Universidad Nacional del Sur
Departameto de Cs e Ingeniería de Computación
Proyecto Final

Sistemas de trading y Deep Reinforcement Learning

Autor:

Leonardo Jose Caramello

Director:

Diego Martinez

Bahia Blanca, Buenos Aires

Agosto - 2017

Resumen

Vivimos en un mundo cada vez más digitalizado donde el acceso a la información es cada vez más fácil y abundante, un mundo que en los últimos años se ha transformado radicalmente y que sin duda lo seguirá haciendo en los años venideros. Los avances en machine learning, han permitido automatizar tareas, que hasta hace algunos años parecía imposible, ejemplos de esto son Google self-driving car, sistemas de recomendación como los usados por netflix o mercado libre, o Deep Mind.

Uno de los desafíos más interesantes en este área, es el desarrollo de sistemas de trading que permitan automatizar la comercialización de activos financieros, con el fin de administrar eficientemente un porfolio de inversiones. El siguiente trabajo esta inspirado en Deep Mind, un agente que combinando reinforcement learning y deep learning aprende a jugar juegos de atari.
En este papper buscamos investigar la aplicabilidad y efectividad de las estas técnicas, en el desarrollo de sistemas de trading.

Índice general

Índice de figuras

Nomenclatura

s	estado
a	acción
t	tiempo discreto
T	Instante final de la simulación o el episodio
S	Conjunto de estados
$A(s)$	conjunto de acciones validas en el estado s.
S_t	Estado en el que se encuentra el agente en el instante t.
A_t	Acción elegida por el agente en el instante t.
R_t	Reward obtenido por el agente en el instante t+1
π	política de decisión, regla de decisión.
$\pi(s)$	acción tomada en el estado s, siguiendo la política determinista π
$q_\pi(s, a)$	el valor de tomar la acción a, en el estado s, bajo la política de decisión π
$q_*(s, a)$	el valor de tomar la acción a, en el estado s, bajo la política de decisión π
γ	factor de descuento de rewards.
α	razón de aprendizaje
ε	probabilidad de elegir una acción aleatoria en una política de decisión greedy

1 Introducción

Este documento forma parte del proyecto final de carrera y se acompaña junto con el código del agente desarrollado, disponible también online. En este documento se pretende dar una presentación formal a los resultados de investigación obtenidos durante el desarrollo del proyecto, como así también una descripción del problema a resolver y de la solución adoptada, detallando cuales fueron cada una de las decisiones de diseño adoptadas y por que se adoptaron.

1.1 Alcance del proyecto

El proyecto se planteo como un trabajo de investigación que permitiera analizar cuales son las herramientas que brinda el machine learning para desarrollar un agente capaz de invertir en activos financieros(acciones, bonos, obligaciones negociables, etc), en particular, a través del uso de reinforcement learning.

El proyecto pretende ser una prueba de concepto, que permita abordar el desarrollo de sistemas de trading utilizando reinforcement learning. En particular, se buscara identificar cada uno de los componentes que plantea el framework de reinforcement learning, de forma tal, de tratar de encontrar una representación adecuada.
A su vez, durante el proceso de investigación se tratara de comprender la complejidad y las problemáticas propias del dominio del problema, en este caso, los mercados financieros y el trading en ellos, las cuales sera necesario comprender, para poder modelar una solución mas eficiente.
Queda fuera del alcance de este proyecto lo siguiente:

- Desarrollar un nuevo algoritmo de RL
- Desarrollar un agente capaz generar ganancias
- Desarrollar una comparación de diferentes implementacion del RL.
- Cubrir todas las peculiaridades de un mercado financiero.

1.2 Objetivos

A continuación se detallan los objetivos del proyecto

Objetivo general

Investigar la posible aplicabilidad de reinforcement learning en el desarrollo de sistemas de trading que permitan optimizar y automatizar la toma de decisiones de un potencial inversor.

Objetivos específicos

- Brindar una descripción general del funcionamiento de los mercados financieros.
- Brindar una descripción general de Reinforcement Learning.
- Brindar una especificación detallada de cómo modelar el problema de trading utilizando los elementos que propone el framework de RL.
- Desarrollar un entorno que permita realizar la simulación de un mercado financiero
- Desarrollar un agente inteligente capaz de percibir su entorno y tomar decisiones de compra o venta de un activo financiero

1.3 Metodología

En primer lugar se definirá el tópico central de la investigación, junto con los alcances y limitaciones del proyecto. En una segunda etapa, se hará una investigación sobre el funcionamiento de los mercados financieros, en particular, se buscará entender el funcionamiento del análisis técnico de los mercados y/o de los principales indicadores bursátiles, de forma tal de poder capturar estos conceptos y poder modelarlos en el desarrollo del agente. En tercer lugar se llevará a cabo una investigación de los diferentes algoritmos de RL y de cómo modelar el problema de trading utilizando los elementos que propone el framework. Por último, se proseguirá con el desarrollo del agente inteligente, de forma tal de poder integrar todo el conocimiento adquirido en las etapas previas.

1.4 Contenido

Por último, concluimos este capitulo introductorio con una breve descripción de cada uno de los capítulos siguientes

- Capitulo 2 - Mercados Financieros: En este capitulo se presenta el problema del trading, se dará una breve explicación del funcionamiento de

un mercado financiero y de como operan los inversores en el, abarcado conceptos como acciones, precio, análisis técnico vs análisis fundamental, drivers, indicadores bursátiles, tendencias, etc.

- Capitulo 3 - Reinforcement learning y deep learning: En este capitulo de introducirá brevemente el framework de reinforcement learning y cada uno de sus componentes, para luego esbozar la representación que se adoptara junto con un pseudo algoritmo que mostrara una primera aproximación a la solución elegida

- Capitulo 4 - Arquitectura y Diseño: Aquí se hará una descripción detallada de la arquitectura y el diseño del sistema implementado, se mostraran detalles y demás peculiaridades de implementacion, en particular parámetros de configuración, funcionalidades, etc. Se mostraran algunas las partes mas relevantes del código del agente.

- Capitulo 5 - Evaluación y desempeño: En esta sección analizaremos el desempeño y eficacia de la solución implementada, mostrando y analizando los diferentes resultados obtenidos durante la ejecución del agente.

- Capitulo 6 - Conclusiones y Recomendación: Como punto final, esta sección estará destinada a comentar diferentes conclusiones que pudimos establecer, junto con algunas recomendaciones y/o observaciones de posibles mejoras a futuro.

- Capitulo 7 - Bibliografía y Referencias: Listado de las diferentes referencias que formaron parte de la investigación.

2 Mercados Financieros

2.1 Introducción

En economía, un mercado financiero es un espacio (físico, virtual o ambos) en el que se realizan los intercambios de instrumentos financieros y se definen sus precios. Los mercados financieros están afectados por las fuerzas de oferta y demanda. La clave del éxito está en saber predecir el futuro y actuar en consecuencia. Quedarse largo en una posición (comprar) si se piensa que el mercado va a subir, o deshacer posiciones o quedarse corto si se piensa que el mercado va a bajar. Si se consigue hacer esto en forma reiterada se podrán obtener ganancias y con ello incrementar el capital inicial.

