ESTIMACIÓN MEDIANTE APRENDIZAJE PROFUNDO DE LA SUPERFICIE DIARIA DE VOLATILIDAD DE UN ÍNDICE PARA SU USO EN ALGORITMOS DE INVERSIÓN

Submitted by:

Rodrigo Cepeda Marín
Hernán López Rodríguez
Rodrigo Hernández Bernardo

24 de julio de 2022

Resumen

El presente Trabajo de Fin de Máster explora las opciones financieras cotizadas para el MiniIBEX 35 y estima su superficie diaria de volatilidades implícitas, con el objetivo de encontrar ineficiencias en el mercado que permitan obtener un beneficio económico.

Esta estimación se realiza mediante una arquitectura de redes neuronales, basada en un caso especial de una combinación de convolucional y LSTM, que intenta explotar tanto información espacial como la temporal. Finalmente se comentan las limitaciones del estudio, así como ideas futuras para mejorarlo.

Índice general

Capítulo 1

Introducción

El presente Trabajo de Fin de Máster tiene como objetivo la predicción de la superficie de volatilidad de cierre diaria de un índice subyacente mediante el uso de redes neuronales profundas para su posterior utilización en algoritmos de inversión. Estos algoritmos compararán los precios de las opciones cotizadas en mercado, contra los precios estimados por la superficie de volatilidad predicha en el modelo, y tomarán posición diaria dependiendo de la diferencia observada. El trabajo constará de la siguiente estructura:

1. Inicialmente, se describirá el proceso de construcción de las superficies de volatilidad diarias a partir del precio de las opciones cotizadas en el mercado.

2. Posteriormente se introducirán y describirán las estructuras neuronales que serán utilizadas en la tarea de predicción de las superficies de volatilidad.

3. A continuación, se describirá el proceso de entrenamiento del modelo construido, así como la infraestructura elegida para dich entrenamiento.

4. Posteriormente se utilizarán los resultados obtenidos en el entrenamiento en varios algoritmos de inversión de diferente complexitud.

5. Finalmente, se presentarán las conclusiones obtenidas tras la completitud del proceso de estimación, entrenamiento y utilidazión de superficies de volatilidad como algoritmo de inversión.

Capítulo 2

Superficie de volatilidad

La superficie de volatilidad es una estructura tridimensional que recoge las volatilidades implícitas de las opciones cotizadas en mercado en función del *Strike* K y del tiempo a vencimiento T para un mismo subyacente.

En el presente trabajo, haremos uso de los precios de las opciones cotizadas en MEFF para el MiniIBEX 35 entre los años 2013 y 2021 para la costrucción de sus correspondientes matrices de volatilidad mediante diferentes métodos numéricos que serán explicados a continuación.

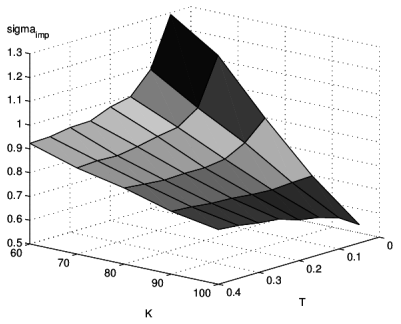


Figura 2.1: Superficie de volatilidad

El proceso que hemos seguido consta de tres pasos diferenciados, comenzando con la lectura y tratamiento del dato inicial para la obtención del precio de las opciones, continuando con el cálculo de la volatilidad en el marco de Black-Scholes y terminando por la contrucción de una matriz de volatilidad continua en función del *moneyness* m y del tiempo a *maturity* T.

2.1. Obtención de los precios diarios

Las fuentes de información utilizadas para este trabajo nos han permitido construir una base de datos que contiene el precio de la totalidad de las opciones sobre el MiniIBEX 35 cotizadas en MEFF para cada día entre 2013 y 2021. A continuación se puede ver una extracción de la base de datos para el primer día disponible:

Session date	**Mat. Date**	**Strike**	**Type of contract**	**Price**
2013-11-20	2013-11-20	10200	Call	23
2013-11-20	2013-11-20	9200	Put	90

Esta base de datos contiene más de un millón de registros para un total de 2048 días. En adelante $t_1 = 2048$.

Para cada día t, se puede construir una matriz $\mathcal{P}_{m,T}$ que recoge, para cada moneyness y para cada tiempo a vencimiento, el precio de las opciones cotizadas. Esta es una matriz incompleta ya que no existe precio para todas las posibles combinaciones de m y T que se dan en mercado.

Un extracto de la primera matriz de precios obtenida se muestra a continuación:

T/m	9700	9800	9900	10000	10100	10200	10300
2013-12-20	214.5	230.5	253	281.5	315.5	23	396.5
2014-01-17	296	143	112	87	NAN	417.5	355
2014-02-21	NAN	222	NAN	NAN	126	102	NAN
2014-03-21	330	288	250	479.5	NAN	NAN	131

La diferencia de la prima de las opciones en función del moneyness y del vencimiento hace imposible una comparación a simple vista utilizando los precios. Es por este motivo que tradicionalmente, se utiliza la volatilidad implícita de las opciones para compararlas. A continuación se explica el proceso de extracción y cálculo de la volatilidad.

2.2. Obtención de volatilidades implícitas diarias

Para obtener la volatilidad implícita de cada opción cotizada se ha aplicado el método de Newton-Raphson a una variante de Black-Scholes, modelo con el cual las opciones están *priceadas* en primer lugar.

El modelo de Black-Scholes presentado en 1976 por los economistas Fischer Black y Myron Scholes en su trabajo Scholes y Black 1973 expresaba el desarrollo del precio de una acción S como:

$$dS_t = rS_t dt + \sigma S_t dW_t$$

donde r representa el tipo de interés libre de riesgo, σ representa la volatilidad y dW representa un proceso browniano.

Sean entonces $max\{0, S_t - K\}$ y $max\{0, K - S_t\}$ opciones call y put sobre el subyacente S en tiempo T, el precio de dichos instrumentos en el universo Black-Scholes viene dado por:

$$C = S_0\phi(d_1) - Ke^{-r(T-t)}\phi(d_2)$$

$$P = Ke^{-r(T-t)}\phi(-d_2) - S_0\phi(-d_1)$$

donde

$$d_1 = \frac{\ln(\frac{S}{K}) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

El modelo de Black 76, o modelo de Black, es una variación del modelo original en el cual el subyacente es un contrato futuro F. Las fórmulas para la call $max\{0, F - K\}$ y la put $max\{0, K - F\}$ sobre un contrato de futuros vienen dadas por:

$$C = e^{-rT}(F\phi(d_1) - K\phi(d_2))$$

$$P = e^{-rT}(-F\phi(-d_1) + K\phi(-d_2))$$

donde

$$d_1 = \frac{\ln(\frac{F}{K}) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

Estas fórmulas serán las indicadas para calcular la volatilidad implícita σ en funcion del grupo de parámetros $\{C, T, r, F, K\}$ ó $\{P, T, r, F, K\}$ dependiendo del tipo de contrato.

Dada la complejidad de la fórmula, se ha utilizado el método de *Newton-Raphson* para la extracción de la volatilidad en cada opción cotizada.

El método de Newton Raphson es un procedimiento algorítmico que permite hallar raíces de funciones, conocido un valor numérico cercano a la raíz. Es un método abierto e iterativo, en general de rápida convergencia, muy útil para el cálculo de raíces cuadradas y de mayor grado, aunque para algunos casos el método presenta inconvenientes, por ejemplo si existen raíces múltiples, en este caso se tendría que aplicar diferentes soluciones para así lograr encontrar la raíz sin abandonar el método. Cantero s.f.. La fórmula, apoyándose en el Teorema de Taylor es la siguiente:

$$x_{x+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

, que adaptándose a Black-Scholes es:

$$\sigma_{n+1} = \sigma_n - \frac{e^{-rT}(F\phi(d_1) - K\phi(d_2)) - C}{K\phi(d_1)\sqrt{T}}$$

para una opción call, y

$$\sigma_{n+1} = \sigma_n - \frac{e^{-rT}(-F\phi(-d_1) + K\phi(-d_2)) - P}{K\phi(d_1)\sqrt{T}}$$

para una opción put. Siendo además el denominador, la derivada del precio con respecto de la volatilidad, es decir, la *vega* de la opción.

Tras el proceso de cálculo de iterativo de las volatilidades implícitas se obtiene una matriz para t_1 días como la que se muestra a continuación para $t_1 = 1$.

T/m	**9700**	**9800**	**9900**	**10000**	**10100**	**10200**	**10300**
2013-12-20	0.1856	0.1845	0.1830	0.1813	0.1798	0.1799	0.1787
2014-01-17	0.1837	0.1832	0.1807	0.1790	NAN	0.1737	0.1719
2014-02-21	NAN	0.1898	NAN	NAN	0.1826	0.1804	NAN
2014-03-21	0.2001	0.1979	0.1959	0.1907	NAN	NAN	0.1878

Se puede observar que la matriz continúa siendo incompleta, es por ello que a continuación se describe una técnica de interpolación y alisado para la cada superficie de volatilidad $\mathcal{V}_{t_1}$

2.3. Stochastic Volatility Inspired

El modelo *Stochastic Volatility Inspired*, SVI en adelante, fue desarrollado en *Merrill Lynch* en 1999 e inicialmente presentado al público por *Jim Gatheral* en la conferencia *Global Derivatives & Risk Management, 2004* en Madrid.

