



## **ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

### **Proyecto fin de Grado**

**Grado en Ingeniería Informática - Ingeniería del Software**

**Estudio del mercado de divisas mediante algoritmos Deep Learning**

**Realizado por**  
**Carlos Sevilla Barceló**  
**Fermín Fernández Borrego**

**Dirigido por**  
**Jorge García Gutierrez**

**Departamento**  
**Lenguajes y Sistemas Informáticos**

**Sevilla, 06/2019**

---

## Resumen

---

Las grandes mejoras conseguidas en los últimos años en el hardware aumentando su capacidad de procesamiento y almacenamiento han propiciado que campos como el *Deep Learning* hayan sufrido una gran evolución y se hayan convertido en las herramientas más potentes para la extracción de información a partir de datos.

En este proyecto se pretende implementar un sistema basado en arquitectura Big Data que alimente con un flujo constante de información un conjunto de redes neuronales para la predicción de valores bursátiles de divisas en el mercado Forex. Para ello nuestra solución propuesta se apoya en herramientas ya consolidadas como **Apache Spark** y **TensorFlow**.

A su vez el sistema pretende ser un estudio sobre la viabilidad de la precisión de las predicciones de los valores conociendo únicamente el histórico de los mismos, gracias a la capacidad de abstracción y reconocimiento de patrones de las **redes neuronales artificiales**. Como parte del estudio se implementarán dos tipo de redes, redes recurrentes y redes convolucionales.

---

## Abstract

---

The great improvements achieved in recent years in the hardware increasing its processing and storage capacity have led to fields such as Deep Learning have undergone a great evolution and have become the most powerful tools for extracting information from data.

This project aims to implement a system based on big data architecture that feeds with a constant flow of information a set of neural networks for the prediction of securities of currencies in the Forex market. For this purpose, our proposed solution is based on already consolidated tools such as **Apache Spark** and **TensorFlow**.

At the same time the system pretends to be a study on the viability of the precision of the predictions of the values knowing only the historical of the same ones, thanks to the capacity of abstraction and recognition of patterns of the **artificial neural networks**. As part of the study, two types of networks will be implemented: recurrent networks and convolutional networks.

---

## Agradecimientos

---

En el final de esta larga etapa, que ha sido la universidad, que concluye con este proyecto, me gustaría mencionar y agradecer a ciertas personas que han estado conmigo durante camino.

Para empezar, quiero agradecer a Antonio y María, mis padres, todo lo que han hecho por mí en estos años de carrera, porque sin ellos este proyecto y toda mi carrera estudiantil no habría sido posible. Pese a haberles dado motivos más que suficientes para no confiar en mi labor como estudiante, no han dejado de hacerlo y lo han dado todo para yo pueda llegar hasta aquí. Gracias a ellos vivo la informática con gran pasión y me han enseñado a dar siempre lo máximo de mí mismo.

Quiero agradecer también a mi hermano Guillermo, que ha hecho que mi estancia en Sevilla haya sido mucho más divertida y todo el interés que ha prestado en este proyecto.

Me gustaría mencionar a mis abuelos Antonio y Paquita, que aunque ya no están para leer ésto, sé que les hubiera hecho mucha ilusión vivir este momento conmigo. Juan y Estrella sé que lo están haciendo, como no paran de demostrármelo.

Quiero agradecer también a Jorge, mi tutor, que me enseñó a programar y a amar la programación en primer curso en Fundamentos de la Programación, y está llevando este proyecto sin parar de aportar ideas y soluciones.

Y junto a los ya mencionados, quiero agradecer a mis compañeros de la carrera por estos años maravillosos, pero en especial a Fermín, mi compañero de proyecto, que nos hemos desvivido para sacar más de una asignatura importante y este proyecto.

Muchísimas gracias a todos.

*Carlos Sevilla Barceló*

Con este proyecto pretendo cerrar el largo camino que ha sido la carrera universitaria y me gustaría agradecerselo a las personas que han hecho posible que haya llegado hasta aquí.

En primer lugar, a mis padres Fermín y Rosario y a mi hermano Pascual por soportarme durante todo este largo camino que ha sido la carrera universitaria.

A mis abuelos por estar siempre ahí para apoyarme cuando lo necesitaba aunque algunos ya no estén conmigo.

A mi tutor Jorge quién no se ha cansado de ayudar y aportar ideas hasta convertir este proyecto en lo que es ahora.

A todos los compañeros y amigos que he conocido durante la carrera, pero en particular a Carlos coautor de este proyecto sin el cual aún faltaría bastante camino que recorrer.

Muchísimas gracias a todos.

*Fermín Fernández Borrego*

---

# Índice general

---

<b>Índice general</b>	<b>IV</b>
<b>Índice de cuadros</b>	<b>VII</b>
<b>Índice de figuras</b>	<b>IX</b>
<b>Índice de código</b>	<b>XII</b>
<b>1 Introducción</b>	<b>1</b>
1.1 Contexto . . . . .	1
1.2 Definición de objetivos . . . . .	2
1.3 Motivación . . . . .	2
1.4 Estructura del proyecto . . . . .	3
<b>2 Estudio teórico</b>	<b>5</b>
2.1 Forex . . . . .	5
2.1.1 Estructura . . . . .	5
2.1.2 Influencias . . . . .	5
2.1.3 Horario de Forex . . . . .	6
2.1.4 Indicadores financieros . . . . .	7
2.2 Resolución del problema . . . . .	15
2.2.1 Definición del problema . . . . .	15
2.2.2 Clasificación . . . . .	15
2.2.3 Regresión . . . . .	16
2.2.4 Aplicación de la regresión al problema . . . . .	17
2.3 Big Data . . . . .	18
2.4 Deep Learning . . . . .	19
2.4.1 Redes neuronales . . . . .	19
2.4.2 Funciones de activación . . . . .	21
2.4.3 Arquitectura de una red neuronal . . . . .	26
2.4.4 Redes Neuronales Recurrentes . . . . .	29
2.4.5 Capas de redes neuronales recurrentes . . . . .	30
2.4.6 Redes neuronales convolucionales . . . . .	34
2.4.7 Capas de las redes neuronales de convolución . . . . .	35
2.4.8 Capas comunes entre arquitecturas . . . . .	37
2.4.9 Entrenamiento de las redes neuronales . . . . .	39
<b>3 Diseño del sistema</b>	<b>44</b>
3.1 Tecnologías troncales . . . . .	46
3.1.1 Python . . . . .	46
3.1.2 Docker . . . . .	46

3.2	Productores de datos . . . . .	47
3.2.1	Extracción del dato . . . . .	47
3.2.2	Envío de la información . . . . .	47
3.2.3	Repetición . . . . .	47
3.2.4	Tecnologías utilizadas . . . . .	47
3.3	Procesado y almacenamiento . . . . .	49
3.3.1	Recepción . . . . .	49
3.3.2	Tratamiento . . . . .	49
3.3.3	Almacenamiento . . . . .	49
3.3.4	Tecnologías utilizadas . . . . .	50
3.4	Red neuronal . . . . .	51
3.4.1	Consideraciones previas . . . . .	51
3.4.2	Red Neuronal Recurrente . . . . .	53
3.4.3	Red Neuronal Convolucional . . . . .	55
3.4.4	Tecnologías utilizadas . . . . .	57
3.5	Monitorización . . . . .	58
3.5.1	Tecnologías utilizadas . . . . .	59
<b>4</b>	<b>Arquitectura del sistema</b>	<b>61</b>
4.1	Apache Spark . . . . .	62
4.1.1	Fundamentos de Spark . . . . .	63
4.1.2	Cómo funciona Spark . . . . .	65
4.1.3	Modulos de Spark . . . . .	67
4.2	TensorFlow . . . . .	69
4.2.1	Grafos . . . . .	69
4.2.2	Tensor . . . . .	70
4.2.3	Keras . . . . .	70
4.3	Apache Kafka . . . . .	71
4.3.1	Broker . . . . .	71
4.3.2	Topic . . . . .	71
4.3.3	Productor . . . . .	71
4.3.4	Consumidor . . . . .	71
4.3.5	Zookeeper . . . . .	72
4.3.6	Garantías . . . . .	72
<b>5</b>	<b>Análisis del sistema</b>	<b>74</b>
5.1	Data-Driven Application . . . . .	74
5.2	Procesos del sistema . . . . .	75
5.3	Catálogo de requisitos . . . . .	76
5.3.1	Requisitos generales . . . . .	76
5.3.2	Requisitos de información . . . . .	77
5.3.3	Requisitos funcionales . . . . .	79
5.3.4	Requisitos no funcionales . . . . .	81
5.4	Casos de usos . . . . .	82
5.5	Matriz de trazabilidad . . . . .	85
<b>6</b>	<b>Análisis temporal y costes de desarrollo</b>	<b>87</b>
6.1	Desglose de tareas . . . . .	87
6.2	Ánalisis temporal . . . . .	90
6.2.1	Diagrama de Gantt . . . . .	92
6.3	Costes de desarrollo . . . . .	95
6.3.1	Recursos de hardware . . . . .	95

6.3.2 Recursos de personal . . . . .	97
6.3.3 Coste total . . . . .	97
<b>7 Manual de usuario</b>	<b>99</b>
7.1 Preparación del entorno . . . . .	99
7.1.1 Requisitos mínimos . . . . .	99
7.1.2 Entorno . . . . .	99
7.2 Instalación e implantación . . . . .	100
<b>8 Pruebas</b>	<b>104</b>
8.1 Spark y Kafka . . . . .	104
8.2 Estudio y selección de hiperparámetros de redes neuronales . . . . .	105
8.3 Red Recurrente - EUR/USD - Puesta en producción . . . . .	107
8.4 Red Recurrente - EUR/USD - 24 horas en producción . . . . .	108
8.5 Red Recurrente - BTC/USD - Puesta en producción . . . . .	109
8.6 Red Recurrente - BTC/USD - 24 horas en producción . . . . .	110
8.7 Red Convolucional - EUR/USD - Puesta en producción . . . . .	111
8.8 Red Convolucional - EUR/USD - 24 horas en producción . . . . .	112
8.9 Red Convolucional - BTC/USD - Puesta en producción . . . . .	113
8.10 Red Convolucional - BTC/USD - 24 horas en producción . . . . .	114
8.11 Resultados globales . . . . .	115
<b>9 Conclusiones</b>	<b>117</b>
<b>10 Desarrollos futuros</b>	<b>119</b>
<b>Glosario de términos</b>	<b>121</b>
<b>Referencias</b>	<b>125</b>

---

## Índice de cuadros

---

2.1	Horario Forex . . . . .	6
2.2	Festivos . . . . .	6
2.3	Información sobre la función de activación lineal . . . . .	21
2.4	Información sobre la función de activación sigmoide . . . . .	22
2.5	Información sobre la función de activación tangencial hiperbólica . . . . .	23
2.6	Información sobre la función de activación ReLu . . . . .	24
2.7	Información sobre la función de activación Leaky ReLu . . . . .	25
2.8	Información sobre la función de activación softmax . . . . .	26
2.9	Operaciones posibles en una red neuronal . . . . .	27
5.1	Requisito general 001 . . . . .	76
5.2	Requisito general 002 . . . . .	76
5.3	Requisito general 003 . . . . .	76
5.4	Requisito de información 001 . . . . .	77
5.5	Requisito de información 002 . . . . .	77
5.6	Requisito de información 003 . . . . .	78
5.7	Requisito funcional 001 . . . . .	79
5.8	Requisito funcional 002 . . . . .	79
5.9	Requisito funcional 003 . . . . .	79
5.10	Requisito funcional 004 . . . . .	80
5.11	Requisito funcional 005 . . . . .	80
5.12	Requisito no funcional 001 . . . . .	81
5.13	Caso de uso 001 . . . . .	82
5.14	Caso de uso 002 . . . . .	83
5.15	Caso de uso 003 . . . . .	84
6.1	Equipo 1 . . . . .	95
6.2	Equipo 2 . . . . .	95
6.3	Equipo 3 . . . . .	95
6.4	Equipo 4 . . . . .	96
6.5	Servidor AWS . . . . .	96
6.6	Amortización . . . . .	96
6.7	Precio/hora por perfil . . . . .	97
6.8	Coste total por perfil . . . . .	97
6.9	Coste total del proyecto . . . . .	97
8.1	Soluciones a la prueba de selección de hiperparámetros . . . . .	105
8.2	Error cuadrático asociado a las predicciones de la puesta en producción del par EUR/USD . . . . .	107
8.3	Error cuadrático asociado a las predicciones tras 24 horas en producción del par EUR/USD . . . . .	108

8.4	Error cuadrático asociado a las predicciones de la puesta en producción del par BTC/USD con la red recurrente . . . . .	109
8.5	Error cuadrático asociado a las predicciones después de 24 horas de la puesta en producción del par BTC/USD con la red recurrente . . . . .	110
8.6	Error cuadrático asociado a las predicciones de la puesta en producción del par EUR/USD . . . . .	111
8.7	Error cuadrático asociado a las predicciones tras 24 horas en producción del par EUR/USD . . . . .	112
8.8	Error cuadrático asociado a las predicciones de la puesta en producción del par BTC/USD con la red convolucional . . . . .	113
8.9	Error cuadrático asociado a las predicciones después de 24 horas de la puesta en producción del par BTC/USD con la red convolucional . . . . .	114

---

## Índice de figuras

---

2.1	Acumulación/Distribución . . . . .	7
2.2	SMA vs EMA . . . . .	8
2.3	Bandas de Bollinger . . . . .	9
2.4	CCI . . . . .	10
2.5	Puntos pivote . . . . .	11
2.6	ROC . . . . .	12
2.7	Oscilador estocástico . . . . .	13
2.8	Momentum . . . . .	14
2.9	Gráfico con variable objetivo de dos problemas de clasificación . . . . .	15
2.10	Fórmula de la regresión lineal . . . . .	16
2.11	Ejemplo de secuencia móvil . . . . .	17
2.12	Deep Learning . . . . .	19
2.13	Diagrama de una neurona . . . . .	20
2.14	Estructura de datos . . . . .	20
2.15	Función de activación lineal . . . . .	21
2.16	Función de activación sigmoide . . . . .	22
2.17	Función de activación tangencial hiperbólica . . . . .	23
2.18	Función de activación ReLu . . . . .	24
2.19	Función de activación Leaky ReLu . . . . .	25
2.20	Arquitectura del perceptrón multicapa . . . . .	26
2.21	Modelo MLP con datos de ejemplo . . . . .	27
2.22	Contenido de las capas ocultas de una red neuronal con el dataset MSNIT . . . . .	28
2.23	Esquema básico de una red neuronal recurrente . . . . .	29
2.24	Ejemplo de overfitting en red neuronal recurrente . . . . .	29
2.25	Diagrama que muestra la arquitectura de una neurona LSTM . . . . .	30
2.26	Primer paso en el procesamiento de una neurona LSTM . . . . .	31
2.27	Segundo paso en el procesamiento de una neurona LSTM . . . . .	31
2.28	Tercer paso en el procesamiento de una neurona LSTM . . . . .	32
2.29	Último paso en el procesamiento de una neurona LSTM . . . . .	32
2.30	Esquema de neuronas conectadas secuencialmente . . . . .	32
2.31	Comparación entre neuronas recurrentes GRU y LSTM . . . . .	33
2.32	Arquitectura de una red neuronal Convolucional . . . . .	34
2.33	Ejemplo de convolución en una dimensión . . . . .	35
2.34	Ejemplo de convolución con Stride tomando valor 1 y 2 . . . . .	36
2.35	Ejemplo del funcionamiento de la capa Max Pooling . . . . .	36
2.36	Diagrama de la arquitectura de una capa Dense . . . . .	37
2.37	Ejemplo de funcionamiento de la capa Flatten . . . . .	37
2.38	Diagrama explicativo de la función de la capa Dropout . . . . .	38
2.39	Diagrama de descenso del gradiente . . . . .	40
2.40	Diagrama explicativo de un mínimo local y el mínimo global . . . . .	40

---

2.41 Diagrama explicativo de la diferencia entre learning rate bajo, medio y alto . . . . .	41
2.42 Comparativa entre tipos de gradient descent . . . . .	41
2.43 Diagrama explicativo de Backpropagation . . . . .	42
3.1 Diagrama de diseño de flujo de la información . . . . .	44
3.2 Docker VS VM . . . . .	46
3.3 Ejemplo del vector de entrada con un momento temporal concreto . . . . .	51
3.4 Ejemplo del vector resultado con un momento temporal concreto . . . . .	52
3.5 Gráfico de la precisión en el aprendizaje indicando la época óptima para parar el aprendizaje . . . . .	52
3.6 Arquitectura de red neuronal recurrente utilizada en el proyecto . . . . .	53
3.7 Resultado típico de la predicción de la red neuronal recurrente . . . . .	54
3.8 Arquitectura de red neuronal convolucional usada en el proyecto. . . . .	55
3.9 Resultado típico de la predicción de la red neuronal convolucional . . . . .	56
3.10 Valores actuales de las divisas . . . . .	58
3.11 Representación de los resultados . . . . .	58
3.12 Histórico del BTC/USD . . . . .	58
3.13 Histórico del BTC/USD con indicadores . . . . .	59
4.1 Diagrama de arquitectura del proyecto . . . . .	61
4.2 Logo de Apache Spark . . . . .	62
4.3 Ejemplo de un DAG de Spark . . . . .	64
4.4 Tipos de transformaciones en Spark . . . . .	65
4.5 Operaciones en Spark . . . . .	66
4.6 Módulos de Spark . . . . .	67
4.7 Logo de Tensorflow . . . . .	69
4.8 Esquema de una operación representada como un grafo . . . . .	69
4.9 Ejemplo de tensor . . . . .	70
4.10 Logo de Apache Kafka . . . . .	71
4.11 Arquitectura de Apache Kafka . . . . .	72
5.1 Entrenamiento de una red neuronal . . . . .	75
5.2 Visualización de resultados . . . . .	75
5.3 Matriz de trazabilidad de casos de uso, requisitos de información y requisitos funcionales contra requisitos generales . . . . .	85
6.1 Planificación temporal I . . . . .	90
6.2 Planificación temporal II . . . . .	91
6.3 Diagrama Gantt I . . . . .	93
6.4 Diagrama Gantt II . . . . .	94
8.1 Resultado de las predicciones de la red recurrente tras ponerse en producción con el par EUR/USD . . . . .	107
8.2 Resultado de las predicciones de la red recurrente tras 24 horas en producción con el par EUR/USD . . . . .	108
8.3 Resultado de las predicciones de la red recurrente tras ponerse en producción con el par BTC/USD . . . . .	109
8.4 Resultado de las predicciones de la red recurrente tras 24 horas en producción con el par BTC/USD . . . . .	110
8.5 Resultado de las predicciones de la red convolucional tras ponerse en producción con el par EUR/USD . . . . .	111
8.6 Resultado de las predicciones de la red convolucional tras 24 horas en producción con el par EUR/USD . . . . .	112

8.7	Resultado de las predicciones de la red convolucional tras ponerse en producción con el par BTC/USD . . . . .	113
8.8	Resultado de las predicciones de la red convolucional tras 24 horas en producción con el par BTC/USD . . . . .	114

---

## Índice de código

---

3.1	Búsqueda de elementos span . . . . .	48
7.0	Instalación de requisitos . . . . .	99
7.0	Instalación de docker . . . . .	100
7.0	Inicialización de docker . . . . .	100
7.0	Instalación de docker-compose . . . . .	100
7.0	Verificación de la instalación . . . . .	100
7.0	Lanzamiento de Kafka . . . . .	100
7.0	Creación de topics . . . . .	101
7.0	Lanzamiento de scraper BTC/USD . . . . .	101
7.0	Lanzamiento de scraper EUR/USD . . . . .	101
7.0	Lanzamiento de la base de datos . . . . .	101
7.0	Lanzamiento del consumer EUR/USD . . . . .	101
7.0	Lanzamiento del consumer BTC/USD . . . . .	101
7.0	Inicialización de la red recurrente EUR/USD . . . . .	101
7.0	Inicialización de la red convolucional EUR/USD . . . . .	101
7.0	Inicialización de la red recurrente BTC/USD . . . . .	102
7.0	Inicialización de la red convolucional BTC/USD . . . . .	102
7.0	Lanzamiento de Dash . . . . .	102

# CAPÍTULO 1

---

## Introducción

---

### 1.1– Contexto

En el momento en el que se empieza a utilizar un objeto simbólico que define el valor de todo lo que nos rodea, los seres humanos, inventan una unidad de medida para el valor de todo ello.

El mercado financiero existe desde que se inventa la moneda de cambio. La bolsa de valores, que nace en el siglo XVI [01], es un sistema financiero en el cual se negocian e intercambian acciones y obligaciones de distintas empresas. Más tarde, se crea un mercado en el cual se regula el valor de la moneda de cada país, estableciendo así las equivalencias entre monedas. Este mercado, se considera el más importante de todos a día de hoy, dado que el valor de un par de monedas tiene la capacidad de influir en una decisión o acción que engloba a países enteros.

En la actualidad, tanto grandes entidades, como pueden ser los bancos, como los pequeños inversores, se dedican a obtener beneficios en función del incremento/descenso de un par de valores. Este tipo de movimiento frente a movimientos bursátiles de otro tipo, como puede ser el IBEX 35, presentan una mayor seguridad y estabilidad a la hora de invertir, debido a que los cambios a largo plazo no son muy significativos y son muy atractivos porque su cambio de valor o volatilidad se ve reflejado en el movimiento intradiario, siendo estos los movimientos más bruscos, como podría ser una subida del 20 % respecto a su valor original, la cual se da en períodos de tiempo muy cortos. Esto es debido a que el valor de las monedas está muy regulado y tiende a estandarizarse lo más rápido posible.

Respecto a lo anteriormente mencionado, los inversores tratan de aprovechar al máximo un incremento o un descenso en un par de monedas, especulando con él para obtener el máximo beneficio posible. Este mercado está muy influenciado por las noticias económicas y políticas, dando así una cierta facilidad para poder prever un movimiento. Este tipo de inversión tiene un método asociado, llamado **análisis fundamental**, el cual no se basa en el valor en sí, sino en el contexto que lo acompaña: situación política, situación militar, crisis financiera, etc. Existe otro método de inversión, que podríamos considerar opuesto al anterior, llamado **análisis técnico**. Este análisis tiene en cuenta el valor actual de un activo financiero. Además, tiene en cuenta el valor histórico de esa moneda y distintos indicadores matemáticos, como puede ser la esperanza matemática de un valor.

Gracias a que el análisis técnico puede automatizarse, a día de hoy, existe un concepto llamado **Algotrading**, que consiste en predicciones financieras en base a algoritmos. Para ello se necesita, además de un fuerte conocimiento matemático y financiero, una fuente de datos continua y fiable y una capacidad de cómputo muy alta, debido a que, como hemos mencionado anteriormente, en estas operaciones hay que estar preparado para cualquier movimiento brusco.

No es hasta 2017 [02] cuando se empieza a utilizar las técnicas de Deep Learning en el trading algorítmico. Este tipo de algoritmo, basado en las **redes neuronales**, es capaz de buscar patrones

en el histórico de valores de un activo bursátil. Dichos patrones, no siempre son fuertes y robustos en los que poder confiar tu dinero, de hecho, la mayoría son patrones muy volátiles, debido a que dependen de unas condiciones muy concretas, las cuales al verse alteradas mínimamente dejan de cumplirse. Sin embargo, estos patrones tienen una esperanza matemática muy sólida.

Una red neuronal necesita de un histórico de valores lo suficientemente grande como para poder proporcionar unas predicciones con robustez. Además, para predecir valores en tiempo real, necesitan de un suministro de datos continuo y fiable, debido a que si el algoritmo toma una premisa falsa como válida, las predicciones resultantes carecen de valor alguno. Gracias a los avances en Big Data, existen herramientas que puedan recoger datos en tiempo real, procesarlos y alimentar las redes neuronales con datos nuevos.

Por todo lo mencionado anteriormente, el objetivo de este trabajo fin de grado es desarrollar una arquitectura software basada en un sistema Big Data que alimente con los datos en tiempo real a las dos implementaciones de modelos predictivos basados en redes neuronales. Concretamente, las arquitecturas de redes neuronales a utilizar serán **el modelo recurrente**, y **el modelo convolucional**.

## 1.2– Definición de objetivos

El objetivo principal de este proyecto es elaborar un sistema Big Data que permita predecir los valores de un activo bursátil, recogiendo los datos en tiempo real, procesando y alimentando la red neuronal, pudiéndose plasmar sus predicciones en un panel de control.

Para conseguir este objetivo principal se han definido los siguientes objetivos específicos:

- Aprender las nociones básicas sobre mercados financieros para poder entender las predicciones de nuestro sistema.
- Estudiar las arquitecturas y parámetros necesarios para utilizar técnicas de Deep Learning en un problema de regresión.
- Crear sistemas que permitan obtener la información de un valor en tiempo real.
- Diseñar un flujo de trabajo (pipeline) que permita almacenar datos y poder procesarlos a medida que la red los necesite.
- Investigar sobre la funcionalidad que ofrece Spark para el procesamiento de datos en streaming.
- Investigar sobre la funcionalidad que ofrece Tensorflow para la implementación de las redes neuronales.
- Crear un manual de usuario para poder facilitar la instalación e implantación del sistema, así como el uso e interpretación del mismo.

## 1.3– Motivación

Este proyecto, además de profundizar en aspectos matemáticos e informáticos, se encuentra dentro de la rama de **Data Science**. La idea general es crear un sistema Big Data autónomo, para poder crear un entorno inteligente con el poder de analizar el mercado financiero.

El proyecto parte de la premisa de que poder predecir los valores de la bolsa con alta precisión es un reto muy complejo, debido a que los valores dependen de muchos factores externos. Por ello, nos hemos propuesto como objetivo, obtener la máxima precisión posible usando solo y exclusivamente indicadores matemáticos del histórico de ese valor. Estamos alertados tanto por nuestro tutor, como por varios expertos, que el proyecto es ambicioso y que los resultados puede que no sean los esperados, pero todo ello nos sirve como estímulo para avanzar.

Además, como motivación extra, en este proyecto se va a profundizar en las redes neuronales, campo aún en investigación, en el que día a día hay grandes avances. Las redes neuronales suponen un avance en el campo de la inteligencia artificial, el cual se está implantando en nuestra sociedad en un ritmo cada vez más acelerado. En este proyecto vamos a investigar sobre arquitecturas de los dos tipos de redes neuronales más importantes en la actualidad, las cuales son: redes neuronales recurrentes y redes neuronales convolucionales. Sin embargo, no existe tanta información como cabría esperar sobre el tema a tratar, lo que supone un problema para la extracción de conocimientos.

Todo este proyecto engloba unas tecnologías muy revolucionarias en la actualidad, como son **Spark**, **Kafka** y **TensorFlow**. Spark y Kafka son herramientas que pertenecen al ámbito del Big Data, y son utilizados en las compañías más grandes del mundo, como la **NASA** y **Ebay** [03]. Tensorflow es un framework desarrollado y mantenido por **Google** en su proyecto **Google BrAIn**, y se utiliza en prácticamente todas las empresas que están incorporando soluciones de Deep Learning en sus productos o servicios.

## 1.4– Estructura del proyecto

Finalizando con este apartado el capítulo de introducción, el documento seguirá con el **estudio teórico** del problema, el segundo capítulo.

El tercer capítulo tratará del **diseño del sistema** que se va a implementar, detallando los pilares que componen el proyecto.

En el cuarto capítulo se expondrá la **arquitectura y tecnologías** del sistema.

El quinto capítulo muestra el **análisis del sistema** a implementar que se ha realizado, especificando los requisitos que debe cumplir.

El sexto capítulo detalla el **análisis temporal** del proyecto y los **costes de desarrollo** del mismo.

El séptimo capítulo contiene un **manual de usuario** que detalla la preparación del entorno y la instalación e implantación del sistema.

En el octavo capítulo se explican las diferentes **pruebas** que se le han realizado al sistema en su conjunto y se analizarán sus resultados.

En el noveno capítulo se discutirán las **conclusiones** obtenidas tras la finalización del proyecto.

En el décimo y último capítulo se expondrán los posibles **desarrollos futuros** con los que mejorar el presente proyecto.



# CAPÍTULO 2

---

## Estudio teórico

---

Seguidamente se abordarán los conceptos teóricos necesarios para el desarrollo del proyecto. Se elaborará una introducción al mercado de divisas en las que se define el tipo de problema al que nos enfrentamos y se profundiza en los dos conceptos en los que se basarán el sistema, **Big Data y Deep Learning**.

### 2.1– Forex

El mercado Forex (abreviatura de Foreign Exchange) es un mercado mundial y descentralizado donde se opera con todas las divisas del mundo. Las operaciones en este mercado siempre involucran un par de divisas, la primera llamada par y la segunda llamada contraparte, de esta forma se expresa el valor de la primera divisa en base a la segunda. Ej: En el EUR/USD se expresa el valor del euro en dólares.

Forex es el mercado más grande y líquido del mundo con un volumen diario aproximado de cinco billones de dólares, aunque la mayor parte de las operaciones se concentran en el **Dólar**, el **Euro** y el **Yen**.

#### 2.1.1. Estructura

El mercado Forex es una combinación de los mercados **Spot, Forward y Futuros**.

El mercado Spot es el que ocupa el mayor volumen en Forex, ya que maneja los precios de las divisas y los intercambios inmediatos.

Tanto el mercado Forward como el mercado de futuros lidian con las transacciones que tendrán lugar en una fecha determinada, entre uno u ocho meses en el futuro. El mercado Forward se utiliza para llevar a cabo transacciones personalizadas, mientras que el mercado de Futuros trata con contratos estándar.

#### 2.1.2. Influencias

El valor de una divisa está influenciado por múltiples factores sociales, económicos, estacionales, la propia actividad del mercado, etc., siendo las políticas en los bancos centrales las mayores ajustadoras del suministro de capital, lo que implica que las decisiones sobre las políticas monetarias sean un factor primordial de influencia en el valor de las divisas.

La Reserva Federal, el Banco Central Europeo, el Banco de Inglaterra y el Banco de Japón son los organismos con mayor influencia en el mercado.

Los gráficos con los que se analiza el mercado Forex representan el constante cambio de oferta y demanda de los pares de divisas. La filosofía del balance de los precios es clave para entender

cómo funciona el Trading en este mercado, ya que todos los eventos económicos del mundo son relevantes en este mercado, pero sólo tiene influencia en la oferta y la demanda del activo de forma directa. [04]

### 2.1.3. Horario de Forex

A diferencia de otros mercados, como el mercado de acciones, el mercado de divisas está abierto veinticuatro horas al día, durante cinco días a la semana. En el cuadro 2.1 se puede ver el horario de apertura y cierre.

Apertura	Cierre
Domingo a las 23:00	Viernes a las 22:00

Cuadro 2.1: Horario Forex

Aunque se puede operar durante las 24 horas del día eso no implica que se produzcan cambios en los tipos de cambio de las divisas ya que éstos están afectados por los mercados de los países de origen de la divisa en cuestión. Por ejemplo, el cambio EUR/USD viene afectado por la bolsa de Nueva York y por las bolsas de las capitales europeas que sí cierran durante el día, por lo que en dichos momentos la variación en el tipo de cambio es mínima o incluso nula. Además de los horarios de apertura de las distintas bolsas, la actividad del Forex también se ve afectada por los días festivos; pudiendo catalogarse éstos en globales y locales. Siendo los primeros festivos que afectan a todo el mundo o una gran parte de él, provocando que la actividad del mercado de divisas sea mínima. En cambio, los festivos locales solo afectan a países individuales o a un grupo pequeño de ellos provocando que solo se vea afectada la actividad sobre la moneda de dicho país o países. En el cuadro 2.2 se muestran algunos de los festivos más relevantes.