2.2 Análisis Técnico y Análisis Fundamental

Para intentar saber cómo va a estar un valor en el futuro, se distinguen tradicionalmente dos corrientes bien diferenciadas: los que siguen el análisis técnico y los que siguen el análisis fundamental. Los fundamentales se basan en que el valor de una acción esta dado por los beneficios futuros de la empresa. Ni más, ni menos. Lo que intentan es determinar cuáles serán esos beneficios futuros, y para ello tratan de conocer diferentes detalles de la empresa: noticias que les afecten, posibles movimientos societarios, estrategias, competidores, nuevos productos, etc. Toda la información micro económica tiene impacto en dichos beneficios futuros. Así como también la macro económica: cómo evoluciona el entorno general de la empresa, el entorno regulatorio, el entorno político, etc. Se trata, en definitiva, de analizar la mayor cantidad de información posible, y de convertir esa información en cuentas de resultados provisionales que se puedan descontar para hallar el valor actualizado de la acción. El análisis técnico, por el contrario, se basa en que el precio de la acción lo descuenta todo. Es decir, todos los factores relevantes a la inversión, cualquiera que ellos sean, pueden ser reducidos al nivel de precios de la acción y volumen transado. El precio de mercado representa el total conocimiento de los inversionistas respecto de cualquier activo dado en un momento particular. Además, refleja todas las noticias sobre el mercado así como la suma de conocimientos de los participantes en éste. Aquí se habla de tendencias alcistas o bajistas, de líneas de soporte (cotizaciones donde se cree que la acción dejará de bajar y tendrá un rebote"), líneas de resistencia (cotizaciones en las que el valor de

la acción se atascará y que le costará romper"), etc. La lógica nos dice que el análisis fundamental es el que tiene más sentido. Sin embargo, en la realidad esto no siempre es así. Lo cierto es que cuanta más gente crea en los análisis técnicos, más probabilidad tendrán de ser reales sus predicciones (ya que la gente actuará como si fueran reales, contribuyendo a su efectiva realización).

Este proyecto se basa fundamentalmente en las ideas del análisis técnico y las herramientas que este brinda, las cuales, serán los pilares fundamentales sobre los que el agente tomara sus decisiones. Para comprender un poco mas acerca de este y visualizar como es posible tomar decisiones acertadas solamente apoyándonos en el análisis técnico, debemos mirar mas en detalles, algunos aspecto de la teoría clásica financiera.

2.3 Teoría Clásica

Gran parte de la teoría financiera clásica parte del principio fundamental que los inversores son racionales y que los precios del mercado reflejan en todo momento y de manera instantánea el valor fundamental de los títulos. Este principio fundamental establece que la competencia entre los distintos participantes que intervienen en el mismo, conduce a una situación de equilibrio en la que el precio de mercado de un activo constituye una buena estimación de su precio teórico, es decir, que los precios que se negocian en el mercado reflejan toda la información existente y se ajustan total y rápidamente a los nuevos datos que puedan surgir. La consecuencia de este principio es que un inversor racional no puede hacer nada para "ganar" al mercado.

De acuerdo con este principio de racionalidad económica, lo que debe hacer un inversor es intentar maximizar su riqueza final. Para lograr esto, lo mejor que puede hacer este inversor racional es invertir en el mercado de manera diversificada de una forma igual a la del mercado y permanecer en esta misma cartera salvo por necesidades de liquidez o debido a variaciones de su situación actual o de cambio en sus necesidades futuras.

A pesar de la cantidad de libros de texto y de artículos que sostienen los principios anteriores sobre la forma en que deben comportarse los inversores, lo cierto es que la evidencia empírica nos dice que las cosas no suceden de la forma en la que deberían suceder según este principio,o por lo menos, no enteramente. Para poder explicar este fenómeno deberemos detenernos en algunos aspectos fundamentales en el proceso de decisión de los inversores.
Un claro ejemplo de esto podemos verlo si analizamos el proceso de decisión de venta, de acuerdo con la teoría clásica, los precios de cualquier activo si-

guen un movimiento aleatorio. Esto significa que la mejor predicción sobre el precio futuro es la que se tiene hoy. La consecuencia inmediata de esto es que no tiene ningún sentido vender, para a continuación, volver a comprar éste mismo activo u otro diferente. Dado que las expectativa de ganancia debido a la diferencia entre los precios venta y de compra sería nula. Solo tendríamos una pequeña pérdida debido al coste de la transacción. En otras palabras, el mercado no es predecible y, en consecuencia, no es posible obtener un beneficio realizando trading. Sin embargo, veamos algunos conceptos que contradice esto:

- *La creencia de los inversores en la reversión a la media*, es el principio según el cual existe un valor medio de cada acción al cual se acaba volviendo en algún momento. Así, si un valor tiene un precio que el inversor cree que está por debajo del que le corresponde (su valor "medio") , tarde o temprano, el precio de esta acción subirá hasta llegar a ese precio. Y, en consecuencia, recuperará las pérdidas que está teniendo en este momento. Lo mismo puede decirse cuando el valor esta por encima del valor medio. Así, si el inversor tiene una serie de valores que entiende que están infravalorados por el mercado, tiende a mantenerlos esperando que vuelvan a su valor "medio".

- *La aversión a la pérdida.* Si por alguna razón, tenemos alguna predicción confiable que nos dice que el precio de un activo va a bajar, lo racional es vender, independientemente de que con el activo a vender se obtenga un beneficio por su venta o no. La realidad nos muestra que esto no siempre es así, y que en general, los inversores tienden a no realizar las pérdidas, es decir, a no efectuar la venta real, en valores en los que pierden y, en cambio, vender antes de tiempo aquellos en lo que tienen ganancias. Una gran parte de los inversores venden valores ganadores y mantienen los perdedores.

- *El efecto disposición.* El comportamiento debido a la aversión a la pérdida es una parte de un comportamiento más general de los inversores, llamado efecto disposición según el cual, los inversores mantendrían demasiado tiempo activos en pérdidas y venderían demasiado pronto activos con ganancias. Esto se da debido a que los inversores son mucho más sensibles a las pérdidas que a las ganancias en el sentido de que las primeras influyen el doble que las segundas.

- *El efecto atención.* Los inversores tienden a invertir en aquellos valores que llaman mas su atención por la razón que sea, incluso aunque esta atención sea debida a noticias negativas, y como consecuencia, gran parte de los inversores diversifica mucho menos de lo que debería.

Estos argumentos y otros, nos indican que el inversor no siempre se comporta de manera esperada. Tal vez, el principal error de la teoría clásica sea partir de que los inversores son entes racionales y que no operan según sentimientos, euforia, miedo, avaricia o codicia.

2.4 Fundamentos del análisis técnico

Si bien es cierto que existen diversos argumentos en contra del análisis técnico, como por ejemplo:

- La profecía del auto cumplimiento
- El pasado no sirve para predecir el futuro
- El paseo aleatorio
- Mercados Eficientes

Lo interesante aquí es entender que el análisis técnico no trata de predecir el valor futuro del precio de un activo, sino mas bien de responder la siguiente pregunta:

¿cómo cree el conjunto de inversionistas que evolucionará el precio en el futuro?

