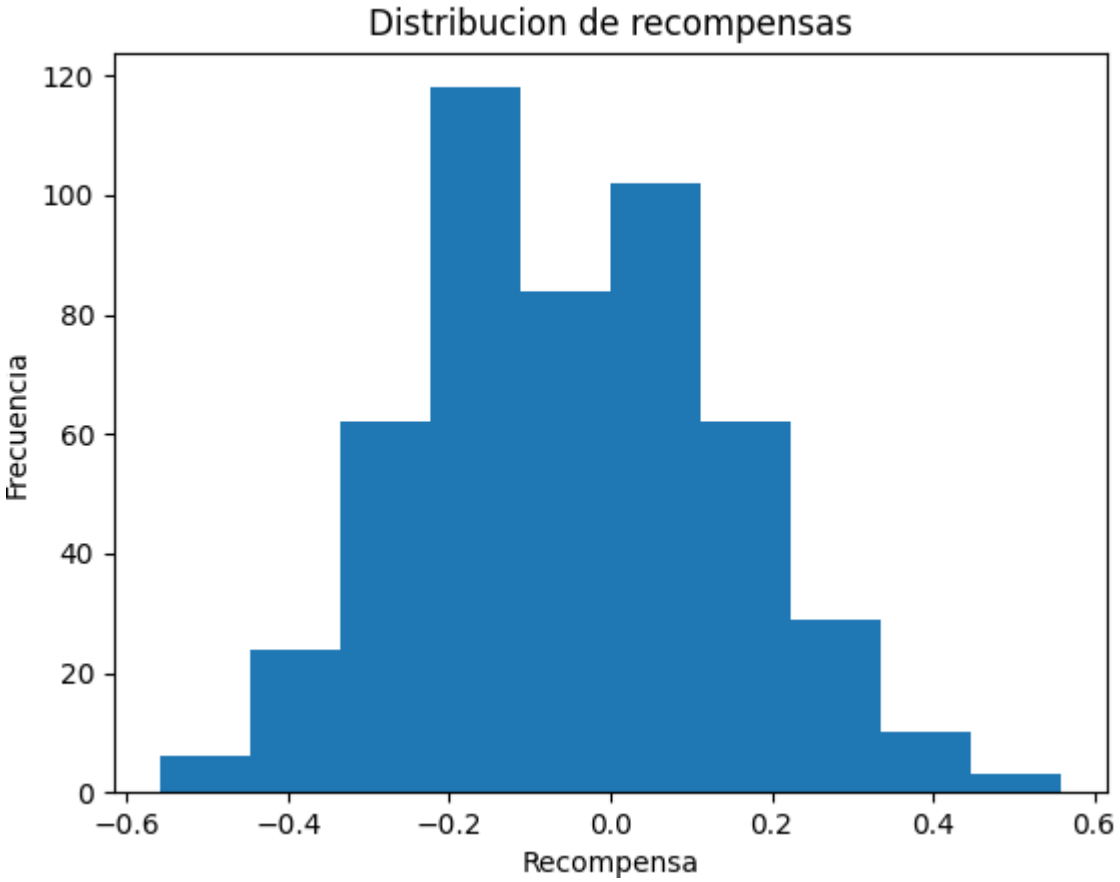

Imagen x: BlackJack entrenamiento con agente DQN: 50.000 manos en 25 agentes.

