DEEP LEARNING METHODS FOR COMMODITY TRADING

Juan Mateo Pilapanta Coral

Trabajo de investigación 21/015

Master en Banca y Finanzas Cuantitativas

Tutores: Dr. Álvaro Montealegre

Universidad Complutense de Madrid

Universidad del País Vasco

Universidad de Valencia

Universidad de Castilla-La Mancha

www.finanzascuantitativas.com

VNIVERSITAT DE VALÈNCIA

Trabajo Fin de Máster - Curs 2020/ 2021

Deep Learning Methods for commodity trading

Autor/a: **Juan Mateo Pilapanta Coral**

Tutor/a: Álvaro Montealegre Moyano

Facultad de Economía

Máster en Banca y Finanzas Cuantitativas

Índice general

Capítulos **Página**

1. Introducción **8**
1.1. Motivación . . . 8
1.2. Objetivos . . . 9

2. Revisión de la literatura **11**

3. Datos **14**

4. Indicadores de análisis técnico **17**
4.1. Introducción . . . 17
4.2. Indicador técnico MACD . . . 18
4.3. Indicador técnico RSI . . . 19
4.4. Indicador técnico Bollinger Bands . . . 20

5. Machine Learning **23**
5.1. Conceptos básicos del aprendizaje automático . . . 23
5.1.1. Algoritmos de aprendizaje . . . 26
5.1.2. Algoritmos supervisados . . . 29
5.1.3. Algoritmos no-supervisados . . . 30
5.1.4. Contruyendo un algoritmo de aprendizaje automático . . . 32
5.2. Deep Learning . . . 34
5.2.1. Red Neuronal Prealimentada . . . 34
5.2.2. Regularización de los modelos . . . 36
5.2.3. Optimización del entrenamiento de los modelos . . . 40
5.2.4. LSTM Recurrent Neural Network . . . 43
5.2.5. Convolutional Neural Network . . . 46

6. Implementación de los modelos **53**
6.1. Ánalisis técnico . . . 53
6.2. LSTM Recurrent Neural Network . . . 56
6.3. CNN Neural Network . . . 60

7. Conclusiones **63**

Índice de figuras

3.1. Precio de cierre del Brent en el período 2000-2020 15

5.1. Ejemplo de diferentes representaciones: supongamos que queremos separar dos categorías de datos trazando una línea entre ellos en un diagrama de dispersión. En la figura de la izquierda, se utilizan las coordenadas cartesianas, y la tarea es imposible. En la figura a la derecha, representamos los datos con coordenadas polares y la tarea se vuelve simple para resolver con una línea vertical. 24
5.2. Ilustración de un modelo de Deep Learning. 25
5.3. Un diagrama de Venn que muestra cómo el aprendizaje profundo es un tipo de aprendizaje por representación, que es a su vez una especie de aprendizaje automático, que se utiliza para muchos pero no todos enfoques de la IA. Cada sección del diagrama de Venn incluye un ejemplo de tecnología de IA. 26
5.4. Ilustración de la representación de los datos, aprendida a trravés de PCA. 32
5.5. Ejemplo de una arquitectura MLP, con una capa oculta 35
5.6. Una ilustración del efecto de la regularización L^2 en el valor de la w óptima. 39
5.7. Izquierda: circuito de red recurrente ordinario con recurrencia de oculto a oculto, visto como un circuito, con matrices de pesos U, V, W para los tres tipos diferentes de conexiones (de entrada a oculto, de oculto a salida y de oculto a oculto, respectivamente). Cada círculo indica un vector completo de activaciones. Derecha: lo mismo visto como un gráfico de flujo en el que cada nodo está asociado a una instancia temporal concreta. 44
5.8. Ilustración del forzamiento del profesor para las RNN, que surge naturalmente del objetivo de entrenamiento de log-verosimilitud 45
5.9. Conectividad dispersa, vista desde abajo: Destacamos una unidad de entrada, X_3, y también resaltamos las unidades de salida en S que se ven afectadas por esta unidad. (Izquierda) Cuando S se forma por convolución con un núcleo de anchura 3, sólo tres salidas se ven afectadas por X_3. (Derecha) Cuando S se forma por multiplicación matricial, la conectividad ya no es escasa, por lo que todas las salidas se ven afectadas por X_3. 48

5.10. Compartición de parámetros: Destacamos las conexiones que utilizan un parámetro concreto en dos modelos diferentes. (Izquierda) Destacamos los usos del elemento central de un núcleo de 3 elementos en un modelo convolucional. Debido a la compartición de parámetros, este único parámetro se utiliza en todas las posiciones de entrada. (Derecha) Destacamos el uso del elemento central de la matriz de pesos en un modelo totalmente conectado. En este modelo no se comparten los parámetros, por lo que el parámetro se utiliza sólo una vez. 49
5.11. Las etapas de una capa típica de una red neuronal convolucional. 50

6.1. 54
6.2. 55
6.3. 55
6.4. Ejemplo de serie temporal dependiente multivariable. 57
6.5. Ejemplo de entrada de la primera muestra. 57
6.6. Ejemplo de salida de la primera muestra. 57
6.7. Ejemplo de salida de la división en muestras de una serie temporal para la previsión de varios pasos temporales. 58

7.1. Precios predichos bajo el modelo LSTM en el conjunto de entrenamiento. 64
7.2. Precios predichos bajo el modelo LSTM en el conjunto de test. 65
7.3. Precios predichos bajo el modelo CNN en el conjunto de entrenamiento. 66
7.4. Precios predichos bajo el modelo CNN en el conjunto de test. 67

Índice de cuadros

7.1. Resultados del modelo LSTM 63
7.2. Resultados del modelo CNN 65

Resumen

En este trabajo se pretende analizar una de las metodologías de Machine Learning y ver que aplicaciones puede tener al mundo de las finanzas. En concreto, se pretende profundizar en la metodología conocida como Deep Learning. Para ello se analizarán algunos de los métodos, los cuales consideremos que son los más adecuados y que más se utilicen en la industria, que podemos encontrar dentro de la metodología Deep Learning.
Una vez hayamos analizado los métodos elegidos pretendemos implementarlos en un entorno de programación, en este caso hemos elegido Python, y posteriormente construiremos y evaluaremos un caso práctico. El caso en cuestión será la predicción de los precios de commodities, concretamente trabajaremos con el precio del Brent. En nuestro modelo de predicción se utilizarán diferentes indicadores de análisis técnico que, aunque somos conscientes de que su uso no tiene una fundamentación estadística sólida la realidad es que en la práctica una gran parte de los traders hacen uso de ellos y ante la ausencia de shocks macroeconómicos los resultados que se obtienen al usarlos son, en general, bastante adecuados.
Para finalizar, compararemos los resultados de las predicciones obtenidas bajo los diferentes modelos y calcularemos cual ha sido el modelo que mayor beneficio nos aportaría.

Capítulo 1

Introducción

En los últimos años la inteligencia computacional en finanzas ha sido uno de los grandes tópicos tanto en el mundo académico como en el mundo de la industria. Dentro del campo de Machine Learning, Deep Learning ha empezado a captar una gran atención debido a que los resultados obtenidos bajo esta metodología han mejorado los que se obtenían al usar los modelos clásicos. Existe una gran cantidad de algoritmos y métodos de implementación de Deep Learning, es por ello que en este trabajo procederemos a estudiar algunos de ellos.
El aprendizaje profundo es un conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de alto nivel en datos usando arquitecturas computacionales que admiten transformaciones no lineales múltiples e iterativas de datos expresados en forma matricial o tensorial. Tras el concepto deep learning se encuentra un grupo de algoritmos que imita al cerebro humano 'aprendiendo' a reconocer patrones de repetición, palabras concretas, comportamientos frecuentes, etc. Estos algoritmos –definidos por la RAE como un "conjunto ordenado y finito de operaciones que permite hallar la solución de un problema"– trabajan 'imitando' la labor neuronal del cerebro.
La predicción de los mercados de valores, el trading algorítmico, la construcción de carteras ... son entre otras, las áreas donde los investigadores de Machine Learning han centrado su atención y donde se han desarrollado modelos que pueden proporcionar soluciones en tiempo real para la industria financiera.
Como habíamos dicho anteriormente, dentro de Machine Learning, Deep Learning se ha convertido en un área emergente que cada vez despierta más interés. Como resultado, un gran número de modelos para finanzas, basados en esta metodología, han comenzado a aparecer en conferencias y revistas científicas. En este trabajo nosotros nos centraremos en el estudio de dos modelos, los cuales consideramos que son algunos de los más importantes y más implementados en el área de las finanzas. Estos modelos son: LSTM Recurrent Neural Network y CNN Neural Network, dos modelos que pertenecen al conjunto de modelos de aprendizaje profundo pero que abordan la solución al problema de dos maneras distinta, en los capítulos siguientes desarrollaremos estos dos modelos en profundidad.

1.1. Motivación

Como ya hemos explicado anteriormente, las técnicas de Machine Learning, en especial Deep Learning, cada vez se usan más en nuestra industria. Es por ello que hemos decidido realizar este trabajo, ya que tanto en el aspecto teórico, es decir, el aprender nuevos modelos estadísticos, como el potencial

práctico que tienen consideramos que pueden ser un gran complemento en nuestro aprendizaje.
Consideramos que la base que se nos ha proporcionado a lo largo del máster es una base lo suficientemente extensa como para poder trabajar y comprender los modelos y sus implementaciones. Si bien es cierto, el plan académico de nuestro máster abarca una amplia gama de conocimientos. Por ejemplo, tenemos un gran conocimiento en áreas como pueden ser los productos derivados, instrumentos de renta fija, medición de riesgos o modelos econométricos. Además de todos esos conocimientos, que se han visto en profundidad a lo largo de estos dos años de aprendizaje, también hemos recibido una introducción al mundo del aprendizaje automático en uno de los seminarios del máster.
Al recibir este seminario y empezar a investigar sobre este conjunto de técnicas de Machine Learning nos dimos cuenta de que es un campo que aún puede ofrecer mucho más de lo que se ha obtenido hasta ahora. Esta es una de las razones por la cual a día de hoy muchos investigadores centran la atención de sus trabajos en Machine Learning, del mismo modo que nosotros hemos decidido hacer con nuestra tesis.

1.2. Objetivos

Existen dos objetivos principales en este trabajo. Como hemos comentado anteriormente, en este máster se nos ha formado ampliamente en diversos ámbitos, pero no especialmente en Machine Learning. Así pues, el primer objetivo de este trabajo es el de formarnos en este nuevo conjunto de técnicas. En Machine Learning existe una gran variedad de metodologías, pero nosotros hemos decidido centrarnos en Deep Learning. Por lo tanto, el objetivo será comprender en que se fundamenta el Deep Learning, cuáles son sus predecesores, que tipos de modelos se pueden aplicar a finanzas y profundizar en dichos modelos para poder implementarlos en un caso práctico.
El otro gran objetivo de este trabajo viene ligado al primer objetivo, como acabamos de decir una vez conozcamos y comprendamos la fundamentación teórica que respalda a los modelos pretendemos implementarlos. Luego, el segundo objetivo será aprender las librerías necesarias y el entorno de programación necesario para la implementación de dichos modelos. Nosotros hemos elegido Python pues es uno de los entornos de programación que más se usa en la industria, y además es el entorno en el que más se está desarrollando la inteligencia artificial.

Capítulo 2

Revisión de la literatura

Machine Learning ha sido una de las técnicas más novedosas y que más atención ha captado en los últimos años. En este capítulo pretendemos hacer un breve repaso de en qué campos y para qué aplicaciones en concreto se ha usado o se está usando el Deep Learning. Además, haremos una revisión más específica para el campo del mundo de las finanzas.

Una de las aplicaciones más populares del Deep Learning es la de los asistentes virtuales como por ejemplo Alexa, Siri o Google Assistant. Cada interacción que tenemos con nuestro asistente de voz les proporciona una oportunidad para aprender más sobre nuestra voz y nuestro acento. Los asistentes virtuales usan Deep Learning para saber más sobre nosotros empezando desde nuestras preferencias en las cenas fuera de casa, nuestras páginas web favoritas o nuestras canciones preferidas. Otra capacidad de los asistentes virtuales es la de traducir nuestros discursos en texto. Con aplicaciones de Deep Learning, como generación de texto y resúmenes de documentos, los asistentes virtuales también pueden ayudar a crear o enviar una copia de correo electrónico adecuada.

Otra de las múltiples aplicaciones es la detección de fraude. Implementaciones de Deep Learning han sido desarrolladas para detectar los fraudes con las tarjetas de crédito y así ahorrar grandes cantidades de dinero en los costes de seguros para las instituciones financieras. La prevención y detección de fraude se basan en la identificación de patrones en las transacciones de los clientes, identificando comportamientos anómalos y valores atípicos.

La coloración de imágenes en blanco y negro es otra de las aplicaciones más usadas en Deep Learning. Este proceso se trata de tomar imágenes en blanco y negro (como input) y producir imágenes en color (como output). Este proceso, normalmente era realizado a mano por el ser humano. Sin embargo, con la tecnología Deep Learning de hoy, este proceso se realiza de forma automática utilizando implementaciones de los algoritmos. En esencia, esta aproximación envuelve el uso de redes neuronales convolucionales en capas supervisadas que recrean la imagen con el color pertinente.

Hemos nombrado tres de las aplicaciones más usadas y más conocidas, pero existen muchas más como pueden ser: la conducción automática de coches, la medicina, el reconocimiento facial, las predicciones electorales ... Por otra parte, vamos a hacer un repaso de las aplicaciones que ha tenido el Deep Learning en nuestro campo, el mundo financiero. En la literatura existen muchas aplicaciones financieras, pero nosotros vamos a centrar nuestra revisión sobre los trabajos que se han realizado en el campo del trading algorítmico.

Trading Algorítmico

El trading algorítmico se define como el conjunto de decisiones de compra-venta tomadas únicamente basándose en modelos algorítmicos. Estas decisiones pueden estar basadas en reglas simples, modelos matemáticos, o como en nuestro caso en machine/deep learning. Con la introducción de las plataformas de trading electrónico online el trading algorítmico ganó una gran importancia en la industria financiera. Como resultado de ello, modelos de trading algorítmico basados en Deep Learning también han empezado a atraer la atención de los investigadores.

La mayoría de estudios de trading algorítmico se concentran en la predicción de precios de acciones o índices bursátiles. Así, LSTM (un tipo diferente de red de aprendizaje profundo destinada específicamente al análisis de datos secuenciales. La ventaja de las redes LSTM reside en el hecho de que tanto los valores a corto plazo como a largo plazo en la red pueden ser recordados) ha sido el modelo Deep Learning más utilizado en estas implementaciones. Bao et al. [1] usan indicadores de análisis técnico como input en los modelos WT (método que transforma una función en un conjunto de ondículas, una ondícula es una oscilación ondulatoria localizada en el tiempo), LSTM y SAEs (red neuronal que consta de varias capas de autocodificadores dispersos en las que la salida de cada capa oculta se conecta a la entrada de la capa oculta sucesiva) para la predicción de precios de acciones. En [2], los modelos CNN (tipo de red neuronal basada en la arquitectura de pesos compartidos de los núcleos o filtros de convolución que se deslizan a lo largo de las características de entrada y proporcionan respuestas equivariantes de traducción conocidas como mapas de características) y LSTM son implementados conjuntamente (CNN se usa para la selección de acciones, LSTM se usa para la predicción de los precios).

Para la predicción de índices bursátiles, podemos encontrar los siguientes artículos. En [3], la predicción del precio del índice S&P500 se realiza mediante la implementación de un modelo LSTM. Young et al. [4] utilizó el método DNN (red neuronal artificial (RNA) con múltiples capas entre la de entrada y la de salida. Hay diferentes tipos de redes neuronales, pero siempre constan de los mismos componentes: neuronas, sinapsis, pesos, sesgos y funciones) feed-forward y Open, Close, High, Low (OCHL) de los datos de la serie, para predecir los datos del índice del mercado de valores de Singapur.

Capítulo 3

Datos

Como habíamos dicho en los apartados anteriores una vez hayamos analizado los modelos seleccionados y los hayamos implementado en Python pretendemos aplicarlos a un caso concreto. Concretamente nosotros vamos a trabajar con los precios del Brent.
El Brent es un tipo de petróleo que se extrae principalmente del mar del Norte. En cuanto a las características del brent destacaremos que es un petróleo liviano, que resulta ser ideal para la producción de gasolina. Suele ser refinado en los países de Europa Noroccidental.
Con lo que respecta a la cotización del brent decir que el barril (42 galones estadounidenses o lo que es lo mismo unos 159 litros) originalmente se negociaba en la Bolsa Internacional del Petróleo de Londres desde 1988, pero desde 2005 se negocia en el Intercontinental Exchange electrónico, conocido como ICE. Destacar el hecho de que a principios de 2020 el petróleo tuvo una gran caida en su cotización, este hecho fue causado por la pandemia mundial del coronavirus y por la Guerra de precios del petróleo entre Rusia y Arabia Saudita.
Como habíamos dicho antes el brent cotiza en el ICE así que ahora veremos algunas de las especificaciones más relevantes del contrato de brent en este mercado. En primer lugar, el tamaño de contrato es de 1000 barriles y precio de negociación de 0.01$ por barril.
En segundo lugar, ICE Clear Europe actúa como contraparte central de las operaciones realizadas en las bolsas de Londres. Esto le permite garantizar la ejecución financiera de todos los contratos registrados en ella por sus miembros (los miembros compensadores de las bolsas) hasta la entrega, el ejercicio y/o la liquidación. ICE Clear Europe no tiene ninguna obligación ni relación contractual con los clientes de sus miembros que no son usuarios de los mercados bursátiles, ni miembros no compensadores de las bolsas.
En cuanto al límite de posición destacar que el futuro del crudo Brent es un contrato que se liquida en efectivo. El régimen de gestión de posiciones diarias de la Bolsa exige que todas las posiciones de cualquier mes de contrato se comuniquen a la Bolsa diariamente. La Bolsa tiene poderes para prevenir el desarrollo de posiciones excesivas o especulación injustificada o cualquier otra situación indeseable y puede tomar cualquier medida necesaria para resolver tales situaciones, incluyendo la capacidad de ordenar a los miembros que limiten el tamaño de tales posiciones o que reduzcan las posiciones cuando sea apropiado.
Por último, decir que con lo que respecta a las condiciones de entrega/liquidación el contrato de futuros sobre el Brent Crudo ICE es un contrato entregable basado en la entrega intercambio de futuros

por físico (EFP) con una opción de liquidación en efectivo contra el precio del Índice Brent ICE para el último día de negociación del contrato de futuros. La Bolsa publicará un precio de liquidación en efectivo (el precio del Índice ICE Brent) el día de negociación siguiente al último día de negociación del mes del contrato.
A continuación, vamos a adjuntar un gráfico en el que podremos observar la evolución del precio de cierre del Brent.

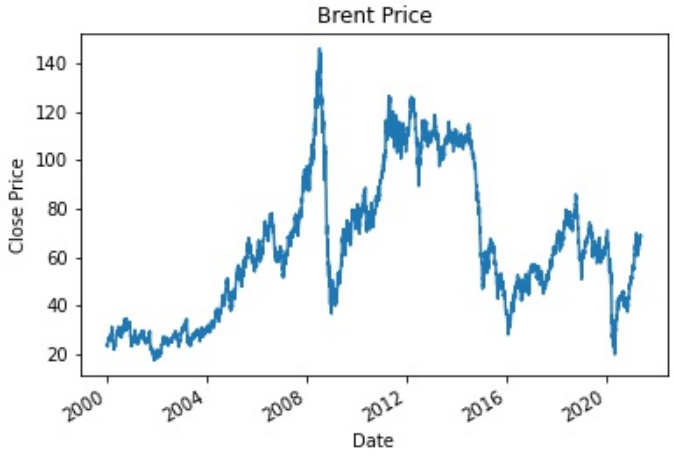

Figura 3.1: Precio de cierre del Brent en el período 2000-2020

Así pues, para nuestro ejercicio práctico tomaremos una submuestra de esta serie temporal y construiremos nuestros modelos basándose en dicha submuestra. Posteriormente haremos una predicción de los precios fuera de la muestra y su respectiva evaluación.

