



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO OFICIAL EN CIENCIA DE DATOS
E INGENIERÍA DE COMPUTADORES

Trading financiero con técnicas de Aprendizaje por Refuerzo

Autor

Eduardo Martín Izquierdo

Director

José Manuel Benítez Sánchez



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Septiembre de 2019

Trading financiero con técnicas de Aprendizaje por Refuerzo

Eduardo Martín Izquierdo

Palabras clave: Aprendizaje por Refuerzo, Trading, Mercados Financieros, Q-Learning, Redes Neuronales, Agente de Bolsa

Resumen

El uso de *robots automáticos* en la actividad del *trading* de altas frecuencias es un tema muy prometedor y bastante investigado recientemente. Siguiendo las ideas de varias citas referenciadas en el documento, se propone un enfoque usando *Aprendizaje por Refuerzo* para la construcción de un agente de *trading* en altas frecuencias para intentar maximizar su beneficio económico interactuando con una serie temporal cualquiera de la Bolsa de Valores en un intervalo de tiempo determinado.

Se hace un *análisis general* sobre posibles soluciones a este problema usando *Aprendizaje por Refuerzo*, discutiendo las ventajas e inconvenientes de cada componente propuesto. Más adelante, se describe el *Marco teórico* referente al *Aprendizaje por Refuerzo* exponiendo las ventajas y desventajas de cada técnica en base al problema a resolver.

Una vez hecho estos análisis, se describe la *solución propuesta* para este problema, es decir, tomando el análisis general anterior se describen los componentes finales usados justificando por qué son los más apropiados. Estos componentes añaden unas mejoras propuestas en el trabajo para adaptarse mejor a cualquier serie.

Finalmente, estas mejoras quedan demostradas empíricamente llegando a la conclusión de que son capaces de soportar niveles bajos de comisión. Sin embargo, este sistema no es capaz de superar un nivel estándar de comisión de forma constante.

Financial trading with Reinforcement Learning techniques

Eduardo, Martín Izquierdo

Keywords: Reinforcement Learning, Trading, Stock Markets, Q-Learning, Neural Networks, Broker

Abstract

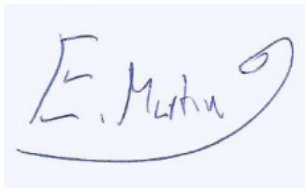
The use of trading bots in high frequencies is a very promising topic and quite recently researched. Following some references that are posted in the main document, it's proposed an approach using *Reinforcement Learning* for building a trading bot of high frequencies in order to maximize his profit dealing with any Time Series of the Stock Market in a given time interval.

A general analysis is made on possible solutions to this problem using *Reinforcement Learning*, discussing the advantages and disadvantages of each proposed technique. Later, it's described the state of the art related to *Reinforcement Learning*, exposing the advantages and disadvantages of each technique based on the problem to be solved.

Once these analyzes have been done, the proposed solution for this problem is described, that is, taking in account the last general analysis, the final components are described justifying why they are used. These components add some improvements proposed for this problem in order to deal with any time series.

Finally, these improvements are empirically demonstrated, reaching the conclusion that this solution can deal with low broker commission's levels. However, this *Reinforcement Learning* system can't deal regularly with standard broker commission levels.

Yo, **Eduardo Martín Izquierdo**, alumno del **Máster Universitario Oficial en Ciencia de Datos e Ingeniería de Computadores** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 05715664-A, declaro explícitamente que el trabajo presentado es original, entendido en el sentido de que no he utilizado ninguna fuente sin citarla debidamente.

A handwritten signature in blue ink, appearing to read 'E. Martín', enclosed within a light blue rectangular box. The signature is stylized with a large, sweeping flourish at the end.

Fdo: Eduardo Martín Izquierdo

Granada a 10 de Septiembre de 2019.

Agradecimientos

Gracias a mis padres, que sin ellos nunca hubiera podido llegar a esta etapa final de mis estudios. Me han apoyado en los malos momentos y en las decisiones importantes de mi vida. Espero que estén orgullosos de mí al haber alcanzado este objetivo.

Agradecer al destino, de haber acabado estudiando sobre el apasionante área de la Ciencia de Datos, la cual está ahora en auge. También a las geniales personas que he conocido en este Máster, de las que espero que consigan sus objetivos en la vida.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivo	2
1.3. Estructura del documento	2
2. Análisis del problema	5
2.1. Trading intradía	5
2.2. Enfoque con RL	8
2.2.1. Agente	9
2.2.2. Entorno	10
2.2.3. Estados/Observaciones	10
2.2.4. Acciones	11
2.2.5. Recompensa	13
2.2.6. Entrenamiento y evaluación	15
3. Marco teórico	17
3.1. Métodos value-based	18
3.1.1. Value-Iteration	20
3.1.2. Q-Learning	20
3.1.3. SARSA	21
3.1.4. Deep Q-Learning (DQN)	22
3.2. Métodos policy-based	24
3.2.1. Método Cross-entropy (CEM)	24
3.2.2. Métodos Policy Gradients (PG)	25
3.2.3. Métodos para espacios continuos de acciones	29
3.2.4. Métodos Trust Regions	31
4. Solución propuesta	33
4.1. Referentes	33
4.2. Innovaciones y mejoras propuestas	34
4.3. Implementación del sistema de RL	34
4.3.1. Agente	34
4.3.2. Estados/Observaciones	35

4.3.3. Acciones	39
4.3.4. Recompensa	40
4.3.5. Entorno	40
4.3.6. Técnicas de RL usadas	43
5. Estudio experimental	45
5.1. Diseño experimental	45
5.1.1. Conjunto de datos de los experimentos	45
5.1.2. Hiperparámetros de las NNs	50
5.1.3. Particionamiento del conjunto de datos	51
5.1.4. Medidas de rendimiento	51
5.1.5. Estructura y desarrollo de los experimentos	52
5.2. Resultados experimentales	54
5.2.1. Resultados sin comisión	54
5.2.2. Resultados con comisión	62
5.2.3. Resultados globales de los experimentos	69
5.3. Análisis y conclusiones de los resultados	71
5.3.1. Análisis de resultados sin comisión	71
5.3.2. Análisis de resultados con comisión	76
5.3.3. Análisis de resultados globales y conclusiones de los análisis	80
6. Conclusiones y mejoras futuras	83
6.1. Conclusiones	83
6.2. Mejoras futuras propuestas	84
Bibliografía	88

Índice de figuras

2.1. Explicación velas japonesas	5
2.2. Cabecera de la serie	7
2.3. Gráfico de velas	7
2.4. Función a aproximar	8
2.5. Entidades RL y flujo	9
2.6. Ejemplo espacio de acciones	13
2.7. Recompensa de <i>Incremental Profit</i>	15
2.8. Entrenamiento y evaluación del agente	15
3.1. Tipos de enfoques RL	18
3.2. Distribución de probabilidad de acciones en métodos <i>policy-based</i>	24
3.3. Arquitectura método Actor-Critic	28
4.1. Normalización incremental	38
4.2. Ejemplo de Ruido en el espacio de acciones	39
4.3. Interacción del agente y el entorno	42
4.4. Formato de las observaciones	44
5.1. Serie de Apple con distinta granularidad	47
5.2. Precios de Intel, Apple y Qualcomm	49
5.3. Hiperparámetros usados en las NNs	50
5.4. Gráficos del entrenamiento de Apple	55
5.6. Conjunto de entrenamiento de Apple transformado	56
5.5. Compra y venta en series de Apple. Punto rojos: <i>Compras</i> . Puntos verdes: <i>Ventas</i> . Tamaño del punto: <i>Ponderado por la cantidad</i> . Eje X: <i>Tiempo</i> . Eje Y: <i>Precio</i>	57
5.7. Compras y ventas de A3C con MLP en el conjunto de prueba de Apple. Punto rojos: <i>Compras</i> . Puntos verdes: <i>Ventas</i> . Tamaño del punto: <i>Ponderado por la cantidad</i> . Eje X: <i>Tiempo</i> . Eje Y: <i>Precio</i>	58
5.8. Gráficos de entrenamiento en serie de Apple transformada	59
5.9. Gráficos del entrenamiento de Intel	60

5.10. Compra y venta en series de Intel. Punto rojos: <i>Compras</i> . Puntos verdes: <i>Ventas</i> . Tamaño del punto: <i>Ponderado por la cantidad</i> . Eje X: <i>Tiempo</i> . Eje Y: <i>Precio</i>	61
5.11. Gráficos del entrenamiento de Qualcomm	63
5.12. Compra y venta en series de Qualcomm. Punto rojos: <i>Compras</i> . Puntos verdes: <i>Ventas</i> . Tamaño del punto: <i>Ponderado por la cantidad</i> . Eje X: <i>Tiempo</i> . Eje Y: <i>Precio</i>	64
5.13. Recompensas medias serie de Apple con 0.1 % de comisión . .	65
5.14. Gráficos de Apple con 90 % de ruido. <i>Gráficos de compras y ventas</i> \Rightarrow <i>Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio</i>	66
5.15. Gráficos de Intel con 80 % de ruido. <i>Gráficos de compras y ventas</i> \Rightarrow <i>Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio</i>	68
5.16. Gráficos de Qualcomm con 85 % de ruido. <i>Gráficos de compras y ventas</i> \Rightarrow <i>Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio</i>	70
5.17. Explicación de comisión vs. beneficio	76
6.1. Arquitectura general <i>autoencoder</i>	85
6.2. Gráfico de compra ventas en serie de granularidad minuto a minuto. <i>Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio</i>	86

Índice de cuadros

5.1. Comparación de beneficio con distinta granularidad	46
5.2. Resultados de Apple en el conjunto de entrenamiento	55
5.3. Resultados de Apple en el conjunto de prueba	55
5.4. Resultados de Apple en el conjunto de entrenamiento con variables alternativas y sin tendencia	58
5.5. Resultados de Apple en el conjunto de prueba con variables alternativas y sin tendencia	58
5.6. Resultados de Intel en el conjunto de entrenamiento	60
5.7. Resultados de Intel en el conjunto de prueba	60
5.8. Resultados de Qualcomm en el conjunto de entrenamiento . .	62
5.9. Resultados de Qualcomm en el conjunto de prueba	62
5.10. Resultados de Apple sin tendencia con 90 % de ruido	65
5.11. Resultados de Intel con 80 % de ruido	67
5.12. Resultados de Qualcomm con 85 % de ruido	69
5.13. Resultados globales series profit relativo por hora para con- juntos de entrenamiento	69
5.14. Resultados globales series profit relativo por hora para con- juntos de prueba	69

Capítulo 1

Introducción

1.1. Motivación

El *análisis técnico* del conjunto de datos histórico del mercado bursátil ha sido una herramienta útil para el estudio de las inversiones. Sin embargo, es un ámbito donde pocos analistas han conseguido el éxito.

El **mercado bursátil** se puede definir como el conjunto de aquellas entidades que realizan transacciones de productos financieros en distintas Bolsas internacionales. Este mercado está regulado y centralizado, permitiendo a distintas **empresas** financiar sus actividades a partir de la venta de distintos títulos o activos. Estos títulos o activos son comprados por distintos **inversores** que buscan una rentabilidad a corto o largo plazo.

Un elemento del mercado bursátil es la **Bolsa de Valores**, que es una organización privada que permite la compra o venta de **acciones**. Estas acciones hacen referencia a los distintos títulos o activos mencionados anteriormente.

La **negociación bursátil** o *trading* en frecuencias altas, como minutos u horas, es solamente viable para *robots automáticos*, ya que son capaces de hacer cálculos matemáticos en velocidades altísimas. Una persona dedicada al *análisis técnico* no sería capaz de igualar estas velocidades de cálculo y toma de decisiones.

El uso de *robots* automáticos en *trading* de **altas frecuencias** está justificado por los siguientes puntos:

- Se tienen muchos más ejemplos y cuanto más datos, más posibilidades de que el agente aprenda patrones interesantes en el tiempo.
- Si son intervalos muy pequeños, el agente podría ser capaz de ver patrones interesantes que un humano no tendría tiempo para ver.

- Estos patrones son impredecibles en un *trading* a largo plazo, ya que se tendrían muchos menos ejemplos en los datos y faltaría información como noticias relevantes al producto.

El auge del área del *Aprendizaje Automático* junto con el *Deep Learning* han impulsado esta motivación para la construcción de un agente automático de inversión capaz de tomar decisiones inteligentes con el fin de maximizar su beneficio económico.

1.2. Objetivo

El **objetivo** principal de este proyecto es la construcción de un *robot* automático de *trading* intradía que busque maximizar las ganancias económicas en un intervalo de tiempo determinado. El tiempo determinado en este caso es el tamaño de la serie temporal, que puede ser variable.

1.3. Estructura del documento

El presente documento está compuesto de los siguientes capítulos:

1. **Introducción:** Es el presente capítulo, donde se ha presentado la motivación para abordar este trabajo así como el objetivo de este. Se muestran también algunos conceptos básicos de la Bolsa de Valores para los lectores que no tengan muchos conocimientos sobre este área.
2. **Análisis del problema:** Se muestra un análisis o estudio general para resolver este problema de **trading**. Se definen los componentes principales de RL con sus distintas versiones alternativas para este problema.
3. **Marco teórico:** En este capítulo se definen una serie de técnicas de RL agrupadas según su tipo, desde las clásicas a las últimas del *estado del arte*.
4. **Solución propuesta:** Mientras que en el segundo capítulo se analiza de forma general como abordar este problema, en este capítulo se muestra detalladamente la construcción de la solución propuesta con sus debidos componentes.
5. **Estudio Experimental:** Con la solución ideada en el capítulo cuatro, se van a mostrar un estudio experimental con tres series temporales distintas con el fin de mostrar la efectividad general de este sistema de RL.

6. **Conclusiones:** Finalmente, se muestran las conclusiones obtenidas por la realización de este trabajo y sus resultados. Además, se proponen mejoras para trabajos futuros con el fin de mejorar los resultados obtenidos.

Capítulo 2

Análisis del problema

2.1. Trading intradía

Cuando se habla de *trading* intradía, se refiere a la capacidad de operar en tiempos menores que un día (Ej: hora, 30 min, 15 min).



Figura 2.1: Explicación velas japonesas

Para la construcción o entrenamiento del agente, se requiere de un conjunto de datos de la Bolsa de Valores. El conjunto de datos es una serie temporal de valores históricos sobre una entidad cualquiera. Estos valores se pueden ver como una **serie temporal multivariante**, donde cada instancia es una **vela japonesa** desde el punto de vista del *análisis técnico*. Las velas describen el valor que ha tomado el activo en un intervalo de tiempo. En la Figura 2.1 se muestra la forma de las velas con sus partes indicadas.

Algunas características de las velas se describen a continuación:

- Las velas con un cuerpo grande indican gran fuerza en la dirección que toma el precio.

- Las velas con poco cuerpo muestran indecisión.
- Las velas con alguna mecha muy acentuada indica un cambio de dirección del precio.
- El color de la vela especifica la dirección del mercado. En este caso el verde indica dirección positiva y el rojo dirección negativa.

Los campos de las velas tienen el formato OHLC (Open, High, Low, Close), seguido de varios campos más. Estos campos se pueden identificar gráficamente como se muestra en la Figura 2.1. OHLC hace referencia a **precio apertura, máximo de la sesión, mínimo de la sesión y precio cierre** respectivamente. En la Figura 2.2 se muestra la cabecera de un ejemplo de serie temporal de valores históricos intradía. Se puede ver como cada fila representa un minuto. Cada campo de esta serie se define a continuación:

1. **Datetime:** Fecha de inicio de la vela. En este caso es a escala por minuto.
2. **Open:** Valor de entrada de la empresa en ese intervalo temporal.
3. **High:** Valor máximo de la empresa en la vela.
4. **Low:** Valor mínimo de la empresa en la vela
5. **Close:** Valor final de la empresa en la vela.
6. **Volume:** Volumen de negociaciones cotizadas en la vela. Normalmente representa el grosor de esta. Es un campo importante porque muestra la liquidez del mercado.
7. **Number of Trades:** Número de negociaciones o *tradeos* realizados en la vela. Por lógica, este campo está directamente relacionado con el volumen.
8. **Weighted average price:** Es la media ponderada del precio en esa vela. Este campo ya viene proporcionado por los datos, pero también se podría calcular a mano. Por ejemplo, se podría calcular como:

$$Average = \frac{(Low + High)}{2}$$

	DATETIME	OPEN	HIGH	CLOSE	LOW	VOLUME	NUMBER OF TRADES	WEIGHTED AV. PRICE
0	2012-07-02 09:30:00	26.51	26.62	26.59	26.49	8667	630	26.520
1	2012-07-02 09:31:00	26.59	26.61	26.60	26.58	808	333	26.590
2	2012-07-02 09:32:00	26.60	26.63	26.62	26.59	319	152	26.608
3	2012-07-02 09:33:00	26.62	26.62	26.61	26.61	441	195	26.613
4	2012-07-02 09:34:00	26.62	26.62	26.61	26.57	1050	447	26.593
5	2012-07-02 09:35:00	26.60	26.71	26.70	26.60	1133	521	26.649
6	2012-07-02 09:36:00	26.70	26.73	26.71	26.70	1154	567	26.715
7	2012-07-02 09:37:00	26.71	26.73	26.68	26.68	1047	465	26.708
8	2012-07-02 09:38:00	26.67	26.69	26.68	26.61	1314	464	26.643

Figura 2.2: Cabecera de la serie

En la Figura 2.3 se puede ver un ejemplo de gráfico de velas en el tiempo para el intercambio Bitcoin/USD en granularidad minuto a minuto. Cada vela es un minuto en el tiempo, que está definida por los campos OHLC descritos antes.



Figura 2.3: Gráfico de velas

Además, la idea es de hacer un agente que sea **general**, es decir, que pueda actuar correctamente en cualquier serie. Por ejemplo, que sea capaz de ganar beneficio en la serie de valores de IBM y Microsoft.

El área de *Aprendizaje Automático* usada para la construcción de este robot de trading es el **Aprendizaje por Refuerzo** o *Reinforcement Learning* (RL). En la siguiente sección se explica el enfoque de RL para abordar de forma general este problema.

2.2. Enfoque con RL

Se ha considerado la técnica más apropiada para abordar este problema ya que se trata de *aprendizaje no supervisado* al no tener ningún tipo de etiquetas que especifiquen como debería comportarse el agente. Las técnicas de RL pueden encontrar un comportamiento de inversión óptimo únicamente usando una *función de recompensa*.

En el área del RL, la **política** se define como el conjunto de reglas que controlan el comportamiento del agente. Formalmente se define como la distribución de probabilidad de las acciones del agente para cada posible estado:

$$\pi(a|s) = P[A_t = a | S_t = s] \quad (2.1)$$

En este problema, hay que aprender o aproximar una **función** para mapear las observaciones con las acciones que dan lugar a una **política óptima** que maximice el valor total de las recompensas recibidas del entorno. La política óptima garantiza la mejor acción posible en cada estado. La Figura 2.4 muestra esta función, que relaciona las observaciones con las acciones. Para aproximar esta función, hay técnicas de RL que tienen la capacidad de hacerlo.



Figura 2.4: Función a aproximar

En RL, las componentes básicas con sus relaciones entre sí se pueden ver relacionadas en la Figura 2.5. Estas componentes principales se definen a continuación:

- **Agente:** Agente o *bot* que queremos hacer que aprenda un comportamiento o política determinada.
- **Acciones:** Son las distintas acciones que puede realizar el agente en cada instante de tiempo.
- **Entorno:** Es el entorno donde se encuentra el agente. Este es el encargado de proporcionar las recompensas y observaciones al agente dependiendo de sus acciones.
- **Estados:** Son los distintos estados en los que se puede encontrar el agente en ese entorno. Por ejemplo, en el juego *tic-tac-toe* el espacio de estados es 3^9 . Hay veces que los estados pueden ser muy complejos

dependiendo del entorno que se esté abordando. Puede haber entornos cuyo estado quede definido por un vector formado por una serie de valores continuos, como será este caso.

- **Observaciones:** Es lo que el agente observa o ve dentro del entorno en cada estado.
- **Recompensa:** Es un valor escalar obtenido de forma periódica desde el entorno. El propósito que tiene es decir a nuestro agente cómo de bien se está comportando en ese entorno. De esta forma, el agente aprenderá a base de *prueba y error*, ya que gracias a esta recompensa sabrá distinguir lo bueno de lo malo.

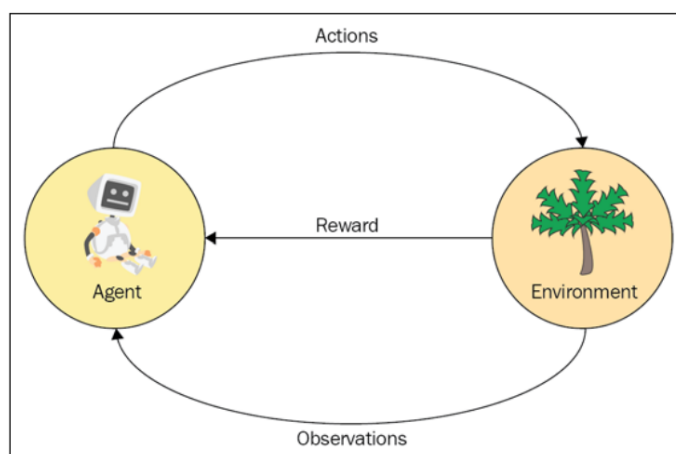


Figura 2.5: Entidades RL y flujo

Para este sistema es muy importante definir estos componentes antes de elegir la técnica de RL a usar. Estos componentes se definen ahora en términos generales para resolver este problema. Más adelante en el capítulo 4 se especificará exactamente como se compone cada componente con su justificación correspondiente.

2.2.1. Agente

El **agente** es el **robot** de *trading* especificado anteriormente. Este agente se puede comparar como una persona inversora que toma decisiones de *trading* dependiendo del estado actual del mercado y el valor de su portafolio.

Este agente puede tener inicialmente una cantidad inicial de capital y acciones. Este tiene como objetivo maximizar su portafolio (formada por su dinero y el valor de sus acciones) al final del intervalo de tiempo dado. Este problema se puede ver como un **juego**, donde el jugador es el agente,

que tiene que conseguir el mayor *profit* posible. Este *juego* es lo que en RL se define como **episodio**. Este episodio normalmente está formado por los pasos que realiza el agente desde el primer estado hasta cuando llega a la última instancia definida en la serie temporal. Sin embargo, también se le pueden añadir normas al agente para que aprenda a no comprar cuando no tiene suficiente dinero, o que no venda cuando no tiene acciones. En este caso el agente **perdería** y empezaría de nuevo el siguiente episodio.

Hay que remarcar que normalmente un agente de bolsa no opera directamente sobre el mercado, sino que lo hace a través de un *broker*. Un *broker* es un individuo o entidad que organiza las transacciones entre el comprador y el vendedor en el mercado a cambio de una **comisión**. Una comisión estándar puede ser el 0.1 % del coste de la transacción. Por ejemplo, si la acción en tal instante cuesta 1000 euros, el coste total de comprar esa acción sería de 1001 euros ($1000 + 1000 * 0.001$). Por este motivo, conviene que el agente sea conservador en cierta medida para que no abuse de la comisión en la compra y venta de acciones. Como se verá en el capítulo de desarrollo, la comisión es una variable muy a tener en cuenta para que el agente sea capaz de obtener beneficio.

2.2.2. Entorno

Como se ha dicho antes, la **Bolsa de Valores** es el entorno del sistema RL. Pero hay un aspecto importante a remarcar, y es que hay muchos otros agentes, ya sean automáticos o personas, *tradeando* en el mismo mercado. Estos agentes también influyen en este entorno, y no se tiene una información completa sobre ellos, por lo tanto, es algo que no se puede controlar.

Este entorno funcionará recibiendo una acción por parte del agente, ya sea comprar, vender o no hacer nada, y este le devolverá una recompensa, dependiendo de lo buena que haya sido esa acción, y la siguiente observación del entorno.

2.2.3. Estados/Observaciones

Normalmente, los métodos de RL básicos usan un formalismo de **Proceso de Decisión de Markov** (MDP), donde se asume que el entorno sigue la **propiedad de Markov**. Esta propiedad se basa en que las observaciones del entorno es todo lo que el agente necesita para actuar de forma óptima, es decir, que las observaciones le permiten distinguir unos estados de otros.

Sin embargo, antes se ha dicho que no podemos saber el estado del resto de agentes dentro del entorno. Por ejemplo, no se puede saber cuantos agentes hay en el mercado ni su portafolio. Esto quiere decir que se está tratando con un **MDP Parcialmente Observado** (POMDP). Un POMDP

es básicamente un MDP pero sin cumplir la propiedad de Markov, y estos son muy importantes en la práctica. Entonces, se puede asumir que lo que el agente observa no es el estado actual del entorno S_t , sino una derivación de este. Así, llámese X_t a la observación del agente, que se calcula a partir de una función aplicada al estado completo:

$$X_t \approx O(S_t) \quad (2.2)$$

Una observación X_t puede estar formada por los siguientes campos:

- **Valor de la vela actual.** Es la observación de la serie temporal en el intervalo actual.
- **Las N barras o velas pasadas.** Esto es las N últimas observaciones de la serie temporal con formato OHCL. Estos valores se añaden con el fin de que el agente sepa en ese estado una información pasada del precio para intentar predecir hacia donde irá en el futuro.
- **Información del estado actual del agente:** Los dos anteriores campos vienen directamente de la serie temporal utilizada. Sin embargo, el estado del agente también está definido por su **información propia en ese momento**. Esta información puede referirse al número de acciones que contiene, el capital, el valor total del portafolio, etc.

