Aplicación del algoritmo Q-Learning al juego de azar BlackJack

Kalafatic Emiliano, Sola Jerónimo
UNL – FICH – emiliano.kalafatic@gmail.com
UNL – FICH - cuchujero@hotmail.com

***Resumen* — Este documento presenta los resultados del análisis de un algoritmo de aprendizaje reforzado aplicado al juego de cartas "Black Jack". En el mismo un jugador debe vencer al contrincante con las cartas que cuenta en la mesa. Para alcanzar este objetivo se utilizará el algoritmo mencionado con el fin de lograr que el jugador (computador o agente) aprenda a tomar la mejor acción teniendo en cuenta los diferentes escenarios posibles que se presentaran en cada partida.**

Se obtendrán conclusiones empleando el método de entrenamiento reforzado "Q-learning", con el fin de establecer una relación y aprendizaje con los conceptos dados en la cátedra, implementando un simulador del juego y el algoritmo empleando el software Matlab (versión R2015a).

***Palabras clave* - Q-learning, agente, estados, acciones**

I. INTRODUCCIÓN

El juego de cartas "Black Jack" contiene una baraja o mazo que está formado por trece tipos de cartas distintas, en el cual cada uno posee cuatro veces cada una de estas cartas. Por lo tanto, posee cincuenta y dos cartas en total con un valor numérico definido.

En cada partida, los jugadores compiten teniendo en cuenta los puntos que tienen sumados con las cartas formadas en su juego. El jugador que tiene el valor más grande de puntos gana. Sin embargo, si un jugador supera la cantidad de veintiún puntos, ese jugador pierde; en el caso en que ambos jugadores tienen misma cantidad de puntos empatan.

En el comienzo de una partida se repartirán al azar dos cartas para el jugador y dos cartas para la banca o contrincante (una de estas dos últimas estará boca abajo, por lo cual no será visible). Luego, el jugador tendrá cuatro opciones o decisiones a disposición:

- Pedir: Adhiere una carta más de la baraja para sumarla a las que tiene en su juego incrementando sus puntos.

- Plantarse: El jugador no modificara sus puntos en juego. La banca revela sus cartas, debe quedarse obligatoriamente si posee una cantidad de puntos mayor a 16, en caso contrario debe sacar cartas de la baraja hasta que llegue a tener una cantidad de puntos mayor que 16.

- Doblar: El jugador dobla la apuesta y se le otorga una sola carta más para sumar a las demás que posee en juego. Sin embargo, luego de esa decisión se planta automáticamente.

- Separar o dividir: Esta decisión solamente se puede tomar cuando el jugador tiene únicamente en mesa dos cartas iguales (que equivalen al mismo valor de puntos). Se separan las cartas en dos juegos distintos, de esta manera se dobla la apuesta (se realiza una apuesta mínima distinta en cada uno de los juegos por separado). A cada juego separado se le sumará una sola para luego competir contra la banca como juegos independientes normales.

Fue implementado un simulador de Black Jack en el software Matlab. En este, fue empleado un total de cuatro mazos mezclados entre sí, en los cuales a medida que las cartas salen en juego se van descartando, y luego de diez partidas se reingresan las cartas que salieron en dentro de las barajas con fin de mezclarse nuevamente en las mismas. Fue determinado un total de $1000 como dinero inicial para el jugador, y establecido como monto fijo de apuesta $100.

Como limitaciones no será tenida en cuenta la posibilidad de otros jugadores compitiendo contra la banca, ni tampoco poder variar la cantidad de dinero dentro de las apuestas iniciales.

En esta simulación, fue aplicado el algoritmo de aprendizaje reforzado "Q-learning". Este se basa en un agente inteligente, el cual toma decisiones dentro de su entorno que pueden llevarlo a lograr su objetivo. Dentro de su contexto hay diversos estados (escenarios en su entorno), donde cada decisión lo llevará a un siguiente estado que lo acercará o alejará de su objetivo final, obteniendo por esto una recompensa establecida. De esta forma el agente aprenderá a tomar la mejor decisión para lograr su objetivo en base a recompensas.

Aplicados estos conceptos a nuestro simulador el agente se correspondería al jugador, los estados a diferentes variables del juego (entre ellas la sumatoria de las cartas), el objetivo a si gana la partida contra la banca, y las diferentes recompensas se definen como números fijos dependiendo de si gana o pierde la partida.

