

Máster en Big Data, Tecnología y Analítica Avanzada

Machine Learning II

**Práctica Reinforcement Learning**

Memoria descriptiva

Autores:

Celia Quiles Alemañ

Álvaro Ezquerro Pérez

María Calvo de Mora Román

Madrid

15 – Mayo – 2024

**Índice de contenido**

[**1.** **Introducción** 3](#_Toc166428045)

[1.1. Contexto del proyecto y entorno Blackjack de Gym 3](#_Toc166428046)

[1.1.1. Contexto y objetivos del proyecto 3](#_Toc166428047)

[1.1.2. Entorno Blackjack 3](#_Toc166428048)

[1.2. Descripción general de técnicas empleadas 4](#_Toc166428049)

[1.3. Estructura del proyecto 5](#_Toc166428050)

[**2.** **Agente aleatorio base** 6](#_Toc166428051)

[2.1. Introducción 6](#_Toc166428052)

[2.2. Entrenamiento 6](#_Toc166428053)

[2.3. Resultados 6](#_Toc166428054)

[**3.** **Q- Learning** 7](#_Toc166428055)

[3.1. Introducción 7](#_Toc166428056)

[3.2. Entrenamiento 8](#_Toc166428057)

[3.3. Resultados 9](#_Toc166428058)

[**4.** **Deep Q-Network (DQN)** 11](#_Toc166428059)

[4.1. Introducción 11](#_Toc166428060)

[4.2. Entrenamiento 12](#_Toc166428061)

[4.3. Resultados 12](#_Toc166428062)

[**5.** **Dueling Double DQN y Dueling DQN** 13](#_Toc166428063)

[5.1. Introducción 13](#_Toc166428064)

[5.2. Entrenamiento 14](#_Toc166428065)

[5.3. Resultados 14](#_Toc166428066)

[**6.** **Advantage Actor – Critic (A2C)** 16](#_Toc166428067)

[6.1. Introducción 16](#_Toc166428068)

[6.2. Entrenamiento 16](#_Toc166428069)

[6.3. Resultados 17](#_Toc166428070)

[**7.** **Conclusiones del proyecto** 19](#_Toc166428071)

[7.1. Idoneidad de los algoritmos empleados 19](#_Toc166428072)

[7.2. Comparativa de métodos empleados y análisis de resultados 19](#_Toc166428073)

[7.3. Conclusiones 19](#_Toc166428074)

# **Introducción**

## Contexto del proyecto y entorno Blackjack de Gym

### Contexto y objetivos del proyecto

El presente trabajo aborda el campo del Aprendizaje por Refuerzo (Reinforcement Learning, RL), una rama del aprendizaje automático que se ocupa de cómo los agentes deben tomar decisiones en un entorno para maximizar alguna noción de recompensa acumulativa. Este proyecto tiene como objetivo explorar y aplicar diferentes técnicas de Aprendizaje por Refuerzo para entrenar una serie de agentes capaces de jugar al Blackjack, no solo como un juego de azar clásico, sino como un problema estratégico donde cada acción debe ser calculada en función de maximizar las posibilidades de ganar contra el crupier.

Así, con el mencionado objetivo, se hace uso del entorno de la librería Open AI Gymnasium Blackjack disponible a través del siguiente [enlace](https://gymnasium.farama.org/environments/toy_text/blackjack/). Gracias a este entorno, nos será posible implementar el juego, entrenar y evaluar a nuestros agentes, así como estudiar la idoneidad de las técnicas de Aprendizaje por Refuerzo desarrolladas. El código del proyecto, tanto su implementación como sus resultados, se encuentran disponibles en el siguiente [repositorio de GitLab](https://gitlab.com/cma_mbd_2023/rl/-/blob/main/README.md?ref_type=heads). Además, junto al código en el repositorio mencionado y la presente memoria del trabajo, se entrega un breve vídeo de menos de un minuto de duración con los resultados de los algoritmos ensayados.

Por tanto, este proyecto pretende demostrar prácticamente cómo las técnicas de Aprendizaje por Refuerzo pueden ser empleadas para entrenar a un agente a jugar al Blackjack. Al comparar el rendimiento y funcionamiento de diferentes algoritmos, se podrán obtener conclusiones acerca de las fortalezas y debilidades de cada una de las técnicas, permitiendo determinar su idoneidad para resolver un problema y reto como el mencionado.

### Entorno Blackjack

El entorno Blackjack, proporcionado por la biblioteca Gym de OpenAI, modela el juego clásico de cartas donde el objetivo es vencer al crupier obteniendo una suma de cartas más cercana a 21 que las cartas del crupier, sin superar 21. Se trata de un entorno del tipo "Toy Text", que ofrece una representación simplificada pero desafiante del juego, ideal para aplicar y estudiar algoritmos de aprendizaje por refuerzo.

El espacio de acción en este entorno es discreto, constando de dos posibles acciones: pedir carta (hit) o no pedir más cartas y plantarse (stand). El espacio de observación se compone de tres características: la suma actual de las cartas del jugador, el valor de la carta visible del crupier y si el jugador posee un as utilizable. El juego comienza con el crupier y el jugador recibiendo dos cartas, aunque solo una de las cartas del crupier es visible para el jugador. El jugador entonces decide su acción basada en su observación actual, buscando acercarse lo más posible a 21 sin pasarse.

Así, el mencionado espacio de observación muestra las tres informaciones comentadas: la suma del valor de las cartas del jugador y una de las cartas del crupier, la cual se encuentra boca arriba. Además, se informará si el jugador tiene un as utilizable, cuyo valor puede ser 1 u 11 en función del valor de resto de cartas en mano del jugador. Para mostrar esto de manera representativa, la Figura 1 muestra la visión del espacio de observación del entorno.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 1. Ejemplo del espacio de observación del agente en el entorno Blackjack de Gymnasium.

