



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

ANÁLISIS DE MODELOS DE REDES NEURONALES EN LA
PREDICCIÓN DE PRECIOS DE LAS ACCIONES EN LA BMV

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

MIGUEL ÁNGEL LIERA MONTAÑO

T U T O R

DRA. VERÓNICA ESTHER ARRIOLA RÍOS



2024

$A \dots$

Agradecimientos

Índice general

Abreviaturas	xvi
1. Introducción	1
2. Descomposición de señales	3
2.1. Transformada de Fourier (FT)	3
2.2. Transformada de Ondículas (WT)	6
3. Redes Neuronales para análisis de series de tiempo	11
3.1. Entrenamiento de una Red Neuronal	12
3.1.1. Algoritmos de Optimización	13
3.2. Redes Neuronales Auto-regresivas	17
3.3. Redes Neuronales LSTM	18
3.3.1. Entrenamiento de una red LSTM	21
3.4. Redes Neuronales GRU	23
3.4.1. Entrenamiento de una red GRU	24
4. Construcción del Modelo	27
4.1. Descomposición de datos por DWT	28
4.2. Modelo NARNN	32
4.3. Modelo LSTMnn	33
4.4. Modelo GRUnn	34
5. Proceso de Entrenamiento	35
5.1. Entrenamiento por reforzamiento del profesor	38
5.1.1. NARNN	39
5.1.2. DWT-NARNN	41

5.1.3. LSTMnn	43
5.1.4. DWT-LSTMnn	44
5.1.5. GRUUnn	46
5.1.6. DWT-GRUUnn	47
5.2. Entrenamiento Auto-predictivo	49
5.2.1. NARNN	50
5.2.2. DWT-NARNN	53
5.2.3. LSTMnn	55
5.2.4. DWT-LSTMnn	57
5.2.5. GRUUnn	59
5.2.6. DWT-GRUUnn	60
6. Evaluación de desempeño	63
6.1. Predicción Estándar	64
6.2. Predicción Auto-regresiva	66
6.3. Predicción Auto-predictiva con Corrección	68
7. Conclusiones	77
A. Desarrollo del gradiente o_t y a_o	79
B. Implementación de algoritmos	81
B.1. DWT multinivel	81
B.2. Implementación NARNN	82
B.3. Implementación LSTMnn	83
B.4. Implementación GRUUnn	84
B.5. Algoritmo Levenberg-Marquardt	85
C. Especificaciones de software	87
Bibliografía	89

Índice de figuras

2.1. La Transformada de Fourier de Tiempo Reducido: la función $f(t)$ es multiplicada con la función $g(t)$, obteniendo $f(t)g(t)$, repitiendo este proceso para $g(t - t_0)$, $g(t - 2t_0)$ (Imagen recuperada de [1]) .	4
2.2. Plano de frecuencia contra tiempo: la manera en como actúa la transformación es la misma para cada ventana de tiempo, pues su tamaño es fijo e invariable para analizar cualquier frecuencia alta o baja (Recuperado de [2]).	6
2.3. Plano de frecuencia contra tiempo: a diferencia de la TFTR, la CWT encuentra en la ampliación o compresión de $\psi^{a,b}$ la capacidad para realizar un análisis frecuencial adecuado en cada caso (Recuperado de [2]).	7
2.4. Ejemplos de funciones de ondículas madre: Los momentos de desaparición (que se ven reflejados por el número al lado del nombre de la ondícula) se refieren a la capacidad de una ondícula para <i>desvanecerse</i> al ser integrada con polinomios de cierto grado. En términos simples, si una ondícula tiene n momentos de desaparición, esto implica que el área bajo la ondícula multiplicada por un polinomio de grado n (o inferior) es igual a cero. (Recuperado de [2]).	8

2.5. Descomposición de los componentes de aproximación $A_{2j-1}X$ en sus respectivos componentes de aproximación $A_{2j}X$ y de detalle $D_{2j}X$, pasándolos por los filtros de paso alto y bajo \hat{g} y \hat{h} y aplicando un submuestreo.	9
2.6. Componentes de detalle y aproximación en cinco niveles de los datos de ACTINVRB	10
2.7. Reconstrucción de los componentes de aproximación $A_{2j+1}X$ a partir de sus componentes de aproximación $A_{2j}X$ y de detalle $D_{2j}X$, pasándolos por los filtros de síntesis de paso alto y bajo g y h y aplicando un muestreo ascendente.	10
3.1. Esquema general de una RNN	17
3.2. Esquema general de una célula LSTM (Recuperado de [3])	21
3.3. Esquema general de una célula GRU (Recuperado de Jeblad, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=66234713)	24
4.1. Precios de cierre semanal de ACTINVRB del 1 de enero de 2016 al 31 de diciembre de 2023	28
4.2. Funciones de descomposición (superior) y reconstrucción (inferior) de escala ϕ y de ondícula ψ	29
4.3. Extrapolación con modo <i>Symmetric</i>	29
4.4. Componentes de Detalle y Aproximación de los datos de ACTINVRB	30
4.5. Conjunto de entrenamiento y prueba de los datos de ACTINVRB	30