Festivos globales	Festivos locales
Año nuevo	4 de julio en los EEUU
24 de diciembre	Año nuevo chino
31 de diciembre	14 de julio en Francia

Cuadro 2.2: Festivos

El conocer cómo y cuándo opera este mercado es crucial a la hora de crear un modelo acertado. Si nuestro modelo conoce los horarios de actividad de una divisa, por ejemplo, el EUR/USD, puede ser capaz de predecir que el valor cuando abra la bolsa de Nueva York será muy parecido al valor con el que cerró el día anterior, ya que durante el tiempo que esté cerrada la actividad es mínima.

### 2.1.4. Indicadores financieros

En el análisis financiero los indicadores son coeficientes o razones que proporcionan unidades contables y financieras de medida y comparación, a través de los cuales la relación por división entre sí de dos datos financieros directos, permite analizar el estado actual o pasado de una organización, en función de niveles óptimos definidos para ella.

Los indicadores financieros cuantifican numerosos aspectos y forman una parte integral del análisis de los estados financieros del mercado.

A continuación se procede a explicar algunos de los indicadores más representativos.

#### Acumulación/Distribución

La acumulación/distribución o AD mide la cantidad de dinero que entra y sale de un activo en un periodo.

Una de las grandes utilidades que presenta este indicador es el tema de las divergencias. Si el precio del mercado está en una dirección y el indicador está en la dirección contraria suele ser un aviso de que se puede producir un cambio de tendencia del mercado.

De esta divergencia se puede extraer dos tipos de señales, de compra y de venta.

La señal de compra aparecerá con una divergencia positiva, es decir, cuando el precio del mercado cae pero el AD está subiendo. En este caso existen muchas posibilidades de que el precio se gire próximamente al alza.

Una señal de venta por divergencia bajista surgiría cuando el precio del mercado sube pero el AD está cayendo. En este caso existen muchas posibilidades de que el precio se gire próximamente a la baja.

Básicamente, si el precio va subiendo el AD irá tornándose cada vez más positivo y si es al contrario, el mismo irá a negativo, pero ello también dependerá del importe del volumen.

$$MFM = \frac{(close - low) - (high - close)}{high - low}$$

$$MFV = MFM * volume$$

$$AD_n = AD_{n-1} + MFV$$



Figura 2.1: Acumulación/Distribución

## Media móvil

La media móvil muestra el valor medio de un activo durante un periodo de tiempo.

Como cualquier indicador, el periodo seleccionado es un elemento crítico. Cuanto más corto sea el periodo, el indicador será más sensible a cambios de precio, pero también será menos consistente. Por el contrario, si escogemos periodos largos el indicador será menos sensible a los cambios en el precio, pero a la vez más consistente.

La media móvil es un indicador de tendencia que genera una interpretación más clara de la acción del precio que otros indicadores y facilita el reconocimiento de la tendencia. Hay varias formas en las que este indicador puede ser utilizado para determinar la tendencia:

- Pendiente: Cuando la pendiente de la media es positiva la tendencia es alcista, cuando la pendiente es negativa la tendencia es bajista y cuando es necesaria se considera sin tendencia.
- Posición de la media respecto al precio: Si el precio se encuentra por encima de la media se considera una tendencia alcista, mientras que si el precio se encuentra por debajo de la media se considera una tendencia bajista.

Se calcula como la suma de los valores de cierre en cada periodo, dividido por el número de periodos.

$$SMA = \frac{\sum_{i=0}^n close_i}{n}$$

## Media móvil exponencial

La media móvil exponencial, al igual que la media móvil, muestra el valor medio de un activo durante un periodo de tiempo, pero a diferencia de ésta, pondera con mayor importancia a los datos más recientes. En la práctica todo ello se traduce a una reacción más rápida ante los cambios de precio que pueden llevar a falsas señales. Al igual que media móvil, la media móvil exponencial puede ser utilizada como indicador de la tendencia y se le pueden aplicar los mismos criterios.

$$EMA_n = \alpha * value + (1 - \alpha) * EMA_{n-1}$$

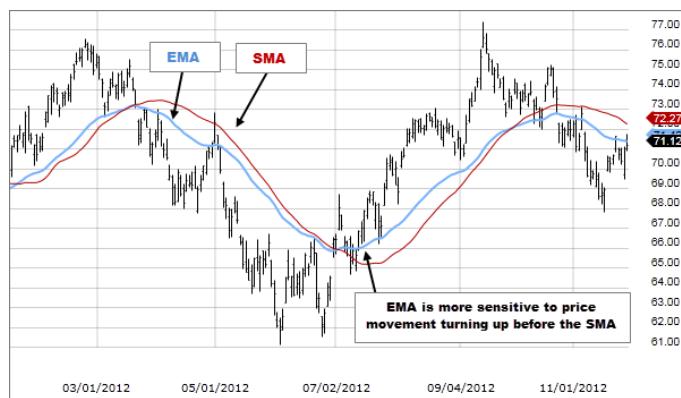


Figura 2.2: SMA vs EMA

### Bandas de Bollinger

Las bandas de Bollinger son dos curvas que envuelven al gráfico las cuales miden la volatilidad del precio del activo. Si los precios sobrepasan la banda por arriba indica que el mercado está sobrecomprado y si sobrepasan por debajo indica que está sobrevenido.

Si los precios se encuentran por encima de la media y cercanos a la banda superior están relativamente altos, lo que significa que pueda haber sobrecompra. Si están por debajo de la media y cercanos a la banda inferior están relativamente bajos, lo que significa que pueda haber sobreventa.

Si las bandas se estrechan sobre los precios éstas indican que el valor es muy poco volátil y, al contrario, si las bandas se ensanchan indican que el valor es volátil. Todo lo anterior proporciona una ayuda muy importante al inversor que opera con opciones.

Se suelen producir movimientos importantes y rápidos en los precios después de períodos en los que se han estrechado las bandas.

Los movimientos de precios que se originan en una de las bandas suelen tener como objetivo la banda opuesta, lo que facilita el hecho de determinar estos objetivos de precios. Muchos de los precios extremos (máximos o mínimos) de los movimientos tienen lugar en la banda o sus cercanías.

- Cuando los precios superan la banda superior es un síntoma de fortaleza del valor, si por el contrario se sitúan por debajo de la banda inferior es una señal de debilidad. Cuando los precios se sitúan fuera de cualquiera de las bandas es asumible la continuación del movimiento. Su utilización conjunta con otros indicadores ayuda a determinar con alta probabilidad los techos y suelos de los mercados.
- Cuando las bandas se mantienen cercanas, son síntoma de un periodo de baja volatilidad en el precio de la acción. Cuando se mantienen lejos una de otra, están indicando un periodo de alta volatilidad. Cuando tienen sólo una ligera pendiente y permanecen aproximadamente paralelas durante un tiempo suficientemente largo, se encontrará que el precio de la acción oscila arriba y abajo entre las bandas, como en un canal. Cuando esta conducta se repite regularmente junto a un mercado en consolidación, el inversor puede, con cierta confianza, utilizar un toque (o casi) a la banda superior o a la inferior como una señal de que el precio de la acción se está acercando al límite de su rango de negociación, y por ello es probable un cambio en la tendencia del precio.

Las bandas se calculan mediante una media móvil desplazándola hacia arriba y hacia abajo un número de desviaciones estándar (normalmente dos).



Figura 2.3: Bandas de Bollinger

### Índice de canales de productos básicos

El índice de canales de productos básicos o CCI, por sus siglas en inglés, mide la fuerza detrás de un cambio de precios. El CCI compara el precio actual con una medida anterior pudiendo decidirse así en términos relativos la fuerza o debilidad del mercado. Éste es un indicador versátil que puede ser utilizado tanto para identificar la tendencia como para detectar la sobrecompra y la sobreventa.

En general, indicará mayor tendencia alcista cuanto mayor sea el valor y viceversa. La mayoría de valores que puede tomar el indicador se encuentran entre -100 y 100, entendiéndose que un valor fuera de ese rango indica un comportamiento inusual del precio y se puede esperar que el movimiento en la dirección señalada continúe.

Detectar sobrecompra o sobreventa con CCI puede ser bastante inexacto debido a que CCI, en teoría, no tiene límite superior ni inferior, ya que la interpretación de situación es una interpretación subjetiva. Se debe tener en cuenta que esta detección es diferente según la situación del mercado y las características del activo. Así, en situaciones de tendencia fuerte podría ser necesario un rango de valores límite más amplio, como, por ejemplo, -200 y 200, siendo éste un nivel mucho más complicado de alcanzar e indicaría una situación de mercado en tendencia. Este rango también debe adaptarse a la volatilidad del activo, ya que a más volatilidad mayor necesidad de un rango más amplio.

$$p_t = \frac{\text{high} + \text{low} + \text{close}}{3}$$

$$CCI = \frac{p_t - MA}{0,015 * MD(p_t)}$$



Figura 2.4: CCI

### Puntos pivot

Los puntos pivot se usan para delimitar los niveles en los que el precio tiene una alta probabilidad de rebotar. Para obtenerlos se calcula el punto base y a partir de él se calculan los puntos de soporte y de resistencia.

Dichos niveles de soporte se pueden utilizar para obtener ganancias o para iniciar operaciones. Operadores menos activos pueden usar análisis técnicos de grandes figuras para ayudar a establecer un mercado diagonal y después utilizar los puntos pivot, los soportes y las resistencias para ayudar a ingresar y manejar una operación. Finalmente, los mercados con rangos más amplios tienden a proporcionar números más significativos que aquellos con rangos más estrechos.

$$PP = \frac{high + low + close}{3}$$

$$S_1 = 2 * PP - high$$

$$R_1 = 2 * PP - low$$

$$S_2 = PP - (high - low)$$

$$R_2 = PP + (high - low)$$



Figura 2.5: Puntos pivot

### Tasa de cambio

La tasa de cambio o ROC mide el porcentaje de cambio entre el precio actual y el precio un número determinado de periodos atrás en el tiempo.

El hecho de que sea un oscilador indica que su valor fluctúa por encima y debajo de cero. Y el hecho de que sea un indicador de momento expresa que cuando éste se encuentra por encima de cero en el momento actual existe presión compradora y viceversa.

Este indicador puede ser utilizado para determinar momentos de compra y venta. Cuando el ROC supera la línea cero de abajo a arriba, pasa de valor negativo a positivo, siendo ésta una señal de compra y cuando supera la línea de arriba a abajo, pasa de valor positivo a negativo, siendo ésta una señal de venta.

$$ROC_n = \frac{(close_n - close_{n-t}) * 100}{close_{n-t}}$$

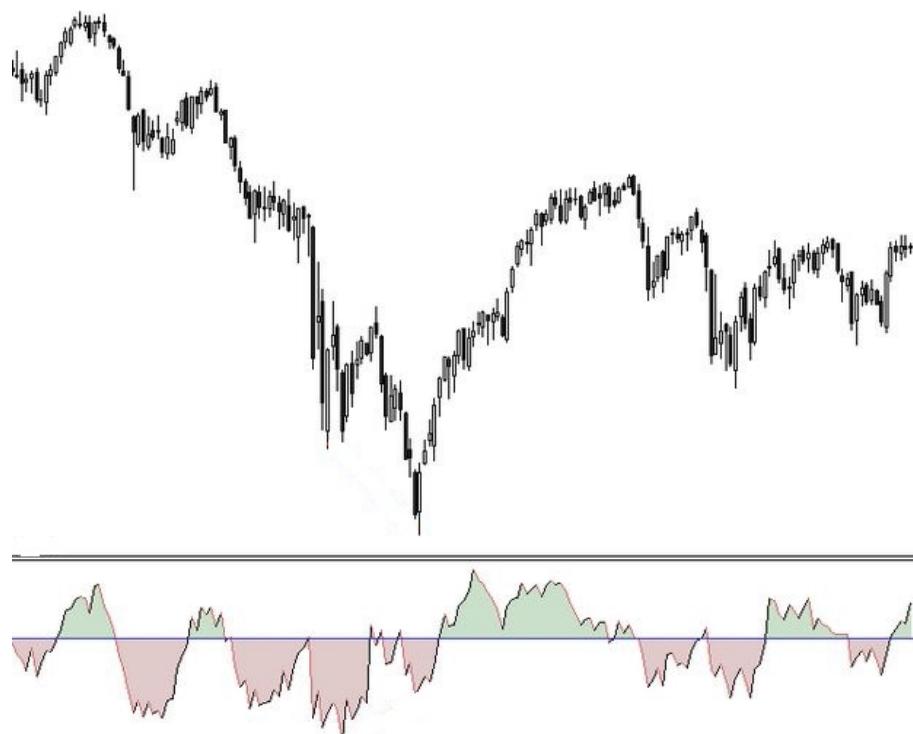


Figura 2.6: ROC

### Oscilador estocástico

El oscilador estocástico muestra el lugar del cierre en relación al rango máximo-mínimo a lo largo de un conjunto de períodos.

Dentro del gráfico del indicador se presentan dos zonas muy relevantes, generalmente denominadas zona de sobrecompra y zona de sobreventa.

Para trabajar con esta variable se traza una línea horizontal en el 50 %, a la que comúnmente se le llama “zona neutral”. Además, se dibujan dos líneas adicionales y paralelas a la primera; una en el 80 % y otra en el 20 %. Ésto hace que la zona de sobrecompra se ubique en la parte superior del gráfico, dentro del intervalo 100 % y 80 %. y la zona de sobreventa se encuentre en la parte inferior de éste, específicamente en el intervalo 0 % a 20 %

$$K = 100 * \frac{\text{close} - \text{low}}{\text{high} - \text{low}}$$



Figura 2.7: Oscilador estocástico

## Momentum

El momentum indica la tendencia de los precios.

Mientras la tendencia sea alcista, tanto el momentum como el ROC se mantienen positivos, mientras que en una tendencia bajista se mostrarán negativos. Una vez que ocurre un traspaso al alza del nivel 0 en el gráfico del momentum, puede interpretarse como una señal de compra, mientras que si el traspaso ocurre a la baja se puede interpretar como una señal de venta. Los niveles máximos y mínimos que puedan alcanzar, tanto el momentum como el ROC, muestran cómo de fuerte es la tendencia predominante en ese momento.

En general existen dos formas principales de interpretar éste. La primera consiste en utilizar el momentum como un indicador de tendencia. Una vez que el indicador llega hasta abajo y toca fondo comienza a subir nuevamente, siendo ésta una clara señal de compra, sin embargo, si por el contrario el indicador sube, toca el techo y comienza a bajar, la señal es de venta. Además, cada vez que el momentum alcance niveles altos o bajos extremos en comparación con sus valores históricos, se debe asumir que habrá continuación de la tendencia, lo que significa que al estar en niveles altos el momentum empieza a caer es probable que los precios del activo analizado continúen al alza. Lo mismo ocurre en el caso contrario, cuando este indicador se encuentra en niveles bajos con un mercado en tendencia bajista y de repente comienza a subir. Al igual que con cualquier otro oscilador, el momentum puede llegar a brindar señales falsas en un mercado con tendencia clara, por lo que se recomienda más su uso en mercados laterales.

La segunda forma de interpretar este indicador es utilizar el momentum como un indicador adelantado, basándose en la presunción de que los techos en el mercado se pueden identificar por rápidos incrementos en los precios, mientras que los suelos se identifican fácilmente por caídas rápidas en los precios. No obstante, aunque ésto es lo habitual en realidad, se puede decir que resulta una generalización bastante simple.

Durante los máximos del mercado, se puede observar como el indicador subirá de manera casi vertical hasta caer, creando de esta forma una divergencia con los precios que aún suben o se mueven con tendencia lateral, creándose una divergencia bajista. Por el contrario, en los mínimos del mercado, el momentum cae con fuerza hasta que revierte su movimiento antes que lo hagan los precios, con lo que se crea una divergencia alcista.

Las divergencias alcistas y bajistas pueden ser utilizadas de este modo como señales para entrar al mercado, tal como se muestra en el siguiente gráfico, en el cual se pudo haber abierto una posición bajista una vez identificada la divergencia.

Se calcula como el valor actual dividido por el valor un número determinado de períodos atrás en el tiempo, multiplicado por 100.

$$momentum_n = \frac{close_n}{close_{n-t}} * 100$$



Figura 2.8: Momentum

## 2.2– Resolución del problema

Una vez introducidos los conceptos más importantes relacionados con el mercado Forex, se define el problema relacionado con la predicción de valores.

### 2.2.1. Definición del problema

Como se ha expuesto anteriormente, los mercados de valores nos muestran el precio de una acción de un determinado mercado o producto. El objetivo de este proyecto es predecir ese precio en un futuro cercano, para así poder generar un beneficio con las operaciones de compra y venta.

Existen dos formas de predecir un valor, siendo una de ellas la predicción del valor como tal, del número, y siendo la otra la clasificación del valor. Todo ello reduce los tipos de problemas a dos: **regresión y clasificación**.

Este problema se puede enfocar de diferentes formas, pudiendo así adaptar el problema a un problema de regresión o de clasificación.

### 2.2.2. Clasificación

Cuando el solución de un problema no trata de valores de salida numéricos y continuos, sino de etiquetas, grupos o valores numéricos discretos, el problema y conjunto de técnicas para resolverlo se denominan de **clasificación**.

Si mostramos en un gráfico la **variable objetivo** o *target*, como se puede observar en la figura 2.9, el algoritmo obtiene una función cuyos valores actúan como delimitadores para los diferentes grupos de datos.

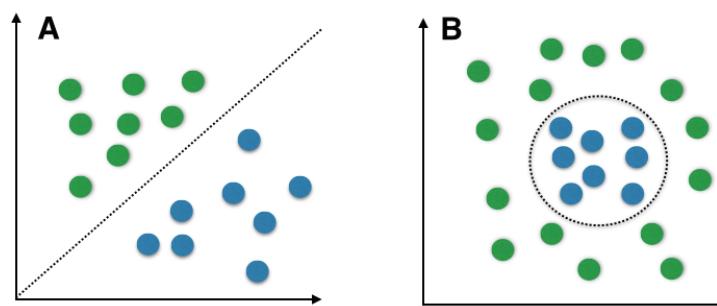


Figura 2.9: Gráfico con variable objetivo de dos problemas de clasificación

En el ámbito de estudio del Forex un ejemplo sencillo de clasificación puede ser que un modelo realice predicciones acerca de si la compra/venta es viable o no. Se podría generar un conjunto de datos con operaciones de transacciones que se hayan realizado en el pasado. Dicho conjunto de datos tendría como variable objetivo si la operación ha sido rentable o no.

El problema en este planteamiento está en considerar cuál es la rentabilidad de una operación de forma objetiva, porque una operación puede ser considerada rentable teniendo únicamente en cuenta el beneficio económico que ha generado la operación por sí misma. Una operación puede ser rentable aunque no tenga beneficio económico, debido a que ha podido interactuar con otras operaciones que pudiera tener activas el inversor. Teniendo en cuenta lo anterior, la complejidad del problema es mucho mayor porque habría que definir el modo de uso de algoritmo antes de plantearlo, habría que elegir el modo de uso durante la preparación de los datos.

La forma en la que se ha planteado la solución al problema en este proyecto no requiere predecir una característica, sino más bien predecir otro valor o valores nuevos. Este tipo de problemas son conocidos como **problemas de regresión**.

### 2.2.3. Regresión

La regresión está definida como un proceso para determinar relación entre variables. Este método, a diferencia de la clasificación, obtiene como resultado valores numéricos continuos.

Existen diferentes tipos de regresión, entre las que cabe destacar dos grupos principales:

- **Regresión lineal.**

- Regresión lineal simple.
- Regresión lineal compuesta.

- **Regresión no lineal.**

- Regresión exponencial.
- Regresión logarítmica.
- Regresión polinomial.

Las redes neuronales utilizan los diferentes métodos de regresión en sus cálculos, pero cabe destacar el modelo matemático de la **regresión lineal**, dado que es el que se encuentra más presente a la hora de hacer cálculos complejos entre neuronas.

La regresión lineal se calcula con la siguiente fórmula:

$$Y = a + bX$$

Figura 2.10: Fórmula de la regresión lineal

La variable dependiente  $Y$  es la variable a predecir mientras que la variable independiente  $X$  hace referencia al dato que conocemos.  $a$  y  $b$  son las variables que la red neuronal tiene que ajustar para poder obtener solución.

Cada neurona contiene una **regresión lineal**, por lo que una red neuronal es un **conjunto de regresiones** que dan un resultado.

### 2.2.4. Aplicación de la regresión al problema

Mediante regresión se puede obtener un valor o una cadena de valores. Si  $X$  es una lista de números en vez de ser un único número, podemos seguir utilizando la fórmula aplicando métodos matriciales.

En este problema se pretende predecir los **45 minutos posteriores** al momento en el que se realiza la predicción en base a los **180 minutos anteriores**. Por ello, nuestra variable independiente  $X$  será un vector de 180 números. Los cálculos necesarios para llevar a cabo este tipo de operaciones son muy costosos si se realiza por el método tradicional, por ello se delegarán los cálculos al ordenador.

Se aplicará el método de la **secuencia móvil** para crear los datos que se utilizarán en los cálculos de la regresión. Este método permite crear secuencias de valores introduciendo datos nuevos consecutivos y eliminando datos antiguos. Todo ello se puede entender fácilmente en la figura 2.11.

Datos				
1	2	3	4	5
Secuencia 1	1	2	3	
Secuencia 2		2	3	4
Secuencia 3			3	4
				5

Figura 2.11: Ejemplo de secuencia móvil

La red neuronal va a plantear tantas regresiones lineales como neuronas tenga, combinándolas todas en un único vector resultado. No sabemos en base a qué características se van a plantear dichas regresiones, e intentar comprenderlo es una labor que se encuentra en proceso de investigación. A fecha en la que está escrito este documento, lo único que se sabe es que las redes realizan diferentes regresiones de forma aleatoria.

## 2.3– Big Data

Big Data es un ámbito de la informática difícil de definir correctamente. Por definición literal, hace referencia a sistemas que tienen una gran cantidad de datos, ya sean estructurados, semiestructurados o no estructurados. Aunque esta definición es muy ambigua, ¿cuántos datos son una gran cantidad de datos?, ¿hay un mínimo?

La definición más valida de Big Data que consideramos, es la de Oracle: Big Data son datos que contienen una mayor variedad y que se presentan en volúmenes crecientes y a una velocidad superior [05]. Esta definición se conoce como la definición de "tres V", refiriéndose a volumen, velocidad y variedad, que son tres necesidades que deben de cubrirse de forma equilibrada.

En este proyecto se ven identificadas las tres V.

- **Volumen:** Recibimos datos de dos monedas diferentes con su valor, con una frecuencia de unos 5 segundos. Esto significa, que cada hora obtenemos 720 nuevos datos de cada moneda, 1440 entre ambas. Puede parecer poco, pero este sistema debe ser escalable por si se añaden más monedas. Si se añadieran los 79 pares [06] que existen actualmente, tendríamos 56880 datos nuevos cada hora.
- **Velocidad:** Como se ha mencionado anteriormente, las operaciones en bolsa en un mercado intradiario, tienen que ser lo más rápidas posible. Si una operación se realiza más tarde de lo necesario para ejecutarse correctamente, las consecuencias pueden desembocar en una pérdida de dinero no deseada. Por ello, nuestro sistema debe procesar el dato lo más rápido posible, dejando el máximo tiempo de maniobrabilidad posible para que la red saque su predicción y el usuario tenga todo el tiempo posible para planificar una estrategia.
- **Variedad:** Los datos que utiliza el sistema descrito en esta memoria, no proceden de una única fuente. Esto es debido a que nuestro sistema tiene que ser independiente del resto. Si una página web que suministra datos, o una API, se cae, nuestro sistema no puede caer detrás. Por ello, usamos diversas fuentes de datos para asegurar el correcto funcionamiento del sistema.

Estos tres puntos planteados han sido clave para diseñar el sistema y las herramientas a utilizar.

Recoger los datos en tiempo real se conoce como **data streaming**, y es uno de los pilares fundamentales de este proyecto. Para este proceso, hemos escogido la herramienta Apache Kafka, que actúa como almacén de datos, y lo hace de forma distribuida. Está orientado para que su lanzamiento se haga en un cluster, aunque admite el modo standalone. Está basado en un modelo de Productor-Consumidor, un modelo muy simple y eficaz. Se explica en profundidad en la sección de tecnologías utilizadas.

Los datos almacenados en Kafka se pueden consultar y utilizar con diferentes herramientas. En este proyecto usamos Apache Spark, con su API PySpark. Nos hemos decantado por Spark, frente al resto de opciones porque Spark es la herramienta más rápida de todas. Spark trabaja en procesos distribuidos, con lo que, en caso de tener una carga de trabajo muy alta en un momento puntual, Spark distribuye la carga de trabajo entre todos los núcleos, haciendo así la tarea mucho más rápido que cualquier otra alternativa.

Spark tiene varias librerías nativas para procesar datos en streaming, y una de ellas está ideada para el uso de Kafka, dejándonos así una integración entre sistemas muy rápida y sencilla. Basta con configurar un par de parámetros y podemos conseguir que Spark se comunique con Kafka perfectamente. Spark puede ser utilizado tanto para producir datos como para consumirlos. En el caso de este proyecto, utilizaremos a Spark exclusivamente como consumidor de datos.

## 2.4– Deep Learning

### 2.4.1. Redes neuronales

Una red neuronal artificial es un sistema de computación inspirado indirectamente por las redes neuronales biológicas que se constituyen en el cerebro de los seres vivos. [07] Actualmente, en la disciplina informática de la inteligencia artificial, existe un subgrupo que estudia este tipo de arquitecturas y técnicas. A este subgrupo se le conoce como Deep Learning.

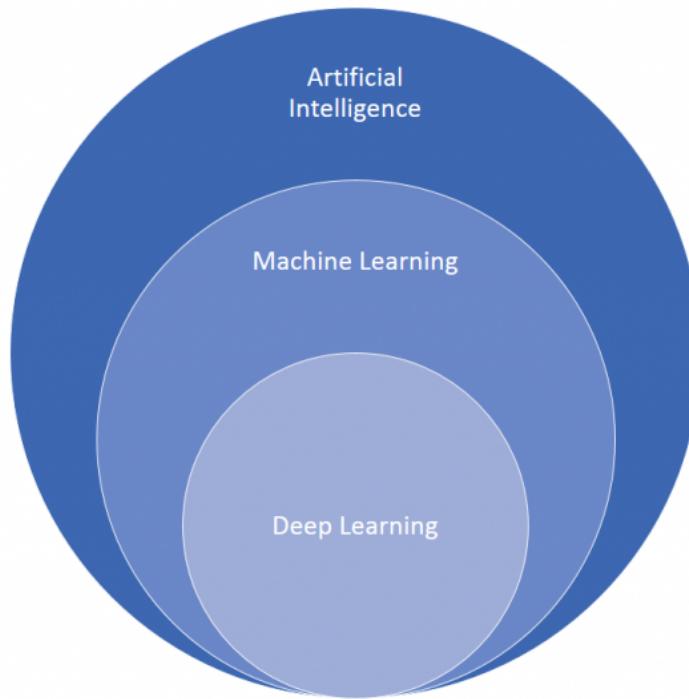


Figura 2.12: Deep Learning

Los algoritmos de Deep Learning tienen multitud de propósitos. Podemos destacar el tratamiento de imagen y vídeo, una técnica muy novedosa que está sirviendo en aplicaciones como la conducción autónoma, la reconstrucción de imágenes, el coloreado realista de imágenes en blanco y negro, etc. Son utilizados en multitud de áreas, siendo una de ellas las series temporales. Una serie temporal o cronológica es una secuencia de datos, observaciones o valores, medidos en determinados momentos y ordenados cronológicamente [08]. Lo más destacable de las series temporales, es que existe una dependencia entre la variación de los datos y el momento temporal al que se refieren o son tomados. Esto permite el descubrimiento de posibles patrones en los datos. Este proyecto enfoca el problema de la regresión vista como una serie temporal, donde cada valor depende totalmente del valor anterior, dando peso al momento al que se refiere.

Los conceptos básicos de una red neuronal, que veremos en profundidad a continuación son **la neurona, la arquitectura de la red, y el proceso de aprendizaje**.

### 2.4.1.1. La neurona

La neurona es la unidad mínima de procesamiento de una red neuronal. Es realmente una operación matemática que realiza la red, también recibe el nombre de **perceptrón simple**. Se puede ver su arquitectura en la figura 2.13

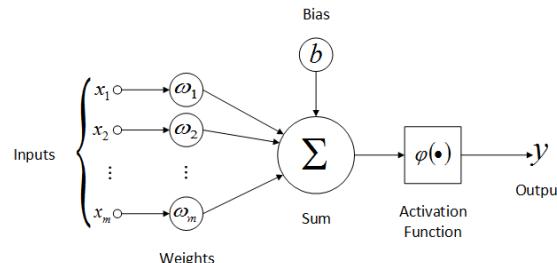


Figura 2.13: Diagrama de una neurona

Los elementos que la componen son, de izquierda a derecha:

- **Inputs:** Es la entrada de la neurona. Contiene la información que va a ser procesada. Esta información, puede estar en forma de **vector** o en forma de **tensor**. Podemos ver un ejemplo visual en la figura 2.14.



Figura 2.14: Estructura de datos

- **Weights:** Cada input tiene un peso asociado que determina la importancia del dato que entra.
- **Sum:** Aquí se realiza la regresión lineal, que se ha visto en la sección anterior, con la suma de pesos y entradas, más el bias, que hace de intersección, obtenemos la función de regresión lineal.
- **Función de activación:** Es el operador que se encarga de aplicar la no-linealidad a la salida de la neurona. Este operador es importante porque se da por supuesto en cualquier modelo predictivo, de cualquier tipo, o no existe una linealidad o no está implícita; por ello es necesario aplicarlo.
- **Output:** Es la salida de la neurona. Esta salida, realmente es la solución a la operación que realiza la neurona.

### 2.4.2. Funciones de activación

La función de activación tienen como objetivo convertir y/o eliminar la linealidad del vector resultante de cada neurona. Gracias a éstas, se puede obtener un resultado que permita aplicar regresión o clasificación a un problema.

Existen diferentes tipos de funciones de activación, pero las más importantes son las que se muestran a continuación.

#### 2.4.2.1. Función lineal

La función lineal o identidad es la función más simple de todas, dado que simplemente muestra el resultado obtenido en la neurona.

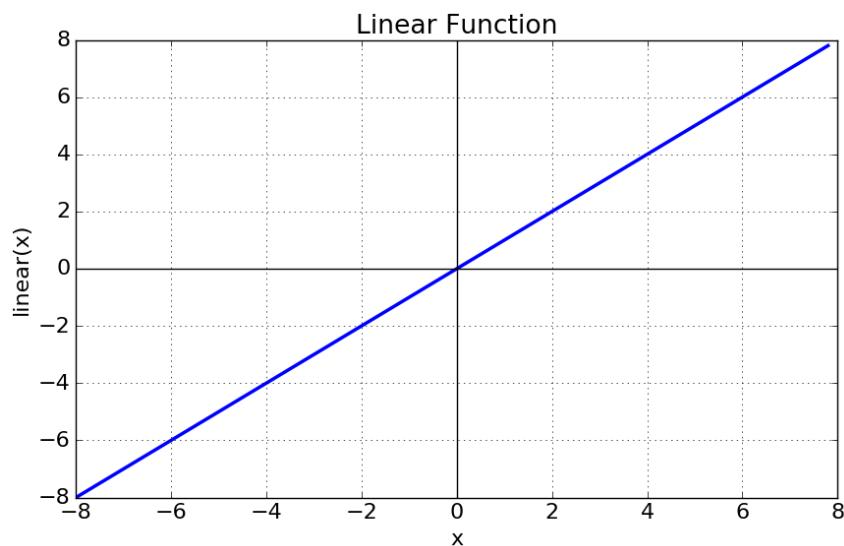


Figura 2.15: Función de activación lineal

<b>Ecuación</b>	$f(x) = x$
<b>Rango</b>	$[-\infty, \infty]$
<b>Características</b>	

Cuadro 2.3: Información sobre la función de activación lineal

### 2.4.2.2. Función sigmoide

La función sigmoide transforma los valores obtenidos en una escala (0,1), donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0.