## Descripción general de técnicas empleadas

En este proyecto, se evaluarán en términos de rendimiento e idoneidad varias técnicas de Aprendizaje por Refuerzo, cada una con características particulares que podrían adaptarse mejor o peor al entorno de Blackjack proporcionado por la librería Gym. Entre dichas técnicas se encuentran: Q-Learning, Deep Q-Network (DQN), Dueling Deep Q-Network y Advantage Actor-Critic (A2C):

* **Q-Learning:** es un algoritmo de Aprendizaje por Refuerzo basado en valores. Específicamente, permite el aprendizaje y obtención de una tabla de valores Q (*Quality values*) para cada uno de los pares (estado, acción) del entorno con el que interactúa el agente mediante la exploración del entorno y la obtención de recompensas. Además, emplea una política denominada *epsilon-greedy* que permite determinar el compromiso entre la *exploración* (tomar nuevas acciones aleatorias con el fin de explorar nuevas posibilidades e intentar reducir la posibilidad de caída en mínimos locales) y la *explotación* (tomar acciones ya conocidas que aseguran un buen resultado) del comportamiento basado en acciones que se va adquiriendo.

En términos de nuestro caso de uso específico, dado que el entorno Blackjack es un entorno con un espacio de estados y acciones relativamente pequeño, discreto y bien definido, Q-Learning parece especialmente adecuado. Este método será útil para establecer una base de comparación con técnicas más avanzadas, observando cómo un algoritmo relativamente simple puede o no capturar la estrategia óptima en un juego de decisión secuencial como el Blackjack.

* **Deep Q-Network (DQN):** este algoritmo extiende Q-Learning al basarse en Deep Learning mediante el empleo de una Red Neuronal para aproximar la función Q. Esto permite al algoritmo gestionar espacios de estado más grandes y complejos que no se pueden abordar fácilmente con métodos tabulares como los basados en valores. En relación al entorno Blackjack que nos concierne, aunque el espacio de estado del Blackjack no es excesivamente grande y ya habrá sido explorado con técnicas tabulares, utilizar DQN nos permitirá adentrarnos en el mundo de técnicas más complejas y explorar la efectividad de las aproximaciones de redes neuronales en entornos que involucran decisiones secuenciales.
* **Dueling Deep Q-Network:** de manera similar a DQN utiliza una red neuronal profunda para aproximar una política o una función de valor. La diferencia clave ante una DQN radica en que, en vez de predecir el Q-valor de cada acción, predice el valor de cada estado y la ventaja de cada acción. Esto se realiza dividiendo la última capa en dos, una para cada predicción (valor de estado y ventaja), por último, se unen ambas predicciones para obtener el Q-valor de cada acción. Este algoritmo se implementará, tanto con dos redes una principal y una target, como solo con la principal.
* **Advantage Actor-Critic (A2C):** este algoritmo permite combinar los dos casos ya explorados con las técnicas anteriores, esto es, combinar la estimación de la función valor Q y la aproximación de la política óptima. Específicamente, se realiza mediante el uso de dos modelos de Redes Neuronales separados: una de ellas para el actor y otra para el crítico. La primera, correspondiente al actor, toma decisiones dentro del entorno en base al estado actual. La segunda, la cual concierne al crítico, aprende a estimar el valor de las acciones tomadas por el actor a través de computar sobre ellas la función valor, de manera que proporciona un feedback directo al actor sobre si sus acciones son óptimas o no a la vez que aprende el valor de la ventaja A.

En este sentido, para este caso A2C puede ofrecer un balance entre las ventajas de los métodos basados en valores y los basados en políticas, posiblemente conduciendo a un aprendizaje más estable al reducir la varianza del actor al centrarse el crítico en el aprendizaje de los valores de la ventaja, así como provocar una mayor eficiencia en el aprendizaje.Así, todas las mencionadas técnicas serán evaluadas no solo en términos de rendimiento, considerando la tasa de victorias, el total de recompensas y penalizaciones obtenidas durante el entrenamiento, así como el número total de pasos dados por el agente. Si no que también se evaluarán las técnicas en términos de facilidad de implementación y eficiencia computacional.

## Estructura del proyecto

El proyecto consta de varias partes:

* **Memoria descriptiva**: el presente documento que contiene detalle de las técnicas y los resultados obtenidos, así como la comparación entre las técnicas empleadas
* **Vídeo de las técnicas empleadas**: en la carpeta de la entrega *‘docs/videos’* se encuentran los videos de los resultados de los algoritmos ensayados. Todo ellos, no suman más de un minuto de duración total.
* **Repositorio**: la carpeta *‘rl’* contiene el [repositorio de GitLab](https://gitlab.com/cma_mbd_2023/rl) con el código elaborado para el entrenamiento de todos los modelos implementados.
* **Modelos**: la carpeta *‘src/models’* contiene el resultado de los distintos algoritmos implementados para su posible reutilización posterior sin necesidad de volver a realizar el entrenamiento.

# **Agente aleatorio base**

## Introducción

Como primera aproximación a la búsqueda de una solución para el juego en el entorno de Blackjack, se procedió a establecer un punto de referencia básico que permita evaluar la eficacia de las técnicas implementadas en el proyecto, entre las que se encuentran algoritmos más sofisticados. Para este propósito, se ha implementado un agente aleatorio en el entorno de Blackjack. Este agente no sigue ninguna estrategia en particular; simplemente selecciona acciones al azar en cada paso de la partida, sin ningún aprendizaje o ajuste basado en la retroalimentación del entorno. La performance de este agente sirve como línea base y punto de referencia para comparar la efectividad de los algoritmos de Aprendizaje por Refuerzo que se implementarán posteriormente, de modo que se pueda determinar si dichos algoritmos implementados suponen una mejora significativa o no frente a la estrategia de buscar una solución a la fuerza bruta mediante la toma de acciones aleatorias dentro del espacio de acciones posibles.

## Entrenamiento

Dado que el agente aleatorio no requiere entrenamiento en el sentido tradicional, el "entrenamiento" en este contexto simplemente implica ejecutar 1000 episodios en los cuales el agente toma decisiones completamente al azar. Este proceso es importante para establecer las métricas base de desempeño del entorno sin ninguna intervención inteligente de entrenamiento. El código empleado para el entrenamiento del mencionado agente se puede encontrar en el archivo ‘*Q-learning.ipynb* ‘al inicio de dicho archivo.

## Resultados

Los resultados del agente aleatorio proporcionan una perspectiva clara sobre el comportamiento esperado sin estrategias de aprendizaje, simplemente buscando una solución a la fuerza bruta y de manera aleatoria. Se realizaron 1000 pruebas de evaluación para asegurar la robustez estadística de los resultados.

Gráfico, Gráfico en cascada

Descripción generada automáticamente

Figura 2. Estadísticas resultado del desempeño del Agente Aleatorio durante el proceso de evaluación.