4.6. Componentes de detalle y aproximación en cinco niveles del conjunto de datos de prueba de ACTINVRB	30
4.7. Arquitectura NARNN (Recuperado de [4])	32
4.8. Arquitectura LSTMnn	33
4.9. Arquitectura GRUUnn	34
5.1. Diagrama del reforzamiento del profesor	38
5.2. Tabla comparativa entre combinaciones de parámetros de la NARNN: tasa de aprendizaje contra número épocas	39
5.3. Desempeño de la NARNN en conjunto de prueba: Muestro aleatorio y temporal	40
5.4. Tabla comparativa entre combinaciones de parámetros de la DWT-NARNN: tasa de aprendizaje contra número épocas. (Los datos resaltados en verde representan la mejor combinaciones para las componentes de alta frecuencia y azul para los de baja).	41
5.5. Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro aleatorio.	41
5.6. Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro temporal.	42
5.7. Desempeño de la DWT-NARNN en la reconstrucción del conjunto de prueba: Muestro aleatorio y temporal.	42
5.8. Desempeño de la LSTMnn en conjunto de prueba: Muestro aleatorio y temporal.	43

5.9.	Tabla comparativa entre parámetros para componentes de alta frecuencia de la DWT-LSTMnn: tasa de aprendizaje contra número épocas por cada componente.	44
5.10.	Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro aleatorio.	44
5.11.	Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro temporal.	45
5.12.	Desempeño de la DWT-LSTMnn en la reconstrucción del conjunto de prueba: Muestro aleatorio y temporal.	45
5.13.	Desempeño de la GRUUnn en conjunto de prueba: Muestro aleatorio y temporal	46
5.14.	Predicciones de la DWT-GRUUnn de los componentes del conjunto de pruebas de muestro aleatorio	47
5.15.	Predicciones de la DWT-GRUUnn de los componentes del conjunto de pruebas de muestro temporal	47
5.16.	Desempeño de la DWT-GRUUnn en la reconstrucción del conjunto de prueba: Muestro aleatorio y temporal.	48
5.17.	Diagrama del entrenamiento auto-predictivo	49
5.18.	Tabla comparativa entre parámetros de la NARNN: tasa de aprendizaje contra número épocas durante entrenamiento auto-predictivo.	50
5.19.	Desempeño auto-predictivo de la NARNN en conjunto de prueba: Muestro aleatorio y temporal.	50
5.20.	Tabla comparativa entre parámetros de la NARNN: tasa de aprendizaje contra número épocas y factor de decaimiento durante entrenamiento auto-predictivo.	52

5.21. Desempeño auto-predictivo de la NARNN en conjunto de prueba bajo la implementación del factor de decaimiento: Muestro aleatorio y temporal.	52
5.22. Tabla comparativa entre parámetros de la DWT-NARNN para componentes de alta frecuencia: tasa de aprendizaje contra número épocas durante entrenamiento auto-predictivo.	53
5.23. Predicciones auto-predictivas de la DWT-NARNNnn de los componentes del conjunto de pruebas de muestro aleatorio.	53
5.24. Predicciones auto-predictivas de la DWT-NARNNnn de los componentes del conjunto de pruebas de muestro temporal.	54
5.25. Desempeño auto-predictivo de la DWT-NARNNnn en conjunto de prueba: Muestro aleatorio y temporal.	54
5.26. Tabla comparativa entre parámetros de la LSTMnn: tasa de aprendizaje contra número épocas en un entrenamiento auto-predictivo.	55
5.27. Tabla comparativa entre parámetros de la LSTMnn: tasa de aprendizaje contra número épocas en un entrenamiento auto-predictivo con factor de decaimiento.	56
5.28. Desempeño auto-predictivo de la LSTMnn en conjunto de prueba: muestreo aleatorio y temporal.	56
5.29. Tabla comparativa entre parámetros de la DWT-LSTMnn: tasa de aprendizaje contra número épocas y factor de decaimiento durante entrenamiento auto-predictivo	57
5.30. Predicciones auto-predictivas de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro aleatorio.	57
5.31. Predicciones auto-predictivas de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro temporal.	58

5.32. Desempeño auto-predictivo de la DWT-LSTMnn en conjunto de prueba: muestreo aleatorio y temporal.	58
5.33. Desempeño auto-predictivo de la GRUUnn en conjunto de prueba. .	59
5.34. Predicciones auto-predictivas de la DWT-GRUUnn de los componentes del conjunto de pruebas de muestro aleatorio.	60
5.35. Predicciones auto-predictivas de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro temporal.	60
5.36. Desempeño auto-predictivo de la DWT-GRUUnn en conjunto de prueba: muestreo aleatorio y temporal.	61
6.1. Desempeño del modelo NARNN	69
6.2. Desempeño del modelo DWT-NARNN	70
6.3. Desempeño del modelo LSTM	71
6.4. Desempeño del modelo DWT-LSTM	72
6.5. Desempeño del modelo GRU	73
6.6. Desempeño del modelo DWT-GRU	74
6.7. Diagrama de caja para métrica RMSE de las predicciones de todos los conjuntos de datos.	75
6.8. Diagrama de caja para métrica MAPE de las predicciones de todos los conjuntos de datos.	75
6.9. Diagrama de caja para métrica DS de las predicciones de todos los conjuntos de datos.	76