Es importante notar, que lo que importa no es la evolución del precio, sino lo cree *la masa de inversores sobre la evolución del precio*, y ¿Por que esta pregunta es relevante?. Pues por que en el corto y mediano plazo, los precios se mueven mas por cuestiones psicológicas de los inversores que por variables financieras.

El análisis técnico nos dará las herramientas necesarias poder analizar desde un punto de vista estadístico, cual fue el comportamiento de la masa de inversores en el pasado, ante una situación similar, para luego poder tomar la decisión mas conveniente.

En otras palabras lo que se busca es reducir el nivel de incertidumbre que tiene el inversor a la hora de decidir si compra o vende un activo, haciendo uso de diferentes herramientas matemáticas como las medias móviles, lineas de tendencias o patrones con el objetivo de determinar cual es el comportamiento mas probable de la masa de inversores.

3 Reinforcement Learning y Deep Learning

3.1 Introducción

En esta sección buscaremos dar una descripción mas detallada, tanto del problema a resolver, como así también de como se aplicaron los conceptos de reinforcement learning y deep learning a las soluciones adoptas durante el desarrollo del mismo. Esta sección no pretende ser una guía o una introducción a reinforcement learning, ni a deep learning, sin embargo es necesario contar con conocimientos básicos de ellos para poder comprender los detalles de la solución propuesta. Dejamos en manos del lector la capacitación en estos a través de los diferentes recursos que pueden hallarse en linea. (ver Capitulo 7 - Bibliografía).

Recordemos que el objetivo es desarrollar un agente inteligente, que a partir de un capital inicial, sea capaz de realizar compras y ventas de un activo financiero. El agente no poseerá ningún conocimiento previo acerca del funcionamiento del mercado, ni de la empresa sobre la que opera, ni ningún otro tipo de información. Solo conocerá el precio actual de la acción, y su evolución durante los días previos, junto un conjunto de indicadores bursátiles.
El agente va a interactuar con un simulador de un mercado bursátil, esta interacción se llevará a cabo siguiendo una serie de acciones, observaciones y recompensas. Cada interacción será llevada a cabo en episodios que tendrán una duración de m días

En cada instante t, el agente tendrá que seleccionar una acción a_t de un conjunto válido de acciones A = $a_1, a_2, \ldots, a_n$. La acción será pasada al entorno, el cual modificará su estado interno y como respuesta a esta interacción el agente recibirá una recompensa r_{t+1}, la cual será calculada por el entorno. El entorno será estocástico, es decir, su comportamiento será no determinístico. Cada uno de los estados contendrá información relevante sobre el papel, como el precio, el volumen, y diferentes indicadores bursátiles, como medias móviles, medias móviles exponenciales, etc.

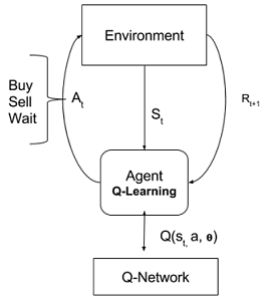

Figura 3.1: Reinforcement Learning Architecture Overview.

3.2 Configuración de estados

Ya sea de que se trate de un inversor experimentado o un agente inteligente, difícilmente sea posible tomar una decisión acertada sobre, la compra o venta de un activo, solamente mirando lo que sucede con el precio actual del activo, es necesario considerar una ventana de tiempo de tamaño n, para poder analizar la variación que ha tenido y así poder determinar la situación actual del activo, es decir, si se encuentra en una tendencia alcista o bajista, si se encuentra realizando una corrección de precio o no, si se encuentra en sobre compra o sobre venta, etc. Una manera eficiente de ver esto, es a través, de un gráfico de vela, donde cada vela, representa un periodo, el cual puede ser, un año, una semana, o un mes.

Figura 3.2: Gráfico de vela.

La representación que vamos a adoptar para cada uno de los estados que recibirá el agente, consiste en tomar esta idea de time frame y plasmarla en la estructura de los estados, es decir, el agente será capaz de ver una ventana de tiempo de tamaño n hacia atrás y de diferentes tipos de periodos.

Teniendo en cuenta esto, vamos a definir a cada estado S_t compuesto de 3 capas o layers, de la siguiente forma:

$$S_t = L_{dias}, L_{semanas}, L_{meses} \qquad donde \qquad L_i = p_1, p_2, ..., p_n$$

y en donde cada periodo p_i tendrá la siguiente estructura:

$$p_i = [capital,\ position,\ open,\ close,\ min,\ max,\ vol,\ ind_1,\ ind_2,\ ...,\ ind_n]$$

Los indicadores que se van a utilizar son los siguientes:

- Promedio del rango verdadero (ATR)
- Media móvil de 8 días (MA8)
- Media móvil exponencial de 20 días (EMA20)
- Media móvil exponencial de 50 días (EMA50)
- Media móvil exponencial de 200 días (EMA200)
- Indice relativo de fuerza (RSI)
- Indice de movimiento direccional (DMI)
- Promedio móvil convergencia/divergencia (MACD)
- Bandas de bollinger (BollingerBands)

3.3 Acciones

En cada instante t el agente podrá decidir comprar, vender o esperar, cada vez que decida comprar o vender, la cantidad de acciones operadas, se calculara en base a una estrategia de entrada y salida del activo pre configurada en el agente. El agente no tendrá limitación alguna en cuanto a la cantidad de acciones a comprar o vender. Si bien, en la realidad esto no necesariamente seria posible dado que para que pueda comprar una cantidad n a un precio p, debe existir algún vendedor dispuesto a vender n acciones a un precio p cada una. Ademas cada operación de venta o compra, tendrá un costo asociado c, el cual sera descontado de su capital. El agente no podrá ejecutar la acción elegida si su capital no alcanza para cubrir el costo total de la operación.

3.4 Reward

El entorno deberá calcular, en cada instante t, la recompensa asociada a la acción elegida por el agente en el instante $t-1$. Recordemos que según lo

que nos plantea el framework de reinforcement learning, la recompensa sera el mecanismo con el cual deberemos ir induciendo el aprendizaje del agente. Una mala elección de este, podría terminar en un mal desempeño de nuestro algoritmo. Para ello, hemos elegido la siguiente estrategia que vamos a denominar Winnings Over Loosings. Bajo esta estrategia, el entorno ira calculando el factor de ganancias R que va teniendo el agente. Dicho factor estará dado por suma total de ganancias durante los últimos 14 días, sobre la suma total de perdidas de los últimos 14 días.