Tal y como se puede observar en la Figura 2, el agente aleatorio mostró una tasa de victorias extremadamente baja (27%), reflejando la ineficacia de tomar decisiones al azar en un juego que requiere decisiones estratégicas. Las pérdidas fueron predominantes (70%), lo que indica que las acciones aleatorias tienden a implicar acciones poco eficaces para alcanzar la meta. Los empates, alcanzan un porcentaje del 1%. Así, dado el alto número de decisiones subóptimas, el ratio de pérdidas fue significativamente alta. Por último, la media de pasos en cada episodio muestran la cantidad de acciones tomadas antes de terminar el juego, en este caso alcanzando un total de 1.35 pasos, proporcionando una idea de cuán rápido el agente termina sus partidas, ya sea por alcanzar un 21 o por decisiones de detenerse.

Por tanto, los resultados reflejan la naturaleza aleatoria y no estratégica del agente, subrayando la importancia de aplicar técnicas de aprendizaje por refuerzo que puedan aprender y adaptarse a las estructuras y reglas del Blackjack para mejorar significativamente estos resultados iniciales. En este sentido, los algoritmos detallados a continuación pretenden aprender de una forma u otra (ya sea mediante el aprendizaje de los valores Q, la función Q o la política ) aplicando técnicas de Aprendizaje Automático con el objetivo de mejorar las estadísticas puramente aleatorias.

# **Q- Learning**

## Introducción

El algoritmo Q-Learning es una técnica fundamental en el campo de Aprendizaje Automático que permite a un agente aprender cómo actuar óptimamente en un entorno con los que inicialmente no están familiarizados, es decir, desconocido, mediante la iteración de experiencias y el cálculo y actualización de la función valor Q. Más concretamente, este método se clasifica como un algoritmo de aprendizaje “*off-policy*”, lo que significa que el agente aprende explorando el entorno y a la vez explotando el conocimiento actual mediante una política .

El algoritmo Q-Learning es una adaptación del algoritmo de iteración del valor Q, y está diseñado para solucionar problemas de decisión secuenciales donde las probabilidades de transición y las recompensas no son conocidas a priori. Opera actualizando las estimaciones de la función Q, que representa el valor esperado de tomar una acción *a* en un estado *s*, y se actualiza iterativamente utilizando la regla:

Ecuación 1. Formula de actualización de valores Q en Q-Learning

Donde:

* representa el valor estimado en la k-ésima iteración para la acción *a* en el estado *s*
* es la ratio de aprendizaje (*learning rate*), que controla cuánto influyen las nuevas informaciones en la actualización de los valores Q.
* la recompensa recibida tras realizar la acción *a* en el estado *s*
* es el factor de descuento, que pondera la importancia de las recompensas futuras respecto a las inmediatas
* es el máximo valor estimado de Q para el siguiente estado , evaluando todas las posibles acciones . Esto refleja la mejor recompensa esperada en el próximo paso bajo la política óptima.

Así, esta ecuación de actualización se basa en la idea de que para cada par estado-acción (*s, a*), registra una media de las recompensas *r* que obtiene el agente al abandonar el estado *s* con la acción *a*, más la suma de las recompensas futuras descontadas que espera obtener. Para estimar esta suma, tomamos el máximo de las estimaciones de Q-Value para el siguiente estado *s′,* ya que suponemos que la política objetivo actuaría de forma óptima a partir de ese momento. Por tanto, este enfoque permite que el agente actualice sus estimaciones de Q basándose en las recompensas inmediatas y el potencial de las futuras, guiando sus decisiones hacia comportamientos que maximicen las recompensas acumulativas.

El proceso de aprendizaje en Q-Learning funciona observando a un agente jugar (por ejemplo, aleatoriamente) y mejorando gradualmente sus estimaciones de los valores Q. Una vez que las estimaciones de los valores Q son precisas (o lo suficientemente cerca), la política óptima consiste en elegir la acción que tenga el valor Q más alto (es decir, la política codiciosa).

En nuestro contexto del Blackjack, el agente utiliza Q-Learning para determinar si es más conveniente pedir una carta adicional ("hit") o detenerse ("stand") en función de la situación actual de la partida. Las acciones se toman en base a la política derivada de la función Q, que se actualiza continuamente a medida que el agente experimenta diferentes resultados de sus acciones. Para ello, se emplea la matriz Q como una estructura de datos para almacenar los valores de la función Q, que indica el valor de la recompensa para cada acción en un estado determinado. El espacio de estados se define por la combinación en forma de tupla, que contiene (suma de las cartas del jugador, la carta visible del crupier y la presencia de un as utilizable). Así, considerando que sólo hay las dos posibles acciones mencionadas (“hit” o “stand”) en cada estado y que cada uno de estos estados viene representado por una tupla, nuestra matriz Q presenta filas donde cada fila representa el valor Q estimado de las acciones en diferentes estados del juego. Así, por ejemplo, una fila de la matriz tendría el aspecto: (20, 8, False) à [ 2.20662255 -0.98262223], donde:

* El estado *s* (20, 8, False) indica que la suma de las cartas del jugador es 20, la carta visible del crupier es un 8 y no hay presencia de un as utilizable.
* Valor estimado de Q para las posibles acciones en este estado [ 2.20662255 -0.98262223], implicando que el valor Q estimado para la acción de “quedarse” en ese estado es 2.21, mientras que el valor Q estimado para la acción de “pedir carta” en ese estado es 0.98.

Tal y como indica el algoritmo, al iniciar el entrenamiento del agente inicializaremos los valores de la matriz Q de forma aleatoria, de manera que a medida que el agente juega se actualizarán los valores Q utilizando la Ecuación 1. Así, con el tiempo, lo valores de la matriz Q convergen hacia los valores óptimos, permitiendo al agente tomar las acciones óptimas en cada estado una vez entrenado.

## Entrenamiento

Como se ha comentado anteriormente, el proceso de entrenamiento de nuestro agente Q-Learning se enfoca en el desarrollo de una tabla Q que captura el valor esperado de cada acción en cada estado posible del juego de Blackjack. Esta tabla es fundamental para que el agente pueda tomar decisiones informadas y mejorar su desempeño a lo largo del tiempo.

Para la configuración del entorno y los parámetros del algoritmo se utiliza de nuevo, igual que para el Agente Aleatorio, el entorno Blackjack-v1 de OpenAI Gym para simular las partidas de Blackjack. En cuanto a los parámetros de configuración para el entrenamiento se emplea una ratio de aprendizaje de 0.1, un factor de descuento de 0.6 y una ratio de exploración de 0.1.