Índice de tablas

C.1. Paquetes y sus versiones	87
C.2. Especificaciones del dispositivo	88

Abreviaturas

FT	Fourier Transform
STFT	Short Time Fourier Transform
WFT	Windowed Fourier Transform
WT	Wavelet Transform
CWT	Continous Wavelet Transform
DWT	Discrete Wavelet Transform
ANNs	Artificial Neural Networks
MSE	Mean Square Error
MAE	Mean Absolute Error
GD	Gradient Descent
SGD	Stochastic Gradient Descent
ADAM	Adaptative Moment Estimation
LM	Levenberg-Marquardt
NARNNs	Not linear Autorregresive Neural Networks
RNNs	Recurretn Neural Networks
RMLP	Recurrent MultiLayer Perceptron
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit

LTR	Linea de Tiempo Retrasado
RMSE	Rooted Mean Squared Error
MAPE	Mean Average Percentage Error
DS	Directional Symmetry
FD	Factor de Decaimiento

Capítulo 1

Introducción

El saber cómo cambiará el precio de una acción a futuro, el poder conocer qué momento es el ideal para realizar una transacción impacta directamente no solo en las decisiones que un inversionista tomará al comprar o vender o en los beneficios que el futuro propietario tendrá con su compra, si no en la planificación de estrategias financieras complejas de empresas que buscan maximizar sus ganancias. Sin embargo el mercado bursátil se caracteriza por su alta volatilidad, y es bien sabido el hecho de que éste se ve afectado por factores no solo económicos si no sociales, políticos y culturales. Es precisamente por este complejo reto por el cual en el campo de la inteligencia artificial es de especial interés el poder predecir el comportamiento de fenómenos bursátiles, pues representan un conjunto de datos no lineales, que más bien se puede modelar como un fenómeno aleatorio que pone a prueba las técnicas del pronóstico de datos.

En este sentido, las redes neuronales son una herramienta popular entre los investigadores a la hora de atacar este tipo de problemas. Se tratan de modelos matemáticos que tienen la capacidad de adaptarse a cierto tipo de datos mediante un entrenamiento, bien conocidas en la actualidad por sus aplicaciones en el reconocimiento de patrones, teniendo como entrada la voz, texto o cualquier conjunto de datos.

Aunado a esto, podemos encontrar en la literatura el uso de técnicas de preprocesamiento de datos para la limpieza de datos. Esto nos es especialmente útil desde que podemos encontrar patrones que a priori parecen desdibujados, además de darle a los modelos de aprendizaje un conjunto con el que pueden trabajar de

manera más eficiente.

Por lo anterior, el objetivo de este trabajo es crear y evaluar el desempeño de algunas arquitecturas de redes neuronales con el fin de encontrar aquella que ofrezca mejores resultados durante la predicción de los precios de las acciones de empresas que se encuentren listadas en la Bolsa Mexicana de Valores.

Capítulo 2

Descomposición de señales

Una serie de tiempo, en el ámbito financiero, se entiende como una sucesión de datos ordenados cronológicamente que, en el caso de nuestro estudio, representan el precio de las acciones. Presentan fluctuaciones importantes, pues el valor en el mercado de estos instrumentos en un espacio de tiempo corto, digamos de 2 semanas, puede parecer ir a la alza, hasta que algún factor externo actué sobre él e interrumpa su curso. Estas perturbaciones en la tendencia hacen difícil generar una observación de los patrones que la componen y aun más si tratamos el problema desde un análisis frecuencial, pero, como veremos a lo largo del capítulo, no es difícil llegar a estos resultados usando las técnicas adecuadas.

2.1. Transformada de Fourier (FT)

Dentro del campo del procesamiento de señales una de las técnicas más populares en su análisis es sin duda la Transformada de Fourier (*Fourier Transform*) (TF):

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)e^{-i\omega t} dt$$

Donde:

- $f(t)$ es la señal original en el dominio del tiempo.

- $e^{-i\omega t}$ es el término de frecuencia compleja.
- t es la variable tiempo.
- ω es la frecuencia.

Esta herramienta nos permite conocer las frecuencias que componen una señal pasando de un dominio de intensidad contra tiempo a uno que representa su frecuencia y su magnitud. Los coeficientes resultantes representan que tanto contribuyen las funciones base (*basis functions*), senos y cosenos, a la construcción de nuestra señal original. Es precisamente por ello que, como menciona Graps, A. [2], la notable desventaja de la Transformada de Fourier, es no poder ubicar temporalmente cada una de esas frecuencias. Simplemente conocemos que funciones y como intervienen en la creación de la onda que estudiamos, pero no cuando ocurren estas intervenciones.