3.5 El Agente

Dado que el objetivo principal de nuestro agente es seleccionar acciones que maximicen su recompensa futura, es decir, maximizar la cantidad de trades con ganancias, para esto vamos a asumir que las recompensa futuras están descontadas por un factor γ, por cada instante de tiempo t, es decir, vamos a valorizar los rewards más cercanos en el tiempo. Definimos así el, future discounted reward:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t \tag{3.1}$$

Donde T es el instante en el agente termina de invertir(o bien por qué perdió todo su capital inicial o bien porque se acabaron los datos de simulación). También vamos a definir una función estado-acción óptima, $Q^*(s_t, a_t)$, como la máxima recompensa esperada que podemos alcanzar estando en el estado s_t, y seleccionando la acción a_t, y luego continuando con una estrategia π

$$Q^*(s, a) = max_\pi \, \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{3.2}$$

Esta función tiene una propiedad importante, conocida como ecuación de Bellman, que intuitivamente se basa en lo siguiente: Si conociéramos el valor óptimo de $Q^*(s', a')$ para el próximo estado s' y para cada posible acción posible a', entonces la estrategia óptima de selección de una acción para el estado actual s que maximice la recompensa esperada está dada por

$$Q^*(s, a) = max_\pi \, \mathbb{E}_{s' \sim_\varepsilon}[r + \gamma \; max_{a'} Q^*(s', a') \mid s, a] \tag{3.3}$$

La idea general de nuestro algoritmo de aprendizaje va a ser estimar esta función Q^* usando la ecuación de bellman en forma iterativa, esto es:

$$Q^*_{i+1}(s,a) = \mathbb{E}\left[r + \gamma\ max_{a'} Q^*_i(s',a') \mid s,a\right] \tag{3.4}$$

donde $i \to \infty$ y $Q_i \to Q^*$

Esta es la idea detrás del algoritmo de Q-learning que implementará el agente, la cual en principio, parecía suficiente para lograr que el agente aprenda a invertir, sin embargo en la práctica, veremos que en general los estados del mercado no van a repetirse en forma idéntica y que ademas la cantidad de estados posibles hará que sea prácticamente inviable lograr la convergencia hacia Q^*.

3.6 Q-Network

Para poder solucionar este inconveniente, vamos a utilizar una función de aproximación, cuyo objetivo será obtener una generalización de los estados, lo cual permitirá en teoría, lograr más rápidamente la convergencia a Q^*.
Para implementar esta función utilizaremos un red neuronal con con una matriz de pesos θ, inicializada aleatoriamente:

$$Q(s,a,\theta) \approx Q^*(s,a) \tag{3.5}$$

a la cual denominaremos como Q-Network. Dicha Q-network será entrenada optimizando una función de pérdida L y usando el algoritmo del gradiente descendente.

$$L_i(\theta) = \mathbb{E}_{s,a \approx p(.)}[(y_i - Q(s,a,\theta_i)^2)] \tag{3.6}$$

$$donde \quad y_i = \mathbb{E}_{s \approx \varepsilon}[r + \gamma \quad max_{a'} Q(s',a',\theta_{i-1})|s,a] \tag{3.7}$$

Ademas se usara experience replay para entrenar la Q-Network, es decir, los datos de ejemplos serán generados a partir de la propia experiencia que vaya adquiriendo el agente. Para lograr esto, en cada instante de tiempo t, el agente

deberá guardar en un replay memory D, cada experiencia $\epsilon = (s_t, a_t, r_t, s_{t+1})$, observada. Transcurridos n días el agente deberá ejecutar la fase de entrenamiento de su Q-Network, para esto se generara un mini batch de experiencias pasadas, obtenidas de su replay memory y con ella se procederá a realizar una última fase actualización de la función de estimación de $Q(s, a, \theta_i)$, usando la regla de actualización de Q-learning y la nueva matriz de pesos θ_{i+1}.

3.7 Algoritmo de aprendizaje

A continuación detallamos el pseudo algoritmo de nuestro agente.

Algorithm 1: Q-Learning: Algoritmo general del funcionamiento del agente

```
1  D ← new memory_replay(n);
2  θ ← random();
3  Q*(s, a) ← Q_Network(s, a, θ);
4  s_{t−1} ← null ;
5  a_{t−1} ← null ;
6  while (episode ← generate_episode()) != null do
7      for s_t ∈ episode do
8          if (random(0, 1) <= ε_greedy) then
9              a_t ← random_a(A);
10         else
11             a_t ← max_a Q*(s_t, a);
12         end
13         r_t ← stock_exchange.execute(a_t);
14         D.save_experience(s_{t−1}, a_{t−1}, r_t, s_t);
15         s_{t−1} ← s_t;
16         a_{t−1} ← a_t;
17     end
18     mini_batch ← D.get_mini_batch(size);
19     training_samples ← [ ];
20     for experience ∈ mini_batch do
21         var s_t ← experience.s_t;
22         var a_t ← experience.a_t;
23         var s_{t+1} ← experience.s_{t+1};
24
25         estimate ← Q*(s_t, a_t);
26         expected ← Q*(s_t, a_t) + α (r_t + γ max_a Q*(s_{t+1}, a) − Q*(s_t, a));
27
28         sample ← new tupla(estimated, expected);
29         training_samples.add(sample);
30     end
31     Q_Network.train(training_samples);
32     Q*(s, a) ← Q_Network(s, a, θ);
33 end
```

4 Arquitectura y Diseño

4.1 Introducción

Para poder ejecutar el agente, se desarrollaron un conjunto de aplicaciones que permiten la ejecución del algoritmo presentado en el capitulo anterior en un entorno simulado, que representara nuestro mercado financiero. El código puede verse y descargarse de github, en el también podrá encontrarse las instrucciones para su ejecución. El sistema implementado que acompaña a este documento, esta compuesto por dos soluciones principales, una aplicación cliente, desarrollada en Typescript y Angular 4, y una aplicación de backend, encargada de emular el mercado financiero y ejecutar los agentes en el.

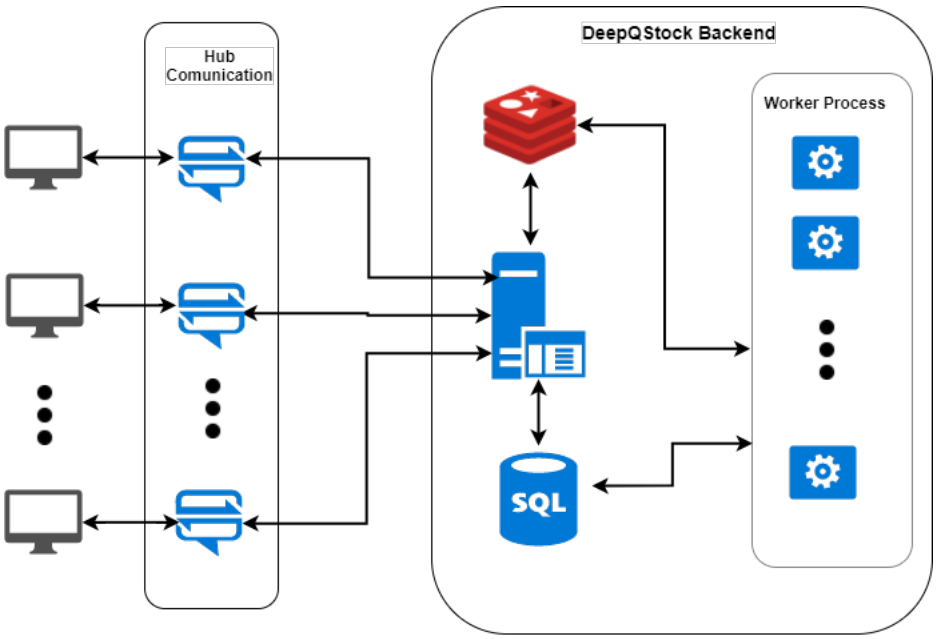

Figura 4.1: Arquitectura del sistema

Aplicación cliente

Esta aplicación permitirá crear agentes y sus entornos de simulación, permitiendo definir diferentes parámetros de configuración, tanto del algoritmo de

Q-learning como del entorno de simulación.