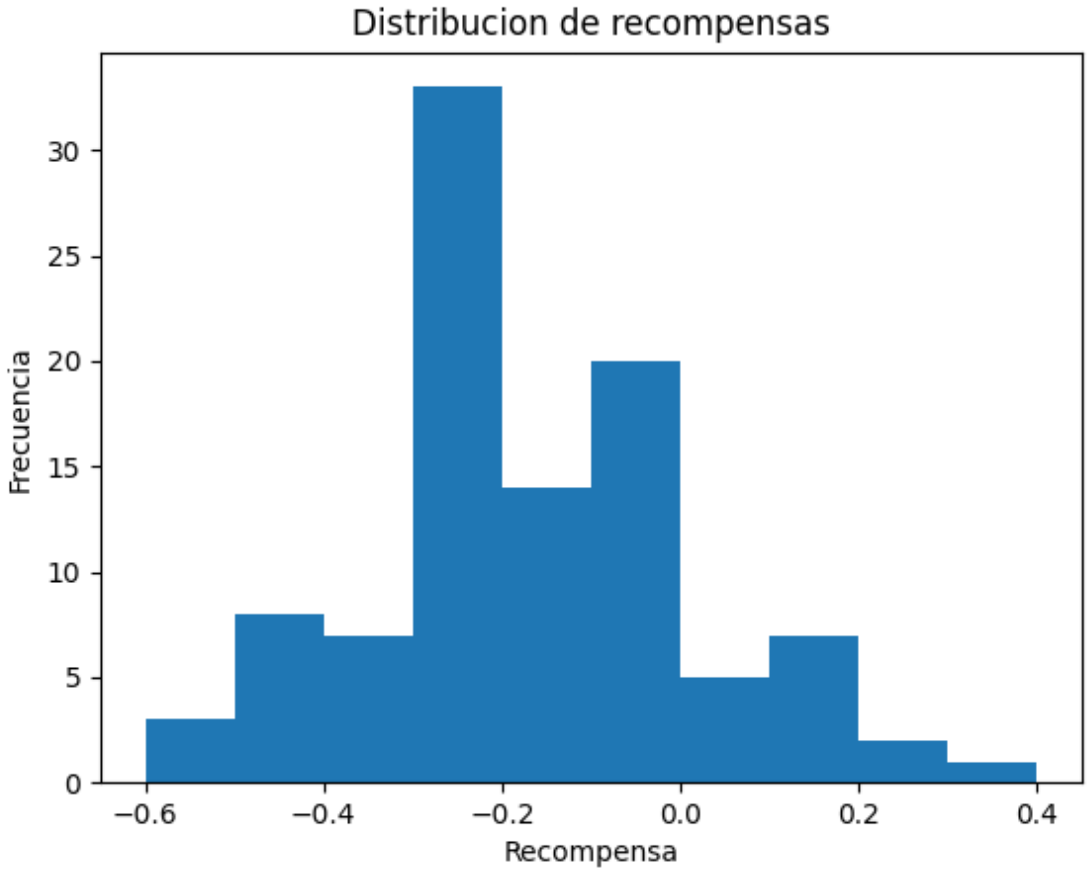

Imagen x: BlackJack test con agente DQN: 10.000 manos en 25 agentes.