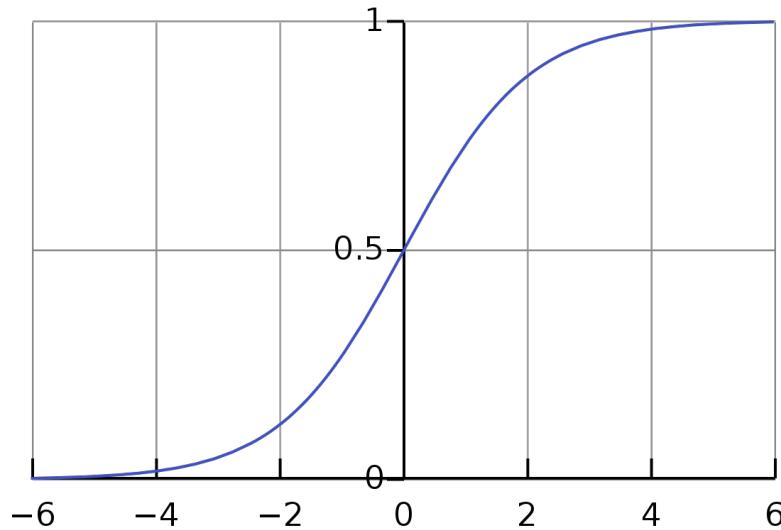


Figura 2.16: Función de activación sigmoide

<b>Ecuación</b>	$f(x) = \frac{1}{1 + e^{-x}}$
<b>Rango</b>	[0, 1]
<b>Características</b>	<ul style="list-style-type: none"> <li>• No es muy eficiente en términos de rendimiento, debido a que satura el gradiente.</li> <li>• Es la capa final en métodos de clasificación binaria.</li> <li>• No está centrada en el cero.</li> </ul>

Cuadro 2.4: Información sobre la función de activación sigmoide

### 2.4.2.3. Función tangencial hiperbólica

La función tangente hiperbólica transforma los valores introducidos a una escala (-1,1), donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1. Es similar a la sigmoide.

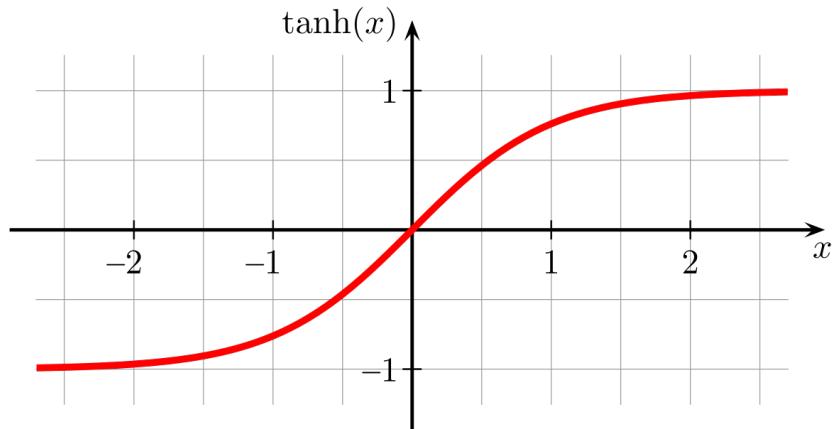


Figura 2.17: Función de activación tangencial hiperbólica

<b>Ecuación</b>	$f(x) = \frac{2}{1 + e^{-2x}} - 1$
<b>Rango</b>	$[-1, 1]$
<b>Características</b>	<ul style="list-style-type: none"> <li>Muy similar a la sigmoide.</li> <li>Se utiliza para decidir entre una opción y la contraria.</li> <li>Obtiene buen rendimiento en redes recurrentes.</li> <li>No está centrada en el cero.</li> </ul>

Cuadro 2.5: Información sobre la función de activación tangencial hiperbólica

#### 2.4.2.4. Función ReLu

La función ReLU transforma los valores introducidos, anulando los valores negativos, mientras que los positivos no los modifica.

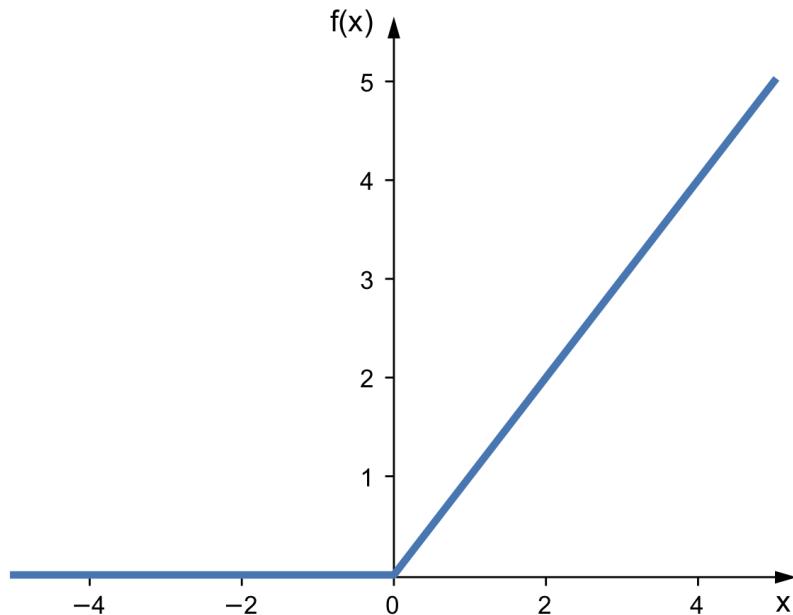


Figura 2.18: Función de activación ReLu

<b>Ecuación</b>	$f(x) = \max(0, x)$
<b>Rango</b>	$[0, \infty]$
<b>Características</b>	<ul style="list-style-type: none"><li>• Muy potente en términos de rendimiento.</li><li>• Solo se activa si el resultado es positivo.</li><li>• No está acotada.</li><li>• Puede inutilizar muchas neuronas en caso de overfitting.</li></ul>

Cuadro 2.6: Información sobre la función de activación ReLu

#### 2.4.2.5. Función Leaky ReLu

La función Leaky ReLU transforma los valores introducidos multiplicando los números negativos por un factor rectificante, mientras que los positivos no los modifica.

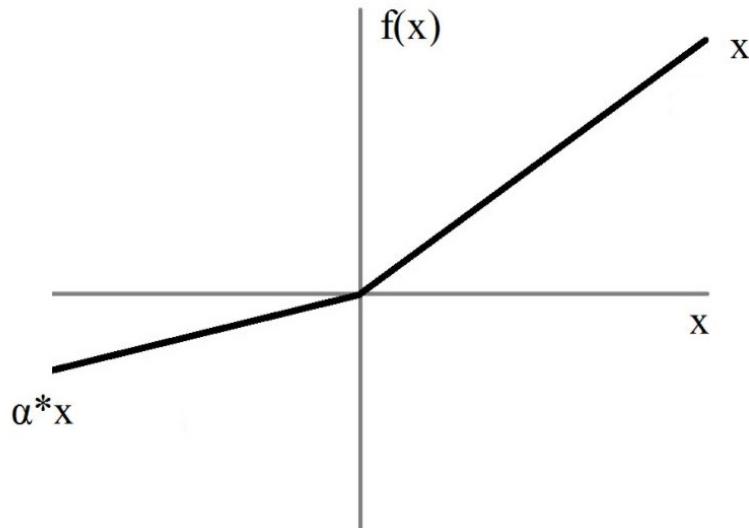


Figura 2.19: Función de activación Leaky ReLu

<b>Ecuación</b>	$f(x) = \max(0, 1*x, x)$
<b>Rango</b>	$[-\infty, \infty]$
<b>Características</b>	<ul style="list-style-type: none"> <li>• Muy similar a la función ReLu.</li> <li>• Muy potente en términos de rendimiento.</li> <li>• No está acotada.</li> <li>• Obtiene buen resultado cuando trata imágenes.</li> </ul>

Cuadro 2.7: Información sobre la función de activación Leaky ReLu

#### 2.4.2.6. Función Softmax

La función Softmax transforma las salidas a una representación vectorial en forma de probabilidades numéricas. El sumatorio de todas las probabilidades de las salidas es 1.

No se adjunta gráfico de esta función porque es imposible de representar de forma genérica, debido a que su cálculo depende de cada problema.

<b>Ecuación</b>	$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$
<b>Rango</b>	[0, 1]
<b>Características</b>	<ul style="list-style-type: none"> <li>• Función final en problemas de clasificación multiclas.</li> <li>• Es muy diferenciable.</li> </ul>

Cuadro 2.8: Información sobre la función de activación softmax

#### 2.4.3. Arquitectura de una red neuronal

Las redes neuronales tienen muchos tipos de arquitecturas. La más común, y en la que se basan las arquitecturas más importantes a día de hoy, es el **Multi layer perceptron (MLP)** o **Perceptrón multicapa**, el cual se puede observar en la figura 2.20.

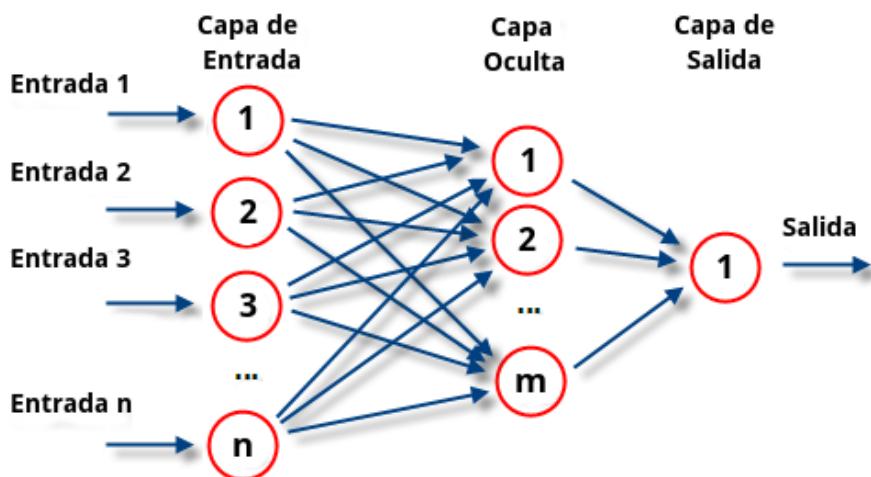


Figura 2.20: Arquitectura del perceptrón multicapa

Esta arquitectura es una mejora del **Perceptrón Simple**, y se caracteriza porque combina muchos perceptrones simples, permitiendo así encontrar relaciones entre **modelos linealmente independientes**. El perceptrón multicapa consta de una capa de entrada, una capa de salida y una o más capas ocultas. Dichas capas se unen de forma total hacia delante, es decir, la capa entrada se une con la primera capa oculta, esta con la siguiente y la última capa oculta se une con la capa de salida. Los valores que el perceptrón multicapa acepta son reales [09].

El funcionamiento de esta arquitectura es simple. Reciben un vector de datos de entrada, realiza las operaciones necesarias en la capa oculta, y obtienen el resultado por la capa de salida. Cada capa está compuesta de neuronas, que recordamos que son las encargadas de realizar las operaciones. Explicaremos su funcionamiento con un sencillo ejemplo:

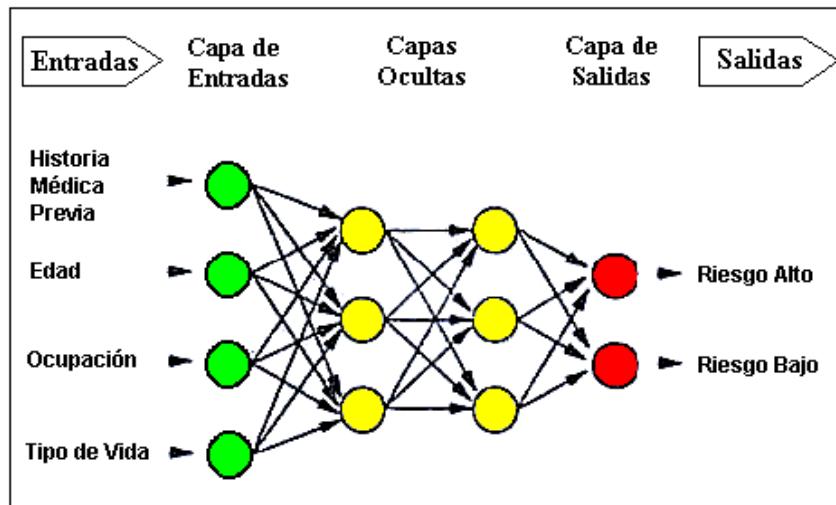


Figura 1. Esquema de una red neuronal antes del entrenamiento. Los círculos representan neuronas, mientras las flechas representan conexiones entre las neuronas.

Figura 2.21: Modelo MLP con datos de ejemplo

Existen unos datos de entrada, que contienen los datos de una persona. Estos datos se vectorizan, pasando por la entrada de la red. En las capas ocultas, cada neurona va a realizar una operación, la cual nosotros desconocemos, para poder encontrar relaciones entre las variables. Un ejemplo de relaciones posibles se puede ver en el cuadro 2.9

Operaciones posibles
Edad x Tipo de vida
Historial médico x edad
(Historial médico // ocupación) x tipo de vida

Cuadro 2.9: Operaciones posibles en una red neuronal

Aunque en un primer momento puedan parecer datos que no se relacionarán, bien por ser incoherentes entre ellos o bien por la magnitud de éstos, la red será la encargada de probar múltiples combinaciones pudiendo encontrar alguna que establezca una relación válida.

Actualmente no existe mucha información acerca del porqué de las combinaciones que se realizan dentro de un algoritmo de Deep Learning. A día de hoy lo poco que se sabe es que se realizan de forma aleatoria, mediante ensayo y error hasta encontrar combinaciones válidas. Para visualizarlo, en la figura 2.22 se puede observar el contenido de las capas ocultas usando el dataset MSNIT, que contiene imágenes de números escritos a mano. En esas capas, se realizan unos filtros para poder reconocer el contenido de las imágenes, pero para el ojo humano no carece de sentido.

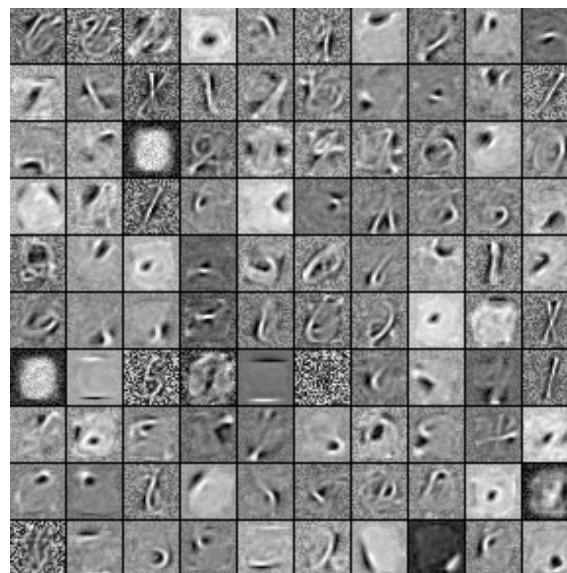


Figura 2.22: Contenido de las capas ocultas de una red neuronal con el dataset MSNIT

Una vez visto el Perceptrón Multicapa, vamos a hablar de las arquitecturas más utilizadas en la actualidad: las redes neuronales recurrentes y las redes neuronales convolucionales.

#### 2.4.4. Redes Neuronales Recurrentes

Una red neuronal recurrente tiene como característica principal la **retroalimentación de las salidas de cada neurona**. Las capas recurrentes utilizan como entrada un dato nuevo y la salida de una neurona anterior. Este proceso permite que las redes mantengan una memoria entre entradas pasadas y modele los problemas en base a su memoria. A diferencia de la neurona simple, vista anteriormente, este tipo de neuronas presentan la arquitectura que se muestra en la figura 2.23.

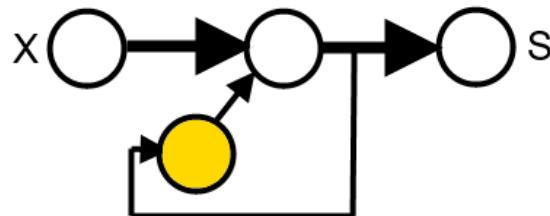


Figura 2.23: Esquema básico de una red neuronal recurrente

Siendo  $X$  la entrada y  $S$  la salida, se observa que, peculiarmente, la neurona central tiene una salida doble. Por un lado va a la salida y por otro lado va a una neurona de color amarilla. A esta neurona amarilla la llamaremos memoria, la cual se combina con los datos que vienen por  $X$ , generando así una salida. La memoria va almacenando datos de las entradas que han estado alimentando a la red, de tal forma que es capaz de condicionar la salida de la neurona en base a lo que ya ha visto durante el entrenamiento.

Este tipo de arquitectura tiene como ventaja principal la capacidad de aprender de los datos tomando como elemento fundamental el orden en el que los recibe. Existen problemas como el procesamiento de lenguaje natural donde esto es crucial, debido a que si se está desarrollando un algoritmo a que genere oraciones, tiene que saber que no es lo mismo “Invité a María y Juan a dar un paseo por el centro” que “María y centro el por Juan invitó paseo a un a dar” aunque ambas frases tengan las mismas palabras.

No obstante, este tipo de redes neuronales son más difíciles de entrenar que un perceptrón multicapa, debido a que si tienen un número elevado de neuronas por capa, en ocasiones los pesos obtienen un valor muy bajo acercándose al cero. Esto supone un problema cuando el volumen de datos es demasiado grande y todos los valores se encuentran en rangos muy acotados, por ejemplo:  $[0,00001, 0,00002]$ .

Otro caso de sobreajuste que suele ocurrir con mucha frecuencia, es cuando una red da como resultado el mismo valor que el último valor de una secuencia. Se puede entender fácilmente en la figura 2.24:

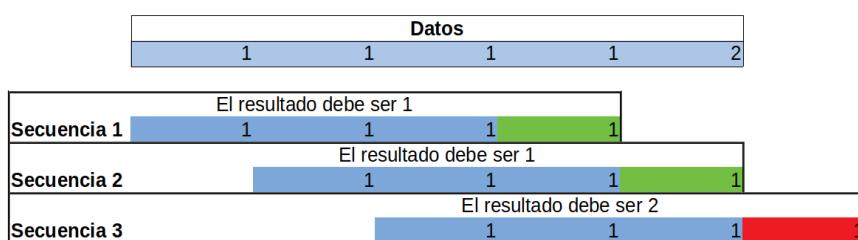


Figura 2.24: Ejemplo de overfitting en red neuronal recurrente

### 2.4.5. Capas de redes neuronales recurrentes

Las redes neuronales recurrentes están formadas por capas que utilizan la arquitectura de neurona recurrente explicada anteriormente. Aunque existen varias capas, las que cabe destacar son las **capas LSTM** y las **capas GRU**. En este proyecto, utilizamos únicamente las **capas LSTM**.

#### 2.4.5.1. LSTM

Las capas recurrentes LSTM, **Long-Short Term Memory**, son capas que **permiten la persistencia de datos** que ya ha recibido la red mediante el uso de **estados**. Dichos estados indican a la red qué información pueden **recordar u olvidar**, y cómo influirán los datos nuevos en función de los datos que ya conoce. La arquitectura de las neuronas de la capa LSTM se muestran en la figura 2.25:

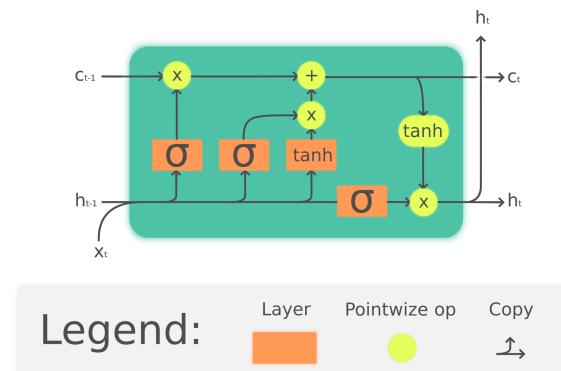


Figura 2.25: Diagrama que muestra la arquitectura de una neurona LSTM

Como se puede apreciar, esta neurona difiere mucho de la neurona clásica; ya que contiene varias entradas y varias salidas.

Entradas:

- $X_t$  : Dato nuevo.
- $H_{t-1}$ : Salida (resultado) de la neurona anterior.
- $C_{t-1}$  : Estado anterior de la neurona.

Salidas:

- $H_t$ : Salida (resultado) de la neurona. Esta salida es doble.
- $C_t$  : Estado nuevo de la neurona.

Señales:

- $\sigma$ : Señal sigmoide.
- $tanh$ : Señal tangencial.

La salida  $h_t$  es doble porque por un lado sale de la neurona para dar un resultado y por otro lado sale a la neurona nueva, convirtiéndose en la entrada  $h_{t-1}$  de la neurona nueva.

El proceso que se realiza en cada neurona se desarrolla en cuatro pasos, más un paso inicial.

- Paso 0 - Combinación inicial Se da lugar la combinación de  $X_t$  y  $H_{t-1}$ . Esta combinación puede interpretarse como una suma. A esta combinación la llamaremos **ENTRADA** durante la explicación.
- Paso 1 - Puerta de olvido.

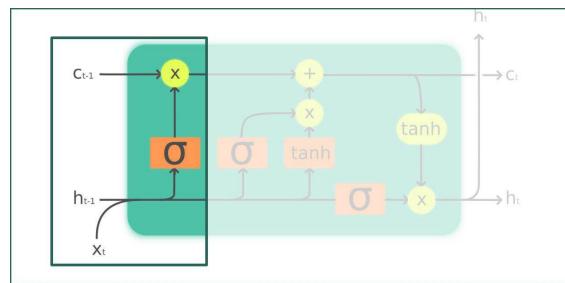


Figura 2.26: Primer paso en el procesamiento de una neurona LSTM

En este primer paso, la red va a decidir si va a olvidar la información que ya conoce o no. Ésto lo hace multiplicando el estado anterior  $C_{t-1}$  por la **ENTRADA**. El resultado de esa operación entra por una señal sigmoide, que tiene como resultado valores entre 0 y 1. Si son cercanos a 0 significa que va a olvidar información, y si son cercanos a 1 indica que va a mantener esa información.

- Paso 2 - Puerta de entrada

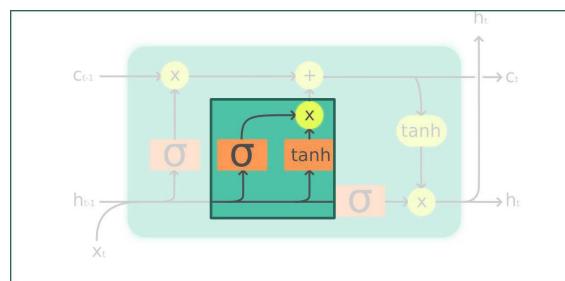


Figura 2.27: Segundo paso en el procesamiento de una neurona LSTM

Para actualizar el estado de la neurona, antes hay que pasar por la puerta de entrada. Primero, pasa la **ENTRADA** por una señal sigmoide. Esta decide qué valores van a ser actualizados. Los valores cercanos a 0 no serán importantes y los cercanos a 1, por el contrario, si lo serán. Además, la **ENTRADA** pasa por una función tangencial, que realiza la función de normalizar los valores entre -1 y 1 para ayudar a regularizar la red. A continuación, se multiplica la salida de la señal sigmoide y la señal tangencial. Los valores que salen de la señal sigmoide deciden qué valores de los que salen de la señal tangencial son importantes mantener.

- Paso 3 - Estado de la neurona

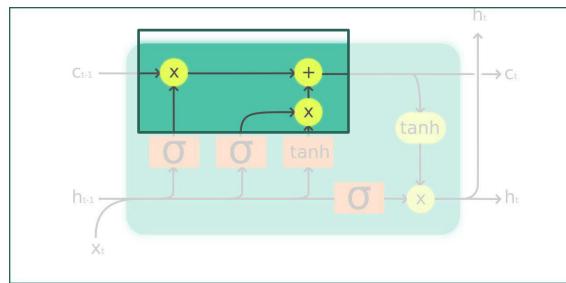


Figura 2.28: Tercer paso en el procesamiento de una neurona LSTM

Con la salida del paso 1 y el paso 2 ya tenemos información para calcular el nuevo estado de la neurona, a continuación la salida del paso 1 y 2 se combinan, siendo eliminados los valores cercanos a cero. Este estado nuevo ya contiene los pesos necesarios para saber qué información es relevante y cual no.

- Paso 4 - Puerta de salida

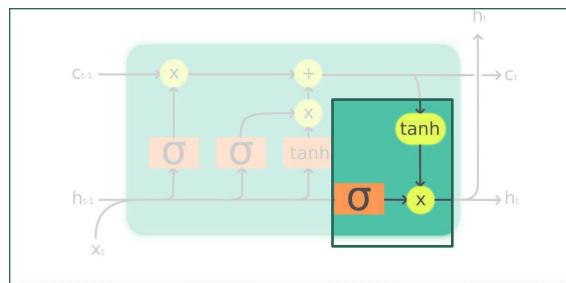


Figura 2.29: Último paso en el procesamiento de una neurona LSTM

Finalmente, tenemos la puerta de salida, que decide qué información saldrá como solución y como nueva entrada de la próxima neurona. Primero, el estado nuevo calculado en el paso 3, pasa por una señal tangencial. Después, la **ENTRADA** pasa por una señal sigmoide. El resultado de ambas señales se multiplican y obtenemos el vector resultado.

El vector resultado sale de la neurona como solución y como entrada de la próxima neurona. El estado nuevo, se transmite a la siguiente neurona.

Esta capa hay que verla como si todas las neuronas que la componen estuvieran conectadas en serie, debido a que este proceso, se repite por cada neurona que tenga la capa, como en la figura 2.30.

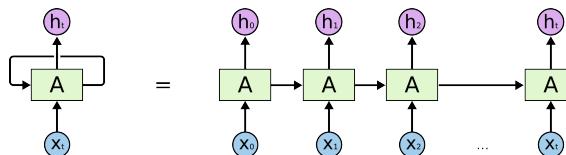


Figura 2.30: Esquema de neuronas conectadas secuencialmente

#### 2.4.5.2. Diferencia con la capa GRU

La capa GRU, **Gated Recurrent Unit**, es un modelo más sencillo de la capa LSTM. La diferencia principal entre ambas es que **las capas GRU no utilizan estados**, realizan operaciones directamente con la salida de la neurona anterior. Las capas GRU son más rápidas en términos de rendimiento, llegando a encontrar una solución válida antes que las capas LSTM, aunque su precisión es más baja.

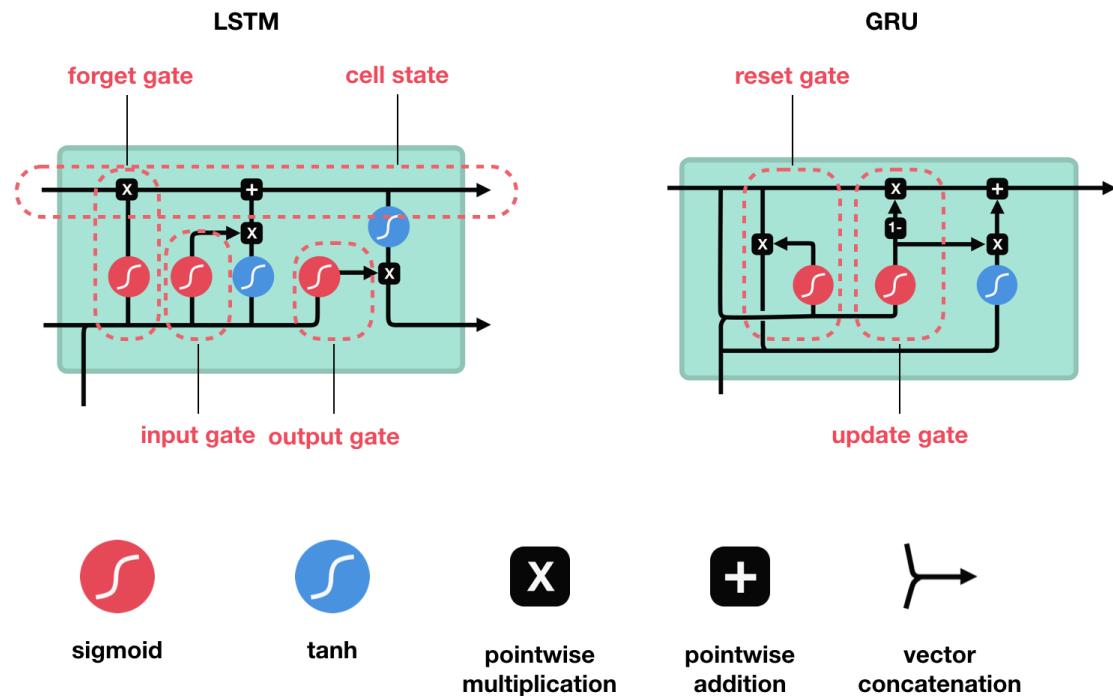


Figura 2.31: Comparación entre neuronas recurrentes GRU y LSTM

En este proyecto la precisión es un factor vital, por ello se han descartado las capas GRU para la arquitectura recurrente.

#### 2.4.6. Redes neuronales convolucionales

Las redes neuronales convolucionales son redes compuestas por múltiples capas de **filtros convolucionales** de una o más dimensiones. Una **convolución** es un operador matemático que transforma dos funciones,  $f$  y  $g$ , en una tercera función que en cierto sentido representa la magnitud en la que se superponen  $f$  y una versión trasladada e invertida de  $g$ . [10]

En la figura 3.8 se muestra un ejemplo de arquitectura convolucional:

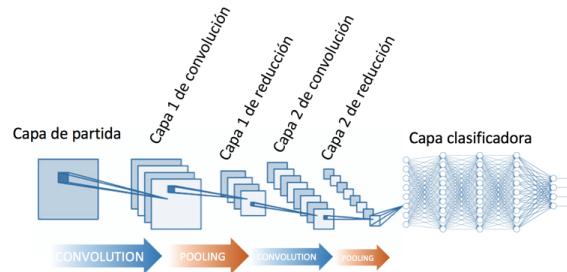


Figura 2.32: Arquitectura de una red neuronal Convolutacional

Su arquitectura es muy similar a la arquitectura del perceptrón multicapa. Sin embargo, los datos no son tratados de forma individual, sino como un **conjunto**. Las redes convolucionales intentan encontrar patrones en base a unos "mapas" de datos.

Este tipo de arquitectura ha demostrado un rendimiento muy alto en problemas de visión artificial. En la actualidad, se utilizan en problemas de clasificación de imágenes, recoloreado de imágenes, alteración de elementos en vídeo, etc. No obstante, también pueden ser utilizadas con datos estructurados.

### 2.4.7. Capas de las redes neuronales de convolución

La variedad de capas convolucionales es mayor que la de las capas recurrentes, debido a que existen las mismas capas con funcionalidades diferentes para varias dimensiones de datos.