Capítulo 4

Indicadores de análisis técnico

En este capítulo se pretende hacer una introducción al entorno de los indicadores de análisis técnico. Luego introducir y explicar los indicadores que se emplearán en el caso práctico final. Además, pretendemos implementar una estrategia de compra-venta solo basándonos en los indicadores de análisis técnico.

4.1. Introducción

Antes de empezar con el estudio de las técnicas y herramientas utilizadas en el análisis técnico, es necesario que definamos que es el análisis técnico, que discutamos las premisas en las que se basa, establezcamos algunas diferencias claras entre el análisis técnico y el análisis fundamental y finalmente hablaremos sobre algunas de las críticas más frecuentes en contra del enfoque técnico.
En primer lugar, vamos a definir el tema. *El análisis técnico es el estudio de la acción de mercado, primordialmente basado en el uso de gráficos, con la intención de predecir las futuras tendencias de los precios.* El término "acción de mercado" incluye las tres principales fuentes de información disponible para el analista, es decir, volumen, precio e interés abierto.
Por otra parte, decir que el enfoque técnico se basa en las siguientes premisas:

- Las acciones de mercado descuentan todo.
- Los precios se mueven según las tendencias.
- La história se repite periódicamente

Ahora veamos algunas de las diferencias entre el análisis técnico y el análisis fundamental. Mientras el análisis técnico se concentra en el estudio de las acciones de mercado, el análisis fundamental se centra en que la oferta y la demanda causa que los precios se muevan al alza, la baja o que se mantengan. El enfoque fundamental examina todos los factores relevantes que afectan a los precios de mercado para determinar el valor intrínseco del mercado. Si el valor intrínseco está por debajo del precio actual de mercado, entonces el mercado estará sobrevalorado y deberíamos vender. Si el precio de mercado está por debajo del valor intrínseco, entonces el mercado estará infravalorado y deberíamos comprar.
Ambos enfoques de predicción intentan resolver el mismo problema, este es, determinar la dirección en la que se pueden mover los precios. Dichos enfoques aproximan el problema desde distintos puntos de

vista. El fundamentalista estudia la causa del movimiento de mercado, mientras que el técnico estudia el efecto. El técnico, por supuesto, cree que el efecto es todo lo que necesita conocer y las razones son innecesarias. El fundamentalista siempre tiene que conocer el porqué.
Si se aceptan las premisas en las que se basa el análisis técnico, se puede entender porque los técnicos creen que su enfoque es superior a los de los fundamentalistas. Si el trader tiene que elegir solo uno de los dos enfoques, la elección debería ser el enfoque técnico. Porque, por definición, el enfoque técnico incluye el fundamental. Si los fundamentos están reflejados en el precio de mercado, entonces el estudio de esos fundamentos se convierte en innecesario. El estudio de los gráficos se convierte en un atajo del análisis fundamental. El recíproco no es cierto. El análisis fundamental no incluye el estudio de las acciones del precio. Es posible hacer transacciones en los mercados financieros solo utilizando el enfoque técnico. En cambio, es inseguro realizar transacciones solo utilizando el enfoque fundamental sin ninguna consideración de la parte técnica del mercado.
Existe una gran variedad de indicadores de análisis técnico. Éstos pueden agruparse en los siguientes grupos: indicadores de momento, indicadores de volumen, indicadores de volatilidad, indicadores de tendencia, y otros indicadores. A continuación, vamos a introducir los indicadores técnicos con los que trabajaremos a lo largo del trabajo. Además, en cada uno de ellos, como ya habíamos dicho anteriormente, propondremos una estrategia de inversión. Por último, implementaremos dichas estrategias y presentaremos los resultados obtenidos.

4.2. Indicador técnico MACD

Empezaremos introduciendo el Moving Average Convergence Divergence (MACD) indicador. El indicador MACD es uno de los indicadores técnicos de oscilación más populares.
MACD nos ayuda a comprender la relación entre las medias móviles. Convergencia es cuando las líneas se mueven cerca una de otras y divergencia es cuando las líneas se mueven lejos unas de otras. Cuando hablamos de líneas nos referimos a las medias móviles.
MACD es un indicador de momento que sigue la tendencia. Puede ayudarnos a evaluar la relación entre dos medias móviles de precios. Posteriormente, el indicador MACD se puede utilizar para calcular una estrategia comercial que nos indica cuándo comprar o vender una acción.
Una media móvil es un valor de media móvil de un período histórico predefinido. Por ejemplo, el promedio móvil simple de 10 días se calcula calculando el promedio del período de los últimos 10 días. La media móvil exponencial, por otro lado, asigna mayor importancia a los valores recientes. Puede ayudarnos a captar mejor los movimientos del precio de una acción.
A continuación, vamos a presentar los 3 pasos principales para calcular el MACD:
Paso 1: Calcular la línea MACD:

1. Calcular el promedio móvil ponderado exponencialmente (EMA) de 26 días del precio. Esta es la línea a largo plazo. La fórmula necesaria para el cálculo de EMA es la siguiente:

$$EMA_{hoy} = (Valor_{hoy} \frac{Smoothing}{1 + Dias}) + EMA_{ayer}(1 - \frac{Smoothing}{1 + Dias})$$

   Normalmente el valor del $Smoothing$ es igual a 2, nosotros tomaremos ese valor.

2. Calcular el promedio móvil ponderado exponencialmente de 12 días del precio. Esta es la línea a corto plazo.

3. Calcular la diferencia entre la EMA de 26 días y las líneas de la EMA de 12 días. Esta es la línea MACD. Luego tenemos que:

$$MACD = EMA_{12 \text{ días}} - EMA_{26 \text{ días}}$$

Paso 2: Calcular la línea Señal de la línea MACD:

1. Calcular el promedio móvil ponderado exponencialmente de 9 días de la línea MACD. Esto se conoce como línea de señal.

Paso 3: Calcular el histograma: distancia entre el MACD y la Señal:

1. Luego podemos calcular la diferencia entre el MACD y la línea de señal y luego trazarla como un histograma. El histograma puede ayudarnos a determinar cuándo está a punto de ocurrir el cruce.

La longitud del histograma se puede utilizar para comprender mejor la tendencia. Cuando las barras del histograma no aumentan, puede implicar que los precios no son volátiles y pronto podría ocurrir un gran movimiento en la dirección opuesta.
Aunque el enfoque habitual es utilizar los parámetros como se describe anteriormente, realmente depende de las acciones, el mercado y el inversor. Podemos elegir diferentes parámetros y optimizar los parámetros que mejor se ajusten a nuestro estilo de negociación y al mercado que nos interese.
Estrategia:
Podemos utilizar el cruce entre MACD y la línea de señal para crear una estrategia comercial simple. Aquí es donde la línea MACD y la línea de señal se cruzan.

- Señal de venta: el cruce se produce cuando la línea MACD está por debajo de la línea Señal.
- Señal de compra: el cruce se produce cuando la línea MACD está por encima de la línea Señal.

Alcista vs Bajista:

- Bajista: cuando las líneas MACD y la Señal están por debajo de 0 entonces el mercado es bajista.
- Alcista: cuando las líneas MACD y la Señal están por debajo de 0 entonces el mercado es alcista.

4.3. Indicador técnico RSI

Ahora vamos a analizar el indicador RSI. RSI hace referencia a Relative Strenght Index. Es un indicador técnico muy utilizado y esto se debe principalmente a su simplicidad. Se basa en el mercado y podemos usar el indicador para determinar cuándo comprar o vender una acción.
RSI requiere el cálculo de las ganancias y pérdidas recientes. El período de tiempo reciente se puede especificar de manera subjetiva. Usamos el indicador RSI para medir la velocidad y el cambio de los movimientos de precios.
RSI es un indicador oscilante. Puede ayudarnos a comprender mejor el momento. Hay que tener en cuenta que el momento es el cambio de precio y tamaño. Por lo tanto, el indicador RSI puede ayudarnos a comprender cuándo el precio de las acciones cambiará su tendencia.
La clave para usar este indicador es comprender si una acción está sobrecomprada o sobrevendida.
Cálculo:
El cálculo es relativamente simple.

1. En primer lugar, tenemos que determinar el período de tiempo. Por lo general, se elige un período de 14 días, pero podría depender de la visión que tenga el inversor sobre el mercado y las acciones.

2. En segundo lugar, tenemos que calcular la fuerza relativa que se conoce como RS. RS es la ganancia promedio sobre la pérdida promedio. Para explicarlo con más detalle, RS es la ganancia promedio cuando el precio ha subido por encima de la pérdida promedio, cuando el cambio de precio ha sido negativo. La fórmula para el cálculo de RS es la siguiente:

$$RS = \frac{AvgU}{AvgD}$$

   donde,

   $AvgU =$ suma de todos los movimientos al alza (U) en los últimos N compases dividida por N
   $AvgD =$ suma de todos los movimientos a la baja (D) en los últimos N compases dividida por N

3. Luego calcularemos RSI de la siguiente forma

$$RSI = 100 - \frac{100}{1 + RS}$$

Notar que el valor que obtengamos para el indicador RSI oscilará entre 0 y 100.
Estrategia:
Sobrecompra: cuando el RSI está por encima del 70 %. Básicamente, la sobrecompra se produce cuando el precio de una acción ha aumentado rápidamente durante un breve período de tiempo, lo que implica que este sobrecomprada.
El precio de una acción sobrecomprada suele bajar de precio.
Sobreventa: cuando el RSI está por debajo del 30 %. Básicamente, la sobreventa es cuando el precio de una acción ha disminuido rápidamente durante un breve período de tiempo, lo que implica que este sobrevendida.
El precio de una acción sobrevendida generalmente aumentará de precio.
Existe una gran variedad de estrategias que dependen del indicador RSI. Una estrategia simple es utilizar el RSI de manera que:

- Vender: cuando el RSI incrementa por encima del 70 %.
- Comprar: cuando el RSI disminuye por debajo del 30 %.

Podríamos decidir utilizar diferentes parámetros. La cuestión es que podemos optimizar los parámetros que mejor se ajusten a nuestro estilo de negociación, el mercado y las acciones que nos interesen.

4.4. Indicador técnico Bollinger Bands

Por último, vamos a introducir el indicador técnico Bollinger Bands. Una vez más, es uno de los indicadores técnicos más populares. Y esto se debe principalmente a su sencillez.
Hay dos componentes principales en este indicador:

- Volatilidad.
- Medias móviles.

Principalmente, los pasos para calcular este indicador son los siguientes:

1. Banda media: calculamos la media móvil del precio, normalmente media móvil de 20 días.
2. Banda superior: calculamos dos desviaciones estándar por encima de la media móvil.
3. Banda inferior: calcula dos desviaciones estándar por debajo de la media móvil.

Cuanto más volátiles sean los precios de las acciones, más anchas serán las bandas de la media móvil. Es importante observar la forma/tendencia de las bandas junto con la brecha entre ellas para comprender mejor la tendencia y el mercado.
Las desviaciones estándar capturan los movimientos volátiles y, por lo tanto, calculamos las desviaciones estándar por encima y por debajo de las bandas superior e inferior para capturar los valores atípicos. En consecuencia, el 95 % de los movimientos de precios caerán entre las dos desviaciones estándar.
Estrategia:
Una estrategia simple podría ser:
Vender: Tan pronto como el precio de mercado toque la banda superior de Bollinger.
Compra: tan pronto como el precio de mercado toque la banda inferior de Bollinger.
Esto se basa en el supuesto de que la acción debe retroceder (desde la tendencia alcista) y finalmente tocar la banda inferior.
A veces, el indicador de banda de Bollinger nos indica que compremos una acción, pero un evento externo del mercado, como una noticia negativa, puede cambiar el precio de la acción. Por lo tanto, es importante utilizar el indicador como un indicador que a veces puede ser incorrecto.

Capítulo 5

Machine Learning

En este capítulo vamos a desarrollar toda la teoría en la que se basarán los métodos que implementaremos en nuestro caso práctico. Este capítulo es sin lugar a duda el más importante y por ello es el que más desarrollaremos. No obstante, el área de Machine Learning es muy extensa y no podremos ver ni todas las técnicas ni todos los modelos que existen en la actualidad. Por ello lo que pretendemos hacer en este capítulo es una intrducción general en la que repasaremos los puntos más importantes del aprendizaje automático. A continuación, nos centraremos en la técnica de Machine Learning elegida, Deep Learning, y explicaremos en profundidad los modelos elegidos.

5.1. Conceptos básicos del aprendizaje automático

Los inventores han soñado durante mucho tiempo con crear máquinas que piensen. En la antigua Grecia los mitos hablaban de objetos inteligentes, como estatuas animadas de seres humanos o mesas que llegen llenas de comida y bebida cuando se les llame.
Hoy en día, la inteligencia artificial (IA) es un campo próspero con muchas aplicaciones prácticas y temas de investigación activos. Buscamos software inteligente para automatizar trabajos rutinarios, comprender el habla o las imágenes, hacer diagnósticos médicos y apoyar investigaciones científicas.
En los primeros días de la inteligencia artificial, el campo abordó y resolvió rápidamente problemas que son intelectualmente difíciles para los seres humanos pero relativamente sencillos para los ordenadores -problemas que pueden describirse mediante una lista de reglas formales. El verdadero desafío para la inteligencia artificial es resolver aquellas tareas que son fáciles de realizar para las personas pero difíciles de describir formalmente, aquellos problemas que resolvemos intuitivamente.
En este trabajo estudiaremos una de las múltiples soluciones a estos problemas. Esta solución se basa en permitir que los ordenadores aprendan de la experiencia y comprendan el mundo en términos de una jerarquía de conceptos, con cada concepto definido en términos de su relación con otros conceptos más simples. Al recopilar conocimientos a través de la experiencia, se evita la necesidad de que los humanos especifiquen formalmente todos los conocimientos que la máquina necesita. La jerarquía de conceptos permite que la máquina aprenda conceptos complicados construyéndolos a partir de conceptos más simples. Si dibujasemos una gráfica mostrando cómo estos conceptos se construyen uno encima del otro, el gráfico tendría cierta profundidad, con muchas capas. Es por ello que este enfoque se conoce como aprendizaje profundo, en inglés Deep Learning.

Las dificultades a las que enfrentan los sistemas que se basan en conocimientos codificados sugieren que los sistemas de inteligencia artificial necesitan la capacidad de adquirir su propio conocimiento mediante la extracción de patrones a partir de los datos originales. Esta capacidad se conoce como aprendizaje automático. La introducción del aprendizaje automático permitió a los ordenadores abordar problemas relacionados con el conocimiento del mundo real y tomar decisiones que parecen subjetivas. Un simple algoritmo de aprendizaje automático llamado regresión logística puede determinar cuando recomendar un parto por cesárea [5]. Un algoritmo de aprendizaje automático llamado Bayes puede separar el correo electrónico legítimo del correo no deseado.
El rendimiento de estos simples algoritmos de aprendizaje automático depende en gran medida de la representación que se les de a los datos. Esta dependencia a las representaciones es un fenómeno general que aparece tanto en la informática como incluso en la vida diaria. En informática, operaciones como buscar en una colección de datos puede resolverse exponencialmente más rápido si la colección está estructurada e indexada de forma inteligente. Las personas pueden realizar aritmética fácilmente en números arábigos, pero encontrar una aritmética en números romanos supone invertir mucho más tiempo. Por lo tanto, no es de extrañar que la elección de la representación tenga un enorme efecto sobre el rendimiento de los algoritmos de aprendizaje automático. Para una simple visualización observar la Fig. 5.1.

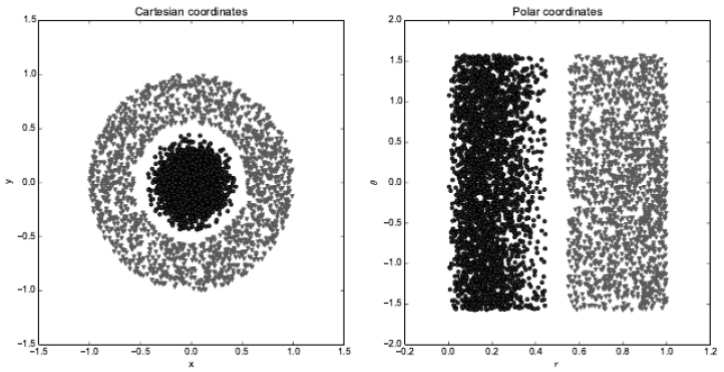

Figura 5.1: Ejemplo de diferentes representaciones: supongamos que queremos separar dos categorías de datos trazando una línea entre ellos en un diagrama de dispersión. En la figura de la izquierda, se utilizan las coordenadas cartesianas, y la tarea es imposible. En la figura a la derecha, representamos los datos con coordenadas polares y la tarea se vuelve simple para resolver con una línea vertical.

Una solución a este problema es utilizar el aprendizaje automático para descubrir no solo el mapeo de la representación de la salida, sino también la propia representación. Este enfoque se conoce como aprendizaje de representación. Las representaciones de aprendizaje normalmente dan como resultado un rendimiento mucho mejor al que se puede obtener con representaciones hechas manualmente. También permiten que los sistemas de IA se adapten rápidamente a nuevas tareas, con una mínima intervención humana. Un algoritmo de aprendizaje de representación puede descubrir un buen conjunto de características para una tarea simple en minutos, o una tarea compleja en horas o meses. Mientras que el diseño manual de características para una tarea compleja requiere una gran cantidad de tiempo y esfuerzo humano; el cúal puede llevar décadas para toda una comunidad de investigadores.

Al diseñar funciones o algoritmos para funciones de aprendizaje, nuestro objetivo suele ser separar los factores de variación que explican los datos observados. En este contexto, usamos la palabra "factores"simplemente para referirnos a fuentes de influencia separadas. Estos factores a menudo no son cantidades que se observen directamente pero pueden existir objetos o fuerzas en el mundo físico que afecten a las cantidades observables, o son construcciones en la mente humana que proporcionan explicaciones simplificadoras útiles o causas inferidas de los datos observados. Pueden considerarse conceptos o abstracciones que nos ayudan a dar sentido a la rica variabilidad de los datos. Por supuesto, puede ser muy difícil extraer características abstractas de tan alto nivel a partir de los datos originales.
Deep Learning resuelve este problema en el aprendizaje de representación al introducir representaciones que se expresan en términos de otras representaciones más simples. El aprendizaje profundo permite a la máquina construir conceptos complejos a partir de conceptos más simples. La Fig 5.2 muestra cómo un sistema de aprendizaje profundo puede representar el concepto de la imagen de una persona mediante la combinación de conceptos más simples, como esquinas y contornos, que a su vez se definen en términos de bordes.

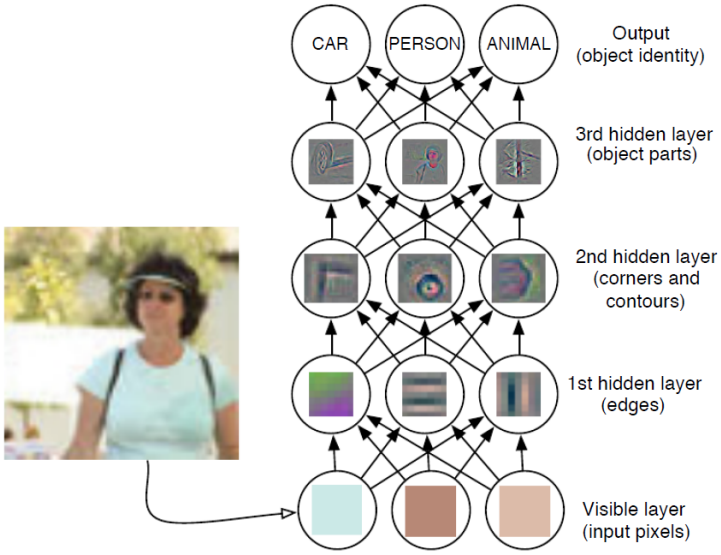

Figura 5.2: Ilustración de un modelo de Deep Learning.