Una solución a las limitaciones de la TF es la Transformada de Fourier de Tiempo Reducido (*Short Time Fourier Transform*) (TFTR) o también llamada Transformada de Fourier por Ventanas (*Windowed Fourier transform*) (WFT), pues trata a la transformada ubicándola temporalmente en un intervalo fijo en el tiempo de la señal no estacionaria en el que tratamos a esta como si lo fuera, para así obtener el análisis frecuencial. Al momento de realizar la convolución, solo se emplea sobre la región que delimita la función $g(t - \tau)$, como se ve:

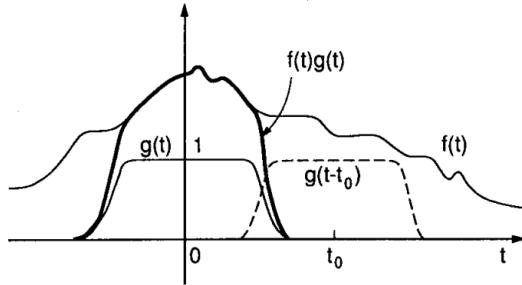


Figura 2.1: La Transformada de Fourier de Tiempo Reducido: la función $f(t)$ es multiplicada con la función $g(t)$, obteniendo $f(t)g(t)$, repitiendo este proceso para $g(t - t_0)$, $g(t - 2t_0)$ (Imagen recuperada de [1])

De manera que se obtiene una descomposición temporal más detallada, se define como sigue:

$$F(\tau, \omega) = \int_{-\infty}^{\infty} f(t)g(t - \tau)e^{-i\omega t} dt$$

Donde ¹:

- $f(t)$ es la señal original en el dominio del tiempo.
- $w(t - \tau)$ es la función de ventana que se aplica a la señal en el tiempo t . Esta función de ventana suele ser una función que tiene un valor máximo en t y disminuye hacia los lados.
- $e^{-i\omega t}$ es el término de frecuencia compleja.
- ω es la frecuencia.

Los componentes de baja frecuencia de $f(t)$ son representados en intervalos de tiempo más grandes, por lo que el tamaño de la ventana, afecta directamente a la calidad del análisis respecto a las frecuencias que podemos rescatar: a mayor tamaño de ventana se podrá observar un mayor detalle en los componentes de baja frecuencia, caso opuesto a menor espacio de tiempo, donde las frecuencias que resulten serán más altas.

Si nos ubicamos en el plano de frecuencia contra tiempo, en el caso de la TFTR el cómo la transformación actúa es la misma para todas las locaciones pues independientemente de si tenemos una frecuencia alta o baja, el tamaño de la ventana respecto al tiempo es fijo, su relación es fija:

¹Formula recuperada de [5]

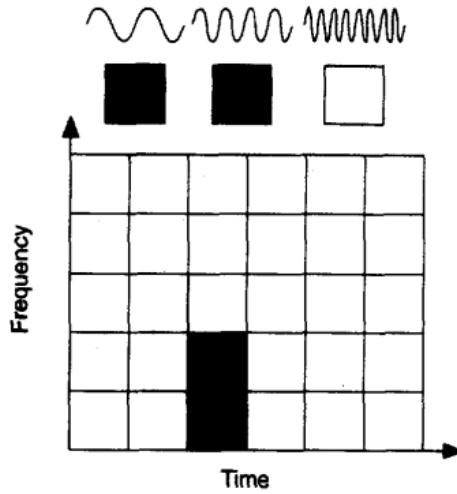


Figura 2.2: Plano de frecuencia contra tiempo: la manera en como actúa la transformación es la misma para cada ventana de tiempo, pues su tamaño es fijo e invariable para analizar cualquier frecuencia alta o baja (Recuperado de [2]).

2.2. Transformada de Ondículas (WT)

La Transformada de Ondículas o Transformada de Ondeletas (*Wavelet Transform*) (WT) es una técnica avanzada en el procesamiento de señales que descompone datos o funciones en sus coeficientes de frecuencias. Depende de dos variables, la escala o frecuencia a y la traslación sobre tiempo b , permitiendo un análisis frecuencial-temporal de la señal.

Notamos dos enfoques de Transformada de Ondículas: La Transformada Continua de Ondículas (*Continuos Wavelet Transform* (CWT)):

$$W_f(a, b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t) \psi^* \left(\frac{t-b}{a} \right) dt$$

Donde:

- $f(t)$ es la función original.
- $\psi^*(t)$ es la función conjugada compleja de ψ .

- a y b , parámetros de escala y translación respectivamente.

Se define como la integral sobre la convolución entre nuestra señal original y una función de corta duración, que pertenece a una familia de funciones referenciadas por a y b que tienen la forma $\psi^{a,b}(s) = |a|^{-1/2}\psi(s - b/a)$, donde ψ es llamada ondícula madre (*mother wavelet*), que será nuestra función base.