#### 2.4.7.1. Capa Conv1D

La capa de convolución en una dimensión aplica a los datos de entrada un conjunto de filtros que son objeto de aprendizaje para obtener un mapa de características de los datos originales. En la figura 2.33 se puede ver un ejemplo:

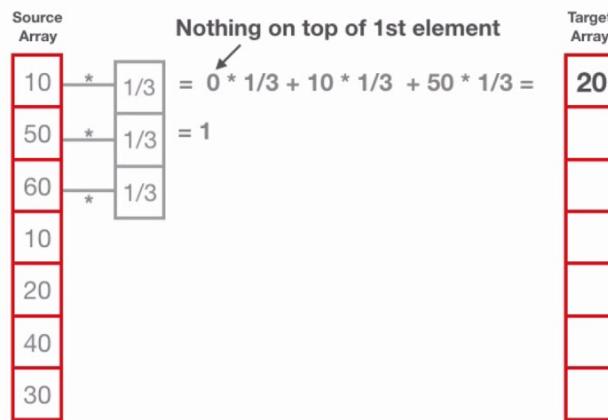


Figura 2.33: Ejemplo de convolución en una dimensión

La capa ha obtenido un conjunto reducido de los datos de entrada y ha realizado una operación con ellos que devuelve un único dato como solución.

Existen tres factores fundamentales a la hora de realizar una convolución:

- **Profundidad:** Es el número de filtros que se desea usar, los distintos mapas de características que deseamos obtener, cada uno intentando buscar patrones diferentes.
- **Stride:** Indica el número de datos que avanza la ventana deslizante de convolución en cada cálculo. Cuando el stride toma valor 1, el filtro se mueve de uno en uno manteniendo el tamaño constante. Sin embargo, cuando el stride toma valor 2, el filtro avanza dos posiciones, resultando en una reducción del volumen de salida.
- **Padding:** El uso de esta característica permite añadir datos extra para preservar el tamaño original de la entrada. De esta forma se evita descartar datos que pueden ser útiles en los cálculos. Estos datos nuevos normalmente suelen tomar valor 0, valor que no influye en el resultado.

En la figura 2.34 se puede ver un ejemplo de Stride igual a 1 (izquierda), y Stride igual 2 (derecha).

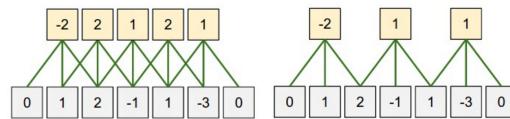


Figura 2.34: Ejemplo de convolución con Stride tomando valor 1 y 2

Se puede ver un ejemplo con una animación en este *enlace*.

#### 2.4.7.2. Capa MaxPooling1D

La capa **MaxPooling** es un tipo de capa convolucional en la que se **reduce la dimensionalidad de un vector** de datos. Existen tres tipos de capas: **1D**, para una dimensión; **2D**, para dos dimensiones; y **3D** para tres dimensiones.

Esta capa realiza la operación de seleccionar el valor más grande de un subconjunto de valores del vector, los cuales salen de la capa como vector resultado como podemos ver en la figura 2.35

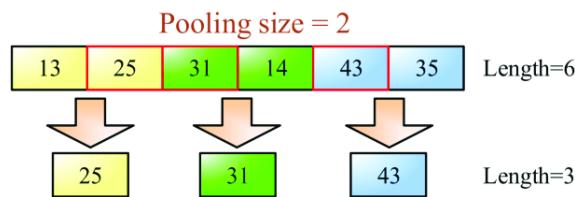


Figura 2.35: Ejemplo del funcionamiento de la capa Max Pooling

### 2.4.8. Capas comunes entre arquitecturas

#### 2.4.8.1. Capa Dense

La capa **Dense** es una capa que realiza una conexión completa entre todas las neuronas que la conforman. Realiza un producto cartesiano entre todas las neuronas para obtener resultados, probando diferentes combinaciones.

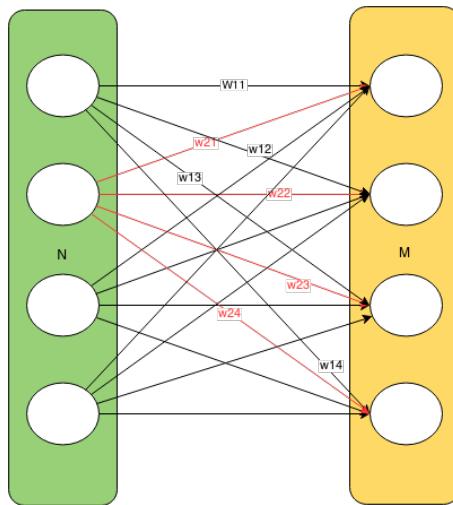


Figura 2.36: Diagrama de la arquitectura de una capa Dense

Como se puede ver en la figura 2.36, la capa de entrada N tiene 4 neuronas y se conecta con la capa de salida M, que tiene 4 neuronas. El resultado de esta capa es el producto cartesiano de ambas.

#### 2.4.8.2. Capa Flatten

La capa **Flatten** es una capa de utilidad de las redes neuronales la cual se utiliza para regularizar un vector cambiando su formato. Esta capa se utiliza cuando la salida de una capa no se puede conectar a la entrada de otra. En la figura 2.37 se puede ver una demostración.

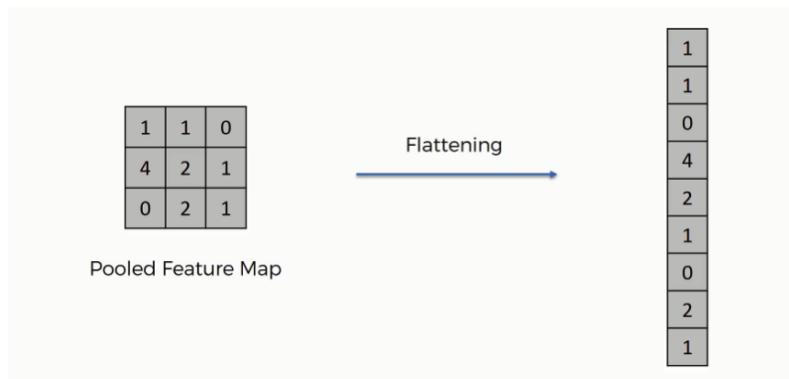


Figura 2.37: Ejemplo de funcionamiento de la capa Flatten

#### 2.4.8.3. Capa Dropout

La capa **Dropout** es una capa de utilidad que elimina aleatoriamente los resultados de una neurona. Aunque puede parecer algo negativo o contraproducente, es bastante importante debido a que controla el sobreajuste de aprendizaje. En la figura 2.38 se puede ver un ejemplo de una red con capas densas, la imagen a la izquierda, y a la derecha está la misma red neuronal pero con capas Dropout tras cada capa.

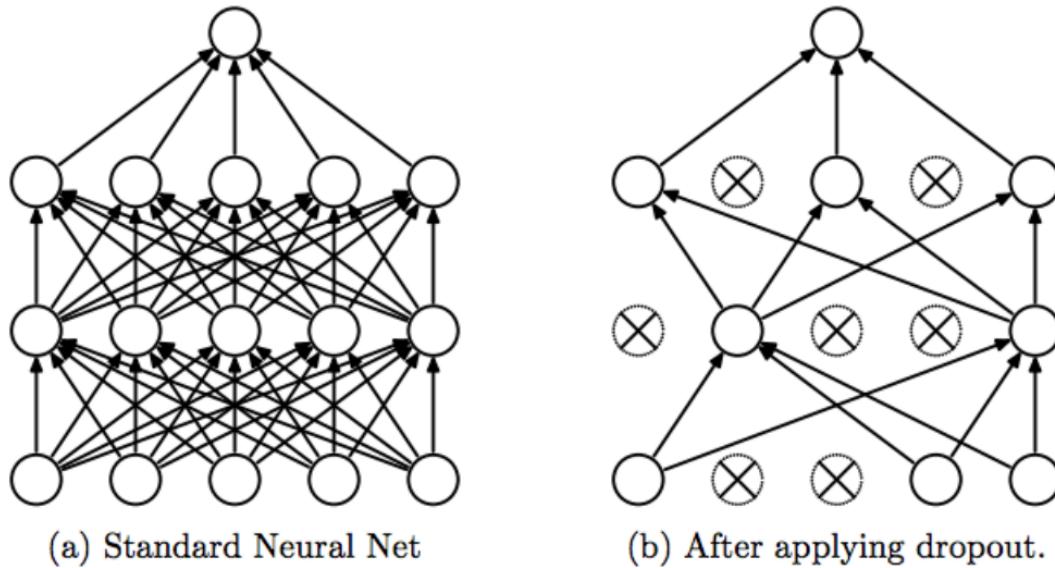


Figura 2.38: Diagrama explicativo de la función de la capa Dropout

De esta forma, se le obliga a la red que elija las combinaciones que más generalizan en ésta, obteniendo un resultado más preciso en la solución.

#### 2.4.9. Entrenamiento de las redes neuronales

Cuando nos referimos a que una red está aprendiendo de unos valores, se está hablando del valor de sus weights, sus pesos. Cada neurona está conectada con otra mediante un enlace, el cual tiene asociado un peso que indica la relación entre ambas neuronas. El valor de este peso se va alterando en función de los datos que la red va recibiendo. Este proceso de alteración del valor del peso, en busca de un valor perfecto que relacione todas las neuronas, se denomina entrenamiento.

En el ámbito del Machine Learning, se conoce por aprendizaje supervisado a aquel problema en el que, en el momento del entrenamiento, se le da al algoritmo la solución de una operación, con el fin de que la red vea el error cometido de su predicción contra la realidad y ella misma pueda ajustarse. Todos los algoritmos de Deep Learning tienen un aprendizaje supervisado.

Un ejemplo para entender el proceso de forma más clara puede ser el siguiente: Tengo seis imágenes. Dos de ellas son imágenes de manzanas, otras dos son imágenes de plátanos, y las dos últimas son imágenes de naranjas. En el momento del procesado de la información, se traslada cada imagen a una matriz que contiene en cada celda la información de un pixel. Para una imagen de 40x40 píxeles, tengo una matriz de 40x40x3. Estos 3 contienen la información del pixel RGB. La red, al recibir este tensor, no es capaz de saber de qué se le está hablando. Por ello, se crea un segundo conjunto de datos que contiene el resultado de la operación, pero de forma mapeada. Tendremos un vector de 3 posiciones, en el que todos los valores son 0 salvo el valor que hace referencia a la fruta que queremos adivinar.

- $[1, 0, 0]$  = Manzana
- $[0, 1, 0]$  = Plátano
- $[0, 0, 1]$  = Naranja

Si es una naranja, tendrá el vector que refiera a la naranja, en este caso es  $[0, 0, 1]$ .

La red debe de realizar las operaciones que estime necesarias para llegar a uno de esos tres vectores. Por ejemplo, si recibe una imagen de una manzana tiene que dar como vector resultado el vector  $[1, 0, 0]$ .

Para entender cómo se realiza este proceso de aprendizaje, debemos hablar de dos conceptos fundamentales: **el descenso del gradiente y la retropropagación**.

### 2.4.9.1. Descenso del gradiente

El descenso del gradiente, o **Gradient Descent**, es un algoritmo de entrenamiento simple en el que se busca la dirección del máximo descenso en un gradiente que representa todos los valores de una función de coste.

En la figura 2.39 se puede ver el descenso más grande en un gradiente.

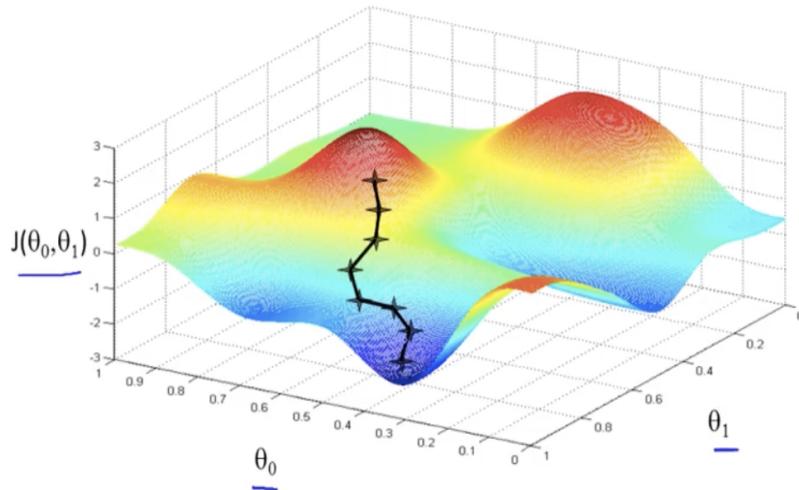


Figura 2.39: Diagrama de descenso del gradiente

En este gradiente, el error mínimo se representa con el color azul, y el más grande del color rojo. El algoritmo gradient descent busca el punto azul más profundo, pero, como se puede ver en la imagen, no hay un único valor azul.

Ésto hace referencia a los mínimos locales y mínimos globales. Dichos términos se refieren a ocasiones en las que la red ha encontrado un punto de aprendizaje donde su error es bajo, un mínimo local. Sin embargo, en ese punto se siguen cometiendo errores, ya que no es el estado perfecto de la red. Por todo ello, hay que seguir entrenando la red para encontrar el error más bajo de todos, el mínimo global.

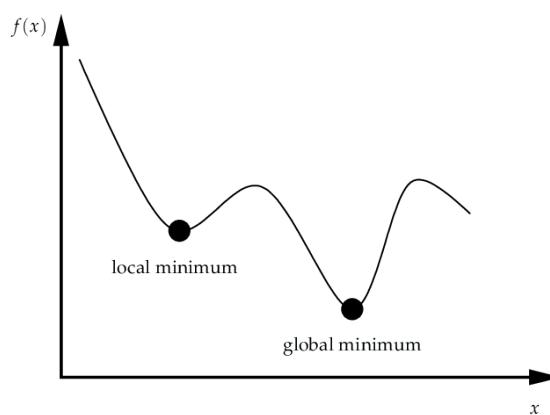


Figura 2.40: Diagrama explicativo de un mínimo local y el mínimo global

Para llegar hasta ese punto, hay que tener en consideración diferentes aspectos, entre los cuales se encuentra uno fundamental, siendo éste el ratio de aprendizaje. El ratio de aprendizaje es el porcentaje de cambios que va a realizar en sus pesos para encontrar una solución. Para encontrar este ratio no hay una ley escrita, puesto que si das un ratio muy grande es posible que el algoritmo no pueda encontrar un punto intermedio entre dos posibles soluciones, y si das un valor de ratio de aprendizaje bajo, la red tardará muchísimo tiempo encontrar una solución.

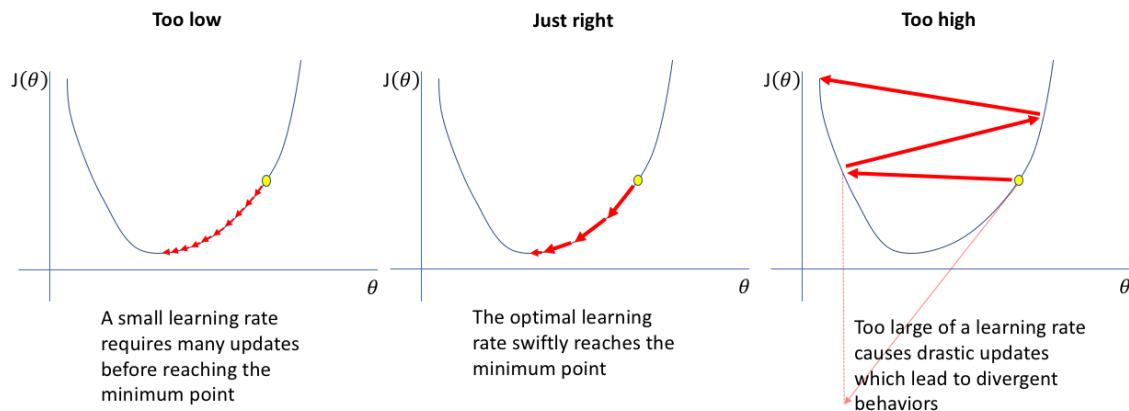


Figura 2.41: Diagrama explicativo de la diferencia entre learning rate bajo, medio y alto

Existen diferentes tipos de Gradient Descent:

- **Batch Gradient Descent:** Se utiliza todo el conjunto de datos de entrenamiento para calcular el gradiente de la función de coste.
- **Mini-Batch Gradient Descent:** Se utiliza un conjunto reducido de los datos de entrenamiento de  $n$  elementos para calcular el gradiente de la función de coste.
- **Stochastic Gradient Descent (SGD):** Se utiliza un único elemento del conjunto de entrenamiento para calcular el gradiente.

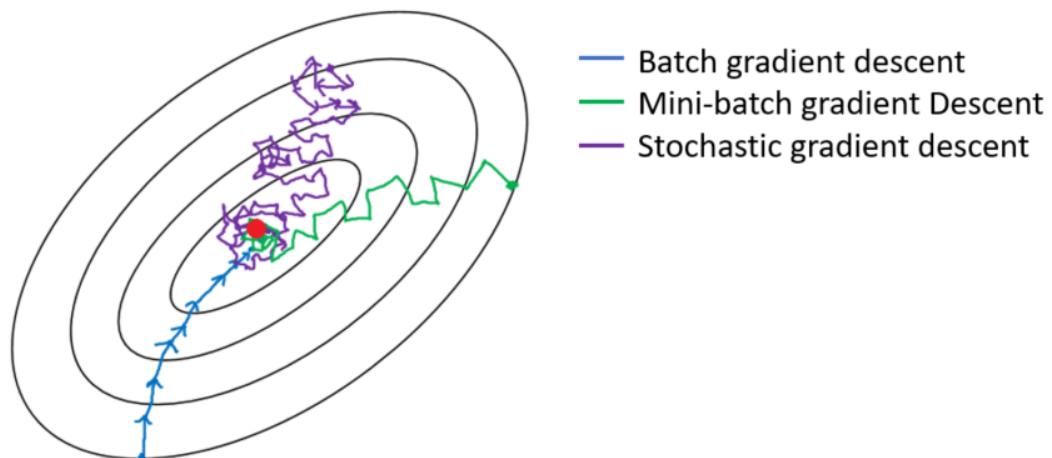


Figura 2.42: Comparativa entre tipos de gradient descent

### 2.4.9.2. Retropropagación

El algoritmo de retropropagación, o **Backpropagation**, calcula y ajusta el valor de los pesos de la red neuronal, en base a una función de coste.

Este algoritmo se realiza en cuatro procesos:

- 1 - Cálculo de solución. En primer lugar, se realiza un cálculo con un dato de entrada, pasando por todas las capas de la red hasta llegar a la solución. Durante este proceso se almacenan los pesos.
- 2 - Cálculo de error. Una vez que tenemos un dato de salida, se valida el resultado con su solución real, la cual conocemos. Ésto genera un error mediante la función de coste definida.
- 3 - Propagación hacia atrás. El error generado en el paso 2 recorre la red neuronal en orden inverso. Este error se calcula con el gradiente explicado anteriormente, capa por capa, dando así un error para cada peso de cada neurona.
- 4 - Actualización de pesos. Una vez que el error se ha propagado hasta el inicio, tenemos el error que ha cometido cada peso en cada una de las capas. A continuación, se recalculan los pesos en base a los errores recibidos por la retropropagación.

En la figura 2.43 se puede ver la orientación de los pasos 1 y 3.

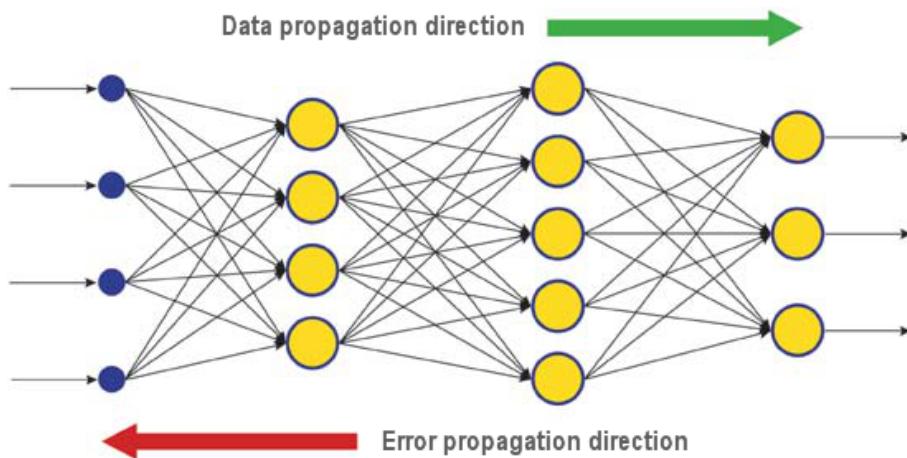


Figura 2.43: Diagrama explicativo de Backpropagation

La función de coste, o **loss function**, es una función en la que se define el error cometido en las predicciones de la red, en base a cuánto se han alejado de la predicción correcta. La función de coste se puede definir de diferentes formas. Una de las más comunes es el error cuadrático medio (MSE: Mean Squared Error), que se expresa utilizando la siguiente fórmula:

$$MSE = \sum \frac{1}{2} (y_{pred} - y_{real})^2$$

Siendo  $y_{pred}$  el valor obtenido en las predicciones de la red y  $y_{real}$  el valor real.

Una vez definida la función de coste, este problema adquiere una nueva dimensión, convirtiéndose también en un problema de optimización, cuyo objetivo es minimizar el error cometido.

La red debe encontrar cuáles son los pesos que más influyen al error y modificarlos para reducir el coste lo máximo posible. Ésto se hace aplicando el método del descenso del gradiente.



# CAPÍTULO 3

## Diseño del sistema

El objetivo principal del sistema es proveer un modelo de aprendizaje profundo que genere predicciones sobre el mercado Forex. Antes de entrar en cuestiones de diseño, tenemos que hablar acerca del flujo lógico por el que debe pasar la información.

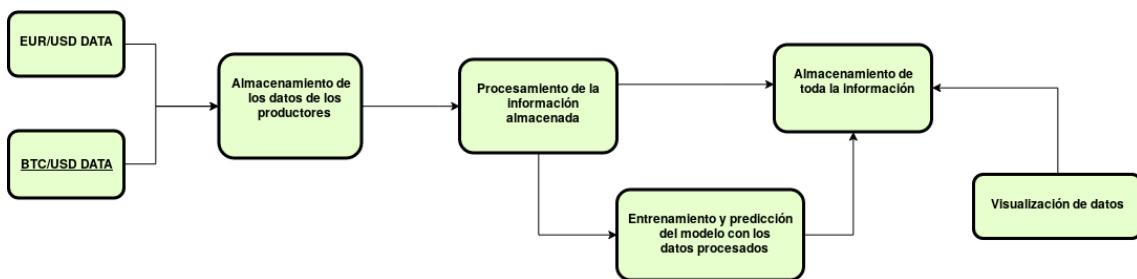


Figura 3.1: Diagrama de diseño de flujo de la información

En la figura 3.1 se ilustra del diseño el sistema y el flujo de la información a través del mismo.

- La información es generada por los productores, uno por divisa, la cual es enviada al gestor de cola de mensajes.
- El gestor almacena temporalmente todos los mensajes recibidos por los productores hasta que son requeridos.
- La información viaja al módulo de procesamiento, donde se realizan las transformaciones necesarias sobre los datos.
- Una vez se tienen los datos tratados se almacenan en una base de datos donde se ponen a disposición de las redes neuronales.
- Las redes toman los datos para su entrenamiento y generan las predicciones. Se genera una predicción para los siguientes cuarenta y cinco minutos por red y divisa.
- Las predicciones se almacenan en la base de datos donde una interfaz de usuario web puede acceder a ellas y pueden ser representadas.

Una vez visto el flujo del sistema, se ha agrupado las tareas del sistema en cuatro grupos, estos son:

- **Obtención:** Se necesitan datos históricos de las divisas para poder entrenar el modelo.
- **Tratamiento:** La información obtenida necesita ser tratada antes de que el modelo pueda aprender de ella.
- **Almacenamiento:** Como cualquier sistema de información se necesita un sistema de almacenamiento permanente.
- **Construcción del modelo:** El desarrollo y entrenamiento del modelo.
- **Visualización:** Los datos generados por el modelo se deben poder representar correctamente para su monitorización e interpretación.

Para resolver todos estos problemas se ha diseñado una arquitectura modular en la que cada módulo resuelve una de las problemáticas vistas anteriormente. Se ha optado por esta arquitectura ya que el desacoplamiento de los componentes permiten, por una parte, un diseño, implementación y testeo sencillos y por, otra parte, permite aumentar el número de instancias en ejecución de cada módulo en caso de necesidad, usando tecnología de contenedores. Los módulos que componen el sistema son:

- Productores de datos
- Procesado y almacenamiento
- Red neuronal
- Monitorización

Todos ellos serán explicados en este capítulo.

### 3.1– Tecnologías troncales

Existen algunas tecnologías comunes a todo el desarrollo, las cuales se encuentran en menor o mayor medida en todo el sistema, las cuales son las tecnologías troncales.

#### 3.1.1. Python

Python es un lenguaje de programación interpretado multiparadigma y multiplataforma de tipado dinámico.

Python junto con R son los lenguajes predominantes en el campo del Machine Learning y el Deep Learning. En el caso de Python, su éxito se debe a varios factores, como son su sintaxis sencilla, la cual favorece un código legible y un desarrollo ágil, facilidad para adaptar librerías de lenguajes de más bajo nivel, como son C, C++ o CUDA, los cuales tienen un desempeño mejor pero no es viable implementar la totalidad de los modelos con ello y por una gran variedad de librerías y frameworks open source. También hay que tener en cuenta el apoyo de grandes empresas como Google, con su framework TensorFlow, siendo uno de los frameworks más utilizados para Deep Learning.

Se ha elegido este lenguaje tanto por las razones previamente presentadas como porque, al ser un lenguaje multiparadigma usado en campos tan variados como el desarrollo web, scripting, machine learning, ciberseguridad, matemáticas, física, etc., nos permite implementar la totalidad del sistema con él, permitiendo una integración más sencilla de los componentes.

#### 3.1.2. Docker

Docker es un proyecto para la automatización y el despliegue de aplicaciones dentro de contenedores de software que proporciona una capa extra de abstracción y virtualización de aplicaciones en múltiples sistemas operativos. Gracias a docker no es necesario virtualizar sistemas operativos completos, como ocurre con las máquinas virtuales, sino que se pueden virtualizar únicamente los componentes que sean necesarios.

Varios módulos del sistema se han virtualizado con docker, lo que permite un despliegue sencillo en cualquier servicio de hosting con soporte de contenedores, como es Amazon Web Services.

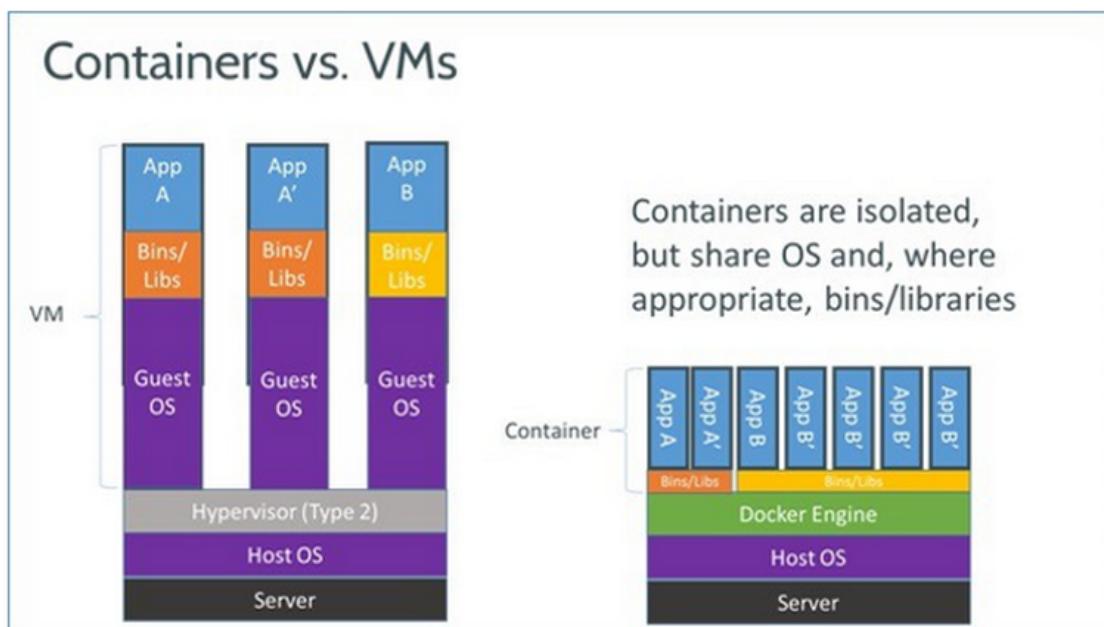


Figura 3.2: Docker VS VM

## 3.2– Productores de datos

El módulo de producción de datos es el encargado de recolectar la información necesaria para el funcionamiento del sistema.

En este caso, los datos necesarios son los valores del **EUR/USD** y los valores del **BTC/USD**. En el mercado existen diversas herramientas para obtener la información, pero éstas han sido descartadas, bien por ser herramientas de pago o debido a que la calidad y precisión del dato no es la suficiente. Finalmente, después de sopesar las opciones disponibles, se ha decidido obtener la información de la página <https://es.investing.com/> mediante **web scraping**.

### 3.2.1. Extracción del dato

Para conseguir la información de las divisas desde la fuente de datos necesitamos primero descargar el fichero HTML resultante de realizar una petición HTTP a las URLs:

- EUR/USD - <https://es.investing.com/currencies/eur-usd>
- BTC/USD - <https://es.investing.com/crypto/bitcoin/btc-usd>

Una vez descargados los ficheros es necesario un estudio de los mismos, para detectar así que elementos HTML contienen la información deseada. En este caso, el valor de la divisa se encuentra dentro de un elemento span con identificador *last.last*. Debido a que la estructura de los ficheros es idéntica ha sido posible generalizar la extracción de información para cualquier divisa de la página web, por lo que el número de divisas con las que opera el sistema es altamente escalable.

### 3.2.2. Envío de la información

Una vez se ha obtenido el valor de la divisa éste necesita ser enviado al módulo siguiente. Para gestionar todos los mensajes generados por los productores de datos se necesita un gestor de cola de mensajes, el cual actúa de intermediario entre los productores de datos y el resto del sistema, asegurando que los mensajes enviados llegan a los componentes que los necesiten. Actualmente el flujo de información generado por los productores es ínfimo pero una vez se amplíe el número de divisas y el número de fuentes de información se puede presentar una situación en la que se produzcan problemas de congestión y perdida de mensajes.

### 3.2.3. Repetición

Para obtener un flujo continuo de información se necesita que los dos pasos explicados anteriormente se repitan en el tiempo a intervalos regulares. Debido a la velocidad de las operaciones del mercado Forex se ha configurado la extracción de información cada minuto.

### 3.2.4. Tecnologías utilizadas

Para implementar las distintas partes de este módulo se ha hecho uso de las siguientes tecnologías

- BeautifulSoup
- Requests
- Confluent-kafka
- Schedule
- Docker

### BeautifulSoup

Beautifulsoup es una librería de Python para el análisis y extracción de datos en ficheros HTML. Ésta crea una estructura en árbol que puede ser utilizada para la extracción de información de ficheros HTML.

```
1  from bs4 import BeautifulSoup
2
3  soup = BeautifulSoup(response.content, features="html.parser")
4  res = soup.find('span')
```

Código 3.1: Búsqueda de elementos span

### Requests

Requests es una librería Python que permite realizar peticiones HTTP, pensada para ser usada por humanos, por lo que presenta una sintaxis sencilla y legible. La librería soporta desde los métodos HTTP básicos, como POST o GET, hasta gestión de cookies, SSL, proxy y streaming.