El **valor del portafolio** es un elemento importante dentro del sistema de RL. Este valor está definido por la suma del valor económico de la cuenta del agente. El valor del portafolio en el estado s se define por la siguiente fórmula:

$$valor_{portafolio} = dinero + n_{acciones} * valor_{accion} \quad (2.3)$$

Se puede observar que todas las variables de las observaciones son continuos o con una variabilidad bastante alta y además hay una alta dimensionalidad de estos. Esto significa que lo más seguro es que la técnica de RL más apropiada para este problema deba utilizar una **Red Neuronal (NN)** para aproximar la función de la figura 2.4.

2.2.4. Acciones

En RL, el espacio de acciones puede ser desde discreto (finito) hasta continuo (infinito). Además, puede ser un espacio de acciones unidimensional o multidimensional. Por ejemplo, un agente puede realizar más de un tipo de acción en un determinado estado.

Un problema añadido que se debe tener en cuenta, es que el número de acciones a comprar o vender es variable en los distintos estados. Ambos casos se justifican a continuación:

- **Comprar:** El comprar siempre depende del **dinero** que tenga el agente en ese momento y del **precio de la acción**. Estas variables varían en el tiempo, por lo que el número de acciones que puede comprar el agente también es variable.
- **Vender:** El vender depende del número de acciones que el agente tenga en ese momento. También es una variable que varía en el tiempo.

Como el **espacio de acciones** debe estar sesgado y definido desde el principio, no se puede tomar el número de acciones a comprar o vender como posibles acciones a realizar por el agente. Una solución simple sería definir el espacio de acciones de la siguiente forma:

- **Comprar una acción:** El agente compra una acción si tiene el dinero disponible, en caso contrario no hace nada o pierde, dependiendo de como se haya definido el problema.
- **Vender una acción:** El agente vendería una acción si tiene mínimo una en su portafolio, en caso contrario no hace nada o pierde.
- **No hacer nada:** El agente pasa un estado sin comprar ni vender.

En este caso se tendría un espacio de acciones discreto de 3 valores posibles. Se simplifica mucho el problema, pero se están quitando posibilidades al agente de realizar acciones más realistas. Por lo tanto, puede que el agente esté limitado para ganar más beneficio.

Una solución más compleja pero que posiblemente se ajuste más al problema sería tener un espacio de acciones **Multi-Discreto**. Las dos dimensiones serían las siguientes:

1. **Decisión:** Esta variable especifica si el agente va a comprar, vender o no hacer nada. Se trata de una variable discreta con 3 valores.
2. **Cantidad:** Define la fracción de dinero o acciones a usar para comprar o vender. Es una variable discreta en principio, donde el número de fracciones puede ajustarse dependiendo del método de RL que se aplique. Por ejemplo, si hay un método de RL que es frágil ante grandes espacios de acciones, mejor introducir un número pequeño de fracciones. Sin embargo, si se añade un número de fracciones muy grande, esta variable puede quedar cerca de ser continua, dando lugar al conflicto de variable discreta contra continua. Si se considerase continua, habría que pensar en un método de RL que soporte este espacio de acciones. Esta variable no se tiene en cuenta cuando en la **decisión** se toma el *no hacer nada*.

En la Figura 2.6 se puede ver un ejemplo del espacio de acciones donde se divide la cantidad a 6 fracciones.

Decisión	Cantidad
No hacer nada	1/6
	2/6
Comprar	3/6
	4/6
Vender	5/6
	6/6

Figura 2.6: Ejemplo espacio de acciones

Se puede pensar por qué no juntar las 2 dimensiones en una para simplificar este espacio de acciones. Un ejemplo sería pasarlo a una dimensión con las siguientes acciones: (*Comprar 1/3, Comprar 2/3, Comprar 3/3, No hacer Nada, Vender 1/3, Vender 2/3, Vender 3/3*). Parece que se simplifica el espacio de estado, pero hay un problema y es que ahora disminuye mucho la probabilidad de que el agente llegue a la acción *no hacer nada*. Mientras que en el ejemplo de la Figura 2.6 la probabilidad es $1/3$, en este ejemplo la probabilidad es $1/7$. Esta probabilidad interesa que sea alta para que no se pierda mucho por culpa de la comisión.

Esto se puede solucionar duplicando la acción no hacer nada de la siguiente forma: (*Comprar 1/3, Comprar 2/3, Comprar 3/3, No hacer Nada, No hacer Nada, No hacer Nada, Vender 1/3, Vender 2/3, Vender 3/3*).

2.2.5. Recompensa

La recompensa es uno de los puntos más delicados, cuya definición puede influir mucho en el aprendizaje del agente, haciendo una convergencia más rápida o lenta.

En [Britz, 2018], se muestran distintos tipos de recompensas que se pueden aplicar a este problema. La recompensa más básica es la *Realized PnL*, que se basa en el **beneficio neto** obtenido entre la compra y la venta de acciones. Por lo tanto, el agente solo tomaría la señal de refuerzo cuando vendiera las acciones. Esta recompensa tiene los siguientes inconvenientes para este problema:

- La señal de refuerzo para el agente es poco frecuente, ya que solo se recibiría cuando el agente venda. Esto puede hacer que la convergencia se mucho más lenta y que el agente nunca venda y solo se dedique a comprar.

- Es difícil de implementar por el hecho de que el n° de acciones compradas o vendidas es variable.

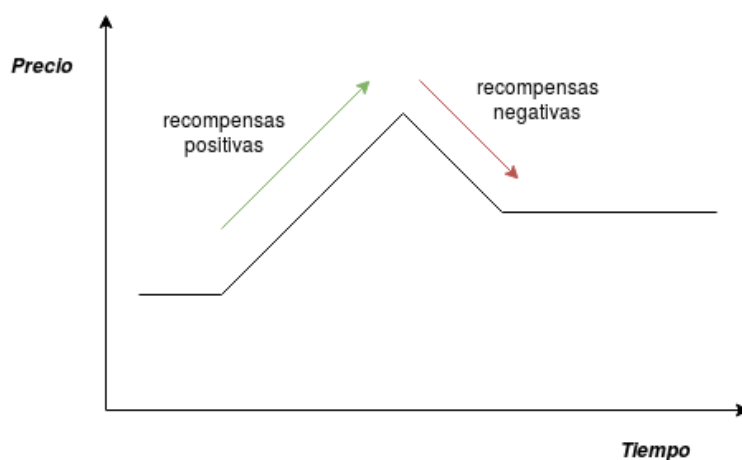
Otro tipo de recompensa es *Unrealized PnL*, la cual se puede definir como el beneficio o *profit* del agente respecto a su dinero inicial. Se puede mostrar en forma de dinero o porcentaje. Básicamente es ver la diferencia del dinero inicial con el valor actual del portafolio. Por ejemplo, si empezó con 3000 y en ese momento el valor del portafolio es 4000, la recompensa es de 1000 o 33 %. Si se compara esta recompensa con la anterior, la señal de refuerzo sería mucho más frecuente, dando lugar a posiblemente una convergencia más rápida.

Sin embargo, esta última recompensa tiene el problema de que su valor es muy influido por las acciones de los estados anteriores. Además, de que puede estancarse en **mínimos locales** ya que el beneficio puede ser positivo pero estar perdiendo dinero en ese estado y solo darse cuenta el agente cuando sea negativo el *profit*, no sabiendo qué acciones fueron las que provocaron esa pérdida de *profit*.

Una forma de solucionar algunos de los problemas anteriores, es usando la recompensa *Incremental profit*. Este tipo de recompensa fue propuesta en el artículo [King, 2019], que expone con pruebas como esta recompensa es mejor que *Unrealized Pnl*. Esta recompensa se define según la siguiente fórmula:

$$recompensa = valor_{portafolio}(s_t) - valor_{portafolio}(s_{t-1}) \quad (2.4)$$

Por lo tanto, es simplemente el valor del portafolio en el estado actual menos el valor del portafolio en el estado anterior. De esta forma se tiene una recompensa más influyente en el estado actual, que además de recompensar cuando el precio de la acción sube, también recompensa cuando se vende y el precio iba a disminuir próximamente. Esto se representa en la Figura 2.7, donde se muestran los signos las recompensas (positivas o negativas) obtenidas suponiendo que se mantienen acciones desde el principio. Una vez visto esto por el agente, pensará en comprar en la subida y en vender en el máximo de la subida. En cambio, si la recompensa en ese caso fuera *Unrealized PnL*, la recompensa sería siempre positiva, ya que el precio es siempre mayor al del inicio de la serie temporal.

Figura 2.7: Recompensa de *Incremental Profit*

2.2.6. Entrenamiento y evaluación

La estrategia para entrenar el agente y evaluarlo es importante. La idea es evaluar al agente de RL usando **backtesting**, que es el proceso de testear una estrategia de *trading* antes de emplearla. Permite al usuario conocer si la aproximación que está usando es correcta y si el modelo tiene la cobertura deseada.

En la Figura 2.8 se puede ver el proceso. Primero se aplica el *backtest* en datos históricos, y una vez que se considera bueno, se aplica el agente entrenado en datos futuros. Esto se puede simular dividiendo la serie temporal en un conjunto de entrenamiento y prueba. Por lo que se entrenaría en el de de entrenamiento y se evaluaría en el de prueba. Y si funciona bien en el conjunto de prueba, se podría usar en un entorno de tiempo real.

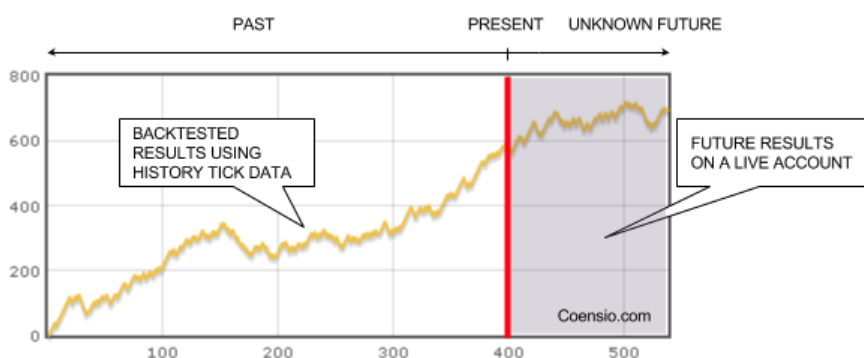


Figura 2.8: Entrenamiento y evaluación del agente

Capítulo 3

Marco teórico

Una vez expuesto un análisis general para resolver este problema con RL, hay que investigar sobre distintas técnicas de RL a utilizar con su justificación de uso para aproximar la función de la Figura 2.4.

Tal y como se menciona en [Lapan, 2018], los métodos de RL se agrupan en distintas familias. En la Figura 3.1 se pueden ver los distintos tipos de enfoques de RL. Primeramente se dividen según el modelo:

- **Model-based:** Intentan predecir cuál es la próxima observación o recompensa. Dependiendo de la predicción toma la mejor acción.
- **Model-free:** No construye un modelo del entorno o la recompensa. Solo conecta observaciones a acciones.

Dentro del enfoque **Model-free** tenemos dos subenfoques:

- **Policy-based:** Estos métodos están aproximando la política del agente, es decir, qué acciones tiene que llevar a cabo el agente en cada estado. No usan una **función valor** que mapea un estado y acción con su valor. En cambio, tratan de mapear los estados con las acciones, lo que puede ser estocástico. Una ventaja importante sobre los métodos **Value-based** es que permiten aprender más fácilmente en un espacio continuo de acciones, por si se quiere añadir al agente para mejorar el beneficio.
- **Value-based:** En vez de calcular la probabilidad de cada acción, el agente calcula el valor de cada posible acción en ese estado, tomando así la mejor acción. Para eso aprende una **función valor**.

Dentro de la **política** se pueden diferenciar los métodos:

1. **On-policy**: El método necesita nuevos datos (datos frescos) obtenidos del entorno.
2. **Off-policy**: Puede aprender de datos históricos.

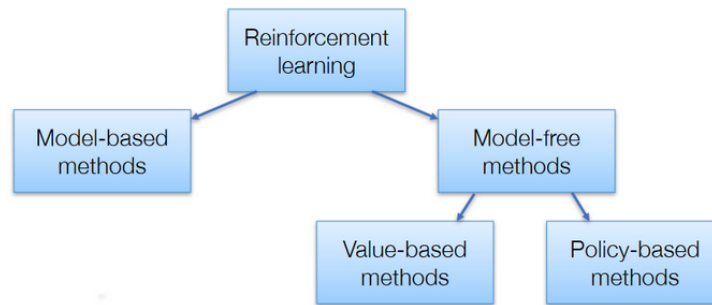


Figura 3.1: Tipos de enfoques RL

3.1. Métodos value-based

En [Sutton y Barto, 2014] (considerado como la biblia del RL), se muestran y describen los elementos y conceptos principales de los métodos **value-based**.

Estos métodos suelen estar enfocados a entornos MDP. MDP se utiliza para describir formalmente la interacción agente-entorno. Consisten en una tupla de 4 elementos:

1. \mathcal{S} : Conjunto de estados.
2. \mathcal{A} : Conjunto de acciones.
3. $p(s_{t+1}|s_t, a_t)$: Modelo de transición de estados que muestra cómo el estado del agente cambia cuando hace una acción a en el estado s .
4. $p(r_{t+1}|s_t, a_t)$: Modelo de recompensa que describe el valor recompensa cuando el agente realiza una acción a en el estado s .

Se suelen usar en entornos **deterministas**, es decir, cada acción es un éxito y siempre se empieza el episodio desde el mismo estado y cuando llega al estado x , se termina el episodio.

Como se ha mencionado en el capítulo 2, el objetivo del agente es tomar la mejor política que maximice la recompensa total recibida del entorno. La

recompensa total descontada se define como:

$$recompensaTotalDescontada = \sum_{i=1}^T \gamma^{i-1} r_i \quad (3.1)$$

Donde el **factor de descuento** γ se usa para dar mayor peso a las recompensas cercanas y T es la longitud del episodio, que puede ser infinito si hay una longitud máxima para el episodio.

Estos métodos los podríamos agrupar en el tipo **value-based**, ya que tratamos de aprender una **función-valor** o **valor de estado**, que depende de la política π seguida por el agente:

$$V^\pi(s) = E\left[\sum_{i=1}^T \gamma^{i-1} r_i\right] \quad \forall s \in \mathbb{S} \quad (3.2)$$

Donde r_t es la recompensa local obtenida en el paso t del episodio y γ es el factor de descuento que controla la importancia de recompensas futuras. El valor de estado representa cómo de bueno es un estado para que el agente se encuentre ahí.

Existe una **función-valor óptima** que obtiene el mayor valor para cada estado s :

$$V^*(s) = \max_{\pi} V^\pi(s) \quad \forall s \in \mathbb{S} \quad (3.3)$$

Por conveniencia en la práctica, se introduce la *función-Q*, que mapea cada par *estado-acción* con su valor y se traduce a como de bueno es para el agente tomar una acción a en el estado s :

$$Q(s, a) = v \quad (3.4)$$

Ahora, se puede definir la *función-valor* a partir de la *función-Q*:

$$V(s) = \max_{a \in \mathbb{A}} Q(s, a) \quad \forall s \in \mathbb{S} \quad (3.5)$$

Este tipo de métodos usan la **ecuación de Bellman**, que actualiza recursivamente los valores de estas funciones usando programación dinámica para llegar a la **función-Q óptima**:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s, a) V^*(s') \quad (3.6)$$

Se dice que la *función-Q óptima* es la suma de la recompensa inmediata $R(s, a)$ y la recompensa futura esperada descontada después de la transición al siguiente estado s' .

3.1.1. Value-Iteration

El método **Value-Iteration** calcula la *función-valor* $V(s)$ óptima a partir de mejorar la estimación de $V(s)$ de forma iterativa. El algoritmo inicializa los valores de todos los estados de $V(s)$ de forma aleatoria. Posteriormente actualiza iterativamente $Q(s, a)$ y $V(s)$ hasta que converjan. Esta actualización se hace con la ecuación de Bellman (3.6) y actualiza $V(s)$ con la relación (3.5).

Sin embargo, este método no es viable para el problema de este proyecto por la razón de que el entorno es POMDP. Esto quiere decir que inicialmente no se dispone de los siguientes elementos esenciales para este método:

- $p(s_{t+1}|s_t, a_t)$: El **modelo de transición** de estados no se tiene inicialmente, sino que habría que calcularlo sobre la marcha.
- $R(s, a)$: El **modelo de recompensa** tampoco se tiene a priori.

Si fuera un entorno MDP, se dispondría de estos modelos y sería viable la aplicación del método Value-Iteration. De todas formas, si se tuvieran estos elementos, sería un método muy ineficiente ya que el espacio de estados y acciones es demasiado grande. Además, se tendría que almacenar demasiados datos en memoria y realizar muchas iteraciones ya que este método obliga a iterar sobre cada estado del espacio de estados.

3.1.2. Q-Learning

Como solución al problema del método *Value-Iteration*, existe la familia de métodos *Q-Learning*. Estos métodos se diferencian con el método *Value-Iteration* en los siguientes puntos:

- No necesitan un modelo de transición ni de recompensa para encontrar la política óptima.
- Solo actualiza los valores de los estados a los que llega el agente, es decir, no necesita actualizar todos los estados del espacio de estados.

Esto da lugar a un método que converge más rápido y necesita de menos elementos para llegar a la política óptima. Como el método *Q-Learning* no necesita de los modelos de transición ni de recompensa, se dice que es *model-free*, ya que es capaz de llegar a la política óptima únicamente mediante interacciones con el entorno.

Por lo tanto, este método solo se centra en aproximar la *función-Q*. Esta función se suele definir como la *tabla-Q*, donde las filas son los estados y las

columnas las acciones (en un entorno básico). Dentro de cada celda de la tabla se encontraría el valor, que habría que actualizar mediante la ecuación de Bellman asociada:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a' \in \mathbb{A}} Q(s', a') - Q(s, a)) \quad (3.7)$$

Como se puede ver, esta ecuación actualiza los valores de la *tabla-Q* tomando en cuenta su valor actual, que se basa en básicamente hacer un promedio entre los *valores-Q* antiguos y nuevos usando un *ratio de aprendizaje* α para establecer el tamaño del salto en las actualizaciones. Además, el *ratio de aprendizaje* permite al agente converger de forma suavizada, incluso si el entorno es muy ruidoso.

El pseudocódigo del método *Q-Learning* abreviado según [Lapan, 2018] se define a continuación:

1. Se empieza con una tabla vacía para $Q(s, a)$.
2. Se interactúa con el entorno y se obtiene la tupla (*estado, acción, recompensa, nuevo estado*). En este paso se decide qué acción tomar siguiendo las pautas de *exploración contra explotación*, es decir, haciendo que el agente no sea totalmente *greedy*. Una estrategia de exploración posible es la ϵ -*greedy*.
3. Se actualiza $Q(s, a)$ usando la ecuación de Bellman (3.7).
4. Se comprueban las condiciones de convergencia. Si no se cumplen, repetir desde el paso 2.

3.1.3. SARSA

EL método SARSA viene del acrónimo **State-Action-Reward-State-Action**. Es un método de la familia *Q-Learning*, diferenciándose del original en que es *on-policy*, mientras que el original es *off-policy*. Además, la ecuación de Bellman para este método cambia un poco respecto a la de *Q-Learning* (3.7), siendo de la siguiente forma:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)) \quad (3.8)$$

El principal cambio se puede ver en que ahora la acción a' en el estado próximo s' no es la que maximiza el *valor-Q*. En este caso se selecciona la acción a' usando la misma política para seleccionar la acción a en el estado actual s (ej: ϵ -*greedy*).

En [Sutton y Barto, 2014] se hace una comparación extensa de *Q-Learning* con SARSA, de la que se obtienen las siguientes conclusiones:

- *Q-Learning* asegura converger en una política óptima, mientras que SARSA llega a una política **casi** óptima a la vez que explora. Una opción para hacer que SARSA obtenga siempre una política óptima es aplicar el método *ε-greedy*, haciendo ϵ decreciente en el tiempo.
- Los métodos *off-policy* tienen una varianza por muestra mayor que los *on-policy*, dando como resultado posibles problemas en convergencia.
- SARSA se acercará a la convergencia permitiendo penalizaciones por acciones con fines exploratorios, mientras que *Q-Learning* los ignorará. Esto hace a SARSA más conservador. Por ejemplo, si existe el riesgo de una gran recompensa negativa cerca del óptimo, *Q-Learning* pasará por esa recompensa mientras explora, mientras que SARSA optará por evitar ese camino óptimo peligroso y solo tenderá a usarlo cuando su tendencia de exploración se reduzca (ϵ bajo).

Una desventaja importante de ambos métodos para el problema de este proyecto es abordar un espacio de acciones y estados tan **grande y complejo**. Esto hará que la *tabla-Q* sea muy grande y con una dimensión muy alta, necesitando un gran número de datos para llegar a la política óptima.

El hecho de usar una tabla o matriz, obliga a discretizar cada variable continua del espacio de acciones y estados. Además, la discretización de variables creará muchos problemas en la práctica, porque no se sabe cuantos intervalos o *bins* se van a necesitar para representar estas variables. Por ejemplo, un número pequeño de intervalos puede suponer una pérdida grande de información, mientras que muchos intervalos puede dar lugar a una *tabla-Q* muy compleja.

3.1.4. Deep Q-Learning (DQN)

Viendo el problema anterior de los métodos *Q-Learning* y SARSA, hay una mejora posible para facilitar la aproximación de la *función-Q*. El artículo [Hornik et al., 1989] publicó el llamado *Teorema de aproximación universal para redes neuronales*, en el que se dice que para cualquier función continua f , existe una NN *feedforward* con una única capa oculta que es capaz de aproximar f . Este problema se podía solucionar cuando *DeepMind* publicó en 2015 el artículo [Mnih et al., 2013], donde describe un método dentro de la familia *Q-Learning* que puede usar una NN, llamada *Q-Network*, para aproximar la *función-Q*. Este método es el llamado **Deep Q-Learning**.

Utiliza **Descenso de Gradiente Estocástico (SGD)** para actualizar esa *función-Q* minimizando el error respecto a los parámetros del modelo. Este **error** se define como:

$$L = (Q(s, a) - (r + \gamma \max_{a' \in A} Q(s', a'))^2$$

Para que se pueda optimizar con SGD, los datos del train necesitan ser **independientes y distribuidos de forma idéntica**. Por lo tanto, hay que utilizar lo que se llama **replay buffer**. Este *buffer* guarda las experiencias pasadas y se toman ejemplos de ahí para los datos de entrenamiento. Así se hacen un poco más independientes.

Como puede haber correlaciones entre distintos pasos en un episodio ($Q(s, a)$ a $Q(s', a')$), se usa un truco que es el llamado **target network**. Esto consiste en tener una copia de la NN y usarla para la aproximación de $Q(s', a')$. Esta copia de la NN se sincroniza con la original cada N pasos.

Este método no se suele usar para entornos MDP, sino para POMDP (Partial Observable MDP). Esto es porque se necesitan más de una observación para tomar nuestra decisión. Por ejemplo, si se tratara de un videojuego como el Pong, con solo una imagen del estado actual no podemos saber la dirección de la bola. Por lo tanto, se suelen usar redes neuronales con arquitectura RNN o CNN. Esto se puede enfocar al problema de este proyecto en que no se puede saber qué acción tomar si no se sabe la tendencia que lleva el precio, es decir, se necesitan saber los N valores anteriores.

Entre las distintas desventajas que se pueden remarcar de esta técnica, es que el tiempo que tarda el modelo en converger puede ser muy elevado. Otra desventaja podría ser que este método es únicamente *value-based*. Por lo tanto, podría significar que el tiempo de convergencia es más lento que un método *policy-based*, ya que este último tiene una exploración innata y no es necesario obligarla como en *Q-Learning* con una técnica *ϵ -greedy*.

Una diferencia importante de los métodos *policy-based* con Q-Learning es que no se usa **replay buffer**. Los métodos *policy-based* son *on-policy*, por lo que no pueden entrenar con datos antiguos. Esto es bueno y malo. La parte buena es que estos métodos convergen antes normalmente. La parte mala es que requieren muchas más interacciones con el entorno que los métodos *off-policy* como DQN. Sin embargo, el número de interacciones con el entorno no es una desventaja en nuestro caso, ya que nuestro entorno es una serie temporal y no hay demora para recibir las observaciones. Se podría decir que los métodos *policy-based* son preferibles en entornos baratos y rápidos, como es este caso.