El código de implementación de este algoritmo se puede encontrar en el archivo *‘Q-learning.ipynb’* del repositorio. Así mismo, la matriz Q resultante del entrenamiento se puede encontrar en el archivo *‘src/models/q\_table\_100000.pkl’*

Como ya se mencionó, la tabla Q se inicializa con valores pequeños y aleatorios cercanos a cero, lo que permite cierta variabilidad inicial en las acciones seleccionadas por el agente. El proceso de entrenamiento se repite para un número de 100000 episodios, permitiendo al agente experimentar y aprender de una amplia gama de situaciones.

## Resultados

Después de entrenar al agente, evaluamos su desempeño en términos de victorias, pérdidas, empates y pasos medios realizados en un total de 1000 episodios de prueba. Esto nos proporciona una medida cuantitativa de cuán bien el agente ha aprendido a jugar al Blackjack y ajustar su estrategia basada en el aprendizaje adquirido durante el entrenamiento.

Gráfico, Gráfico de barras

Descripción generada automáticamente

Figura 3.Estadísticas resultado del desempeño del Agente Q-Learning durante el proceso de evaluación.

La Figura 3 muestra el desempeño del Agente Q-Learning. En comparación con el Agente Aleatorio y sus resultados representados en la Figura 2, este Agente Q-Learning logró una tasa de victorias del 37.1%, una mejora sustancial comparado con el 27% del agente aleatorio, reflejando una capacidad mejorada para tomar decisiones estratégicas. Aunque la tasa de pérdidas sigue siendo alta, se observa una reducción en la misma de 70% a 53.2%, indicando un manejo más eficaz de las situaciones de juego. Así mismo, los empates se aumentaron significativamente de 1% a 9.7%, lo que sugiere que el agente ha aprendido a evitar decisiones que conducen a la pérdida si no que el agente prefiere empatar a perder. Por último, en relación a los pasos dados de media por episodio se observó un aumento de 1.35 a 1.69 pasos. Todos estos cambios sugieren que el agente Q-Learning está tomando decisiones más informadas y efectivas gracias a la actualización de la tabla Q, que ofrece una guía sobre qué acciones son más prometedoras en términos de recompensas futuras, el agente aprende estrategias que no sólo evitan pérdidas, sino que también buscan optimizar el resultado de cada juego.

La matriz Q resultante del entrenamiento tiene la siguiente forma:

Texto

Descripción generada automáticamente

Figura 4. Matriz Q de entrenamiento Q-Learning tras 100000 episodios.

En la Figura 4, se observa como que aumenta la suma de cartas del jugador, acercándose a 21 y sin la presencia de un as utilizable, aumenta a su vez el valor estimado Q de la acción de “quedarse”. Para comprender mejor cómo el agente ha aprendido a evaluar sus opciones, examinamos algunos valores de la tabla Q obtenida después del entrenamiento:

* (20, 8, False) 🡪 [2.21, -0.98]: Indica que plantarse con un total de 20 cuando el crupier muestra un 8 es muy favorable, mientras que pedir es desventajoso.
* (8, 8, False) 🡪 [-0.25, 0.27]: Muestra que es ligeramente mejor pedir otra carta cuando el total es bajo y el crupier muestra un 8.
* (12, 10, False) 🡪 [-0.75, -0.56] : Refleja la situación difícil de tener un total de 12 contra un 10 del crupier; sin embargo, pedir carta sigue siendo ligeramente menos riesgoso que plantarse.

Estos ejemplos ilustran cómo el agente utiliza la tabla Q para tomar decisiones basadas en el estado actual del juego y las cartas mostradas por el crupier. Además, resultan bastante evidentes, y las decisiones a las que ha llegado el modelo serían las mismas que tomaría un jugador que conociera la dinámica del juego, lo que nos lleva a pensar que el mecanismo de aprendizaje es correcto.

Finalmente, el algoritmo de Q-Learning demostró una mejora notable respecto al agente aleatorio. Aunque las mejoras no son dramáticas, dada la naturaleza del juego donde el crupier generalmente tiene ventaja estadística, el algoritmo ha demostrado ser eficaz para incrementar la tasa de victorias y disminuir las penalizaciones. Esto valida la capacidad del agente para aprender y adaptar su estrategia de forma efectiva. El rápido tiempo de respuesta de este algoritmo, la facilidad en su implementación y la mejora en la toma de decisiones sugieren que Q-Learning es una opción viable para juegos de decisión rápida como el Blackjack.

A pesar de los buenos resultados obtenidos con un método tabular tal como el Q-Learning, es importante considerar que este algoritmo tiene la desventaja de funcionar de manera no excesivamente óptima en algunos entornos estocásticos (con fin) tal y como lo es el entrono empleado en el presente trabajo Blackjack. En dichos entornos, el Q-Learning tiende a sobreestimar los valores de las acciones debido al uso de a la hora de actualizar los valores de la tabla Q. Además, Q-Learning no escala bien para casos de procesos con muchos estados y acciones, donde se vuelve ineficiente trabajar con tablas Q excesivamente grandes. Debido a estas desventajas del mencionado algoritmo, y teniendo en cuenta la posible sobreestimación y ineficiencia en algunos procesos estocásticos como el que nos concierne, se decide probar algoritmos no tabulares y de esta manera comprobar si la aplicación de dichos algoritmos más complejos supone una mejora significativa.

# **Deep Q-Network (DQN)**

## Introducción

El algoritmo Deep Q-Network (DQN) representa una evolución significativa en el campo del aprendizaje por refuerzo, integrando técnicas de aprendizaje profundo con los principios clásicos del Q-learning. Un DQN emplea redes neuronales profundas para aproximar la política o la función Q, que evalúa el valor esperado de realizar una acción en un estado determinado, buscando maximizar las recompensas futuras. Así, a diferencia del Q-learning tradicional, que requiere una tabla para almacenar los valores Q para cada par estado-acción (lo cual se vuelve impracticable en entornos con un gran número de estados o acciones), DQN utiliza un aproximador funcional. Esto permite al DQN manejar espacios de estado y acción de alta dimensionalidad o incluso continuos, donde establecer una tabla completa sería ineficiente o imposible.