El modelo es una parametrización del smile de volatilidad implícita y permite interpolar la superficie de volatilidad discontinua extraída de mercado. Tiene dos propiedades que le otorgan gran popularidad entre los profesionales:

- Para t un tiempo a vencimiento fijo, la varianza implícita de *Black-Scholes* $\sigma^2_{BS}(k, t)$ es linear en el log-strike k mientras $|k| \to \infty$, siendo consistente con la fórmula del momento de *Roger Lee*.

- Se pueden ajustar precios de opciones listadas de manera relativamente sencilla asegurando la ausencia de arbitraje *calendar spread*.

La parametrización original de SVI viene dada por la fórmula:

$$w(k, X_R) = a + b(\rho(k-m) + \sqrt{(k-m)^2 + \sigma^2})$$

donde $w(k; X_R) = \sigma^2(K,T)T$ es la varianza dado un strike K a un vencimiento T, $k = \log(\frac{K}{F_T})$, $a \in \mathbb{R}$, $b \geq 0$, $|\rho| < 1$, $m \in \mathbb{R}$, $\sigma > 0$ y $a + b\sigma\sqrt{1-\rho^2} \geq 0$.

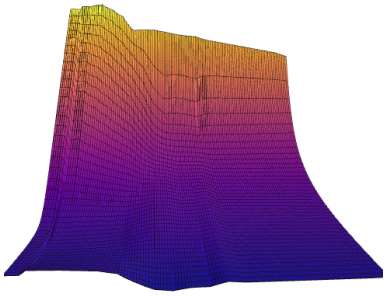


Figura 2.2: Superficie de volatilidad con SVI

De esta manera se construye, para cada T en cada día t_1, una función contínua que extrae volatilidades para cualquier moneyness introducido. Sabiendo que las varianzas se interpolan linearmente, virtualmente se ha conseguido construir una superficie de volatilidad $\mathcal{V}'_{t_1}$ de tal manera que $\exists\ \sigma_{m,T}\ \forall m, T$ y $\sigma_{m,T} \in \mathcal{V}'_{t_1}$.

Tras el proceso de interpolacion de las superficies de volatilidad para t_1, se ha obtenido una estructura de tamaño $[2048, 10, 8]$ que contiene volatilidades implícitas de t_1 días para m_k moneyness y T_k días a vencimiento, donde

$$m_k = [0{,}96; 0{,}9725; 0{,}985; 0{,}9975; 1{,}01; 1{,}0225; 1{,}035; 1{,}0475]$$

$$T_k = [60; 70; 80; 90; 100; 110; 120; 130; 140; 150]$$

.

A continuación se representan las volatilidades implícitas para todo t_1 con $m_k = 1{,}01$ y $T_k = 60$. Se puede apreciar el entorno fuertes subidas de volatilidad que ocurrieron a raiz de la pandemia del COVID a principios de 2020.

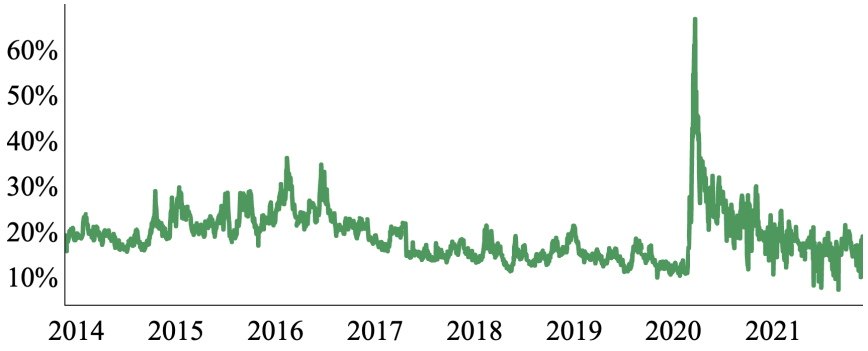


Figura 2.3: Histórico de volatilidades implícitas

Como conclusión del presente capítulo se remarca que en el proceso descrito anteriormente, se han mostrado los cálculos seguidos para obtener la estructura de información, las superficies de volatilidad diarias de cierre, que alimentarán al modelo predictivo descrito en 3 y 5.

Capítulo 3

Redes Neuronales

El presente capítulo tiene como objetivo la descripción de las características del modelo que se utilizará en la predicción de superficies de volatilidad implícita.

3.1. Redes neuronales artificiales

Las redes neuronales artificiales estándar, tmabién conocidas como redes *feed-forward* reciben este nombre debido a que la información fluye desde la función que está siendo evaluada, a través de los cálculos intermedios que definen la función y que llegan a la salida. Además, se llaman redes porque se suelen representar como una composición de funciones unidas de forma nodal. Heaton 2018. Este tipo de estructuras también se conocen como redes totalmente conectadas (*fully connected networks*), ya que cada neurona se conecta con todas las que le siguen.

La estructura básica de este tipo de redes se compone de una capa de entrada, una o varias capas ocultas intermedias y, finalmente, una capa de salida. El número de las capas ocultas determina la profundidad de la red. Además, estas capas tienen varias unidades que actúan en paralelo y que se suelen llamar neuronas o perceptrones, ya que reciben una entrada de datos de sus conexiones con las unidades previas y tratan la información a través de su propia función de activación (si esta información pasan el umbral establecido).

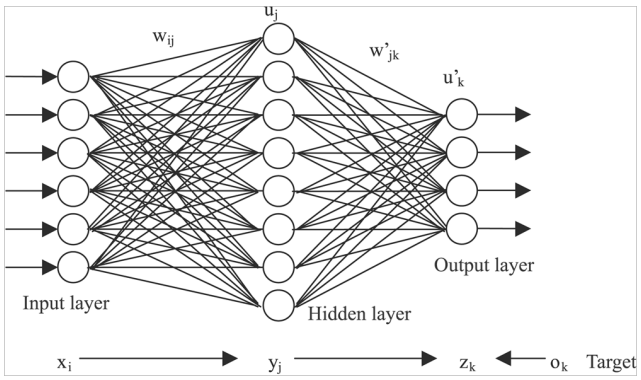


Figura 3.1: Representación de una red neuronal totalmente conectada

Posteriormente, los pesos de las neuronas se actualizan a través de un algoritmo de propagación inverso *backpropagation* que trata de minimizar el error entre la salida de la red y los datos reales. Además, dado que la red aprende sus propios pesos es

capaz de establecer qué características son relevantes para la tarea a ejecutar Medel 2016.

Estas redes se suelen utilizar en problemas de aprendizaje supervisado fundamentalmente para tareas de clasificación y predicción. En cada caso se establecen unas funciones de activación más indicadas para la tarea, especialmente en las capas de salida.

Sin embargo, este tipo de redes tienen ciertas limitaciones que las hacen poco recomendables para ciertos tipos de análisis. En concreto, para tratar información espacial y temporal que en este proyecto son de vital interés. Por suerte, se han desarrollado otros métodos que permiten explotar todas las capacidades de este tipo de redes y que, además, expanden sus capacidades para adaptarse a otro tipo de problemas.

3.2. Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal utilizada para el procesado de datos con una estructura en forma de rejilla Heaton 2018. Este tipo de redes ha sido ampliamente utilizado con éxito en muchas aplicaciones en las que existe información espacial relevante, es decir, no solo importa el dato concreto sino también su posición en el espacio. Estas redes se llaman así porque en, al menos, alguna de sus capas se utiliza una operación de convolución en vez de multiplicaciones de matrices generales. Esta operación, en su forma más general, es una operación de dos funciones que se suele denotar con un asterisco:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.1}$$

Donde x es la entrada de datos, w es el filtro o *kernel* y el resultado se le denomina mapa de características *(feature map)*.

Estas redes utilizan tres mecanismos durante el proceso de entrenamiento: campos receptivos locales, compartición de pesos y submuestreo. Los campos receptivos locales se organizan en el mencionado mapa de características donde los campos comparten los mismo pesos. Capa campo detecta patrones locales dentro de la imagen mediante la conexión de cada neurona con una región limitada de la entrada de datos, lo que explota las correlaciones espaciales de unos píxeles con sus vecinos. De esta manera, deslizar el campo receptivo local por la imagen permite encontrar las características independientemente de su posición Medel 2016. Los pesos están especificados en el filtro convolucional y se van aprendiendo junto con el sesgo (bias). Sin la entrada tiene varios canales, la neurona es la suma de las operaciones convolucionales a través de los canales de una misma región.