II. DESARROLLO CENTRAL

El algoritmo "Q-learning" está enfocado en que el agente aprenda interactuando con su entorno (en este caso el juego) mediante las decisiones posibles a tomar ya mencionadas. Cada situación o escenario del juego se definirá como un estado, los cuales son tenidos en cuenta con el fin de que el agente tome una acción (representada por las decisiones), la cual pretenda ser la mejor a ejecutar

teniendo en consideración el estado en ese instante (ver Fig. 1).

Los diferentes estados posibles serán definidos por una matriz (matriz de estados), en la cual cada columna contiene los siguientes elementos:

- Sumatoria de puntos del jugador (S_m)
- Cantidad de cartas en mesa del jugador (CJ_m)
- Si cartas son iguales (si puede o no separar) (I_m)
- Si dinero es mayor o igual a 200 (si puede o no doblar) (M_m)
- Si hay entre las cartas del jugador un as (As_m)
- La carta boca arriba de la banca (CC_m)

Estos datos obtenidos del juego (ya sea expresado como un número real o binario) representarán un estado del juego, y en cada fila de la matriz se representan la cantidad de estados posibles. Obteniendo de esta manera un total de 9602 estados (filas) dados por 6 datos posibles (columnas), generando la matriz de estados.

Esta matriz tiene una relación directa con las recompensas a establecer, por lo que vincularemos esta matriz a una nueva denominada "matriz de recompensas" (ver Fig. 2). En esta un numero de fila se encuentra vinculado con una fila completa de la matriz de estados (la cual representa ese escenario particular) y posee cuatro columnas (una por cada decisión posible a tomar). De esta forma la matriz de recompensas tiene un tamaño de 9602 (filas) por 4 (columnas), y cada uno de sus elementos posee el valor de la recompensa a tomar teniendo en cuenta un estado y la decisión a tomar (ver Fig. 2). Esta matriz determina el espacio de soluciones de nuestro problema.

La función del "Q-learning" mide que tan considerada o buena es la acción a tomar por el agente dependiendo los estados del contexto. Para la misma son tenidas en cuenta diversas variables, una de estas es una recompensa estimada, la cual fue definida según los siguientes criterios:

R = 50 → Si gana la partida con la apuesta minima
R = -50 → Si pierde la partida con la apuesta minima
R = 80 → Si gana la partida con el doble de la apuesta
R = -90 → Si pierde la partida con el doble de la apuesta
R = -300 → Decisión seleccionada no correspondiente

Estas recompensas son tenidas en cuenta a la hora de la actualización, para que mediante el aprendizaje pueda lograr obtener mejores resultados dadas las acciones escogidas en cada escenario del ambiente o contexto.

Los valores numéricos de la recompensa son introducidos en la ecuación del algoritmo "Q-learning" (Ver Ecu. 1) para su actualización ajustada. La misma le suma a la recompensa actual (Q) un factor α (tasa de aprendizaje) de la recompensa que se le pase como argumento más otro factor γ (tasa de descuento) de la recompensa del siguiente estado.

Una mejora que se utilizó fue crear dos estados finales de victoria y pérdida. Entonces, si el siguiente estado finalmente conduce a una victoria, se recompensará con un valor positivo mayor al resto, si por el contrario el siguiente estado conduce al estado pérdida, se castigará con una recompensa de alto valor negativo.

De esta manera programamos el aprendizaje del agente, el cual, adentrándonos en el algoritmo, consiste en:

1) Inicializar la matriz de recompensas en 0.
2) Inicializar la matriz de estados con todos los posibles casos a tener en cuenta.
3) Realizar la etapa de entrenamiento.
   Se ejecutarán muchas series de partidas, en las cuales el agente tomara decisiones basándose en una variable épsilon. Esta variable determinará con que probabilidad el agente tomará una acción, pudiendo ser de la matriz o al azar. La misma se inicializa en el valor 0.9 e ira decayendo linealmente conforme pasen las iteraciones. Mientras mayor sea, más probable es que la decisión del agente sea al azar.
4) Actualizar los valores de la matriz de recompensas en cada partida con la formula del "Q-learning".
5) Terminar el bucle luego de cierto criterio de error o una cantidad de iteraciones considerables.