En resumen, el aprendizaje profundo, el tema de esta tesis, es un enfoque de la IA. Específicamente, es un tipo de aprendizaje automático, una técnica que permite al ordenador trabajar con sistemas que mejoran a partir de la experiencia y de los datos. Según algunos autores, Deep Learning es el único enfoque viable para crear sistemas de IA que operen en entornos complicados del mundo real. El aprendizaje profundo es un tipo particular de aprendizaje automático que logra un gran poder y flexibilidad al aprender a representar el mundo real como una jerarquía anidada de conceptos y representaciones, con cada concepto definido en relación con conceptos más simples y más abstractos; y las representaciones calculadas en términos de otras menos abstractas. La Fig 1.4 ilustra la relación entre estas diferentes disciplinas de IA.

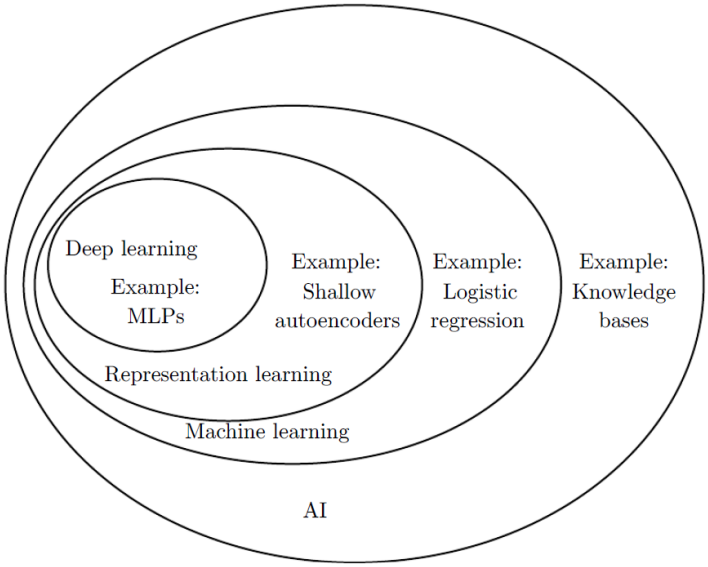

Figura 5.3: Un diagrama de Venn que muestra cómo el aprendizaje profundo es un tipo de aprendizaje por representación, que es a su vez una especie de aprendizaje automático, que se utiliza para muchos pero no todos enfoques de la IA. Cada sección del diagrama de Venn incluye un ejemplo de tecnología de IA.

5.1.1. Algoritmos de aprendizaje

Un algoritmo de aprendizaje automático es un algoritmo que es capaz de aprender a partir de los datos. Pero ¿qué entendemos por aprendizaje? En el ámbito de la informática una definición de aprendizaje es: "Se dice que un programa informático aprende de la experiencia E con respecto a una clase de tareas T y una medida de rendimiento P, si su rendimiento en las tareas de T, medido por P, mejora con la experiencia E"[6]. A continuación, se ofrecen descripciones intuitivas y ejemplos de los diferentes tipos de tareas, medidas de rendimiento y experiencias que pueden utilizarse para construir algoritmos de aprendizaje automático.

Las tareas, T

El aprendizaje automático es interesante sobre todo por las tareas que podemos realizar con él. En esta definición relativamente formal de la palabra "tarea", el proceso de aprendizaje en sí mismo no es la tarea.

El aprendizaje automático permite resolver muchos tipos de tareas. Algunas de las tareas de aprendizaje automático más comunes son las siguientes:

- *Clasificación*: En este tipo de tarea, se pide al programa informático que especifique a cuál de las k categorías pertenece una entrada. Para resolver esta tarea, el algoritmo de aprendizaje suele pedirle que produzca una función $f : R^n \longrightarrow \{1, \cdots, k\}$ que puede aplicarse a cualquier entrada. En este caso, la salida de $f(x)$ puede interpretarse como una estimación de la categoría a la que pertenece x. Existen otras variantes de la tarea de clasificación, por ejemplo, en las que f produce una distribución de probabilidad sobre las clases. Un ejemplo de tarea de clasificación es el reconocimiento de objetos, en la que la entrada es una imagen y la salida es un código

numérico que identifica el objeto en la imagen.

- *Regresión* : En este tipo de tarea, se pide al programa informático que prediga un valor numérico dado un dato de entrada. Para resolver esta tarea, al algoritmo de aprendizaje se le pide que emita una función $f : R^n \longrightarrow R$. Este tipo de tarea es similar al de clasificación, salvo que el formato de la salida es diferente. Un ejemplo de regresión es la predicción de la cantidad esperada de siniestros que hará un asegurado o la predicción de los precios futuros de los valores. Este tipo de predicciones también se utilizan para el trading algorítmico.

- *Detección de anomalías*: En este tipo de tareas, el programa informático examina un subconjunto de un conjunto de eventos u objetos y señala algunos de ellos como inusuales o atípicos. Un ejemplo de tarea de detección de anomalías es la detección de fraudes con tarjetas de crédito.

- *Traducción*: En una tarea de traducción, la entrada consiste en una secuencia de símbolos en algún idioma y el programa informático debe convertirla en una secuencia de símbolos en otro idioma. Esto se suele aplicar a las lenguas naturales, como la traducción del inglés al francés.

Por supuesto, son posibles muchas otras tareas y tipos de tareas. Los tipos de tareas que hemos enumerado aquí sólo pretenden ofrecer ejemplos de lo que el aprendizaje automático es capaz de hacer.

Las medidas de rendimiento, P

Para evaluar las capacidades de un algoritmo de aprendizaje automático, debemos diseñar una medida cuantitativa de su rendimiento. Normalmente, esta medida de rendimiento P es específica de la tarea T que realiza el sistema.

Para tareas como la clasificación, la clasificación con entradas perdidas y la transcripción, se suele medir la precisión del modelo. La precisión es simplemente la proporción de ejemplos para los que el modelo proporciona la salida correcta. También podemos obtener información equivalente midiendo la tasa de error, la proporción de ejemplos para los que el modelo produce una salida incorrecta.

Por lo general, nos interesa saber qué rendimiento tiene el algoritmo de aprendizaje automático en datos que no ha visto antes, ya que esto determina lo bien que funcionará cuando se despliegue en el mundo real. Por lo tanto, evaluaremos estas medidas de rendimiento utilizando un conjunto de datos de prueba que este separado de los datos utilizados en la construcción del modelo de aprendizaje automático.

La elección de la medida de rendimiento puede parecer sencilla y objetiva, pero a menudo es difícil elegir una medida de rendimiento que se corresponda con el comportamiento deseado del sistema.

Las experiencias, E

Los algoritmos de aprendizaje automático pueden clasificarse en términos generales como no supervisados o supervisados dependiendo del tipo de experiencia que se les permita tener durante el proceso de aprendizaje.

La mayoría de los algoritmos de aprendizaje pueden entenderse como si se les permitiera experimentar un conjunto de datos completo. Un conjunto de datos es una colección de muchos objetos llamados ejemplos, y cada ejemplo contiene muchas características que han sido objetivamente medidas. A veces también llamaremos a los ejemplos puntos de datos.

Los algoritmos de aprendizaje no supervisado experimentan un conjunto de datos que contiene muchas características, luego aprenden propiedades útiles de la estructura de este conjunto de datos. En el contexto del aprendizaje profundo, normalmente queremos aprender toda la distribución de probabilidad

que ha generado un conjunto de datos, ya sea explícitamente, como en la estimación de la densidad, o implícitamente para tareas como la síntesis o la eliminación de ruido.
Los algoritmos de aprendizaje supervisado trabajan con un conjunto de datos que contiene características, pero cada ejemplo está asociado a una etiqueta o a un objetivo.
A grandes rasgos, el aprendizaje no supervisado consiste en observar varios ejemplos de un vector aleatorio x, e intentar aprender implícita o explícitamente la distribución de probabilidad $p(x)$, o alguna probabilidad $p(x)$, o algunas propiedades interesantes de esa distribución, mientras que el aprendizaje supervisado consiste en observar varios ejemplos de un vector aleatorio x y un valor o vector asociado y y aprender a predecir y a partir de x, por ejemplo, estimando $p(y|x)$. El término aprendizaje supervisado tiene su origen en la idea de que el objetivo y es proporcionado por un instructor o profesor que muestra al sistema de aprendizaje automático lo que debe hacer. En el aprendizaje no supervisado, no hay instructor, y el algoritmo debe aprender a dar sentido a los datos sin esta guía.
El aprendizaje no supervisado y el aprendizaje supervisado no son términos formalmente definidos. Los límites entre ellos son a menudo borrosos. Muchas tecnologías de aprendizaje automático pueden utilizarse para realizar ambas tareas. Por ejemplo, la regla de la cadena de probabilidad establece que para un vector $x \in R^n$, la distribución conjunta puede descomponerse como

$$p(x) = \prod_{i=1}^{n} p(x_i|x_1, \cdots, x_{i-1})$$

Esta descomposición significa que podemos resolver el problema aparentemente no supervisado dividiéndolo en n problemas de aprendizaje supervisado. Alternativamente, podemos resolver el problema de aprendizaje supervisado, $p(y|x)$, utilizando tecnologías de aprendizaje no supervisado para aprender la distribución conjunta $p(x, y)$ e infiriendo

$$p(y|x) = \frac{p(x, y)}{\sum_y p(x, y)}$$

Aunque el aprendizaje no supervisado y el supervisado no son conceptos completamente formales o conceptos distintos, ayudan a categorizar de forma aproximada algunas de las cosas que hacemos con los algoritmos de aprendizaje automático. Tradicionalmente, la gente se refiere a la regresión, clasificación y problemas de salida estructurada como aprendizaje supervisado. Tareas de estimación de densidad se suelen considerar aprendizaje no supervisado.
La mayoría de los algoritmos de aprendizaje automático simplemente trabajan con un conjunto de datos. Un conjunto de datos puede describirse de muchas maneras. En todos los casos, un conjunto de datos es una colección de ejemplos. Cada ejemplo es una colección de observaciones llamadas características, recogidas en un momento o lugar diferente. Una forma común de describir un conjunto de datos es con una matriz de diseño. Una matriz de diseño es una matriz que contiene un ejemplo diferente en cada fila. Cada columna de la matriz corresponde a una característica diferente.
Al igual que no existe una definición formal de aprendizaje supervisado y no supervisado no existe una taxonomía rígida de conjuntos de datos o experiencias. Las estructuras descritas aquí cubren la mayoría de los casos, pero siempre es posible diseñar otras para nuevas aplicaciones.

5.1.2. Algoritmos supervisados

La mayoría de los algoritmos de aprendizaje supervisado se basan en la estimación de una distribución de probabilidad $p(y|x)$. Podemos hacerlo simplemente utilizando la estimación de máxima verosimilitud condicional para encontrar el mejor vector de parámetros θ para una familia paramétrica de distribuciones $p(y|x;\theta)$.
Es trivial comprobar que la regresión lineal corresponde a la familia $p(y|x;\theta) = N(y|\theta^T x, I)$. Podemos generalizar la regresión lineal al escenario de la clasificación definiendo una familia diferente de distribuciones de probabilidad. Si tenemos dos clases, la clase 0 y la clase 1, entonces sólo necesitamos especificar la probabilidad de una de estas clases. La probabilidad de la clase 1 determina la probabilidad de la clase 0, porque estos dos valores deben sumar 1.
La distribución normal sobre números de valor real que utilizamos para la regresión lineal está parametrizada en términos de una media. Cualquier valor que proporcionemos para esta media es válido. Una distribución sobre una variable binaria es un poco más complicada, porque su media debe estar siempre entre 0 y 1. Una forma de resolver este problema es utilizar la función sigmoidea logística para fijar la salida de la función lineal en el intervalo $(0, 1)$ e interpretar ese valor como una probabilidad:

$$p(y = 1|x;\theta) = \sigma(\theta^T x).$$

Este enfoque se conoce como regresión logística (un nombre un tanto extraño ya que utilizamos el modelo para la clasificación y no para la regresión).
En el caso de la regresión lineal, se puede encontrar los pesos óptimos resolviendo las ecuaciones normales. La regresión logística es algo más difícil. No existe una solución de forma cerrada para sus pesos óptimos. En su lugar, debemos buscarlos maximizando la log-verosimilitud. Podemos hacerlo minimizando la log-verosimilitud negativa (NLL) utilizando el descenso de gradiente.
Esta misma estrategia puede aplicarse a cualquier problema de aprendizaje supervisado, escribiendo una familia paramétrica de probabilidad de distribuciones condicionales sobre el tipo adecuado de variables de entrada y salida.
Máquinas de vectores de apoyo
Uno de los enfoques más influyentes del aprendizaje supervisado es la máquina de vectores de apoyo [7]. Este modelo es similar a la regresión logística en el sentido de que se rige por una función lineal $w^T x+b$. A diferencia de la regresión logística, la máquina de vectores de apoyo no proporciona probabilidades, sino que sólo da como resultado una identidad de clase.
Una innovación clave asociada a las máquinas de vectores de apoyo es el truco del núcleo. El truco del núcleo consiste en observar que muchos algoritmos de aprendizaje automático pueden escribirse exclusivamente en términos de productos escalares entre ejemplos. Por ejemplo, se puede demostrar que la función lineal utilizada por la máquina de vectores de apoyo puede reescribirse como:

$$w^T x + b = b + \sum_{i=1}^{m} \alpha_i x^T x^{(i)}$$

donde $x^{(i)}$ es un ejemplo de entrenamiento y α es un vector de coeficientes. Reescribiendo el algoritmo de aprendizaje nos permite sustituir x por la salida de una función determinada $\phi(x)$ y el producto escalar con una función $k(x, x^{(i)}) = \phi(x)^T \phi(x^{(i)})$ llamada núcleo.

A continuación, podemos hacer predicciones utilizando la función:

$$f(x) = b + \sum_i \alpha_i k(x, x^{(i)}).$$

Esta función es lineal en el espacio al que mapea ϕ, pero no es lineal en función de x.
El truco del núcleo es potente por dos razones. En primer lugar, nos permite aprender modelos que son no lineales en función de x utilizando técnicas de optimización convexas que garantizan una convergencia eficiente. Esto sólo es posible porque consideramos ϕ fija y sólo optimizamos α, es decir, el algoritmo de optimización puede ver la función de decisión como si fuera lineal en un espacio diferente. En segundo lugar, la función k del núcleo no necesita implementarse en términos de aplicar explícitamente el mapeo ϕ y luego aplicar el producto de escalar. El producto escalar en el espacio ϕ puede ser equivalente a una operación no lineal pero computacionalmente menos costosa en el espacio x. Por ejemplo, podríamos diseñar un mapeo de características de dimensión infinita $\phi(x)$ sobre los enteros no negativos. Supongamos que este mapeo devuelve un vector que contiene x unos seguidos de infinitos ceros. Construir explícitamente esta cartografía, o tomar el producto escalar entre dos vectores de este tipo, cuesta un tiempo y una memoria infinitos. Pero podemos escribir una función $k(x, x^{(i)}) = min(x, x^{(i)})$ que es exactamente equivalente a este producto escalar de dimensión infinita.
El núcleo más utilizado es el núcleo gaussiano

$$k(u, v) = N(u - v; 0, \sigma^2 I)$$

donde $N(x; \mu, \Sigma)$ es la densidad normal estándar. Este núcleo corresponde al producto escalar $k(u, v) = \phi(x)^T \phi(x)$ en un espacio de características de dimensión infinita ϕ y también tiene una interpretación como función de similitud, actuando como una especie de plantilla.
Las máquinas de vectores de apoyo no son el único algoritmo que puede mejorarse utilizando el truco del núcleo. Muchos modelos lineales pueden mejorarse de este modo. Esta categoría de algoritmos se conoce como máquinas kernel o métodos kernel.
Una de las principales desventajas de las máquinas kernel es que el coste de aprendizaje de los coeficientes α es cuadrático en el número de ejemplos de entrenamiento. Un problema relacionado es que el coste de evaluar la función de decisión es lineal en el número de ejemplos de entrenamiento, porque el i-ésimo ejemplo contribuye con un término $\alpha_i k(x, x^{(i)})$ a la función de decisión. Las máquinas de vectores de apoyo son capaces de mitigar esto aprendiendo un vector α que contiene mayoritariamente ceros. La clasificación de un nuevo ejemplo requiere entonces la evaluación de la función kernel sólo para los ejemplos de entrenamiento que tienen un α_i distinto de cero. Estos ejemplos de entrenamiento se conocen como vectores de soporte.

5.1.3. Algoritmos no-supervisados

La distinción entre algoritmos supervisados y no supervisados no está definida de manera formal y rígida ya que no existe una prueba objetiva para distinguir si un valor es una característica o un objetivo proporcionado por un supervisor. Informalmente, el aprendizaje no supervisado se refiere a la mayoría de los intentos de extraer información de una distribución que no requiere el trabajo humano para anotar ejemplos. El término suele asociarse a la estimación de la densidad, el aprendizaje para extraer muestras de una distribución, aprender a eliminar el ruido de los datos de alguna distribución,

encontrar una distribución cerca de la cual se encuentran los datos, o agrupar los datos en grupos de ejemplos relacionados.
Aprender una representación de los datos es una tarea clásica de aprendizaje no supervisado que consiste en encontrar la "mejorrepresentación de los datos. Por "mejor"podemos entender diferentes cosas, pero en general buscamos una representación que conserve la mayor cantidad de información posible sobre la información contenida en x y que, al mismo tiempo, obedezca a alguna penalización o restricción para que la representación sea más sencilla o más accesible que la propia x.
Hay múltiples formas de definir una representación más simple, algunas de las más comunes son las representaciones de menor dimensión, representaciones dispersas y representaciones independientes. Las representaciones de baja dimensión intentan comprimir toda la información posible sobre x en una representación más pequeña. Las representaciones dispersas suelen incluir el conjunto de datos en una representación de alta dimensión en la que el número de entradas no nulas es pequeño. Esto da lugar a una estructura general de la representación que tiende a distribuir los datos a lo largo de los ejes del espacio de representación. Las representaciones independientes intentan desentrañar las fuentes de variación que subyacen a la distribución de los datos, de modo que las dimensiones de la representación sean estadísticamente independientes.
Por supuesto, estos tres criterios no se excluyen mutuamente. Las representaciones de baja dimensión suelen producir elementos que tienen menos dependencias o dependencias más débiles que los datos originales de alta dimensión. Esto se debe a que una forma de reducir el tamaño de una representación es encontrar y eliminar las redundancias. Identificar y eliminar más redundancias permite al algoritmo de reducción de la dimensionalidad lograr una mayor compresión descartando menos información.