Esta librería ha sido utilizada para descargar los ficheros HTML que después serán procesados por beautifulsoup.

### Confluent-kafka

Confluent-kafka es la librería que nos provee de los métodos necesarios para conectarnos al componente Apache kafka del siguiente módulo. Esta librería es necesaria, ya que registra al scraper como un producer de kafka y permite que la información enviada para ellos se gestione y se distribuya adecuadamente.

### Schedule

Schedule es una librería Python para la programación de tareas periódicas, permitiendo ejecutar funciones en períodos personalizables.

### Docker

Se ha creado una imagen de docker genérica que implementa las tareas necesarias para la extracción de los datos. Para inicializar la imagen solo es necesario proporcionar el nombre de la divisa que se quiere obtener.

### 3.3– Procesado y almacenamiento

El módulo de procesamiento y almacenamiento de datos se ocupa de recoger la información generada por el resto de módulos, tratarla, almacenarla y enviarla a los módulos que la necesiten.

Las tareas que desempeña este módulo son:

- Recepción.
- Tratamiento.
- Almacenamiento.

#### 3.3.1. Recepción

El módulo recibe directamente la información en bruto de los productores. Para que no se produzca una saturación entre mensajes y puedan perderse se necesita un gestor de colas de mensajes. Este gestor recibe los mensajes de los productores y los almacena temporalmente, agrupándolos por topics previamente configurados, hasta que son demandados, momento en el cual el gestor envía los mensajes hasta los demandantes.

Este proceso se realiza en Apache Kafka.

#### 3.3.2. Tratamiento

Una vez obtenida la información es necesario tratarla para poder alimentar la red neuronal con ella. La información de cada divisa se encuentran en grupos de quince minutos, y pasa un proceso de procesado que consta de los siguientes pasos:

1. **Obtención del dato.** Se elimina toda la información no necesaria, como posibles letras, guiones o espacios, y se almacena en los diferentes campos de la fila del dataset.
2. **Casteo a número real.** Los datos numéricos que nos llegan cambian de tipo para poder realizar operaciones matemáticas de coma flotante.
3. **Casteo de fechas.** Los datos en forma de fecha que llegan al sistema se castean a un tipo Datetime, que consta de fecha y hora, para poder utilizarlo fácilmente a la hora de hacer búsquedas.
4. **Ordenado de dataframe.** Como medida de seguridad, se ordena el dataset de forma creciente tomando como criterio la fecha.
5. **Retorno logarítmico.** Se calcula el retorno logarítmico entre los valores que han llegado, para facilitarle los cálculos al módulo de la red neuronal.
6. **Almacenamiento.** Como paso final, se almacenan los datos en la base de datos.

Este proceso se realiza en Apache Spark.

#### 3.3.3. Almacenamiento

Con la información ya agrupada y tratada se procede a almacenarla en un sistema de almacenamiento a largo plazo donde el resto de módulos puedan consultarlas.

Este proceso se realiza en MySQL

### 3.3.3.1. Estructura de la base de datos

Las tablas de la base de datos se separan en tres grupos: predicciones, tiempo real e histórico.

Las tablas de predicciones almacenan los resultados generados por las redes neuronales. Hay cuatro tablas de este tipo, una por cada par red/divisa.

Las tablas de tiempo real almacenan los datos actuales hasta un máximo de quince en el pasado. Por tanto, si el sistema se encuentra en el instante t10 esta tabla contará con los valores del momento actual y los 9 instantes pasados, pero al llegar al instante t16 la tabla vuelca sus datos en las tablas de histórico, conservando únicamente el valor del instante actual. Existe una tabla de este tipo por divisa

En las tablas de histórico se guardan los datos pasados de las divisas. Existe una tabla por divisa.

### 3.3.4. Tecnologías utilizadas

Por cada tarea a cumplir se han desplegado tres sistemas que cumplen las exigencias necesarias. Estos sistemas son:

- Apache Kafka
- Apache Spark
- Base de datos MySQL
- Docker

Los componentes Apache Spark y Apache Kafka se explicarán en más detalle en el capítulo 4.

#### 3.3.4.1. MySQL

Para el almacenamiento a largo plazo de los datos se ha optado por una base de datos MySQL.

MySQL es un sistema de gestión de bases de datos relacional, desarrollado por Oracle Corporation bajo una licencia dual pública general/comercial. MySQL está desarrollado en C y C++ y destaca por su gran adaptación a diferentes entornos, permitiendo su integración con los lenguajes de programación más utilizados como Java y Python.

#### 3.3.4.2. Docker

Se ha utilizado Docker para virtualizar tanto el Apache Kafka como la base de datos MySQL. El hecho de virtualizar Apache Kafka permite desplegar instancias del mismo a demanda con relativa facilidad.

## 3.4– Red neuronal

Las redes neuronales son el pilar fundamental del presente proyecto, ya que son el sistema encargado de predecir nuevos movimientos bursátiles, en base a un histórico conocido.

En este proyecto se van a desarrollar dos arquitecturas diferentes de redes neuronales, siguiendo una de ellas la arquitectura **recurrente** y la otra la arquitectura **convolucional**.

### 3.4.1. Consideraciones previas

Antes de explicar la arquitectura de las redes neuronales, debemos entender dos conceptos fundamentales.

Predecir movimientos bursátiles es un problema de regresión aplicado a series temporales. En dichas series suelen confundirse los conceptos de pasado, presente y futuro. Cuando se habla de **presente** no se está hablando del presente actual, ni del momento en el que se lee este párrafo sino al **momento temporal de referencia**, que por convención se le llama **t0**. Expresado de otro modo, cada vez que vamos a realizar una operación sobre una serie temporal se debe tomar un momento de referencia y una vez se selecciona debemos imaginar el contexto del problema como si el tiempo se parase. Es decir, si se dice que se van a realizar operaciones durante un momento temporal, el tiempo que se tarda en realizar esas operaciones **no se tiene en cuenta**.

#### Vector de entrada

El vector de entrada de la red tiene una longitud temporal de **tres horas**, o lo que es lo mismo **cientos ochenta minutos**. Este vector contiene información del valor de la moneda 3 horas antes a t0.

Para exemplificarlo, consideramos t0 el día de hoy las 18:00. Una vez elegido t0, se crea una secuencia con todos los valores de la moneda desde t-180 hasta t-1, es decir, todos los valores de la moneda por minuto desde las 15:00 hasta las 17:59. Dicho ejemplo se presenta en la figura 3.3.

Hora	15:00	15:01	15:02	15:03		17:57	17:58	17:59	
Valor	1,23345	1,23389	1,23402	1,23394	...	...	1,20210	1,20345	1,20439
Array index	0	1	2	3		177	178	179	

Figura 3.3: Ejemplo del vector de entrada con un momento temporal concreto

Esta secuencia se convierte en un vector, y así creamos **vector de entrada**.

#### Vector resultado

El vector resultado de la red contiene la información de la predicción que ha hecho la red neuronal. Este vector tiene una longitud temporal de cuarenta y cinco **minutos**.

Siguiendo el mismo ejemplo que en el vector de entrada, este vector resultado contiene los valores de la moneda desde t0 hasta t44, lo que genera el valor de la moneda en todos los minutos desde las 18:00 hasta las 18:45. Dicho ejemplo se presenta en la figura 3.4.

Hora	18:00	18:01	18:02	18:03		18:42	18:43	18:44	
Valor	1,23345	1,23389	1,23402	1,23394	...	...	1,20210	1,20345	1,20439
Array index	0	1	2	3		42	43	44	

Figura 3.4: Ejemplo del vector resultado con un momento temporal concreto

Este vector es el resultado de los cálculos que ha realizado la red neuronal. Es importante recordar que los datos vienen ordenados, por lo que una vez obtenido el vector resultado se pueden visualizarse directamente.

### Early Stopping

El Early Stopping es una herramienta que proporciona Tensorflow para detener el entrenamiento de la red neuronal en el caso de que ésta deje de aprender. Esta herramienta se utiliza para evitar el sobreajuste durante el entrenamiento.

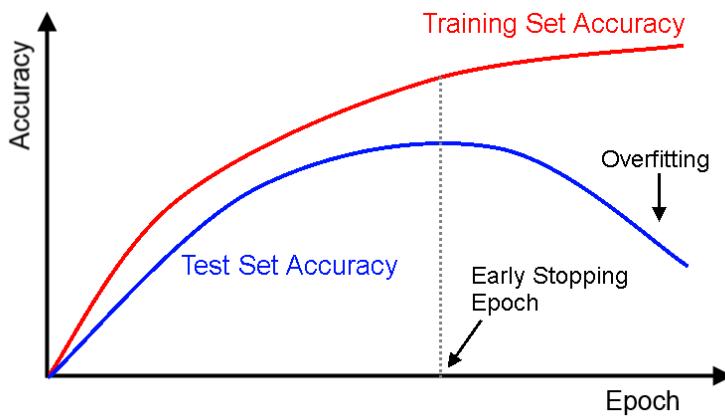


Figura 3.5: Gráfico de la precisión en el aprendizaje indicando la época óptima para parar el aprendizaje

Como se puede apreciar en la figura 3.5, la precisión empieza a bajar pasado un número de épocas debido a que está sobreajustándose a los valores que ya conoce. La línea roja representa la precisión de las predicciones de los datos que está utilizando para entrenar, mientras que la línea azul representa la precisión en las predicciones de unos datos que la red no está usando para entrenar, por lo que no conoce.

El early stopping tiene varios parámetros de configuración. Podemos destacar el **valor delta**, el cual indica el valor que debe **mejorar** la métrica de error para considerar que sigue aprendiendo y la **paciencia**, el cual indica el número de épocas que permite a la red seguir entrenando sin mejorar su delta.

### 3.4.2. Red Neuronal Recurrente

Las redes neuronales recurrentes proporcionan muy buenos resultados en problemas de **series temporales**. Como se ha podido observar en el capítulo 2, la mejor capa que podemos utilizar en este tipo de redes es la capa **LSTM**.

El algoritmo que se ha construido es el siguiente:

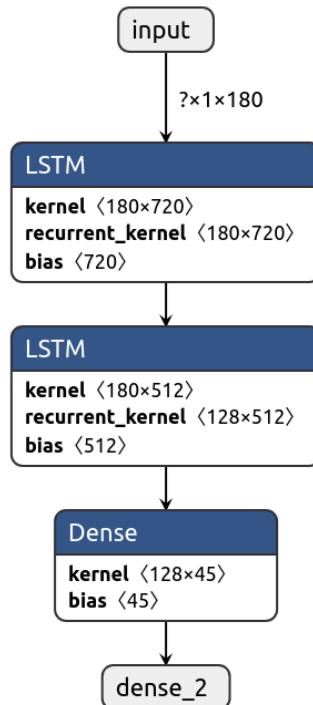


Figura 3.6: Arquitectura de red neuronal recurrente utilizada en el proyecto

En este esquema se puede observar cómo se han implementado dos capas LSTM, la primera de 180 neuronas y la segunda de 128 neuronas además de una capa Dense de 45 neuronas que genera la salida de la red neuronal. Las capas LSTM tienen asociadas unas capas Dropout que no se ven en el diagrama. Este dropout elimina el 10 % de las neuronas, escogidas aleatoriamente.

#### Funcionamiento

Para cada vector que pertenece a la lista de vectores de entrada el proceso que se le realiza es el siguiente:

1. El vector entrada entra por la capa de input teniendo una longitud de 180 neuronas.
2. El vector entra en la primera capa LSTM, donde se procesa secuencialmente. El resultado de esta capa es un vector de dimensión 128.
3. El vector que ha resultado del paso 2 entra por la segunda capa LSTM, procesándose secuencialmente. El resultado de esta capa es una lista de 128 vectores de dimensión 1, es decir, un vector por neurona.
4. La capa dense recibe todos los vectores y los interconecta entre sí con 45 neuronas que van a generar el vector resultado.
5. Desde la capa densa sale un vector de longitud 45 el cual es el vector salida.

## Entrenamiento

La red neuronal recurrente entrena durante un periodo de ocho días de anterioridad respecto al momento  $t_0$ . Los datos obtenidos se transforman en secuencias de 3 horas desde las 00:00 del octavo día anterior, sin contar los días en los que el mercado cierra.

Se ha seleccionado el periodo de 8 días debido a que es un periodo lo suficientemente largo (se crean alrededor de 10000 secuencias) para que la red pueda encontrar patrones sin sobreajustarse. Hay que tener en cuenta que los valores oscilan en magnitudes muy pequeñas. Debido a esto hay que normalizar los datos entre 0 y 1, consiguiendo así que la red pueda realizar cálculos con números de mayor magnitud.

La red tiene un Dropout muy bajo debido a que los patrones que encuentra no son patrones muy generalizados. Por ello, al poner una tasa de Dropout mayor la red es incapaz de encontrar un patrón mínimamente válido.

La red es sometida a 150 épocas de entrenamiento. Durante estas 150 épocas, la red va a ver los datos con el fin de encontrar patrones nuevos cada vez que los revisa, o reforzar el conocimiento que tiene sobre un patrón. La red tiene configurado un EarlyStopping de 10 épocas con un delta de 0.00001 de mejora en su métrica de evaluación. Por norma general, el EarlyStopping salta sobre la época 110.

## Benchmark

La red ha sido sometida a muchos benchmarks para evaluar su rendimiento, tanto en tiempo de procesamiento como en calidad de la predicción.

En la figura 3.7 se puede observar un ejemplo de una salida típica de la red usando el par EUR/USD.

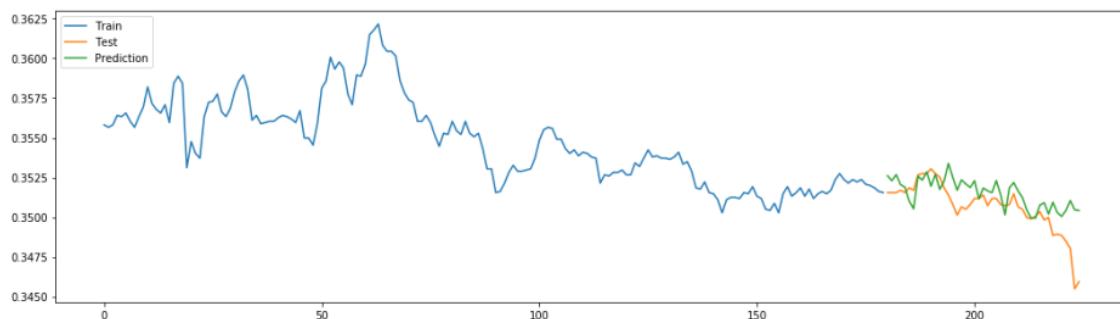


Figura 3.7: Resultado típico de la predicción de la red neuronal recurrente

La línea azul representada en la leyenda como *train*, representa los 180 valores que ha utilizado la red para predecir la línea verde, representada en la leyenda como *prediction*. La línea naranja representa los valores reales que ha tomado el par en el mismo periodo que ha predicho la red.

Lo primero que se observa es que no tiene una precisión total, y así es. Como ya hemos explicado en la sección de fundamentos teóricos, esta red se está basando exclusivamente en el histórico del valor, sin recibir ninguna información extra, por lo que una bajada brusca que se deba a un factor externo le resulta muy difícil predecirla.

No obstante, la red está prediciendo fácilmente la tendencia que está siguiendo el valor de la moneda. Además, en la línea naranja, se está mostrando el valor al que se cierra el par en ese minuto. Ésto no quiere decir, que el valor que la red ha predicho no haya ocurrido, sino que durante ese minuto ese valor ha oscilado entre varios. Suponiendo que cambia su valor cada segundo, habrá oscilado 60 veces entre dos puntos de la gráfica. Existe una probabilidad muy alta de que el valor que la red ha predicho, aunque no haya sido el valor final, sea un valor que el par ha tomado durante ese minuto.

### 3.4.3. Red Neuronal Convolucional

Las redes neuronales convolucionales originalmente no están diseñadas para ser utilizadas en problemas de series temporales, no obstante, éstas obtienen patrones basándose en los detalles de los datos. Si extrapolamos el problema, haciéndole ver a la red que lo que recibe es un **mapa** de datos en vez de una secuencia, la red puede funcionar correctamente y predecir resultados. Este mapa tiene la **misma dimensión** que la secuencia, pero la red convolucional no va a darle importancia al orden en el que recibe los datos, tal y como hace la red recurrente. Al necesitar que los datos estén ordenados obligatoriamente en la predicción, le indicaremos a la red que trate todos los mapas que va a procesar separándolos por conjuntos independientes. De esta forma la red sabe que cada secuencia, las cuales ahora son un mapa, deben ser tratadas y procesadas de forma individual. De esta manera la red va a ir aprendiendo y entendiendo la información que proporciona cada uno, adaptando sus pesos para conseguir una solución para todos los mapas que va a recibir.

El algoritmo que se ha construido es el siguiente:

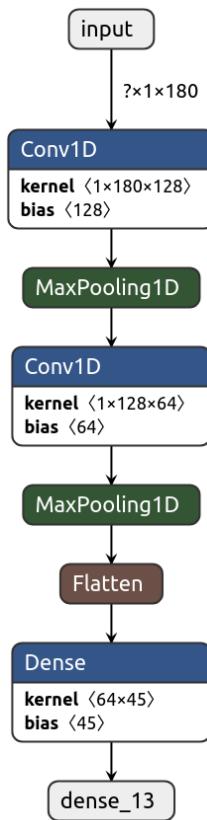


Figura 3.8: Arquitectura de red neuronal convolucional usada en el proyecto.

En el anterior esquema se puede apreciar cómo se han implementado dos capas Conv1D, la primera con 128 neuronas, y la segunda de 64 neuronas, encontrándose ambas concatenadas con capas MaxPooling1D. A ésto se le añade una capa Flatten y una capa Dense de 45 neuronas, lo que genera la salida de la red neuronal.

## Funcionamiento

Para cada vector que aparece en la lista de vectores de entrada, el proceso que se realiza en la red es el siguiente:

1. El vector entrada entra por la capa de input, teniendo una longitud de 180.
2. El vector entra en la primera capa Conv1D, lo que hace que se reduzca la dimensionalidad de esta capa a 64. El resultado de esta capa es un vector de dimensión 64.
3. El vector que ha resultado del paso 2 entra por la capa MaxPooling1D, reduciendo su dimensionalidad a la mitad.
4. Tras el paso 3, los datos vuelven a pasar por una capa Conv1D, siendo el resultado de esta capa un vector de dimensión 32.
5. El vector resultante pasa por la capa MaxPooling1D, reduciendo su dimensión a la mitad.
6. El vector entra en una capa Flatten, donde los datos se concatenan en una lista.
7. El vector resultante del paso 7 entra en una capa Dense, en la cual se van a interconectar entre sí las neuronas de la capa Flatten con 45 neuronas, generando así el vector resultante.
8. De la capa Dense sale un vector de longitud 45. Este es el vector salida.

## Entrenamiento

La red neuronal convolucional entrena durante un periodo de diez días de anterioridad respecto al momento  $t_0$ . Estos datos se transforman en secuencias de 3 horas desde las 00:00 del décimo día anterior excluyendo los días en los que los mercados cierran.

Se ha escogido la cifra de 10 días frente a los 8 días de la red recurrente porque esta red necesita un volumen mayor de datos para aprender, pero, al igual que en la red recurrente, si el periodo es muy largo la red se sobreajustará basándose en los datos que conoce.

La red es sometida a 150 épocas de entrenamiento, al igual que las redes recurrentes. Tiene configurado un EarlyStopping de 10 épocas con un delta de 0.00001 de mejora en su métrica de evaluación. Por norma general, el EarlyStopping salta, aproximadamente, en la época 64.

## Benchmark

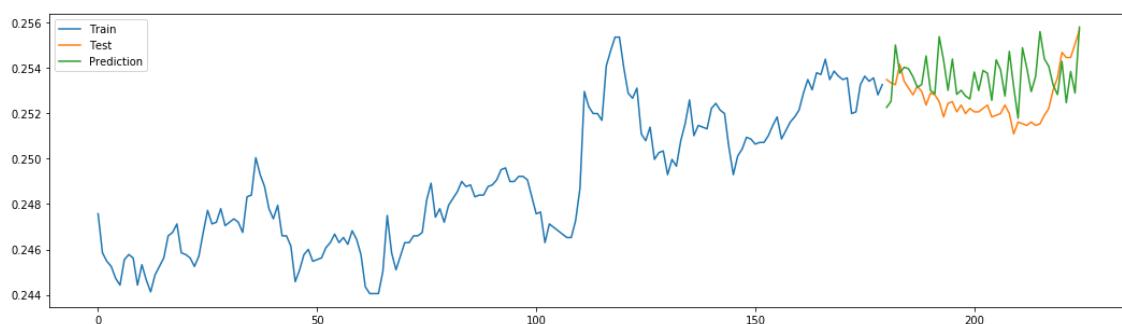


Figura 3.9: Resultado típico de la predicción de la red neuronal convolucional

Al igual que sucede con la red recurrente, la red convolucional es sometida a muchos benchmarks para evaluar su rendimiento.

En la figura 3.9 se presenta un ejemplo de una salida típica de la red usando el par EUR/USD, con unos datos similares al benchmark de la red recurrente.

La línea azul representada en la leyenda como *train* representa los 180 valores que ha utilizado la red para predecir la línea verde, representada en la leyenda como *prediction*. La línea naranja representa los valores reales que ha tomado el par en el mismo periodo que ha predicho la red.

A diferencia de la red recurrente, las predicciones de esta red parecen más volátiles o erráticas. Esto es debido a la magnitud de los datos, la cual es muy pequeña, aunque los datos se normalicen, resulta difícil seguir la predicción con una precisión total. Esta red también es capaz de seguir la tendencia de los valores, pero sus predicciones son menos fiables que las que proporciona la red recurrente.

#### 3.4.4. Tecnologías utilizadas

Para la implementación de ambas redes neuronales se ha utilizado TensorFlow que se explicará en profundidad en el capítulo 4.

### 3.5– Monitorización

El módulo de monitorización se encarga de la visualización y representación de los datos generados por el sistema, permitiendo al usuario interpretarlos a través de una interfaz de usuario web. Los valores generados por el sistema son las predicciones de las divisas hechas por las redes neuronales. Por cada divisa el sistema proporciona dos predicciones, cada una generada por una red neuronal diferente.

Lo primero que muestra la interfaz es el valor actual de las divisas y su cambio respecto al valor anterior, mostrándose de color verde, rojo o blanco, dependiendo de si el valor es mayor, menor o igual al valor anterior, respectivamente. Todo ello permite al usuario de forma rápida y sencilla conocer la situación actual de las divisas. En la figura 3.10 se puede observar un ejemplo de la misma.



Figura 3.10: Valores actuales de las divisas

A continuación, se encuentra la gráfica principal, en la cual se muestran las predicciones de los próximos cuarenta y cinco minutos, el valor actual de la divisa y los últimos treinta minutos del histórico de valores, tal y como se puede observar en la figura 3.11. Mediante un menú desplegable podemos seleccionar la divisa que queremos visualizar y, a través de las pestañas se puede alternar entre las dos redes neuronales.



Figura 3.11: Representación de los resultados

Finalmente, el módulo presenta una gráfica con el histórico completo de la divisa seleccionada. En la figura 3.12 se puede observar un ejemplo con el histórico del BTC/USD.



Figura 3.12: Histórico del BTC/USD

En esta gráfica podemos elegir mediante el menú desplegable de la izquierda el tipo el gráfico que queremos representar y, mediante el menú desplegable de la derecha podemos agregar indicadores al gráfico. En la figura 3.13 se puede observar un ejemplo con el histórico del BTC/USD junto a los indicadores MA y EMA.



Figura 3.13: Histórico del BTC/USD con indicadores

Los indicadores de los que dispone el sistema son:

- Acumulación distribución
- Bandas de Bollinger
- Media móvil
- Media móvil exponencial
- Índice de canales de productos básicos
- Tasa de cambio
- Puntos pivote
- Oscilador estocástico
- Momentum

### 3.5.1. Tecnologías utilizadas

Para la visualización y representación de los datos se ha escogido el framework Dash.

#### 3.5.1.1. Dash

Dash es un framework de Python destinado a la creación de aplicaciones web de código abierto, desarrollado por Plotly. Dash está implementado sobre Flask, Plotly.js y React.js, especializándose en la creación de aplicaciones reactivas de visualización de datos con interfaces de usuario altamente personalizables en Python, abstrayendo todas las tecnologías y protocolos necesarios para construir una aplicación interactiva basada en la web. El código de la aplicación Dash es declarativo y reactivo, lo que facilita la creación de aplicaciones complejas que contienen muchos elementos interactivos, siendo cada elemento estético de la aplicación completamente personalizable (tamaño, color, posición, fuente, ...) Las aplicaciones de Dash son desplegadas mediante Flask y se comunican mediante JSON sobre HTTP. La interfaz de Dash presenta componentes utilizando React.js, la biblioteca de interfaz de usuario de Javascript escrita y mantenida por Facebook. Los componentes de Dash son clases de Python que codifican los valores y propiedades de los componentes de React.js, siendo éstos serializados en JSON, aunque también existen clases Python para todos los componentes HTML y sus propiedades, por lo que mediante código Python se pueden declarar y manipular todos los elementos de una interfaz y sus interacciones entre ellos.



# CAPÍTULO 4

---

## Arquitectura del sistema

---

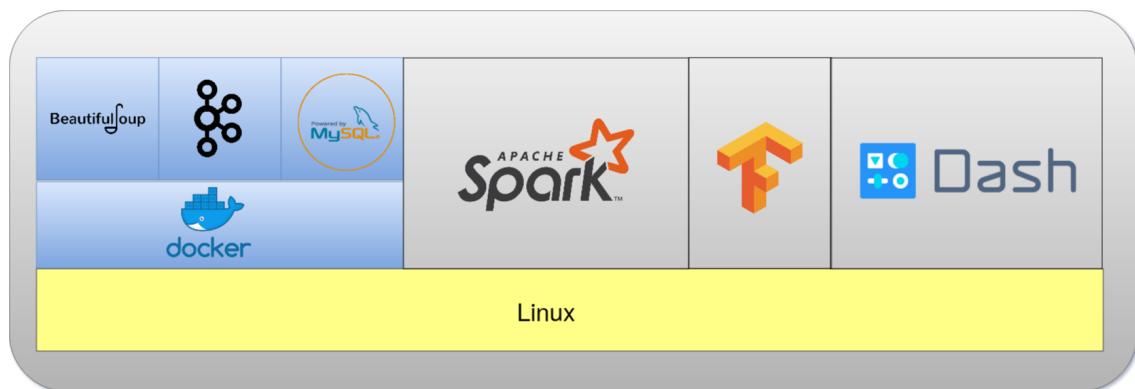


Figura 4.1: Diagrama de arquitectura del proyecto

Como se puede observar en la figura 4.1, el sistema desarrollado en este proyecto se asienta sobre un sistema Linux, el cual nos permite la ejecución de los contenedores proporcionados por Docker, que han sido cruciales para la implementación de los **productores**, la base de datos **MySQL** y Apache **Kafka**.

Sin embargo, elementos como Apache Spark y Tensorflow se asientan directamente sobre Linux. Ésto es debido a que ambos componentes necesitan el número máximo de recursos posibles, y aunque Docker no consume un número elevado de recursos, para el correcto funcionamiento de estos componentes hay que disponer de todos los recursos disponibles.

Esta arquitectura es totalmente modular y se ha diseñado con el propósito de que todos sus componentes puedan ser separados en diferentes máquinas, e incluso en cluster, para poder suministrar los recursos que se necesiten en tiempo de ejecución. Está inspirada en el modelo de microservicios, aunque no se cumplen estrictamente todos los requisitos.

En este capítulo se explicarán con detalle los componentes más importantes del sistema: **Apache Spark**, **Apache Kafka**, y **Tensorflow**.

## 4.1– Apache Spark



Figura 4.2: Logo de Apache Spark

Apache Spark es un sistema de computación en clúster orientado a la velocidad y de propósito general escrito en **Scala**. Proporciona una API de alto nivel en **Java**, **Scala**, **Python** y **R** y un motor optimizado que admite grafos de ejecución generales. Incluye un amplio conjunto de herramientas de alto nivel que incluyen **Spark SQL** para consultas SQL y procesamiento de datos estructurado, **MLlib** para aprendizaje automático, **GraphX** para procesamiento de gráficos y **Spark Streaming**. [11]

Las características principales de Spark son:

- **Alta velocidad.** Spark ofrece gran velocidad en el tratamiento de grandes volúmenes de datos, siendo cien veces más rápido si los datos están en memoria, y diez veces más rápido si los datos están en disco.
- Más rápido que MapReduce. Cuando el Big Data empieza a surgir nace MapReduce, un algoritmo que permite tratar gran volumen de datos de forma rápida. Con el paso del tiempo, este algoritmo empieza a dejar de ser eficiente por sí solo, debido a que el volumen de datos crece muy rápido. Spark tiene implementado su propio MapReduce y es hasta cien veces más rápido que el algoritmo original.
- Dinámico, gracias al paralelismo que ofrece spark y a sus ochenta operadores de alto nivel disponibles.
- **Computación directamente en memoria.** El incremento de velocidad es posible gracias al procesamiento in-memory que está implementado.
- Reusabilidad. Spark permite lanzar querys ad-hoc para procesamiento en streaming, y además está preparado para batch-processing, lo que permite que se pueda reutilizar código fácilmente.
- **Tolerancia a fallos.** El core de Spark ofrece una característica llamada abstraction-RDD. Ésta permite que una orden enviada a cualquier worker que haya en el cluster pueda ser reenviada a otro worker sin coste alguno, en caso de que algún nodo falle.
- **Integración total con Hadoop.** A diferencia de otros frameworks, Spark está totalmente integrado con Hadoop, pudiendo así controlar toda la potencia de Hadoop de forma mucho más simple y sin posibles errores.
- **Procesamiento en tiempo real.** Hadoop no permite procesamiento en tiempo real, lo que es un gran problema a la hora de tratar con una gran cantidad de datos en tiempo real, debido a que Hadoop solo puede hacer operaciones con datos que ya están almacenados. Para solucionarlo, Spark sí implementa operaciones en tiempo real, que posteriormente pueden ser almacenadas en Hadoop.

- **Lazy Evaluation.** Esta es una característica muy interesante de Spark, que permite planificar muchas operaciones y transformaciones que no se ejecutan línea a línea, sino que Spark las agrupa para reutilizar procesamiento. La veremos en detalle más adelante.
- Soporte a diferentes lenguajes, como Scala, Python, Java y R.

#### 4.1.1. Fundamentos de Spark

Spark fué diseñado para trabajar en cluster, debido a características novedosas como el planificador de procesos. No obstante, está adaptado para poder funcionar en una sola máquina sin perder eficiencia.

Además, Spark cuenta con una serie de elementos y características que son cruciales para entender su funcionamiento.