Una vez terminado este proceso, la matriz contiene los suficientes valores (recompensas) para poder ser utilizada nuevamente, consiguiendo de esta manera que el agente ya tenga los conocimientos adquiridos sin necesidad de volver a aplicar el entrenamiento.

$$Q_n(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma(\max(Q_s(s',a')) - Q(s,a)] \quad (1)$$

- Nueva recompensa para el estado y acción actual
- Tasa de aprendizaje
- Recompensa a sumar para el estado y acción actual
- Recompesa actual
- Máxima recompensa del siguiente estado s' y de todas sus posibles acciones.
- Tasa de descuento

Ecu. 1: Ecuación de Q-learning

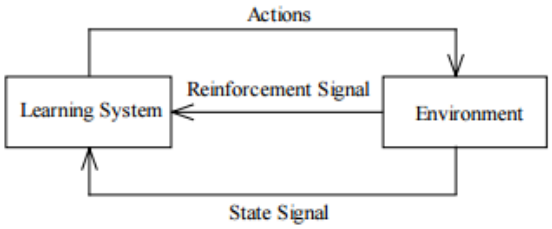

Fig. 1: Sistema de aprendizaje reforzado

Estados

$$\begin{bmatrix} S_1 & CJ_1 & I_1 & M_1 & As_1 & CC_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ S_m & CJ_m & I_m & M_m & As_m & CC_m \end{bmatrix}$$

Acciones

$$\begin{matrix} 1 & 2 & 3 & 4 \end{matrix}$$
$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ \vdots & \vdots & \vdots & \vdots \\ R_{m1} & R_{m2} & \cdots & R_{m4} \end{bmatrix}$$

Fig. 2: Matriz de estados y matriz de recompensas

III. RESULTADOS

El sistema fue entrenado aplicando el algoritmo descripto definiendo un épsilon de valor 0.9, con decaimiento de 0.1 cada 500 iteraciones, y aplicando un total de 5000 iteraciones de entrenamiento, donde cada iteración

corresponde a un juego (reproducir partidas hasta que el agente pierda todo el dinero o alcance un máximo de 300 partidas jugadas).

Los resultados obtenidos con una prueba de 50 iteraciones fueron exitosos comparados a decisiones realizadas al azar dentro del simulador (Ver Fig. 3, Fig.4 y Fig.5). Además, para obtener una visión más acorde y comprobar realmente los resultados se aplico la mediana dentro de las pruebas, debido a que, siendo sencilla de calcular, es una de las medidas más representativas para poder comparar nuestros resultados de manera prudente y considerada (Ver Fig.4 y Fig.5).

Fue realizada una segunda prueba con la misma cantidad de iteraciones con el fin de asegurar el funcionamiento del algoritmo y suponer su comparativa correcta, la misma fue expresada en un grafico de barras (Ver Fig. 6)