En el DQN, la red neuronal se configura para recibir el estado del entorno como entrada y producir como salida el valor Q para cada acción posible (Figura 5). Esto convierte al problema de aprendizaje por refuerzo en uno más cercano al aprendizaje supervisado, facilitando el uso de técnicas estándar de optimización como el descenso de gradiente para entrenar la red.

Diagrama

Descripción generada automáticamente

Figura 5. Esquema explicativo de la estrctura del algoritmo de Deep Q-Learning.

Una característica clave del DQN es el uso de la memoria de repetición (*experience replay*), donde las experiencias pasadas (definidas por el estado, la acción realizada, la recompensa recibida y el nuevo estado alcanzado) se almacenan. Durante el entrenamiento, se extraen muestras aleatorias de esta memoria para actualizar los pesos de la red. Este método no solo mejora la eficiencia de los datos, permitiendo que experiencias únicas influyan múltiples actualizaciones, sino que también ayuda a minimizar las correlaciones entre muestras consecutivas, evitando caer en mínimos locales durante el entrenamiento, el sobre entrenamiento y la no convergencia.

Por tanto, el algoritmo DQN transfiere el poder del aprendizaje profundo al dominio del aprendizaje por refuerzo, permitiendo a los agentes operar eficazmente en entornos complejos y variados, haciendo uso de la capacidad de generalización de las redes neuronales para manejar situaciones que los enfoques tabulares no pueden resolver eficientemente.

## Entrenamiento

El proceso de entrenamiento del modelo DQN en el entorno de Blackjack se estructura alrededor de una red neuronal profunda, diseñada para aprender y optimizar la función Q a través de interacciones continuas con el entorno. Este método permite al agente desarrollar una estrategia eficaz para maximizar las recompensas a largo plazo basándose en las predicciones de la red.

Específicamente, en nuestro caso, el modelo DQN implementado consta de varias capas densas: la primera capa tiene 128 neuronas, seguida por capas intermedias de 64 y 32 neuronas, todas utilizando la función de activación 'relu'. La capa de salida corresponde al número de acciones posibles en el entorno de Blackjack, proporcionando así una estimación directa del valor Q para cada acción posible dada la entrada del estado actual. Para llevar a cabo el entrenamiento se ha seleccionado un factor de descuento γ = 0.95, y un tamaño de batch de 32. Mencionar también que, para equilibrar la exploración y la explotación, se emplea una política -greedy, que inicialmente favorece la exploración aleatoria de acciones, pero, con el tiempo y a medida que el modelo aprende más sobre el entorno, incrementa progresivamente la proporción de decisiones tomadas desde las predicciones de la red.

El modelo es optimizado usando el optimizador Adam con una tasa de aprendizaje de 0.01 y se entrena a través de 600 episodios. Posteriormente, el modelo entrenado se guarda para su posterior evaluación y posible uso en el archivo *‘src/models/dqn\_model.pkl’.* Así mismo, el código empleado para la implementación de este algoritmo se encuentra en el archivo *‘DQN.ipynb’*

## Resultados

Como en todos los casos anteriores, tras entrenar al agente, evaluamos su desempeño en términos de victorias, pérdidas, empates y pasos medios realizados en un total de 1000 episodios de prueba.

Gráfico, Gráfico de barras

Descripción generada automáticamente

Figura 6. Estadísticas resultado del desempeño del Agente DQN durante el proceso de evaluación.

En la evaluación del modelo DQN representada en la Figura 6, muestra resultados positivos en comparación con los métodos anteriores. Logró una tasa de victorias del 41.8%, lo que representa una mejora sustancial frente al 27% del agente aleatorio, y también frente al 37.1% del agente Q-Learning. Sin embargo, la tasa de pérdidas sigue siendo elevada, alcanzando 48.4%, mejorando las pérdidas del agente aleatorio (70%) y las del agente Q-Learning las cuales llegaban a ser de 53.2%. Esto indica que el modelo incurre en decisiones efectivas que afectan positivamente el rendimiento general.

Finalmente, la media de pasos por episodio fue de 1.55, mayor que los 1.35 del Agente Aleatorio y menor que los 1.69 del Q-Learning, indicando que el DQN está tomando decisiones más informadas y efectivas que el agente aleatorio y el Q-Learning.

Estos resultados evidencian cómo la implementación del DQN resulta en la mejora de la tasa de victoria y en la reducción de las pérdidas en comparación con un enfoque aleatorio. Esto evidencia el hecho de que al trabajar con entornos discretos que conllevan procesos estocásticos con no demasiadas acciones y estados posibles, como es el caso de Blackjack, el hecho de aplicar algoritmos complejos tales como redes neuronales pueden no resultar en una mejora que merezca la pena frente a la complejidad introducida.

# **Dueling Double DQN y Dueling DQN**

## Introducción

Los algoritmos Dueling Double DQN y Dueling DQN contienen un cambio respecto a las Double DQN y DQN. Dividir la predicción de estos algoritmos en dos, en vez de predecir directamente los Q-valores de cada acción, se predicen dos partes y luego se combinan para obtener los Q-valores.

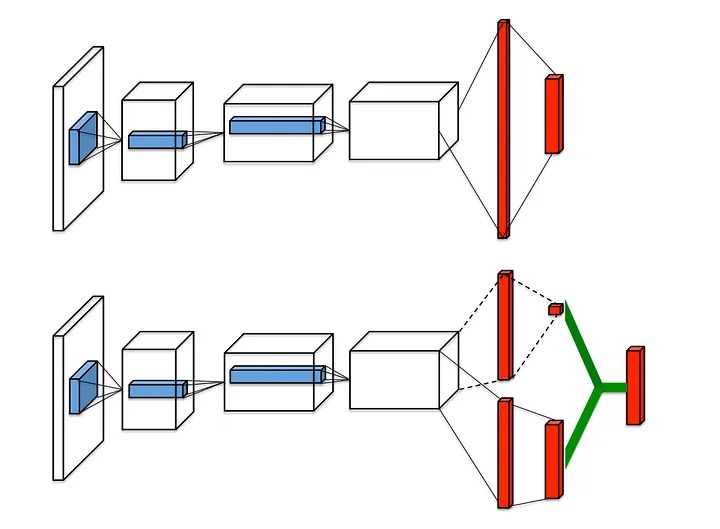
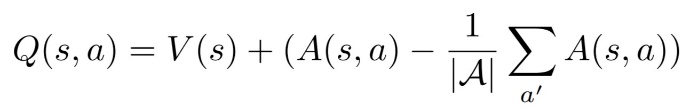


Figura 7. Esquemática comparativa entre arquitectura DQN vs. Dueling DQN.