El análisis de componentes principales

Uno de los métodos de aprendizaje no supervisado más utilizados: El análisis de componentes principales (PCA). El PCA es una transformación lineal y ortogonal de los datos, los cuales son proyectados en una representación en la que los elementos no están correlacionados entre sí.
Podemos utilizar el PCA como un método sencillo y eficaz de reducción de la dimensionalidad que preserva la mayor cantidad posible de información en los datos (medido por el error de reconstrucción por mínimos cuadrados). A continuación, examinaremos otras propiedades de la representación PCA. En concreto, estudiaremos cómo la representación PCA descorrelaciona la representación de datos original X.
Consideremos la matriz de diseño m-dimensional X. Supondremos que los datos tienen una media de cero, $E[x] = 0$. Si este no es el caso, los datos pueden ser fácilmente centrados. La matriz de covarianza muestral insesgada asociada a X viene dada por:

$$Var[x] = \frac{1}{n-1} X^T X$$

Un aspecto importante del PCA es que encuentra una representación (mediante una transformación lineal) $z = Wx$ donde $Var[z]$ es diagonal. Para ello, utilizaremos la descomposición del valor singular (SVD) de $X : X = U\Sigma W^T$, donde Σ es una nm matriz diagonal rectangular con los valores singulares de X en la diagonal principal, U es una matriz nn cuyas columnas son ortonormales (es decir, de longitud unitaria y ortogonal) y W es una matriz mm también compuesta por vectores columna ortonormales. Utilizando la SVD de X, podemos reexpresar la varianza de X como

$$Var[x] = \frac{1}{n-1} W\Sigma^2 W^T,$$

donde definimos Σ^2 como una matriz diagonal mm-dimensional diagonal con los cuadrados de los valores singulares de X en la diagonal, es decir, el i-ésimo elemento de la diagonal viene dado por $\Sigma^2_{i,i}$.
Esta capacidad de PCA para transformar los datos en una representación en la que los elementos no están correlacionados entre sí es una propiedad muy importante de PCA. Es un ejemplo sencillo de una representación que intenta desentrañar los factores desconocidos de variación subyacentes en los datos. En el caso del PCA, este desentrañamiento adopta la forma de encontrar una rotación del espacio de entrada (mediada por la transformación W) que alinea los ejes principales de la varianza con la base del nuevo espacio de representación asociada a z, como se ilustra en la Fig. 5.4. Aunque la correlación es una categoría importante de dependencia entre los elementos de los datos, también estamos interesados en aprender representaciones que desentrañen formas más complicadas de dependencias entre características.

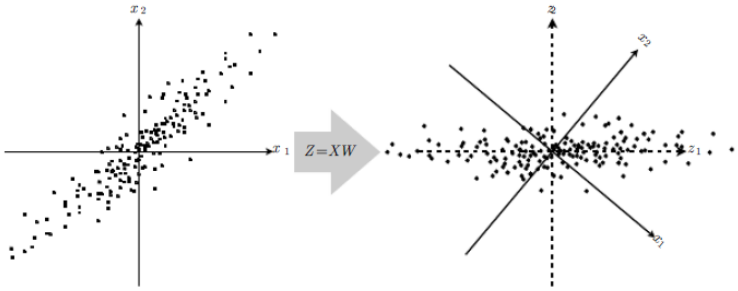

Figura 5.4: Ilustración de la representación de los datos, aprendida a trravés de PCA.

5.1.4. Contruyendo un algoritmo de aprendizaje automático

Casi todos los algoritmos de aprendizaje profundo pueden describirse como instancias particulares de una receta bastante sencilla: combinar una especificación de un conjunto de datos, una función de coste, un procedimiento de optimización y un modelo.
Por ejemplo, el algoritmo de regresión lineal combina un conjunto de datos formado por X e y, la función de coste

$$J(w, b) = -E_{x,y\sim p_{data}} p_{model}(y|x)$$

y la especificación del modelo $p_{model}(y|x) = N(y|x^T w + b, 1)$. Normalmente J se simplifica en el error cuadrático medio y podemos optar por optimizarlo de forma cerrada resolviendo las ecuaciones normales con la pseudoinversa de Moore-Penrose.
Ahora bien, si modificamos alguno de estos componentes podemos obtener una una gran variedad de algoritmos.
La función de coste suele incluir al menos un término que hace que el proceso de aprendizaje realice una estimación estadística. La función de coste más común es la probabilidad logarítmica negativa, de modo que la minimización de la función de coste provoca una estimación de máxima probabilidad. Este término principal de la función de coste suele descomponerse como una suma sobre los ejemplos de entrenamiento de alguna función de pérdida. Por ejemplo, la probabilidad logarítmica condicional

negativa de los datos de entrenamiento puede escribirse como:

$$J(\Theta) = E_{x,y\sim p_{data}} L(x, y, \Theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \Theta)$$

donde y es la pérdida por ejemplo $L(x, y, \Theta) = logp(y|x; \Theta)$. Podemos diseñar muchas funciones de coste diferentes simplemente tomando la expectativa a través del conjunto de entrenamiento de diferentes funciones de pérdida por muestra.
La función de coste también puede incluir términos adicionales, como términos de regularización. Por ejemplo, podemos añadir un decaimiento del peso a la función de coste de regresión lineal para obtener

$$J(w, b) = \lambda ||w||_2^2 - E_{x,y\sim p_{data}} p_{model}(y|x)$$

Esto todavía permite una optimización de forma cerrada.
Una forma común y sencilla de optimizar estas funciones objetivo aditivas es mediante el descenso de gradiente estocástico o SGD. El SGD es una forma de descenso de gradiente en la que no utilizamos el gradiente real, sino un estimador estocástico del gradiente (cuyo valor esperado debe ser igual al verdadero gradiente). En el contexto del aprendizaje automático el estimador se forma considerando un ejemplo o un subgrupo que contiene unos ejemplos, elegidos al azar, y utilizando únicamente el gradiente de estos ejemplos para actualizar los parámetros. Por ejemplo, con las definiciones anteriores de pérdida, por ejemplo L, los parámetros podrían actualizarse iterativamente como sigue:

$$\Theta \Longleftarrow \Theta - \epsilon \frac{\partial L(x^{(i)}, y^{(i)}, \Theta)}{\partial \Theta}$$

donde i es el índice de un ejemplo de entrenamiento elegido al azar.
Si cambiamos el modelo para que sea no lineal, entonces la mayoría de las funciones de coste ya no pueden optimizarse de forma cerrada. Esto nos obliga a elegir un procedimiento de optimización numérico iterativo de optimización, como el descenso de gradiente.
Esta receta admite tanto el aprendizaje supervisado como el no supervisado. El ejemplo de la regresión lineal muestra cómo apoyar el aprendizaje supervisado. El aprendizaje no supervisado puede ser apoyado por la definición de un conjunto de datos que contiene sólo X y proporcionando un coste y un modelo no supervisados adecuados. Por ejemplo, podemos obtener el primer vector PCA especificando que nuestra función de pérdida es

$$J(w) = E_{x\sim p_{data}} ||x - r(x; w)||_2^2$$

mientras que nuestro modelo se define para tener w con norma uno y función de reconstrucción $r(x) = w^T wx$.
En algunos casos, la función de coste puede ser una función que no podemos evaluar, por razones computacionales. En estos casos, podemos minimizarla aproximadamente utilizando la optimización numérica iterativa siempre que tengamos alguna forma de aproximar sus gradientes.
La mayoría de los algoritmos de aprendizaje automático utilizan esta receta, aunque no resulte trivial a primera vista. Si un algoritmo de aprendizaje automático parece especialmente único o diseñado a mano, normalmente se puede entender que utiliza un optimizador especial. Algunos modelos, como los árboles de decisión, requieren optimizadores especiales porque sus funciones de coste son planas e

inapropiadas para la minimización mediante optimizadores basados en el gradiente. Reconocer que la mayoría de los algoritmos de aprendizaje automático pueden describirse con esta receta ayuda a ver los diferentes algoritmos como parte de una taxonomía de métodos para realizar tareas relacionadas que funcionan a través de razones similares y no como una larga lista de algoritmos que tienen justificación por separado.

5.2. Deep Learning

El aprendizaje profundo tiene una larga historia y muchas aspiraciones. Se han propuesto varios enfoques los cuales aún no han dado sus frutos por completo. Varios objetivos ambiciosos que aún no se han hecho realidad.
El aprendizaje profundo moderno proporciona un marco muy potente para el aprendizaje supervisado. Al añadir más capas y más unidades dentro de una capa, una red profunda puede representar funciones de complejidad creciente. La mayoría de las tareas que consisten en mapear un vector de entrada a un vector de salida y que son fáciles de realizar por una persona pueden llevarse a cabo mediante el aprendizaje profundo, dado un modelo suficientemente grande y un conjunto de datos de ejemplos de entrenamiento correctamente etiquetados. Otras tareas, que no pueden describirse como la asociación de un vector a otro, o que son lo suficientemente difíciles como para que una persona necesite tiempo para pensar y reflexionar, quedan fuera del alcance del aprendizaje profundo por ahora.

5.2.1. Red Neuronal Prealimentada

Las redes profundas de avance, también conocidas como perceptrones multicapa (MLP), son las redes profundas por excelencia. Son funciones paramétricas que se definen componiendo muchas funciones paramétricas. Cada una de estas funciones componentes tiene múltiples entradas y múltiples salidas. En la terminología de las redes neuronales, nos referimos a cada subfunción como una capa de la red, y a cada salida escalar de una de estas funciones como una unidad o, a veces, como una característica. Aunque cada unidad implementa una relativamente sencilla transformación de su entrada, la función representada por toda la red puede llegar a ser arbitrariamente compleja.
No todos los algoritmos de aprendizaje profundo pueden entenderse en términos de definición de una función determinista como las redes profundas de avance, pero todos ellos comparten la propiedad de contener muchas capas de muchas unidades. Podemos pensar que el número de unidades en cada capa representa la anchura de un modelo de aprendizaje automático, y el número de capas su profundidad. Las redes profundas de avance proporcionan un ejemplo conceptualmente sencillo de un algoritmo que capta las numerosas ventajas de tener una anchura y profundidad significativas. Las redes profundas de avance son también la tecnología clave que subyace en la mayoría de las aplicaciones comerciales contemporáneas de aprendizaje profundo a grandes conjuntos de datos.
Las redes neuronales nos permiten aprender nuevos tipos de no linealidad. Otra forma de ver esta idea es que las redes neuronales nos permiten aprender las características proporcionadas a un modelo lineal. Desde este punto de vista, las redes neuronales permiten automatizar el diseño de las características, una tarea que hasta hace poco se realizaba de forma gradual y colectiva, mediante el esfuerzo combinado de toda una comunidad de investigadores.
Las redes neuronales supervisadas de tipo avance fueron uno de los primeros y más exitosos algoritmos de aprendizaje no lineal [8]. Estas redes aprenden al menos una función que define las características,

así como una función (normalmente lineal) de las características a la salida. Las capas de la red que corresponden a las características en lugar de a las salidas se denominan capas ocultas. Esto se debe a que los valores correctos de las características son desconocidos. Las características deben ser creadas por el algoritmo de entrenamiento. La entrada y la salida de la red son, por el contrario, observadas o visibles en los datos de entrenamiento. La Fig 6.1 muestra una arquitectura MLP clásica de los década de 1980, con una sola capa oculta. Una versión más profunda se obtiene simplemente teniendo más capas ocultas. Las versiones modernas incluyen cambios en las no linealidades utilizadas y el procedimiento de entrenamiento.

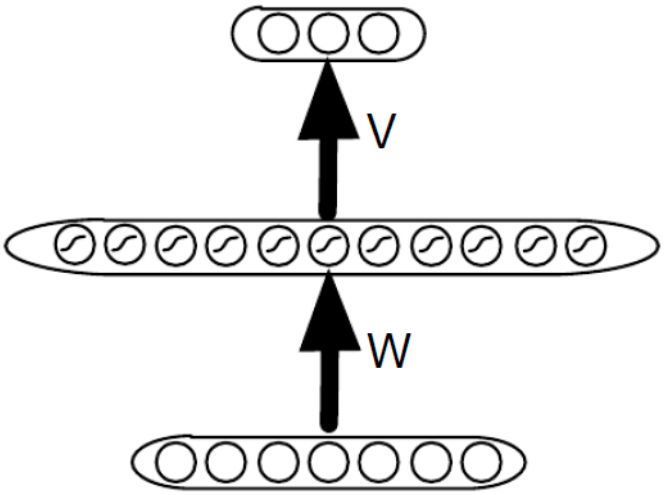

Figura 5.5: Ejemplo de una arquitectura MLP, con una capa oculta

A continuación, introducimos un ejemplo en el que se presentan las ecuaciones de un MLP superficial para la regresión similar a las introducidas en la década de 1980, ilustradas en la Fig 6.1.
Red neuronal multicapa superficial para la regresión
Basándonos en las definiciones anteriores, la familia de funciones de entrada-salida elegida es la siguiente:

$$f_\Theta(x) = b + V sigmoide(c + Wx),$$

ilustrada en la Fig 6.1, donde $sigmoide(a) = 1/(1 + e^{-a})$ se aplica por elementos, la entrada es el vector $x \in R^{n_i}$, las salidas de la capa oculta son los elementos del vector $h = sigmoide(c+Wx)$ con n_h entradas, los parámetros son $\Theta = (b, c, V, W)$ (con Θ también visto como la versión vectorial aplanada de la tupla) con $b \in R^{n_o}$ un vector de la misma dimensión que la salida (n_o), $c \in R^{n_h}$ de la misma dimensión que h (número de unidades), siendo $V \in R^{n_o n_h}$ y $W \in R^{n_h n_i}$ matrices de pesos.
La función de pérdida para este ejemplo clásico podría ser el error al cuadrado $L(\vec{y}, y) = ||\vec{y} - y||^2$. El regularizador podría ser el decaimiento ordinario del peso de L^2 $||\omega||^2 = (\sum_{i,j} W_{i,j}^2 + \sum_{k,i} V_{k,i}^2)$, donde definimos el conjunto de pesos ω como la concatenación de los elementos de las matrices W y V . El decaimiento del peso L^2 penaliza la norma cuadrada de los pesos, siendo λ un escalar que es más grande para penalizar los pesos más fuertes, con lo que se obtienen pesos más pequeños. Durante el entrenamiento, minimizamos una función de coste que se obtiene sumando la pérdida cuadrada y el

término de regularización:

$$J(\Theta) = \lambda||\omega||^2 + \frac{1}{n}\sum_{t=1}^{n}||y^{(t)} - (b + Vsigmoide(c + Wx^{(t)}))||^2$$

donde $(x^{(t)}, y^{(t)})$ es el ejemplo de entrenamiento t-ésimo, un par (entrada, objetivo). Por último el procedimiento de entrenamiento clásico en este ejemplo es el descenso de gradiente estocástico que actualiza iterativamente Θ según

$$\omega \Longleftarrow \omega - \epsilon(2\lambda\omega + \Delta_\omega L(f_\Theta(x^{(t)}, y^{(t)})))$$

$$\beta \Longleftarrow \beta - \epsilon\Delta_\beta L(f_\Theta(x^{(t)}, y^{(t)}))$$

donde $\beta = (b, c)$ contiene los parámetros de *offset1*, $\omega = (W, V)$ las matrices de pesos, ϵ es una tasa de aprendizaje y t se incrementa después de cada ejemplo de entrenamiento, módulo n.
Hay muchas maneras de definir la familia de funciones de entrada-salida, la función de coste (incluyendo regularizadores opcionales) y el procedimiento de optimización. Las más comunes se describen a continuación, mientras que las más avanzadas se dejan para la investigación particular del lector interesado. Una motivación para la familia de funciones definidas por las redes neuronales multicapa es componer transformaciones simples para obtener otras altamente no lineales.
La mayoría de ellas se combinan típicamente con una transformación afín $a = b + Wx$ y aplicada de forma elemental:

$$h = \phi(a) \Longleftrightarrow h_i = \phi(a_i) = \phi(b_i + W_{i,:}x)$$

Veamos algunos de los casos más comunes:

- Rectificador o unidad lineal rectificada (ReLU) o parte positiva: transformación de la salida de la capa anterior: $\phi(a) = max(0, a)$. Esta es, con mucho, la unidad oculta más popular en las redes de avance actuales.
- Tangente hiperbólica: $\phi(a) = tanh(a)$.
- Sigmoide: $\phi(a) = 1/(1 + e^{-a})$.
- Softmax: Es una transformación de vector a vector $\phi(a) = softmax(a) = e^{a_i}/\sum_j e^{a_j}$ tal que $\sum_i \phi_i(a) = 1$ y $\phi_i(a) > 0$, es decir, la salida softmax puede considerarse como una distribución de probabilidad sobre un conjunto finito de resultados.

Esta lista no es exhaustiva, pero cubre la mayoría de las no linealidades, los cálculos unitarios de las no linealidades y los cálculos unitarios vistos en la literatura de aprendizaje profundo y redes neuronales. Sin embargo, son posibles muchas variantes.

5.2.2. Regularización de los modelos

Un problema central en el aprendizaje automático es cómo hacer un algoritmo que funcione bien no sólo con los datos de entrenamiento, sino también con nuevas entradas. Muchas estrategias utilizadas en el aprendizaje automático están diseñadas explícitamente para reducir el error de prueba, posiblemente a costa de aumentar el error de entrenamiento. Estas estrategias se conocen colectivamente como

regularización.
La regularización es cualquier componente del modelo, proceso de entrenamiento o procedimiento de predicción que se incluye para tener en cuenta las limitaciones de los datos de entrenamiento, incluyendo su finitud. Hay muchas estrategias de regularización. Algunas imponen restricciones adicionales a un modelo de aprendizaje automático, como la adición de restricciones a los valores de los parámetros. Algunas añaden términos extra en la función objetivo que pueden ser considerados como los correspondientes a una restricción suave en los valores de los parámetros. Si se eligen con cuidado, estas restricciones y penalizaciones adicionales pueden conducir a un mejor rendimiento en el conjunto de pruebas. A veces estas restricciones y penalizaciones están diseñadas para codificar tipos específicos de conocimiento previo. Otras veces, estas restricciones y penalizaciones están diseñadas para expresar una preferencia genérica por una clase de modelo más simple con el fin de promover la generalización. A veces las penalizaciones y restricciones son necesarias para hacer un problema indeterminado. Otras formas de regularización, conocidas como métodos de conjunto, combinan múltiples hipótesis que explican los datos de entrenamiento.
En el contexto del aprendizaje profundo, la mayoría de las estrategias de regularización se basan en estimadores de regularización. La regularización de un estimador funciona intercambiando un mayor sesgo por una varianza reducida. Un regularizador eficaz es el que hace un intercambio rentable, es decir, que reduce significativamente la varianza sin aumentar excesivamente el sesgo.
El control de la complejidad del modelo no va a ser una simple cuestión de encontrar el modelo del tamaño correcto, del número correcto de parámetros. En lugar de ello, podríamos encontrar -y de hecho en los escenarios prácticos de aprendizaje profundo, casi siempre encontramos- que el modelo que mejor se ajusta (en el sentido de minimizar el error de generalización) es uno que posee un gran número de parámetros que no son totalmente libres de abarcar su dominio.
Como veremos, hay una gran cantidad de formas de regularización disponibles para el profesional del aprendizaje profundo. De hecho, el desarrollo de regularizadores más eficaces ha sido uno de los principales esfuerzos de investigación en este campo.
La mayoría de las tareas de aprendizaje automático pueden verse en términos de aprender a representar una función $\hat{f}(x)$ parametrizada por un vector de parámetros Θ. Los datos consisten en de entradas $x^{(i)}$ y (para algunas tareas) objetivos $y^{(i)}$ para $i \in \{1, \cdots, m\}$. En el caso de de la clasificación, cada $y^{(i)}$ es una etiqueta de clase entera en $\{1, \cdots, k\}$. En el caso de las tareas de regresión cada $y^{(i)}$ es un número real o un vector de valor real. Entonces denotamos el objetivo en negrita, como $\mathbf{y}^{(i)}$. Para las tareas de estimación de la densidad, simplemente no hay objetivos.
A veces tenemos variables $\mathbf{y}$ que pueden considerarse naturalmente como objetivos a predecir y deseamos estimar la distribución conjunta sobre x e $\mathbf{y}$. En este caso, simplemente definimos un nuevo vector x' que contiene x e $\mathbf{y}$ concatenados. A continuación, podemos realizar una estimación de la densidad sobre x'. Podemos agrupar estos ejemplos en una matriz de diseño X y un vector de objetivos y (cuando $y^{(i)}$ es un escalar), o una matriz de objetivos Y (cuando $\mathbf{y}^{(i)}$ es un vector).
En el aprendizaje profundo, nos interesa principalmente el caso en que la función $\hat{f}(x)$ tiene un gran número de parámetros y como resultado posee una alta capacidad para ajustar funciones relativamente complicadas. Esto significa que los algoritmos de aprendizaje profundo suelen requerir de un gran conjunto de entrenamiento o una cuidadosa regularización (destinada a reducir la capacidad efectiva del modelo o de guiarlo hacia una solución específica utilizando información previa) o ambos.