Algunas mejoras para DQN se muestran en [Hessel et al., 2017] y [Lapan, 2018], las cuales mejoran la convergencia, la estabilidad y *sample efficiency*. Con *sample efficiency* se refiere a la eficiencia del agente en cuanto al n° de datos o ejemplos que necesita para aprender. Por lo tanto, si necesita pocos, se llamaría *sample efficient*. Un listado con estas mejoras explicado de forma resumida se muestra a continuación:

- **N-steps DQN**: Se mejora la estabilidad y la velocidad de convergencia con el uso de un desenrollado de la ecuación de Bellman.

- **Double DQN:** Se trata la sobre-estimación de los *valores- Q* por parte de DQN.
- **Noisy networks:** Trata de mejorar la exploración del agente añadiendo ruido a los pesos de la NN.
- **Prioritized replay buffer:** Tomar ejemplos del *replay buffer* con un muestreo uniforme no es la mejor idea. Por ello, se hace un muestreo priorizado.
- **Dueling DQN:** Se mejora la velocidad de convergencia haciendo la arquitectura de la NN más ajustada a el problema a resolver.
- **Categorical DQN:** Se mejora la exploración mediante el uso de una distribución de probabilidad sobre los *valores- Q* .

3.2. Métodos policy-based

A continuación se muestran los métodos **policy-based** más característicos según [Lapan, 2018]. Hay que remarcar que todos usan al menos una NN para aprender una **función no-lineal** que conecta observaciones con acciones. Se podría decir que es la función no-lineal de la Figura 2.4.

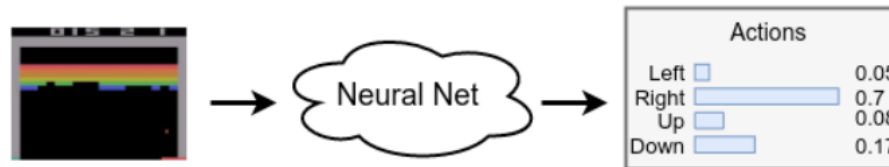


Figura 3.2: Distribución de probabilidad de acciones en métodos *policy-based*

Una diferencia importante con los métodos *value-based* que se menciona anteriormente, es que la salida de la NN es una distribución de probabilidad para un conjunto discreto de acciones. De esta forma, se realiza una exploración innata. Esta representación se puede ver en la Figura 3.2, donde para un juego de Atari, la NN obtiene una observación del juego y tiene como salida la distribución de probabilidad para las distintas acciones.

3.2.1. Método Cross-entropy (CEM)

Como primer método *policy-based*, tenemos al CEM. Además es *model-free* y *on-policy*. Este método se podría considerar el más sencillo de esta familia y se suele usar como introducción para posteriormente proponer métodos que resuelvan sus debilidades.

El pseudocódigo resumido es el siguiente:

1. Juega N episodios usando nuestro modelo y el entorno actual.
2. Calcula la recompensa para cada episodio y los ordena de forma decreciente.
3. Posteriormente se queda con un primer porcentaje de ellos (por ejemplo, el primer 20%).
4. Entrenar con esos buenos episodios usando las observaciones como entradas y acciones como salida.
5. Repetir desde el paso 1 hasta estar satisfechos.

Las **ventajas** son las siguientes:

1. Simple.
2. Fácil de paralelizar.
3. *Gradient free*.

Las **desventajas** son:

1. Para el entrenamiento, los episodios tienen que ser cortos y finitos.
2. La recompensa total de los episodios debería tener una variabilidad grande para separar los buenos de los malos.
3. No hay una indicación intermedia sobre si el agente está haciéndolo bien durante el episodio. Por lo tanto, no aprende sobre las acciones individuales.

Se podría decir que este método podría funcionar para una serie temporal no muy larga. Ya que en largas series este método no tendría una señal de refuerzo lo suficientemente frecuente para ayudar al agente a aprender.

3.2.2. Métodos Policy Gradients (PG)

Estos métodos PG se pueden ver como una mejora del anterior CEM. El **PG** se puede definir como:

$$L = -Q(s, a) \log \pi(a|s)$$

Esta fórmula se puede ver como una función de coste a optimizar por el SGD. Este PG define la dirección en la que se necesitan cambiar los parámetros de la red con el fin de mejorar la política que aumente la recompensa acumulada. Además, se intenta incrementar la probabilidad de las acciones que dan al agente mayor recompensa y decrementar las que dan peor recompensa.

Método REINFORCE

Este método es una mejora del CEM. El CEM toma el $Q(s,a)$ de la ecuación PG como 1 si son acciones que pertenecen a buenos episodios o 0 si pertenecen a malos episodios. De esta forma se mejora la granularidad de la evaluación de las acciones. Usando este método, se usará $Q(s,a)$ para el entrenamiento, haciendo que se incremente la probabilidad de las buenas acciones en el inicio del episodio y se decrementen las acciones cerca del fin del episodio.

El pseudocódigo del método REINFORCE es el siguiente:

1. Se inicializan los pesos de forma aleatorio de la NN.
2. Se juegan N episodios guardando las transiciones (s, a, r, s') .
3. Para cada paso del episodio k, se calcula la recompensa total descontada para pasos posteriores $Q(k,t)$.
4. Se calcula la función de coste para todas las transiciones.
5. Se aplica SGD para actualizar los pesos de la red minimizando el coste.
6. Se repite desde el paso 2 hasta que converja.

Viendo las diferencias con el método Q-Learning, están las comentadas anteriormente como la exploración innata y el no usar el *replay buffer*. Además, no se usa una *target network* para el cálculo de los valores Q . Estos valores se obtienen simplemente de la experiencia con el entorno. Por lo tanto, se hace un acceso menos a la NN dando lugar a una mayor rapidez en el entrenamiento.

Limitaciones del método REINFORCE:

- Todavía se necesita esperar que termine el episodio entero antes de empezar a entrenar. Esto se suma a que este método y el CEM necesitan más episodios para el entrenamiento, ya que a más datos, más precisión en el error PG. Esto significa que los episodios con cientos o miles de observaciones tendrán problemas, como es el caso de la serie temporal, a no ser que la serie se acorte demasiado. Para solucionar esto se propone usar el método *Actor-Critic*, que es el más famoso dentro de la familia PG, o usar la ecuación de Bellman desenrollada para ver más adelante en el tiempo.
- En este método hay alta varianza en los gradientes. Esto quiere decir que si hay un episodio en el que ha habido mucha suerte, esto puede afectar muy fuerte al gradiente final (cuando no debería porque ha sido

un episodio de suerte). Algunas soluciones a esto es restar un valor a Q , como puede ser la media de las recompensas descontadas.

- En cuanto a la exploración, el agente se puede quedar atascado aunque tenga una exploración innata. Esto se puede solucionar aplicando el llamado **bonus de la entropía**, que suma a la función de coste un valor de entropía. De esta forma penaliza al agente cuando tiene muy claro que acción tomar (poca entropía).
- Como se ha dicho anteriormente, para la optimización con SGD, se usaba el *replay buffer* para solucionar el tema de las observaciones independientes. El problema es que los métodos PG no pueden usar este *replay buffer*. La solución es entonces es usar entornos paralelos, comunicándonos con varios entornos a la vez para tomar sus transiciones para los datos de entrenamiento.

Método Actor-Critic (A2C)

Este método es uno de los más potentes dentro del RL, ya que actualmente se encuentra en el estado del arte. Se puede decir que es una mejora del anterior método REINFORCE, donde este último falla en entornos complejos y largos, como puede ser el caso de esta serie temporal.

Este método se centra en la debilidad del método REINFORCE en cuanto a reducir la varianza del gradiente. Esta reducción en este caso se aplica dependiendo del estado del agente, esto quiere decir, que el tipo de reducción que se aplica depende del estado.

Para decidir sobre la viabilidad de una acción en un estado, se usa la recompensa total descontada de la acción. Sin embargo, la recompensa total se puede representar como un *valor del estado* mas la *ventaja de la acción*:

$$Q(s, a) = V(s) + A(s, a)$$

Donde $V(s)$ es el valor del estado y $A(s, a)$ es la ventaja de la acción. Por lo tanto, se puede tomar el valor del estado como la base, y la ventaja de la acción se encarga de escalar el gradiente. Sin embargo, no se dispone del valor del estado $V(s)$, por lo que habría que usar otra NN para aproximar esa función.

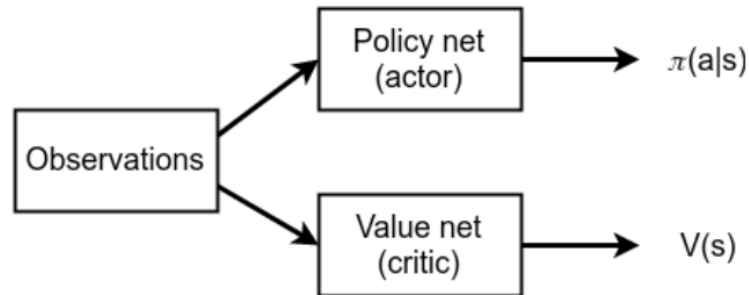


Figura 3.3: Arquitectura método Actor-Critic

En la Figura 3.3 se muestra la arquitectura de este tipo de métodos. Las 2 NNs de esta arquitectura se definen a continuación:

1. **Policy network:** Es el llamado **actor**, ya que controla cómo se comporta el agente. Devuelve la distribución de probabilidad del agente, y dice al agente qué hacer.
2. **Value network:** Aproxima el valor del estado $V(s)$, que se usa para calcular el error PG y actualizar la *Policy network* para incrementar la probabilidad de las acciones con buenos valores de ventaja.

Este método se puede ver como una fusión de las dos familias *policy-based* (Policy-network) y *value-based* (Value-network). Por lo que toma las buenas propiedades de cada familia. Entender esta arquitectura es esencial para entender los algoritmos del estado del arte.

Asynchronous Advantage Actor-Critic (A3C)

Gracias a la propuesta del artículo [Mnih et al., 2016], se usa una mejora en el método A2C que se basa en añadirle un comportamiento asíncrono. De esta forma se obtiene el método A3C, que es uno de los más utilizados por los ingenieros de RL en la actualidad.

Un enfoque usado para mejorar la estabilidad de los métodos de la familia PG es usar **varios entornos en paralelo**. La razón de esto es para evitar que las observaciones sean dependientes y ayuda a que las observaciones sean distribuidas de forma idéntica.

Estos entornos paralelos proporcionan transiciones entre sí, y todos ellos explotan *la política actual*. La mayor **desventaja** es la **ineficiencia de muestreo** ya que se tiran todas las experiencias del agente después de haberlas usado solo una vez en el entrenamiento. Sin embargo, ya se comentó que el problema de ineficiencia de muestreo no es un inconveniente para el entorno de este problema.

El método A2C hace esta paralelización de entornos creando una pila de estos y almacenando el estado para cada uno, de esta forma se accede a ellos en orden. Sin embargo, esto se puede mejorar si se usa el multiproceso, es decir, paralelizando los entornos con distintos núcleos CPU. De esta manera, en A3C se pueden seguir las siguientes estrategias:

- **Paralelismo de datos:** Se pueden tener distintos procesos donde cada uno se comunica con uno o más de ellos proporcionando transiciones o ejemplos. Estas transiciones o ejemplos se reúnen juntas en un único proceso de entrenamiento, donde se actualizan los pesos de la NN y se difunde al resto de entornos.
- **Paralelismo de gradientes:** Se tienen distintos procesos calculando los gradientes para sus propios ejemplos de entrenamientos. Después estos gradientes se agrupan juntos para realizar la actualización SGD en un único proceso.

Si se tiene una máquina potente con muchas GPUs, se recomienda usar el paralelismo de gradientes ya que mejoraría mucho el proceso. Si solo se tiene una GPU, se recomienda el primer enfoque ya que es más fácil de implementar.

3.2.3. Métodos para espacios continuos de acciones

Como se mencionó anteriormente, los métodos *policy-based* se pueden adaptar mejor y más fácilmente a un espacio continuo de acciones. Este apartado se explica en el caso de que se quiera añadir complejidad al agente en el caso de querer transformar la variable **Cantidad** del espacio de acciones a muchos intervalos, dejándola cercana a una variable continua. De esa forma el espacio de acciones tendrá una variabilidad bastante alta, por lo que puede que estos métodos funcionen mejor.

Deep Deterministic Policy Gradient (DDPG)

El método DDPG propuesto en el artículo [Lillicrap et al., 2015] propone una adaptación del método *Deep Q-Learning* para que sea exitoso en espacios de acciones continuos.

Es una variación del método A2C, pero con la propiedad de ser *off-policy*. En este método, la política es **determinista**, que significa que proporciona directamente al agente qué acción realizar. No es como el método A2C, donde la política era estocástica, es decir, se devolvía una distribución de probabilidad para las acciones discretas.

Al ser determinista, esto permite aplicar la regla de la cadena al *valor-Q* y maximizar Q . Como se tienen dos NNs, se tiene la función $\mu(s)$ para la *actor network*, que convierte un estado en una acción; y $Q(s,a)$ para el *crit network* que da el *valor-Q*. De esta forma se puede introducir la función del *actor* dentro de la función *crit network* de la forma $Q(s, \mu(s))$. De esta forma, aplicando la regla de la cadena, se obtiene el gradiente:

$$\nabla_a Q(s, a) \nabla_{\theta_\mu} \mu(s)$$

Por lo tanto, lo que diferencia a los métodos DDPG y A2C es la forma en que se usa la *crit network*. Como la política es determinista, se pueden calcular los gradientes desde Q y hacer que todo el sistema en su conjunto sea diferenciable y optimizable mediante SGD. Con A2C, el *crit network* es opcional de usar.

Como ahora la política es determinista, se ha perdido la exploración innata del método A2C. Esto se soluciona añadiendo ruido a las acciones devueltas por la *actor network*.

Distributed Distributional Deterministic Policy Gradients (D4PG)

En el artículo [Barth-Maron et al., 2018], se proponen distintas mejoras al método DDPG, que dan lugar a resultados muy buenos, llegando a alcanzar un rendimiento del estado del arte.

Este método mejora la estabilidad, convergencia y *sample efficiency* del método DDPG. La idea principal es reemplazar el único *valor-Q* obtenido por la *crit network* con una distribución de probabilidad. La ecuación de Bellman se reemplaza por el operador de Bellman. Como ahora en vez de un valor, se obtiene una distribución de probabilidad, la función de coste que hay que minimizar mide la distancia entre dos distribuciones (error de distribución TD) usando el operador de Bellman.

Como mejoras adicionales, se aplica la **ecuación de Bellman con n-pasos**, desenrollando así la ecuación para aumentar la velocidad de convergencia. También utiliza el **buffer replay priorizado** en vez de un buffer que toma ejemplos con una distribución uniforme. Estas mejoras coinciden con las propuestas en DQN, por lo que posiblemente vengan inspiradas de ahí. Además, para aumentar la velocidad usa K actores distribuidos de forma paralela.

Twin Delayed DDPG (TD3)

Otro método que también mejora el inicial DDPG es el propuesto en el artículo [Fujimoto et al., 2018]. Un problema típico en el método DDPG es

que la *función-Q* aprendida tiende a sobre-estimar los *valores-Q*. Las tres mejoras empleadas son las siguientes:

1. TD3 aprende dos *funciones-Q* en vez de una (de ahí el nombre *twin*). Después usa el menor de estos dos *valores-Q* para formar las etiquetas de las funciones de coste usando Bellman.
2. En el modelo A2C, las actualizaciones de la política (*actor network*) y el valor *target network* están muy relacionadas. Las estimaciones del valor divergen cuando la política es pobre y la política será pobre cuando las estimaciones del valor sean imprecisas. Por lo tanto, para reducir la varianza se actualiza la política con menos frecuencia que la *función-Q*.
3. Se introduce una estrategia de suavizado en la *función valor*. Esto se hace porque las políticas deterministas pueden quedar atascadas en mínimos locales.

3.2.4. Métodos Trust Regions

Estos métodos mejoran la estabilidad de los métodos de políticas estocásticas, como es el A2C. Por lo tanto, mejoran la estabilidad de la actualización de la política durante el entrenamiento.

Por una parte, se quiere entrenar lo más rápido posible, haciendo largos pasos durante la optimización SGD. Por otro lado, una actualización grande en la política puede arruinar la política aprendida hasta ese momento, ya que puede ser irreversible o costaría mucho arreglarlo. Por ello, se han hecho investigaciones sobre posibles soluciones, y una factible es intentar estimar el efecto que va a tener la actualización de la política en un futuro.

Proximal Policy Optimization (PPO)

El primer método de esta familia es Trust Region Policy Optimization (TRPO), pero tiene el inconveniente de que es extremadamente difícil de implementar en la práctica.

Unos años más tarde, *OpenAI* publicó el artículo [Schulman et al., 2017], donde describe la familia de algoritmos PPO, que suponen una simplificación del algoritmo TRPO. Mientras que TRPO resuelve este problema con métodos de segundo-orden, PPO los resuelve con métodos de primer-orden usando varios trucos para mantener las nuevas políticas cercanas a las antiguas. Además, el rendimiento de ambos métodos es teóricamente parecido y PPO es más fácil de implementar. El objetivo de PPO es intentar hacer

la mayor mejora en el paso de optimización de la política, sin pasarnos de zancada dando lugar a un colapso

La principal mejora de PPO sobre el método A3C (que parece el más potente visto hasta ahora dentro de un espacio de acciones discreto) es cambiar la expresión usada para estimar el PG. En vez de usar el gradiente de la probabilidad logarítmica de la acción tomada, PPO usa la distancia entre la nueva y la antigua política escalada por la ventaja de la acción. Sin embargo, si se intenta constantemente maximizar este valor, se puede llegar a actualizaciones muy largas para los pesos de la *policy network*. Para limitar esta actualización, la distancia entre las nuevas y viejas políticas se define como:

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{vieja}(a|s)}$$

De esta forma se limita la distancia entre las viejas y nuevas políticas a un intervalo $[1 - \epsilon, 1 + \epsilon]$, donde ϵ es un hiperparámetro. Este intervalo limitante hace que se pierda la motivación en incrementar a extremos la actualización de la política para obtener mejores recompensas. Esto se puede ver como una forma de hacer más **conservador** al agente y minimizando el **riesgo** en la compra y venta de acciones. Ambas características son necesarias y ventajosas en el *trading* real.

PPO se ha probado en un conjunto de tareas desafiantes del estado del arte dando lugar a resultados sorprendentes con gran simplicidad. Además, puede usarse en espacios de acciones continuos y soporta una paralelización como A3C.

Capítulo 4

Solución propuesta

En el capítulo anterior se han visto distintas técnicas de RL, desde las más clásicas a las más modernas, justificando las ventajas y desventajas de cada una para ser aplicadas a este problema.

También en el capítulo 2 se mostraron las ideas generales de afrontar este problema con técnicas de RL. En este capítulo se va a mostrar exactamente qué componente de esas ideas generales se han usado en este entorno.

4.1. Referentes

Lo primero de todo, es recordar que el enfoque para desarrollar un agente de *trading* con RL no ha sido idea del presente autor. La idea del desarrollo de este proyecto ha venido inspirada y ayudada por las dos siguientes referencias:

1. El libro [Lapan, 2018] muestra un ejemplo de un entorno simple de RL para *trading* con la técnica DQN. Es un entorno simple porque usa pocas variables de la serie temporal multivariante y solo permite al agente comprar y vender una acción. Además, solo puede tener una acción como máximo. Esto le lleva que el problema sea tan simple que necesite una barbaridad de pasos para llegar a una convergencia. La motivación del desarrollo del presente TFM ha sido inspirada prácticamente por este libro.
2. El artículo [King, 2019] ha dado algunas ideas para la correcta normalización de las observaciones de la serie. Es el artículo gracias al que se ha conocido la recompensa *Incremental Profit* y la librería *stable baselines*, que permite fácilmente usar las técnicas más avanzadas de RL.

4.2. Innovaciones y mejoras propuestas

A parte de las ideas tomadas por las anteriores citas mencionadas, el presente autor ha añadido mejoras para que el modelo de RL sea capaz de enfrentarse a varias series e incluso ganar beneficio con una comisión añadida a la compra y venta de acciones. Estas mejoras se explicarán en siguientes secciones de forma detallada. Las mejoras se listan de forma abreviada a continuación:

1. A partir de la serie temporal multivariante, se han añadido nuevas variables a las observaciones del entorno para que el agente sea capaz de aprender con series **ruidosas**.
2. Se demuestra más adelante que la granularidad de las series en intervalos de un minuto es demasiado elevada y da problemas de mucho ruido también. Se propone usar series con una granularidad mayor como una hora.
3. Se ha implementado una novedosa idea para combatir a la comisión en estos problemas. Esta idea se basa en introducir en el entorno de RL una **distorsión** en la toma de decisiones del agente para que el agente compre y venda menos, de esta forma la comisión tendrá menos impacto. Esta distorsión se basa en establecer en cada paso una probabilidad menor de que el agente compre o venda, dando lugar a una política más refinada.
4. Se ha probado la robustez del enfoque de RL ante distintas series del Mercado Financiero. Ninguna de las dos citas anteriores hace estas pruebas, ya que ellos solo muestran sus resultados con una serie. Este tipo de pruebas es necesario para demostrar que esta sistema es general ante cualquier serie.

4.3. Implementación del sistema de RL

Como se ha comentado antes, en el capítulo 2 se ha mostrado un enfoque general para resolver este problema con técnicas de RL. Entonces, en esta sección se va a definir cual es el enfoque específico usado.

4.3.1. Agente

En cuanto al agente automático de inversión, se le han impuesto unas **normas** para que su entrenamiento no sea muy enrevesado. Las normas que sigue son las siguientes:

- El agente **no pierde** en ningún momento, es decir, puede realizar acciones inválidas y seguir jugando. Por lo tanto, si no tiene acciones y quiere vender, en ese caso es como si hubiera hecho la acción *no hacer nada*. Y de forma alternativa, si quiere comprar pero no tiene capital suficiente, es como si hiciera la acción *no hacer nada*.
- El **episodio** en el que el agente recorre el entrenamiento está definido desde el principio de la serie temporal hasta el final.
- Se establece un **máximo** al dinero, número de acciones y valor del portafolio que el agente puede tener en su disposición. Esto se hace únicamente para facilitar la normalización de las observaciones del entorno, por lo tanto, se puede decir que son límites **simbólicos**, es decir, el agente en realidad tiene la posibilidad de superar esos límites. Como se ponen de forma simbólica, se indica un máximo bastante alto, que superarlo sería bastante complicado.

4.3.2. Estados/Observaciones

En la sección 2.2.3 se definió de una forma general la **observación del estado**. Sin embargo, en esta sección vamos a definir exactamente la observación.

Variables de la observación

La observación se divide en dos tipos de información:

1. **La información obtenida únicamente a partir de la serie temporal multivariante.** Esto se refiere a las variables tenidas en cuenta de la serie temporal. A continuación se comentarán cuales son.
2. **El estado actual de la cuenta del agente.** Este estado está formado por el **capital o dinero del agente**, el **número de acciones** y su **valor de portafolio**. Estos estados son esenciales para que el agente vea su situación actual y sepa el número de acciones que puede comprar y vender.

Las variables principales usadas de la serie temporal son la mayoría existentes y algunas que se han creado a partir de ellas:

1. **Open:** Valor de entrada de la empresa en ese intervalo temporal.
2. **High:** Valor máximo de la empresa en la vela.
3. **Low:** Valor mínimo de la empresa en la vela

4. **Close:** Valor final de la empresa en la vela. Hay que mencionar que esta variable es la usada por el entorno para representar el precio de ese instante. Es el precio más representativo ya que ha sido el último de la vela, que se supone que es justo el momento en el que el agente realiza la acción.
5. **Volume:** Volumen de negociaciones cotizadas en la vela. Normalmente representa el grosor de esta. Es un campo importante porque muestra la liquidez del mercado.
6. **Number of Trades:** Número de negociaciones o *tradeos* realizados en la vela. Por lógica, este campo está directamente relacionado con el volumen.
7. **Weighted average price:** Es la media ponderada del precio en esa vela.
8. **High relativo:** Es la diferencia porcentual entre el precio máximo de la vela y el precio de entrada. Esto ayuda a interpretar mejor la forma de la vela para la NN. Se puede definir como:

$$highRelativo = \frac{(High - Open)}{Open}$$

9. **Low relativo:** Es la diferencia porcentual entre el precio mínimo de la vela y el precio de entrada. Como en la variable anterior, ayuda a interpretar mejor la forma de la vela para la NN. Se puede definir como:

$$lowRelativo = \frac{(Low - Open)}{Open}$$

10. **Close relativo:** Es la diferencia porcentual entre el precio final de la vela y el precio de entrada. Esto ayuda a interpretar mejor la forma de la vela para la NN. Se puede definir como:

$$closeRelativo = \frac{(Close - Open)}{Open}$$

A estas variables principales, se les añaden unas **variables alternativas** que miran más atrás en el tiempo para que se pueda combatir el ruido. En el próximo capítulo se mostrará en un ejemplo como su uso puede mejorar mucho la convergencia. Las variables añadidas han sido para intervalos de tiempo de doce horas de antelación, veinticuatro horas, una semana, un mes y todo el tiempo desde el estado actual hacia atrás:

- **Máximo del precio.** Puede ayudar para que el agente aprenda los máximos locales.