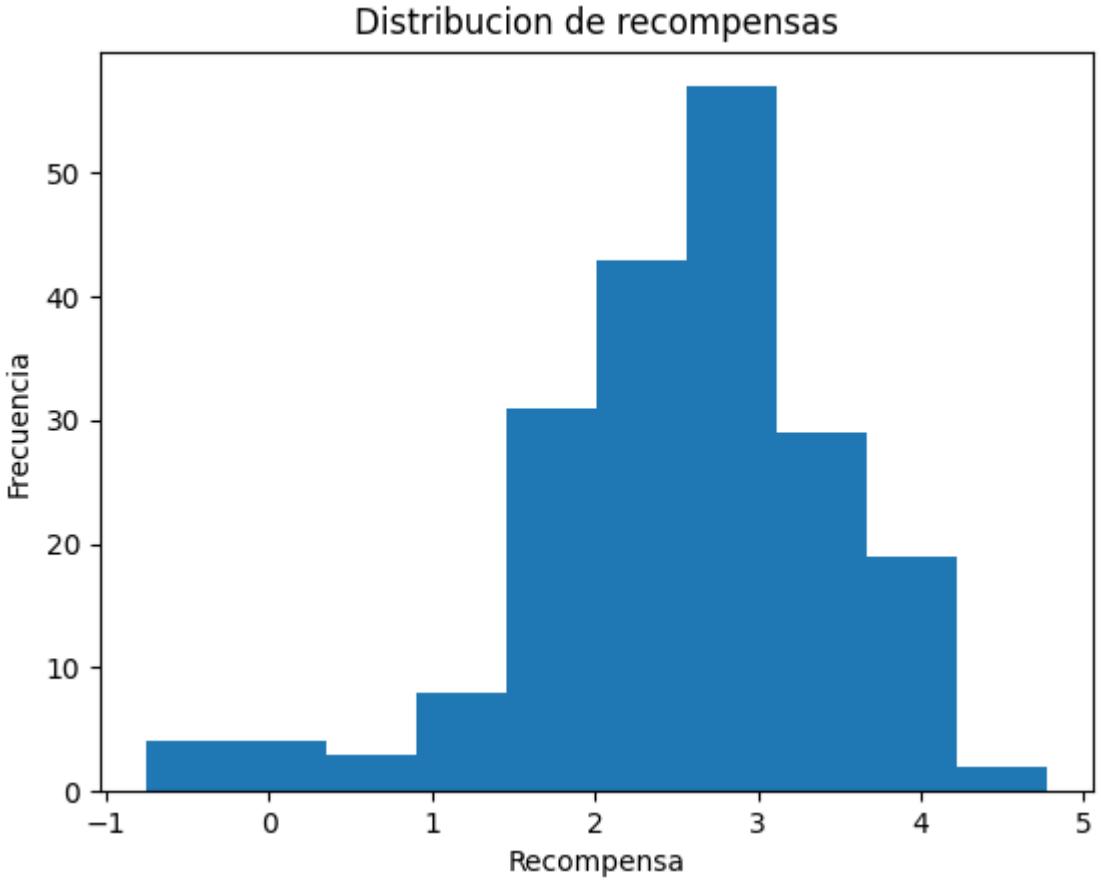

Imagen x: LimitHoldem test con agente DQN: 20.000 manos en 25 agentes.