Regularización de parámetros L2

Uno de los tipos más simples y comunes de regularización clásica: la penalización de la norma de los pa-

rámetros L^2, comúnmente conocida como decadencia del peso. Esta estrategia de regularización acerca los parámetros al origen añadiendo un término de regularización $\Omega(\Theta) = \frac{1}{2}||w||_2^2$ a la función objetivo. Aquí, w es el subconjunto de parámetros llamados pesos. Por pesos, nos referimos a los parámetros de una transformación lineal. No todos los parámetros son pesos. Por ejemplo, los sesgos parametrizan el desplazamiento de una transformación afín, por lo que no se consideran pesos en este contexto.
En varios contextos, la regularización L^2 también se conoce como regresión Ridge o regularización de Tikhonov.
En el contexto de las redes neuronales, a veces es deseable utilizar una penalización con un coeficiente α diferente para cada capa de la red. Dado que α es un hiperparámetro y que puede ser costoso buscar el valor correcto de múltiples hiperparámetros, también es razonable utilizar el mismo decaimiento de peso en todas las capas para reducir el espacio de búsqueda.
Podemos obtener alguna información sobre el comportamiento de la regularización del decaimiento del peso estudiando el gradiente de la función objetivo regularizada. Para simplificar la presentación, suponemos que no hay término de compensación, por lo que Θ es sólo w. Tal modelo tiene el siguiente gradiente de la función objetivo total:

$$\Delta_w \tilde{J}(w; X, y) = \alpha w + \Delta_w J(w; X, y) \tag{5.1}$$

Para dar un único paso de gradiente para actualizar los pesos, realizamos esta actualización:

$$w \longleftarrow w - \epsilon(\alpha w + \Delta w J(w; X, y)).$$

Escrito de otra manera, la actualización es:

$$w \longleftarrow (1 - \epsilon\alpha)w - \epsilon \Delta w J(w; X, y).$$

Podemos ver que la adición del término de decaimiento del peso ha modificado la regla de aprendizaje para reducir multiplicativamente el vector de pesos en un factor constante en cada paso, hacia cero, justo antes de realizar la actualización habitual del gradiente. Esto describe lo que ocurre en un solo paso. Pero, ¿qué ocurre a lo largo de todo el curso del entrenamiento?
Simplificaremos aún más el análisis considerando una aproximación cuadrática a la función objetivo en el entorno del valor empíricamente óptimo de los pesos w^*. (Si la función objetivo es realmente cuadrática, como en el caso de ajustar un modelo de regresión lineal con error cuadrático medio, entonces la aproximación es perfecta).

$$\hat{J}(\Theta) = J(w^*) + \frac{1}{2}(w - w^*)^T H(w - w^*) \tag{5.2}$$

donde H es la matriz hessiana de J respecto a w evaluada en w^*. No hay término de primer orden en esta aproximación cuadrática, porque w^* se define como un mínimo, donde el gradiente desaparece. Asimismo, como w^* es un mínimo, podemos concluir que H es semidefinida positiva.

$$\Delta_w \hat{J}(w) = H(w - w^*) \tag{5.3}$$

Si sustituimos el gradiente exacto de la ecuación 5.1 por el gradiente aproximado en la ecuación 5.3, podemos escribir una ecuación para la localización del mínimo de la función objetivo regularizada:

$$\tilde{w} = (H + \alpha I)^{-1} H w^* \tag{5.4}$$

A partir de esto, vemos que la presencia del término de regularización mueve el óptimo de w^* hacia $\tilde{w}$. A medida que α se acerca a 0, $\tilde{w}$ se acerca a w^*. Pero, ¿qué ocurre a medida que α crece? Como H es real y simétrica, podemos descomponerla en una diagonal diagonal $\bigwedge$ y una base ortonormal de vectores propios, Q, tal que $H = Q \bigwedge Q^T$. Aplicando la descomposición a la ecuación 5.4, obtenemos

$$Q^T \tilde{w} = (\bigwedge + \alpha I)^{-1} \bigwedge Q^T w^*. \tag{5.5}$$

Si interpretamos el término $Q^T \tilde{w}$ como la rotación de nuestros parámetros de solución $\tilde{w}$ en la base definida por los vectores propios Q de H, entonces vemos que el efecto del decaimiento del peso es reescalar los coeficientes de los vectores propios. Específicamente el i-ésimo componente se reescala por un factor de $\frac{\lambda_i}{\lambda_i + \alpha}$.
A lo largo de las direcciones donde los valores propios de H son relativamente grandes, por ejemplo donde $\lambda_i >> \alpha$, el efecto de la regularización es relativamente pequeño. Sin embargo, los componentes con $\lambda_i << \alpha$ se reducirán hasta tener una magnitud casi nula. Este efecto se ilustra en la Fig. 7.1.

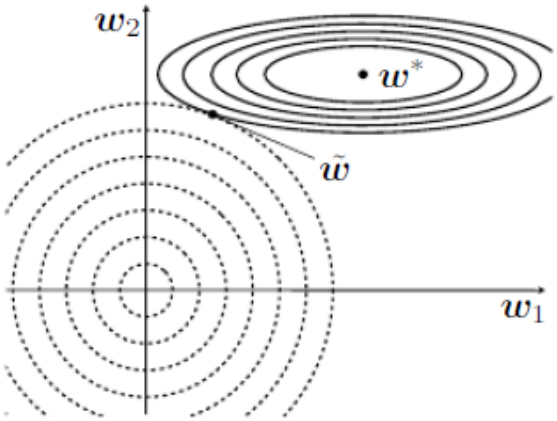

Figura 5.6: Una ilustración del efecto de la regularización L^2 en el valor de la w óptima.

Sólo las direcciones a lo largo de las cuales los parámetros contribuyen significativamente a reducir la función objetivo se conservan relativamente intactas. En las direcciones que no contribuyen a reducir la función objetivo, un valor propio pequeño del hessiano nos indica que el movimiento en esta dirección no aumentará significativamente el gradiente. Los componentes del vector de pesos correspondientes a esas direcciones sin importancia se eliminan mediante el uso de la regularización a lo largo del entrenamiento. Este efecto de supresión de las contribuciones al vector de parámetros a lo largo de estas direcciones del hessiano H se recoge en el concepto de número efectivo de parámetros, que se define como

$$\gamma = \sum_i \frac{\lambda_i}{\lambda_i + \alpha}. \tag{5.6}$$

A medida que se incrementa α, el número efectivo de parámetros disminuye.
Otra forma de intuir el efecto de la regularización L^2 es estudiar su efecto en la regresión lineal. La

función objetivo no regularizada para regresión lineal es la suma de errores al cuadrado:

$$(Xw - y)^T(Xw - y).$$

Cuando añadimos la regularización L^2, la función objetivo cambia a

$$(Xw - y)^T(Xw - y) + \frac{1}{2}\alpha w^T w.$$

Esto cambia las ecuaciones normales para la solución de

$$w = (X^T X)^{-1} X^T y,$$

a

$$w = (X^T X + \alpha I)^{-1} X^T y.$$

Podemos ver que la regularización L^2 hace que el algoritmo de aprendizaje "perciba"que la entrada X tiene una mayor varianza, lo que hace que reduzca los pesos de las características cuya covarianza con el objetivo de salida es baja en comparación con esta varianza añadida.

5.2.3. Optimización del entrenamiento de los modelos

Los algoritmos de aprendizaje profundo implican la optimización en muchos contextos. Por ejemplo, a menudo resolvemos problemas de optimización de forma analítica para demostrar que cierto algoritmo posee una propiedad determinada. La inferencia en un modelo probabilístico puede plantearse como un problema de optimización. De todos los problemas de optimización relacionados con el aprendizaje aprendizaje profundo, el más difícil es el entrenamiento de las redes neuronales. Es bastante común invertir días o meses de tiempo en cientos de máquinas para resolver incluso una sola instancia del problema de entrenamiento de la red neuronal. Dado que este problema es tan importante y tan caro en términos computacionales, se ha desarrollado un conjunto especializado de técnicas de optimización para resolverlo. Esta sección presenta estas técnicas de optimización para el entrenamiento de redes neuronales.
Esta sección se centra en un caso particular de optimización: la minimización de una función objetivo $J(\Theta)$ con respecto a los parámetros del modelo Θ, que también es implícitamente una función de los datos de entrenamiento. Normalmente, esa función objetivo puede reescribirse como un promedio sobre el conjunto de entrenamiento, como por ejemplo

$$J(\Theta) = E_{(x,y)} \sim \hat{p}_{data} L(f(x;\Theta), y), \tag{5.7}$$

donde L es la función de pérdida por muestra, $f(x;\Theta)$ es la salida predicha cuando la entrada es x, y es la salida objetivo y $\hat{p}_{data}$ es la distribución empírica, en el caso del aprendizaje supervisado. Sin embargo, normalmente preferimos minimizar la función objetivo correspondiente donde la expectativa se toma a través de los datos generando la distribución p_{data} en lugar de sólo sobre el conjunto de entrenamiento finito:

$$J^*(\Theta) = E_{(x,y)} \sim p_{data} L(f(x;\Theta), y). \tag{5.8}$$

Los algoritmos de optimización utilizados para el entrenamiento de modelos profundos difieren de los algoritmos de optimización tradicionales en varios aspectos. El aprendizaje automático suele actuar

de forma indirecta - nos preocupamos por alguna medida de rendimiento P sobre la cúal no sabemos cómo influir directamente, por lo que reducimos alguna función objetivo $J(\Theta)$ con la esperanza de que mejore P. Esto contrasta con la optimización pura, en la que minimizar J es un objetivo en sí mismo. Los algoritmos de optimización para el entrenamiento de modelos profundos también suelen incluir alguna especialización en la estructura específica de las funciones objetivo del aprendizaje automático. Supongamos que tenemos un vector de características de entrada x y objetivos y, muestreados a partir de alguna distribución conjunta desconocida $p(x, y)$, así como una función de pérdida $L(x, y)$. Nuestro objetivo final es minimizar $E_{x,y} \sim p(x, y)[L(x, y)]$. Esta cantidad se conoce como el riesgo. Aquí destacamos que la expectativa se toma sobre la verdadera distribución subyacente, por lo que el riesgo es una forma de error de generalización. Si conociéramos la verdadera distribución $p(x, y)$, ésta sería una tarea de optimización que podría resolver un algoritmo de optimización. Sin embargo, cuando no conocemos $p(x, y)$, sino que sólo tenemos un conjunto de muestras de la misma, tenemos un problema de aprendizaje automático.
La forma más sencilla de volver a convertir un problema de aprendizaje automático en un problema de optimización es minimizar la pérdida esperada en el conjunto de entrenamiento. Esto significa sustituir la distribución verdadera $p(x, y)$ por la distribución empírica $\hat{p}(x, y)$ definida por el conjunto de entrenamiento. Ahora minimizamos el riesgo empírico

$$E_{x,y} \sim \hat{p}(x, y)[L(f(x; \Theta), y)] = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \Theta), y^{(i)})$$

donde m es el número de ejemplos de entrenamiento.
El proceso de entrenamiento basado en la minimización de este error medio de entrenamiento se conoce como minimización del riesgo empírico. En este contexto, el aprendizaje automático sigue siendo muy similar a la optimización directa. En lugar de optimizar el riesgo directamente, optimizamos el riesgo empírico y esperamos que el riesgo también disminuya significativamente. Una serie de resultados teóricos establecen las condiciones en las que se puede esperar que el riesgo verdadero disminuya considerablemente.
Sin embargo, la minimización del riesgo empírico es propensa al sobreajuste. Los modelos con alta capacidad pueden simplemente memorizar el conjunto de entrenamiento. En muchos casos, la minimización empírica del riesgo no es realmente factible. Los algoritmos de optimización modernos más eficaces se basan en el descenso del gradiente, pero muchas funciones de pérdida útiles, como la pérdida 0-1, no tienen derivadas útiles (la derivada es cero o indefinida en todas partes). Estos dos problemas hacen que, en el contexto del aprendizaje profundo, rara vez utilicemos la minimización empírica del riesgo. En su lugar, debemos utilizar un enfoque ligeramente diferente, en el que la cantidad que realmente optimizamos es aún más diferente de la cantidad que realmente queremos optimizar.

Descenso estocástico del gradiente

El Descenso Gradiente Estocástico (SGD) y sus variantes son probablemente el algoritmo de optimización más utilizado para el aprendizaje automático en general y para el aprendizaje profundo en particular. Es muy similar al descenso de gradiente (por lotes), excepto que utiliza un estimador estocástico (es decir, ruidoso) del gradiente para realizar su actualización. En el caso del aprendizaje automático, esto se obtiene normalmente mediante el muestreo de uno o un pequeño subconjunto de m de los ejemplos de entrenamiento y calculando su gradiente, como se muestra en el Algoritmo 8.1. Cuando los ejemplos son i.i.d., significa que el valor esperado $E[\hat{g}]$ de este gradiente estimado (promediando sobre diferentes extracciones de los ejemplos utilizados para calcular el gradiente estimado) es

igual al gradiente verdadero. Por lo tanto, el estimador del gradiente es insesgado:

$$E[\hat{g}] = g$$

donde g es el gradiente total.
Cuando $m = 1$, el algoritmo 8.1 se denomina descenso de gradiente en línea. Cuando $m > 1$ pero m es

Algorithm 1 Actualización del descenso de gradiente estocástico (SGD) en la iteración de entrenamiento k

Require: Ritmo de aprendizaje η.
Require: Parámetro inicial Θ.
while Criterio de parada no cumplido **do**
  Muestrear un minilote de m ejemplos del conjunto de entrenamiento $\{x^{(1)}, \cdots, x^{(m)}\}$.
  Fijar $\hat{g} = 0$
  for $i = 1$ hasta m **do**
    Calcular el gradiente estimado: $\hat{g} \longleftarrow \hat{g} + \Delta_\Theta L(f(x^{(i)}; \Theta), y^{(i)})/m$
  end for Aplicar actualización: $\Theta \longleftarrow \Theta_k - \eta\hat{g}$
end while

una fracción del número de ejemplos de entrenamiento, este algoritmo se denomina a veces minibatch SGD.
Un hiperparámetro crucial que se introduce cuando se aplica el SGD es la tasa de aprendizaje (η_k en el Algoritmo 8.1). Mientras que el descenso de gradiente ordinario puede funcionar con una tasa de aprendizaje fija, es necesario permitir que la tasa de aprendizaje de SGD disminuya a un ritmo adecuado durante el entrenamiento, si se quiere converger a un mínimo. Esto se debe a que el estimador del gradiente SGD introduce una fuente de ruido que no se convierte en 0 ni siquiera cuando llegamos a un mínimo (mientras que el verdadero gradiente se hace pequeño y luego 0 cuando nos acercamos a un mínimo y lo alcanzamos). Una condición suficiente, pero no necesaria, para garantizar la convergencia es que

$$\begin{aligned} \sum_{k=1}^{\infty} \eta_k &= \infty, \text{y} \\ \sum_{k=1}^{\infty} \eta_k^2 &< \infty \end{aligned} \tag{5.9}$$

La propiedad más importante del SGD y de la optimización en línea basada en el gradiente es que el tiempo de cálculo por actualización no crece con el número de ejemplos de entrenamiento. Esto permite la convergencia incluso cuando el número de ejemplos de entrenamiento se hace muy grande, llegando al límite en línea o en flujo, donde cada ejemplo sólo se ve una vez.
De nuevo, usamos k para denotar el número de iteraciones, μ el valor propio más pequeño del hessiano y L su mayor valor propio. Con SGD, el error converge en $O(1/\sqrt{k})$ en el caso convexo y en $O(1/k)$ en el caso fuertemente convexo y estos límites no se pueden mejorar a menos que se asuman condiciones adicionales.
Dado que se pueden realizar muchas más actualizaciones estocásticas por el precio de una actualización determinista, el gradiente estocástico converge inicialmente mucho más rápido el descenso de gradiente

determinista (o por lotes). Por otro lado, después de un cierto número de iteraciones, el descenso de gradiente determinista (o, de forma equivalente, el uso de minilotes cada vez más grandes) convergerá a valores más bajos de la función objetivo, debido a su mayor velocidad. Sin embargo, en las aplicaciones reales, para las condiciones en las que se entrenan grandes redes neuronales en grandes conjuntos de datos, las variantes de SGD siguen siendo la elección de los profesionales. En principio, el error de entrenamiento puede mejorarse en gran medida siguiendo el SGD por un método de optimización determinista basado en el gradiente, pero hay que tener en cuenta que puede ser a costa de un peor error de generalización.

5.2.4. LSTM Recurrent Neural Network

Las redes neuronales recurrentes [9], o RNN, son la principal herramienta para el manejo de datos secuenciales, que implican entradas o salidas de longitud variable. Para pasar de las redes multicapa a las redes recurrentes, tenemos que aprovechar una de las primeras ideas del aprendizaje automático y de los modelos estadísticos de los 80: compartir parámetros entre las diferentes partes de un modelo. La compartición de parámetros permite ampliar y aplicar el modelo a ejemplos de diferentes formas y generalizar a través de ellos. Si tuviéramos parámetros separados para cada valor del índice temporal, no podríamos generalizar a longitudes de secuencias que no se hayan visto durante el entrenamiento, ni compartir la fuerza estadística a través de diferentes longitudes de secuencias y a través de diferentes posiciones en el tiempo.
En comparación con una red multicapa, los pesos de una RNN se comparten de las neuronas artificiales, cada una de las cuales está asociada a diferentespasos de tiempo. Esto nos permite aplicar la red a secuencias de entrada de diferentes longitudes porque los mismos pesos se reutilizan en cada paso de tiempo. Esta idea se hace más explícita en los primeros trabajos sobre redes neuronales de retardo temporal [10][11], donde una red totalmente conectada se sustituye por una con conexiones locales que se comparten entre diferentes instancias temporales de las unidades ocultas.
Uno de los primeros diseños de circuitos para redes neuronales recurrentes se ilustra en la Fig. 10.3. En dicha figura se muestra la arquitectura de red recurrente más sencilla, cuyas ecuaciones se establecen a continuación en la Ec. 10.4. Este tipo de red neuronal ha demostrado ser una máquina de aproximación universal para secuencias discretas. Esto es vagamente análogo al teorema del aproximador universal para redes neuronales de avance. Específicamente, cualquier función computable por una máquina de Turing puede ser calculada por una red recurrente de tamaño finito. La salida puede leerse de la RNN después de un número de pasos de tiempo, asintóticamente lineal en el número de pasos de tiempo utilizados por la máquina de Turing y asintóticamente lineal en la longitud de la entrada [12]. Nótese que las funciones computables por una máquina de Turing son discretas, por lo que estos resultados se refieren a la implementación exacta de la función, no aproximaciones. Las salidas de la RNN deben ser discreta para que pueda tomar una secuencia binaria en la entrada y producir una secuencia binaria en la salida. Nótese también que la .$^{\text{en}}$trada"de la máquina de Turing es una especificación de la función a calcular, por lo que sólo una red recurrente de tamaño finito es suficiente para todos los problemas. La RNN puede simular una pila ilimitada representando sus activaciones y pesos con números racionales de precisión no limitada.