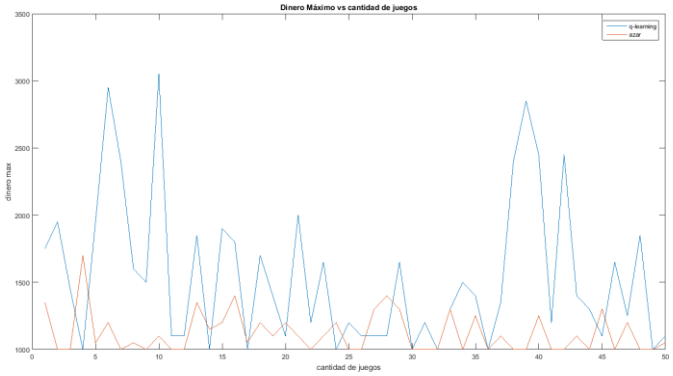

Fig. 3: Dinero máximo obtenido vs n° de juegos

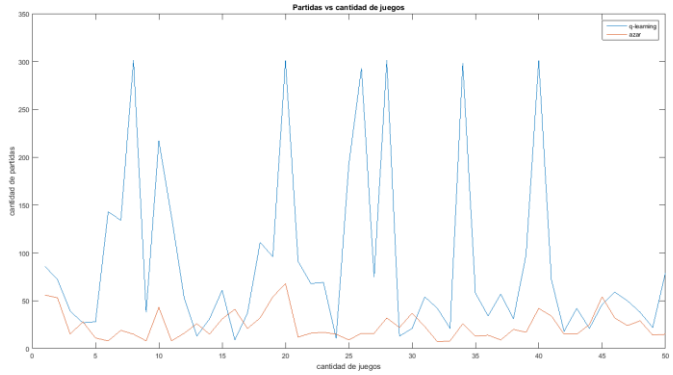

Fig. 4: Cantidad de partidas vs n° de juegos

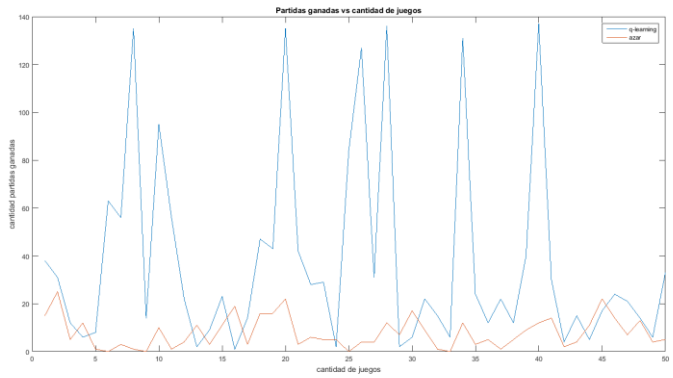

Fig. 5: Cantidad de partidas ganadas vs n° de juegos

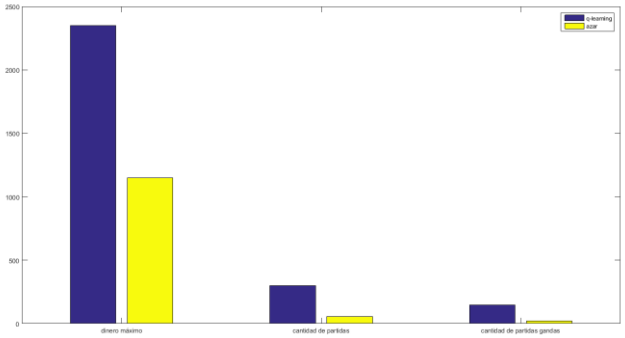

Fig. 6: Grafico de barras comparativo de datos reales del simulador

TABLA I

TABLA COMPARATIVA ENTRE EL ALGORITMO Y DECISIONES AL AZAR DE DATOS REALES TOMADOS DEL SIMULADOR EN LA PRIMER PRUEBA REALIZADA. LA MISMA CONTIENE LOS VALORES MÁXIMOS ALCANZADOS CORRESPONDIENTES A CADA VARIABLE REPRENSETADA.

	q-learning	azar
Dinero	3100	1700
Partidas	300	55
Partidas ganadas	147	20

IV. CONCLUSIONES

Apreciamos que luego de un tiempo considerable de entrenamiento, el agente adquiría aprendizaje y adoptaba estrategias para ganar en el simulador del juego (teniendo en cuenta el objetivo, el cual abarca incrementar el dinero disponible). En cuanto a la evolución del agente a lo largo de las partidas, se debe destacar que tuvo un resultado casi tres veces mejor comparativamente con decisiones al azar.

Luego de realizar este proyecto podemos considerar los resultados del experimento como positivos, logrando una implementación y un funcionamiento correctos del algoritmo, pudiendo ser aplicado a videojuegos o programas de otro ámbito.

Como limitación observamos que es necesario realizar un extenso entrenamiento con fin de obtener un aprendizaje óptimo a través de los elementos de la matriz, con lo cual conlleva mucho tiempo de cómputo almacenar la información en la misma debido a su gran tamaño. La recomendación es adaptar este algoritmo a redes neuronales para poder almacenar la información obtenida dentro de los pesos, conllevando así menos procesamiento de cómputo y logrando reducir el tiempo de entrenamiento.

REFERENCIAS

[1] Deepshikha Pandey, Punit Pandey, “Approximate Q-Learning: An Introduction”,2010.

[2] Farhang Sahba, “Deep Reinforcement Learning for Object Segmentation in Video Sequences”, 2016.

[3] David R. R. Smith, Thomas C. Walters, and Roy D. Patterson,” Implementation of Q Learning and Deep Q Network For Controlling a Self Balancing Robot Mode”, 2018.

[4] Melrose Roderick, James MacGlashan and Stefanie Tellex,” Implementing the Deep Q-Network”, 2017.

[5] Jianhui Wang,” Conjectural Variation-Based Bidding Strategies with Q-Learning in Electricity Markets”, 2009.

[6] Koichi Moriyama,” Learning Desirable Actions in Two-Player Two-Action Games”, 2009.