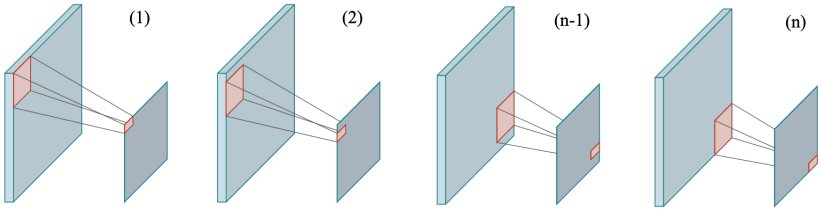


Figura 3.2: Representación de una red neuronal convolucional

Por lo general, se suelen utilizar convoluciones sobre más de un eje al mismo tiempo. En el caso del uso de imágenes se hace sobre dos ejes y, finalmente, si tenemos grupos de imágenes con una relación temporal (vídeo) se emplean los tres ejes de coordenadas.

Hay una serie de ventajas de las redes convoluciones sobre las totalmente conectadas, concretamente:

1. *Sparse connections*: Esto se debe a que el tamaño del kernel es menor al tamaño del input, lo que implica el almacenamiento de menos parámetros que, a su vez y, por tanto, una reducción importante en las exigencias de memoria y aumento en la eficiencia estadística del modelo.

2. *Parameter sharing*: Se refiere al uso de los mismos parámetros para varias funciones en el modelo. Concretamente, en una red convolucional, cada para parte del kernel se usa en cada posición del input con la excepción de los datos en los extremos dependiendo del criterio aplicado en este sentido. Por tanto, el uso de los parámetros compartidos en una red implica reusarlos en toda la función en vez de aprenderlos por separado en cada localización. Esto no afecta al tiempo de ejecución, pero reduce la exigencia de memoria del modelo, es decir, es otra vez más eficiente.

3. *Equivariance to translation*: Una función equivariante es aquella en la que si los datos de entrada cambian, los resultados cambian de la misma manera. Es decir, sea $f(x) = y$ entonces $f(x + \beta) = y + \beta$. Al reutilizar los parámetros del kernel en cada posición el modelo acaba aprendiendo las mismas representaciones independientemente de a donde se trasladen (linealmente) estas. Sin embargo, las convoluciones no son equivariantes a otros tipos de transformaciones como cambios de escala o rotaciones.

A este tipo de redes se les puede añadir una capa de agregado (*pooling*), es decir, aplica a la salida de una capa convolucional una función a un grupo de valores vecinos. Lo más común suele ser el máximo o la media de este conjunto de valores.

El objetivo de esta capa es el de conseguir que la representación sea aproximadamente invariante a pequeñas translaciones de la entrada, es decir, que si se produce una pequeña translación de los valores sobre los que se aplica esta función, el resultado agregado apenas varia. Este atributo puede ser especialmente importante en los casos en los que es más importante el hecho de que una característica esté presente a la posición exacta en la que está Heaton 2018.

Además, es también útil para evitar sobreajuste en casos con alto nivel de detalle y, en consecuencia, también reduce la complejidad del problema Medel 2016.

Por último, aparte de agregar los datos de valores vecinos, se pueden agregar las salida de diferentes convoluciones, con lo que se podría conseguir que las características de la red aprendieran a qué transformaciones hacerse invariantes.

3.3. Redes Neuronales Recurrentes

Mientras que las redes neuronales que se han presentado asumen que los datos de entrada son independientes unos de otros, las redes neuronales recurrentes (RNN) son un tipo de red neuronal enfocada al procesado de datos secuenciales. Estas redes se basan en uno de los conceptos presentados en la parte de convolucionales: el uso de parámetros compartidos, sin embargo, el proceso para compartirlos es diferente.

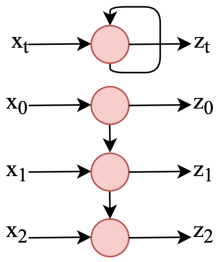


Figura 3.3: RNN

En el caso de las RNN, cada miembro de la salida se produce utilizando la misma regla de actualización aplicada a las salidas previas. De esta manera, la formulación recurrente resulta en una compartición de parámetros a través de un grafo computacional profundo. Es decir, las redes recurrentes implican la composición de la misma función una vez cada paso temporal. En particular, esta función compuesta es análogo a una multiplicación matricial.

En la práctica, las redes recurrentes funcionan en lotes de las secuencias de entrada y cada paso temporal no se asocia necesariamente con el paso del tiempo físico, es decir, se puede referir simplemente a una posición en la secuencia de datos. El principal problema matemático del aprendizaje de dependencias a largo plazo se da cuando el grafo computacional que opera a lo largo de la red es demasiado profundo. Esto ocurre especialmente en el caso de las redes recurrentes ya que, al repetir la misma operación en cada paso temporal a lo largo de una secuencia larga, se construye un grafo muy profundo. En estos casos, se podrían dar dos problemas: desvanecimiento y la explosión del gradiente. El primero dificulta la optimización de la función de coste (falta de memorización), mientras el segundo hace que el aprendizaje sea inestable. Por otra parte, asumiendo que los parámetros hacen que la red recurrente es estable (memorización de la información mientras que los gradientes no explotan), la dificultad con las dependencias a largo plazo aparece por los pesos exponencialmente menores dadas las iteraciones a largo plazo, comparado con las de corto plazo.

En la práctica, se ha demostrado que según se incrementa el espacio temporal que se quiere estudiar, la optimización basada en gradiente aumenta de dificultad. Heaton 2018.

Un modelo diseñado específicamente para abordar los problemas inherentes a las RNN son las LSTM, presentadas en Hochreiter y Schmidhuber 1997 que se explicarán a continuación. Las LSTM son un tipo de RNN que controlan el flujo de información a las neuronas ocultas y preservan las características extraídas de pasos de tiempo anteriores mientras evitan el problema del gradiente de desaparición y explosión.

3.3.1. Long Short-Term Memory LSTM

El modelo propuesto en Hochreiter y Schmidhuber 1997 aplica bucles internos para generar caminos por los que el gradiente fluye en amplios espacios de tiempo. Es decir, en vez de tener una unidad que aplica un transformación no lineal elemento a elemento, las redes recurrentes LSTM tienen celdas con recurrencia interna además de la recurrencia externa de una RNN.

A estas celdas se las dota de la habilidad para recordar u olvidar la información pasada de forma selectiva mediante el uso de tres puertas que controlan la información recibida que se transfiere después. Se utilizan tres funciones de activación en la celda para la entrada, la salida y para las funciones de activación de las puertas. A las dos primeras se les suele asignar una tanh y para la el control de las puertas siempre una sigmoide del producto escalar de la entrada y los pesos. Estas puertas tienen sus propios pesos y se comparten para cada iteración dentro de la celda. Los estados de la celda y los ocultos se controlan mediante las puertas de entrada y salida: la de entrada controla si los datos de la celda anterior se tienen en cuenta mientras que la de salida si la información de la celda se transmite al siguiente estado. Los estados de salida controlan la información que se propaga de la iteración anterior, siendo los estados

ocultos la salida real de la celda LTSM. Finalmente, en algunos casos se utiliza una conexión peephole que permiten que la celda pueda acceder y propagar información guardada de pasos anteriores, es decir, las capas de las puertas pueden ver el estado de la celda Medel 2016.

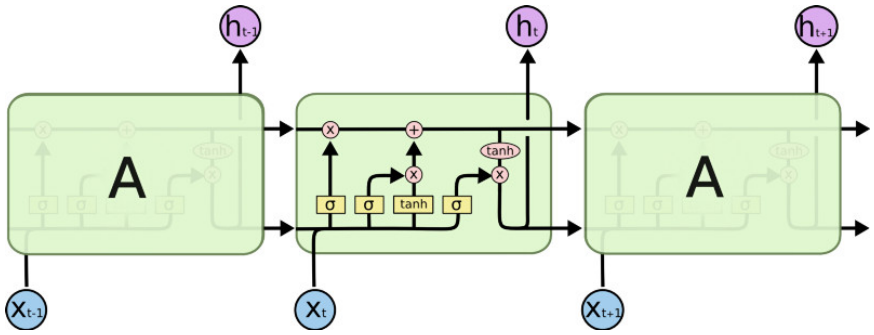


Figura 3.4: Red LSTM

Las fórmulas de las LSTM, obtenidas de pagina 410 Heaton 2018 son:

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}) \tag{3.2}$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{3.3}$$

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{3.4}$$

$$h_i^{(t)} = tanh(s_i^{(t)}) q_i^{(t)} \tag{3.5}$$

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}) \tag{3.6}$$

Para comprender con más profundidad el funcionamiento de una celda, se presenta su estructura a continuación:

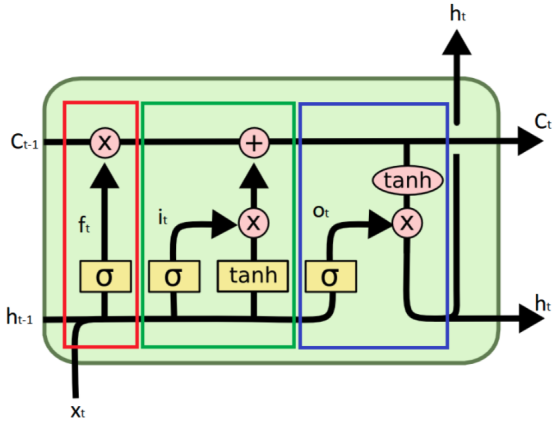


Figura 3.5: Detalle de la celda LSTM

El estado de la celda (la flecha horizontal superior) es la información que fluye y sobre la cual se producen operaciones en base a las decisiones de las puertas. Es decir, la celda tiene la habilidad de añadir o eliminar información a los datos de entrada a través de las puertas que, como ya se ha comentado, se activan en base a una sigmoide que regula el flujo tomando valores entre 0 y 1. De manera resumida este sería el proceso dentro de la celda tal y como se explica en *Understanding LSTM networks* s.f..