##### 4.1.1.1. Elementos

Los elementos que componen a Spark son:

- **Jobs.** Su traducción directa al castellano es trabajo, y hace referencia a cualquier operación que tenga que hacer. Unos ejemplos pueden ser leer una entrada de datos de HDFS, o hacer una agrupación de datos.
- **Stages.** Los jobs están divididos en Stages, los cuales están clasificados en dos tipos: Map y Reduce. Estos stages se calculan en base a los límites computacionales de cada máquina, para que todas las operaciones se puedan realizar sin perder datos. Suelen crearse muchos stages para un job.
- **Task.** Cada Stage, está dividida en Task. Existe una task por cada partition de datos, y se ejecutan en cada partición de datos de un mismo executor.
- **Partition.** Es una separación lógica de los datos. Fuera del ámbito de Spark, una Partition es lo mismo que un Chunk o un Split de datos.
- **Executor.** Es el proceso del sistema que se encarga de ejecutar las tareas. En una misma máquina puede haber varios ejecutores.
- **Driver.** Es el proceso del sistema responsable de ejecutar el Job. Este proceso hace de cerebro en Spark, es el encargado de crear todas las Stages y Task, y enviarlas a los ejecutores que tiene Disponible. Solo hay un proceso Driver por cada sesión de Spark.
- **Master.** Es la máquina que está ejecutando el proceso driver. Como Spark puede ejecutarse en una sola máquina, esta máquina puede ser Master y Worker al mismo tiempo.
- **Worker.** Es una máquina que tiene procesos solamente del tipo Executor.

#### 4.1.1.2. Características

Una vez vistos los elementos de Spark, se debe hablar de las características sobre las cuales se asienta Spark. Hay que destacar dos, el **planificador** y su **modo de ejecución**, aunque también haremos mención de una característica algo menos importante, pero que siempre está presente en el desarrollo con Spark, la **sesión**.

- **Planificador de Tareas.**

Como se ha mencionado anteriormente, Spark se caracteriza por su potencia y velocidad. Esta velocidad surge de la idea de crear una especie de cola de tareas antes de la ejecución de todas ellas, con la finalidad de ver si en tiempo real se puede optimizar alguna operación.

Se crea entonces el planificador DAG, que consiste en un Grafico Acíclico Directo que se encarga de enumerar, agrupar y priorizar todas las tareas que se ejecutan en Spark.

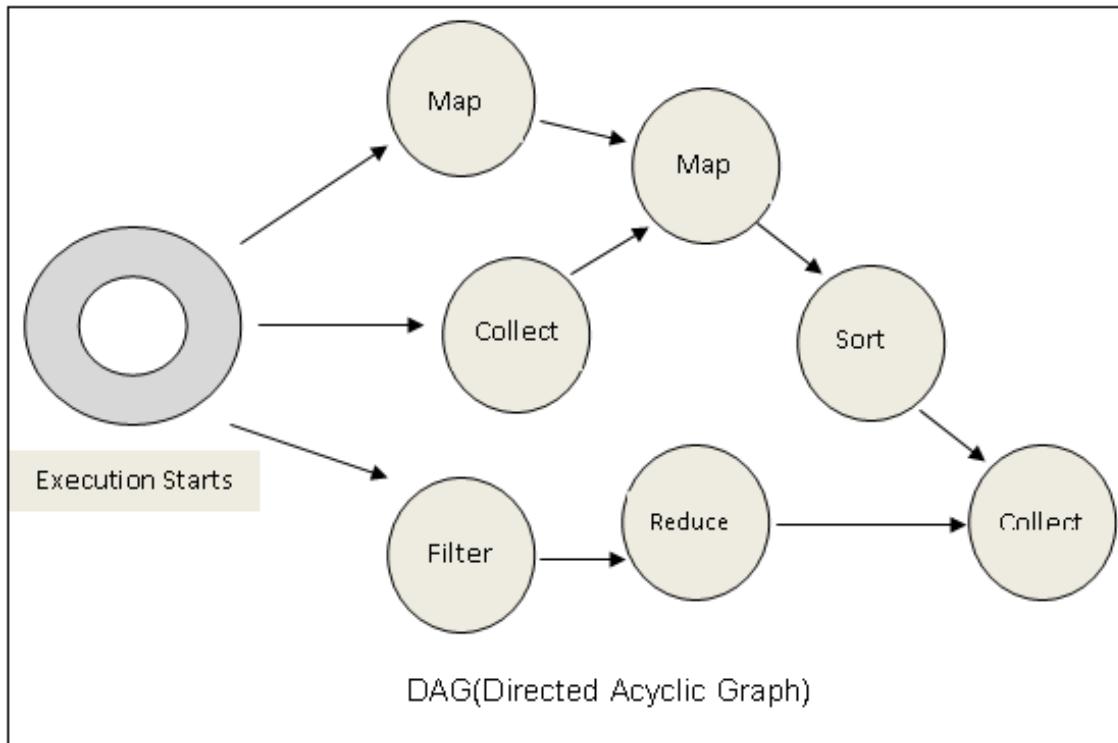


Figura 4.3: Ejemplo de un DAG de Spark

En la figura 4.3 se muestra un ejemplo de cómo Spark realiza un Job. En este caso, va a hacer una operación llamada Collect, la cual necesita que se realicen varios Maps and Reduces previos. Al tener la operación planificada, puede hacer las tareas de forma paralela y continua, en vez de realizarlas todas una detrás de otra, ralentizando el proceso.

- **Lazy Evaluation.**

La Lazy Evaluation, o evaluación perezosa en castellano, es una estrategia de evaluación de funciones que retrasa el cálculo de una operación hasta que su valor sea necesario por otra operación. Ésto evita repetir la evaluación en caso de ser necesaria en posteriores ocasiones. Dicha estrategia supone una enorme reducción en el tiempo de ejecución de ciertas funciones de forma exponencial, comparado con la evaluación de funciones tradicional.

- **Spark Session.**

El core de Spark está siempre funcionando en una máquina donde está instalado, pero no se puede acceder a él directamente. Para realizar operaciones en Spark se ha diseñado un sistema que funciona por sesiones en el cual obligatorio crear una sesión para conectarse al core de Spark. En una misma máquina pueden crearse diferentes sesiones de Spark para diferentes cometidos.

Por ejemplo, si una máquina A crea una sesión de Spark y ejecuta un código para que realice ciertas transformaciones en los datos y, a su vez, otra máquina diferente se conecta a la sesión que ha creado la máquina A, esta nueva máquina puede acceder a los datos ya procesados por ella. Todo ello es muy interesante a la hora de trabajar con Data Lakes.

#### 4.1.2. Cómo funciona Spark

Una vez explicados los elementos y características fundamentales de Spark, se pasa a exponer cómo funciona. El funcionamiento interno de Spark es muy complejo y extenso, por lo que veremos su funcionamiento en alto nivel.

Como se ha observado anteriormente, en los elementos, Spark separa sus objetivos en unidades más pequeñas para poder repartirlas por los workers y poder ejecutar tareas de forma paralela.

Su funcionamiento está basado en dos tipos de operaciones principales: **acciones** y **transformaciones**.

Una **transformación** es una operación en la cual se realizan cálculos y modificaciones sobre los datos originales. Por ejemplo, crear una nueva columna, realizar una suma de los valores de varias columnas, modificar una columna que contiene strings, etc. Estas transformaciones no se hacen directamente gracias a la Lazy Evaluation y éstas se pueden dividir en otros dos grupos, siendo éstos:

- **Narrow transformation.** Son aquellas en las que los datos que se necesitan tratar están en la misma partition y no es necesario realizar una mezcla de dichos datos para obtenerlos todos. Por ejemplo, las funciones filter() o map().
- **Wide transformation.** Son aquellas que, por el contrario, los datos que necesitan tratar están en partitions diferentes por la lógica de la aplicación. Es necesario mezclar dichas partitions para agrupar los datos necesarios. Por ejemplo, groupByKey() o reduceByKey().

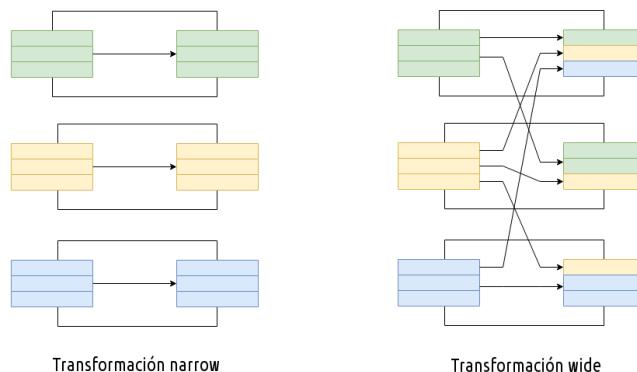


Figura 4.4: Tipos de transformaciones en Spark

Una **acción** hace referencia a las acciones propiamente dichas que se pueden realizar con un Dataset, como, por ejemplo, mostrar algo por pantalla, o guardar los datos en disco.

Spark en su planificación recoge todas las transformaciones que se realizan entre acciones y prepara como las va a ejecutar. Aunque la ejecución del DAG es paralela, sus transformaciones se aplican siempre de forma ordenada, para no interferir en el resultado final.

Cuando Spark tiene un DAG preparado para su ejecución, consulta los recursos que tiene disponibles, los cuales se encuentran en los diferentes workers que tiene asociados. Pese a que Spark permite que cada worker tenga una configuración hardware diferente, se recomienda que todos tengan los mismos recursos. Cuando nos referimos a recursos, principalmente nos referimos al número de núcleos y a la memoria RAM disponible.

Las ejecuciones se realizan por partition, en la que están los datos con los que estamos tratando. El usuario puede definir el número de partition que quiere o la clave para almacenar los datos, pero no el contenido de cada partition. Cuando Spark tiene que ejecutar transformaciones de tipo Wide, se genera la memoria de tipo **Shuffle**. En la figura 4.5 se puede observar un ejemplo de flujo en el que en cada una de las stages se ejecutan operaciones de diferentes tipos, tratándose los datos de forma individual para después moverse entre particiones para dar un resultado.

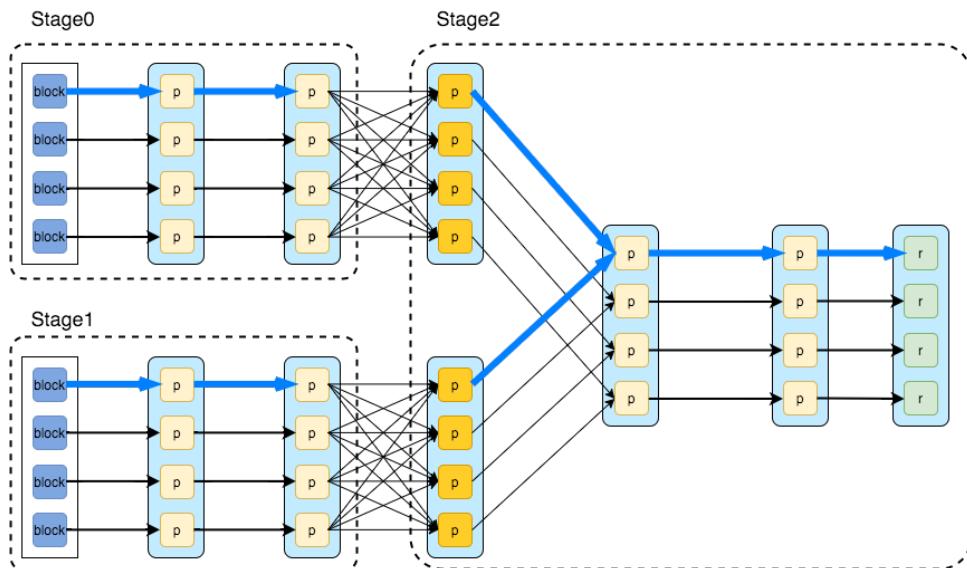


Figura 4.5: Operaciones en Spark

#### 4.1.3. Módulos de Spark

Apache Spark está dividido en diferentes módulos que separan la funcionalidad base de la funcionalidad extra. Toda la funcionalidad base está en el módulo **Core**. Está totalmente escrita en Scala y en ella se realizan todos los procesos que planifica Spark.

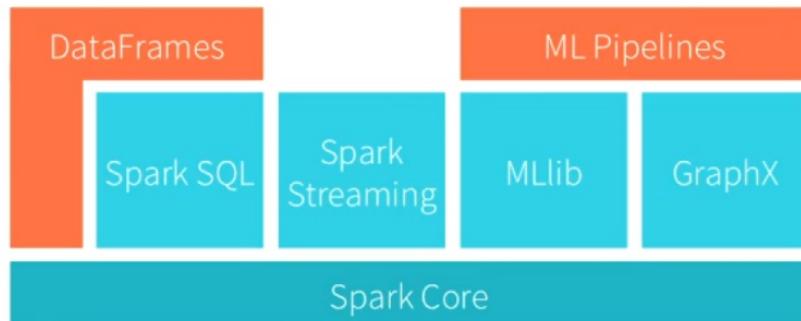


Figura 4.6: Módulos de Spark

Como se puede ver en la figura 4.6, el resto de módulos están divididos en **Spark SQL**, para consultas y transformaciones de datos; **Spark Streaming**, para el procesamiento de datos en tiempo real; **Spark MLlib**, una librería con algoritmos de aprendizaje automático; y **GraphX**, para manejar grafos. Además de los mencionados, de cada uno de estos cuatro módulos dependen diferentes librerías más específicas.

A continuación, se describirán los paquetes utilizados en este proyecto.

##### 4.1.3.1. Spark SQL

Spark SQL es un módulo de Spark para el procesamiento de datos estructurados. A diferencia de la API básica de **Spark RDD**, las interfaces proporcionadas por Spark SQL proporcionan a Spark más información sobre la estructura de los datos y el cálculo que se realiza. Internamente, Spark SQL utiliza esta información adicional para realizar optimizaciones adicionales. Existen varias formas de interactuar con Spark SQL, incluyendo SQL y la **API Dataset**. Al calcular un resultado, se utiliza el mismo motor de ejecución, independientemente de la API que se esté utilizando para expresar el cálculo. Esta unificación significa que los desarrolladores pueden cambiar fácilmente entre diferentes API en función de lo que proporcionan, la forma más natural de expresar una transformación determinada.

Un uso de Spark SQL es ejecutar consultas SQL, aunque también se puede utilizar para leer datos de una instalación existente de **Hive**. Al ejecutar SQL desde otro lenguaje de programación, los resultados se mostrarán como un **DataSet/DataFrame**, y, además, puede interactuar con la interfaz SQL utilizando la línea de comandos o sobre **JDBC/ODBC**.

##### 4.1.3.2. DataFrame

Un DataFrame es un conjunto de datos organizado en columnas con nombre. Conceptualmente es equivalente a una tabla en una base de datos relacional o un marco de datos en R/Python, pero con mejores optimizaciones. Los DataFrames se pueden construir a partir de una amplia gama de fuentes, tales como: archivos de datos estructurados, tablas en Hive, bases de datos externas o RDD existentes. El paquete DataFrame pertenece al módulo de Spark SQL. La API de DataFrame está disponible en Scala, Java, Python y R. [12]

#### 4.1.3.3. Spark Structured Streaming

Spark Structured Streaming es un motor de procesamiento de flujos escalable y tolerante a fallos construido en el motor Spark SQL, que procesa datos estructurados por un esquema. Esta librería pertenece al módulo Spark Streaming.

Spark Structured Streaming lo controla el motor Spark SQL, el cual se encargará de ejecutarlo de forma incremental y continua, y actualizará el resultado final a medida que continúen llegando los datos de transmisión.

Esta librería integra el paquete DataFrame, por lo que se pueden expresar agregaciones de transmisión, ventanas de eventos, combinaciones de flujo por lote, etc.

Finalmente, el sistema ofrece garantías de tolerancia a fallos de una sola vez de extremo a extremo a través de puntos de control y registros de escritura anticipada. En resumen, Structured Streaming ofrece un procesamiento rápido, escalable, tolerante a fallos y de extremo a extremo sin que el usuario tenga que razonar acerca de la transmisión. [13]

## 4.2– TensorFlow



Figura 4.7: Logo de Tensorflow

TensorFlow es una librería de código abierto desarrollada por Google para el desarrollo de redes neuronales en el campo del aprendizaje profundo.

La idea en la que se basa TensorFlow es la del flujo de información por un grafo en el que los nodos representan operaciones y las aristas son los datos que viajan entre los nodos, a esto último lo llamamos tensor.

### 4.2.1. Grafos

Un grafo de TensorFlow se puede definir como un conjunto de operaciones interconectadas entre las que fluye información que se puede representar como un grafo. Un grafo está formado por dos partes:

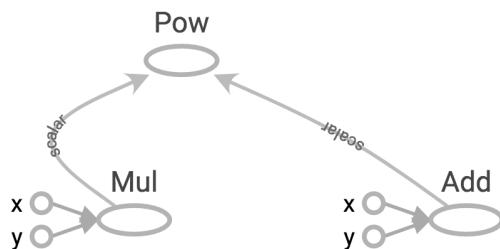


Figura 4.8: Esquema de una operación representada como un grafo

- **Operaciones:** Representadas por nodos son operaciones matemáticas que reciben y producen tensores.
- **Tensores:** Son las aristas del grafo. Representan la información que fluye por el grafo entre funciones.

Por ejemplo, una multiplicación de escalares se representaría como un nodo con dos aristas de entrada y una arista de salida. Una de las ventajas más importantes de esta forma de ejecución de operaciones es identificar aquellas que se pueden ejecutar simultáneamente. En la figura 4.8 podemos ver una representación de dos operaciones como grafos.

#### 4.2.2. Tensor

El tensor es el objeto principal de TensorFlow. Un tensor es un conjunto de valores ordenados en un array n-dimensional. Un tensor se caracteriza por el número de dimensiones o rank y el tamaño de cada dimensión o shape.

Un programa en TensorFlow primero construye el grafo con tensores detallando los cálculos a realizar y posteriormente ejecuta el grafo para extraer los resultados deseados. La figura 4.9 ilustra el concepto de tensor.

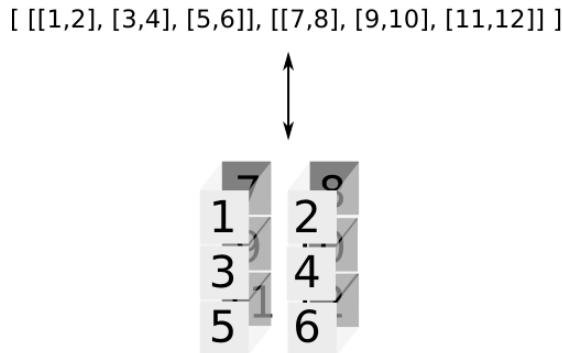


Figura 4.9: Ejemplo de tensor

#### 4.2.3. Keras

Tensorflow ha publicado recientemente su versión 2.0, la cual incluye multitud de cambios destacables, tanto en términos de rendimiento como de funcionalidad.

Una de las mejoras más importantes ha sido la de la integración total de la librería **Keras** en su núcleo. Esta librería, escrita por Francois Chollet, facilita mucho las labores de desarrollo de las redes neuronales, debido a que incluyen capas completas de las redes totalmente funcionales. Además, el uso de estas capas es muy sencillo para el desarrollador, ya que basta con importar la capa que deseas utilizar y pasarle por parámetro las características que necesitas en la misma.

Esta librería abstrae al usuario del uso de Grafos y Tensores, debido a que todas las capas ya tienen implementada la lógica propia de capa, junto a los tensores necesarios. Además, todas las capas están optimizadas para obtener el máximo rendimiento en ejecución.

Ésto permite que el desarrollo de la red neuronal del presente proyecto sea una labor más ágil, pudiendo probar diferentes arquitecturas posibles en un tiempo reducido.

## 4.3– Apache Kafka



Figura 4.10: Logo de Apache Kafka

Apache Kafka es una plataforma distribuida para la manipulación en tiempo de real fuentes de datos de código abierto y desarrollada por la **Apache Software Fundation** en los lenguajes **Java** y **Scala**. Sus principales características son la baja latencia y la escalabilidad masiva.

Apache Kafka funciona bajo el patrón **productor-consumidor** agrupando la información en temas, por lo que podemos ver a kafka como un gestor de cola de mensajes altamente escalable.

### 4.3.1. Broker

Una de las principales características de Apache Kafka es su carácter altamente escalable en una arquitectura de cluster. A los nodos de cluster se les denomina brokers, por tanto, un cluster de Apache Kafka está compuesto por brokers, cada uno con su identificador y particiones internas. Las particiones le indican al broker cuantos topics puede gestionar pudiéndose configurar distintos números de particiones en los brokers, permitiendo, que nodos con mayor capacidad gestionen más topics o que topics con un flujo de menor de información se agrupen en nodos aprovechando su capacidad.

### 4.3.2. Topic

Un topic es una categoría en la que se agrupan los mensajes en Kafka. Cada topic cuenta con un número de partición que indica el número máximo de veces que puede ser particionado un topic o, lo que es lo mismo, el número de brokers diferentes en un cluster que pueden contener al topic, también cuenta con *topic log* que registra toda la actividad generada sobre ese topic.

### 4.3.3. Productor

Los productores se encargan de publicar datos a los temas de su elección. El productor es el responsable de elegir la partición a la que enviar el dato dentro del tema. Esta elección se puede hacer mediante **round-robin** o con alguna función de partición semántica.

### 4.3.4. Consumidor

Los consumidores reciben los datos de los temas a los que están suscritos. Éstos se etiquetan a sí mismos con el nombre de un grupo de consumidores y cada registro publicado sobre un tema se entrega a una instancia del consumidor dentro de cada grupo de consumidores suscritos. Las instancias de los consumidores pueden estar en procesos separados o en máquinas separadas. Si todas las instancias del consumidor tienen el mismo grupo de consumidores, entonces, los registros se cargarán de manera efectiva sobre las instancias del consumidor. Si todas las instancias del consumidor tienen grupos de consumidores diferentes, entonces, cada registro se transmitirá a todos los procesos del consumidor.

#### 4.3.5. Zookeeper

Cuando se ejecuta una instancia o cluster de Apache Kafka, éste necesita de un servicio de Apache Zookeeper para su gestión interna. Este sistema proporciona a kafka un servicio centralizado para la gestión de la configuración, registro de cambios y servicios de descubrimiento, entre otros.

Zookeeper se comunica con todos los integrantes del ecosistema de Apache Kafka (brokers, consumidores y productores), compartiendo metadatos, tales como direcciones, estados, carga de trabajo y reasignaciones.

#### 4.3.6. Garantías

En alto nivel, Kafka proporciona las siguientes garantías:

- Los mensajes enviados por un productor a una partición de un tema en particular se adjuntará en el orden en el que se envía. Es decir, si un registro R1 es enviado por el mismo productor que un registro R2 y R1 se envió primero, entonces R1 aparecerá antes en el registro del tema.
- Una instancia de consumidor ve los datos en el orden en el que se almacenan en el tema.
- Para un tema replicado  $n$  veces, se toleran hasta  $n-1$  fallos del servidor sin perder ningún registro confirmado.

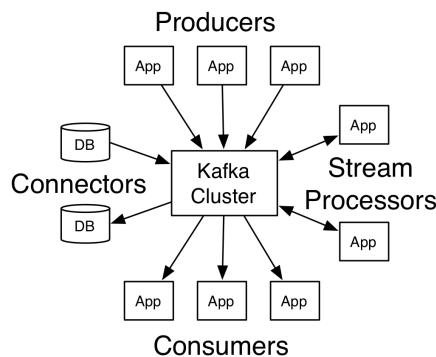


Figura 4.11: Arquitectura de Apache Kafka



# CAPÍTULO 5

---

## Análisis del sistema

---

A continuación, analizaremos las características del sistema, los procesos que debe cubrir, los requisitos a cumplir y los casos de uso que presenta.

### 5.1– Data-Driven Application

Este proyecto va a desarrollar una aplicación final con un concepto que no estamos acostumbrados aún a ver, el paradigma Data-Driven.

Las aplicaciones que conocemos actualmente tienen actores y procesos, donde el actor suele ser el usuario final y los procesos son las acciones que el actor puede realizar. Toda la aplicación se hace pensando siempre en el usuario final, siendo el elemento fundamental de ésta.

Para adaptar la filosofía Data-Driven y sus patrones de diseño necesitamos saber cómo usan los usuarios la aplicación, y en base a este conocimiento se pueden desarrollar nuevas funcionalidades cruciales y adaptar funcionalidad existente para facilitar el uso al usuario.

Sin embargo, en la aplicación que se va a desarrollar en este proyecto los actores no son los usuarios. Este escenario no es un escenario común, por lo que la aplicación debe ir adaptada al verdadero actor del sistema, **la red neuronal**. Dicha red es un algoritmo, por lo que no se puede comunicar con nosotros sobre algo que no sea lo que hemos desarrollado como salida.

Este un hándicap para el diseño de toda la aplicación, pero en especial para la capa de presentación, ya que vamos a tener que desarrollar nosotros mismos la mejor forma de utilizar y visualizar los datos de la red sin ninguna referencia.

Como solución a este problema, hemos basado el diseño de la capa de presentación en un panel de control más que en una aplicación tradicional. Este panel tiene interactividad con el usuario que consulta los datos que genera la red, pero no puede interactuar con la red en sí. La red trabajará de forma autónoma, al igual que lo harán los sistemas de almacenamiento y procesamiento de datos.

## 5.2– Procesos del sistema

En esta sección se muestran los procesos a cumplir por el sistema:

- Entrenamiento de una red neuronal
- Visualización de los resultados

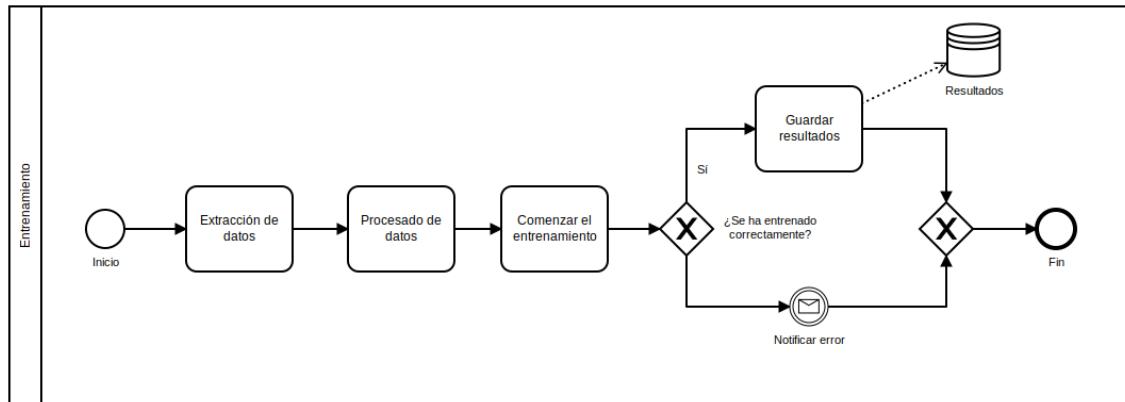


Figura 5.1: Entrenamiento de una red neuronal

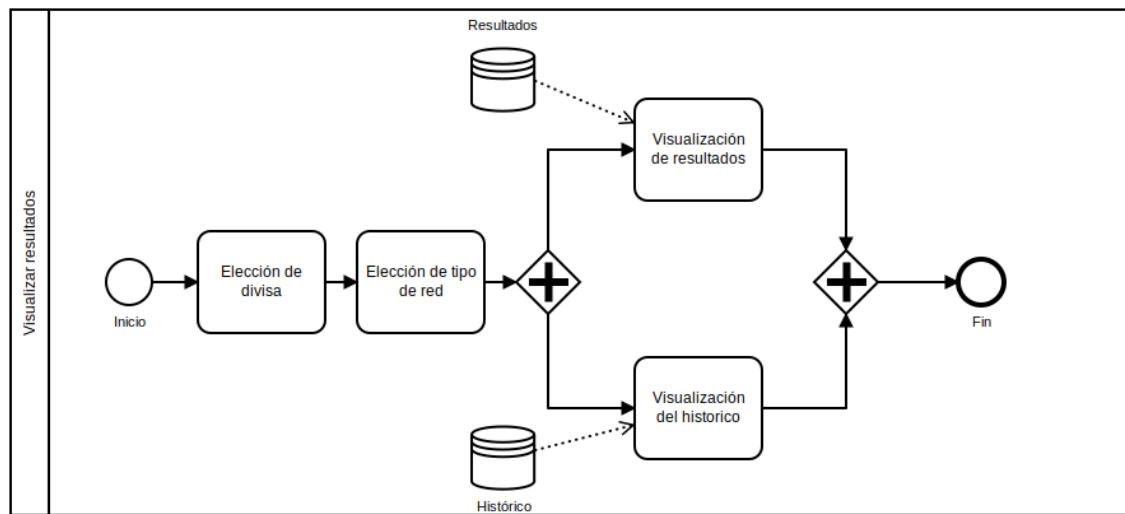


Figura 5.2: Visualización de resultados

## 5.3– Catálogo de requisitos

En esta sección detallamos los requisitos a cumplir por el sistema.

### 5.3.1. Requisitos generales

OBJ-001	Entrenamiento de una red convolucional
<b>Descripción</b>	El sistema deberá entrenar una red neuronal convolucional.
<b>Subobjetivos</b>	<ul style="list-style-type: none"> <li>• <b>OBJ-004</b> Entrenamiento con los datos EUR/USD: El sistema debe entrenar una red neuronal convolucional con los datos del EUR/USD.</li> <li>• <b>OBJ-005</b> Entrenamiento con los datos BTC/USD: El sistema debe entrenar una red neuronal convolucional con los datos del BTC/USD.</li> </ul>

Cuadro 5.1: Requisito general 001

OBJ-002	Entrenamiento de una red recurrente
<b>Descripción</b>	El sistema debe entrenar una red neuronal recurrente.
<b>Subobjetivos</b>	<ul style="list-style-type: none"> <li>• <b>OBJ-006</b> Entrenamiento con los datos EUR/USD: El sistema debe entrenar una red neuronal recurrente con los datos del EUR/USD.</li> <li>• <b>OBJ-007</b> Entrenamiento con los datos BTC/USD: El sistema debe entrenar una red neuronal recurrente con los datos del EUR/USD.</li> </ul>

Cuadro 5.2: Requisito general 002

OBJ-003	Visualización de resultados
<b>Descripción</b>	El sistema deberá mostrar los resultados de cada red.