Se sabe que Q-valor asociado a una acción y un estado se puede descomponer en dos sumandos: la ventaja de dicha acción y el valor de dicho estado . Basándose en ello, lo que cambia en una Dueling DQN frente a una DQN es que las últimas capas de la red se dividen en dos: una parte que predice el valor del estado y otra que predice la ventaja de cada una de las acciones. Una vez predichos ambas partes, estas se unen para obtener el Q-valor de cada estado usando la siguiente fórmula Ecuación 2.

Ecuación 2. Fórmula del cálculo del Q-valor en Dueling DQN.



La arquitectura de las redes consiste en una primera capa densa que recibe 3 inputs y devuelve 32 outputs, una capa densa de 32 a 32 para la ventaja y otra igual para el valor de estado y una capa densa de 32 a 2 para la ventaja y otra de 32 a 1 para el valor de estado. Por último, una vez se tienen ambos outputs (ventaja y valor de estado) se combinan usando la fórmula previamente mencionada. Entre cada una de las capas se aplica una función relu para modelar no linealidades.

El caso de Dueling DQN, es una sola con la anterior arquitectura que realiza las predicciones. Por otra parte, el caso de Dueling Double DQN, consiste en dos redes con la misma arquitectura (la anteriormente descrita): la principal y la target. El objetivo de añadir una segunda red es no sobreestimar los Q-valores.

## Entrenamiento

Se han entrenado dos modelos, el primero Dueling Double DQN y el segundo Dueling DQN, cada uno con 100.000 episodios y los mismos parámetros.

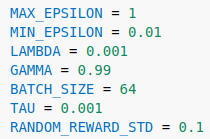


Figura 8. Parámetros empleados en el entrenamiento de Dueling DQN.

El entrenamiento del Dueling DQN es equivalente al de una DQN (Sección 3), en cambio, en el caso del Dueling Double DQN, al haber dos redes, cambia ligeramente.

Durante el entrenamiento del Dueling Double DQN, la red principal predice la mejor acción para un estado de entrada y la red target se utiliza para evaluar el error y actualizar los pesos de la principal. En cada iteración del entrenamiento, la red target se actualiza como combinación lineal de los pesos de la red principal y los actuales de la red target ponderadas por el valor del parámetro .

Ecuación 3. Actualización de los pesos de la red target en Dueling DQN.



## Resultados

Una vez entrenados ambos algoritmos, se han evaluado en un total de 1.000 episodios, obteniendo los siguientes resultados. En el caso de Dueling DQN, un 57% de las partidas han sido perdidas, un 4.5% empatadas y un 38.5% ganadas.

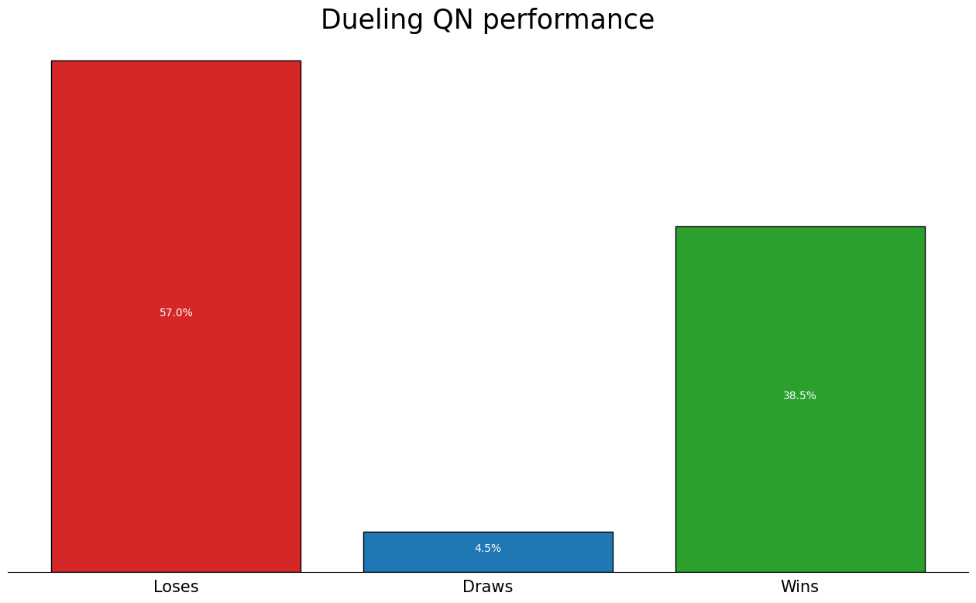


Figura 9. Estadísticas resultado del desempeño del Agente Dueling DQN durante el proceso de evaluación.

En el caso de Dueling Double DQN, un 55.9% de las partidas han sido perdidas, un 5.7% empatadas y un 38.4% ganadas.

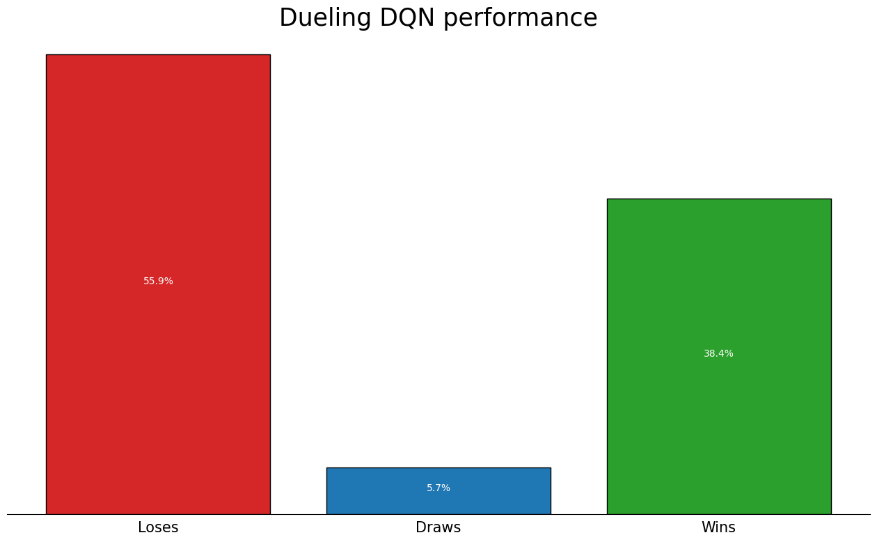


Figura 10. Estadísticas resultado del desempeño del Agente Double Dueling DQN durante el proceso de evaluación.