- **Mínimo del precio.** Puede ayudar para que el agente aprenda los mínimos locales.
- **Media del precio.** Da una idea general del precio, tomando en cuenta extremos, por lo que no es robusto al ruido.
- **Mediana del precio.** Da una idea general del precio robusta al ruido, ya que no tiene en cuenta extremos.
- **Desviación estándar del precio.** Es un buen indicador de la varianza de los datos en ese rango.

Estas variables también se añaden con la idea de que un inversor no mira únicamente los datos de unas pocas horas atrás, sino que se fija en una semana e incluso meses más allá. Estas variables se pueden crear fácilmente usando la función *rolling* y *expanding* de la librería *Pandas*.

Como se explicó en la sección 2.2.3, estas variables de los dos tipos de información hay que usarlas para la vela actual y para las N últimas. Este tamaño de valores solo puede ser procesado por una NN de forma correcta. Se podría intentar aplicar en *Q-Learning* poniendo muchas menos velas anteriores y discretizando con muchísimos estados, pero no es la estrategia más fácil y viable para este problema. Además, la *dimensionalidad* de la observación es bastante alta. Por ejemplo, si se usan 25 velas anteriores y en total hay 38 variables, se estaría hablando de 988 ($26 * 38$) valores por observación.

Normalización de la observación

Como en los métodos más avanzados de RL hay que usar una NN para la aproximación de la función 2.4, es necesario normalizar las observaciones del agente. La normalización que se aplica es en el rango $[0,1]$, ya que las capas de la NN usan la función de activación *Relu*.

Como se ha visto antes, la observación se descompone en dos tipos de información:

- **Observaciones de la serie:** Se normaliza siguiendo la siguiente fórmula estándar de normalización:

$$x_{norm} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

- **Estado interno del agente:** Las tres variables capital, número de acciones y valor de portafolio se normalizan en cada paso por su máximo establecido en el entorno. Usan la misma fórmula anterior pero en

este caso el máximo es fijo y el mínimo siempre es cero. De esta forma, el valor normalizado se obtiene de la siguiente fórmula:

$$x_{norm} = \frac{x_i}{\max(x)}$$

Tal y como se menciona en [King, 2019], la normalización de las observaciones de la serie **no se debe** aplicar a los datos antes de introducirlos en el entorno. Esto es porque las observaciones en un estado determinado se deben normalizar únicamente respecto a las observaciones de todos los estados anteriores, ya que en un entorno de **tiempo real**, solo se conocen los ejemplos pasados, y usando límites puestos por datos todavía no observados provocaría sesgos en el agente.

Por ejemplo, si se normalizara con anterioridad y el precio en el que se encuentra es cercano a cero, el agente ya sabe que ese precio es de los mínimos de la serie, por lo que lo mas seguro es que compraría en ese estado. Por ello, la normalización usada es de tipo **incremental**, como se muestra gráficamente en la Figura 4.1. La idea es en cada paso del episodio, normalizar la serie respecto a los datos pasados únicamente. Esto provocaría un funcionamiento del entorno mucho más lento, ya que se normaliza en cada paso, pero es la forma correcta de abordar el problema.

Al intentar simular un entorno en **tiempo real**, se puede hablar de un problema relacionado con el área del **Aprendizaje Incremental**, que no es exactamente Minería de Flujo de Datos, ya que los datos se reutilizan en el entrenamiento en cada episodio. En este caso se estaría usando una **ventana de tamaño incremental** para la normalización. Hay que remarcar que esta ventana se reinicia cuando acaba el episodio, no se deja crecer de forma indefinida, cosa que no tendría sentido en RL.

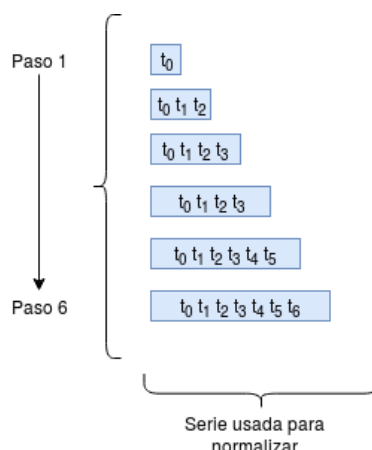


Figura 4.1: Normalización incremental

4.3.3. Acciones

Se ha decidido que el **espacio de acciones** a usar en este sistema el espacio de acciones **Multi-Discreto** mostrado en la Figura 2.6. Lo bueno de este espacio de acciones es que se asemeja mucho a la realidad, ya que la cantidad expresa también el riesgo de la acción en ese momento. Además, de esta forma el agente puede aprender a ganar mucho más beneficio, ya que el riesgo es mayor que si solo se compra o vende una acción como máximo.

A este espacio de acciones se le puede cambiar el número de fracciones para simplificar el problema o añadirle complejidad. Hay que tener cuidado de que si se le añade muchas fracciones, el espacio puede pasar a ser **Multi-Continuo**, y como se ha visto en la teoría, no todas las técnicas de RL soportan este tipo de espacio de acciones.

Hay que mencionar, que el número de acciones que se compra o vende debe pertenecer al conjunto de los números naturales \mathbb{N} . Por lo tanto, cuando se elige un número de acciones con la fracción correspondiente, este número se redondea. Esta restricción se introduce porque en la **Bolsa de Valores** solo se permite comprar y vender acciones enteras.

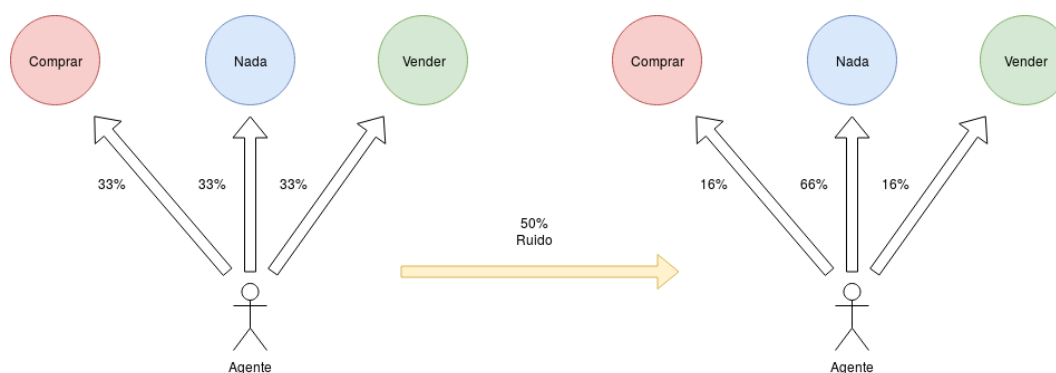


Figura 4.2: Ejemplo de Ruido en el espacio de acciones

Ruido en el espacio de acciones

Como novedad en este enfoque de RL, se añade una variable aleatoria para reducir el número de compras y ventas por parte del agente. Si se activa esta variable aleatoria con una probabilidad determinada, cada vez que el agente quiera comprar y vender, se generará un número aleatorio en el intervalo $[0,1]$. Por lo tanto, si el número es mayor a esa probabilidad, el agente comprará o venderá.

De esta forma se estaría aprendiendo un modelo donde comprará un porcentaje menos de veces según esa probabilidad determinada y aprenderá

un modelo más refinado debido a que tiene que forzar la compra o venta en puntos específicos.

En la Figura 4.2 se muestra gráficamente esta inclusión de ruido para que se entienda mejor. En el gráfico izquierdo se muestra al agente en un entorno sin ruido, donde según si espacio de acciones tiene una probabilidad igual de comprar, vender o no hacer nada. Entonces, si se introduce un 50 %, como se muestra en el gráfico derecho, se duplicaría la probabilidad de que el agente no hiciera nada y se quitaría la mitad de la probabilidad a comprar y vender.

Es **importante** indicar, que cuando se usa este ruido es necesario entrenar sin comisión. Esto es porque si se le introduce también comisión al agente, se le están metiendo demasiadas restricciones que le harán más imposible todavía aprender una buena política. Las comisiones se introducirán en el entorno a la hora de evaluar los modelos.

4.3.4. Recompensa

En el capítulo 2 se muestra un listado de tres recompensas disponibles para este problema. Ahí se explica una por una con sus ventajas y desventajas, demostrando que la mejor de esas tres para este problema es el *Incremental Profit*.

En el gráfico de la Figura 2.7 se justificaba porque esta recompensa es muy buena para indicarle al agente como comprar en mínimos locales y vender en máximos locales. Además, al dar una recompensa con mayor frecuencia que *Realized PnL*, dará una mayor rapidez en la convergencia.

4.3.5. Entorno

Una vez definidas la mayoría de componentes del sistema RL, cabe definir como se ha construido el **entorno**, que es el que engloba a todas las anteriores.

El entorno de *trading* se ha construido con la librería **Gym** de *Python* creada por la compañía OpenAI. Lo bueno de esta librería es que permite construir entornos personalizados usando unos *wrappers* proporcionados.

El entorno se puede ver como una clase de *Python* formado por las siguientes funciones principales:

- *Reset*: Este método se llama cada vez que se empieza un episodio, donde se inicializan las variables necesarias. Por lo tanto, se llama cuando se termina la serie y cuando se empieza a entrenar. Esta función devuelve la primera observación del episodio.

- *Step*: Se llama cada vez que el agente aplica una acción en el entorno. Por lo tanto, tiene como argumento la acción del agente. Esta función devuelve la tupla (*obs*, *reward*, *done*, *info*), donde *obs* es la observación del próximo estado, *reward* es la recompensa obtenida por la acción aplicada, *done* indica si el episodio se ha acabado o no, e *info* indica información sobre el estado y la acción.

De este modo, cuando el agente de RL ha aprendido con un método cualquiera de RL, en el pseudocódigo 1 se muestra la forma de uso del entorno.

Algoritmo 1: Uso del entorno

```
env = TradingEnvironment(dataFrameTrain);
obs = env.reset();
while !done do
    | action = modelo.predict(obs);
    | obs, reward, done, info = env.step(action);
end
```

Además de los métodos mencionados antes, el entorno tiene unas variables que lo definen y que pueden cambiarse al inicializarlo. Las variables que lo forman son las siguientes:

- **Precios históricos**: Es la serie temporal únicamente con la variable *Close* que se usará para establecer el precio de la acción en la variable actual.
- **Observaciones de la serie temporal normalizadas de forma incremental**: Se introducen las observaciones normalizadas de forma incremental previamente al entorno. Esto se hace para ahorrar tiempo y no tener que normalizar mientras el entorno está funcionando.
- **Dinero inicial**: Es el capital que se le proporciona al agente inicialmente para usarlo en el episodio. Por defecto se ha puesto 10.000. Lo ideal es cambiar el capital inicial dependiendo de los precios de la serie de valores a usar y cuando se incluye comisión en el problema.
- **Comisión del broker**: Es la comisión que se aplica en cada compra y venta del agente. Por defecto se ha puesto 0.1 %.
- **Últimas velas observadas**: Se refiere a las N últimas velas al tiempo actual que se añadirán a la observación para que el agente al aprender pueda saber más sobre el estado actual. Por defecto se añaden las últimas 25 velas.
- **Cantidad**: Es el número de fracciones que se usa en el espacio de acciones. Por defecto es 10.

- **Máximo número de acciones:** Se refiere al número máximo de acciones que puede tener el agente en su portafolio. Esto se usa para facilitar la normalización de las observaciones únicamente, ya que no se aplica directamente al agente. Por defecto es 10000. Si la serie temporal tiene unos precios normalmente muy bajos, se debería aumentar este límite para que no se supere con facilidad.
- **Dinero máximo:** Se refiere al dinero máximo que puede tener el agente en su portafolio. Esto se usa para facilitar la normalización de las observaciones únicamente, ya que no se aplica directamente al agente. Por defecto es diez veces más que el dinero inicial. Es una cantidad alta que no se espera superar.
- **Valor de portafolio máximo:** Se refiere al valor del portafolio máximo que puede tener el agente en su portafolio. Esto se usa para facilitar la normalización de las observaciones únicamente, ya que no se aplica directamente al agente. Por defecto es también diez veces más que el dinero inicial.
- **Espacio de acciones:** El espacio de acciones en formato *Multi-Discreto* que su dimensionalidad depende de la variable *Cantidad*.
- **Espacio de observaciones:** Para el espacio de observaciones se tiene que especificar su dimensión y formato al inicializar el entorno. Es necesario cambiar su formato según la NN usada en la técnica de RL.

Una vez definidos los elementos principales del entorno, en la Figura 4.3 se muestra gráficamente la interacción del agente con el entorno. Como se puede ver, al agente solo le llega la recompensa y la observación del estado al que se ha llegado debido a su acción que ha enviado al entorno. Esa es la forma de interactuar principal para que el agente aprenda en este entorno. El agente básicamente es la NN que está aproximando la función de la Figura 2.4.

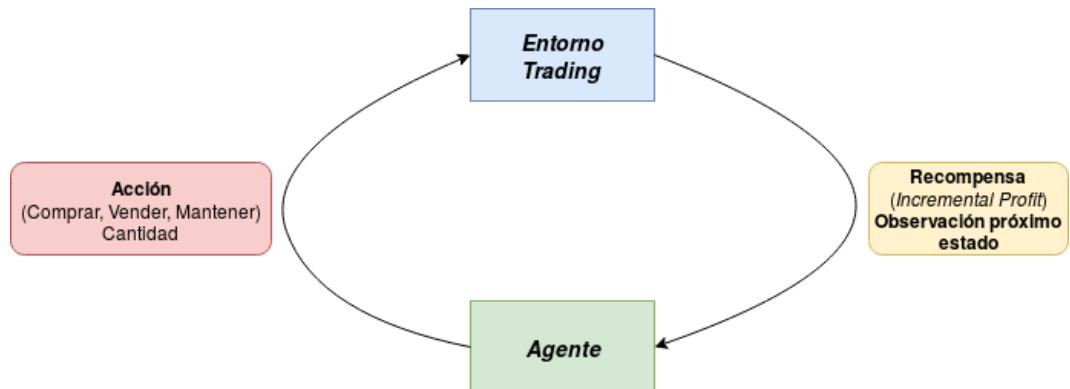


Figura 4.3: Interacción del agente y el entorno

4.3.6. Técnicas de RL usadas

En el capítulo 3 se justificó que **PPO** era la técnica más adecuada y potente de las expuestas en ese capítulo. También se muestra que **A3C** es una de las técnicas más utilizadas por los ingenieros de RL. Hay que recordar que la diferencia principal entre estas dos técnicas es que A3C tiene como objetivo maximizar la recompensa de forma voraz, mientras que PPO es más conservador en la forma que obtiene las recompensas. Por lo tanto, se proponen usar estas dos técnicas para justificar si lo expuesto en la **teoría** en base a PPO se cumple en la **práctica** y ver cual de las dos técnicas funciona mejor en este problema.

Estos dos algoritmos se han implementado tomándolos directamente de la librería **stable baselines**, la cual permite conectar fácilmente técnicas de RL que usan *Deep Learning* con entornos contruidos con la librería *Gym*, como es este caso.

Las técnicas A3C y PPO de esta librería son muy potentes y versátiles gracias a las siguientes ventajas:

1. Permite usar **espacios de acciones multidiscretos y continuos**. Aunque según la teoría, A3C no está preparado para espacio de acciones continuos, mientras que PPO sí.
2. Da la capacidad de usar **procesos paralelos** en el entrenamiento. Esto mejora mucho la rapidez de este.
3. Permite usar NNs con **capas recurrentes**. En este caso, la capa recurrente es una LSTM (Long short-term Memory).

Estas dos técnicas permiten **dos arquitecturas de NNs** distintas para aplicarlas a este problema con esta librería. Además, estos necesitan un formato de los datos también variable. Estas técnicas usan la arquitectura *actor-critic* de la Figura 3.3. Como se usan dos NNs (Policy NN y Value NN), hay que definir la arquitectura que se va a usar en cada una, pero para no complicar el problema mucho, se ha optado por usar la misma arquitectura en ambas NNs.

Las dos arquitecturas de las NNs usadas son las siguientes:

- **Perceptrón multicapa (MLP)**: En este caso, el MLP básicamente se basa en una serie de capas ocultas que están totalmente conectadas entre sí.
- **LSTM con MLP para extracción de características**: Esta arquitectura está formada por varios tipos de capas principales. La primera parte está formada por una serie de capas LSTM, para procesar los

datos secuenciales, donde un lote de datos es un vector de tres dimensiones. Después, esta librería ofrece la opción de añadir una capa *Batch Normalization* para normalizar los pesos, que ayuda a estabilizar el entrenamiento. Después, se aplica una capa *Flatten* con el fin de aplanar el vector de tres dimensiones a dos dimensiones. Por último, se aplica un MLP a este vector aplanado para la extracción de características.

Una vez definidas las arquitecturas, hay que explicar el formato de las observaciones para meterlo en cada arquitectura, ya que **ambas requieren formatos distintos**.

La arquitectura MLP, al requerir cada observación en vectores de una dimensión, hay que introducir todos los valores de cada observación del entorno RL en una única fila. Por lo tanto, se ponen las variables de cada vela e información del agente de forma ordenada como indica la Figura 4.4a. En el caso de la arquitectura LSTM, como permite introducir cada observación en vectores de dos dimensiones, cada observación se puede mostrar en una tabla donde cada fila son las variables de las observaciones y cada columna es un instante de tiempo o vela ordenada de pasado a presente. Este es el formato requerido para las capas LSTM en la librería usada, que se puede ver gráficamente en la Figura 4.4b.

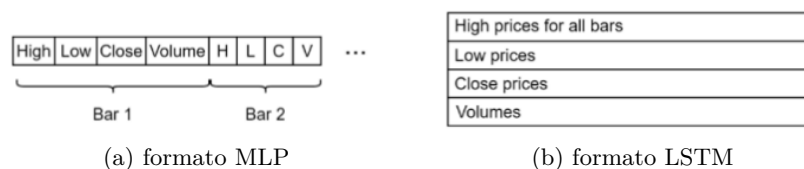


Figura 4.4: Formato de las observaciones

El código de este sistema de RL con el que se han hecho los experimentos del siguiente capítulo se encuentra en el repositorio <https://github.com/pedrajo/TFMTradingRL>.

Capítulo 5

Estudio experimental

En el anterior capítulo se explica como se ha definido el sistema de RL, componente por componente. Entonces, en este capítulo se van a hacer pruebas o experimentos con dos de las técnicas del *estado del arte* de RL que mejor se ajustan a este problema. En la última sección de este capítulo se llevará a cabo un análisis de estos experimentos con sus conclusiones.

5.1. Diseño experimental

En esta sección se van a describir los elementos principales para la preparación de los experimentos. Entre estos elementos están los conjuntos de datos usados, los hiperparámetros usados en las NNs, el particionamiento de los datos y las métricas de evaluación.

5.1.1. Conjunto de datos de los experimentos

Como se comentó en la introducción, la idea es hacer un modelo de RL que sea general ante cualquier serie de valores de cualquier compañía. Por ello, se van a hacer experimentos con tres series de compañías distintas en el intervalo de tiempo muy parecido. Estas tres series se han descargado de la página web firstratedata.com. Lo bueno de esta página es que ofrece de forma gratuita datos a nivel de intradía de muchas entidades financieras.

Ahora, un problema a discutir es la **granularidad** usada en los datos. La granularidad mínima de la que se dispone es de un minuto. Esta granularidad se puede extender hasta una hora para trabajar con valores intradía. Para ver la granularidad que se va a usar en los experimentos, se van a comparar los resultados con una granularidad de un minuto y una hora.

Para esta comprobación se va a usar una serie de Apple desde Julio de

2015 hasta Septiembre de 2015. En las Figuras 5.1a y 5.1b se muestran las series con distinta granularidad. Se puede apreciar que es la misma serie pero en el caso de minuto a minuto hay muchos más **ruido** y muchos más ejemplos como es obvio.

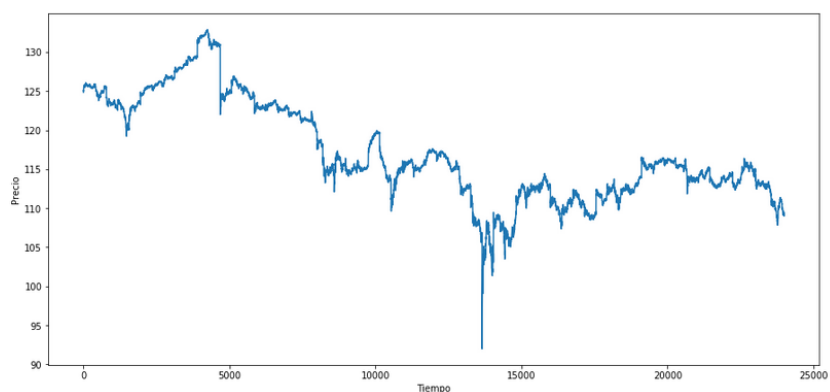
A continuación se va a entrenar un modelo con la técnica PPO, que según la teoría es la técnica más apropiado para este problema, en ambas series. Este modelo PPO utiliza una NN con la arquitectura LSTM con MLP para extracción de características. Los hiperparámetros son los mismos que los que se mostrarán en la sección 5.1.2. Se ha utilizado el entorno con los parámetros por defecto, excepto la comisión, que es 0 %. Como es lógico, el modelo de la serie minuto a minuto necesitará más tiempo para entrenar por tener más ejemplos. Solo con los resultados en el conjunto de entrenamiento es suficiente para ver la gran diferencia. Se va a mostrar para cada serie el *profit*, que básicamente es la diferencia entre el valor del portafolio al inicio y al final de la serie en posesión por el agente.

0 % comisión	Minuto a minuto	Hora a hora
Profit	-10207.02	20301.56
Pasos	1000000	350000

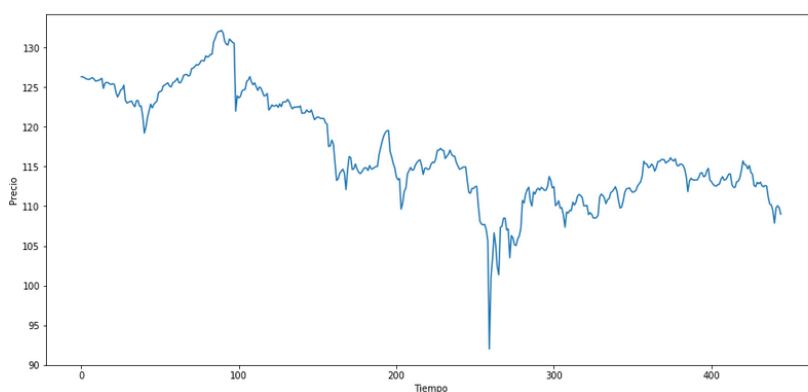
Cuadro 5.1: Comparación de beneficio con distinta granularidad

Como se puede observar en la Tabla 5.1, el modelo con datos minuto a minuto no es capaz ni de obtener *profit* positivo en el conjunto de entrenamiento sin aplicar comisión. Además en las Figuras 5.1c y 5.1d se muestran las gráficas suavizadas de la recompensa media y en las Figuras 5.1e y 5.1f se muestran los errores de entropía durante el entrenamiento que se han obtenido con el framework *Tensorboard*, que es muy bueno para monitorear el entrenamiento.

Se puede ver como el modelo con granularidad horaria es capaz de aprender en el tiempo una buena recompensa media, que progresivamente va mejorando. Además, el error de entropía es cada vez menor, diciendo esto que el agente cada vez tiene más seguro qué hacer, por lo que está aprendiendo una política determinada. Esto no ocurre en el caso de granularidad minuto a minuto, donde la recompensa no se estabiliza ni parece mejorar en el tiempo. Además, el error de entropía no parece disminuir de forma constante, por lo que parece que el agente no tiene claro lo que hacer. Estos datos dan lugar a la conclusión de que el modelo **no es capaz de encontrar patrones característicos** en los datos con la granularidad minuto a minuto por culpa del **ruido extremo**. También se puede decir que se prefiere menos datos y de calidad a más datos y que sean ruidosos.