Cuadro 5.3: Requisito general 003

### 5.3.2. Requisitos de información

RI-001	Divisas
<b>Descripción</b>	El sistema deberá almacenar los datos de las divisas.
<b>Datos específicos</b>	<ul style="list-style-type: none"> <li>• Fecha</li> <li>• Valor</li> <li>• Divisa</li> </ul>
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>• OBJ-001</li> <li>• OBJ-002</li> <li>• OBJ-003</li> </ul>

Cuadro 5.4: Requisito de información 001

RI-002	Resultados
<b>Descripción</b>	El sistema deberá almacenar los resultados de las redes neuronales .
<b>Datos específicos</b>	<ul style="list-style-type: none"> <li>• Fecha</li> <li>• Valor</li> <li>• Divisa</li> <li>• Tipo de red</li> </ul>
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>• OBJ-003</li> </ul>

Cuadro 5.5: Requisito de información 002

RI-003	Metadatos
<b>Descripción</b>	El sistema deberá almacenar los parámetros de entrenamiento de las redes.
<b>Datos específicos</b>	<ul style="list-style-type: none"><li>• Batch</li><li>• Delta</li><li>• Métrica de error</li><li>• Valor de error</li></ul>
<b>Dependencias</b>	<ul style="list-style-type: none"><li>• OBJ-003</li></ul>

Cuadro 5.6: Requisito de información 003

### 5.3.3. Requisitos funcionales

RF-001	Obtención de información
<b>Descripción</b>	El sistema deberá obtener el valor actual del EUR/USD y el BTC/USD por minuto.
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>• OBJ-001</li> <li>• OBJ-002</li> <li>• OBJ-003</li> <li>• RI-001</li> </ul>

Cuadro 5.7: Requisito funcional 001

RF-002	Envío de información
<b>Descripción</b>	El sistema deberá tratar los datos obtenidos y almacenarlos.
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>• OBJ-001</li> <li>• OBJ-002</li> <li>• RI-001</li> </ul>

Cuadro 5.8: Requisito funcional 002

RF-003	Visualización del histórico
<b>Descripción</b>	El sistema deberá mostrar el histórico de cada divisa.
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>• OBJ-003</li> <li>• RI-001</li> </ul>

Cuadro 5.9: Requisito funcional 003

RF-004	Interacción con las gráficas
Descripción	El sistema deberá permitir interactuar con las gráficas.
Dependencias	<ul style="list-style-type: none"><li>• OBJ-003</li><li>• RI-001</li><li>• RI-002</li></ul>

Cuadro 5.10: Requisito funcional 004

RF-005	Visualización de metadatos
Descripción	El sistema deberá mostrar los metadatos relativos a la red.
Dependencias	<ul style="list-style-type: none"><li>• OBJ-003</li><li>• RI-003</li></ul>

Cuadro 5.11: Requisito funcional 005

### 5.3.4. Requisitos no funcionales

RNF-001	Compatibilidad
Descripción	El sistema deberá ser compatible con los principales navegadores.
Dependencias	<ul style="list-style-type: none"><li>OBJ-003</li></ul>

Cuadro 5.12: Requisito no funcional 001

## 5.4– Casos de usos

Los casos de uso que recoge la aplicación se detallan a continuación:

CU-001	Visualización de predicciones
<b>Descripción</b>	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee ver las predicciones generadas por el sistema.
<b>Dependencias</b>	<ul style="list-style-type: none"><li>• OBJ-003</li><li>• RF-003</li></ul>
<b>Precondición</b>	El sistema ha entrenado correctamente y existen resultados en la base de datos.
<b>Secuencia normal</b>	<ol style="list-style-type: none"><li>1. El usuario accede a la interfaz web.</li><li>2. El sistema renderiza una gráfica con los valores de la divisa EUR/USD para la red recurrente como valores por defecto.</li><li>3. El usuario, mediante el menú desplegable, elige una divisa.</li><li>4. El sistema renderiza una nueva gráfica para la divisa seleccionada.</li><li>5. El usuario, mediante las pestañas, cambia de tipo red.</li><li>6. El sistema renderiza una nueva gráfica para el tipo de red seleccionado.</li></ol>

Cuadro 5.13: Caso de uso 001

CU-002	Visualización de metadatos
<b>Descripción</b>	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee ver los metadatos.
<b>Dependencias</b>	<ul style="list-style-type: none"><li>• OBJ-003</li><li>• RF-005</li></ul>
<b>Precondición</b>	El sistema ha entrenado correctamente y existen metadatos en la base de datos.
<b>Secuencia normal</b>	<ol style="list-style-type: none"><li>1. El usuario accede a la interfaz web.</li><li>2. El sistema renderiza una tabla con los metadatos de la divisa EUR/USD para la red recurrente como valores por defecto.</li><li>3. El usuario, mediante el menú desplegable, elige una divisa.</li><li>4. El sistema renderiza una nueva tabla para la divisa seleccionada.</li><li>5. El usuario, mediante las pestañas, cambia de tipo red.</li><li>6. El sistema renderiza una nueva tabla para el tipo de red seleccionado.</li></ol>

Cuadro 5.14: Caso de uso 002

CU-003	Interacción con la gráfica
<b>Descripción</b>	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee interactuar con la gráfica de resultados.
<b>Dependencias</b>	<ul style="list-style-type: none"> <li>• OBJ-003</li> <li>• RF-005</li> <li>• RI-003</li> </ul>
<b>Precondición</b>	El sistema ha entrenado correctamente y existen resultados en la base de datos.
<b>Secuencia normal</b>	<ol style="list-style-type: none"> <li>1. El usuario accede a la interfaz web.</li> <li>2. El sistema renderiza una tabla con los metadatos de la divisa EUR/USD para la red recurrente como valores por defecto.</li> <li>3. El usuario accede a la interfaz web.</li> <li>4. El sistema renderiza una tabla con los metadatos de la divisa EUR/USD para la red recurrente como valores por defecto.</li> <li>5. El usuario, mediante el menú en la esquina superior derecha de la gráfica, aumenta el zoom de la misma.</li> <li>6. El usuario, mediante el menú en la esquina superior derecha de la gráfica, disminuye el zoom de la misma.</li> </ol>

Cuadro 5.15: Caso de uso 003

## 5.5– Matriz de trazabilidad

En esta sección mostraremos la matriz de trazabilidad con la conexión entre los requisitos y los casos de uso.

CU, RI, RF, / OBJ	OBJ-001	OBJ-002	OBJ-003
CU-001	-	-	✓
CU-002	-	-	✓
CU-003	-	-	✓
RI-001	✓	✓	✓
RI-002	-	-	✓
RI-003	-	-	✓
RF-001	✓	✓	✓
RF-002	✓	✓	-
RF-003	-	-	✓
RF-004	-	-	✓
RF-005	-	-	✓

Figura 5.3: Matriz de trazabilidad de casos de uso, requisitos de información y requisitos funcionales contra requisitos generales



# CAPÍTULO 6

---

## Análisis temporal y costes de desarrollo

---

En este capítulo se muestran las tareas a realizar para completar el proyecto. Se detallan las tareas así como las horas estimadas y las fechas de inicio y fin, mostrando además un diagrama de gantt y el coste económico asociado.

### 6.1– Desglose de tareas

Este proyecto se ha dividido en módulos. Este planteamiento nos permitirá cerrar bloques completos para poder avanzar en el desarrollo sin necesidad de volver atrás a modificar/corregir los módulos ya finalizados.

Este trabajo ha sido desarrollado por dos alumnos, por ello todas las responsabilidades se han dividido entre ambos. Se acordó, al principio del proyecto, que ambos alumnos estarían presentes en el desarrollo todos los módulos, con el objetivo de estar los dos al tanto de cualquier problema que surgiera en cada uno de ellos.

Además, se han definido unos roles para el desarrollo de las tareas. De esta forma, nos adaptaremos a la mentalidad que exige el rol para el desempeño de la tarea.

Estos roles son:

- **Jefe de proyecto:** Puesto de mayor responsabilidad, en el que hay tomar decisiones que puedan cambiar el rumbo del proyecto.
- **Analista:** Las tareas del analista serán tareas para definir otras tareas, y para mostrar al cliente el resultado del trabajo.
- **Científico de datos:** Llevará a cabo todas las tareas relacionadas con los algoritmos de Deep Learning.
- **Desarrollador:** Llevará a cabo las tareas de desarrollado e implementación del sistema.

Los módulos en los que se ha dividido este proyecto son:

1. **Inicio del proyecto** - Jefe de proyecto y Analista

Este módulo tiene como fin planificar el desarrollo del proyecto. Se llevarán a cabo diferentes reuniones entre los miembros del equipo y el profesor para realizar una planificación realista con un alcance alto.

Se realizarán las siguientes tareas:

- Identificación de objetivos, alcances, fortalezas, problemas.
- Estudio previo del problema.
- Diseño de la implementación.
- Elección de tecnologías.
- Brain storming para conseguir ideas sobre diseños de arquitecturas.

## 2. Requisitos - Analista

Durante este proceso se realizarán reuniones para elaborar los requisitos en base a las tecnologías a utilizar. Es una fase crucial en el proyecto, ya que a partir de este módulo se empieza el desarrollo.

Se realizarás las siguientes tareas:

- Elicitación de requisitos.
- Documentación de estos requisitos.

## 3. Estudio Formativo - Analista, Científico de datos y Desarrollador

Se han elegido tecnologías muy innovadoras para el desarrollo de este proyecto. Estas tecnologías requieren de un conocimiento previo para poder desarrollar los módulos de forma eficiente.

Durante esta fase se estudiarán:

- Mercado Forex.
- Sistemas Big Data, Apache Spark, Apache Kafka.
- Fundamentos teóricos del Deep Learning y herramientas para poder desarrollar estos algoritmos, como Tensorflow.

## 4. Extracción de datos históricos - Analista

Este módulo tiene como objetivo obtener los datos de las monedas en el proyecto para usarlos durante el desarrollo. Estos datos tienen que cumplir unos requisitos mínimos, como la calidad del dato (6 decimales), y el nivel de detalle (datos a nivel de minuto).

## 5. Módulo Extracción de información - Analista y Desarrollador

En este módulo se definirán y desarrollarán los extractores de datos. Estos datos se deben recoger en tiempo real.

Las tareas a desarrollar son:

- Aprendizaje sobre el framework BeautifulSoup.
- Diseño de un sistema escalable.
- Programación de los scripts de los extractores.
- Pruebas del sistema.

## 6. Módulo Almacenamiento de información - Analista y Desarrollador

En este módulo se configurarán y desarrollarán los sistemas encargados de almacenar la información. Esta información es la información extraída en tiempo real, la información procesada y la información que el modelo va a generar.

Las tareas a desarrollar son:

- Estudio del funcionamiento de Apache Kafka, y las diferentes formas de implementarlo.
- Instalación de las herramientas Apache Spark y Apache Kafka.

- Implementar la conexión entre Spark y Kafka.
- Optimización del sistema Apache Spark.
- Dockerizar el entorno.

**7. Módulo Red Neuronal - Analista y Científico de datos**

Durante esta fase se desarrollarán los modelos predictivos basados en redes neuronales.

Las tareas a desarrollar son:

- Diseñar la arquitectura de la red neuronal recurrente y convolucional.
- Búsqueda y selección de los hiperparámetros de las redes neuronales.
- Pruebas del sistema.
- Selección del formato de los datos y de la salida de las redes neuronales.

**8. Módulo Capa de presentación - Jefe de proyecto, Analista y Desarrollador**

Se desarrollará un sistema front-end para visualizar los datos que ha procesado la red neuronal.

Las tareas a desarrollar son:

- Investigación de las aplicaciones web reactivas.
- Diseño y evaluación de la interfaz.
- Dockerizar el entorno del módulo.
- Integrar este sistema con el resto.

**9. Experimentación - Analista, Científico de datos y Desarrollador**

En este módulo, se realizarán pruebas de todos los sistemas desarrollados, poniendo especial enfasis en las redes neuronales.

- Se evaluarán las redes con monedas de baja volatilidad y alta volatilidad, junto con los resultados de ambas redes.
- Se evaluará el funcionamiento conjunto de los sistemas.
- Se evaluarán las arquitecturas de las redes para encontrar posibles fallos de rendimiento

**10. Memoria - Jefe de proyecto**

Se elaborará la documentación de todos los conocimientos y resultados obtenidos durante el periodo de desarrollo.

**11. Defensa - Jefe de proyecto**

Realización de la defensa del Trabajo de Fin de Grado.

## 6.2– Análisis temporal

La planificación del proyecto que se puede ver en la figura 6.1 y 6.2 detalla el tiempo estimado a cada tarea definida en la sección anterior. Se pueden ver las fechas estimadas de inicio y de fin de cada actividad. Se ha realizado una planificación de 600 horas, que son las requeridas para un trabajo de fin de grado para dos alumnos según el plan de estudios.

Bloque	Tarea	Duración	Fecha Inicio	Fecha Fin	Rol
<b>Inicio del Proyecto</b>		<b>39</b>	<b>1/10/18</b>	<b>12/10/18</b>	
	Identificar objetivos	4	1/10/18	1/10/18	Jefe de Proyecto
	Estudio previo del problema	10	2/10/18	4/10/18	Jefe de Proyecto
	Diseño de implementación	16	5/10/18	10/10/18	Analista
	Elección de tecnologías	4	11/10/18	11/10/18	Analista
	Brain Storming	5	12/10/18	12/10/18	Analista
<b>Requisitos</b>		<b>24</b>	<b>22/11/18</b>	<b>28/11/18</b>	
	Elicitación de requisitos	18	22/11/18	26/11/18	Analista
	Documentación	6	26/11/18	28/11/18	Analista
<b>Estudio</b>		<b>145</b>	<b>14/10/18</b>	<b>24/11/18</b>	
	Estudio forex	20	14/10/18	18/10/18	Analista
	Estudio spark	35	19/10/18	28/10/18	Analista
	Estudio kafka	20	29/10/18	6/11/18	Analista
	Estudio deep learning/tensorflow	60	7/11/18	20/11/18	Científico de datos
	Curso Kafka – Cloudkafka	10	21/11/18	24/11/18	Desarrollador
<b>Extracción de datos históricos</b>		<b>10</b>	<b>1/12/18</b>	<b>3/12/18</b>	Analista
<b>Modulo extracción de información</b>		<b>21</b>	<b>3/12/18</b>	<b>11/12/18</b>	
	Tutorial BeautifulSoup	3	3/12/18	3/12/18	Desarrollador
	Diseño escalable	6	4/12/18	6/12/18	Analista
	Programación	8	8/12/18	10/12/18	Desarrollador
	Testeo	4	11/12/18	11/12/18	Desarrollador
<b>Modulo almacenamiento de información</b>		<b>62</b>	<b>12/12/18</b>	<b>15/1/19</b>	
	Funcionamiento Kafka	12	12/12/18	20/12/18	Desarrollador
	Otras implementaciones	2	12/12/18	12/12/18	Desarrollador
	Instalación del entorno Kafka	4	12/12/18	12/12/18	Analista
	Instalación del entorno Spark	8	17/12/18	18/12/18	Analista
	Conexión Kafka – Spark	12	18/12/18	20/12/18	Desarrollador
	Scripts consumidor Spark	5	20/12/18	20/12/18	Desarrollador
	Param Tuning Spark	15	18/12/18	15/1/19	Analista
	Dockerización entorno	4	22/12/18	22/12/18	Desarrollador

Figura 6.1: Planificación temporal I

Bloque	Tarea	Duracion	Fecha Inicio	Fecha Fin	Rol
<b>Modulo red neuronal</b>		<b>94</b>	<b>20/2/19</b>	<b>24/3/19</b>	
	Arquitectura red recurrente	26	20/2/19	2/3/19	Cientifico de datos
	Arquitectura red convolucional	20	24/2/19	8/3/19	Cientifico de datos
	Seleccion Hiperparametros	14	5/3/19	17/3/19	Cientifico de datos
	Pruebas en redes	22	1/3/19	20/3/19	Cientifico de datos
	Seleccion formato de datos	6	22/3/19	24/3/19	Analista
	Seleccion salida de datos	6	22/3/19	24/3/19	Analista
<b>Modulo capa de presentación</b>		<b>40</b>	<b>1/4/19</b>	<b>15/4/19</b>	
	Investigacion aplicaciones reactivas	10	1/4/19	3/4/19	Analista
	Desarrollo de una maqueta	2	4/4/19	4/4/19	Desarrollador
	Diseño interfaz	15	4/4/19	10/4/19	Analista
	Evaluuar integración en pipeline	3	12/4/19	12/4/19	Analista
	Dockerizacion entorno	4	13/4/19	13/4/19	Desarrollador
	Pruebas User Experience	6	12/4/19	15/4/19	Jefe de Proyecto
<b>Experimentacion</b>		<b>97</b>	<b>4/3/19</b>	<b>25/5/19</b>	
	Pruebas redes baja volatilidad	30	1/5/19	25/5/19	Cientifico de datos
	Pruebas redes alta volatilidad	22	1/5/19	25/5/19	Cientifico de datos
	Evaluacion resultados	10	1/5/19	25/5/19	Analista
	Pruebas arquitectura redes	15	4/3/19	25/3/19	Cientifico de datos
	Pruebas pipeline proyecto	20	15/5/19	25/5/19	Desarrollador
<b>Memoria</b>		<b>60</b>	<b>1/5/19</b>	<b>30/5/19</b>	Jefe de Proyecto
<b>Defensa</b>		<b>8</b>	<b>25/6/19</b>	<b>27/6/19</b>	Jefe de Proyecto

Figura 6.2: Planificación temporal II

### **6.2.1. Diagrama de Gantt**

En las figuras 6.3 y 6.4 se muestra el diagrama de Gantt con la planificación de tareas. Estas figuras se han realizado con la herramienta Gantt Project.

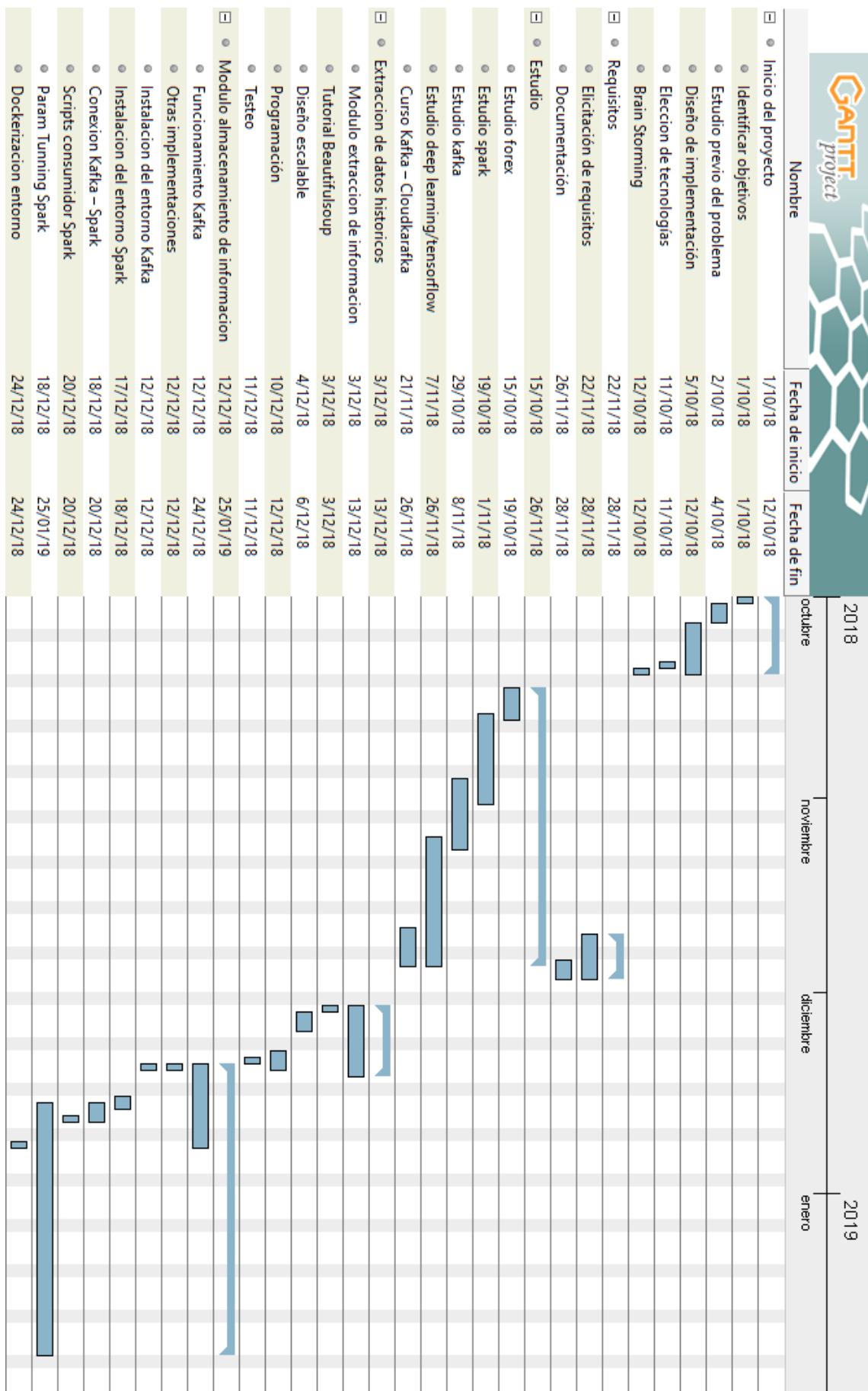


Figura 6.3: Diagrama Gantt I

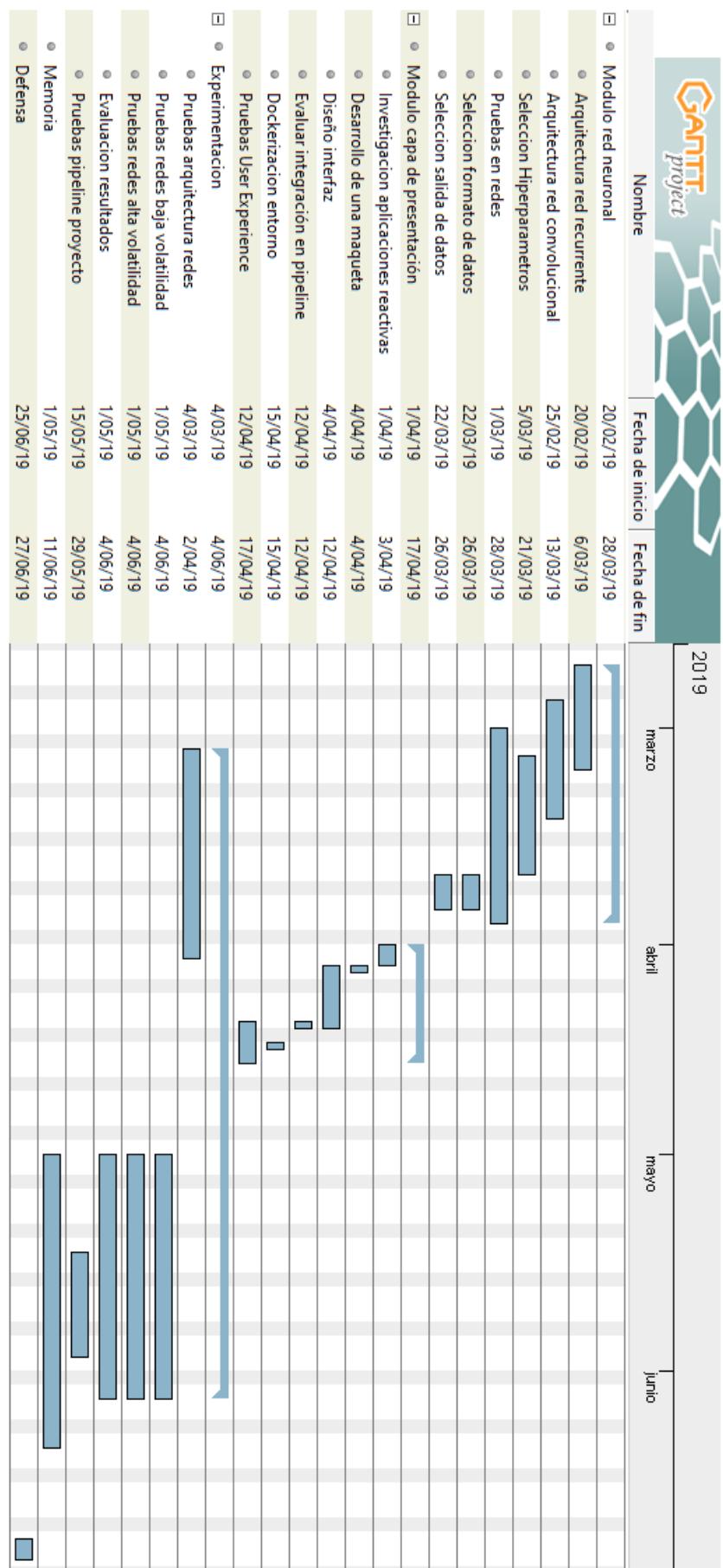


Figura 6.4: Diagrama Gantt II

## 6.3– Costes de desarrollo

En esta sección se detallan los costes estimados del proyecto.

### 6.3.1. Recursos de hardware

A continuación, se detallan recursos de hardware usados para el desarrollo del proyecto.

Equipo 1	
CPU	Intel Core i7-6700K
RAM	32GB
GPU	NVIDIA RTX 2060
SSD	250 GB
Sistema Operativo	Ubuntu 18.04
Coste	2000€

Cuadro 6.1: Equipo 1

Equipo 2	
CPU	Intel Core i5-6600K
RAM	16GB
GPU	NVIDIA GTX 1060
SSD	250 GB
Sistema Operativo	Ubuntu 18.04
Coste	1800€

Cuadro 6.2: Equipo 2

Equipo 3	
CPU	Intel Core i7
RAM	16GB
GPU	Intel Iris Plus Graphics 655
SSD	250 GB
Sistema Operativo	MacOS
Coste	2400€

Cuadro 6.3: Equipo 3

Equipo 4	
CPU	Intel Core i5
RAM	16GB
GPU	NVIDIA GTX 1050
SSD	250 GB
Sistema Operativo	Ubuntu 18.04
Coste	1200€

Cuadro 6.4: Equipo 4

AWS	
Tipo	p3.2xlarge
CPU	8 núcleos
RAM	61GB
GPU	NVIDIA Tesla V100 16 GB
Precio	3.06 €/hora
Tiempo de uso	15 horas
Coste	45.9€

Cuadro 6.5: Servidor AWS

Los equipos 1 y 2 se han usado para tareas de investigación sobre Deep Learning ya que cuenta con una potencia aceptable para los entrenamientos, también han sido usados para las tareas de desarrollo del resto del proyecto y para la realización de la memoria. Los equipos 3 y 4 al ser equipos portátiles y no contar con potencia suficiente para entrenar modelos en un tiempo aceptable han sido destinados para tareas de investigación y desarrollo.

Una vez desarrollado el modelo se han usado el servidor AWS para el entrenamiento del mismo.

### 6.3.1.1. Amortización

A continuación, se calculará la amortización asociada a cada uno de los equipos. Un equipo se amortiza en 3 años aproximadamente con un total de 5760 horas de trabajo siendo el número de horas invertidas en este trabajo de 600 horas a repartir entre los 4 equipos.

Amortización	
Equipo 1	70€
Equipo 2	62.5€
Equipo 3	41.7€
Equipo 4	20.8€

Cuadro 6.6: Amortización

### 6.3.2. Recursos de personal

A continuación, se detallan los roles desempeñados durante el desarrollo del proyecto y su impacto económico. El precio por hora de los distintos perfiles se detallan en el cuadro 6.7.

Precio por hora según perfil	
Jefe de proyecto	50€/hora
Analista	40€/hora
Científico de datos	32€/hora
Desarrollador	32€/hora

Cuadro 6.7: Precio/hora por perfil

El coste total de personal del proyecto calculando las horas por perfil se muestra en el cuadro 6.8.

Coste por personal		
Jefe de proyecto	88 horas	4400€
Analista	217 horas	8680€
Científico de datos	209 horas	6688€
Desarrollador	86 horas	2752€
<b>Coste total</b>		<b>22520 €</b>

Cuadro 6.8: Coste total por perfil

### 6.3.3. Coste total

El coste total del proyecto sumando el coste por personal y el coste por hardware se puede observar en el cuadro 6.9.

Servidor AWS	45.9€
Amortización	195€
Personal	22520€
<b>Coste total</b>	<b>22760.9 €</b>

Cuadro 6.9: Coste total del proyecto



# CAPÍTULO 7

---

## Manual de usuario

---

En este capítulo se mostrará un manual de usuario que detalle los procesos de instalación e implantación del sistema.

El código de este proyecto se encuentra un repositorio de Github, cuyo enlace es:

<https://github.com/carlosevi94/TFG-Forex-DeepLearning>

### 7.1– Preparación del entorno

Antes de poder utilizar el sistema, hay que tener en cuenta unas consideraciones previas.

#### 7.1.1. Requisitos mínimos

- CPU: CPU con 4 núcleos o superior.
- RAM: 8 GB o superior
- SO: Ubuntu 14.04 o superior
- Software: El sistema debe tener instalado Python en su versión 3.5 o superior, junto a la herramienta Pip3.

#### 7.1.2. Entorno

En la carpeta *tools* del código del proyecto se encuentra un archivo llamado *requirements.txt*, que contiene las dependencias necesarias para ejecutar el proyecto. Para instalarlas basta con ejecutar en una consola bash de Ubuntu este comando:

```
1 pip3 install -r requirements.txt
```

Una vez instaladas las dependencias de Python, hay que instalar Spark.

En la carpeta *tools/spark* se encuentra un documento redactando siguiendo la sintaxis Markdown en el que se encuentran una guía de instalación que hemos diseñado para que se pueda instalar Spark de forma sencilla.

Además, hay que instalar Docker y Docker Compose. Para ello, es necesario abrir una consola bash y lanzar los siguientes comandos :

```
1 sudo apt-get update
2 sudo apt install docker.io
```

Una vez instalado, se inicializa el servicio con el comando:

```
1 sudo systemctl start docker
```

Posteriormente, hay que ejecutar estos comandos para instalar Docker Compose:

```
1 sudo curl -L "https://github.com/docker/compose/releases/download
  /1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/
  docker-compose
2 sudo chmod +x /usr/local/bin/docker-compose
```

La instalación se verifica lanzando este comando:

```
1 docker-compose --version
```

Si todo ha ido bien, devolverá por pantalla el siguiente mensaje:

*docker-compose version 1.23.1, build b02f1306*

## 7.2– Instalación e implantación

Una vez que tenemos el entorno preparado, podemos empezar la instalación e implantación del sistema.

Para arrancar el sistema hay que seguir las siguientes instrucciones:

### 1. Iniciar Kafka y Zookeeper.

Apache Kafka necesita de una herramienta adicional de Apache para funcionar correctamente, Apache Zookeeper. Esta herramienta no aporta funcionalidad a este proyecto debido a que su función es configurar diferentes productos Big Data de apache en un mismo sistema. En nuestro sistema Kafka está separado en un Docker, pero es igualmente necesario utilizarlo para que Kafka conozca los recursos que el sistema tiene disponible.

Para lanzarlo, basta con ejecutar en una consola bash.

```
1 cd /tools/kafka
2 docker-compose up -d
```

### 2. Crear topics en Kafka.

Una vez inicializado Kafka, hay que lanzar unos comandos para crear los topics en los que se van a almacenar los datos.

VAR es una variable que se ha de sustituir por la IP que tenga asociada el contenedor de kafka. Esta IP se puede obtener lanzando el comando *docker inspect kafka \_cont*.

```
1 docker-compose exec kafka kafka-topics --create --topic usd_eur --
  partitions 1 --replication-factor 1 --if-not-exists --zookeeper
  VAR:2181
2 docker-compose exec kafka kafka-topics --create --topic btc_usd --
  partitions 1 --replication-factor 1 --if-not-exists --zookeeper
  VAR:2181
```

### 3. Productores de datos.

Para lanzar los productores de datos es necesario abrir dos consolas bash.

En la primera, se tienen que ejecutar los siguientes comandos:

```
1 cd producer
2 python3 producer_BTC_USD.py
```

En la segunda, se tienen que ejecutar los siguientes comandos:

```
1 cd producer
2 python3 producer_EUR_USD.py
```

### 4. Iniciar MySQL.

Se ha incluido una imagen de Docker que contiene una base de datos MySQL con un script que crea la base de datos y las tablas necesarias.

Para iniciararlo, hay que abrir una consola bash y ejecutar los siguientes comandos.

```
1 cd tools/front_end/
2 docker build -t "mysql_dash_tfg" .
3 docker run -d --rm --name mysql_dash_cont mysql_dash_tfg
```

### 5. Archivos de configuración.

Una vez lanzados todos los sistemas, hay que modificar sus archivos de configuración. A partir de este paso, cada script de los que se van a lanzar tiene un archivo de configuración asociado en formato json. Hay que modificar las IPs de todos los sistemas, que se pueden consultar ejecutando en una consola *docker inspect* sobre los contenedores de Docker.