1. El primer paso es decidir qué información se va a descartar del estado de la celda. Como ya se ha comentado, esto se hace mediante la puerta de olvido y su activación sigmoide (parte roja de la imagen). En caso de que el valor sea 1 se mantiene toda la información del estado oculto anterior, mientras que si es 0 se descarta completamente.

2. Lo siguiente es decidir qué información se va a guardar en el estado de la celda. Este proceso se realiza en dos partes (parte verde de la imagen): primero la sigmoide de la puerta de entrada decide qué valores se van a actualizar y, después, la tanh crea un vector con los valores que son candidatos a que se añadan al estado. En el siguiente paso temporal, se combinaran los dos resultados para crear una actualización del estado.

3. Por último, se toma la decisión de la salida de la celda, que está basado en el estado de la celda pero que podrá sufrir alguna alteración (parte azul de la imagen). Para ello, una sigmoide decide de nuevo que partes del estado de la celda van a salida. Después, se pasa el estado por una tanh (para forzar los valores al rango entre -1 y 1) y se multiplica por el resultado de la sigmoide, de manera que solo se devuelve las partes activadas.

3.4. Convolucional LSTM

La capa convolucional LSTM (ConvLSTM), presentada en Shi y col. 2015 integra operaciones de convolución en la entrada de una celda LSTM sustituyendo las operaciones matriciales originales en las conexiones entrada-oculta y oculta-oculta. Como resultado, el flujo de datos que atraviesa la celda mantiene la dimensión, al contrario que en una celda LSTM estándar, en donde el resultado sale como vector.

Las ecuaciones, por tanto, son análogas a las anteriores de LSTM, pero las entradas de datos x se introducen como matrices (imágenes) en vez de vectores, y los pesos, para cada conexión, se reemplazan por filtros convolucionales. En la fórmulas, $*$ denota un operación de convolución y $\circ$ el producto Hadamard:

$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \tag{3.7}$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \tag{3.8}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \tag{3.9}$$

$$o_t = \sigma(W_{xo} * X_t + W_h o * H_{t-1} + W_c o \circ C_t + b_o) \tag{3.10}$$

$$H_t = o_t \circ tanh(C_t) \tag{3.11}$$

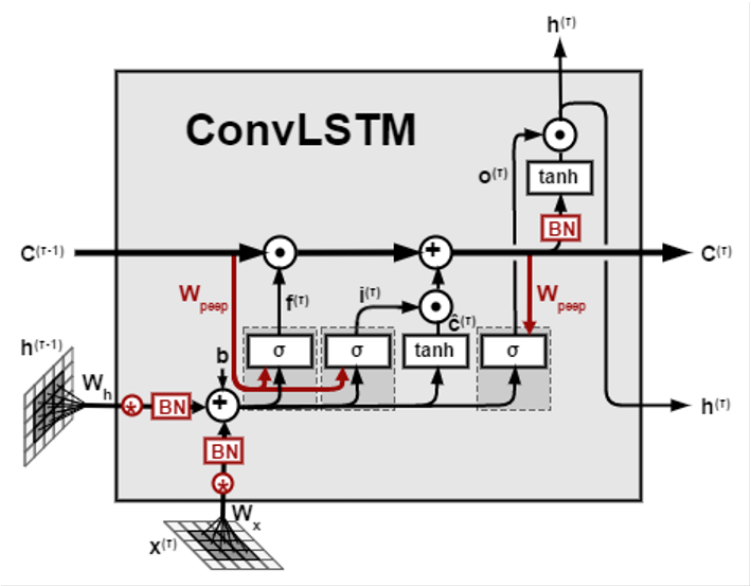


Figura 3.6: Detalle de la celda ConvLSTM

En esta arquitectura de celda, los filtros de las conexiones entrada-oculta determinan la resolución del mapa de características, y el tamaño del filtro de la conexión oculta-oculta establece la información agregada que la celda recibe de la iteración anterior. La transición de estados en la unidad ConvLSTM se puede interpretar como un movimiento de secuencias, al igual que en un vídeo. En consecuencia, los filtros transicionales largos capturan movimientos más rápidos mientras los cortos captan acciones más lentas Medel 2016.

Capítulo 4

Motivación

El objetivo del presente trabajo es evaluar la conveniencia del uso de modelos basados en inteligencia artificial y, más concretamente, de redes neuronales artificiales para la valoración de opciones a través del cálculo de la volatilidad implícita.

Premisas similares han sido presentadas con anterioridad, en concreto Liu, Oosterlee y Bohte 2019 plantearon la sustitución de los métodos actuales de valoración de volatilidades por el uso de redes neuronales artificiales, resultando esto en un menor tiempo de computación.

Otro estudio basado en redes neuronales convolucionales (CNN) concluye que estas son mejores que los modelos paramétricos tradicionales estimando la volatilidad implícita y el precio de las opciones Wei y col. 2020.

Siguiendo el rastro del uso de redes convolucionales, estas se han utilizado también junto con una estructura *Long-Short Term Memory* (LSTM) para la predicción de matrices de covarianzas en Fang, Yu y Tang 2021. Esta propuesta es especialmente interesante ya que se plantea el uso de esta arquitectura de red para explotar tanto patrones espaciales (las covarianzas dispuestas en una matriz) como temporales (cada matriz se corresponde con un día de cotización). Esto, a priori, parece un buen punto de partida para empezar con el problema que se ha elegido. En la siguiente imagen se muestra un esquema de la arquitectura desarrollada en este artículo.

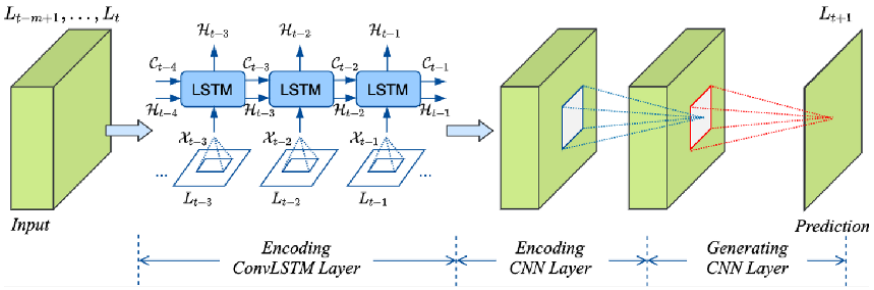


Figura 4.1: Arquitectura ConvLSTM Fang, Yu y Tang 2021

Respecto a los modelos basados en redes recurrentes LSTM y convolucionales, no hay mucho publicado y la práctica totalidad está enfocada al tratamiento de imágenes e incluso vídeos. No olvidemos que nuestros datos se estructuran como una sucesión temporal de matrices de volatilidades para todas las opciones del mercado, es decir, una secuencia de matrices que forman un prisma. Y esta es matemáticamente la misma estructura que una secuencia ordenada de imágenes, es decir, un vídeo. La estructura de entrada y salida de los datos en el modelo será como se muestra en

4.2, donde se describe la arquitectura de entrada que conforman los datos de nuestro problema.

Como hemos descrito en 2, nuestros datos están conformados for $t_1 = 2048$ superficies de volatilidad. Se utilizarán las t_2 superficies anteriores para estimar la superficie en t, dando esto lugar a $t_1 - t_2$ cubos de volatilidad que tendrán como salida $t_1 - t_2$ matrices. La combinación de la predicción de las estructuras temporales y espaciales es necesaria en la búsqueda de la matriz de volatilidades implícitas en t_1+1 utilizando t_2 superficies.

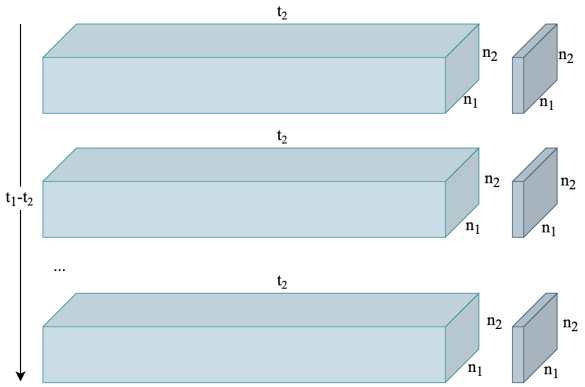


Figura 4.2: Estructura de volatilidades en predicción

Teniendo todo esto en cuenta, se ha prestado especial atención a los trabajos como el de Shi y col. 2015 donde se presenta por primera vez la estructura convolucional LSTM con el objetivo de predecir la intensidad de las precipitaciones en un espacio de tiempo corto y en el que los datos se estructuran como secuencias espaciotemporales semejantes a las que se han descrito. La arquitectura de la red se muestra en la siguiente imagen.