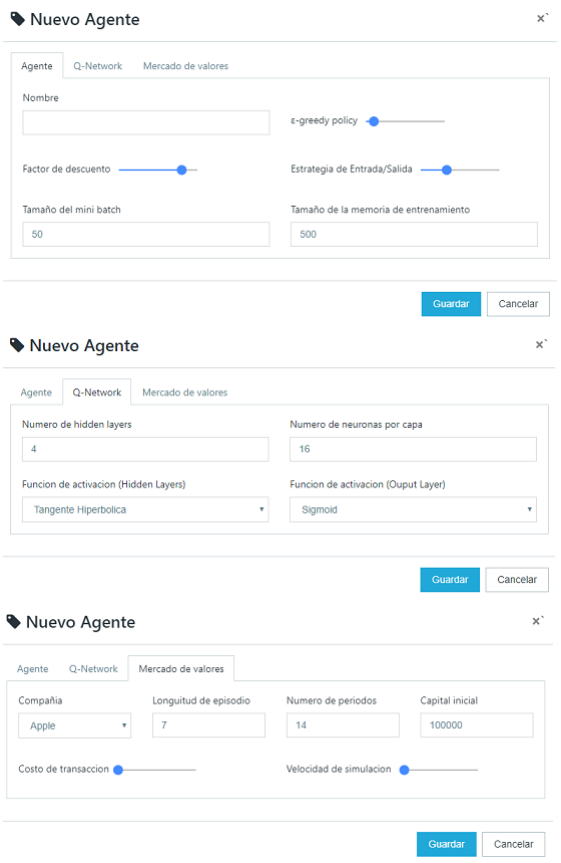

Figura 4.2: Creación de agentes

Cada vez que se cree un agente, se creará también un entorno de simulación con el cual el agente deberá interactuar. El mismo será responsable de la generación de los estados, así como también de los diferentes indicadores bursátiles y del calculo de cada rewards r_t. Se podrá inicial izar con diferentes opciones, como por ejemplo:

- Compañía a tradear
- Capital inicial
- El costo de transacción
- Longitud de episodio (días)

Una vez creado el entorno, se podrá acceder a una panel donde se podrá ver la evolución historia que ha tenido la compañía asociada al agente. Desde este panel podrá darse inicio a la simulación, pausarla o eliminar el agente y sus datos asociados. Una vez iniciada la simulación, podrá visualizarse en tiempo real, en todos los clientes que estuvieran viendo dicho agente, como va tomando las decisiones de compra o venta, junto con un resumen general del progreso, donde podrá verse entre otras cosas, el capital actual, la ganancia acumulada, cantidad de acciones que posee el agente, etc.

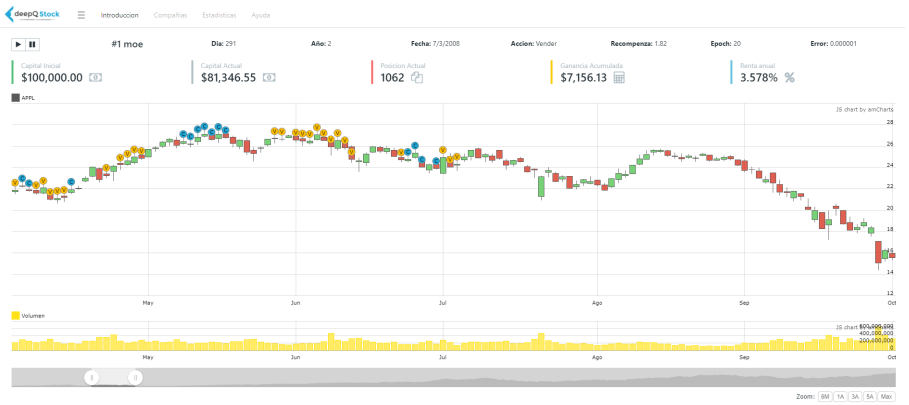

Figura 4.3: Evolución del agente

Desde la pantalla de creación también podrá setearse parámetros de configuración del agente, como por ejemplo:

- Policy ε-greedy, este parámetro representara la frecuencia con la que el agente elegiría en forma aleatoria la próxima acción.
- Factor de descuento, este parámetro permitirá configurar el nivel de valoración que se le dará a los rewards mas cercanos en el tiempo.
- Tamaño del mini-batch, este representa el tamaño de cada mini batch que el agente utilizara para entrenar a su Q-Network después de la finalizacion de cada episodio.
- Tamaño de la memoria de entrenamiento, este parámetro determinara la capacidad de recordar experiencias del agente.

Aplicación backend

La aplicación de backend fue implementada en .NET, consta de una servidor de backend, implementado con SignalR 2, SQL Server, Hangfire y Redis. Los clientes conectados con la aplicación de angular, se comunicaran con el

backend a través de un hub de comunicación implementado con signalR 2, este hub es el que permitirá la comunicación bi-direccional en tiempo real, es decir, desde aplicación cliente hacia el server backend, como desde el server hacia los clientes conectados con la aplicación angular. Este es un requerimiento necesario, dado que los agentes y los simuladores son ejecutados en threads independientes, para lograr esto se necesito integrar el backend con hangfire, un servidor de jobs. Cada vez que se inicie la ejecución de un agente, se solicitara a hangfire la creación de un nuevo job o worker, donde se ejecutara el agente y el entorno simulado. El agente comenzara a trader en el entorno configurado para el, e ira notificando a través de una cola de mensajes implementada con redis, todas las decisión y el progreso que esta teniendo al servicio de backend, una vez que el servicio de backend es notificado por redis, este utilizara el hub de comunicación para notificar a el clientes que estén subscritos a escuchar los mensajes de progreso de dicho agente.

4.2 Descripción del código

A continuación describiremos algunas secciones mas relevantes e interesantes del código implementado en el backend.

Clase StockExchange

Esta clase es la que representa nuestro entorno, en nuestro caso, un mercado financiero, ella sera la responsable de calcular los rewards y de ir actualizando los estados s_t. También sera responsable de la inicializacion de agente como así también del salvado del estado del agente cada vez q el proceso de ejecución sea pausado. El punto de entrada de esta clase sea el método Simulate(). Como se puede observar en la imagen, este método sera el responsable de la generación de los estados y de la simulación de cada episodio. Cada episodio estará compuesto por una cantidad n de días, en el que en cada día, el agente deberá tomar una decisión acerca de su situación, es decir, deberá decidir, si compra, vende o espera. Al finalizar cada episodio, el entorno invocara al método OnEpisodeComplete que como se vera mas adelante sera el responsable de entrenamiento de la red neuronal del agente.

```
/// <summary>
/// Simulate the enviroment and the agent integration
/// </summary>
protected void Simulate(IJobCancellationToken token)
{
    var action = ActionType.Wait;
    double reward = 0.0;
    int? currentYear = null;
    PreviousState = null;
    CurrentState = GenerateState();

    do
    {
        for (int i = 0; i < Parameters.EpisodeLength - 1; i++)
        {
            action = Agent.Decide(CurrentState, reward);
            reward = Execute(action, Agent.Parameters.InOutStrategy);
            DaysSimulated++;

            if (currentYear == null || currentYear != CurrentState.Today.Date.Year)
            {
                currentYear = CurrentState.Today.Date.Year;
                TotalOfYears++;
            }

            DayCompleted(action, reward);

            token.ThrowIfCancellationRequested();
            if (Parameters.SimulationVelocity > 0)
            {
                Thread.Sleep(Parameters.SimulationVelocity);
            }

            PreviousState = CurrentState;
            CurrentState = GenerateState();
            RemoveState(PreviousState);

            if (CurrentState == null)
            {
                break;
            }
        }

        Agent.OnEpisodeComplete();
        EpisodeSimulated++;

    } while (CurrentState != null);
}
```