Al $\psi(t)$ comprimirse o expandirse dependiendo de a , $\psi^{a,0}(s) = |a|^{-1/2}\psi(s/a)$ cubre diferentes rangos de frecuencia. A mayor sea el valor de $|a|$ la salida se entenderá por componentes pertenecientes a frecuencias bajas, y un valor menor de $|a|$ corresponde a frecuencias más altas. Es aquí donde surge la principal diferencia entre esta y la TFTR, encontrando esta última una limitante en el hecho de que $g(t - \tau)$ tiene una longitud fija, lo cual acota el análisis frecuencial en cada ventana, solo w puede trasladarse sobre $f(t)$ a una localización temporal adecuada:

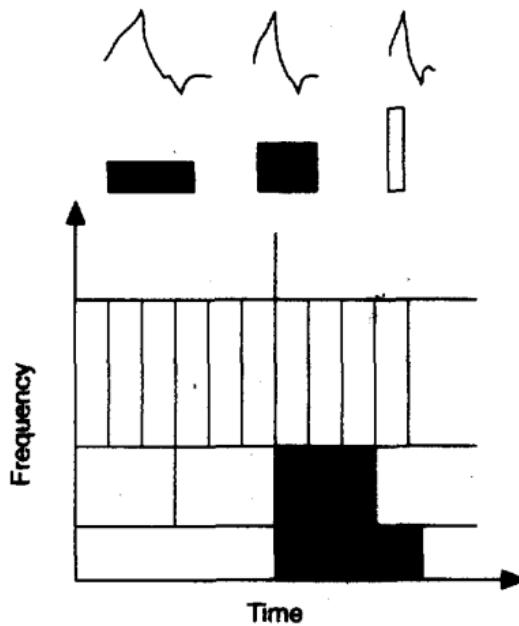


Figura 2.3: Plano de frecuencia contra tiempo: a diferencia de la TFTR, la CWT encuentra en la ampliación o compresión de $\psi^{a,b}$ la capacidad para realizar un análisis frecuencial adecuado en cada caso (Recuperado de [2]).

Ya que el problema que tratamos requiere el manejo de datos discretos, nos apoyaremos de la Transformada de Ondículas Discreta (*Discrete Wavelet Transform*)

(DWT). Se define como sigue:

$$D_X(a, b) = a^{-m} \int_{-\infty}^{\infty} X(t) \psi(a_0^{-m}t - nb_0)$$

Donde:

- $X(t)$ es la función original discretizada, nuestra serie de tiempo.

Típicamente, el valor de $a_0 = 2$ y $b_0 = 1$ para la discretización, de manera que las funciones base que son obtenidas de la ondícula madre se definen de la siguiente manera:

$$\psi_{j,k}(t) = 2^{j/2} \psi(2^j t - k)$$

, una familia de funciones que forman una base ortonormal para $L^2(R)$. Una de las primeras funciones ϕ que cumple con estos requerimientos es la función Haar, aunque en la literatura podemos encontrar otras como la función Daubechies (en la figura 2.4), o la función Biortogonal.

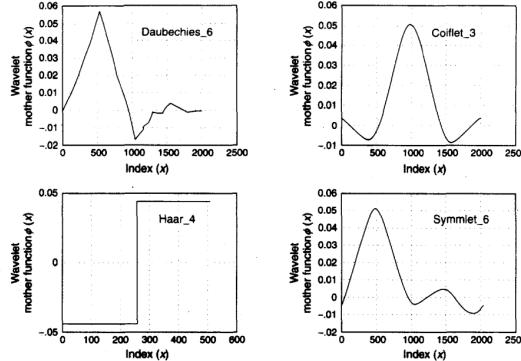


Figura 2.4: Ejemplos de funciones de ondículas madre: Los momentos de desaparición (que se ven reflejados por el número al lado del nombre de la ondícula) se refieren a la capacidad de una ondícula para *desvanecerse* al ser integrada con polinomios de cierto grado. En términos simples, si una ondícula tiene n momentos de desaparición, esto implica que el área bajo la ondícula multiplicada por un polinomio de grado n (o inferior) es igual a cero. (Recuperado de [2]).

Para obtener los componente de baja y alta frecuencia de $X(t)$, la DWT usa dos conjuntos de funciones, llamadas funciones de escala (*scale functions*) ϕ y funciones de ondícula ψ (*wavelet functions*) que están asociadas a un filtro de paso bajo (*low-pass filter*) \hat{g} y a uno de paso alto (*high-pass filter*) \hat{h} respectivamente. Después de filtrar los datos aplicando la transformada usando a ψ y a ϕ como funciones base, se eliminan la mitad de los valores mediante un submuestreo, de manera que esta mitad restante caracterice las componentes bajas o altas de la señal en cada caso. Un filtro de paso bajo permite el paso de las frecuencias menores, que presentan mayor amplitud, así atenuando las características de la señal, obteniendo aquellas que representan de manera más general o suavizan a $X(t)$. Lo que obtenemos serán los coeficientes de aproximación en la resolución 2^j (*Approximation Coefficients*) ($A_{2^j}X$). Por otro lado, si se permite el paso de frecuencias altas o ruido presente en la señal se trata de un filtro de paso alto y se obtendrán los coeficientes de detalle en la resolución 2^j de la señal (*Detail Coefficients*) ($D_{2^j}X$). A este procedimiento se le conoce como Descomposición por Subbandas (*Subband Coding*) [6].