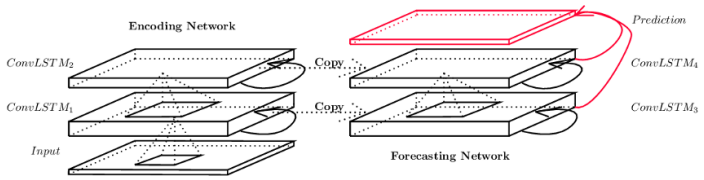


Figura 4.3: Estructura del modelo de Shi y col. 2015

Otro de los trabajos de referencia en el uso de este tipo de arquitecturas es Patraucean, Handa y Cipolla 2015, en el que construyen un autoencoder espacio temporal anidando dos autoencoders: generan un autoencoder temporal con capas convolucionales LSTM que explota el mapeo de características del encoder espacial (convolucional) y cuyo resultado alimenta al decoder espacial. De esta manera, minimizando el error de reconstrucción entre la siguiente secuencia predicha y la imagen real, entrenan el sistema para extraer características útiles para la estimación del movimiento sin necesidad de supervisión. En la siguiente imagen se muestra de forma esquemática la estructura del modelo.

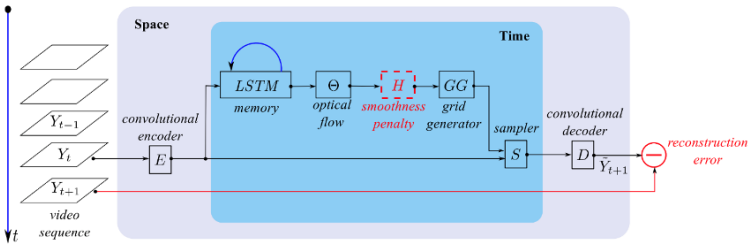


Figura 4.4: Estructura del modelo de Patraucean, Handa y Cipolla 2015

Siguiendo con el uso de autoencoders, es de destacar el trabajo de Medel 2016 en el que se proponen dos modelos con capas convolucionales LSTM cuyo objetivo es predecir la siguiente secuencia de vídeo. El primero es un encoder-decoder que aprende características espacio temporales a partir de un área determinada en una serie de imágenes apiladas y no superpuestas, mientras que la segundo está basado en un autoencoder que utiliza un agregador máximo para aprender una abstracción de la imagen entera. Para ello, estos modelos tienen dos cabezas que dan, por un lado, una reconstrucción de la imagen objetivo y, por otro, una predicción de las imágenes siguientes. Con ello, intentan mejorar la calidad de la predicción en base a la divergencia de la reconstrucción frente a la imagen real. Además, desarrollan los modelos con una modificación extra para condicionar los resultados y así aprender las representaciones más significativas. El objetivo final, es que estos modelos aprendan a modelar actividades normales de situaciones habituales y, concluyen, las arquitecturas con capas Conv-LSTM demuestran dar mejores resultados comparados con otros modelos actuales. Las dos arquitecturas se muestran en la imagen siguiente.

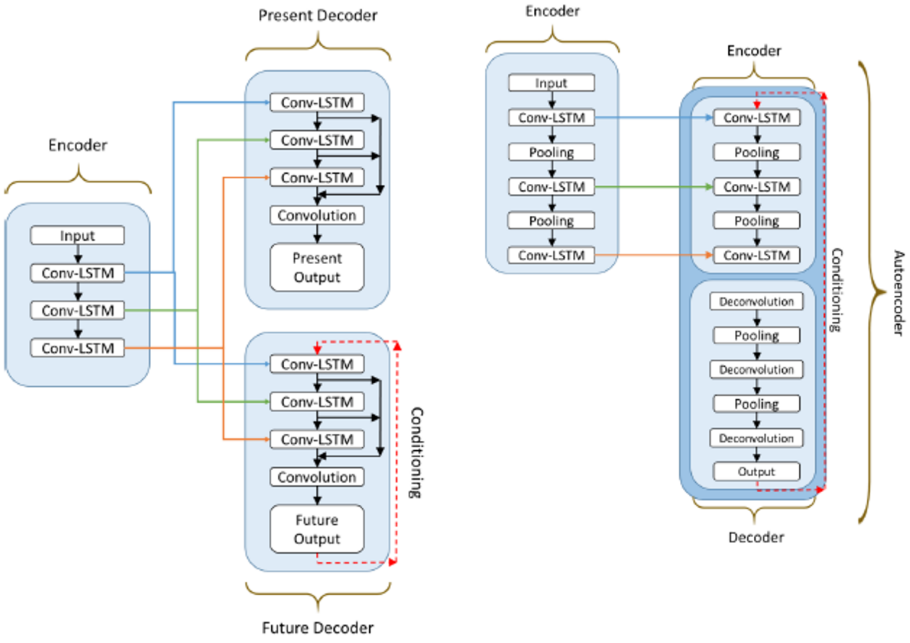


Figura 4.5: Estructura del modelo de Medel 2016

Las estructuras que se han descrito con anterioridad se utilizarán para construir el modelo aplicado al cubo de volatilidades implícitas que conforman la entrada de información. La capacidad de las RNN y LSTM de encontrar correlaciones temporales, la capacidad de las CNN de encontrar correlationes espaciales y la capacidad de la combinación de ambas para encontrar la unión de dichas correlaciones alimentarán las capacidades del modelo de estimar la superficie completa de volatilidades implícitas de las opciones del MiniIBEX 35 cotizadas diariamente.

La arquitectura detallada, así como el proceso de construcción, selección de parámetros y entrenamiento será detallada en la próxima sección.

Capítulo 5

Construcción y entrenamiento del modelo

Una vez explicadas las diferentes arquitecturas neuronales que podrán componer el modelo, el presente capítulo se centra en la construcción del mismo mediante optimización de hiperparámetros, con la intención de encontrar la estructura que se adapte mejor a la naturaleza de los datos utilizados. El modelo inicial, en base a Fang, Yu y Tang 2021 tiene la siguiente configuración:

Input	**Shape**
Input Layer	$(None, S, 10, 8)$
ConvLSTM Layer	$(None, 10, 8, F_1)$
Dropout	$(None, 10, 8, F_1)$
Conv2D	$(None, 10, 8, F_2)$
Conv2D	$(None, 10, 8, 1)$
Reshape	$(None, 10, 8)$

Para el modelo presentado se buscará una configuración optimizada en función de sus hiperparámetros, y se obtendrá el modelo que mejor se adapte a los datos en función de las métricas obtenidas en el conjunto de validación.

5.1. Optimización de hiperparámetros

La optimización de hiperparámetros en el aprendizaje automático tiene por objeto la obtención de los ajustes del modelo que ofrecen el mejor rendimiento medido en un conjunto de validación. Los hiperparámetros son manipulados por el usuario en el momento de la creación del modelo.

A continuación se presentan algunos de los hiperparámetros que afectan al modelo:

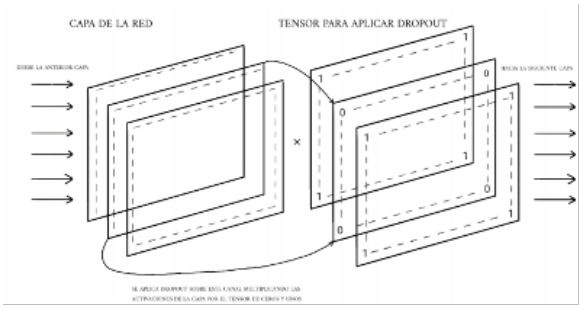
- **Tamaño de la secuencia** S: El tamaño de la secuencia de superficies utilizadas para estimar la superficie en t_1. Se han considerado 15 y 30 días.
- **Filtros**: Un kernel es una pequeña matriz que se utiliza para desenfocar, agudizar, grabar, detectar bordes entre otras características. Esto se logra haciendo una convolución entre el núcleo y una imagen o matriz de valores *Núcleo (procesamiento Digital de Imágenes)* 2021.
- ***Tamaño del filtro***: El tamaño de un filtro será un vector de n dimensiones, siendo n la dimensión de la convolución.

- ***Kernel Regularizer***: La función de los regularizadores es aplicar penalizaciones en los parámetros de la capa o la actividad de la capa durante la optimización. Estas penalizaciones se suman a la función de pérdida que optimiza la red. El objetivo de usar regularizaciones es reducir el sobreajuste de la red. Existen distintos tipos de regularizaciones:

  - *Lasso* $L1$: Esta regularización se utiliza cuando varios de los atributos de entrada sean irrelevantes. Al usar este tipo de regularización, se favorece que algunos de los coeficientes tiendan a cero, por tanto, que el modelo generalice mejor.
  - *Ridge* $L2$: Esta regularización se utiliza cuando varios de los atributos de entrada puedan estar correlados entre ellos. La regularización $L2$ disminuye los coeficientes con el objetivo de disminuir la correlación y mejorar la generalización.
  - *ElasticNet* $L1L2$: Esta regularización combina las dos anteriores.