### 6. Consumidores de datos.

Los consumidores de datos se han diseñado para que se puedan lanzar de forma sencilla para el usuario. Necesitamos abrir dos consolas bash, en la primera se ejecutarán estos comandos:

```
1 cd spark_scripts
2 python3 spark_consumer.py config_usd
```

Y en la segunda consola, se ejecutarán los siguientes:

```
1 cd spark_scripts
2 python3 spark_consumer.py config_btc
```

### 7. Redes neuronales.

Las redes están preparadas para que se lancen con un solo comando y se dejen funcionando.

Se necesitan 4 consolas bash, en cada una hay que ejecutar los siguientes comandos.

#### a) Red Neuronal Recurrente con la moneda EUR/USD

```
1 cd neural_network
2 python3 rnn.py --currency=EURUSD --initalize
```

#### b) Red Neuronal Convolucional con la moneda EUR/USD

```
1 cd neural_network
2 python3 cnn.py --currency=EURUSD --initalize
```

#### c) Red Neuronal Recurrente con la moneda BTC/USD

```
1 cd neural_network
2 python3 rnn.py --currency=BTCUSD --inizalize
```

d) Red Neuronal Convolucional con la moneda BTC/USD

```
1 cd neural_network
2 python3 cnn.py --currency=BTCUSD --inizalize
```

## 8. Aplicación front-end.

Finalmente hay que lanzar el front-end que se ha desarrollado para visualizar los datos.

Hay que abrir una consola bash y ejecutar:

```
1 cd front_end/webapp
2 python3 app.py
```

Si se han seguido estos pasos correctamente, el sistema estará funcionando correctamente. Para comprobarlo, entramos en <https://localhost:8050> y veremos la web como se ve en la figura 3.10.



# CAPÍTULO 8

---

## Pruebas

---

El sistema ha sido sometido a diferentes pruebas para valorar aspectos fundamentales como el rendimiento, la fiabilidad y la tolerancia a fallos.

Todas las pruebas cuentan con el mismo formato: Una **descripción inicial**, una explicación de las **características** y condiciones iniciales y el **resultado** de la prueba.

### 8.1– Spark y Kafka

Se ha realizado una prueba de stress para ver la capacidad de Apache Kafka a tolerar una gran carga de trabajo. La prueba que se ha realizado consiste en crear un script que genera productores de datos que envían datos generados aleatoriamente.

#### Características de la prueba

- Cada productor de datos genera 20 cadenas de texto cada 10 segundos.
- El script comienza con un productor y cada 10 segundos duplica el numero de productores activos.
- Se han utilizado dos equipos. El primero ejecuta a una instancia de Kafka y de Spark procesando los datos, mientras que el segundo ejecuta el script que genera los productores.
- La forma de evaluar esta prueba considera como fallo que el sistema se cuelgue, se apague o tarde más de 20 segundos en procesar los datos y éxito en cualquier otro caso. Para ello se ha configurado Spark de manera que procese todos los datos que reciba de Kafka en tiempo real.

#### Resultado de la prueba

No se ha podido determinar un punto de stress de Kafka, dado que cuando había 128 productores activos el equipo que generaba los datos se bloqueó, mientras que el equipo que ejecutaba a Kafka y Spark seguía funcionando.

El punto de mayor trabajo se asume que ha sido el momento donde había activos 64 productores. Como cada productor generaba 10 cadenas de datos, hace un total de 640 datos nuevos en un periodo de 10 segundos. Este rendimiento es muy superior al exigido, por ello no se realizarán más pruebas de este sistema.

## 8.2– Estudio y selección de hiperparámetros de redes neuronales

Cuando se habla de hiperparámetros en redes neuronales se hace referencia al número de capas que tiene que tener la red, el tipo de dichas capas y el número de neuronas.

Esta prueba tiene como objetivo conocer la arquitectura que mejor se adapta para nuestro problema. Será ejecutada en un servidor de AWS dotado con mucha potencia de computación y posibilidad de lanzamientos paralelos, para reducir el tiempo de ejecución de la misma.

### Características de la prueba

- Se va a buscar el número de capas LSTM óptimo para la red recurrente, probando arquitecturas de 2 a 4 capas.
- Se va a buscar el número de neuronas óptimo para cada capa de la arquitectura, probando entre 16, 32, 64, 128, 256 y 512 neuronas.
- Se ha utilizado la herramienta GridSearch de Scikit-Learn para probar todas las combinaciones explicadas. Esta herramienta realiza una prueba con todas las posibles arquitecturas que puede tomar la red con los parámetros indicados en los dos puntos anteriores, ejecutando así 775 pruebas diferentes.
- Se utilizará el dataset de febrero de 2018 para estas pruebas.
- Para evaluar cada prueba se ha utilizado la métrica MSE.

### Resultado de la prueba

La ejecución de esta prueba se ha realizado en un ordenador con un procesador gráfico con alta capacidad de cómputo, con el que se ha podido parallelizar la ejecución. El tiempo que tarda en ejecutar cada combinación es mayor a medida que se avanza, por ello, las últimas pruebas retrasaron la ejecución. Ésto es debido a que las redes con mayor número de capas y neuronas se han ejecutado las últimas.

Capas	Neuronas	MSE
2	[16,16]	0.000141
<b>2</b>	<b>[128,128]</b>	<b>0.00000041</b>
2	[256,256]	0.000064
3	[64,64,64]	0.000021
3	[128,16,128]	0.00003242
4	[32,32,32,32]	0.00026842
4	[64,32,32,64]	0.00000623
4	[64,128,256,512]	0.000033654
4	[512,256,128,64]	0.000011523
4	[512,512,512,512]	0.000361

Cuadro 8.1: Soluciones a la prueba de selección de hiperparámetros

En el cuadro 8.1 se pueden ver algunos ejemplos de arquitecturas que GridSearch ha probado. Esta muestra se ha escogido en base a las arquitecturas más interesantes.

Como se puede observar a simple vista, el error es siempre muy bajo, ésto es debido a que los datos con los que entrena la red se encuentran muy agrupados. En ciertos momentos del día el valor del par puede que no se haya movido, o que haya fluctuado un orden de 0.0001 sobre sí mismo.

Con lo mencionado anteriormente, se pretende explicar que un error medio cuadrático bajo no garantiza precisión en las predicciones. Un método más fiable es el de reescalar los datos en otra magnitud, usando un factor multiplicador alto o usando técnicas de enconding, como LabelEncoder, lo que mostraría una aproximación más realista sobre los errores cometidos. El inconveniente de utilizar estos métodos es que se necesita una capacidad de cómputo muy superior a la que se ha utilizado en la prueba.

Como se puede ver en el cuadro 8.1, la segunda fila está destacada sobre el resto. Ésto es debido a que la configuración mostrada fue la que obtuvo el error más bajo en esta prueba.

El tiempo de ejecución de la prueba fue de 15 horas aproximadamente. Como la prueba fue ejecutada en un servidor de AWS, que tenía mayores recursos que los equipos que teníamos disponible, era posible la paralelización de la prueba. Al poderse realizar una ejecución de las pruebas en GPU, el tiempo de ejecución se veía reducido, permitiendo así la posibilidad de ejecutar un número de pruebas tan alto.

Como conclusión, la red con mejor resultado está compuesta por dos capas LSTM, cada una con 128 neuronas. Se implementó esta red para el modelo en producción, pero tiempo después se ha descubierto un desempeño mejor si la primera capa la componen 180 neuronas.

### 8.3– Red Recurrente - EUR/USD - Puesta en producción

#### Características de la prueba

- El par que se utiliza en esta prueba es el EUR/USD.
- El sistema acaba de ponerse en producción, lo que conlleva que acaba de realizar un entrenamiento con los últimos datos disponibles.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

#### Resultado de la prueba



Figura 8.1: Resultado de las predicciones de la red recurrente tras ponerse en producción con el par EUR/USD

En el gráfico de la figura 8.1 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red recurrente en el momento de lanzarse en producción en sus primeras predicciones no consigue adaptarse a la tendencia real del par.

También se puede observar que falta un valor, debido a que al ponerse en producción todos los módulos del proyecto puede existir un pequeño retraso a la hora de ejecutar las predicciones. Este error solo puede ocurrir en este momento, debido a que el reloj que controla cuando tiene que re-entrenar la red tiene en cuenta la posibilidad de que ésto ocurra y lo corrige. Este error puede aparecer en otras pruebas similares.

La diferencia entre el resultado real no coincide en los primeros 15 minutos, mientras que a medida que pasa el tiempo la predicción empieza a suavizarse. El sistema debería adaptarse mejor al principio, no obstante, el sistema funcionará mejor cuanto más tiempo lleve funcionando.

En el cuadro 8.2 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.1, donde existen simultáneamente una predicción y un valor real.

Momento	t0	t1	t2	t3	t4
Error	$3,4 \times 10^{-9}$	$4,46 \times 10^{-9}$	$4,2 \times 10^{-9}$	$2,2 \times 10^{-8}$	$8,9 \times 10^{-9}$
Momento	t5	t6	t7	t8	t9
Error	$2,016 \times 10^{-9}$	$2,016 \times 10^{-9}$	$1,28 \times 10^{-9}$	$5,71 \times 10^{-10}$	$2,32 \times 10^{-10}$
Momento	t10	t11	t12	t13	t14
Error	$1 \times 10^{-10}$	$3,6 \times 10^{-11}$	$5,8 \times 10^{-10}$	$8,74 \times 10^{-9}$	$3,72 \times 10^{-10}$

Cuadro 8.2: Error cuadrático asociado a las predicciones de la puesta en producción del par EUR/USD

## 8.4– Red Recurrente - EUR/USD - 24 horas en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el EUR/USD.
- El sistema lleva en producción 24 horas, por lo que se ha podido reentrenar con los datos que el mismo ha obtenido.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.2: Resultado de las predicciones de la red recurrente tras 24 horas en producción con el par EUR/USD

En el gráfico de la figura 8.2 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red recurrente, tras 24 horas en producción, se adapta muy bien en el primer tramo de predicciones, perdiendo confianza en las predicciones más lejanas en el tiempo.

Ha estado recibiendo datos nuevos con los que ha reentrenado este modelo para ajustar los pesos a los nuevos datos.

En el cuadro 8.3 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.2, donde existen simultáneamente una predicción y un valor real.

Momento	<b>t0</b>	<b>t1</b>	<b>t2</b>	<b>t3</b>	<b>t4</b>
Error	$4,9 * 10^{-32}$	$9,94 * 10^{-10}$	$1,19 * 10^{-9}$	$2,5 * 10^{-9}$	$2,5 * 10^{-9}$
Momento	<b>t5</b>	<b>t6</b>	<b>t7</b>	<b>t8</b>	<b>t9</b>
Error	$1,6 * 10^{-9}$	$9 * 10^{-10}$	$1,6 * 10^{-9}$	$1,6 * 10^{-9}$	$9 * 10^{-10}$
Momento	<b>t10</b>	<b>t11</b>	<b>t12</b>	<b>t13</b>	<b>t14</b>
Error	$1,6 * 10^{-9}$	$4,22 * 10^{-9}$	$2,89 * 10^{-8}$	$6,76 * 10^{-8}$	$6,69 * 10^{-8}$

Cuadro 8.3: Error cuadrático asociado a las predicciones tras 24 horas en producción del par EUR/USD

## 8.5– Red Recurrente - BTC/USD - Puesta en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el BTC/USD.
- El sistema acaba de ponerse en producción, lo que conlleva que acaba de realizar un entrenamiento con los últimos datos disponibles.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.3: Resultado de las predicciones de la red recurrente tras ponerse en producción con el par BTC/USD

En el gráfico de la figura 8.3 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red recurrente, al ponerse en producción, en sus primeras predicciones se ajusta bastante bien al precio real. Este par es muy volátil, con lo que las predicciones pierden confianza a medida que avanza en el tiempo.

En el cuadro 8.4 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.3, donde existen simultáneamente una predicción y un valor real.

Momento	t0	t1	t2	t3	t4
Error	6,3001	8,4681	0,0081	12,8164	4,3681
Momento	t5	t6	t7	t8	t9
Error	9,5481	45,1584	43,4281	27,9841	27,9841
Momento	t10	t11	t12	t13	t14
Error	5,8081	19,4481	6,9169	2,4964	0,0484

Cuadro 8.4: Error cuadrático asociado a las predicciones de la puesta en producción del par BTC/USD con la red recurrente

## 8.6– Red Recurrente - BTC/USD - 24 horas en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el BTC/USD.
- El sistema lleva en producción 24 horas, por lo que se ha podido reentrenar con los datos que el mismo ha obtenido.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.4: Resultado de las predicciones de la red recurrente tras 24 horas en producción con el par BTC/USD

En el gráfico de la figura 8.4 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red recurrente tras 24 horas en producción no se adapta al precio real del par.

Esta red no ha obtenido buenos resultados tras los re-entrenamientos, con lo que se necesitaría una estrategia diferente de entrenamiento.

En el cuadro 8.5 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.4, donde existen simultáneamente una predicción y un valor real.

Momento	t0	t1	t2	t3	t4
Error	0,00	0,25	1,00	9,00	12,25
Momento	t5	t6	t7	t8	t9
Error	16,00	16,00	16,00	19,36	24,01
Momento	t10	t11	t12	t13	t14
Error	3,24	1,00	0,16	0,81	9,00

Cuadro 8.5: Error cuadrático asociado a las predicciones después de 24 horas de la puesta en producción del par BTC/USD con la red recurrente

## 8.7– Red Convolucional - EUR/USD - Puesta en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el EUR/USD.
- El sistema acaba de ponerse en producción, lo que conlleva que acaba de realizar un entrenamiento con los últimos datos disponibles.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.5: Resultado de las predicciones de la red convolucional tras ponerse en producción con el par EUR/USD

En el gráfico de la figura 8.5 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red convolucional, al ponerse en producción, genera unas predicciones muy suavizadas. A simple vista se puede apreciar que la red está otorgando un buen resultado, asumiendo que se trate de un periodo en el que no hubiese cambios drásticos en el precio del par.

Tras verificarlo manualmente, se comprobó que en ese periodo hubo altibajos que la red convolucional no consiguió predecir.

En el cuadro 8.6 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.5, donde existen simultáneamente una predicción y un valor real.

Momento	<b>t0</b>	<b>t1</b>	<b>t2</b>	<b>t3</b>	<b>t4</b>
Error	$7,2 * 10^{-9}$	$5,8 * 10^{-9}$	$2,65 * 10^{-9}$	$9,4 * 10^{-9}$	$1,9 * 10^{-9}$
Momento	<b>t5</b>	<b>t6</b>	<b>t7</b>	<b>t8</b>	<b>t9</b>
Error	$1,6 * 10^{-9}$	$1,29 * 10^{-9}$	$5,7 * 10^{-10}$	$2,06 * 10^{-9}$	$5,83 * 10^{-10}$
Momento	<b>t10</b>	<b>t11</b>	<b>t12</b>	<b>t13</b>	<b>t14</b>
Error	$8,82 * 10^{-10}$	$7,42 * 10^{-11}$	$4,41 * 10^{-10}$	$1,31 * 10^{-8}$	$4,43 * 10^{-10}$

Cuadro 8.6: Error cuadrático asociado a las predicciones de la puesta en producción del par EUR/USD

## 8.8– Red Convolucional - EUR/USD - 24 horas en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el EUR/USD.
- El sistema lleva en producción 24 horas, por lo que se ha podido reentrenar con los datos que el mismo ha obtenido.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.6: Resultado de las predicciones de la red convolucional tras 24 horas en producción con el par EUR/USD

En el gráfico de la figura 8.6 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red convolucional, tras 24 horas en producción, obtiene unos resultados ligeramente similares a los resultados reales.

Esta red es capaz de ajustarse mejor con nuevos datos de este par gracias al re-entrenamiento.

En el cuadro 8.7 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.6, donde existen simultáneamente una predicción y un valor real.

Momento	<b>t0</b>	<b>t1</b>	<b>t2</b>	<b>t3</b>	<b>t4</b>
Error	$4,93 * 10^{-32}$	$9,94 * 10^{-10}$	$1,22 * 10^{-8}$	$2,5 * 10^{-9}$	$1 * 10^{-10}$
Momento	<b>t5</b>	<b>t6</b>	<b>t7</b>	<b>t8</b>	<b>t9</b>
Error	$1 * 10^{-10}$	$1,6 * 10^{-9}$	$3,6 * 10^{-9}$	$1 * 10^{-8}$	$1,21 * 10^{-8}$
Momento	<b>t10</b>	<b>t11</b>	<b>t12</b>	<b>t13</b>	<b>t14</b>
Error	$3,28 * 10^{-8}$	$1,52 * 10^{-10}$	$1,71 * 10^{-8}$	$3,98 * 10^{-8}$	$7,2 * 10^{-8}$

Cuadro 8.7: Error cuadrático asociado a las predicciones tras 24 horas en producción del par EUR/USD

## 8.9– Red Convolucional - BTC/USD - Puesta en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el BTC/USD.
- El sistema acaba de ponerse en producción, lo que conlleva que acaba de realizar un entrenamiento con los últimos datos disponibles.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.7: Resultado de las predicciones de la red convolucional tras ponerse en producción con el par BTC/USD

En el gráfico de la figura 8.7 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red convolucional, al ponerse en producción, en sus primeras predicciones no consigue adaptarse a la tendencia real del par.

Esta prueba concluye que la red convolucional no aporta mucho valor con un par de alta volatilidad como es el BTC.

En el cuadro 8.8 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.7, donde existen simultáneamente una predicción y un valor real.

Momento	t0	t1	t2	t3	t4
Error	6,3001	8,4681	0,0081	2,4964	0,8281
Momento	t5	t6	t7	t8	t9
Error	3,6481	7,3984	2,5281	10,8241	18,4041
Momento	t10	t11	t12	t13	t14
Error	0,1681	0,3481	13,1769	2,4964	4,9284

Cuadro 8.8: Error cuadrático asociado a las predicciones de la puesta en producción del par BTC/USD con la red convolucional

## 8.10– Red Convolucional - BTC/USD - 24 horas en producción

### Características de la prueba

- El par que se utiliza en esta prueba es el BTC/USD.
- El sistema lleva en producción 24 horas, por lo que se ha podido reentrenar con los datos que el mismo ha obtenido.
- No se tendrá en cuenta el resultado de la métrica del rendimiento del modelo, sino la precisión con respecto al resultado real.

### Resultado de la prueba



Figura 8.8: Resultado de las predicciones de la red convolucional tras 24 horas en producción con el par BTC/USD

En el gráfico de la figura 8.8 se muestran los resultados de forma visual de esta prueba, mostrándose en azul el histórico de valores de la divisa, en naranja los valores actuales y en verde las predicciones generadas por la red neuronal. La red convolucional, tras 24 horas en producción, obtiene unos resultados ligeramente similares a los resultados reales.

Esta red es capaz de ajustarse mejor con nuevos datos de este par gracias al re-entrenamiento.

En el cuadro 8.9 se muestran los errores cuadráticos para cada momento de la gráfica de la figura 8.8, donde existen simultáneamente una predicción y un valor real.

Momento	t0	t1	t2	t3	t4
Error	0,00	0,49	100,00	1,00	1,44
Momento	t5	t6	t7	t8	t9
Error	3,24	13,69	31,36	20,25	9,00
Momento	t10	t11	t12	t13	t14
Error	7,29	14,44	6,76	4,41	0,00

Cuadro 8.9: Error cuadrático asociado a las predicciones después de 24 horas de la puesta en producción del par BTC/USD con la red convolucional

## 8.11– Resultados globales

En este capítulo se han visto las pruebas más importantes de este proyecto, y pese a que cada prueba tiene una conclusión individual, se pretende dar una conclusión más general de ellas.

Este sistema se ha revisado exhaustivamente utilizando diferentes formas de evaluar el rendimiento para garantizar su correcto funcionamiento. Como se ha podido ver en las pruebas individuales, todas las pruebas de precisión del sistema van acompañadas de un gráfico que representa los datos obtenidos. Esto es debido a que es necesario ver la similitud de la curva de valores de forma visual para poder encontrar similitudes de forma sencilla. Además, dichos gráficos tienen unos ejes con valores muy concentrados, permitiendo así ver los errores.

Estos ejes funcionan agrupando los valores por una escala menor, como por ejemplo, 0.0001 en vez de 0.001. Así, si el valor real es 1.156, y la predicción de 1.155 se puede observar en la gráfica una desviación grande, mientras que en datos se observa que el cambio es mínimo.

Y aunque un error de 0.001 en algunos mercados bursátiles diferentes no supone importancia alguna, en el mercado del EUR/USD este valor es gigantesco. Un error acumulado así podría suponer pérdidas económicas de gran magnitud. En el par BTC/USD el error es relativo, debido a que los valores se han escalado entre cero y 1 para que la red pueda entrenar con facilidad, por ello encontramos valores de error en los entrenamiento. Además, en este caso un error se magnifica más en los resultados debido a este escalado, dado que al reescalar podemos acumular un error de cálculo.

Tras todas las pruebas, se ha concluido que el sistema funciona como es debido, pero las predicciones que obtiene no consiguen una calidad suficiente como para poder confiar un activo económico únicamente a esta fuente de datos. Tras consultarla con diferentes profesionales que se dedican a la inversión en bolsa, se ha concluido que un sistema de predicción o estrategia tiene que tener un error cuadrático medio muy pequeño, del orden del 0.00000001, para poder ser considerado fiable por los usuarios, y poder así gestionar activos bursátiles utilizando este sistema como única fuente de información.



# CAPÍTULO 9

---

## Conclusiones

---

Para concluir la memoria de este proyecto, se va a realizar un repaso de lo aprendido durante el proceso. El objetivo de este proyecto ha sido desarrollar una arquitectura software con Big Data para predecir valores del mercado de valores, aplicando técnicas de Deep Learning.

En la primera fase del desarrollo del proyecto se va a realizar un estudio y análisis de las diferentes técnicas Big Data, algoritmos de Deep Learning, estudio de la ciencia de datos, así como del mercado de valores.

Con el propósito de realizar un proyecto innovador y revolucionario, se ha desarrollado un algoritmo capaz de interpretar los valores de un mercado financiero y realizar predicciones en base a dichos valores. Además, se ha desarrollado todo un ecosistema software, apoyado con la herramienta software más potente del mercado para entornos Big Data. Para visualizar e interpretar los resultados, se ha realizado un panel de visualización con el que el usuario puede estudiar el mercado de forma sencilla e intuitiva.

Respecto al resultado del proyecto se puede concluir que, a pesar de la potencia con la que están dotadas las redes neuronales, entrenar los modelos únicamente con los valores del histórico de una divisa no hace posible que se generen unas predicciones con la precisión necesaria para confiar nuestros activos a un sistema que opere de manera autónoma.

Se han probado arquitecturas muy complejas durante la fase de desarrollo, algunas de ellas tardaban un día en entrenar el modelo por completo, pero el resultado es muy similar a la arquitectura finalmente propuesta, la cual se ha determinado como un equilibrio entre precisión y rendimiento.

Trabajar con tecnologías tan novedosas como TensorFlow o Spark fue costoso al inicio del proyecto. Debido a su curva de aprendizaje se ha necesitado mucho tiempo de investigación y práctica hasta conseguir el nivel de destreza necesaria para desenvolverse con dichas tecnologías. No obstante, ha sido una experiencia enriquecedora en la que hemos disfrutado bastante durante el aprendizaje y el desarrollo, que nos ha aportado mucho conocimiento que ayudará a seguir formándonos y a progresar en el campo de la inteligencia artificial y la ciencia del dato.



# CAPÍTULO 10

---

## Desarrollos futuros

---

Como se ha expuesto en las conclusiones, el nivel de precisión de las predicciones es demasiado bajo como para poder confiar plenamente en él, por lo que no puede utilizarse únicamente este modelo predictivo para realizar inversiones en Bolsa. Durante la investigación de este proyecto, hemos descubierto que existen múltiples proyectos relacionados con el mercado de valores que podrían integrarse con nuestro proyecto, en los que cabe destacar:

- Un modelo de Deep learning que obtenga información de noticias relacionadas con los valores, aplicando técnicas de **Procesamiento de Lenguaje Natural** o **PLN**. Utilizando este método se puede ver si una noticia es positiva, negativa o neutral, y, en función de ello, se podría corregir una predicción en tiempo real.
- El desarrollo de un **metamodelo** que realice predicciones de una secuencia de valores futuros pero de forma individual. En este proyecto se ha desarrollado un modelo que predice quince minutos a futuro, es decir, quince valores diferentes, pero se calculan todos a la vez mediante la salida secuencial de la red LSTM. El metamodelo consistiría en quince redes neuronales diferentes que predicen cada uno de esos valores. Puede parecer algo extraño de comprender, dado que a primera vista no parece tener sentido que una red que recibe los datos realice predicciones sobre un valor que no es su consecuente inmediato, pero en la práctica es posible. Este metamodelo requeriría una gran capacidad de cómputo para que pudiera ser ejecutado en tiempo real.
- En este proyecto se ha investigado el mercado financiero del Forex con intervalos de tiempo o ticks a nivel de minuto, pero también se podría investigar en **otros mercados financieros** como puede ser el S&P500, IBEX35, DOWJONES o DAX, para inversiones a medio y largo plazo. Las inversiones a corto plazo solo son viables en mercados de divisas.
- Un **agente automático** para invertir de forma autónoma en base a las predicciones obtenidas. Ya existen numerosos agentes automáticos en internet, incluso en github se pueden encontrar algunos muy potentes y famosos, pero todos ellos requieren para su funcionamiento una estrategia. Esta estrategia la introduce el usuario en forma de fórmula matemática. Con una ligera modificación en el código base se podría conseguir un agente que realice las operaciones directamente de la salida de la red neuronal.



---

## Glosario de términos

---

**Accuracy** Métrica para evaluar el rendimiento de un modelo de aprendizaje automático, se obtiene dividiendo el número de aciertos entre el total.

**Activo financiero** Título que representa para su poseedor derechos sobre bienes o rentas, y que es un pasivo para el agente que lo ha emitido.

**Algotrading** Técnica en la que se realizan operaciones sobre el mercado financiero automáticamente usando algoritmos.

**Análisis fundamental** Análisis que se realiza sobre un activo para determinar los factores que puede influir sobre su valor.

**Análisis técnico** Estudio de la acción del mercado.

**Aprendizaje no supervisado** Técnica de aprendizaje en la que se entrena un modelo sin datos de salida.

**Aprendizaje supervisado** Técnica de aprendizaje en la que se entrena un modelo con datos de salida.

**Batch** Número de ejemplos que se introducen a la red neuronal en cada época de entrenamiento.

**Big Data** Técnicas, herramientas y algoritmos que permiten trabajar sobre enormes volúmenes de datos.

**Clasificación** Tipo de problema de aprendizaje automático donde el objetivo es asignar una categoría a cada dato de entrada.

**Cluster** Conjunto de ordenadores que se comportan como si fuesen uno solo.

**Convolución** Operación matemática donde se combinan dos funciones en una sola donde el resultado representa la magnitud que se superpone a la primera función sobre una translación invertida de la segunda función.

**Data Science** Campo interdisciplinar que reúne las técnicas y conocimientos de los campos de la inteligencia artificial, el big data, las matemáticas y la estadística para extraer conocimiento de conjuntos de datos.

**Deep learning** Subcategoría dentro del aprendizaje automático que intenta modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no lineales múltiples.

**Descenso del gradiente** Algoritmo iterativo de optimización para encontrar el mínimo de una función. Para encontrar este mínimo, se calcula el gradiente en el punto actual y se avanza en la dirección negativa del gradiente calculado.

**Época** Fase de entrenamiento de una red neuronal. Una época se completa cuando se han introducido todos los datos del entrenamiento tras dividirlos en batches.

**Gradiente** Pendiente de la tangente en un punto de una función multivariable.

**IBEX 35** Índice bursátil de la bolsa española formado por las 35 empresas con más liquidez.

**Marca de tiempo** Secuencia que denota la hora y fecha en las que ocurre un determinado evento.

**Metamodelo** Modelo de un modelo. En machine learning se refiere a un modelo entrenado para generar otros modelos.

**MSE** El error cuadrático medio entre la estimación y el valor real.

**Operación intradía** Operación de venta o compra de activos que se lleva a cabo si el activo alcanza cierto valor durante el día.

**Perceptrón** Red neuronal que se compone de una única neurona.

**Procesamiento de Lenguaje Natural** Rama de conocimiento de la inteligencia artificial que estudia las interacciones entre las máquinas y el lenguaje humano.

**Ratio de aprendizaje** Parámetro asociado al entrenamiento de una red neuronal que indica la distancia de movimiento en la búsqueda del mínimo de la función.

**Red neuronal** Modelo computacional basado en la interconexión de neuronas imitando la estructura de los cerebros orgánicos.

**Redes neuronales convolucionales** Tipo de red neuronal en la que se realiza operaciones de convolución.

**Redes neuronales recurrentes** Tipo de red neuronal en las que existen ciclos entre las neuronas.

**Regresión** Tipo de problema de aprendizaje automático donde la variable de respuesta.

**Sobreajuste** Efecto producido al sobreentrenar un modelo por el cual no es capaz de generalizar correctamente.

**Serie temporal** Secuencia de datos ordenados cronológicamente.

**Standalone** Modo de funcionamiento de sistemas basados en cluster para operar con un único nodo.

**Software libre** Todo programa informático cuyo código fuente pueda ser modificado y utilizado libremente.

**Web scraping** Técnica para la extracción de información de páginas web simulando la navegación por ellas.



---

## Referencias

---

01. <https://blog.caixabank.es/blogcaixabank/2017/04/la-historia-de-la-bolsa-como-empezó-todo.html>. Fecha de consulta: 10 de Marzo de 2019.
02. <https://itnext.io/2017s-deep-learning-papers-on-investing-7489e8f59487>. Fecha de consulta: 10 de Marzo de 2019.
03. <https://spark.apache.org/powerd-by.html>. Fecha de consulta: 10 de Marzo de 2019.
04. <https://www.oracle.com/es/big-data/guide/what-is-big-data.html>. Fecha de consulta: 10 de Marzo de 2019.
05. <https://www.oracle.com/es/big-data/guide/what-is-big-data.html>. Fecha de consulta: 10 de Marzo de 2019.
06. <https://www.deltastock.com/spain/instruments/forex-list.asp>. Fecha de consulta: 10 de Marzo de 2019.
07. [https://en.wikipedia.org/wiki/Artificial\\_neural\\_network](https://en.wikipedia.org/wiki/Artificial_neural_network). Fecha de consulta: 10 de Marzo de 2019.
08. [https://es.wikipedia.org/wiki/Serie\\_temporal](https://es.wikipedia.org/wiki/Serie_temporal). Fecha de consulta: 10 de Marzo de 2019.
09. <http://www.redes-neuronales.com.es/tutorial-redes-neuronales/el-perceptron-multicapa.htm>. Fecha de consulta: 10 de Marzo de 2019.
10. <https://es.wikipedia.org/wiki/Convoluci%C3%B3n>. Fecha de consulta: 10 de Marzo de 2019.
11. <https://spark.apache.org/docs/latest/index.html>. Fecha de consulta: 10 de Marzo de 2019.
12. <https://spark.apache.org/docs/latest/sql-programming-guide.html>. Fecha de consulta: 10 de Marzo de 2019.
13. <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>. Fecha de consulta: 10 de Marzo de 2019.
14. <https://www.tensorflow.org/>. Fecha de consulta: 11 de Marzo de 2019.
15. <https://kafka.apache.org/documentation/>. Fecha de consulta: 18 de Marzo de 2019.
16. <https://dash.plot.ly>. Fecha de consulta: 18 de Marzo de 2019.