Teniendo en cuenta estos resultados, se puede ver que la diferencia entre ambos algoritmos no es determinante. Las ligeras diferencias entre ambos modelos se pueden deber perfectamente al hecho de solo haber usado 1.000 episodios para la evaluación. Ambos modelos tienen una actuación similar.

Además de estos resultados, también se ha calculado la media de pasos realizados en cada episodio, dando en ambos casos 1. Por lo tanto, como cada episodio consta de mínimo un solo paso, podemos concluir que todos los episodios han acabado con la primera decisión.

# **Advantage Actor – Critic (A2C)**

## Introducción

El algoritmo Advantage Actor-Critic (A2C) representa una innovación significativa en el ámbito del aprendizaje por refuerzo, fusionando elementos de aprendizaje profundo con la ventaja del Actor-Critic en la estimación de políticas y valores. A2C es una mejora del clásico algoritmo Actor-Critic, que combina la eficiencia computacional del Actor (encargado de proponer acciones) y el Critic (encargado de evaluar estas acciones), con el concepto de ventaja (advantage), que mide la mejora de tomar una acción en comparación con otras posibles acciones en un estado dado. Esto permite a A2C aprender directamente tanto la política (Actor) como la función de valor (Critic), sin necesidad de un modelo interno del entorno, lo que lo hace altamente escalable y adaptable a una amplia gama de problemas.

La arquitectura A2C utiliza dos redes neuronales: una para el Actor (policy-based) y otra para el Critic (value-based). El actor toma como entrada el estado del entorno y produce una distribución de probabilidad sobre las posibles acciones, mientras que el critic estima el valor del estado actual. Esta separación de funciones permite que A2C aprenda simultáneamente tanto las políticas óptimas como los valores de estado, lo que facilita una convergencia más rápida y estable durante el entrenamiento.

Una característica destacada de A2C es su capacidad para actualizar los parámetros de la red de forma asincrónica y sin memoria, lo que significa que los diferentes hilos de ejecución pueden actualizar los pesos de la red de manera independiente y eficiente. Esto promueve un mejor uso de los recursos computacionales y acelera el proceso de aprendizaje al permitir una exploración más diversificada del espacio de políticas.

Por lo tanto, A2C representa una poderosa herramienta en el campo del aprendizaje por refuerzo, ofreciendo una combinación única de eficiencia computacional, estabilidad y escalabilidad que lo hace adecuado para una amplia variedad de aplicaciones y entornos de aprendizaje por refuerzo.

## Entrenamiento

El proceso de entrenamiento del modelo A2C en el entorno de Blackjack se basa en la interacción continua entre el agente y el entorno, utilizando dos redes neuronales distintas: una para el actor y otra para el critic.

En nuestro caso, la arquitectura del modelo A2C implementado consta de dos redes neuronales completamente conectadas. La red del actor tiene dos capas ocultas de 32 neuronas con función de activación 'relu' para introducir no linealidades en la red, seguida por una capa de salida con activación 'softmax', que genera la distribución de probabilidad sobre las acciones posibles en el entorno de Blackjack.

Por otro lado, la red del critic también tiene otras dos capas ocultas de 32 neuronas con función de activación 'relu'. No obstante, a diferencia de la red del Actor, la capa de salida en este caso consta de una sola neurona, estimando así el valor del estado actual.

Texto

Descripción generada automáticamente

Figura 11. Arquitectura de las 2 redes neuronales en A2C

Durante el entrenamiento, se utilizan varios hiperparámetros importantes:

* El factor de descuento γ (gamma) se fija en 0.90.
* Se utiliza el optimizador Adam (el cual combina el descenso de gradiente estocástico SGD, y el método de adaptación de momentos) con una tasa de aprendizaje de 0.001 para actualizar los parámetros de ambas redes neuronales.

Como se mencionaba anteriormente, dentro del entrenamiento diferenciamos 2 partes en cada jugada:

1. **La primera parte, se corresponde con el Actor:**

El actor recibe el estado, y se predicen las probabilidades de seleccionar cada una de las acciones posibles para ese estado.

Se selecciona aleatoriamente una acción (atendiendo a las probabilidades de cada una de ellas), la cual se ejecuta y nos devuelve un nuevo estado, una recompensa e información sobre si se ha terminado la partida.

Finalmente, almacenamos esto recién obtenido para utilizarlo en la actualización de los pesos de la red.

1. **La segunda parte, se corresponde con el Crítico:**

En esta segunda parte, vamos a proceder a evaluar las acciones implementadas, y en base a esto, actualizaremos los pesos de las redes (se realiza cada n partidas, en nuestro caso n = 5).

Así pues, cada n partidas, convertimos todo lo almacenado a tensores para obtener los gradientes de la "loss" en función de los parámetros del modelo. Entonces se ejecuta el modelo (tanto el actor como el crítico) para obtener las probabilidades de acción y los valores Q del estado actual (q\_value1) y siguiente (q\_value2).

Se calcula la ventaja (advantage) como la diferencia entre el valor objetivo y el valor predicho.

Se calcula la pérdida total como la combinación de la pérdida de política (logaritmo de las probabilidades de acción ponderado por la ventaja) y la pérdida de valor. Se calculan los gradientes de la pérdida con respecto a los parámetros del modelo utilizando la cinta de gradiente. Y se utilizan los gradientes para actualizar los pesos del modelo utilizando el optimizador Adam.

Así vamos actualizando los pesos de las 2 redes y con el paso de las iteraciones, cada uno va mejorando su rendimiento al mismo tiempo, pero de manera independiente.

El modelo se entrena a lo largo de 1000 episodios, lo que permite al agente aprender y refinar sus políticas y valores a través de la retroalimentación del entorno. Una vez completado el entrenamiento, el modelo entrenado se guarda para su posterior evaluación y posible despliegue en el archivo ‘src/models/A2C.keras. El código utilizado para la implementación de este algoritmo se encuentra en el archivo ‘A2C.ipynb’.