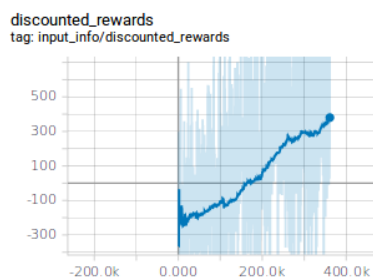
(a) minuto a minuto



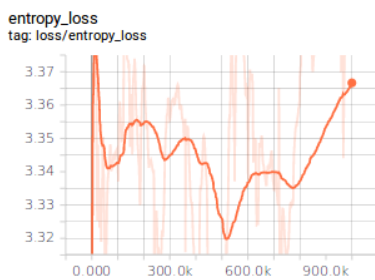
(b) hora a hora



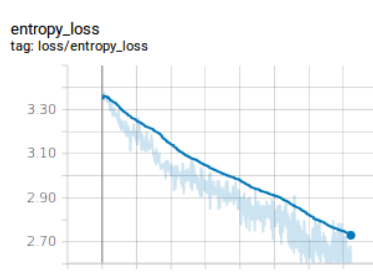
(c) Recompensa minuto a minuto



(d) Recompensa hora a hora



(e) Error de entropía hora a hora



(f) Error de entropía hora a hora

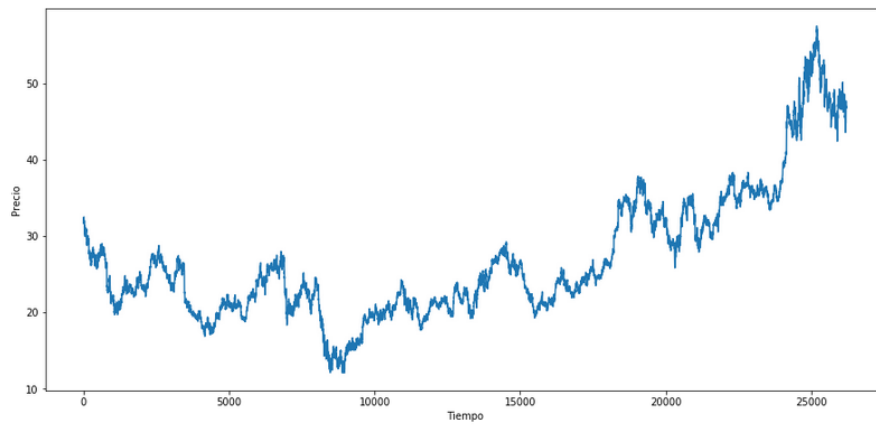
Figura 5.1: Serie de Apple con distinta granularidad

Ahora, con 1 hora de granularidad se pierden muchos más ejemplos en un intervalo determinado, lo que puede dar lugar a un *overfitting* en el conjunto de entrenamiento si se entrena maximizando demasiado la recompensa media. Por lo tanto, hay que buscar un intervalo grande de muchos años para que el agente generalice mejor.

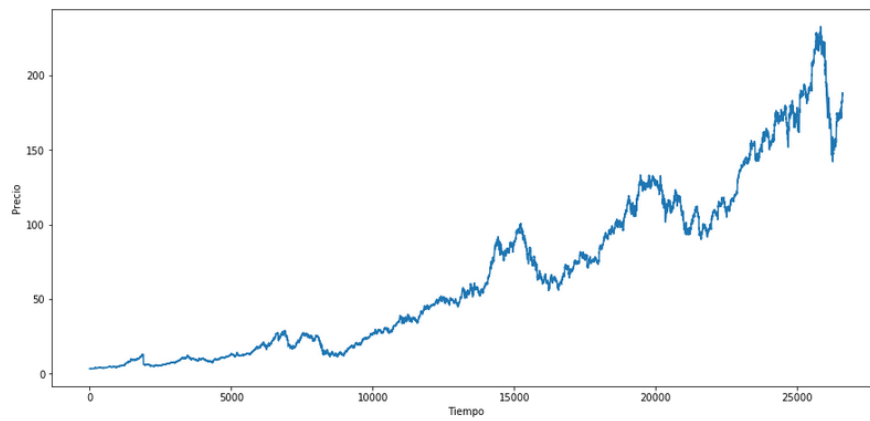
Se han tomado todos los datos históricos disponibles en esta granularidad de las empresas Intel, Apple y Qualcomm. La idea ha sido tomar tres empresas tecnológicas que estén relacionadas de alguna forma. En la Figura 5.2 se muestran las tres series tomando la variable *Close*, que como se ha dicho antes, es la que mejor representa el precio.

Las tres series muestran estas características principales:

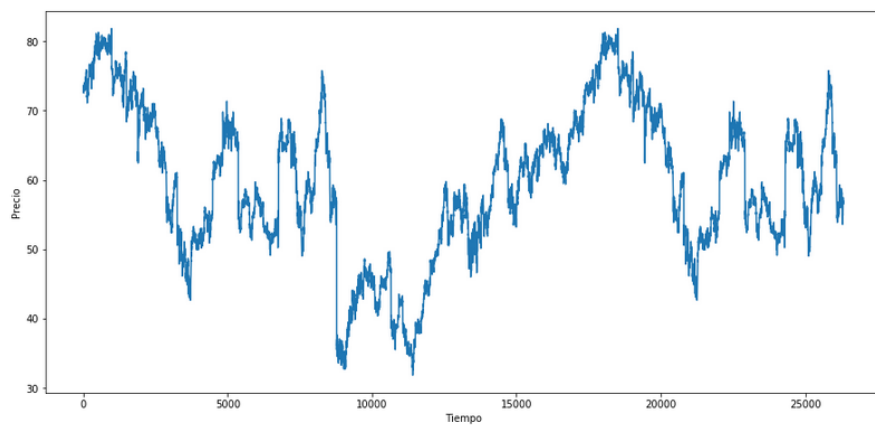
1. **Intel:** Tiene datos desde el 23 de Enero de 2004, hasta el 31 de Diciembre de 2018. Tiene exactamente 26207 ejemplos. Al principio no muestra una tendencia clara, pero a partir de la segunda mitad parece tener una tendencia alcista.
2. **Apple:** Con un intervalo desde el 31 de Enero de 2004, hasta el 18 de Abril de 2019. Tiene 26599 ejemplos. La tendencia es **claramente alcista**, con precios muy cercanos a cero en su inicio y mucho más altos en la época final. Si se compara la segunda mitad de la serie de Apple y la de Intel, se puede ver una correlación entre ambas en alguno de sus picos mayores. Además, a tener cada vez precios mayores, muestra una mayor volatilidad del precio con el paso del tiempo.
3. **Qualcomm:** Tiene un intervalo desde el 2 de Enero de 2004, hasta el 31 de Diciembre de 2018. Tiene 26324 ejemplos. La tendencia no es clara y es una serie con una **volatilidad muy alta**, es decir, los precios varían mucho en el tiempo. Se puede ver a simple vista que es la serie con más **ruido** de las tres.



(a) Intel



(b) Apple



(c) Qualcomm

Figura 5.2: Precios de Intel, Apple y Qualcomm

5.1.2. Hiperparámetros de las NNs

En la sección 4.3.6 se muestran las dos arquitecturas de NNs propuestas. Ahora, se van a mostrar los hiperparámetros finales de estas arquitecturas que se van a usar en los siguientes experimentos.

En las Figuras 5.3a y 5.3b se muestran los hiperparámetros de estas arquitecturas. La justificación de estos parámetros se lista a continuación:

- Con 64 neuronas en las capas LSTM y las totalmente conectadas se obtienen unos buenos resultados sin provocar *overfitting*. Esta es la misma explicación por la que solo se añaden dos capas de cada una.
- Estas capas tienen la función de activación *Relu* ya que es una función estándar dentro de las NNs que dan buenos resultados.
- En la arquitectura que usa capas LSTM se usa también la capa *Batch Normalization* ya que facilita la convergencia del modelo y la estabilidad del entrenamiento.

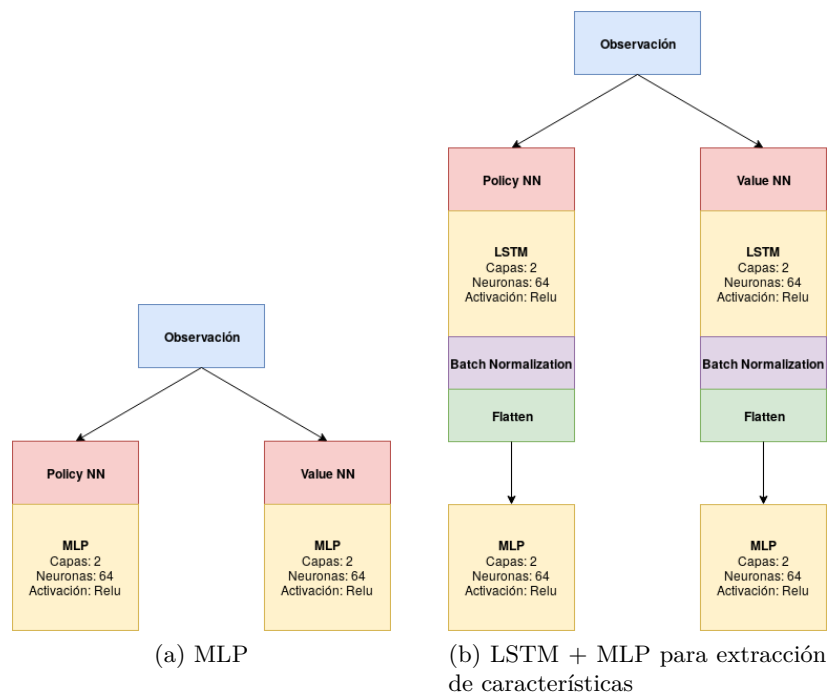


Figura 5.3: Hiperparámetros usados en las NNs

5.1.3. Particionamiento del conjunto de datos

Para entrenar el agente y evaluar su política aprendida se ha dividido la serie respectivamente en **el conjunto de entrenamiento y prueba**. Simplemente se ha dividido el **80 %** primero de la serie en entrenamiento y el restante **20 %** en prueba. Se han utilizado la mayoría de ejemplos para el entrenamiento porque no se disponen de muchos datos y se quiere que el agente aprenda una política bien generalizada.

Se ha añadido una separación entre ambos conjuntos de **50** ejemplos, por posibles problemas de solapamientos en las predicciones.

5.1.4. Medidas de rendimiento

Para evaluar los resultados y comparar los modelos decidiendo cuales han sido mejores o peores, se necesitan las siguientes medidas:

1. **Profit por hora:** Esta **métrica** es importante para saber cuanto dinero puede ganar o perder el agente en un intervalo de tiempo determinado. Esta métrica se define según la siguiente fórmula:

$$\frac{profit}{hora} = \frac{valor_{portafolio}(t_{fin}) - valor_{portafolio}(t_{inicio})}{horas_{serie}}$$

Se ha puesto el beneficio en función del tiempo porque no es lo mismo ganar un beneficio en intervalos de tiempo distintos. Además, como el conjunto de entrenamiento y prueba tienen distinto tamaño, se ha querido poner una métrica que los ponga a ambos por igual. Esta métrica se puede interpretar como la cantidad de dinero que gana o pierde por hora.

2. **Visualización de la política de compra y ventas:** En los distintos conjuntos de entrenamiento y prueba, se dibujarán las compras y ventas del agente. Esto sirve para determinar si el agente ha pensado en una política con sentido.
3. **Gráfico de recompensas medias:** Este gráfico muestra las recompensas medias que obtiene el agente durante su entrenamiento. El eje X hace referencia a los pasos del entrenamiento, y el eje Y al valor medio de la recompensa.
4. **Gráfico de profit por episodio:** Muestra un gráfico con el *profit* obtenido en cada episodio. El *profit* por episodio se puede ver también como la suma de todas las recompensas en ese episodio.
5. **Gráfico de error de entropía:** Muestra el error de entropía a lo largo del entrenamiento. Un valor alto muestra indecisión en la toma

de acciones del agente, mientras que un valor bajo muestra que el agente tiene seguro qué acción tomar en cada paso. La convergencia en una política durante el entrenamiento viene dada en esta gráfica.

El complemento de todos los componentes es imprescindible para evaluar un agente. En cuanto a los gráficos de recompensas medias y *profit* por episodio, en cada experimento solo se muestra el más conveniente de ambos. Ambas gráficas tienen un significado muy parecido, pero hay casos que son difíciles de visualizar.

Por último, al final de los resultados se mostrará una tabla global del **profit relativo por hora** para comparar los beneficios en todas las series con los distintos modelos. Esta métrica se define como:

$$\frac{profit_{relativo}}{hora} = \frac{\frac{valor_{portafolio}(t_{fin}) - valor_{portafolio}(t_{inicio})}{valor_{portafolio}(t_{inicio})}}{horas_{serie}} \quad (5.1)$$

Con esta métrica se pueden comparar todas las series conjuntamente, ya que no hace falta tener en cuenta la inversión inicial de cada uno. Esta métrica se puede interpretar como el porcentaje de dinero respecto a la inversión inicial que gana por hora.

5.1.5. Estructura y desarrollo de los experimentos

Los resultados de los experimentos se muestran en dos partes principales. Una parte es solo los resultados de modelos sin comisión y otra los resultados de los modelos con comisión. La idea de mostrarlos por separado es que la primera parte es mucho más sencilla que la segunda.

Experimentos sin comisión

Lo primero de todo, va a ser mostrar los resultados para cada serie pero **sin aplicar comisión**. La idea de primero mostrar estos resultados es que si el modelo no aprende sin comisión, pues no va a aprender con comisión ya que el problema es mucho más complejo. Entonces, siempre es mejor desarrollar un proyecto desde el punto más fácil al más difícil.

Para los siguientes experimentos se han usado las siguientes variables del entorno de RL:

1. **Número de velas anteriores:** 25.
2. **Número máximo de acciones:** 10000.
3. **Capital inicial:** 10000.

4. Fracciones de cantidad: 10.

Se han entrenado los agentes con **cinco** procesos paralelos para aumentar la rapidez del entrenamiento. Las tablas de resultados de cada serie se obtienen evaluando los conjuntos con cinco procesos distintos debido a que estos modelos no son deterministas. Luego para obtener el *profit por hora* se ha hecho la media de resultados. Los **gráficos de compra y venta** como las de la Figura 5.5 así como los números de movimientos de las tablas se han sacado con uno de ellos cualquiera, ya que la componente estocástica afecta poco en la proporción de acciones de cada tipo y en la visualización de estas.

Algo importante a remarcar es que para cada serie solo se mostrarán los gráficos de compra y venta más característicos. Al tener cuatro modelos para cada serie y tres series distintas, el análisis se haría eterno si se mostraran todos.

Para demostrar la efectividad de las **variables alternativas** expuestas en la sección 4.3.2, se va a comparar los resultados de la serie de Apple con y sin la utilización de estas variables. En el resto de series se obtendrán únicamente los resultados incluyendo estas variables alternativas. Además, se muestran los experimentos para la serie de Apple **sin tendencia**, con el fin de ver los cambios en el aprendizaje del agente.

En esta fase se entrenarán los modelos PPO y A3C con las arquitecturas MLP y LSTM cada uno, dando lugar a cuatro modelos distintos por resultado.

Experimentos con comisión

En la siguiente fase de resultados se muestran los resultados de las tres series aplicando distintas comisiones. Con el fin de demostrar la efectividad de **la distorsión en el espacio de acciones** definido en la sección 4.3.3, en la serie de Apple se entrenará sin este ruido aplicando una comisión de 0.1 % durante su entrenamiento y luego se entrenará con este ruido sin aplicar comisión. De esta forma se compararán los resultados de utilizar y no utilizar este ruido. En el resto de resultados de las series solo se entrenará con ruido y sin aplicar comisión.

Como se comenta en la sección 4.3.3, para este cambio es **importante** entrenar sin comisión para que el modelo pueda aprender una buena política, sino se quedará estancado. A la hora de evaluar los modelos se le introducirá la comisión correspondiente.

Los resultados se muestran con dos comisiones distintas para probar la robustez del modelo. Las dos comisiones con las que se muestran los resultados se listan a continuación:

- **0.1 %**: Es una comisión **relativamente baja** en el mundo de los *brokers* financieros.
- **0.25 %**: Es una comisión **estándar** dentro de los *brokers*.

En esta fase solo se va a entrenar el modelo PPO con la arquitectura LSTM. Esto es, adelantándolo con antelación, porque es claramente el mejor modelo de los experimentos sin comisión. Por lo tanto, si los otros modelos no son buenos en la fase sin comisión, en la fase con comisión es prácticamente imposible que lo sean.

Para los siguientes experimentos se han usado las siguientes variables del entorno de RL:

1. **Número de velas anteriores**: 25.
2. **Número máximo de acciones**: 10000.
3. **Capital inicial**: 25000. Se ha aumentado el capital inicial para que el agente sea capaz de llegar al final de la serie sin perder todo el dinero por las comisiones.
4. **Fracciones de cantidad**: 4. Se ha reducido la dimensionalidad del espacio de acciones para que sea más fácil para el agente aprender dado que la complejidad del problema ahora es mayor.

Como en la fase anterior, el modelo para cada serie se entrena con **cinco** procesos paralelos y los resultados se muestran de la misma forma.

5.2. Resultados experimentales

En esta sección se van a mostrar los resultados obtenidos por los experimentos diseñados según la sección anterior.

5.2.1. Resultados sin comisión

Serie de Apple sin variables alternativas

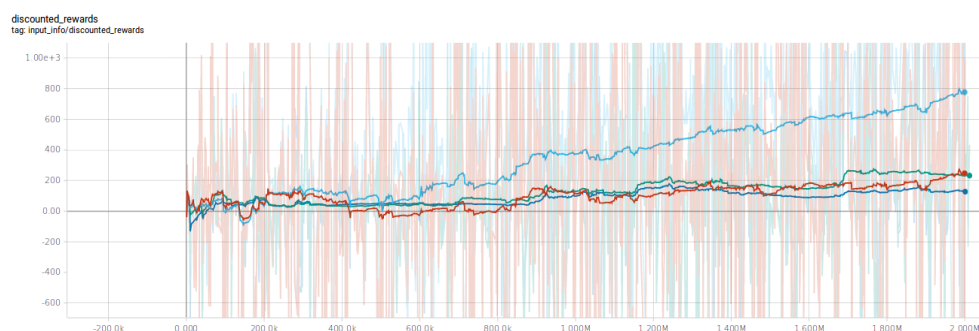
Primeramente, en esta serie solo se han utilizado en la observación con las variables principales, es decir, sin añadir las variables alternativas para mirar más atrás en el tiempo. En esta serie se han entrenado todos los modelos con dos millones de pasos, que es un número considerable para que en todos los casos converjan los modelos. Los resultados de los conjuntos de entrenamiento y prueba se muestran en las Tablas 5.2 y 5.3 respectivamente.

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	13.93	14.27	5.00	7.34
Pasos	2000000	2000000	2000000	2000000
Compras	7	59	3549	8709
Ventas	0	1	2449	2563
Nada	21270	21217	15279	10005

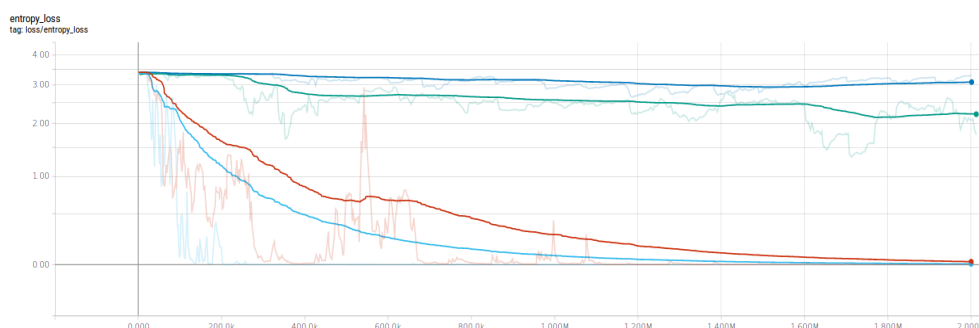
Cuadro 5.2: Resultados de Apple en el conjunto de entrenamiento

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	1.43	1.35	0.79	1.12
Compras	8	29	772	1663
Ventas	0	3	550	694
Nada	5262	5238	3948	2913

Cuadro 5.3: Resultados de Apple en el conjunto de prueba



(a) Recompensas medias



(b) Error de entropía

Figura 5.4: Gráficos del entrenamiento de Apple

También se muestran los gráficos del progreso del entrenamiento en la Figura 5.4. En estos gráficos los colores son A3C con LSTM el azul celeste, A3C con MLP el rojo, PPO con LSTM el verde y PPO con MLP el azul oscuro.

En la Figura 5.5 se han añadido las compras y las ventas a las series de entrenamiento y prueba para PPO y A3C usando solo LSTM. La razón por la que se muestran solo los modelos LSTM es porque tienen menos compras y ventas, por lo que se puede visualizar más fácilmente.

Serie de Apple sin tendencia

Antes de eliminar la tendencia se ha transformado únicamente el conjunto de entrenamiento con un **logaritmo**, ya que la volatilidad de los precios aumenta con el paso del tiempo. Para quitar la tendencia se ha quitado la pendiente obtenida al aproximar la función con una *regresión lineal*. El resultado de la transformación en el conjunto de entrenamiento se puede ver en la Figura 5.6.

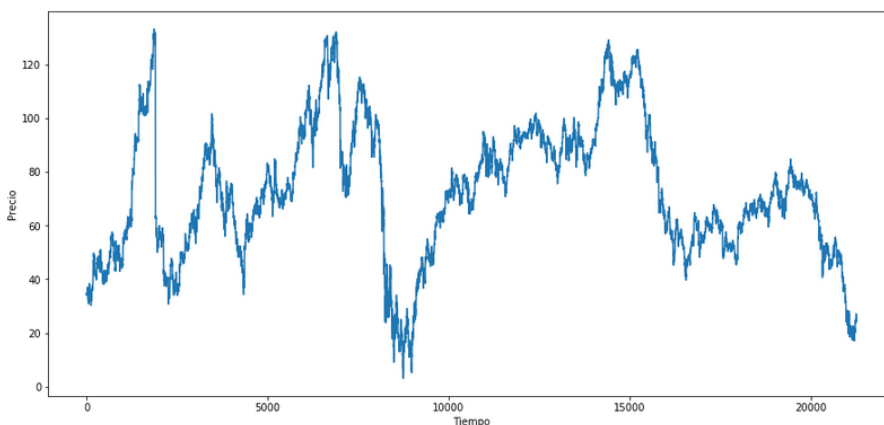


Figura 5.6: Conjunto de entrenamiento de Apple transformado

En la Figura 5.8a se pueden ver el gráfico de recompensas medias del modelo A3C con LSTM **sin las variables alternativas** propuestas. La comparación en el uso de estas nuevas variables se puede ver en la Figura 5.8b mostrando el *profit* por episodio para cada modelo, donde la línea azul es el modelo con las nuevas variables añadidas y la línea naranja el modelo sin esas variables.

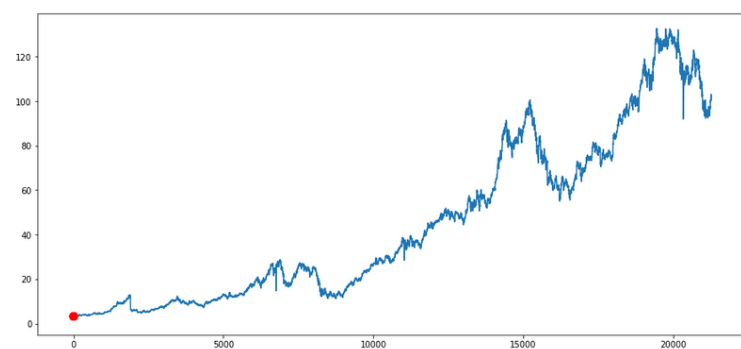
Una vez expuesta la comparación entre ambos modelos en cuanto al uso de estas variables alternativas, se muestran en las Tablas 5.4 y 5.5 los resultados obtenidos con estas nuevas variables para el conjunto de entrenamiento y prueba respectivamente.