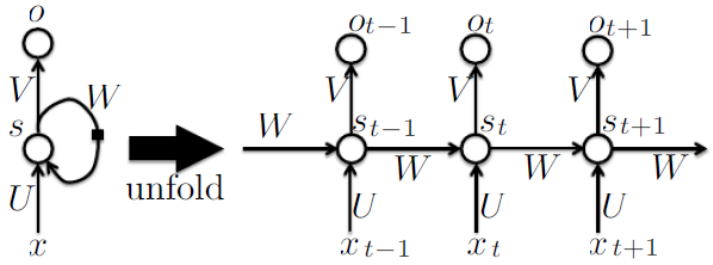

Figura 5.7: Izquierda: circuito de red recurrente ordinario con recurrencia de oculto a oculto, visto como un circuito, con matrices de pesos U, V, W para los tres tipos diferentes de conexiones (de entrada a oculto, de oculto a salida y de oculto a oculto, respectivamente). Cada círculo indica un vector completo de activaciones. Derecha: lo mismo visto como un gráfico de flujo en el que cada nodo está asociado a una instancia temporal concreta.

La red recurrente de la Fig. 10.3 corresponde a las siguientes ecuaciones de propagación hacia delante si asumimos que se utilizan no linealidades tangentes hiperbólicas en las unidades ocultas y $softmax$ en la salida,

$$\begin{aligned} a_t &= b + W s_{t-1} + U x_t \\ s_t &= tanh(a_t) \\ o_t &= c + V s_t \\ p_t &= softmax(o_t) \end{aligned} \tag{5.10}$$

donde los parámetros son los vectores de sesgo b y c junto con las matrices de pesos U, V y W , respectivamente para las conexiones de entrada a oculto, oculto a salida y oculto a oculto. Este es un ejemplo de una red recurrente que mapea una secuencia de entrada a una secuencia de salida de la misma longitud. La pérdida total para un par de secuencias de entrada/objetivo (x, y) sería la suma de las pérdidas sobre todos los pasos de tiempo, por ejemplo:

$$L(x, y) = \sum_t L_t = \sum_t -log p_{t,y_t} \tag{5.11}$$

donde y_t es la categoría que debe asociarse al paso de tiempo t en la secuencia de salida.
El forzamiento del profesor (*Teacher forcing*) es el proceso de entrenamiento en el que las entradas retroalimentadas no son las salidas predichas sino los propios objetivos, como se ilustra en la Fig. 10.5. La desventaja del forzamiento estricto del profesor es que si la red se va a utilizar posteriormente en modo de bucle abierto, es decir, con las salidas de la red (o muestras de la distribución de salida) alimentadas como entrada, entonces el tipo de entradas que la red habrá visto durante el entrenamiento podría ser muy diferente al tipo de entradas que verá en el momento de la prueba cuando la red se ejecute en modo generativo, pudiendo potencialmente, producir generalizaciones muy pobres. Una forma de mitigar este problema es entrenar tanto con entradas forzadas por el profesor como con entradas libres, por ejemplo, prediciendo el objetivo correcto un número de pasos en el futuro a través de las trayectorias recurrentes desplegadas de salida a entrada. De este modo, la red puede aprender a tener en cuenta condiciones de entrada (como las que genera ella misma en el modo de ejecución libre)

no durante el entrenamiento y a volver a mapear el estado para que la red genere salidas adecuadas después de algunos pasos.

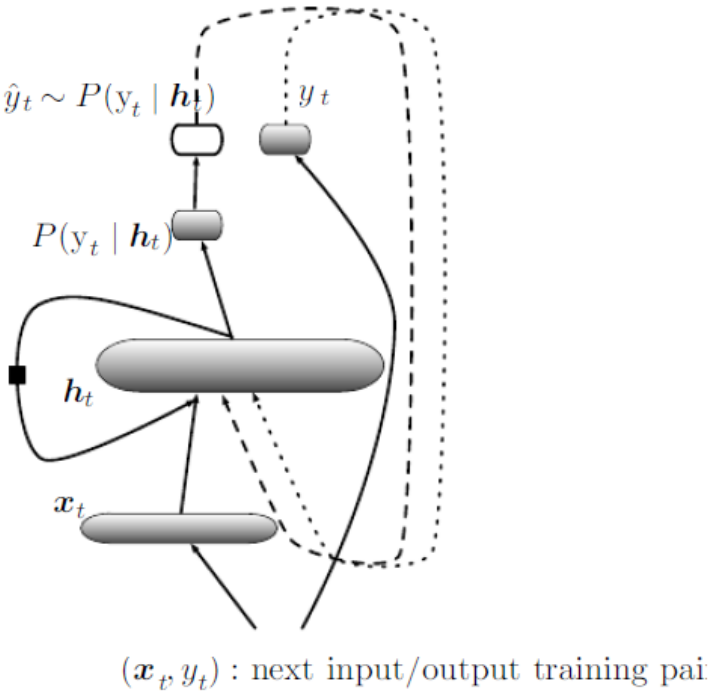

Figura 5.8: Ilustración del forzamiento del profesor para las RNN, que surge naturalmente del objetivo de entrenamiento de log-verosimilitud

Uno de los atractivos de las RNN es la idea de que puedan conectar la información anterior con la tarea actual, como por ejemplo, que el uso de fotogramas de vídeo anteriores pueda informar sobre la comprensión del fotograma actual. Si las RNN pudieran hacer esto, serían extremadamente útiles. ¿Pero pueden hacerlo? Depende.
A veces, sólo necesitamos mirar la información reciente para realizar la tarea actual. Por ejemplo, consideremos un modelo lingüístico que intenta predecir la siguiente palabra basándose en las anteriores. Si intentamos predecir la última palabra de "las nubes están en el", no necesitamos más contexto: es bastante obvio que la siguiente palabra será cielo. En estos casos, en los que la distancia entre la información relevante y el lugar en el que se necesita es pequeña, las RNN pueden aprender a utilizar la información pasada.
Pero también hay casos en los que se necesita más contexto. Considere la posibilidad de intentar predecir la última palabra del texto "Me crié en Francia... hablo francés con fluidez". La información reciente sugiere que la siguiente palabra es probablemente el nombre de un idioma, pero si queremos acotar qué idioma, necesitamos el contexto de Francia, desde más atrás. Es posible que la brecha entre la información relevante y el punto en el que se necesita sea muy grande.
Por desgracia, a medida que esa brecha crece, las RNN se vuelven incapaces de aprender a conectar la información.

En teoría, las RNN son absolutamente capaces de manejar esas "dependencias a largo plazo". Un humano podría elegir cuidadosamente los parámetros para que resolvieran problemas de juguete de este tipo. Lamentablemente, en la práctica, las RNN no parecen ser capaces de aprenderlas. El problema fue explorado en [12] y [13] encontraron algunas razones fundamentales por las que podría ser difícil.

Las redes de memoria a largo plazo

Las redes de memoria a largo plazo (normalmente llamadas "LSTM") son un tipo especial de RNN, capaces de aprender dependencias a largo plazo. Fueron introducidas en [14] y fueron refinadas y popularizadas en trabajos posteriores. Funcionan tremendamente bien en una gran variedad de problemas y ahora son ampliamente utilizadas.

Las LSTM están diseñadas explícitamente para evitar el problema de la dependencia a largo plazo. Recordar información durante largos periodos de tiempo es prácticamente su comportamiento por defecto, no es algo que les resulte difícil de aprender a estos tipos de modelos.

Todas las redes neuronales recurrentes tienen la forma de una cadena de módulos de red neuronal que se repiten. En las RNN estándar, este módulo de repetición tendrá una estructura muy simple, como por ejemplo una sola capa *tanh*.

Las LSTM también tienen esta estructura en cadena, pero el módulo de repetición tiene una estructura diferente. En lugar de tener una sola capa de red neuronal, hay cuatro, que interactúan de una manera muy especial.

La clave de las LSTM es el estado de las celdas. El estado de la celda es una especie de cinta transportadora. Corre en línea recta por toda la cadena, con sólo algunas interacciones lineales menores. Es muy fácil que la información fluya a lo largo de ella sin cambios.

La LSTM tiene la capacidad de eliminar o añadir información al estado de la célula, cuidadosamente regulada por estructuras llamadas puertas. Las puertas son una forma de dejar pasar información de forma opcional. Están compuestas por una capa de red neuronal sigmoide y una operación de multiplicación escalar. La capa sigmoide emite números entre cero y uno, que describen la cantidad de cada componente que se debe dejar pasar. Un valor de cero significa "no dejar pasar nada", mientras que un valor de uno significa "dejar pasar todo".

Una LSTM tiene tres de estas puertas, para proteger y controlar el estado de las células.

5.2.5. Convolutional Neural Network

En el ámbito del aprendizaje profundo, una red neuronal convolucional (CNN o ConvNet) es una clase de red neuronal artificial que se aplica con mayor frecuencia al análisis de imágenes visuales, procesamiento del lenguaje natural y series temporales financieras. También se conocen como redes neuronales artificiales invariantes en el desplazamiento o en el espacio (SIANN), están basadas en la arquitectura de pesos compartidos de los núcleos o filtros de convolución que se deslizan a lo largo de las características de entrada y proporcionan respuestas equivariantes a la traslación conocidas como mapas de características. De forma contraria a la intuición, la mayoría de las redes neuronales convolucionales sólo son equivariantes, en lugar de invariantes, a la traducción.

Las CNN son versiones regularizadas de los perceptrones multicapa. Los perceptrones multicapa suelen ser redes totalmente conectadas, es decir, cada neurona de una capa está conectada a todas las neuronas de la capa siguiente. La çonectividad total"de estas redes las hace propensas a sobreajustar los datos. Las formas típicas de regularización, o de evitar el sobreajuste, incluyen: penalizar los parámetros durante el entrenamiento (decaimiento del peso) o recortar la conectividad (conexiones

omitidas, abandonos, etc.) Las CNN adoptan un enfoque diferente hacia la regularización: aprovechan el patrón jerárquico de los datos y ensamblan patrones de complejidad creciente utilizando patrones más pequeños y sencillos grabados en sus filtros. Por tanto, en una escala de conectividad y complejidad, las CNN se encuentran en el extremo inferior.
Las CNN utilizan relativamente poco preprocesamiento en comparación con otros algoritmos de clasificación de imágenes. Esto significa que la red aprende a optimizar los filtros (o kernels) mediante un aprendizaje automatizado, mientras que en los algoritmos tradicionales estos filtros se diseñan a mano. Esta independencia de los conocimientos previos y de la intervención humana en la extracción de características es una gran ventaja.
El nombre red neuronal convolucionalïndica que la red emplea una operación matemática llamada convolución. Las redes convolucionales son un tipo especializado de redes neuronales que utilizan la convolución en lugar de la multiplicación matricial general en al menos una de sus capas. A continuación, vamos a definir brevemente la operación de la convolución.
La convolución de f en g se escribe como $f * g$, denotando el operandor con el símbolo $*$. Se define como la integral del producto de las dos funciones después de invertir y desplazar una. Así pues, se trata de un tipo particular de transformación integral:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Aunque el símbolo t se utiliza más arriba, no es necesario que represente el dominio del tiempo. Pero en ese contexto, la fórmula de convolución puede describirse como el área bajo la función $f(\tau)$ ponderada por la función $g(-\tau)$ desplazada por la cantidad t. A medida que t cambia, la función de ponderación $g(t-\tau)$ enfatiza diferentes partes de la función de entrada $f(\tau)$.
Para las funciones f, g soportadas sólo en $[0, \infty)$ (es decir, cero para los argumentos negativos), los límites de integración pueden ser truncados, resultando:

$$(f * g)(t) := \int_{0}^{t} f(\tau)g(t-\tau)d\tau \text{para} f, g : [0, \infty) \longrightarrow R$$

La convolución aprovecha tres ideas importantes que pueden ayudar a mejorar un sistema de aprendizaje automático: las interacciones dispersas, la compartición de parámetros y las representaciones equivariantes. Además, la convolución proporciona un medio para trabajar con entradas de tamaño variable. A continuación describimos cada una de estas ideas.
Las capas tradicionales de las redes neuronales utilizan una multiplicación matricial para describir la interacción entre cada unidad de entrada y cada unidad de salida. Esto significa que cada unidad de salida interactúa con cada unidad de entrada. Sin embargo, las redes convolucionales suelen tener interacciones dispersas (también denominadas conectividad dispersa o pesos dispersos). Esto se consigue haciendo el núcleo más pequeño que la entrada. Por ejemplo, al procesar una imagen, la imagen de entrada puede tener miles o millones de píxeles, pero podemos detectar rasgos pequeños y significativos como los bordes con núcleos que sólo ocupan decenas o cientos de píxeles. Esto significa que necesitamos almacenar menos parámetros, lo que reduce los requisitos de memoria del modelo y mejora su eficacia estadística. También significa que el cálculo de los resultados requiere menos operaciones. Estas mejoras en la eficiencia suelen ser bastante grandes. Si hay m entradas y n salidas, la multiplicación de matrices requiere mn parámetros y los algoritmos utilizados en la práctica tienen un tiempo de

ejecución de $O(mn)$. Si limitamos el número de conexiones que puede tener cada salida a k, entonces el enfoque de conexión dispersa requiere sólo kn parámetros y un tiempo de ejecución de $O(kn)$. Para muchas aplicaciones prácticas, es posible obtener un buen rendimiento en la tarea de aprendizaje automático manteniendo k varios órdenes de magnitud más pequeños que m. Para una demostración gráfica de la conectividad dispersa vease la figura 5.7.

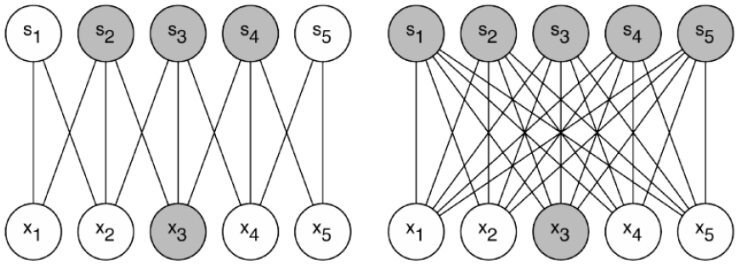

Figura 5.9: Conectividad dispersa, vista desde abajo: Destacamos una unidad de entrada, X_3, y también resaltamos las unidades de salida en S que se ven afectadas por esta unidad. (Izquierda) Cuando S se forma por convolución con un núcleo de anchura 3, sólo tres salidas se ven afectadas por X_3. (Derecha) Cuando S se forma por multiplicación matricial, la conectividad ya no es escasa, por lo que todas las salidas se ven afectadas por X_3.

La compartición de parámetros se refiere al uso del mismo parámetro para más de una función en un modelo. En una red neuronal tradicional, cada elemento de la matriz de pesos se utiliza exactamente una vez al calcular la salida de una capa. Se multiplica por un elemento de la entrada y nunca se vuelve a consultar. Como sinónimo de compartir parámetros, se puede decir que una red tiene pesos ligados, porque el valor del peso aplicado a una entrada está ligado al valor de un peso aplicado en otra parte. En una red neuronal convolucional, cada miembro del núcleo se utiliza en cada posición de la entrada. La compartición de parámetros utilizada por la operación de convolución significa que en lugar de aprender un conjunto separado de parámetros para cada lugar, aprendemos sólo un subconjunto. Esto no afecta al tiempo de ejecución pero reduce aún más los requisitos de almacenamiento del modelo a k parámetros. Dado que m y n suelen tener más o menos el mismo tamaño, k es prácticamente insignificante en comparación con mn. Por lo tanto, la convolución es mucho más eficiente que la multiplicación de matrices densas en términos de requisitos de memoria y eficiencia estadística. Para una representación gráfica de cómo funciona el reparto de parámetros, véase Fig. 5.8.

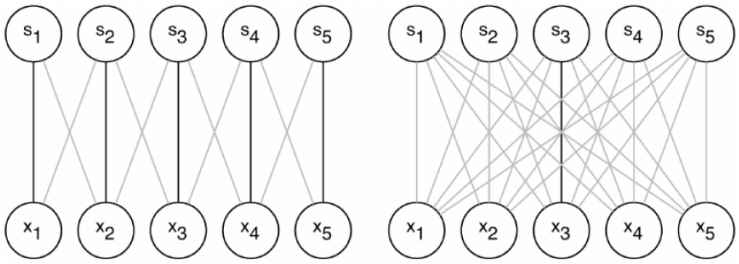

Figura 5.10: Compartición de parámetros: Destacamos las conexiones que utilizan un parámetro concreto en dos modelos diferentes. (Izquierda) Destacamos los usos del elemento central de un núcleo de 3 elementos en un modelo convolucional. Debido a la compartición de parámetros, este único parámetro se utiliza en todas las posiciones de entrada. (Derecha) Destacamos el uso del elemento central de la matriz de pesos en un modelo totalmente conectado. En este modelo no se comparten los parámetros, por lo que el parámetro se utiliza sólo una vez.

En el caso de la convolución, la forma particular de compartir parámetros hace que la capa tenga una propiedad llamada equivarianza a la traslación. Decir que una función es equivariante significa que si la entrada cambia, la salida cambia de la misma manera. En concreto, una función $f(x)$ es equivariante respecto a una función g si $f(g(x)) = g(f(x))$. En el caso de la convolución, si dejamos que g sea cualquier función que traduzca la entrada, es decir la desplace, entonces la función de convolución es equivariante de g. Por ejemplo, definamos $g(x)$ tal que para todo i, $g(x)[i] = x[i-1]$. Esto desplaza cada elemento de x una unidad a la derecha. Si aplicamos esta transformación a x y luego aplicamos la convolución el resultado será el mismo que si aplicamos la convolución a x, y luego aplicamos la transformación a la salida. Al procesar datos de series temporales, esto significa que la convolución produce una especie de línea de tiempo que muestra cuándo aparecen las diferentes características en la entrada. Si movemos un evento más tarde en el tiempo en la entrada, la misma representación exacta del mismo aparecerá en la salida, sólo que más tarde en el tiempo.
Por último, algunos tipos de datos no pueden ser procesados por redes neuronales definidas por multiplicación matricial con una matriz de forma fija. La convolución permite procesar algunos de estos tipos de datos.
Una capa típica de una red convolucional consta de tres etapas (véase la Fig. 5.9).

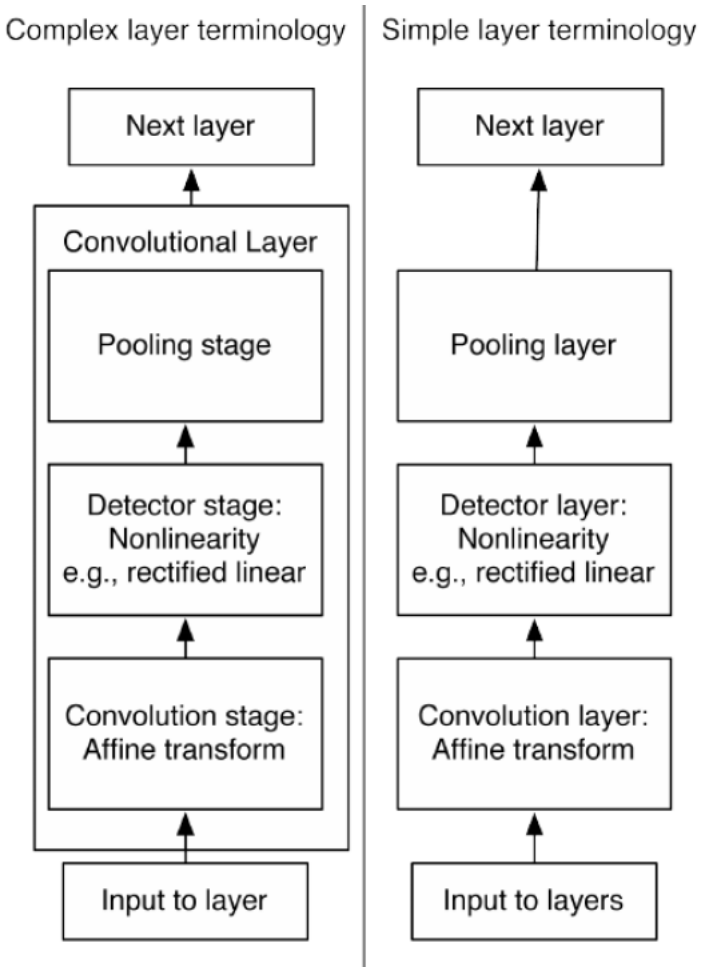

Figura 5.11: Las etapas de una capa típica de una red neuronal convolucional.