## Resultados

Al igual que con los algoritmos anteriores, tras el entrenamiento, evaluamos el desempeño del modelo en términos de tasa de victorias y pasos realizados en un total de 1000 episodios de prueba.

Gráfico, Gráfico de barras, Gráfico en cascada

Descripción generada automáticamente

Figura 12. Estadísticas resultado del desempeño del algoritmo A2C durante el proceso de evaluación.

En la evaluación del algoritmo A2C representada en la Figura 12, se muestra la tasa de victorias en 1000 jugadas de prueba. Se obtienen resultados positivos en comparación con métodos anteriores. Se ha logrado una tasa de victorias del 40.4%, lo que representa una mejora sustancial frente al 27% del agente aleatorio, y también frente al 37.1% del agente Q-Learning. No obstante, no ha resultado siendo el modelo con mejores resultados, sigue siendo inferior a la tasa de victorias de DQN (41%).

Estos hallazgos demuestran que la implementación del algoritmo A2C conlleva una mejora notable en la tasa de éxito y una disminución en las pérdidas, en contraste con un enfoque aleatorio o ciertos algoritmos como DDQN, pero que no llega a ser el mejor modelo pese a la complejidad adicional. Esto sugiere que, al estar ante un entorno discretos con procesos estocásticos y un número limitado de acciones y estados posibles, como el juego de Blackjack, la aplicación de algoritmos más complejos, como este A2C, puede no generar una mejora significativa que justifique la complejidad adicional.

# **Conclusiones del proyecto**

## Idoneidad de los algoritmos empleados

En este proyecto, se han ido evaluando varias técnicas de Aprendizaje por Refuerzo en términos de rendimiento e idoneidad para el entorno de Blackjack proporcionado por la librería Gym. Cada una de estas técnicas posee características particulares que podrían adaptarse mejor o peor al problema en cuestión. Entre las ventajas y desventajas de cada técnica evaluada destacamos:

**1**. **Q-Learning:** Este algoritmo es uno de los enfoques clásicos del Aprendizaje por Refuerzo, que utiliza una tabla para almacenar los valores de acción-estado (Q-Values). Aunque es simple y fácil de implementar, puede enfrentar desafíos en entornos con un gran número de estados o acciones debido a la necesidad de almacenar y actualizar una tabla de valores.

**2. Deep Q-Network (DQN):** DQN representa una evolución significativa al integrar técnicas de aprendizaje profundo con los principios del Q-Learning. Utiliza redes neuronales profundas para aproximar la función Q, lo que le permite manejar espacios de estado y acción de alta dimensionalidad. Sin embargo, puede enfrentar dificultades en la convergencia y estabilidad del entrenamiento debido a la complejidad introducida por las redes neuronales.

**3. Dueling DQN**: Se ha escogido también este algoritmo pues supone una mejora sobre el DQN al descomponer la estimación del valor Q en dos partes separadas: la función de valor estatal (cuán bueno es estar en un estado dado) y la función de ventaja (cuán ventajosa es una acción específica en comparación con otras acciones en el mismo estado). Esto permite una estimación más precisa de los valores Q.

A pesar de mejorar la estabilidad y eficiencia, puede todavía enfrentar problemas de convergencia similares a los del DQN, así como la necesidad de un mayor ajuste de hiperparámetros.

**4. Advantage Actor-Critic (A2C):** A2C combina la eficiencia computacional del actor-critic con la ventaja del aprendizaje profundo en la estimación de políticas y valores. Este enfoque puede ser más escalable y adaptable a una variedad de problemas debido a su capacidad para aprender directamente políticas y valores sin la necesidad de un modelo interno del entorno. Sin embargo, puede requerir un ajuste fino de hiperparámetros y enfrentar desafíos en la estabilidad del entrenamiento.

Elegimos estos algoritmos de entre todos los posibles, pues consideramos que todos ellos eran potenciales “ganadores” a la hora de competir rendimientos. Además, se incluyen métodos basados en valores, así como métodos que combinan tanto el enfoque de valor como el de política.

## Comparativa de métodos empleados y análisis de resultados

Vamos a basar la comparación de rendimientos en base a la tasa de victorias y a la tasa de pérdidas de cada uno de los algoritmos ejecutados.

|  |  |  |
| --- | --- | --- |
| **Algoritmo** | **Tasa de victorias (%)** | **Tasa de pérdidas (%)** |
| Agente aleatorio | 27 | 70 |
| Q-Learning | 37,1 | 53,2 |
| DQN | 41,8 | 48,4 |
| Dueling DQN | 38,5 | 57 |
| Dueling DDQN | 38,4 | 55,9 |
| A2C | 40,4 | 54,8 |

La siguiente tabla muestra las tasas obtenidas en cada uno de los 6 algoritmos del proyecto:

Tabla 1. Tasas de victoria y pérdida en cada uno de los algoritmos

## Conclusiones

Tras la exploración de diversas técnicas en el presente proyecto, abordando desde técnicas más sencillas basadas en valor hasta técnicas más complejas basadas en la política, podemos concluir que, para el presente caso de uso, al estar ante un entorno discreto con procesos estocásticos y un número limitado de acciones y estados posibles, como el juego de Blackjack, la aplicación de algoritmos más complejos, como el A2C o Dueling DQN, no generan una mejora significativa que justifique la complejidad adicional introducida por dichos modelos.

En este sentido, en la Tabla 1, se observa como el DQN es el algoritmo que mayor tasa de victorias logra. Así mismo, dicho algoritmo es el que obtiene un menor número de pérdidas en comparación con el resto de técnicas. Esto, hace reafirmar el hecho de que no a mayor complejidad del modelo, un mayor rendimiento es siempre obtenido, lo cual es reflejado en las métricas obtenidas en algoritmos más elaborados como A2C o Dueling DQN, donde no sólo el número de victorias disminuye, si no que el número de pérdidas aumenta. Sería necesario en estos casos, explorarlos más en profundidad mediante el entrenamiento de dichos modelos con un mayor número de iteraciones con el objetivo de explorar el espacio de acciones más en profundidad o probando distintos parámetros en las redes implicadas en las mencionadas técnicas.

Por ello, tras todo el análisis realizado se reafirma el hecho de que emplear técnicas de Reinforcement Learning mejora sustancialmente los resultados obtenidos por puro azar (Agente Aleatorio Tabla 1), siendo entre ellos el más adecuado el DQN.