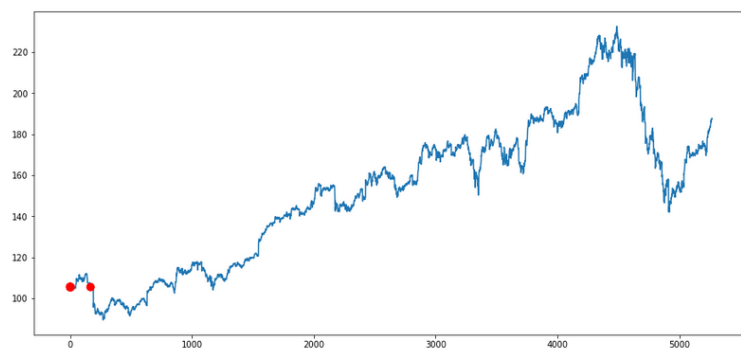
(a) Conjunto de entrenamiento PPO LSTM



(b) Conjunto de prueba PPO LSTM



(c) Conjunto de entrenamiento A3C LSTM



(d) Conjunto de prueba A3C LSTM

Figura 5.5: Compra y venta en series de Apple. Punto rojos: *Compras*. Puntos verdes: *Ventas*. Tamaño del punto: *Ponderado por la cantidad*. Eje *X*: *Tiempo*. Eje *Y*: *Precio*

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	0.27	-0.13	0.79	0.77
Pasos	2000000	2000000	2000000	2000000
Compras	1557	34	2123	10446
Ventas	6044	0	4468	4442
Nada	13676	21243	14686	6389

Cuadro 5.4: Resultados de Apple en el conjunto de entrenamiento con variables alternativas y sin tendencia

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	0.23	1.39	0.81	0.97
Compras	391	16	732	2266
Ventas	1252	0	590	1115
Nada	3627	5254	3948	1889

Cuadro 5.5: Resultados de Apple en el conjunto de prueba con variables alternativas y sin tendencia

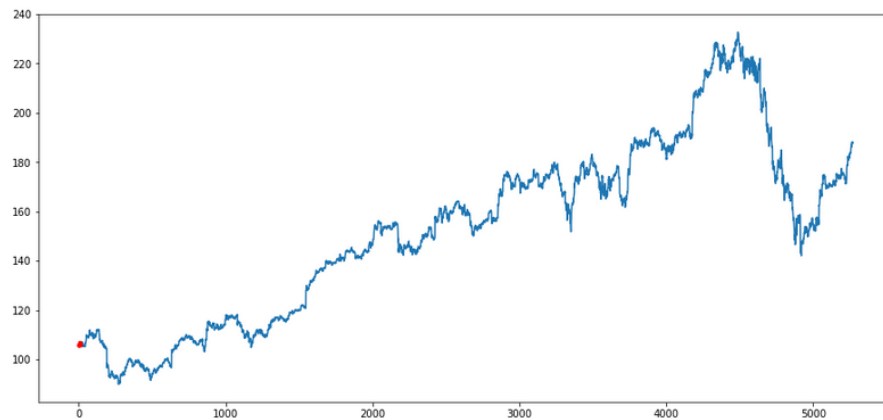


Figura 5.7: Compras y ventas de A3C con MLP en el conjunto de prueba de Apple. Punto rojos: *Compras*. Puntos verdes: *Ventas*. Tamaño del punto: *Ponderado por la cantidad*. Eje X: *Tiempo*. Eje Y: *Precio*

En el gráfico 5.8c se puede ver el *profit* por episodio durante el entrenamiento en estos cuatro modelos. El naranja es A3C con MLP, el azul es el anterior visto A3C con LSTM, el verde es PPO con LSTM y el gris PPO con MLP. En la Figura 5.7 se muestra únicamente el gráfico de compra y

venta para el modelo A3C con MLP con el fin de evaluar su política. Solo se muestra este modelo porque es el que más dudas implican sus resultados de las tablas anteriores.

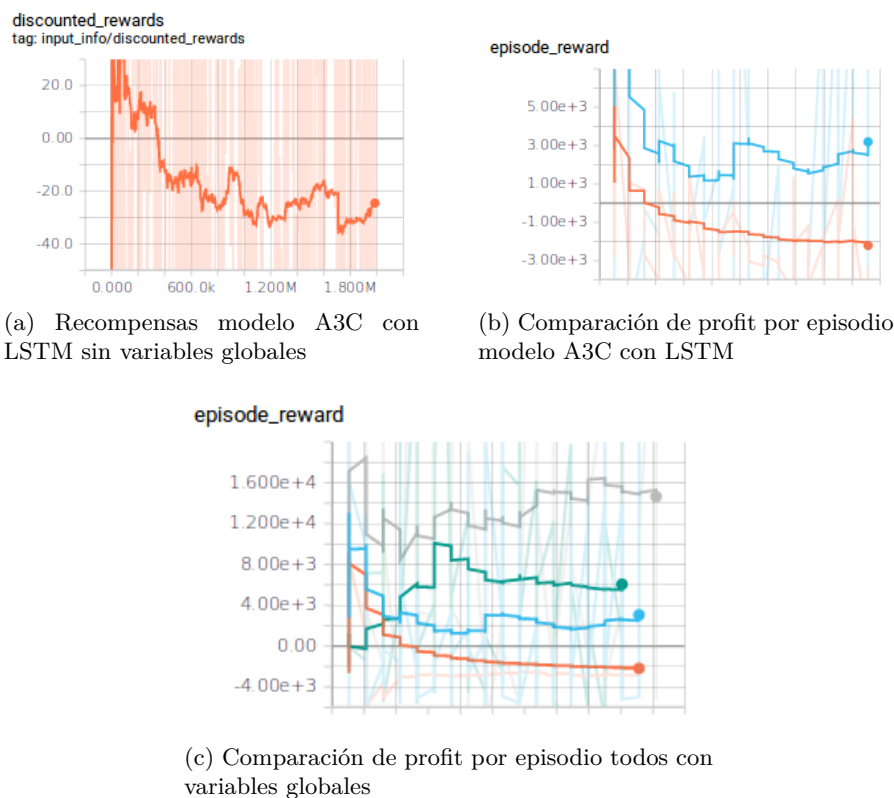
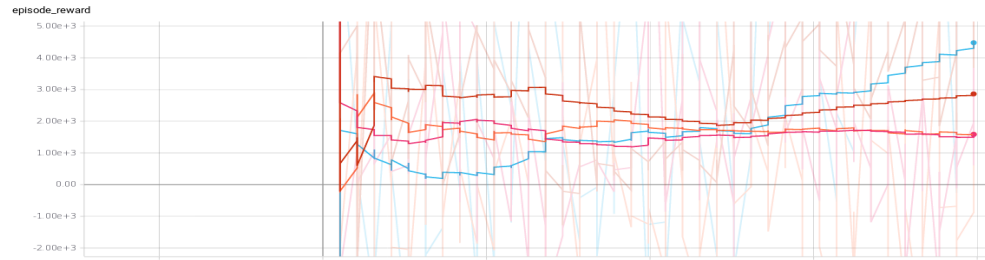


Figura 5.8: Gráficos de entrenamiento en serie de Apple transformada

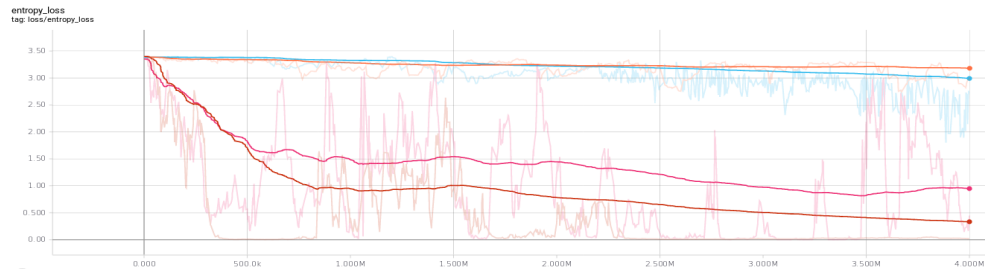
Se recuerda que de aquí en adelante los experimentos incluirán en la observación las **variables alternativas** probadas anteriormente.

Serie de Intel

Los resultados en los conjuntos de entrenamiento y prueba se muestra en las Tablas 5.6 y 5.7 respectivamente. En la Figura 5.9 se muestran los gráficos de los datos relevantes del entrenamiento, teniendo A3C con LSTM en color rosa, A3C con MLP con color Rojo, PPO con LSTM con color azur y PPO con MLP con color naranja. Finalmente, para terminar la evaluación de las políticas aprendidas por los modelos más característicos, se muestran sus gráficos de compras y ventas en la Figura 5.10.



(a) Profit por episodio



(b) Error de entropía

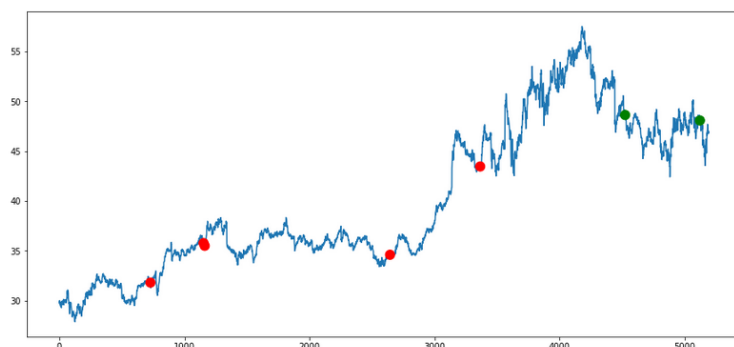
Figura 5.9: Gráficos del entrenamiento de Intel

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	0.09	0.15	0.67	0.07
Pasos	4000000	4000000	4000000	4000000
Compras	524	22	2446	2641
Ventas	185	2	976	4018
Nada	20254	20939	17541	14304

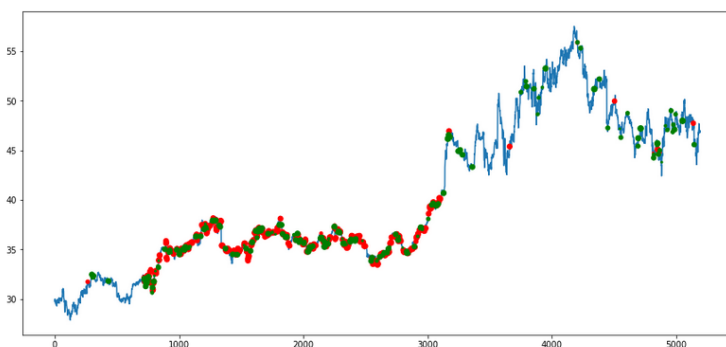
Cuadro 5.6: Resultados de Intel en el conjunto de entrenamiento

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	0.99	0.53	0.69	0.43
Compras	148	5	503	611
Ventas	57	2	202	982
Nada	4987	5185	4487	3599

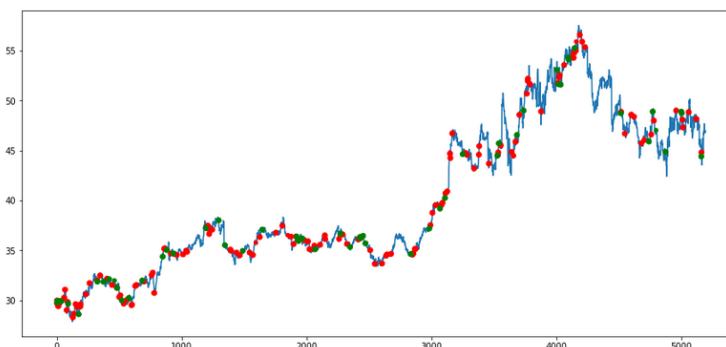
Cuadro 5.7: Resultados de Intel en el conjunto de prueba



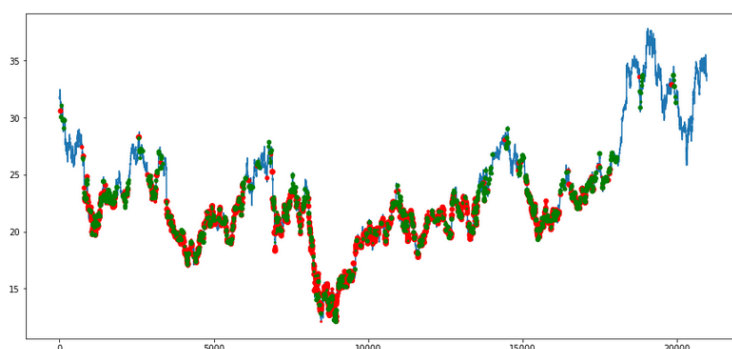
(a) Conjunto de prueba A3C MLP



(b) Conjunto de prueba PPO LSTM



(c) Conjunto de prueba A3C LSTM



(d) conjunto de entrenamiento PPO LSTM

Figura 5.10: Compra y venta en series de Intel. Punto rojos: *Compras*. Puntos verdes: *Ventas*. Tamaño del punto: *Ponderado por la cantidad*. Eje *X*: *Tiempo*. Eje *Y*: *Precio*

Serie de Qualcomm

Esta es la última serie de las propuestas. Los resultados en ambos conjuntos se muestran en las Tablas 5.8 y 5.9. La evolución del entrenamiento se puede ver en la Figura 5.11, donde PPO con LSTM es el naranja, PPO con MLP es el azul oscuro, A3C con LSTM es el azul claro y A3C con MLP es el rojo. Para terminar de evaluar las políticas de cada modelo, se muestra en la Figura 5.12 las compras y ventas de los modelos más característicos.

	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	-0.14	-0.14	0.20	-0.09
Pasos	3100000	1000000	3100000	8000000
Compras	6	7	5162	2259
Ventas	0	0	4650	723
Nada	21052	21051	11246	18076

Cuadro 5.8: Resultados de Qualcomm en el conjunto de entrenamiento

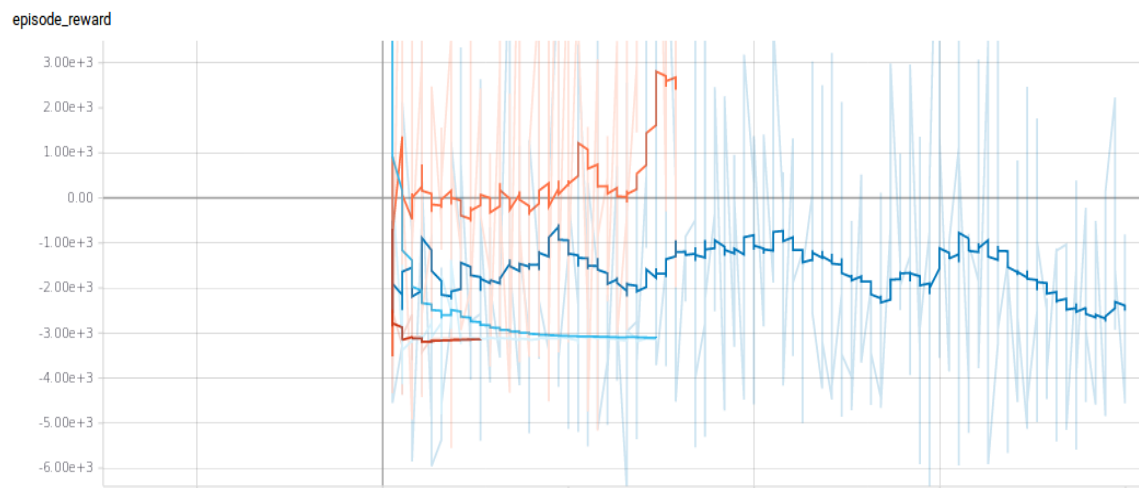
	A3C LSTM	A3C MLP	PPO LSTM	PPO MLP
Profit/hora	0.44	0.44	1.31	0.56
Compras	6	6	1390	648
Ventas	0	0	1295	189
Nada	5209	5209	2530	4378

Cuadro 5.9: Resultados de Qualcomm en el conjunto de prueba

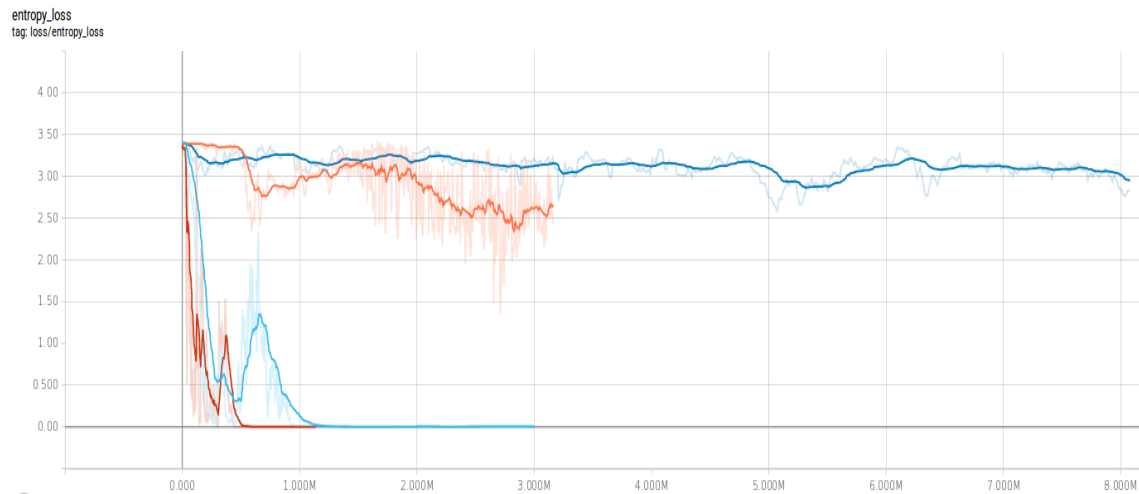
5.2.2. Resultados con comisión

Serie de Apple sin tendencia

Como se hizo en el segundo caso de Apple sin comisión, se va a quitar aquí también la tendencia a la serie de la misma forma. Primeramente se va a mostrar el progreso de recompensas medias del entrenamiento con 0.1 % de comisión. Este progreso de recompensas medias del entrenamiento se puede ver en la Figura 5.13.

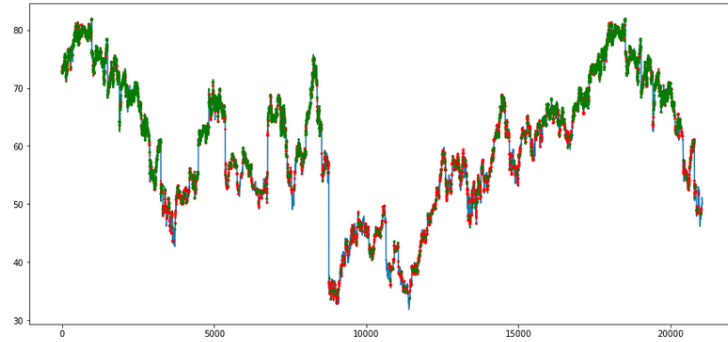


(a) Profit por episodio



(b) Error de entropía

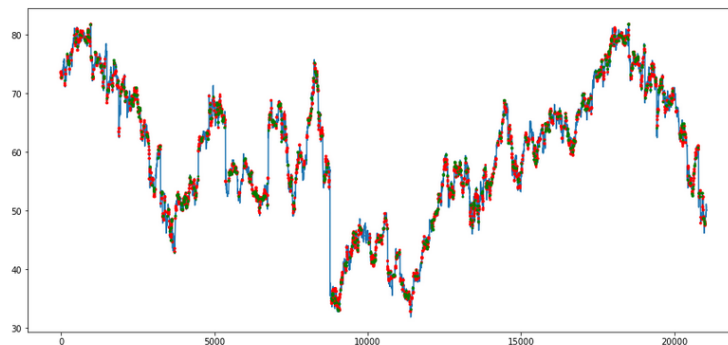
Figura 5.11: Gráficos del entrenamiento de Qualcomm



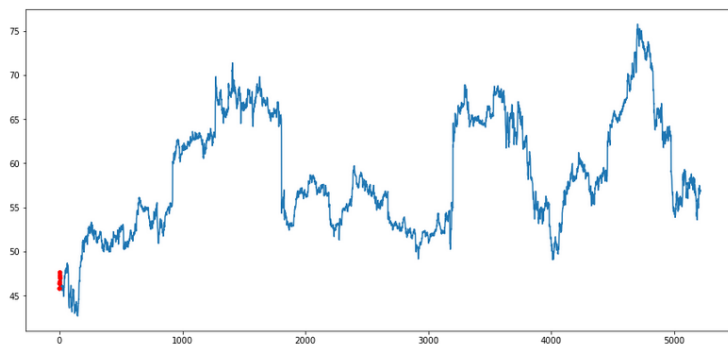
(a) Conjunto de entrenamiento PPO LSTM



(b) Conjunto de prueba PPO LSTM



(c) Conjunto de entrenamiento PPO MLP



(d) Conjunto de prueba en ambos A3C

Figura 5.12: Compra y venta en series de Qualcomm. Punto rojos: *Compras*. Puntos verdes: *Ventas*. Tamaño del punto: *Ponderado por la cantidad*. Eje *X*: *Tiempo*. Eje *Y*: *Precio*

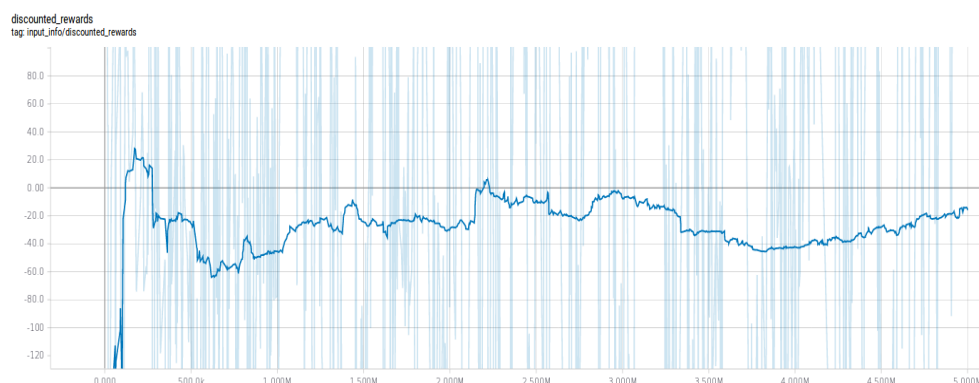


Figura 5.13: Recompensas medias serie de Apple con 0.1 % de comisión

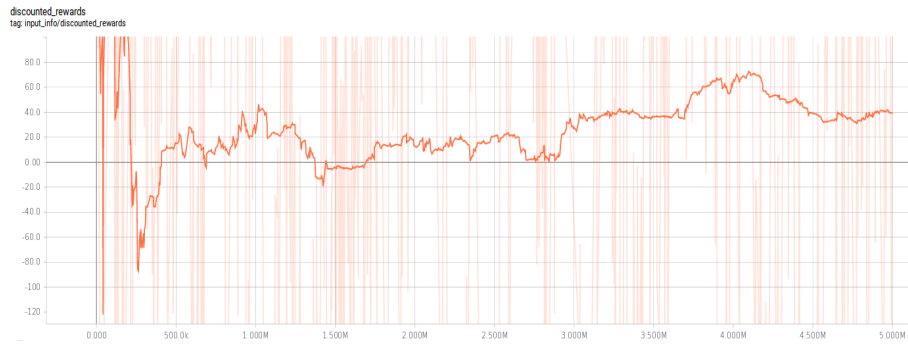
A continuación, se va a introducir el **ruido** en el espacio de acciones comentado en la sección 4.3.3. En la Figura 5.14a se pueden ver las recompensas medias obtenidas a lo largo del entrenamiento usando un ruido del 90 %. Se han probado con distintos niveles de ruido y este ha dado el mejor resultado para esta serie. Mientras que en el Gráfico 5.14b se muestra la evolución del error de entropía, el cual parece estabilizarse en un punto bajo a partir de aproximadamente el paso 4500000.

En la Tabla 5.10 se enseñan los resultados obtenidos con distintas comisiones para ver el impacto de estas en el beneficio obtenido. Para confirmar la buena política aprendida por el agente, hay que ver la serie con sus compras y ventas en los distintos puntos para los conjuntos de entrenamiento y prueba en las Figuras 5.14c y 5.14d respectivamente.

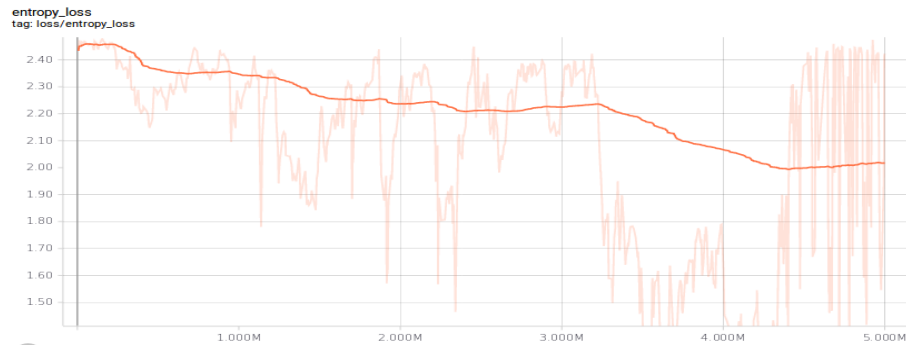
	Conjunto de entrenamiento	Conjunto de prueba
Profit/hora 0.1 %	5.26	1.45
Profit/hora 0.25 %	0.16	0.46
Compras	543	99
Ventas	494	145
Nada	20240	5026

Cuadro 5.10: Resultados de Apple sin tendencia con 90 % de ruido

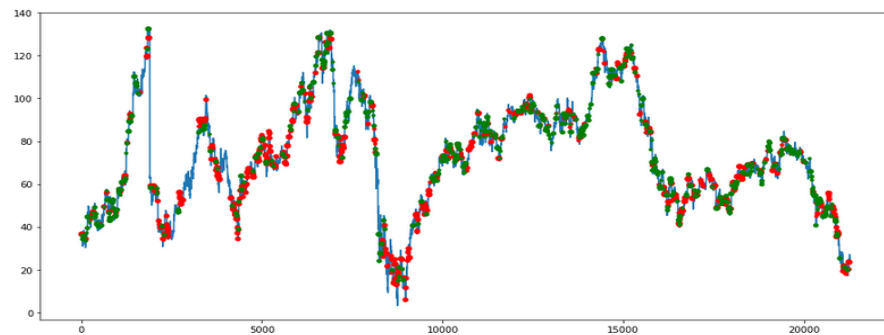
Una vez mostrada la comparación de usar y no usar el ruido dentro del espacio de acciones, en los siguientes experimentos siempre se va a usar esta distorsión del espacio de acciones.