- **Funciones de activación**: Se encargan de devolver una salida que será generada por la neurona a partir de un valor de entrada o de un conjunto. Cada capa tendrá su función de activación que permitirá reconstruir o predecir, normalmente el conjunto de valores de salida se encuentra en un rango determinado. Existen distintos tipos de activación con usos muy diversos. Las funciones analizadas en en la búsqueda de hiperparámetros para el modelo han sido la sigmoide, la tangente hiperbólica, la función *ReLu*, la *LeakyRelu* y la *Selu*.

- ***Dropout***: El dropout es una técnica de regularización basada en la desactivación de neuronas aplicada según la distribución de Bernoulli. El objetivo principal de esta técnica es mitigar la posible aparición del fenómeno conocido como sobreajuste, es decir, la excesiva particularización del ajuste de los parámetros del modelo para el conjunto de datos con los que se entrena, lo cual provoca que el modelo no sea capaz de generalizar y funcionar correctamente con otros datos de entrada.



- ***Batch normalization***: El batch normalization es un método que normaliza la salida de una capa restando la media de los valores de salida y dividiendo por la desviación estándar del mismo batch.

- **Optimizador**: es un algoritmo que minimiza la función de coste mediante el ajuste de los parámetros de la red utilizando el método de *backpropagation*. Los optimizadores analizados son los siguientes:

- *Stochastic Gradient Descent (SDG)*: El SDG es un algoritmo iterativo que comienza desde un punto aleatorio en una función y viaja por su pendiente en pasos hasta que alcanza el punto mínimo de la misma. El descenso del gradiente estocástico sólo pasa pares de datos seleccionados al azar cada vez. La velocidad de cálculo es rápida, pero es probable que la función de pérdida oscile y tarde en converger.
- *Adagrad y Momentum (Adam)*: El algoritmo de Adam obtiene las ventajas de los algoritmos AdaGrad y RMSProp. Adam no solo calcula la tasa de aprendizaje de parámetros adaptativos en función del valor medio del primer momento como el algoritmo RMSProp, sino que también hace un uso completo del valor medio del segundo momento del gradiente, es decir, la varianza no centrada. Específicamente, el algoritmo calcula el promedio móvil exponencial del gradiente, y los hiperparámetros β_1 y β_2 controlan la tasa de disminución de estos promedios móviles. El valor inicial de la media móvil y los valores de β_1 y β_2 están cerca de 1 (valor recomendado), por lo que la desviación de la estimación de momento se aproxima a 0.

- ***Learning rate***: Con el fin de agilizar la convergencia de la función de coste hacia su mínimo, el vector de gradiente se multiplica el factor de aprendizaje (lr). Este factor es un hiperparámetro que habrá que optimizar y que controla cuánto cambia el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo, es decir, controla la rapidez con la que el modelo se adapta al problema. Un valor demasiado pequeño puede resultar en un proceso de entrenamiento estancado en un mínimo local, mientras que un valor demasiado grande puede resultar en el aprendizaje de un conjunto de pesos subóptimo demasiado rápido o en un proceso de entrenamiento inestable.

Por último la función de coste C utilizada para la totalidad de los modelos estudiados ha sido el *Mean Squared Error*, MSE, que captura la media de las diferencias cuadradas entre las superficies comparadas:

$$MSE = \frac{1}{N}\sum_{i=1} N(y_i - y_i')^2$$

Una vez presentados los hiperparámetros que se han tenido en consideración a la hora de buscar la mejor configuración para el modelo propuesto, se presenta el marco utilizado para llevar a cabo los cálculos. La complejidad de las estructuras, unida a la gran cantidad de parámetros e hiperparámetros a utilizar, nos han llevado a realizar dicho entrenamiento en una plataforma en la nube dotada de gran potencia de cálculo.

5.1.1. KerasTuner

Los descripciones del presente capítulo han sido obtenidas de la documentación de Keras.Team s.f.

El marco utilizado para la búsqueda de hiperparámetros ha sido KerasTuner, una herramienta de optimización escalable que incorpora algoritmos de optimización bayseiana, *hyperband* y búsqueda aleatoria, y que está diseñado para poder implementar algoritmos de búsqueda personalizados.

Las diferentes especificaciones del modelo se transladarán a KerasTuner a través de *hp*, un sintonizador de hiperparámetros que permite cuatro formatos diferentes:

1. *hp.Int*(): Se utiliza para establecer el rango de hiperparámetros cuyos valores son números enteros. El número de filtros o el tamaño del kernel entran en esta categoría.

2. *hp.Float*(): Se utiliza para establecer el rango cuando los valores sea números decimales, por ejemplo el learning rate y el dropout.

3. *hp.Choice*(): Ofrece la posibilidad de introducir una secuencia de valores específicos.

4. *hp.Boolean*(): Permite definir los hiperparámetros que tengan una salida binaria, por ejemplo el uso o no de una capa de dropout.

A la hora de elegir un algoritmo de búsqueda para la optimización de hiperparámetros KerasTuner presenta tres opciones:

1. **Búsqueda aleatoria**: Una forma de encontrar los parámetros óptimos es probar todas las combinaciones posibles de los parámetros disponibles (*Grid Search*), pero el número de combinaciones aumenta exponencialmente a medida que crece el número de hiperparámetros. La búsqueda aleatoria ayuda a explorar más espacio de hiperparámetros en menos tiempo en comparación con una búsqueda total. Explorar más espacio de hiperparámetros no garantiza los resultados óptimos absolutos pero si cercanos al óptimo.

2. ***Hyperband***: Esta técnica pretende eliminar el problema principal de los algoritmos de búsqueda aleatoria: el análisis de configuraciones evidentemente malas, que aumentan el coste de computación y no aportan un set de resultados optimizado. Hyperband proporciona una forma de resolver este problema muestreando aleatoriamente todas las combinaciones y, tras un entrenamiento con un número de épocas menor del estipulado, descartar aquellos que hayan arrojado peores resultados. Esta técnica permite hacer un descarte inicial de las peores configuraciones.

3. **Optimización Bayesiana**: Todas las combinaciones de hiperparámetros se eligen aleatoriamente. La elección aleatoria de hiperparámetros ayuda a explorar el espacio de hiperparámetros, pero no garantiza hiperparámetros óptimos absolutos. La solución a este problema proporcionada por la optimización bayesiana consiste en elegir las primeras al azar y, en función del rendimiento de estos hiperparámetros, elegir los siguientes mejores posibles teniendo así en cuenta el historial de las configuraciones probadas.

Para el presente trabajo se ha seleccionado el optimizador de Hyperband, que presenta un algoritmo de búsqueda completo y ajustado a los requerimientos del problema presentado. A continuación se presentan los resultados para los tamaños de secuencia propuestos.

5.1.2. Optimización y entrenamiento

5.1.2.1. Secuencia de 15 días

Los resultados para el entrenamiento utilizando una secuencia de 15 días, es decir, estimando la superficie de volatilidad de t'_1 con la secuencia $\{t'_1-15, t'_1-1\}$ se presentan en la tabla siguiente:

Hiperparámetro	Mejor Valor
Learning rate	0.001
Dropout	True
Dropout rate	0.2
L1	0.001
L2	0.005
Num capas	1
Activación	*tanh*
Optimizador	Adam
F_1	64
Tamaño del kernel ConvLSTM	2
F_2	128
Tamaño del kernel Conv2D	3

Por lo tanto el modelo propuesto tendría la forma que se muestra a continuación:

Input	Shape
Input Layer	$(None, 15, 10, 8)$
ConvLSTM Layer	$(None, 10, 8, 64)$
Dropout	$(None, 10, 8, 64)$
Conv2D	$(None, 10, 8, 128)$
Conv2D	$(None, 10, 8, 1)$
Reshape	$(None, 10, 8)$

Y los resultados del entrenamiento para los sets de validación y test son los siguientes, con un error final de validacion de 0.0016:

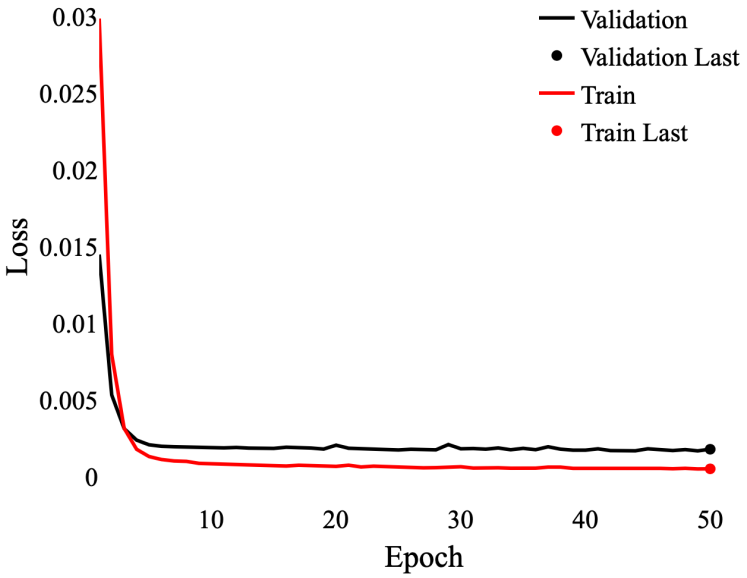


Figura 5.1: 15 dias de secuencia

5.1.2.2. Secuencia de 30 días

Los resultados para el entrenamiento utilizando una secuencia de 30 días, es decir, estimando la superficie de volatilidad de t_1' con la secuencia $\{t_1'-30, t_1'-1\}$ se presentan en la tabla siguiente:

Hiperparámetro	Mejor Valor
Learning rate	0.001
Dropout	True
Dropout rate	0.2
L1	0.001
L2	0.005
Num capas	1
Activación	*tanh*
Optimizador	Adam
F_1	64
Tamaño del kernel ConvLSTM	2
F_2	128
Tamaño del kernel Conv2D	3

Por lo tanto el modelo propuesto tendría la forma que se muestra a continuación:

Input	Shape
Input Layer	$(None, 30, 10, 8)$
ConvLSTM Layer	$(None, 10, 8, 64)$
Dropout	$(None, 10, 8, 64)$
Conv2D	$(None, 10, 8, 128)$
Conv2D	$(None, 10, 8, 1)$
Reshape	$(None, 10, 8)$

Y los resultados del entrenamiento para los sets de validación y entrenamiento son los siguientes, con un error final de validacion de 0.0022:

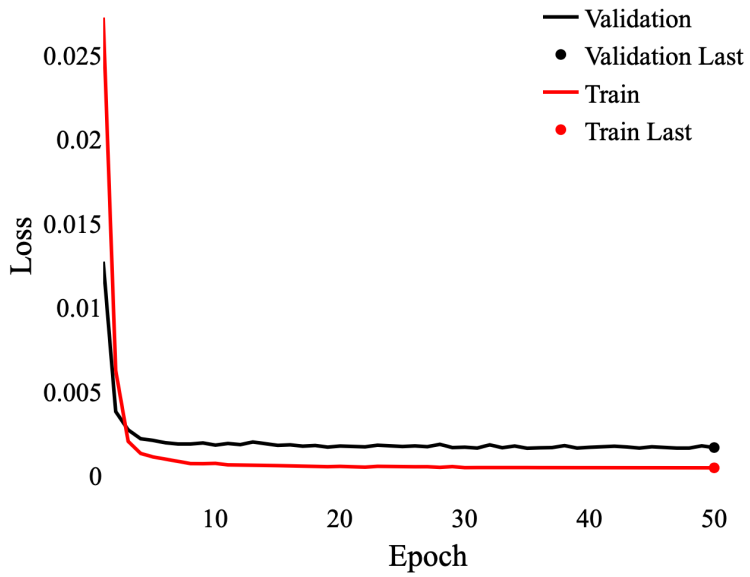


Figura 5.2: 30 dias de secuencia

Como se puede comprobar en los resultados de la optimización de hiperparámetros, las configuraciones para secuencias de 15 y 30 días son iguales, teniendo también un error final en la validación muy similar. Debido a la naturaleza del problema se ha considerado utilizar la secuencia más corta porque, aunque los errores son muy semejantes, se ha estimado que una secuencia de menor tamaño puede capturar mejor la estructura temporal en el histórico de volatilidades implícitas.

Capítulo 6

Estrategia de inversión

El algoritmo de inversión implementado en el presente trabajo ha operado haciendo uso de los contratos de opciones sobre el MiniIBEX 35 negociados en el MEFF. La descripción de los componentes de los contratos se presenta a continuación:

- **Unidad de negociación**: En las Opciones de MEFF sobre el Mini IBEX cada contrato tiene como subyacente un Futuro Mini sobre el IBEX 35 y cuyo nominal es el valor de este índice multiplicado por un euro.

- **Precio de ejercicio**: Precio al que da derecho a compra o vender el Futuro Mini IBEX. Este se fija en puntos enteros del Futuro Mini IBEX con intervalos de 100.

- **Prima**: Precio que paga el comprador de opciones por adquirir el derecho de compra o venta del subyacente. En el caso de las primas sobre las opciones Mini IBEX, estas cotizan en puntos enteros del futuro, siendo un punto un euro.

- **Fecha de vencimiento**: Es la fecha límite en la que se puede ejercer el derecho que otorga la opción. En general, los vencimientos se producen los terceros viernes de los meses de vencimiento, pero también puede ser el día hábil anterior si fuera festivo. Los vencimientos que se negocian generalmente son los trimestres del año en curso, es decir, marzo, junio, septiembre y diciembre. Para algunos subyacentes también se incluyen estos plazos del año siguiente y los semestrales junio y diciembre hasta los 5 años siguientes. Sin embargo, las opciones más líquidas son las de vencimiento más cercano.

- **Tipo de ejercicio**: Las opciones Mini IBEX son de tipo europeo, es decir, solo se pueden ejercer a fecha de vencimiento. El ejercicio se realiza de manera automática si el precio de ejercicio es más favorable para el tenedor de la opción, que el precio de liquidación a vencimiento del futuro. Esto explicar bien. En caso de ejercicio, la posición en opciones se transformar en una posición en futuros del mismo vencimiento, al precio de ejercicio de la opción. Las posiciones resultantes en futuros se crean al cierre de la sesión de la fecha de vencimiento y se liquidan en efectivo por diferencias del precio del futuro con respecto al precio de liquidación a vencimiento.

Para el correcto funcionamiento del algoritmo, se han asumido una serie de suposiciones que se enumeran y explican a continuación:

1. Se operará a cierre de mercado utilizando el precio de cierre. La falta de datos intradiarios nos ha obligado a tomar esta asunción, que permitirá al algoritmo operar al precio de cierre una vez comparados los resultados de volatilidad de modelo y volatilidad de mercado.

2. Se supone un mercado infinitamente líquido en el que existen precios y mercado para todas las opciones deseadas en un rango determinado de moneyness y tiempo a vencimiento.

3. Nuestra operativa es totalmente independiente al mercando, no influyendo esta en los precios de cotización que se ofrecen.

6.1. Estrategias

Una vez expuestos los contratos utilizados y las asunciones tomadas para su ejectución se describen los dos algoritmos que han operado en un total de 204 fechas, desde febrero del 2021 hasta noviembre del mismo año.

6.1.1. Estrategia Long Call-Put *in-n-out*

La estrategria *in-n-out* pretende explotar las ineficiencias de valoracion de opciones a precio de cierre, cometidas en el mercado entre dos días consecutivos.

Para cada día t' de los 204 en los que opera el algoritmo, este calculará, utilizando el modelo ConvLSTM, la superficie de volatilidades implícitas a cierre de día $\mathcal{VI}_{t'}$ y la comparará con la superficie extraída de mercado, $\mathcal{VM}_{t'}$, buscando así ineficiencias en la valoración. Si $\mathcal{VI}_{t'} - \mathcal{VM}_{t'} > \tau$ para alguna volatilidad de la matriz, es decir, si el precio estimado es mayor que el precio de mercado más un umbral se habrá encontrado una opción infravalorada por el mercado.

La estrategia *in-n-out* es exclusiva para posiciones largas, comprando calls o puts en función de la delta del portfolio. La idea detrás de la calibración basada en deltas es la de tener un algoritmo poco afectado por la tendencia del mercado, y en el cual los beneficios reportados provengan exclusivamente de las ineficiencias encontradas. Por el mismo motivo, *in-n-out* se deshace de posiciones que hayan estado en los libros cinco días o más.

6.1.2. Estrategia Long Call-Put *long-run*

La estrategia *long-run*, al igual que la anteriormente presentada, emplea las diferencias entre las superficies de volatilidad $\mathcal{VI}$ y $\mathcal{MI}$ para entrar en posiciones largas a final de día. Esta estrategia conservará en el libro todas las posiciones hasta 10 días antes de vencimiento.

Para ambas estrategias el umbral τ se ha considerado de 0.03, de tal manera que las diferencia entre las volatilidades estudiadas sean suficientemente significativas para estimar que se ha encontrado una posición mal valorada.

6.1.3. Estrategia *random*

Finalmente, la última estrategia para la cual se ha utilizado el cubo de volatilidades estimado por el modelo ConvLSTM es la estrategia *random*. La estrategia *random* compra una call o put presente en mercado cada día, deshaciendose de las posiciones

de más de 5 días de antiguendad en el portfolio. De esta manera se ha comprobado si los modelos estudiados superan la aleatoriedad.