Figura 4.4: Método Simulate

Clase DeepRLAgent

Esta clase es la que implementa al agente, en ella se implementara la policy de decisión usada por el agente, así como también la Q_Network y los métodos necesarios para entrenarla. Como podemos ver, el agente usara este método para decidir en base a una $policy_\pi$

```
/// <summary>
/// Decides the next action
/// </summary>
/// <param name="state">The st.</param>
/// <returns></returns>
public ActionType Decide(State state, double reward)
{
    if (Q == null)
    {
        InitializeQNetwork(state);
    }

    var previuosState = CurrentState;
    CurrentState = state;

    if (previuosState != null)
    {
        AddExperience(previuosState, CurrentAction, reward, CurrentState);
    }

    return PolicyPi();
}
```

Figura 4.5: Método Decide()

Se decidió optar por una policy $\varepsilon-greedy$, la cual consiste en elegir siempre la acción que maximiza el reward esperado, según la función estado-acción $Q^*(s_t, a_t)$ con una probabilidad ε de elegir aleatoriamente, lo cual le permitirá balancear el dilema de exploración vs explotación

```
/// <summary>
/// Select an a random action a' with probability e
/// otherwise select the action a' that max a' Q(s, a')
/// </summary>
/// <returns></returns>
private ActionType PolicyPi()
{
    var probability = RandomGenerator.NextDouble();
    var validActions = GetActions();
    var maxAction = Q[CurrentState].Where(i => validActions.Contains(i.Key)).MaxBy(i => i.Value).Key;

    if (probability <= Parameters.eGreedyProbability)
    {
        var randomActions = validActions.Where(a => a != maxAction).ToList();
        var randomIndex = RandomGenerator.Next(randomActions.Count);
        CurrentAction = randomActions[randomIndex];
    }
    else
    {
        CurrentAction = maxAction;
    }

    return CurrentAction;
}
```

Figura 4.6: Método PolicyPi()

Por ultimo, una vez finalizado un episodio se ejecutara el método UpdateUknowledge el cual sera el responsable de re-entrenar la red neuronal. Como

podemos ver en la imagen, el método generar un mini batch según los parámetros seteados durante la creación del agente, para luego generar los samples de entrenamiento en base a la estimación actual y al valor actualizado usando el algoritmo de Q-learning.

```
/// <summary>
/// Implement the Q Learning update rule using experience replay
/// </summary>
private void UpdateKnowledge()
{
    var mini_batch = GenerateMiniBatch();
    var trainingData = new List<Tuple<State, double[]>>();
    var actions = Enum.GetValues(typeof(ActionType)).Cast<ActionType>();

    foreach (var experience in mini_batch)
    {
        var targetValues = new Dictionary<ActionType, double>();
        var estimatedValues = Q[experience.From];

        foreach (var action in actions)
        {
            targetValues[action] = QUpdate(experience, action, estimatedValues[action]);
        }

        trainingData.Add(new Tuple<State, double[]>(experience.From, targetValues.Values.ToArray()));
    }

    Q.Train(trainingData);
    OnTrainingComplete?.Invoke(this, new OnTrainingCompleteArgs());
}
```

Figura 4.7: Método UpdateUknowledge()

```
/// <summary>
/// Calculates the q update.
/// </summary>
/// <param name="e">The e.</param>
/// <param name="action">The action.</param>
/// <param name="estimated">The estimated.</param>
/// <returns></returns>
private double QUpdate(Experience e, ActionType action, double estimated)
{

    var newEstimation = estimated;
    if (e.Action == action)
    {
        var maxAction = Q[e.To].MaxBy(i => i.Value).Value;
        newEstimation = estimated + Parameters.LearningRate * (e.Reward + (Parameters.DiscountFactor * maxAction) - estimated);
    }

    return Normalizers.Reward.Normalize(newEstimation);
}
```

Figura 4.8: Método QUpdate()

5 Evaluación y Desempeño

5.1 Introducción

Durante este capitulo nos dedicaremos a analizar el desempeño de nuestro agente. Para ello definiremos una serie de configuraciones, donde haremos variar los diferentes parámetros de ejecución y usaremos cada una de estas configuraciones sobre diferentes compañías para luego tratar de evaluar como dichos parámetros, afectan o no, en el desempeño del mismo. Lo interesante sera observar tanto los patrones de compra y venta ejecutados por el agente durante toda la simulación, como así también las ganancias obtenidas en cada configuración.

5.2 Configuraciones

La siguientes configuraciones serán ejecutas múltiples veces sobre diferente compañías. El objetivo principal de estas configuración es tratar de ver como se comporta el algortimo bajo diferentes seteos.

En principio hemos considerado 3 configuración que según creemos una sera la mas optima(moe), una intermedia(lenny) y por ultimo la que peor debería performar sera ralp, pues se acerca mucho a un agente que toma decisiones en forma aleatoria.

Ambas configuraciones se ejecutaran sobre dos compañías cuyos papeles tienen comportamientos completamente diferentes. Una apple donde claramente puede verse que el valor de las acciones generalmente se encuentra dentro de un tendencias alcista durante los 10 años y la otra es microsoft la

Cuadro 5.1: Configuraciones de los agentes

Nombre	moe	lenny	ralph
ε-greedy	0.1	0.3	1
learnig rate α	0.3	0.3	0.3
discount factor γ	0.8	0.5	0.1
mini batch	50	100	5
memory replay	500	1000	10
hidden layers	4	8	2
neurons per layer	16	32	4

cual posee un comportamiento mas estable, con tramos alcistas, bajistas y tramos donde el papel lateraliza. Las decisiones del agente se mostraran con burbujas de color azul para las compras, y burbujas de color amarillo para las ventas, a media que transcurran los días, se irán marcado cada vela, con una burbuja indicando la acción, para poder simplificar la visualización del gráfico, hemos decidido, no marcar con ningún marcador, los decisiones de esperar.

5.3 Resultados

moe

Estos son algunos de los resultados de la primera ejecución de moe sobre ambas compañías.

Figura 5.1: Resultados de moe sobre apple

Figura 5.2: Resultados de moe sobre microsoft

En este caso, el agente logro mejores resultados con apple, probablemente producto de la propia característica de un papel generalmente alcista. Sin embargo, observemos que sucede durante los primeros días de la simulación.