Para una descomposición superior este proceso se puede repetir, aplicando un algoritmo piramidal partiendo de los coeficientes de aproximación a un nivel de resolución 2^{j-1} se puede obtener los coeficientes de detalle y aproximación del nivel 2^j de manera que los coeficientes de detalle de este nivel representan las diferencias entre los coeficientes de aproximación este nivel y el anterior. Tal es el comportamiento del algoritmo de descomposición de multi-resolución propuesto por Mallat [7] como se ve reflejado en la siguiente figura:

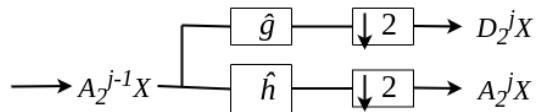


Figura 2.5: Descomposición de los componentes de aproximación $A_{2^{j-1}}X$ en sus respectivos componentes de aproximación $A_{2^j}X$ y de detalle $D_{2^j}X$, pasándolos por los filtros de paso alto y bajo \hat{g} y \hat{h} y aplicando un submuestreo.

Aplicando el algoritmo de descomposición de multi-resolución a datos reales, se ve como sigue:

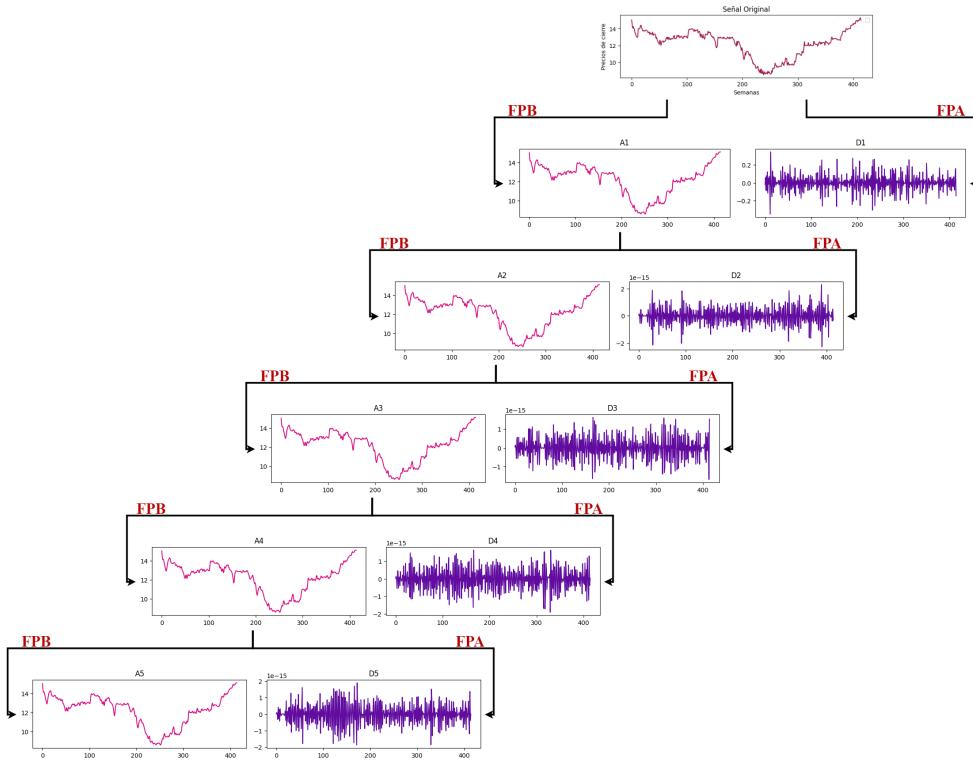


Figura 2.6: Componentes de detalle y aproximación en cinco niveles de los datos de **ACTINVRB**

Para la reconstrucción, las señales son aumentadas (*upsampled*) por un factor de dos, pasadas por los filtros de síntesis $h[n]$ de paso bajo y $g[n]$ de paso alto y luego simplemente sumando para obtener la reconstrucción en una resolución anterior. Es a este nivel, donde trabajaremos con nuestra serie de tiempo.

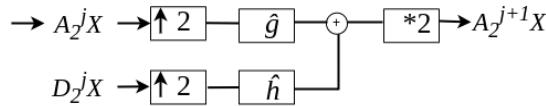


Figura 2.7: Reconstrucción de los componentes de aproximación $A_{2^{j+1}}X$ a partir de sus componentes de aproximación $A_{2^j}X$ y de detalle $D_{2^j}X$, pasándolos por los filtros de síntesis de paso alto y bajo g y h y aplicando un muestreo ascendente.

Capítulo 3

Redes Neuronales para análisis de series de tiempo

Las redes neuronales artificiales (*Artificial Neural Networks*) (ANNs) se han convertido en una poderosa herramienta para resolver problemas complejos desde el reconocimiento de patrones hasta la toma de decisiones autónoma. Inspiradas por la configuración y el funcionamiento del cerebro humano, estas estructuras han mostrado una capacidad enorme para modelar relaciones no lineales entre datos y aprender a partir de ellos.