6.2. Resultados y métricas

A continuación se presentan los resultados de las estrategias descritas anteriormente, así como una serie de métricas y conclusiones explicativas:

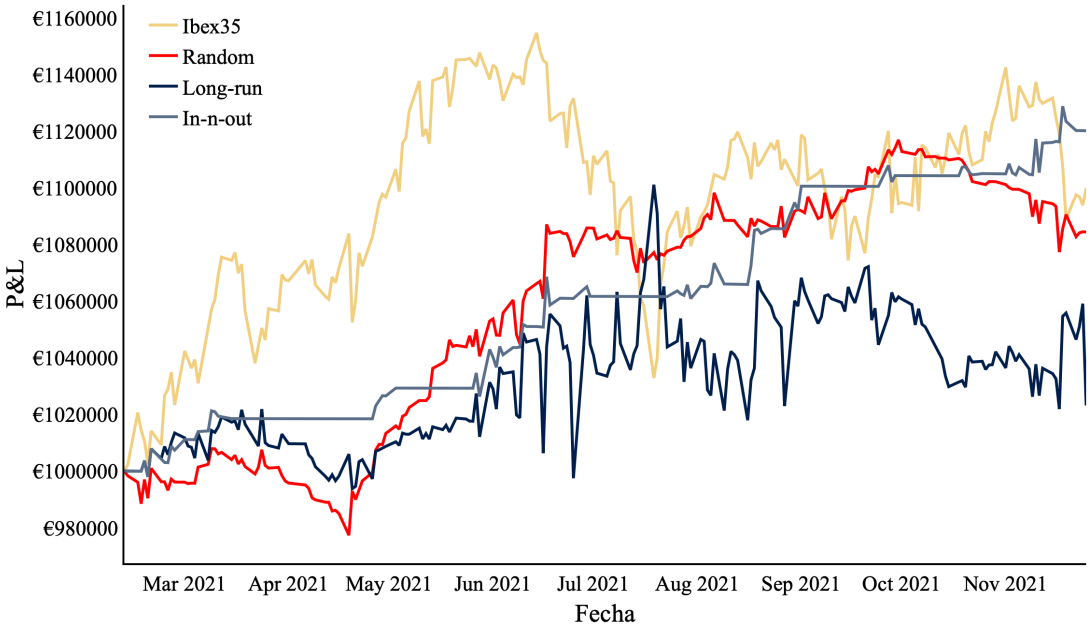


Figura 6.1: P&L de las estrategias

Métrica/Estrategia	***In-n-out***	***Largo Plazo***	***Aleatoria***
Total Operaciones	24	18	102
Operaciones positivas	21	6	55
Capital Inicial	1,000,000.00 €	1,000,000.00 €	1,000,000.00 €
Resultado neto	120,240.30 €	23,150.39 €	84,781.36 €
Rentabilidad	12.04 %	2.31 %	8.43 %
Rentabilidad anualizada	14.87 %	2.85 %	10.41 %
Max Drawdown	-11,860.10 €	-83,135.90 €	-39,606.72 €

Como se puede ver en los resultados mostrados anteriormente, ambos algoritmos han logrado obtener retornos positivos, estando la estrategia *in-n-out* tanto por encima del benchmark como por encima de la estrategia *random*. Por otro lado, la estrategia *long-run* muestra resultados menos prometedores, con un *Max Drowdown* considerablemente más alto que la estrategia cortoplacista y no siendo capaz de batir al benchmark.

Este comportamiento puede tener explicación en que la premisa del ejercicio de estimación se basa en la obtención de volatilidades puntualmente mal apreciadas en el mercado. De esta manera, la estrategia a corto plazo aprovecha las inconsistencias de mercado de manera continuada en el tiempo, creciendo de manera estable y no viéndose afectada por los movimientos en el mercado, ni alcista ni bajistas.

Capítulo 7

Arquitectura Cloud

En el presente capítulo se describirá la arquitectura cloud utilizada para estructurar los flujos de información que han compuesto la totalidad del trabajo.

El entorno seleccionado ha sido *Amazon Web Services* al ser un entorno intuitivo con una capa *free tier* que permite llevar a cabo gran cantidad de funcionalidades.

La estructura del trabajo en el entorno web es como se enumera a continuación:

1. Se leen los ficheros de volatilidades en bruto año por año desde el almacenamiento local.

2. Se crea un pipeline para tratar estos datos y dejar una tabla con las volatilidades de todos los años juntos.

3. Se sube esta tabla por un lado a una base de datos SQL creada con RDS de Amazon Web Service y por otro lado se almacena en un Bucket de S3.

4. Posteriormente se lee esta tabla de S3 o de la base de datos SQL ya que ambas funciones están disponibles en el código del proyecto, y se desarrolla otro pipeline para convertir esta tabla mediante Black Scholes y SVI en las matrices con las que se entrenará el modelo neuronal.

5. Una vez más, se almacena en un bucket de S3 los cubos de volatilidades, ya que estos no tienen la estructura bidimensional de una base de datos tradicional.

6. Posteriormente, el modelo neuronal lee las matrices del bucket de S3 y obtiene unas predicciones las cuales se vuelven a subimos al bucket junto con los archivos de test.

7. Finalmente, el algoritmo de inversión usa dichas predicciones para crear la estrategia y obtener el resultado final.

Capítulo 8

Conclusiones y trabajos futuros

El presente trabajo ha abordado una gran cantidad de materias diferentes, comenzando con la extracción de datos brutos, así como su tratamiento y procesado matemático, continuando con la investigación y creación de un modelo predictivo adecuado para la estructura de información obtenida, y finalizando con la utilización de los resultados para la construcción de un algoritmo que ha operado con contratos de derivados sobre el futuro del índice español. Además, el uso de herramientas web, tales como Keras Tuner, nos ha permitido gestionar gran cantidad de modelos y buscar el más apropiado para aplicar a nuestro entorno de trabajo.

Uno de los algoritmos propuestos, el más cortoplacista, se ha comportado de manera satisfactoria frente al benchmark y frente al modelo aleatorio, teniendo esto sentido debido a la motivación del trabajo; encontrar brechas puntuales en la superficie de volatilidad estimada por el propio mercado.

Por otro lado, para hacer posible este trabajo, se han hecho una serie de simplificaciones y suposiciones que limitan su aplicabilidad a las condiciones reales de trabajo. A continuación, se comentan algunas de las limitaciones e ideas que se podrían desarrollar en el futuro para mejorar tanto la aplicabilidad como el realismo:

- Datos: solo se han utilizado los precios de cierre de las opciones para el índice IBEX, aportados por MEFF. Para mejorar sustancialmente la escasez de datos se podrían emplear los datos con toda la profundidad del mercado en cada sesión. Sin embargo, esto no mejora el problema de la liquidez por lo que también se podrían utilizar opciones sobre otros índices más líquidos como puede ser el Eurostoxx o el S&P 500, por mencionar alguno de los más extendidos.

- Modelo: El modelo optimizado presenta una arquitectura relativamente sencilla. Esto no es necesariamente malo, pero alimentar al modelo con una mayor cantidad de datos y nuevas variables podría implicar una mayor complejidad y, en consecuencia, se podría esperar una mayor explotación de las diferentes dinámicas que pudieran subyacer en el mercado de opciones.

- Estrategia: se ha intentado simplificar al máximo la estrategia para que sea lo suficientemente ágil para trabajar de manera diaria y que no incurra en una operativa con demasiadas operaciones. Sin embargo, se podría enriquecer si también se consideraran posiciones cortas (con su consecuente gestión de garantías) o aquellas estrategias que conlleven una combinación de opciones (spreads, túneles, etc). Incluso se podría establecer estrategias en función de expectativas de mercado (bajista, alcista o neutro).

Bibliografía

Scholes, Myron y Fischer Black (1973). "The pricing of options and corporate liabilities". En: *Journal of political Economy* 81.3, págs. 637-654.

Cantero, Jose Enrique Vargas (s.f.). "Método de Newton Raphson." En: ().

Heaton, Jeff (2018). *Ian goodfellow, yoshua bengio, and aaron courville: Deep learning*.

Medel, Jefferson Ryan (2016). *Anomaly detection using predictive convolutional long short-term memory units*. Rochester Institute of Technology.

Hochreiter, Sepp y Jürgen Schmidhuber (1997). "Long short-term memory". En: *Neural computation* 9.8, págs. 1735-1780.

Understanding LSTM networks (s.f.). URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Shi, Xingjian y col. (2015). "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". En: *Advances in neural information processing systems* 28.

Liu, S, CW Oosterlee y SM Bohte (2019). *Pricing options and computing implied volatilities using neural networks. Risks 7 (1): 16*.

Wei, Xiangyu y col. (2020). "A CNN based system for predicting the implied volatility and option prices." En.

Fang, Yanwen, Philip LH Yu y Yaohua Tang (2021). "CNN-based Realized Covariance Matrix Forecasting". En: *arXiv preprint arXiv:2107.10602*.

Patraucean, Viorica, Ankur Handa y Roberto Cipolla (2015). "Spatio-temporal video autoencoder with differentiable memory". En: *arXiv preprint arXiv:1511.06309*.

Núcleo (procesamiento Digital de Imágenes) (sep. de 2021). URL: https://es.wikipedia.org/wiki/NC3BAcleo_(procesamiento_digital_de_imC3A1genes).

Team, Keras (s.f.). *Keras Documentation: Kerastuner*. URL: https://keras.io/keras_tuner/.