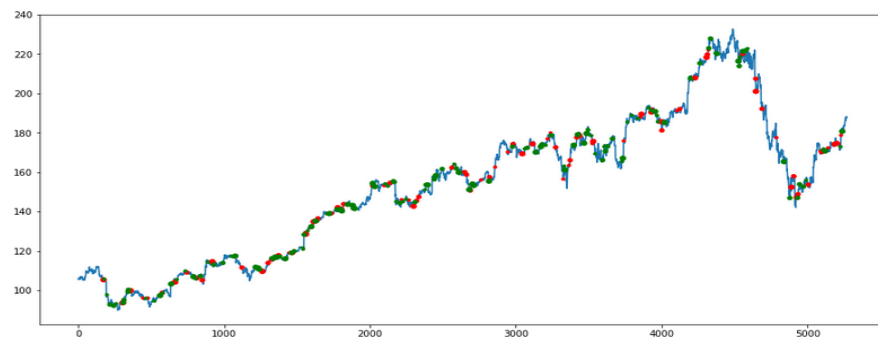
(a) Recompensas medias



(b) Error de entropía



(c) Compras y ventas en conjunto de entrenamiento



(d) Compras y ventas en conjunto de prueba

Figura 5.14: Gráficos de Apple con 90% de ruido. *Gráficos de compras y ventas* \Rightarrow Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio

Serie de Intel

En la segunda serie de los experimentos se ha probado un entrenamiento con distintos porcentajes de ruido, dando lugar a que el mejor resultado se ha conseguido con un porcentaje del 80 %. Los resultados para ambos conjuntos se muestran en la Tabla 5.11.

	Conjunto de entrenamiento	Conjunto de prueba
Profit/hora 0.1 %	0.39	1.11
Profit/hora 0.25 %	-0.21	0.86
Compras	581	224
Ventas	528	288
Nada	19854	4680

Cuadro 5.11: Resultados de Intel con 80 % de ruido

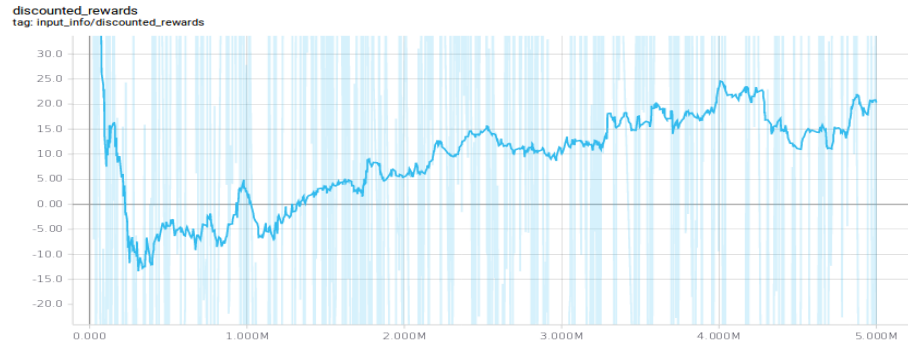
En la Figura 5.15a se muestran el gráfico de recompensa media obtenido en el entrenamiento de esta serie, mientras que en la Figura 5.15b se muestra la evolución del error de entropía.

Falta por conocer el resultado de las políticas gráficamente en sus compras y ventas para los conjuntos de entrenamiento y prueba. Esto se puede ver respectivamente en las Figuras 5.15c y 5.15d.

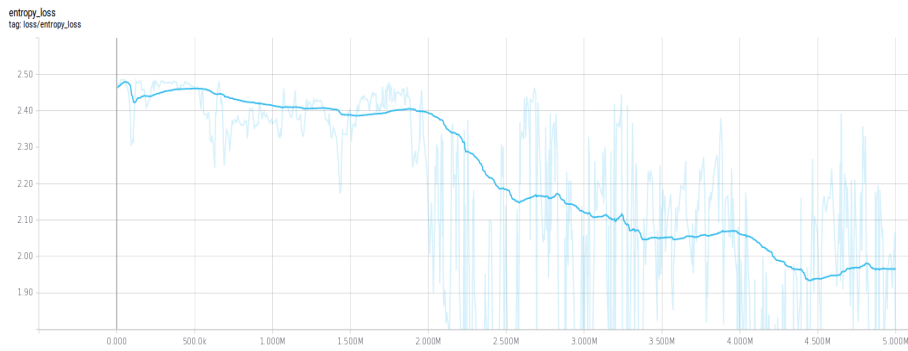
Serie de Qualcomm

En la última serie de las propuestas solo se ha entrenado en dos millones de pasos, ya que el agente empeoraba a partir de ese punto. El mejor porcentaje de ruido encontrado ha sido 85 %. La Tabla 5.12 refleja los resultados en ambos conjuntos.

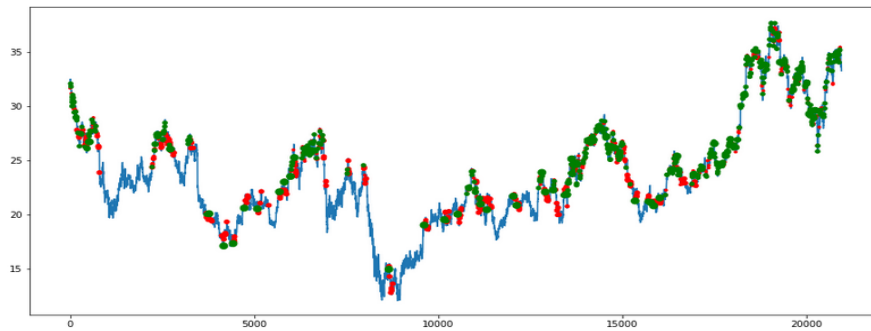
La evolución de las recompensas medias del entrenamiento se muestra en la Figura 5.16a, mientras que la del error de entropía se muestran en la Figura 5.16b. Falta por ver gráficamente como se desenvuelve esta política en ambos conjuntos. En las Figuras 5.16c y 5.16d se puede ver este desempeño en ambas series.



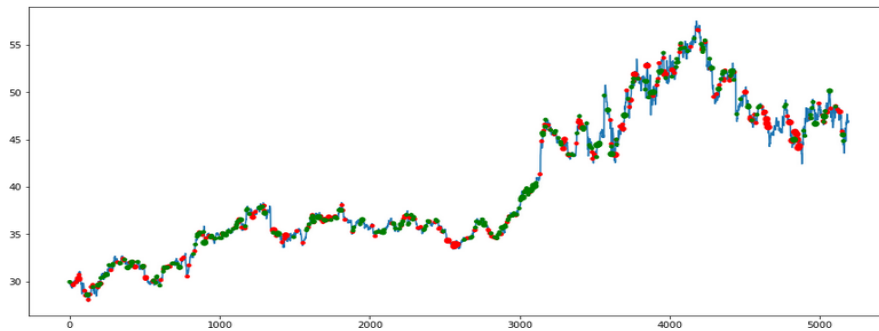
(a) Recompensas medias



(b) Error de entropía



(c) Conjunto de entrenamiento



(d) Conjunto de prueba

Figura 5.15: Gráficos de Intel con 80 % de ruido. *Gráficos de compras y ventas \Rightarrow Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio*

	Conjunto de entrenamiento	Conjunto de prueba
Profit/hora 0.1 %	0.03	0.91
Profit/hora 0.25 %	-0.33	0.28
Compras	729	139
Ventas	792	306
Nada	19537	4770

Cuadro 5.12: Resultados de Qualcomm con 85 % de ruido

5.2.3. Resultados globales de los experimentos

Como resultado final de los experimentos, se van a mostrar dos tablas que engloban todos los *profits relativos por hora* para cada serie. Solo se va a utilizar el modelo PPO con la arquitectura LSTM en este caso, ya que ha sido el único modelo usado en todos los experimentos.

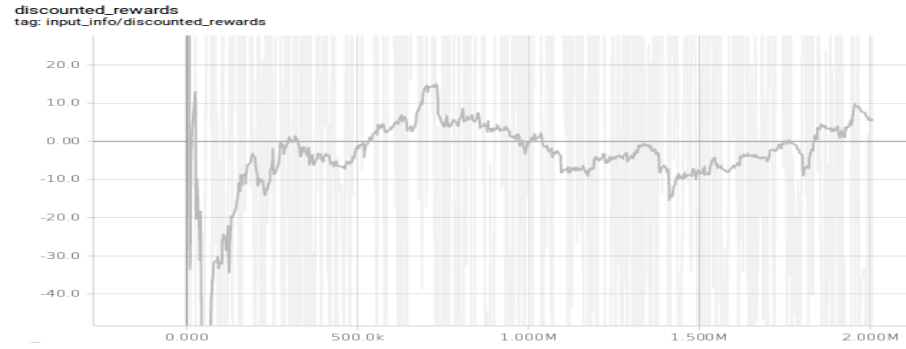
En las Tablas 5.13 y 5.14 se muestran estos resultados para los conjuntos de entrenamiento y prueba respectivamente. En cuanto a la serie de Apple, solo se muestran los resultados de la serie sin tendencia y habiendo introducido en las observaciones las variables alternativas.

	Apple	Intel	Qualcomm
0 %	7.942e-5	6.775e-5	2.073e-5
0.1 %	2.107e-4	1.586e-5	1.421e-6
0.25 %	6.771e-6	-8.484e-6	-1.325e-5

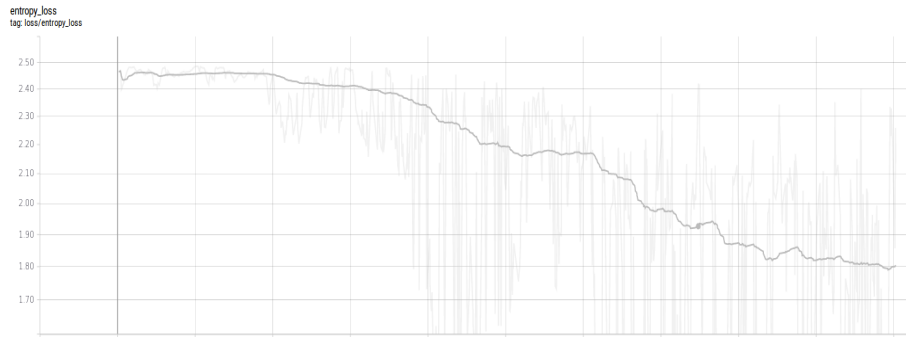
Cuadro 5.13: Resultados globales series profit relativo por hora para conjuntos de entrenamiento

	Apple	Intel	Qualcomm
0 %	7.995e-5	6.926e-5	1.315e-4
0.1 %	5.810e-5	4.451e-5	3.654e-5
0.25 %	1.872e-5	3.471e-5	1.153e-5

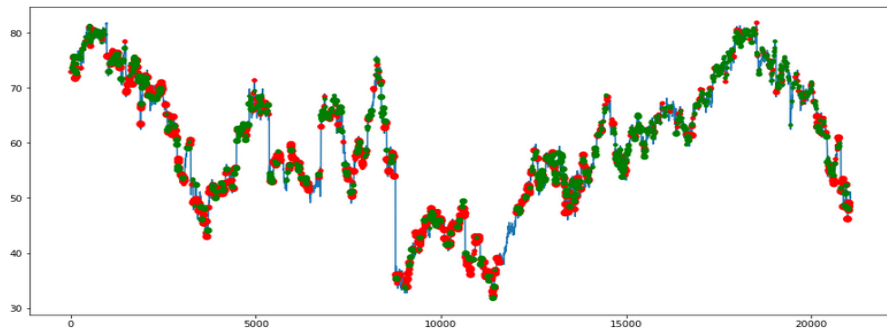
Cuadro 5.14: Resultados globales series profit relativo por hora para conjuntos de prueba



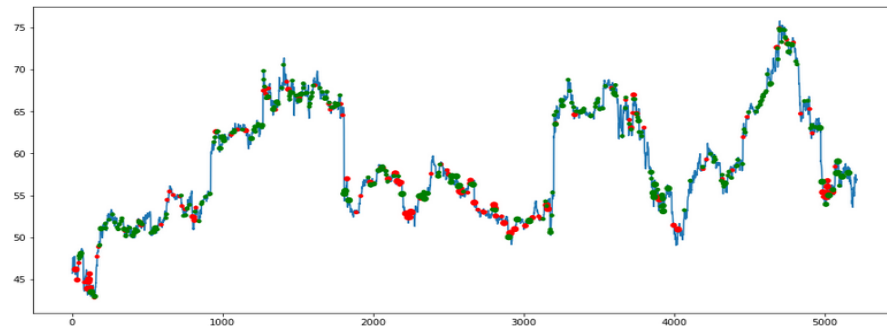
(a) Recompensas medias



(b) Error de entropía



(c) Conjunto de entrenamiento



(d) Conjunto de prueba

Figura 5.16: Gráficos de Qualcomm con 85 % de ruido. *Gráficos de compras y ventas \Rightarrow Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio*

5.3. Análisis y conclusiones de los resultados

Como se comenta en la sección 5.1.5, los resultados anteriores se muestran de forma separada pero siguiendo el orden progresivo de problema más simple a más complejo. Además, se muestra en un orden donde se pueden hacer distintas comparaciones entre los resultados cuando se añaden las *variables alternativas* y el **ruido** en el espacio de acciones.

5.3.1. Análisis de resultados sin comisión

Serie de Apple

Primeramente, en esta serie se muestran los resultados de la serie original, es decir, sin quitarle la tendencia ni añadiendo a las observaciones las variables alternativas.

En cuanto a la serie en bruto, a primera vista el gráfico 5.4b muestra que los modelos A3C convergen mucho más rápido en una política determinada. En cuanto al gráfico de recompensas medias de la Figura 5.4a, la que más destaca es A3C con LSTM, que es el modelo que muestra tener unas recompensas que crecen cada vez más con el paso del tiempo.

Si se quieren comparar las arquitecturas de NNs por separado, se puede ver que los modelos LSTM convergen más rápido y ganan más recompensa de media al final que los modelos MLP. A pesar de ello, en el conjunto de prueba los resultados son muy parecidos en cuanto a *profit por hora*. Una diferencia clara entre ambos es que los modelos LSTM tienden a comprar y vender mucho menos, esto puede ser un indicio claro de que los modelos MLP tardan más en converger y necesitan más pasos en el entrenamiento. En este caso, los modelos MLP parece que han aprendido en esta serie que una buena política es comprar mucho, ya que como la serie es creciente, siempre va a haber un balance positivo.

Viendo los resultados de la Tabla 5.2 del conjunto de entrenamiento, se puede ver la gran diferencia en el *profit por hora* entre los modelos A3C y PPO. A3C es capaz de ganar entre dos y tres veces más de beneficio con un número mínimo de de compras y ventas comparado con PPO. Además, viendo los resultados de la Tabla 5.2 del conjunto de prueba, A3C sigue ganando por bastante a PPO, con un número mínimo de jugadas ofensivas, pareciendo a primera vista un modelo que generaliza mejor.

Sin embargo, como se comenta en la sección 5.1.4, el *profit por hora* no es suficiente para decir si un modelo es mejor que otro. Por lo tanto, para completar la comparación de modelos, es necesario ver la toma de acciones del agente en la serie de precios de Apple, los cuales se muestran en la Figura 5.5.

Las características principales de los gráficos de compras y ventas son los siguientes:

- Se puede ver que en el caso de PPO, el agente compra y vende tanto que es difícil de apreciar los puntos en la serie. A simple vista no hace movimientos inteligentes ya que compra en los valores más altos en entrenamiento y prueba. Además, como la serie tiene tendencia alcista, por muy mala política que aprenda el agente, este normalmente va a ganar *profit* ya que el precio final de la serie está en valores cercanos a los máximos y no se ha aplicado comisión.
- En el caso de A3C, el agente ha aprendido la simple política de comprar al principio y mantener hasta el final, por ser una serie de tendencia alcista. De esta forma gana mucho beneficio, pero está haciendo un claro *overfitting* al conjunto de entrenamiento, ya que en el conjunto de prueba hace exactamente lo mismo, cuando al final de esta serie hay un gran bajón en los precios, y un buen agente debería poder vender ahí arriba. Además, este tipo de políticas no interesa cuando se está intentando aplicar *trading* intradía.

Esto justifica que el *profit por hora* no es lo único en lo que hay que fijarse en estos problemas para saber si el agente ha llegado a una buena política. En este caso hasta se podría decir que es preferible la política aprendida por el modelo PPO, que por lo menos intenta comprar y vender en cualquier instante, mientras que el modelo A3C solo ha aprendido a comprar al principio y mantener, sin importar los bajones de precio.

Se puede deducir de estos resultados que lo más seguro es que el problema de esta serie sea la **tendencia**. Por culpa de esta tendencia alcista, el agente tiene la sensación de que haga lo que haga a largo plazo va a tener beneficio, y la **política óptima** a aprender por el agente es comprar en mínimos locales y vender en máximos locales para obtener el mayor beneficio posible.

Como esta política no es aceptable, se ha **eliminado** la tendencia de la serie **solo del conjunto de entrenamiento** por estas razones:

- El conjunto de prueba debe tener una serie real, ya que simula un entorno real, por lo tanto, no se deben transformar esos datos ni cambiar el precio de compra en el entorno.
- Solo se quiere que el agente no aprenda a comprar al principio y mantener hasta el final, por lo que con modificar solo el conjunto de entrenamiento es suficiente. Además, aquí si se puede permitir el lujo de transformar también la serie de precios histórica de las acciones, ya que si le dejamos los precios originales los modelos no van a cambiar en su política.

Serie de Apple sin tendencia

En el gráfico 5.8a se muestra que al entrenar el modelo A3C con LSTM, que ha sido el que en la serie sin transformar había aprendido la política de comprar al principio y olvidarse, ha tenido unos resultados realmente malos, ya que no ha sido capaz de converger el modelo en una política de beneficio positivo.

Una explicación de esta falta de convergencia es la **alta volatilidad de precios o ruido** que se puede ver en la serie. Con el problema del ruido, se vuelve al anterior dilema de la granularidad discutido en la sección 5.1.1. Se podrían usar series con valores diarios, pero se estaría saliendo del enfoque intradía que se había propuesto. Además, con una granularidad mayor o suavizado de la serie se estaría perdiendo mucha información.

Este problema se resuelve manteniendo la misma granularidad y añadiendo las **variables alternativas** propuestas en la sección 4.3.2. Esta mejora se refleja primeramente en el gráfico 5.8b, en el que al entrenar este modelo A3C con LSTM incluyendo las nuevas variables se obtiene una diferencia considerable en el *profit*.

Si se fija en las Tablas 5.4 y 5.5, a primera vista parece que el modelo A3C con LSTM ha cambiado su política de comprar al principio y mantener, por una parecida a las aprendidas por los modelos PPO. Sin embargo, el modelo A3C con MLP no parece haber aprendido ninguna política aceptable, ya que su balance es negativo en entrenamiento y solo ha hecho 34 compras. Lo más destacable es que A3C con MLP no es capaz de converger positivamente, mientras que PPO con MLP parece el modelo superior en esta serie.

En cuanto al conjunto de prueba, los modelos PPO muestran tener mejor resultado que los A3C, por lo que se confirma que no hay ningún tipo de *overfitting* por su parte. Se puede pensar que el modelo A3C con MLP es el mejor por tener el mayor beneficio en el conjunto de prueba, pero se faltaría ver su comportamiento en el gráfico de compra y venta. Como se puede ver en la Figura 5.7, se confirma que lo único que ha aprendido este modelo es comprar al principio de la serie y no hacer nada más. Esto justifica que el modelo A3C con MLP no se ha adaptado a la transformación de la serie. De esta forma se **confirma la clara superioridad** de PPO sobre A3C en esta serie.

Serie de Intel

En este entrenamiento se ha llegado a cuatro millones de pasos, ya que les ha costado más converger a los modelos, posiblemente por tratarse de una serie más compleja que la de Apple. En la Tabla 5.6 se puede ver que el modelo PPO con LSTM sobresale sobre los demás respecto al *profit por*

hora. Luego, el modelo A3C con MLP sigue haciendo muy pocas compras y ventas, lo que da pensar de que para este problema siempre aprende políticas muy **simples**. En el conjunto de prueba mostrado en la Tabla 5.7, el modelo A3C con LSTM parece el mejor en cuanto a *profit por hora*.

En cuanto a los gráficos del progreso del entrenamiento mostrados en la Figura 5.9, el modelo PPO con LSTM es el que mejor *profit* por episodio aprende con el paso del tiempo. En el caso de los modelos A3C con LSTM y PPO con MLP, parece que se quedan estancados y no mejoran de principio a fin. En el gráfico 5.9b se confirma que los modelos PPO tardan más en converger en una política determinada que los modelos A3C, como pasaba también en la serie de Apple. Esto es un indicio de que los modelos A3C terminan su convergencia muy rápido en políticas simples, mientras que PPO se esfuerza en aprender políticas mejores y más realistas para este problema.

Según los resultados anteriores discutidos, PPO con LSTM parece el mejor modelo en el conjunto de entrenamiento, mientras que A3C con LSTM parece el mejor modelo en el conjunto de prueba. Sin embargo, falta ver las políticas aprendidas de las compras y las ventas de los modelos más característicos, como se ve en la Figura 5.10:

- La política simple del modelo A3C con MLP básicamente resulta en comprar al principio en precios bajos y vender al final de la serie en precios altos. Es una estrategia a largo plazo, que no es mala, pero deja con la duda de si se comportaría igual en una serie con una forma distinta, ya que el conjunto de prueba se parece al de entrenamiento en que los datos más altos se encuentran al final de la serie.
- El modelo A3C con LSTM, que se supone que es el mejor por su gran beneficio en el conjunto de prueba, deja mucho que desear al ver su política. Hay algunos agentes que como el precio al final de la serie de entrenamiento es mayor que en el resto, lo único que aprenden es abusar comprando porque saben que así ganarán beneficio. Este es el caso de este modelo, que tiene la suerte de que el conjunto de prueba es creciente y por eso obtiene buenos resultados. Se confirma entonces que ha hecho *overfitting* en el conjunto de entrenamiento. Además, con comisión esta política no es válida, ya que al abusar de la compra se perdería demasiado dinero.
- En el caso del modelo PPO con LSTM, se ve que ha aprendido una buena política por su resultado en el conjunto de entrenamiento. Se puede observar como hay más compras en los precios bajos y más ventas en los valores altos. Además de que hay muchas compras y ventas, cosa que interesa mucho más que las políticas a largo plazo (en el caso de no haber comisión). En el conjunto de prueba también se desenvuelve bien, ya que igualmente compra mucho más en precios

bajos y vende en precios altos. De esta forma, considerando todos los datos de los resultados, este es claramente el mejor modelo de esta serie.

Serie de Qualcomm

Lo primero que hay que remarcar es la extrema volatilidad de los datos, como ocurre en la serie de Apple transformada de la Figura 5.6, por lo que añadir las variables alternativas propuestas es algo **importante** para que el agente pueda aprender con más facilidad.

Viendo las Tablas 5.8 y 5.9, en esta serie se han usado pasos de distinta longitud en el entrenamiento para intentar forzar la convergencia con beneficio positivo. Hay que reconocer que esta ha sido la serie más **difícil** de entrenar, con la conclusión que todas las series son distintas y tienen una complejidad variable. El único modelo que parece que ha conseguido un buen resultado en el conjunto de entrenamiento según el *profit por hora* es PPO con LSTM, mientras que los otros tienen recompensas negativas. En el conjunto de prueba este modelo también parece destacar con el mayor *profit por hora*. Hay que fijarse que los modelos A3C en este conjunto tienen el mismo resultado, muy probablemente porque han aprendido la misma **política simple**.

Viendo el gráfico 5.11a del *profit* por episodio, se puede ver que el único que ha llegado a la recompensa positiva es PPO con LSTM, mientras que el resto de modelos no han podido mejorar prácticamente desde el principio. En cuanto a la convergencia de la política, en el gráfico 5.11b se puede ver como los modelos A3C han convergido sumamente rápido, esto sumado a que en la tabla muestran un número de compras mínimas, puede interpretarse que posiblemente han aprendido la **simple** política de comprar al principio y olvidarse.

Finalmente, para confirmar las políticas de los distintos modelos hay que fijarse en los gráficos de compra y venta más característicos mostrados en la Figura 5.12. En los modelos PPO con LSTM se confirma la **buena política** aprendida en entrenamiento y prueba ya que se puede ver como compra en precios o picos bajos y vende en picos altos mayormente. Sin embargo, el modelo PPO con MLP no parece que haya aprendido una política con sentido, sino que básicamente se ha dedicado a comprar y vender todo el tiempo sin pensar. En la gráfico 5.12d se **confirma** el problema de la política básica en los modelos A3C. En este caso siguen comprando al principio y se olvidan hasta el final. En el conjunto de entrenamiento aprenden esta política incluso perdiendo *profit*, porque posiblemente estos modelos no hayan sido capaces de encontrar **ningún patrón relevante en los datos** que les conduzca a una buena política.

5.3.2. Análisis de resultados con comisión

Tal y como se ha visto en la anterior sección, el modelo PPO con la arquitectura LSTM ha sido el único modelo capaz de adaptarse y aprender políticas aceptables en las tres series. Es por esto que en los experimentos con comisión solo se han registrado resultados con de modelo con el fin de simplificar los análisis.