Figura 5.3: Primeros días de trading sobre apple

Claramente puede observarse que el agente se predispone mayoritariamente a realizar siempre la misma acción, en el primer caso decide vender, y en

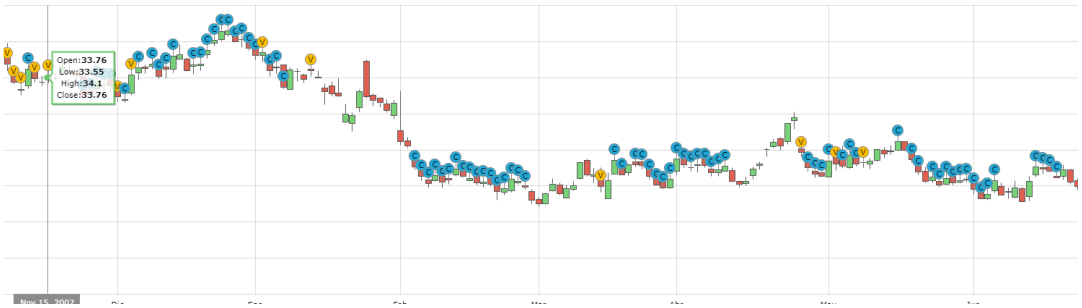

Figura 5.4: Primeros días de trading sobre microsoft

el segundo comprar. Probablemente esto este determinado por la elección de la primera acción, pues sera la que en principio mas se repita, y de la que mayor conocimiento tiene sobre su recompenza futura. A media que vaya experimentado en forma aleatoria otras acciones, gracias a la policy ε-greedy, deberiamos esperar ver un cambio en el patrón de decisiones. Veamos ahora que sucede con los últimos días de trading:

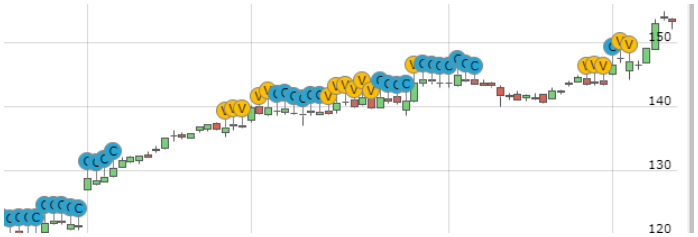

Figura 5.5: Últimos días de trading sobre apple

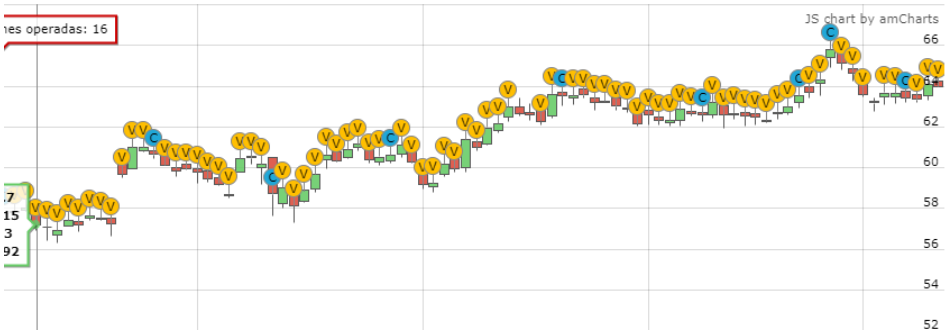

Figura 5.6: Últimos días de trading sobre microsoft

En ambos casos, puede observarse una mejoría, tal vez, en el caso de apple sea mas marcada, dado que hasta puede observarse que durante varios días,

el agente solo decide esperar, y especular esperando que el precio suba para vender luego de varios días gran parte de sus acciones a un precio mucho mayor al que las compro. Para el caso de microsoft, no es tan claro, pero sin embargo, puede verse que cada 4 o 5 días, el agente compra, para luego vender. Es importante notar aquí, que dado el esquema de entrada y salida que se decidió adoptar, los días en que compra, comprar mucho mas de lo que luego vende en los días siguientes.

Un aspecto interesante que surgió durante la investigación, fe preguntarnos que sucedería si volviéramos a ejecutar el agente ya entrado, es decir, reutilizando la matriz de pesos de su red neuronal. ¿Podría el agente mejorar su performance?.

Estos son los resultados que obtuvimos para moe

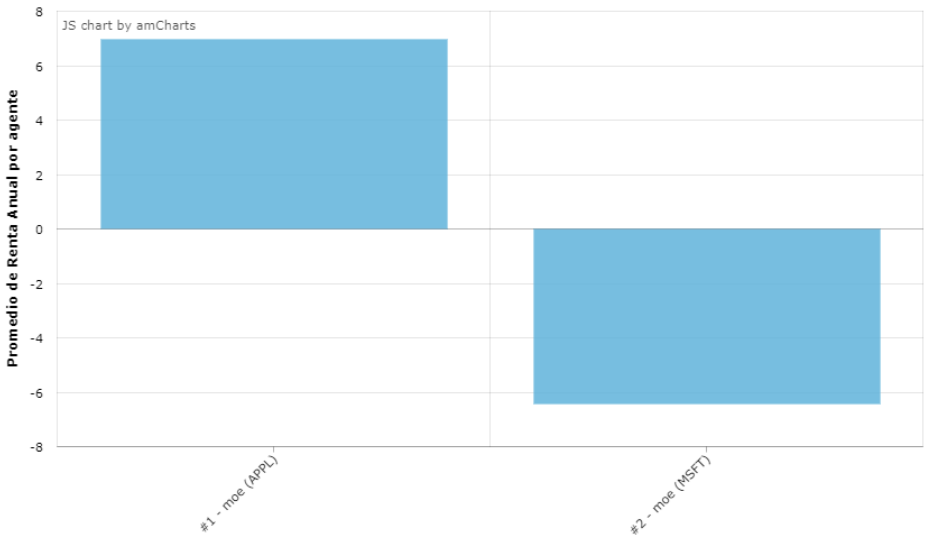

Figura 5.7: Promedio del rate de ganancias finales

Mirando los gráficos resultantes, no pareciera a ver una mejoría significativa, aunque si podemos afirmar que los agentes en general, repitieron su performance. Sin embargo pudimos observar una notoria modificación en el patrón de decisiones del agente:

Claramente podemos notar, que el agente toma decisiones de compra y/o venta mucho mas espaciadas en el tiempo, a comparación de las primeras ejecuciones, donde el agente tendía a sobre operar en el mercado, y terminaba realizado compras y ventas diariamente. No podemos concluir si esto significa una mejora o no, aunque si es posible asociar este tipo de patrón a un comportamiento mucho mas especulativo.