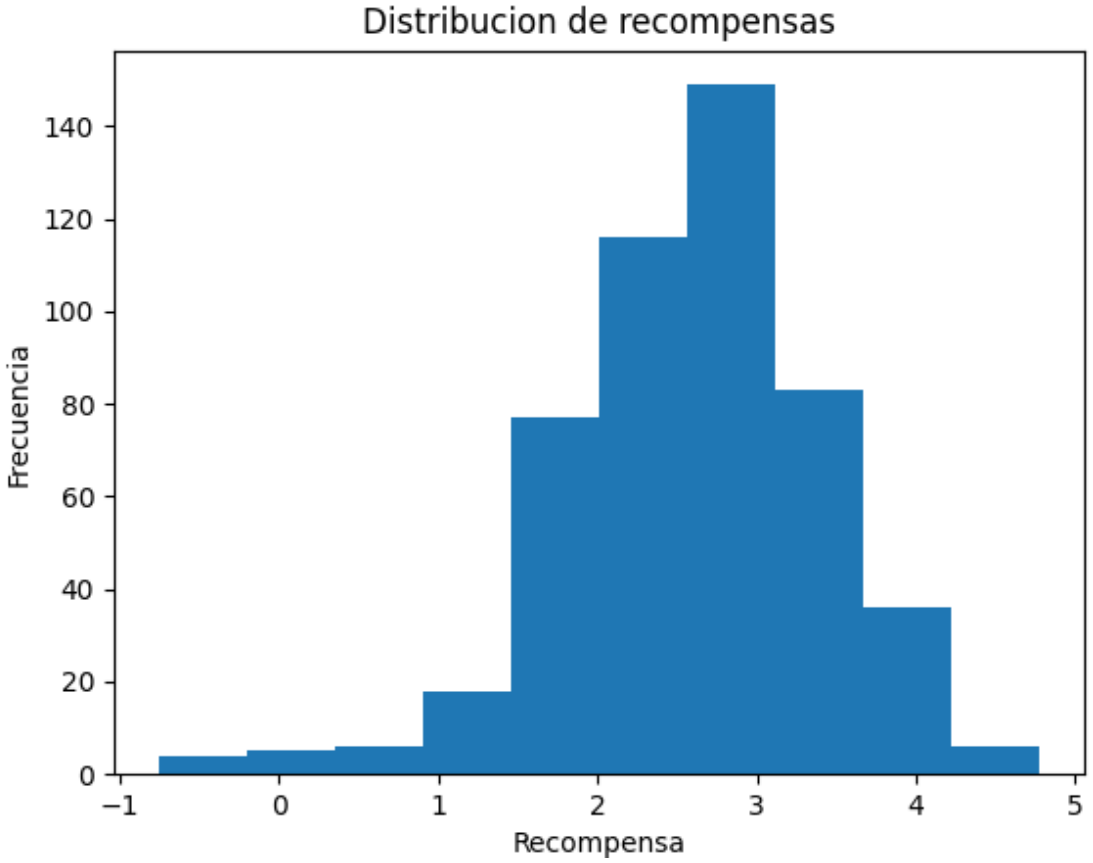

Imagen x: LimitHoldem test con agente DQN: 50.000 manos en 25 agentes.

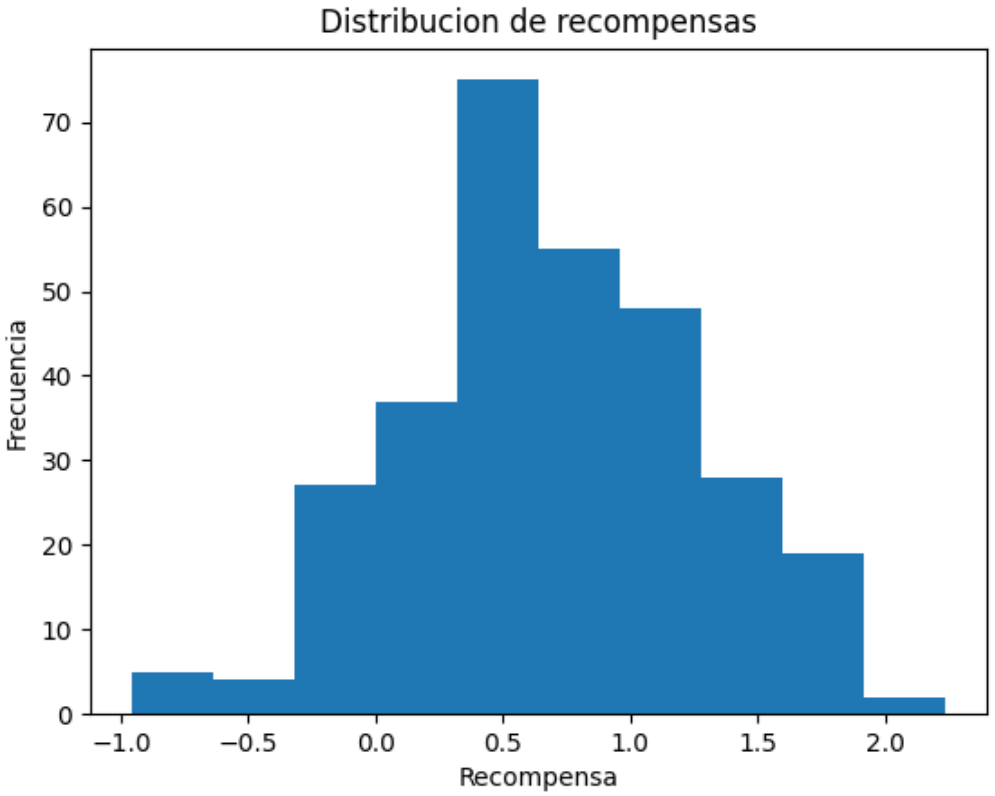

Imagen x: Limtest con agente DQN: 10.000 manos en 25 agentes.