Serie de Apple

Al haber confirmado en los experimentos de la serie de Apple que es necesario **eliminar la tendencia alcista** de la serie para que el agente aprenda una política adecuada, en este caso se elimina directamente para afrontar el problema sin comisión.

La primera dificultad a la que se ha enfrentado con la comisión es que en del modelo entrenado con 0.1 % de comisión, no se ha conseguido converger en un *profit* positivo. Como se muestra en la Figura 5.13, se ha intentado forzar hasta cinco millones de pasos pero se ve como **no es capaz** de pasar la barrera de beneficio positivo fácilmente.

Esto se debe principalmente a que el agente **pierde** demasiado dinero debido a la comisión, ya que sigue comprando y vendiendo muchas veces. Entonces, si el problema es que abusa de la compra y la venta, la pregunta que uno debe hacerse es **¿Por qué el agente no aprende a simplemente comprar y vender menos haciendo mejores jugadas?**.

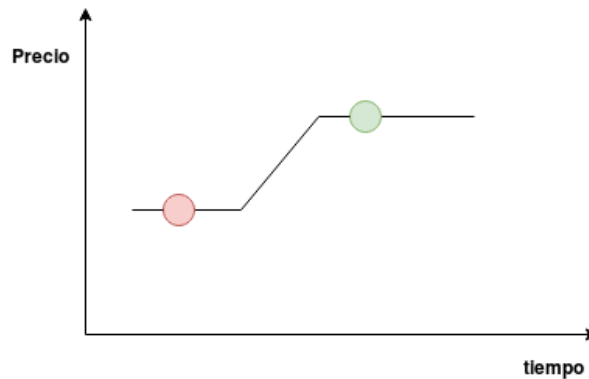


Figura 5.17: Explicación de comisión vs. beneficio

Esta pregunta es mejor justificarla gráficamente viendo la Figura 5.17. A simple vista parece una buena jugada, ya que se ha vendido en un precio más alto del que se ha comprado. Sin embargo, esa diferencia de precios debe ser mayor a la suma de las comisiones de la compra y la venta. Además, como las

acciones permiten el comprar o vender todo, esas jugadas acarrearán un mayor **riesgo** y una mayor comisión, lo que debe dar lugar a que el agente siempre juegue en pocas cantidades. Estas son muchas restricciones que ponen al agente en prácticamente una **misión imposible** para conseguir aprender esa política tan refinada.

Aprender este tipo de política podría funcionar si se deja al agente aprender durante infinidad de pasos, pero esto daría lugar a un seguro *overfitting* haciendo que el agente no funcione bien en el conjunto de prueba.

Como el problema principal de las políticas aprendidas sin comisión es que el agente abusa de las compras y las ventas, aquí es donde entra en juego la mejora añadida en el entorno para **introducir ruido** en el espacio de acciones. Esta mejora se explicó detalladamente en la sección 4.3.3.

Haciendo esta práctica, es **importante** entrenar sin comisión para que el modelo aprenda buenas políticas con pocas acciones ofensivas. Luego, al evaluar en los conjuntos de entrenamiento y prueba, se puede introducir la comisión que se quiera para probar hasta qué nivel de comisión es capaz de soportar el modelo para esa serie. Hay que recordar que cuanto más porcentaje de ruido se introduzca, el modelo aprenderá a hacer menos compras y ventas. Esto puede graduarse según la serie para ajustar el entrenamiento.

Se han probado otras ideas para reducir el número de compras ofensivas, como dar una recompensa fija al agente cuando haga la acción de no hacer nada. El problema de esto es que el agente tiende a siempre no hacer nada y no interesarse por comprar y vender para obtener beneficio.

Ahora, se va a proceder a analizar los resultados de esta serie habiendo introducido este ruido en el espacio de acciones. Tras hacer varias pruebas con distintos niveles de ruido, en este caso el mejor nivel encontrado ha sido del 90 %. En la Figura 5.14a se puede ver como el modelo converge rápidamente en beneficios positivos, llegando al final a conseguir estabilizarse en recompensas altas. De esta forma ya da buenas indicaciones de que ha aprendido una buena política, que ha conseguido estabilizarse a partir del paso 4500000 como se muestra en el error de la entropía de la Figura 5.14b.

Fijándose en la Tabla 5.10, se puede ver como claramente el beneficio va disminuyendo conforme aumenta la comisión. En una comisión baja como es 0.1 %, puede conseguir bastante *profit por hora* tanto en entrenamiento como en prueba. Cuando se usa una comisión normal, como es 0.25 %, el *profit por hora* disminuye mucho pero todavía sigue ganando. También hay que fijarse en que el número de compras y ventas ha **disminuido** mucho respecto a los modelos PPO aprendidos sin comisión en esta serie.

Por último, hace falta analizar el gráfico de compras y de ventas que se muestra en las Figuras 5.14c y 5.14d. Se refleja en el conjunto de entrenamiento una buena política por comprar mayormente en precios bajos

mientras mayormente vende en altos. En cuanto al conjunto de test, hay mayores ventas que compras pero no siguen ningún patrón en específico. También decir que el ser un conjunto de prueba con tendencia alcista ayuda a tener un beneficio al final de la serie. Por lo tanto, a simple vista no se puede saber si el agente ha generalizado bien en nuevos datos.

Serie de Intel

Para el entrenamiento de esta serie se han probado con distintos niveles de ruido, concluyendo que el mejor obtenido es del 80 %. En la Tabla 5.11 obtenida como resultado de este entrenamiento se puede ver que con una comisión baja del 0.1 % es capaz de obtener beneficio en ambos conjuntos. Sin embargo, con 0.25 % no es capaz de obtener beneficio en el conjunto de entrenamiento. Además, en ambos conjuntos se mantiene un número bajo de compras y ventas, que es algo que interesaba desde el principio.

Es algo raro que en el conjunto de entrenamiento tenga *profit por hora* negativo y en el de prueba sea positivo. En este caso, este se debe por dos razones:

- En el conjunto de prueba es más fácil ganar beneficio, ya que tiene la tendencia alcista. Muy mal lo tiene que hacer el agente para también perder en ese conjunto.
- En esta serie se usa menos porcentaje de ruido que en la serie de Apple, por lo que se realizan más compras y ventas. Esto hace que el modelo sea más sensible a la comisión y por eso pierda en el conjunto de entrenamiento.

En este caso, esto es debido a que en el conjunto de prueba es más fácil ganar beneficio, ya que tiene la tendencia alcista. Muy mal lo tiene que hacer el agente para también perder en ese conjunto.

En cuanto a las recompensas medias obtenidas que se pueden ver en la Figura 5.15a, se muestra un buen resultado ya que los beneficios son positivos con el paso del tiempo y parecen estabilizarse alto. En cuanto al error de entropía de la Figura 5.15b, baja considerablemente hasta estabilizarse en un punto, dando a conocer que ha aprendido una política determinada. Estos dos gráficos dan a concluir a priori que el agente ha aprendido una buena política.

Para acabar el análisis del modelo aprendido en esta serie, falta ver los gráficos de compras y ventas mostrados en las Figuras 5.15c y 5.15d. En el conjunto de entrenamiento muestra una buena política de mayormente comprar en valores más bajos que cuando vende. Se puede ver la impresionante política aprendida en entrenamiento ya que vende en los máximos locales

con una gran precisión. En cuanto al conjunto de prueba, esta política no se refleja tan claramente, aunque por lo menos parece que los puntos de compra de mayor cantidad (el círculo es mayor) se reflejan en mínimos locales.

Serie de Qualcomm

Esta serie, como se indica en la sección anterior, muestra la mayor dificultad para conseguir resultados en todos los modelos. El único modelo que fue capaz de obtener un buen resultado en los experimentos sin comisión fue PPO con LSTM, que llegó a conocer una buena política, mientras que el resto no supo aprender la forma de conseguir algo de beneficio.

Como se ha comentado antes, la dificultad de esta serie es especialmente alta, esto añadido a la inclusión de ruido en el espacio de acciones ha hecho muy difícil la búsqueda de un buen porcentaje de ruido para una buena convergencia. Además, en los experimentos ocurría algo que no pasaba en el resto de experimentos. Lo ocurrido ha sido que a partir de un punto del aprendizaje, el agente empezaba a comportarse cada vez peor llegando a perder mucho beneficio. Esta es una de las razones por la que se ha entrenado hasta un punto temprano que tomaba buenos resultados, siendo este punto dos millones de pasos.

Tras varias pruebas con muchos niveles de ruido, el menor resultado en este caso se ha obtenido con 85 %. En la Tabla 5.12 ocurren unos resultados muy parecidos al de la anterior serie. Con una comisión de 0.1 % es capaz de ganar beneficio mientras que con 0.25 % no es capaz de ganar beneficio en el conjunto de entrenamiento, tal y como pasaba en la serie de Intel. Esto justifica que esta serie es bastante complicada, y este sistema de RL solo es capaz de abordarla con una comisión relativamente baja. En cuanto al número de acciones ofensivas, se consigue un número bajo, cumpliendo el objetivo de la mejora del ruido en el espacio de acciones.

Como en la serie de Intel, también ocurre que en el conjunto de entrenamiento se obtenga un *profit por hora* negativo y en el de prueba sea positivo. En este caso, este se debe por estas dos razones:

- En el conjunto de prueba, el precio al final es mayor que al principio. Mientras que en el conjunto de entrenamiento el precio al principio es mucho mayor que al final. Esto es un factor que influye bastante en el *profit por hora*.
- En esta serie se usa menos porcentaje de ruido que en la serie de Apple, por lo que se realizan más compras y ventas. Esto hace que el modelo sea más sensible a la comisión y por eso pierda en el conjunto de entrenamiento. Se hacen aproximadamente 700 compras y ventas, mientras que en el de Apple se hacen aproximadamente 500.

En el gráfico de recompensas medias de la Figura 5.16a se puede ver que no se estabiliza fácilmente en recompensas positivas, solo en un pico intermedio y al final, que es donde se ha parado el entrenamiento. Este gráfico vuelve a demostrar que se estaba tratando con una serie complicada. En cuanto al error de entropía de la Figura 5.16b, parece estabilizarse en un punto bajo, por lo que en el momento de cortar el entrenamiento parece haber encontrado una política determinada.

En la Figura 5.16c y 5.16d se puede ver el desempeño de esta política gráficamente, donde en el conjunto de entrenamiento parece haber aprendido una buena política. Esto es porque en los puntos con precios bajos parece haber un mayor número de grandes compras, mientras que en los máximos locales abundan los puntos de ventas. En cuanto al conjunto de prueba, la política parece haber generalizado bien, ya que las compras mayores se concentran en los mínimos locales, mientras que las ventas mayormente en los puntos mayores.

5.3.3. Análisis de resultados globales y conclusiones de los análisis

En los resultados obtenidos para todos los experimentos de las series propuestas se ha visto la clara superioridad de la técnica PPO sobre A3C. A3C parecía mayormente enfocarse en políticas **simples** ya por no encontrar la forma de aprender ningún patrón en los datos o por ver que la serie tenía una tendencia alcista, basándose únicamente en comprar pronto y esperar el resto de la serie.

Dentro de PPO la NN que ha sabido adaptarse a todas las series ha sido la que tenía arquitectura con capas LSTM. Esto es explicable porque una arquitectura con capas LSTM funcionan normalmente mejor si los datos tienen una **estructura secuencial o temporal**. Además, una capa LSTM estaría tratando con 38 variables en este problema, mientras que MLP trataría con 38 multiplicado por el número de velas pasadas usadas, dando lugar a un claro problema de **dimensionalidad** para esta arquitectura.

Una vez concluido el mejor modelo con la mejor arquitectura, se han englobado todos los *profits relativos por hora* obtenidos al validar todas las series con este modelo para distintas comisiones. Recordar que en el caso de las series de Apple solo se muestra el caso de haber eliminado la tendencia e incluido las variables alternativas. Estos resultados globales se muestran en las Tablas 5.13 y 5.14 para el conjunto de entrenamiento y prueba respectivamente.

Viendo estas tablas, se reafirma el impacto por la puesta en escena de la comisión, donde no es capaz de tener un *profit por hora* positivo en todas las series. También se refleja que cada serie tiene una **complejidad distinta**

al mostrar distintos *profits relativos por hora*. Por ejemplo, se puede ver como en la serie de Apple se ha obtenido mayor beneficio en el conjunto de entrenamiento que en las series de Intel y Qualcomm.

Mirando a lo largo de las comisiones para cada serie, el *profit relativo por hora* disminuye conforme la comisión aumenta. Sin embargo, hay una excepción en la serie de Apple en el conjunto de entrenamiento, donde el modelo con el uso de ruido en el espacio de acciones es capaz de tener un *profit relativo por hora* mayor con 0.1 % de comisión que sin comisión. Esto puede ser una casualidad, porque en los resultados de los conjuntos de prueba se cumple este patrón de decrecimiento por la comisión en las tres series.

Una buena noticia obtenida en estos resultados, es que este modelo parece obtener beneficio de forma general con una comisión del 0.1 %. Puede que con mejoras propuestas en el siguiente capítulo se llegue a este punto con una comisión estándar del 0.25 %.

A parte de los resultados de los modelos, se han obtenido las siguientes conclusiones para aplicar a las series:

1. **Tendencia en la serie:** Se ha visto que si la serie tiene tendencia alcista es mejor quitarle esta tendencia en el conjunto de entrenamiento para que el agente no se acostumbre únicamente a comprar al principio y aguantar.
2. **Confrontación del ruido:** A pesar de usar una granularidad relativamente alta para evitar problemas de **mucho ruido**, se ha visto que las series pueden seguir conteniendo ruido. Para hacer un modelo más resistente al ruido se han añadido **variables alternativas** de un pasado a más largo plazo, de esta forma el agente puede conocer más información de la serie en un estado determinado.
3. **Entrenamiento con comisión:** Como se ha visto en los resultados de la serie de Apple, el entrenar con comisión puede no ser buena idea para que el agente aprenda una buena política con comisión. Por ello, se ha concluido que una solución mejor es incluir una distorsión en el espacio de acciones mientras se entrena sin comisión, como se ha propuesto en el presente trabajo.
4. **Aprendizaje por parte del agente:** No todas las series funcionan igual ante un nivel de ruido determinado en el espacio de acciones, por lo que habría que encontrar un nivel de ruido distinto para cada serie. Como se ha visto en la serie de Qualcomm, estos modelos de RL no convergen siempre hacia mejor durante el entrenamiento, por lo que a veces es necesario parar antes el entrenamiento, ya que si se deja mucho tiempo puede hacer empeorar al agente.

Capítulo 6

Conclusiones y mejoras futuras

Para concluir el presente TFM, se van a mostrar las conclusiones generales de la realización de ese trabajo, así como la descripción de posibles mejoras futuras que pueden mejorar el trabajo hecho.

6.1. Conclusiones

Como conclusiones generales, se muestra la satisfacción del presente autor por mostrar que un enfoque de RL en el *trading* de Mercados Bursátiles puede dar buenos resultados. Este es un enfoque alternativo para tratar con series temporales, el cual no ha sido mostrado en ninguna asignatura del Máster, y puede ser una buena alternativa para distintos problemas.

En el capítulo 3 se justificó que la técnica PPO es la mejor de las listadas para resolver este problema con un enfoque de RL. Esta justificación se ha demostrado **empíricamente**, por lo que se puede deducir que se ha obtenido una buena **relación teórico-práctica**.

Los problemas de la comisión expuestos en las referencias de la sección 4.1 no se han conseguido resolver completamente, pero si se ha conseguido mejorar gracias a la nueva mejora de **inclusión de ruido** explicada en la sección 4.3.3. Se he demostrado también empíricamente que la adicción de las nuevas **variables alternativas** propuestas en la sección 4.3.2 ha dado mejoras frente a no usarlas.

Se ha conseguido un beneficio económico en todas las series propuestas **sin comisión**. Pero por desgracia, en el mundo real normalmente los sitios de *trading* a nivel intradía tienen impuesta una pequeña comisión, por lo que el ganar dinero de forma automática no es una tarea trivial.

Con estos resultados y conclusiones, se concluye que el sistema de RL construido para resolver este problema **no asegura de forma universal** encontrar una política que obtenga un *profit por hora* positivo en todas las series de la Bolsa de Valores con una comisión estándar del 0.25 %. Para llegar a ese punto, se van a proponer mejoras en la siguiente sección.

Se quiere dar a conocer al lector que el código de este trabajo ha sido para el autor extremadamente complicado de desarrollar por los siguientes puntos:

- El tiempo de ejecución del entrenamiento puede ser especialmente alto si no se optimiza bien ni se tienen las máquinas adecuadas. Es necesario una máquina con al menos una GPU para el entrenamiento con NNs. Además, es recomendable en el sistema de RL desarrollado normalizar las observaciones antes de incluirlas en el entorno, como se ha hecho finalmente, ya que así se gana mucho tiempo.
- El querer realizar un modelo que se enfrente a todas las series temporales de la Bolsa de Valores puede ser un problema tedioso ya que todas tienen complejidad distinta. Un analista puede estar haciendo pruebas y pruebas sin saber por qué el modelo no aprende en distintas series. Por suerte, las mejoras propuestas en este trabajo han podido resolver este problema.
- Hay que tener un especial cuidado con las indexaciones en los datos. Este sistema contiene una gran cantidad de variables vectoriales o de *Dataframes*. Al ser un entorno que quiere simular una puesta en tiempo real, hay que tener mucho cuidado con las indexaciones ya que los fallos pueden ocurrir desfases en el tiempo, como ha ocurrido durante el desarrollo.

6.2. Mejoras futuras propuestas

Al ver en la capítulo 5 que uno de los mayores problemas para el aprendizaje del agente ha sido el **ruido**, se propone a incluir en este trabajo una técnica del *estado del arte* que puede mejorar además de las **variables alternativas** expuestas.

En el artículo [Pretorius et al., 2018], se muestra como el uso de una NN con arquitectura **Denoise Autoencoder** (DAE), puede aprender una función f cuya aplicación al conjunto de datos del problema puede quitar gran parte de este ruido buscando una representación alternativa de estos. Normalmente un DAE es un *autoencoder* donde se le introduce a mano un ruido especial que ayude a aproximar esta función de forma correcta. Además, este tipo de técnicas suele funcionar con cientos de miles de ejemplos, por lo que se propone usar con series de una granularidad inferior a una hora.

Una arquitectura general de un *autoencoder* se muestra en en la Figura 6.1. Básicamente toma como entrada y salida el mismo ejemplo, almacenándose en las capas intermedias de la NN una representación comprimida de los datos. En este ejemplo se deduce que es para reducir la dimensionalidad de los datos, pero se puede cambiar la arquitectura a modo del DAE donde las capas internas tienen normalmente más neuronas que el resto, la entrada del *autoencoder* son datos con ruido introducido manualmente y la salida son los datos sin ruido.

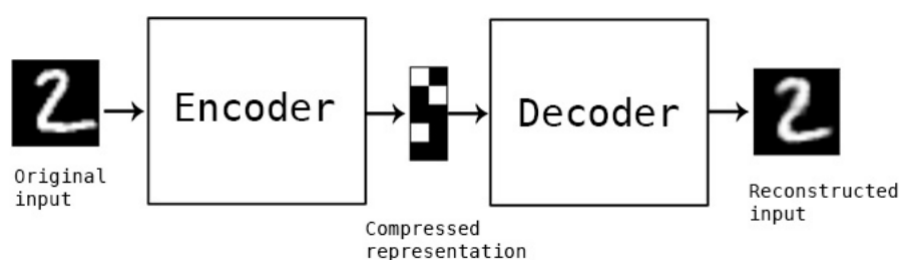


Figura 6.1: Arquitectura general *autoencoder*

En el capítulo 5 también se vió que la NN con arquitectura LSTM daba mejor resultado que el MLP. Como se dice, esto tiene que ver porque las LSTM son capaces de encontrar patrones más fácilmente si los datos tienen una **estructura secuencial o temporal**. Por lo tanto, se propone introducir en la arquitectura de Redes Neuronales Convolucionales (CNN) para probar su resultado, ya que estas pueden tener un resultado parecido o mejor que las LSTM.

Normalmente las CNNs están especialmente enfocadas a problemas con imágenes, pero en este caso en concreto se usarían capas *convolucionales* de una dimensión, ya que se están tratando con series temporales. Algunas ventajas de estas capas convolucionales respecto a las LSTM se muestran a continuación:

- Son más fáciles de paralelizar para un entrenamiento más rápido.
- Tienen menos pesos en cada capa, por lo que necesitan menos tiempo para entrenar.

Por último, se va a comentar una anécdota en el desarrollo del trabajo. Un día se hizo un cambio pequeño en el entorno de RL y empezaron a dar resultados buenísimos con unas ganancias enormes. En la Figura 6.2 se ve un gráfico de ejemplo sobre compra y venta en una serie de granularidad minuto a minuto. Se puede ver como la política es extremadamente buena, e incluso en esta granularidad ruidosa, ya que prácticamente compra siempre en mínimos locales y vende en máximos locales con gran precisión.

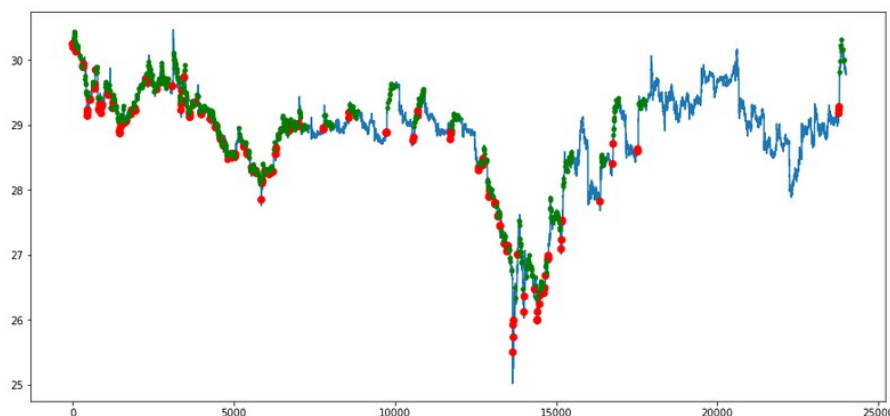


Figura 6.2: Gráfico de compra ventas en serie de granularidad minuto a minuto. *Punto rojos: Compras. Puntos verdes: Ventas. Tamaño del punto: Ponderado por la cantidad. Eje X: Tiempo. Eje Y: Precio*

Por desgracia, otro día se vió que la mejora venía por un error en la indexación de las observaciones, ya que se estaban dando al agente **observaciones futuras al precio actual**, es decir, se le estaban dando datos futuros que no se podían saber en ese momento. Por lo tanto, al agente aprendía que si el precio de la observación era mayor que el actual, entonces compraba, sino vendía. Era una política muy simple pero que de esa forma funcionaba muy bien.

Esta anécdota ha dado la idea de incluir una buena mejora en el sistema de RL propuesto. La idea es introducir un **modelo de regresión para predicción de precios a corto plazo** usando los datos pasados a la observación del estado actual. De esta forma, se añadirían estas predicciones de precios de futuros cercanos a la observación del agente.

Por ejemplo, la librería *statsmodels* de *Python* incluye la técnica SARI-MAX, que básicamente es un modelo Seasonal AutoRegressive Integrated Moving Average con un regresor exógeno. Lo bueno de las predicciones con este modelo es que también ofrecen el **intervalo de confianza** para cada cada predicción. Por lo tanto, puede que el agente decida comprar o vender dependiendo de ese intervalo de confianza.

Se está casi seguro que si se introducen en el trabajo todas estas mejoras propuestas, el modelo mejorará, pero algunas de ellas, como el DAE, pueden ser un reto el implementarlas.

Bibliografía

- [Barth-Maroon et al., 2018] Barth-Maroon, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., y Lillicrap, T. (2018). Distributed distributional deterministic policy gradients. *arXiv*.
- [Britz, 2018] Britz, D. (2018). Introduction to learning to trade with reinforcement learning.
- [Fujimoto et al., 2018] Fujimoto, S., van Hoof, H., y Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv*.
- [Hessel et al., 2017] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., y Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. *arXiv*.
- [Hornik et al., 1989] Hornik, K., Stinchcombe, M., y White, H. (1989). Multilayer feedforward networks are universal approximators. *Cognitivemethodum*.
- [King, 2019] King, A. (2019). Creating bitcoin trading bots don't lose money.
- [Lapan, 2018] Lapan, M. (2018). *Deep Reinforcement Learning Hands-On*. Packt Publishing Ltd.
- [Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., y Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv*.
- [Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., y Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *arXiv*.
- [Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *DeepMind Technologies*.

- [Pretorius et al., 2018] Pretorius, A., Kroon, S., y Kamper, H. (2018). Learning dynamics of linear denoising autoencoders. *arXiv*.
- [Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., y Klimov, O. (2017). Proximal policy optimization algorithms. *OpenAI*.
- [Sutton y Barto, 2014] Sutton, R. S. y Barto, A. G. (2014). *Reinforcement Learning: An Introduction*. Bradford Book.