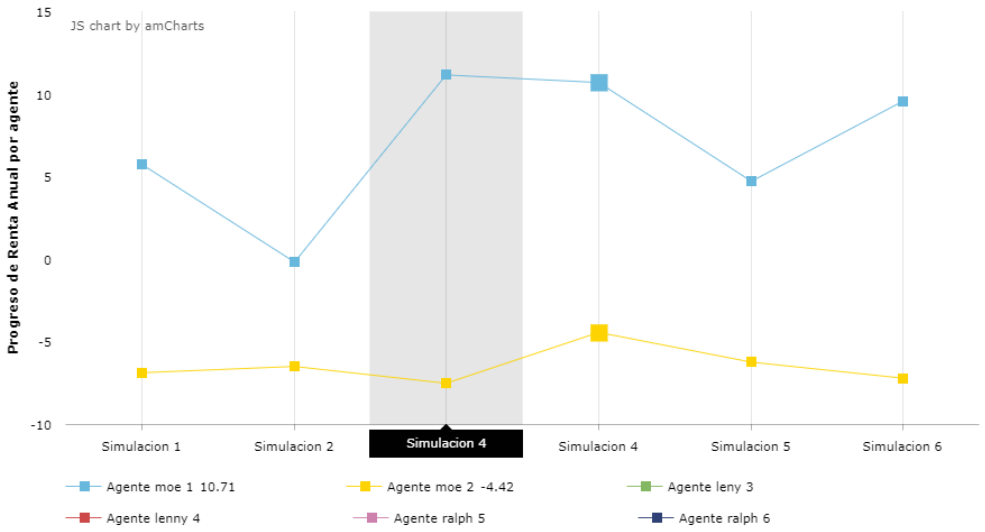

Figura 5.8: Evolución de ganancias

Figura 5.9: Resultados de moe luego de 6 ejecuciones

lenny

Como podíamos esperar, efectivamente lenny, se desempeño peor que moe. Sin embargo también podemos observar un cambio en el patrón de desicion a lo largo del tiempo.

ralph

Por ultimo, nos que analizar con ralp, y nuevamente, se dio lo que esperamos, si bien los resultados en términos del rate de ganancias, no difieren tanto, si podemos observa que el patrón de decisiones del agente se mantiene en el

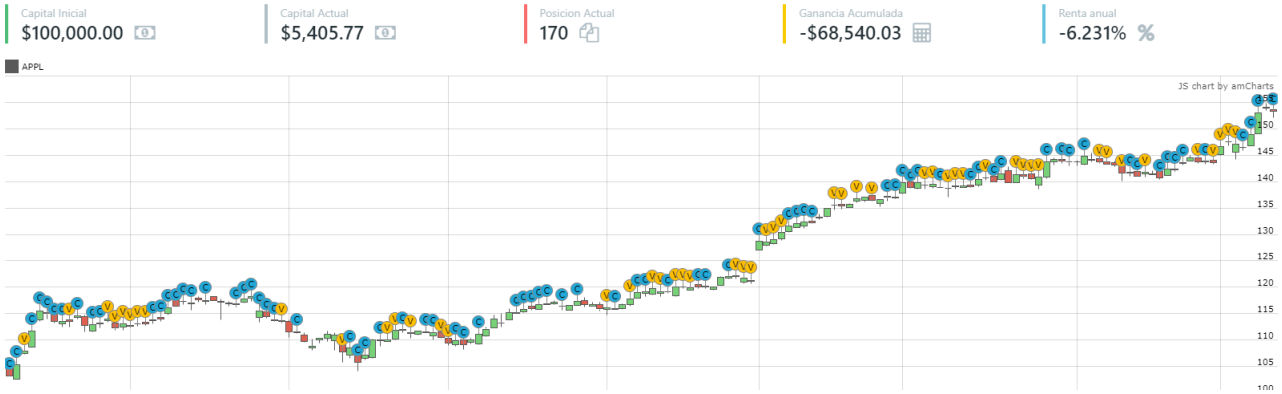

Figura 5.10: Ejecución de lenny sobre apple

Figura 5.11: Ejecución de lenny sobre microsoft

tiempo, dado que efectivamente siempre elige aleatoriamente. También pudimos observar un error muy grande durante el entrenamiento de la QNetwork, y que en general, dicho error no se reduce, provocando la no convergencia de la función Q

5.4 Gráficos

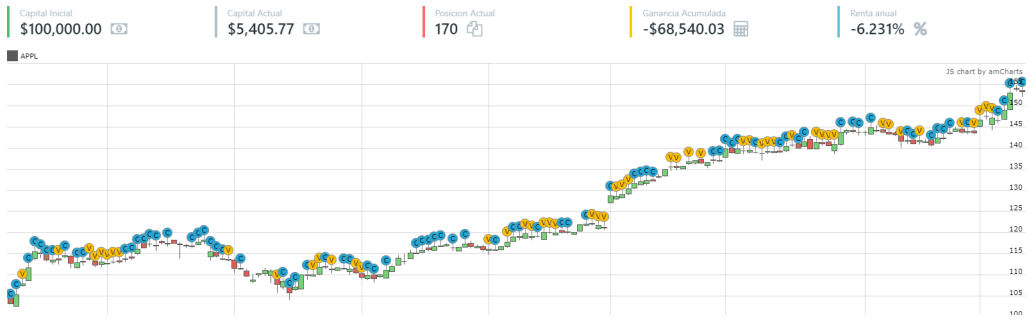

Figura 5.12: Ejecución de ralph sobre apple

Figura 5.13: Ejecución de ralph sobre microsoft

6 Recomendaciones finales

Recordemos que el objetivo inicial del proyecto era investigar la posible aplicabilidad y efectividad del uso de reinforcement learning y deep learning en el desarrollo de sistemas de trading. Luego de varios días de desarrollo y pruebas, y que mas allá de los resultados obtenidos, que tal vez estén lejos de igualar o sustituir a un inversor experimentado, el sistema demostró ser fiable en el sentido de es posible implementar un algoritmo que solamente evaluando la evolución del precio de un activo, aprenda a tomar decisiones sobre el.

Debemos reconocer aquí nuestra falta de experiencia y comprensión en el diseño y desarrollo de redes neuronales haya podido influenciar en la performance del agente. O que tal vez, el uso de una red neuronal no sea la mejor solución para estimar la función de valor $Q(s, a)$, en cualquier caso, creemos que el potencial de mejora en este aspecto es enorme, y junto con ello, la posibilidad de mejorar el desempeño de nuestro a agente.

Por otro lado, queremos resaltar, que no nos resulto sencillo determinar la efectividad de la performance del agente, es decir, no fue sencillo determinar cuando un trade es bueno o no, y en particular poder determinar si una decisión de compra o venta, es correcta o no. Tal vez también, producto de nuestra falta de conocimiento o entendimiento de los mercados. Sin embargo, también, en este sentido, creemos que hay mucha posibilidad de mejoras, y con ello poder modificar la señal de recompensa para tratar de mejorar el proceso de aprendizaje del agente.

En resumen, podemos decir que los resultados son prometedores, y que podemos pensar que en un futuro no muy lejano, este tipo de sistemas sean mas comunes ya sea como soporte a inversores o como sistemas completamente automatizados que permitan la administración de una cartera de inversiones.

7 Bibliografía

- Reinforcement Learning: An Introduction - Richard S. Sutton and Andrew G. Barto.
- Bekerly UC - Curse: Deep Reinforcement Learning
- Stanford - Curse: Convolutional Neural Networks for Visual Recognition
- Neural Networks and Deep Learning
- An Investigation into the Use of Reinforcement Learning Techniques within the Algorithmic Trading Domain
- Algorithm Trading using Q-Learning and Recurrent Reinforcement Learning
- Demystifying deep reinforcement learning
- Original DeepMind paper
- Stock Price prediction using reinforcement learning - Jae Won Lee - June 2001
- Reinforcement learning for optimized trade execution - Yuriy Nevmayvaka, Yi Feng, Micheal Kearns - June 2006
- Reinforcement learning and savings behavior - James J. Choi, David Laibson, Brigitte C. Madrian, Andrew Metrick - November 2009
- Continuous control with deep reinforcement learning, Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra - 2016