En la primera etapa, la capa realiza varias convoluciones en paralelo para producir un conjunto de activaciones presinápticas. En la segunda etapa, cada activación presináptica se somete a una función de activación no lineal, como la función de activación lineal rectificada. Esta etapa se denomina a veces etapa de detección. En la tercera etapa utilizamos una función de agrupación para modificar aún más la salida de la capa.
Una función de agrupación sustituye la salida de la red en un lugar determinado por una estadística resumida de las salidas cercanas. Por ejemplo, la operación de agrupación máxima informa de la salida máxima dentro de una zona rectangular. Otras funciones de agrupación populares incluyen la media de un vecindario rectangular, la norma L2 de un vecindario rectangular, o una media ponderada basada en la distancia desde el píxel central.
En todos los casos, la agrupación ayuda a que la representación sea invariable a pequeñas traslaciones de la entrada. Esto significa que si traducimos la entrada en una pequeña cantidad, los valores de la mayoría de las salidas agrupadas no cambian. La invarianza a la traslación local puede ser una propiedad muy útil si nos importa más saber si alguna característica está presente que saber exactamente dónde

está. Cuando esta suposición es correcta, puede mejorar en gran medida la eficiencia estadística de la red.

Capítulo 6

Implementación de los modelos

En este capítulo vamos a proceder con la implementación de los modelos que hemos desarrollado en el capítulo anterior. Recordemos que estas implementaciones se han efectuado en Python, los códigos de programación se adjuntarán en el anexo del trabajo.

6.1. Ánalisis técnico

A continuación, presentaremos unos gráficos en los que podremos apreciar los resultados obtenidos para las distintas estrategias de cada uno de los indicadores técnicos descritos en el capítulo 4.
Pero antes queremos destacar que para la implementación de las estrategias basadas en los indicadores de análisis técnico se ha utilizado la librería **ta** de Python.
El desarrollo de esta librería se produjo debido a problemas relacionados con la aplicación de Machine Learning a series temporales en un contexto financiero. La librería **ta** tiene como base una de las librerías más usadas de Python, **Pandas**. El uso de esta librería o biblioteca está orientado a los científicos que trabajan con bases de datos financieros (con columnas como "Volume", "High", "Low", etc) y que usan herramientas de Machine Learning/Deep Learning del ecosistema Python para predecir el valor de un activo financiero en el futuro.
Actualmente, la biblioteca tiene implementados 32 indicadores, que dan como resultado 58 características. En todos ellos se puede configurar el tamaño de la/s ventana/s a analizar, las constantes a usar o el rellenado automático de un modo inteligente de los valores NaN producidos por los indicadores.
Con todo ello ahora si adjuntamos las gráficas que se han obtenido. Empezaremos con la gráfica que hemos obtenido para el indicador técnico MACD,

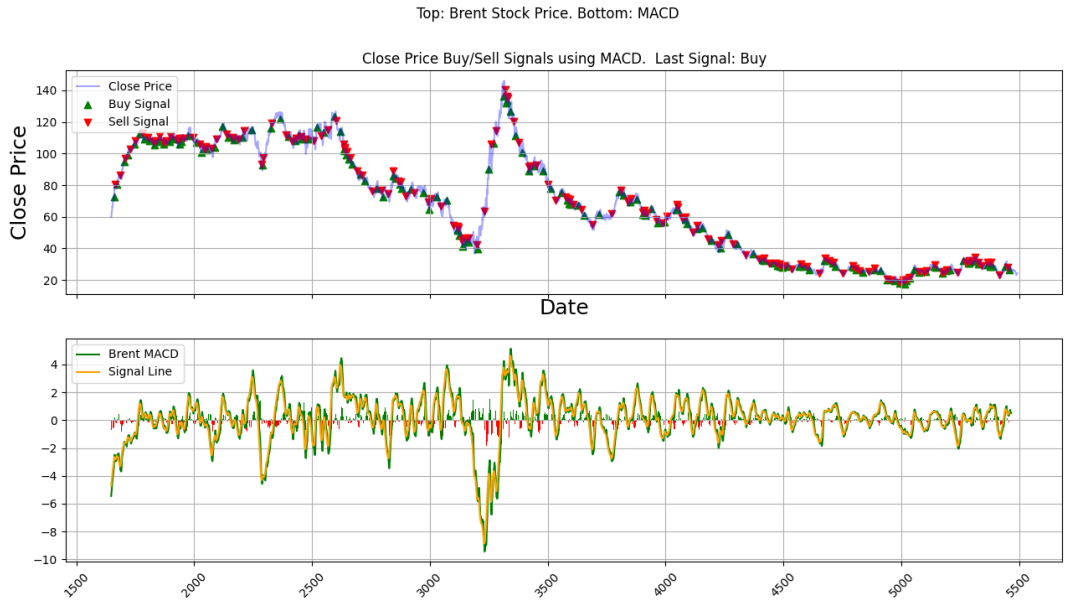

Figura 6.1:

Como podemos observar la gráfica consta de dos gráficos. En uno de ellos podemos observar la evolución de los precios de cierre del Brent, en la submuestra de entrenamiento, junto a las señales de compra/venta, obtenidas bajo la estrategia propuesta. En el otro gráfico se observa la evolución del valor MACD asociado al brent y la línea señal.
A continuación, veamos la gráfica obtenida para el indicador técnico RSI,

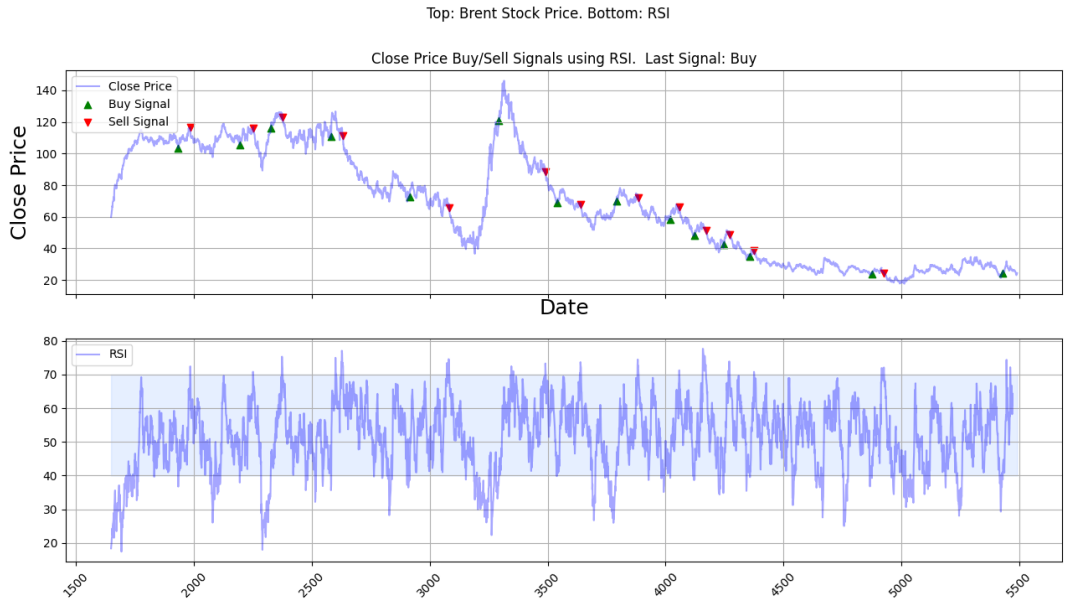

Figura 6.2:

Al igual que en la gráfica anterior consta de dos gráficos. Uno con la evolución del precios y las señales de compra/venta. Otro gráfico con la evolución del valor del indicador técnico RSI.
Por último, veamos la gráfica obtenida para el indicador Bollinger Bands,

Figura 6.3:

En esta ocasión, al igual que en las anteriores, nuestra gráfica esta formada por un gráfico en el que se oberva la evolución del precio junto a las señales de compra/venta y otro gráfico con la evolución de las tres bandas de Bollinger.

6.2. LSTM Recurrent Neural Network

Para la implementación de nuestro modelo LSTM principalmente vamos a utilizar tres librerías. Estas son: **Keras**, **Sklearn** y **NumPy**. A continuación vamos a realizar una breve descripción de cada una de ellas.
Keras es una biblioteca de Redes Neuronales de código abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Keras contiene varias implementaciones de los bloques constructivos de las redes neuronales como por ejemplo los layers, funciones objetivo, funciones de activación, optimizadores matemáticos. Además del soporte para las redes neuronales estándar, Keras ofrece soporte para las Redes Neuronales Convolucionales y para las Redes Neuronales Recurrentes.
Scikit-learn (Sklearn) es una biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión y análisis de grupos entre los cuales están: máquinas de vectores de soporte, bosques aleatorios, Gradient boosting, K-means y DBSCAN. Está diseñada para interoperar con las bibliotecas numéricas y científicas NumPy y SciPy.
NumPy es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. El uso de NumPy en Python brinda una funcionalidad comparable a MATLAB, ya que ambos se interpretan, y ambos permiten al usuario escribir programas rápidos siempre que la mayoría de las operaciones funcionen en vectores o matrices en lugar de escalares.
Ahora que ya hemos descrito las principales librerias que vamos a utilizar, vamos a explicar detalladamente la implementación del modelo. Para ello vamos a distinguir dos partes, una primera parte que corresponderá al tratamiento que precisan los datos y una segunda parte que corresponderá a la construcción del modelo en si. Por último, adjuntaremos los resultados obtenidos mediante una gráfica en la que observar las predicciones de los precios que se han obtenido bajo el modelo y algunas métricas que utilizaremos para la comparación con el otro modelo que desarrollaremos.
Empezemos explicando el tratamiento de datos que hemos efectuado para la posterior implementación del modelo. Nosotros disponemos de un Excel en el que figuran los precios de cierre, máximo, mínimo y apertura del Brent y el volumen negociado. Lo primero que hemos hecho es dividir la muestra total en dos submuestras, una que la usaremos para el entrenamiento del modelo que hemos decidido que sea aproximadamente el 70 % de la muestra total; mientras que el 30 % restante lo usaremos para el test del modelo.
Una vez hemos dividido los datos podemos pasar al verdadero tramiento de datos. Esta va a ser una parte fundamental en la implementación de nuestro modelo ya que en Machine Learning los datos deben tener una estructura determinada para poder trabajar con ellos, este tratamiento se conoce como la transformación de series temporales en un problema de aprendizaje supervisado. El último paso previo antes del tratamiento de datos es identifcar el tipo de modelo LSTM que corresponde a

nuestro caso práctico, los modelos LSTM para la predicción de series temporales se pueden clasificar de la siguiente manera:

1. Modelos LSTM univariantes
2. Modelos LSTM multivariantes
3. Modelos LSTM multipaso
4. Modelos LSTM multivariante multipaso

Así pues, el caso que nos ocupa corresponde con un modelo LSTM multivariante multipaso (problemas de previsión de series temporales multivariantes en los que la serie de salida está separada pero depende de la serie temporal de entrada, y se requieren múltiples pasos temporales para la serie de salida.) ya que nosotros utilizaremos más de un input y queremos predecir el precio de un período de más de un día.
Ahora sí vamos a explicar el tratamiento de datos necesario que hemos llevado a cabo en nuestro ejemplo, para ello vamos a realizar un ejemplo ilustrativo con un conjunto de datos más sencillo que con el que trabajaremos nosotros pero que nos servirá para explicar la manera en la que se va a trabajar con la base de datos real. Supongamos que nuestro conjunto de datos es el siguiente;

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```

Figura 6.4: Ejemplo de serie temporal dependiente multivariable.

Una serie temporal en la que las dos primeras columnas serían las entradas de nuestro modelo y la tercera columna la salida. Ahora supongamos que utilizamos tres pasos temporales previos de cada una de las dos series temporales de entrada para predecir dos pasos temporales de la serie temporal de salida. Por lo tanto la entrada será:

```
10, 15
20, 25
30, 35
```

Figura 6.5: Ejemplo de entrada de la primera muestra.

Y la salida será:

```
65
85
```

Figura 6.6: Ejemplo de salida de la primera muestra.

En nuestro caso nosotros trabajamos con los indicadores de análisi técnico como entradas de nuestro modelo, concretamente tenemos 5 columnas que corresponden a los tres indicadores que se han explicado anteriormente y una columna para la salida del modelo que corresoponderá con los precios de cierre del Brent. Nosotros hemos elegido que el tamaño del paso temporal sea igual a 100. Destacar que en nuestro ejemplo el tamaño del output es distinto al tamaño del paso temporal pero en nuestro caso práctico son iguales, es decir, tanto el tamaño del paso temporal como la longitud del vector de salida será igual a 100.
Para estrucrturar los datos de la forma que acabamos de explicar hemos implementado una función que divida los datos de la forma que nosotros queramos, en el caso del ejemplo ilustrativo que estamos describiendo el resultado de usar dicha función sería el siguiente:

```
(6, 3, 2) (6, 2)

[[10 15]
 [20 25]
 [30 35]] [65 85]
[[20 25]
 [30 35]
 [40 45]] [ 85 105]
[[30 35]
 [40 45]
 [50 55]] [105 125]
[[40 45]
 [50 55]
 [60 65]] [125 145]
[[50 55]
 [60 65]
 [70 75]] [145 165]
[[60 65]
 [70 75]
 [80 85]] [165 185]
```

Figura 6.7: Ejemplo de salida de la división en muestras de una serie temporal para la previsión de varios pasos temporales.

De esta forma tendríamos los datos de la forma necesaria para poder trabajar con ellos. A continuación, el siguiente paso será la normalización de los datos para ello utilizaremos uno de los paquetes de la librería **Sklearn**, concretamente nosotros hemos utilizado *MaxMinScaler*. La tranformación que utiliza este normalizador es la siguiente:

$$X_{std} = \frac{X - X_{min}(axis = 0)}{X_{max}(axis = 0) - X_{min}(axis = 0)}$$
$$X_{scaled} = X_{std} * (max - min) + min$$

donde max, min será el máximo y mínimo de nuetsro conjunto de datos.
Con todo ello podemos empezar a construir el modelo LSTM, para ello necesitaremos los siguientes módulos de la librería **Keras**:

- **Sequential** para inicializar la red neuronal.
- **Dense** para añadir una capa densamente conectada.
- **LSTM** para añadir una capa de memoria Largo-Corto plazo.
- **Dropout** para eliminar algunas capas y prevenir un sobreajuste.

Ahora vamos a explicar un poco más en detalle el módulo principal de nuestro modelo, que en este caso es el **LSTM**. Este módulo posee una gran cantidad de argumentos pero nosotros solo nos centraremos en algunos de ellos que serán los que modificaremos de la forma más adecuada para nuestro modelo. Los argumentos son los siguientes:

- **Units**: Número entero positivo, dimensionalidad del espacio de salida.
- **Activation**: Función de activación a utilizar.
- **Return_sequences**: Booleano. Si se devuelve la última salida en la secuencia de salida, o la secuencia completa.
- **Input_shape**: La dimensión de nuestro conjunto de entrenamiento.

En resumen, el modelo que nosotros hemos construido y que mejores resultados nos ha aportado es el siguiente: tres capas de módulos LSTM en las que cada módulo posee 50 unidades, la función de activación elegida es la función ReLu, la cual recordamos que la definimos en el capítulo 5 de nuestro trabajo, los dos primeros módulos LSTM devuelven las secuencias y la dimensión de entrada es (100,5). Además, en cada módulo LSTM hemos añadido un módulo dropout para eliminar el 20 % de las capas. Por último, hemos añadido una capa densa para la salida de nuestro modelo, la cual especifica las 100 unidades de salida que tenemos en nuestro caso.
Después de esto, compilamos nuestro modelo utilizando el popular optimizador adam (la optimización de Adam es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden) y establecemos la función de pérdida como el error cuadrático medio. Esto calculará la media de los errores al cuadrado. A continuación, ajustamos el modelo para que se ejecute en 100 épocas con un tamaño de lote de 32. El número de épocas o en inglés, *epochs*, es un hiperparámetro que define el número de veces que el algoritmo de aprendizaje trabajará a través del conjunto de datos de entrenamiento. Por otro lado el tamaño de lote o en inglés, *batch size*, es un hiperparámetro que define el número de muestras con las que se trabaja antes de actualizar los parámetros del modelo interno.
Una vez hemos construido nuestro modelo y lo hemos ajustado correctamente el siguiente paso que hemos realizado es la evaluación de dicho modelo para su posterior comparación con el otro modelo con el que trabajaremos. Para la evaluación hemos decidido usar cuatro medidas distintas. La primera medida es el porcentaje de acierto en las predicciones que se obtienen a partir del modelo que hemos ajustado, para ello hemos implementado una función que cuente el número de aciertos en las órdenes de compra/venta. La segunda medida es el beneficio que obtendríamos en el hipotético caso de realizar una inversión basándonos en las predicciones obtenidas, para ello también hemos implementado una función que calcule el supuesto beneficio. Las últimas dos medidas corresponden a dos de las medidas más usadas para modelos de predicción; el coeficiente de determinación, R^2, y el error absoluto medio, MAE. Estas dos últimas medidas no requieren de la implementación de funciones ya que por ejemplo la librería **SkLearn** ya tiene implementadas estas métricas, de todas formas recordemos las fórmulas análiticas de ambas:

- R^2: es un estadístico usado en el contexto de un modelo estadístico cuyo principal propósito es predecir futuros resultados o probar una hipótesis. El coeficiente determina la calidad del modelo

para replicar los resultados, y la proporción de variación de los resultados que puede explicarse por el modelo. La fórmula es la siguiente:

$$\rho^2 = 1 - \frac{\sigma_t^2}{\sigma^2}$$

donde σ^2 es la varianza de la variable dependiente y σ_t^2 es la varianza residual.

- MAE: es una medida de la diferencia entre dos variables continuas. Considerando dos series de datos (unos calculados y otros observados) relativos a un mismo fenómeno, el error absoluto medio sirve para cuantificar la precisión de una técnica de predicción comparando por ejemplo los valores predichos frente a los observados, el tiempo real frente al tiempo previsto, o una técnica de medición frente a otra técnica alternativa de medición. La fórmula es la siguiente:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

donde y_i y x_i son los valores predichos y los observados.

Por último, destacar que estas cuatro medidas se han evaludado tanto en el conjunto de datos del entrenamiento como en el conjunto de datos del test.

6.3. CNN Neural Network

La implementación de este segundo modelo será muy siminlar a la del modelo anterior. Por ello la gran mayoría de los procedimientos que hemos ecxplicado en el apartado anterior nos sirven para este.
Así pues, las librerías que vamos a usar son las misma que hemos detallado en la implementaion del modelo LSTM, recordemos cuales son: **Keras**, **Sklearn** y **NumPy**.
De la misma manera que hicimos hemos dividido nuestra base de dadtos en dos submuestras una que la usaremos para el entrenamiento del modelo y otra que la usaremos para el test. La proporción de la muestra es la misma 70 % para el entrenamiento y 30 % para el test.
Al igual que en el modelo LSTM el siguiente paso para la implementación de nuestro modelo CNN es determinar cual es el tipo de modelo, dentro de los modelos CNN. que se ajusta a nuestro problema. Los modelos CNN se pueden clasificar de la siguiente forma:

1. Modelos CNN univariantes
2. Modelos CNN multivariantes
3. Modelos CNN multipaso
4. Modelos CNN multivariante multipaso

El tratamiento de datos que hemos explicado en el apartado anterior es exactamente el mismo que hemos llevado a cabo en este es por ello que no se va a volver a explicar. Sin embargo, queremos destacar el hecho de que para nuestro modelo CNN no vamos a normalizar los datos, ya que hemos comprobado que los resultados que obtenemos al trbajar con los datos originales son mejores que cuando trabajamos con los datos normalizados mediante la función **MaxMinScaler**; la cual habíamos utilizado en

la implementación del modelo LSTM.
Con todo ello podemos empezar a construir el modelo CNN, este paso si que lo explicaremos detalladamente ya que es distinto a la contrucción del modelo anterior. para ello necesitaremos los siguientes módulos de la librería **Keras**:

- **Sequential** para inicializar la red neuronal.
- **Dense** para añadir una capa densamente conectada.
- **Conv1D** crea un núcleo de convolución que se convoluciona con la entrada de la capa en una sola dimensión espacial (o temporal) para producir un tensor de salidas.
- **MaxPoolind1D** disminuye la muestra de la representación de entrada tomando el valor máximo en una ventana espacial de tamaño $pool\_size$.
- **Flatten** para aplanar la entrada del modelo.