Nos encontramos ante un modelo computacional que consiste en un tejido o red de neuronas o nodos interconectados entre sí. Estos se organizan en capas, dentro de las cuales cada neurona recibe datos como entrada y, junto con cierta ponderación de estos a partir de pesos y sesgos, lleva a cabo una combinación lineal y aplica una función de activación a dicho resultado para generar una salida o impulso. En un sentido general, el número de capas con el que cuenta una red es indistinto, pero siempre se puede identificar una capa de entrada, que es aquella por donde son consumidos los datos, una de salida, de donde se obtiene el resultado del procesamiento en la red y un número arbitrario de capas ocultas.

Para que una red genere cierta salida esperada existe la etapa de entrenamiento. Durante este proceso los pesos de las conexiones entre nodos y sus sesgos se ajustan utilizando algoritmos de aprendizaje.

3.1. Entrenamiento de una Red Neuronal

La capacidad de predicción en una red neuronal está ligada fuertemente al entrenamiento empleado en esta. Este procedimiento se compone de tres partes fundamentales: el paso de propagación hacia adelante (*feedforward*), el cálculo de la perdida a partir de una función de error E y el paso de retro-propagación (*backpropagation*).

El primer paso, propagación hacia adelante, se encarga de evaluar una entrada x_t en la red, de manera que la salida generada por la red a partir de esa entrada, llamémosle s_t , pueda compararse con el valor que esperábamos con dicha entrada, digamos y_t . Ambas se comparan a partir de la función de error E , que nos permite evaluar qué tan cerca se encuentra la predicción del valor real y así medir el desempeño actual de la red. Podemos encontrar funciones de error como el Error Cuadrático Medio (*Mean Square Error*) (MSE), que emplearemos más adelante, Entropía Cruzada (*Cross-Entropy*) o el Error Absoluto Medio (*Mean Absolute Error*) (MAE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

$$\text{Entropía Cruzada} = - \sum_{i=1}^n y_i \log(\hat{y}_i)$$

Es a partir de la función de error que sabemos el desempeño de la red en ese momento. Es aquí cuando se emplea el paso de retro-propagación que se encarga de calcular los gradientes de E respecto a los parámetros de la red. Para calcular cada uno de estos gradientes hacemos uso de la regla de la cadena (*chain rule*) como se menciona en [8], para un solo peso $w_{j,k}^{(c)}$ donde j, k sus respectivos índices para la conexión de este con las capas que preceden y anteceden a aquella a la que pertenece, es decir c . a_j^{c-1} es la ^activación.^o salida de la función de activación de la capa anterior. Se tiene entonces:

$$\frac{\partial E}{\partial w_{j,k}^c} = \frac{\partial E}{\partial z_k^c} a_j^{c-1}$$

Y también respecto a algún sesgo en c :

$$\frac{\partial E}{\partial b_k^c} = \frac{\partial E}{\partial z_k^{(c)}}$$

Así entonces, las actualizaciones de cada peso y sesgo esta dado por:

$$w_{j,k}^{(c)} := w_{j,k}^{(c)} - \alpha \frac{\partial E}{\partial z_k^{(c)}} a_j^{c-1}$$

$$b_k^c := b_k^c - \alpha \frac{\partial E}{\partial z_k^{(c)}}$$

Con los gradientes calculados, el algoritmo de optimización aplica una estrategia para ajustar los pesos de las conexiones en la red y los sesgos de cada neurona para minimizar la diferencia entre la salida que obtuvimos de esta y los valores originales. Algunos ejemplos de algoritmos de optimización más comúnmente usados son Descenso por el Gradiente (*Gradient Descent*) (GD), Descenso por el Gradiante Estocástico (*Stochastic Gradient Descent*) (SGD), Estimación Adaptativa de Momento (*Adaptative Moment Estimation*) (ADAM) y Levenberg-Marquardt (LM). A continuación profundizaremos sobre estos.

3.1.1. Algoritmos de Optimización

- **Descenso por el gradiente** Es el método más común para minimizar E . Ajusta los pesos y sesgos en la dirección opuesta al gradiente multiplicado por la tasa de aprendizaje (*learning rate*) (debido a que el gradiente indica la dirección hacia donde crece la función o donde se encuentra un máximo local). Este proceso se repite iterativamente durante varias épocas ¹, hasta que la función de pérdida converge o se detiene según algún criterio predefinido.

¹Una época comprende una iteración del proceso completo de entrenamiento, es decir propagación hacia adelante, cálculo en E , retro-propagación y algoritmo de optimización

- **SGD** Es una variante específica del Descenso por el Gradiente. En vez de calcular el gradiente utilizando todo el conjunto de datos de entrenamiento, se obtiene utilizando solo un subconjunto de los datos (mini lotes²) seleccionado de manera aleatoria en cada iteración del entrenamiento ayudando a evitar mínimos locales y puntos de estancamiento durante el entrenamiento. Esto hace que el cálculo del gradiente sea más eficiente, especialmente para conjuntos de datos grandes.
- **ADAM** Es un optimizador que combina la idea del descenso por el gradiente con el movimiento promedio de los momentos. Utiliza estimaciones adaptativas del primer y segundo momento de los gradientes para ajustar la tasa de aprendizaje de forma individual para cada parámetro de la red, lo que lo hace especialmente útil en problemas con características de datos variables o no estacionarias. [9] Además esto permite que, durante la optimización de la función de error ADAM da 'pasos' conforme al comportamiento del gradiente en ese instante, alcanzando el mínimo de manera más precisa.