B. Glosario de términos

1. All-in: Apostar todas las fichas restantes en una sola mano.
2. Ante: Apuesta obligatoria que todos los jugadores deben hacer antes de repartir las cartas.
3. Big Blind: Apuesta forzada realizada por el segundo jugador a la izquierda del crupier.
4. Bluff: Hacer una apuesta o movimiento engañoso para intentar que los oponentes se retiren.
5. Board: Las cartas comunitarias visibles en la mesa.
6. Call: Igualar la apuesta realizada por un jugador.
7. Check: Pasar sin apostar en una ronda de apuestas.
8. Chip: Ficha utilizada para representar el valor apostado.
9. Crupier: Persona encargada de repartir las cartas y gestionar el juego.
10. Dealer: Jugador que actúa como crupier en una mano determinada.
11. Fish: Jugador inexperto y susceptible de cometer errores.

12. Flop: Las tres primeras cartas comunitarias repartidas en el centro de la mesa.
13. Fold: Retirarse de la mano sin invertir más fichas.
14. Flush: Mano formada por cinco cartas del mismo palo.
15. Full House: Mano compuesta por un trío y una pareja.
16. Hand: Las cartas individuales que un jugador tiene en su poder.
17. Heads-up: Juego de póker entre solo dos jugadores.
18. Hole Cards: Las cartas individuales repartidas boca abajo a cada jugador.
19. Implied Odds: Las ganancias potenciales esperadas en relación con el costo de una apuesta.
20. Kicker: Carta de mayor valor utilizada para desempatar en caso de empate en una mano.
21. Late Position: Posición de juego cercana al botón del crupier, donde se actúa después de la mayoría de los jugadores.
22. Main Pot: Bote principal en una mano, separado de los botes laterales.
23. Muck: Descartar las cartas sin mostrarlas al final de una mano.
24. Nuts: La mejor posible mano en una ronda determinada.
25. Odds: Probabilidad numérica de que ocurra un evento.
26. Out: Carta que podría mejorar la mano de un jugador.
27. Overpair: Par de cartas de mayor valor que cualquier carta comunitaria en el flop.
28. Pot: El total de fichas en juego en una mano determinada.
29. Pot Limit: Límite de apuesta basado en el tamaño actual del bote.
30. Quads: Cuatro cartas del mismo valor.
31. River: La quinta y última carta comunitaria repartida en la mesa.
32. Royal Flush: La mejor mano de póker, formada por un 10, J, Q, K y A del mismo palo.
33. Semi-Bluff: Hacer una apuesta con una mano incompleta pero con potencial para mejorar.
34. Showdown: El momento final de una mano, donde los jugadores revelan sus cartas y se determina el ganador.
35. Small Blind: Apuesta forzada realizada por el primer jugador a la izquierda del crupier.
36. Split Pot: Dividir el bote entre dos o más jugadores con manos de igual valor.
37. Stack: El número total de fichas que un jugador tiene en su posesión.
38. Straight: Mano formada por cinco cartas consecutivas de cualquier palo.
39. Suited: Cartas del mismo palo.
40. Three of a Kind: Mano compuesta por tres cartas del mismo valor.
41. Tilt: Estado emocional negativo que lleva a un jugador a tomar decisiones irracionales debido a la frustración o la ira.
42. Turn: La cuarta carta comunitaria repartida en la mesa.
43. Under the Gun: La posición de juego justo después de las ciegas, donde se actúa primero en una ronda de apuestas.

44. Value Bet: Realizar una apuesta con la intención de obtener ganancias de una mano sólida.
45. Walk: Cuando todos los jugadores se retiran y el jugador de la ciega recoge el bote sin ver el flop.