Ahora vamos a explicar un poco más en detalle el módulo principal de nuestro modelo, que en este caso es el **Conv1D**. Este módulo posee una gran cantidad de argumentos pero nosotros solo nos centraremos en algunos de ellos que serán los que modificaremos de la forma más adecuada para nuestro modelo. Los argumentos son los siguientes:

- **Filters**: Número entero positivo, dimensionalidad del espacio de salida.
- **Kernel_size**: Un entero o tupla/lista de un solo entero, que especifica la longitud de la ventana de convolución 1D.
- **Activation**: Función de activación a utilizar.
- **Input_shape**: La dimensión de nuestro conjunto de entrenamiento.

En resumen, el modelo que hemos contruido cuenta de una capa del módulo **Conv1D** en la que hemos fijado el número de filtros en 64, el tamño del núcleo igaul a 2, la función de activación es la función $ReLu$ y la dimensión de la entrada es (100,5). Además, hemos añadido una capa del módulo **MaxPooling1D** en la que el tamaño de *pooling* es dos. Por otro lado hemos añadido una capa **Flatten** y por último dos capas del módulo **Dense**, la primera de tamaño 50 con una la función de activación $ReLu$ y la segunda de tamaño la longitud de la muestra de entrenamiento.
Después de esto, compilamos nuestro modelo utilizando el popular optimizador adam (la optimización de Adam es un método de descenso de gradiente estocástico que se basa en la estimación adaptativa de momentos de primer y segundo orden) y establecemos la función de pérdida como el error cuadrático medio. Esto calculará la media de los errores al cuadrado. A continuación, ajustamos el modelo para que se ejecute en 2000 épocas.
Una vez tenemos nuestro modelo ajustado el procedimiento para su evaluación será el mismo que el del modelo del apartado anterior. Es decir, usaremos las mismas medidas que ya habiamos descrito anteriormente, recordemos que estas medidas eran: el porcentaje de aciertos en las operaciones de comra/venta, el beneficio en caso de inversión, el R^2 y el MAE. Destacar que al igual que en el modelo LSTM estas medidas han sido evaluadas tanto en el conjunto de entrenamiento como en el conjunto de test.

Capítulo 7

Conclusiones

En este último capítulo vamos a presnetar los resultados que hemos obtenido tras la implementación de los modelos que se ha descrito en el capítulo anterio. Una vez hayamos presentado los resultados los analizaremos y compararemos entre sí. Para acabar extraeremos las conclusiones pertinentes.
Antes de empezar con la presentación de los resultados recordemos que las entradas de nuestros modelos van a ser los datos que hemos obtenido tras la implementación de los indicadores de análisis técnico.
Así pues, empezaremos analizando los resultados obtenidos mediante el modelo LSTM. Para ello vamos a adjuntar una tabla en la que figuran las cuatro medidas de evaluación, descritas en el capítulo anterior.

LSTM	***Accuracy***	***Profit***	***R2***	***MAE***
Train	57 %	87.8	0.9784	1.8593
Test	52 %	-105.0	0.5890	5.3871

Cuadro 7.1: Resultados del modelo LSTM

Ahora bien, si nos fijamos en la tabla podemos observar que existe una gran diferencia entre las medidas cuando las evaluamos en el conjunto de entrenamiento o cuando las evaluamos en el conjunto test.
Si observamos los resultados obtenidos para el conjunto de datos de entrenamiento podemos decir que son resultados bastante adecuados, no obstante recordemos que nos encontramos en la muestra de entrenamiento por lo cual es lógico que los resultados sean mejores. En cuanto a las dos primeras medidas, obtenemos un 57 % de aciertos en las operaciones de compra/venta, es decir, en mas de la mitad de ocasiones las ordenes de compra o de venta son adecuadas. Por otra parte obtenemos un 87.8 de beneficio lo que indica que en el caso de seguir una estrategia basándonos en las predicciones del modelo LSTM obtendríamos 87.8€ por cada euro invertido. En cuanto a las dos últimas métricas podemos ver que tenemos un R^2 igual a 0.97 lo cual siginifica que el ajuste de nuestro modelo al precio de cierre del Brent es muy adecuado, así mismo el valor de la métrica MAE, 1.85, nos indica que en media el precio predicho dista 1.85 unidades de valor del precio observado, lo cual consideramos que es una variación pequeña.
Sin embargo, si observamos los resultados de las mismas medidas pero en la muestra del test se producen diferencias notables. En el caso del porcentaje de acierto es más o menos similar, pero si observamos el beneficio en caso de inversión pasa a ser un beneficio negativo, es decir, por cada unidad de valor

invertida perdemos 105 unidades. En la misma línea tanto el R^2 como el MAE han tenido grandes cambios. El R^2 ahora tiene un valor igual a 0.58 , lo que indica que nuestro model solo explica al rededor de un 58 % de la variabilidad de los precios de cierre del Brent. En cuanto al MAE ha aumentado hasta un 5.38 lo cual creemos que ya es un error a considerar.
A continuación, vamos a adjuntar dos gráficas en las que podremos observar los precios predichos y observados en las dos submuestras,

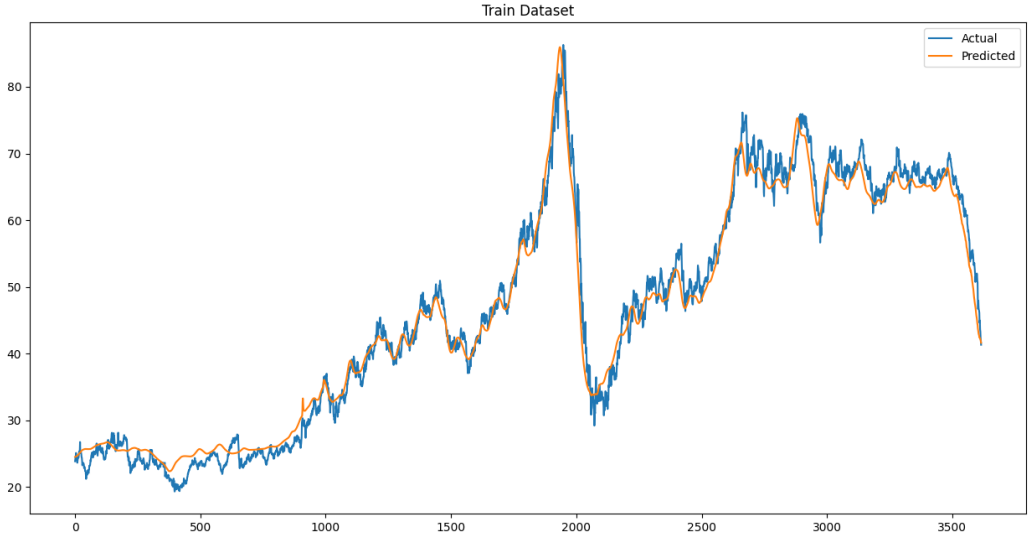

Figura 7.1: Precios predichos bajo el modelo LSTM en el conjunto de entrenamiento.

Como reflejaban los resultados de las medidas en el entrenamiento, el precio que hemos obtenido bajo el modelo es muy parecido al precio observado. Ahora veamos la gráfica pero en la muestra test,

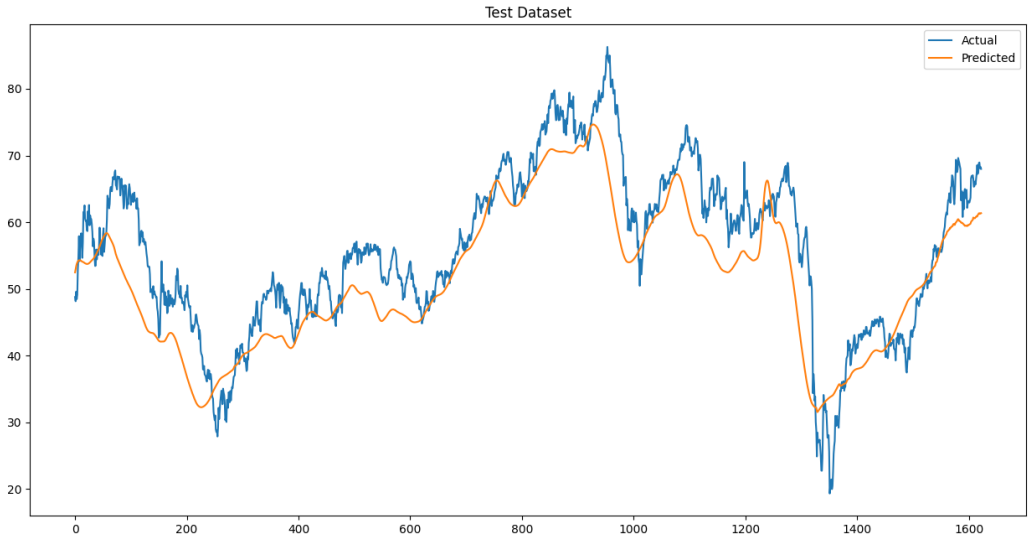

Figura 7.2: Precios predichos bajo el modelo LSTM en el conjunto de test.

Si observamos el gráfico nos damos cuenta de que, como ya reflejaban las medidas de la tabla anterior, los precios ya no son tan parecidos a los observados, ahora existen diferencias en las predicciones y podemos darnos cuenta de que los precios predichos suelen tender a ser menores que los observados.
Ahora veamos los resultados del modelo CNN, al igual que con el modelo anterior vamos a presentar una tabla en la que se recogen los resultados obtenidos.

CNN	***Accuracy***	***Profit***	***R2***	***MAE***
Train	74 %	1194.8	0.9975	1.1727
Test	74 %	541.2	0.9832	1.1355

Cuadro 7.2: Resultados del modelo CNN

Si nos fijamos en la tabla podemos observar que en esta ocasión no existe una gran diferencia entre las medidas cuando las evaluamos en el conjunto de entrenamiento o cuando las evaluamos en el conjunto test.
Para el conjunto de datos de entrenamiento en general podemos decir que son resultados bastante adecuados. En cuanto a las dos primeras medidas, obtenemos un 74 % de aciertos en las operaciones de compra/venta, es decir, aproximadamente tres de cada cuatro ordenes de compra o de venta son adecuadas. Por otra parte obtenemos un 1194.8 de beneficio lo que indica que en el caso de seguir una estrategia basándonos en las predicciones del modelo CNN obtendríamos 1194.8 unidades monetarias por cada unidad invertida. En cuanto a las dos últimas métricas podemos ver que tenemos un R^2 igual a 0.99 lo cual siginifica que el ajuste de nuestro modelo al precio de cierre del Brent es muy adecuado,

así mismo el valor de la métrica MAE, 1.17, nos indica que en media el precio predicho dista 1.17 unidades de valor del precio observado, lo cual consideramos que es una variación pequeña.
De la misma forma si observamos los resultados de las medidas pero en el conjunto test vemos como son resultados muy similares a los del cojunto de entrenamiento. En el caso del porcentaje de acierto es mismo, pero si observamos el beneficio en caso de inversión pasa a ser un beneficio de 541.2 unidades monetarias. El R^2 ahora tiene un valor igual a 0.98 , lo que indica que nuestro model explica al rededor de un 98 % de la variabilidad de los precios de cierre del Brent. En cuanto al MAE ha disminuido ligeramente hasta un 1.13 lo cual creemos que es un error absoluto medio bastante pequeño.
A continuación, vamos a adjuntar dos gráficas en las que podremos observar los precios predichos y observados en las dos submuestras,

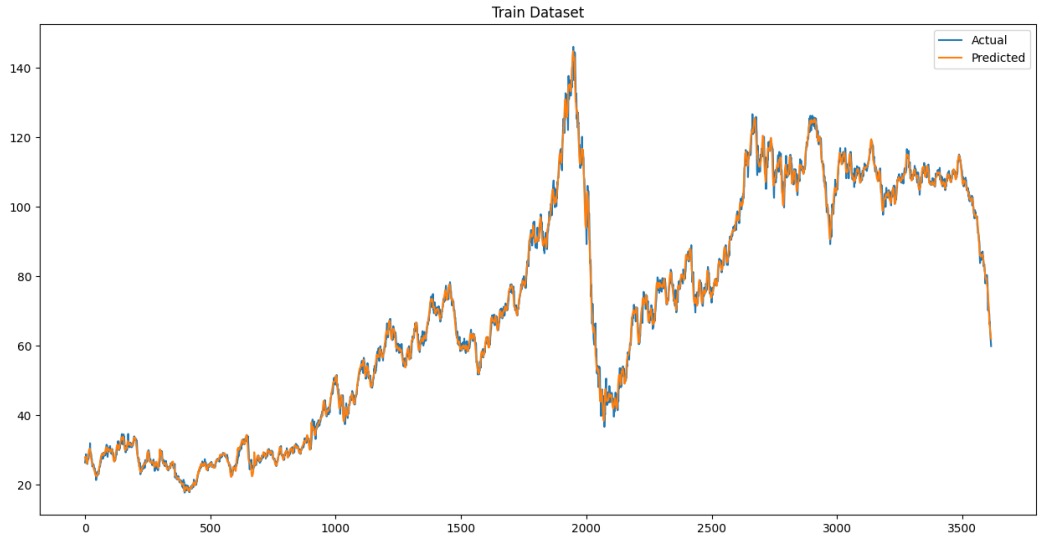

Figura 7.3: Precios predichos bajo el modelo CNN en el conjunto de entrenamiento.

AL igual que sucedia en el caso del modelo anterior los precios bajo el modelo CNN y los precios observados en la muestra de entrenamiento son muy similares, tal y como lo reflejan las medidas vistas anteriormente. AHora vamos a ver el g´rafico de la muestra de datos en el conjunto test,

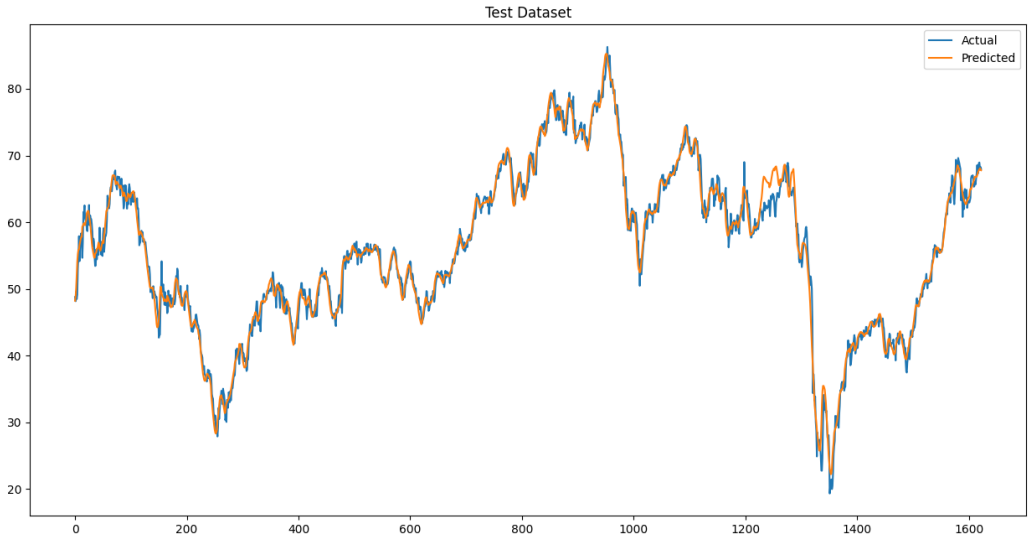

Figura 7.4: Precios predichos bajo el modelo CNN en el conjunto de test.

A diferencia de lo que ocurría en el modelo anterior esta vez los precios predichos y los observados continuan siendo muy similares, tal y como lo reflejan las medidas de la tabla anaterior.
En resumen, como hemos podido observar los resultados bajo el modelo CNN son considerablemente mejores que los resultados que hemos obtenido bajo el modelo LSTM. Por lo tanto, una de las conclusiones que obtenemos en este trabajo es que para el caso de la predcción de precios del Brent un modelo CNN parece ser más adecuado que un modelo LSTM. También destacamos el hecho de que la rapidez computacional del modelo CNN es mucho mayor al modelo LSTM.
No tenemos la suficiente información ni la certeza para afirmar que un modelo sea mejor que el otro pero en lo que respecta a nuestro caso si que podemos afirmar que aporta mejores resultados en todos los aspectos. SE plantea la posibilidad que los indicadores de análisis técnico no sean buenas entradas para un modelo LSTM o que los indicadores elegidos no sean los adecuados cuando utilizamos dicho modelo, pero como ya dijimos en su momento hemos elegido tres de los indicadorers más utilizados y más comunes en su ámbito.

Bibliografía

[1] Wei Bao, Jun Yue, and Yulei Rao. *A deep learning framework for financial time series using stacked autoencoders and long-short term memory.* PLOS ONE, 12(7):e0180944, July 2017.

[2] Shuanglong Liu, Chao Zhang, and Jinwen Ma. *Cnn-lstm neural network model for quantitative strategy analysis in stock markets.* In Neural Information Processing, pages 198–206. Springer International Publishing, 2017.

[3] Thomas Fischer and Christopher Krauss. *Deep learning with long short-term memory networks for financial market predictions.* European Journal of Operational Research, 270(2):654–669, October 2018.

[4] Bang Xiang Yong, Mohd Rozaini Abdul Rahim, and Ahmad Shahidan Abdullah. *A stock market trading system using deep neural network.* In Communications in Computer and Information Science, pages 356–364. Springer Singapore, 2017.

[5] Mor-Yosef S, Samueloff A, Modan B, Navot D, Schenker JG. *Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study.* Obstetrics and Gynecology. 1990 Jun;75(6):944-947. PMID: 2342742.

[6] Machine Learning by Tom M. Mitchell (1997-03-01)

[7] Boser et al., 1992; Cortes y Vapnik, *Incorporating invariances in support vector learning machines* Artificial Neural Networks. 1995

[8] Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1.* MIT Press, Cambridge. 159

[9] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning representations by back-propagating errors.* Nature, 323, 533–536. 159, 330

[10] Lang, K. J. and Hinton, G. E. (1988). *The development of the time-delay neural network architecture for speech recognition.* Technical Report CMU-CS-88-152, Carnegie-Mellon University. 324, 330, 356

[11] Waibel, A., Hanazawa, T., Hinton, G. E., Shikano, K., and Lang, K. (1989). *Phoneme recognition using time-delay neural networks.* IEEE Transactions on Acoustics, Speech, and Signal Processing, 37, 328–339. 330, 397, 402

[12] Siegelmann, H. and Sontag, E. (1991). *Turing computability with neural nets.* Applied Mathematics Letters, 4(6), 77–80. 333

[13] Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen.* Diploma thesis, T.U. M¨unich. 256, 353, 366

[14] Bengio, Y., Simard, P., and Frasconi, P. (1994). *Learning long-term dependencies withgradient descent is difficult.* IEEE Tr. Neural Nets. 256, 257, 353, 361, 366

[15] Hochreiter, S. and Schmidhuber, J. (1997). *Long short-term memory.* Neural Computation, 9(8), 1735–1780. 23, 360, 361