ADAM emplea dos arreglos del tamaño de los parámetros de la red que se encargarán de contener el movimiento promedio o media móvil del primer y segundo momento de los gradientes, estos son actualizados en cada iteración del algoritmo ponderando los gradientes y el cuadrado de este respectivamente junto con m_{t-1} y v_{t-1} , los vectores de la corrida anterior (linea 4 y 5). Finalmente se actualizan los parámetros x_t (linea 7) con los vectores cuando se encuentran corregidos a partir de la taza de decaimiento (linea 6), esto para evitar un sesgo en la inicialización en cero.

²mini batch

Algoritmo 1: Algoritmo de Optimización ADAM

Entrada: Conjunto de datos X , tasa de aprendizaje α , β_1 , β_2 , ϵ **Entrada:** Parámetros del modelo: x_0 **Entrada:** Momentos de primer y segundo orden: $m_0 = 0$, $v_0 = 0$ **Entrada:** Contador de iteraciones: $t = 0$

```

1 mientras no se alcance el criterio de parada hacer
2    $t \leftarrow t + 1;$ 
3   Calcular el gradiente:  $g_t = \nabla_x E(x_{t-1})$ ;
4   Actualizar el momento de primer orden:  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$ ;
5   Actualizar el momento de segundo orden:  $v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$ ;
6   Corregir los momentos de primer y segundo orden:  $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$ ,  $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$ ;
7   Actualizar los parámetros:  $x_t = x_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$ ;
8 fin
Salida : Parámetros del modelo  $x_t$ 

```

- **Levenberg Marquardt (LM)** Es un algoritmo de optimización no lineal utilizado principalmente en problemas de ajuste de curvas y regresión no lineal. A diferencia de los métodos basados únicamente en el gradiente, LM utiliza una estrategia de optimización iterativa que combina técnicas de descenso por el gradiente y gauss-newton. El algoritmo ajusta los parámetros de manera iterativa, utilizando una matriz de aproximación o la misma matriz hessiana en combinación con el gradiente para guiar la búsqueda hacia el mínimo local de la función de error. La dirección de búsqueda del algoritmo se define como sigue:

$$\Delta x = -[H + \lambda I]^{-1} \nabla E(x)$$

Donde:

- x son los parámetros de la red.
- $E(x)$ es la función de error.
- H es la matriz Hesiana de $E(x)$.

Algoritmo 2: Algoritmo de Levenberg-Marquardt

Datos: Función de error $E(x)$, punto inicial x_0 , parámetros de ajuste λ_0 , tolerancia ϵ

Resultado: Solución \mathbf{x}^*

- 1 Inicializar $x \leftarrow x_0$;
 - 2 Inicializar $\lambda \leftarrow \lambda_0$;
 - 3 Definir $\epsilon_1 \leftarrow$ Tolerancia para la convergencia del método;
 - 4 Definir $\epsilon_2 \leftarrow$ Tolerancia para el valor del gradiente;
 - 5 **repetir**
 - 6 Calcular el vector de gradiente $\nabla E(x)$;
 - 7 Calcular la matriz Hessiana H ;
 - 8 Calcular el paso de Levenberg-Marquardt: $\Delta \mathbf{x} = -[H + \lambda I]^{-1} \nabla E(x)$;
 - 9 Calcular el valor de la función de error con el paso propuesto:
 $f_{\text{prop}} = f(\mathbf{x} + \Delta \mathbf{x})$;
 - 10 Calcular el valor de la función de error en el punto actual: $f_{\text{actual}} = f(\mathbf{x})$;
 - 11 **si** $f_{\text{prop}} < f_{\text{actual}}$ **entonces**
 - 12 Aceptar el paso: $\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}$;
 - 13 Actualizar λ : $\lambda \leftarrow \frac{\lambda}{2}$;
 - 14 **en otro caso**
 - 15 Rechazar el paso: No actualizar \mathbf{x} ;
 - 16 Aumentar λ : $\lambda \leftarrow 2\lambda$;
 - 17 **si** $|f_{\text{prop}} - f_{\text{actual}}| > \epsilon_1$ o $\|\nabla E(x)\| > \epsilon_2$ **entonces**
 - 18 Detener el algoritmo;
 - 19 **hasta que** convergencia;
-

3.2. Redes Neuronales Auto-regresivas

Las ARNNs, o redes neuronales auto-regresivas (*Autorregressive Neural Networks* también llamadas NARNN (por *Not linear Autorregressive Neural Networks*), se emplean en el estudio de series temporales y en tareas de predicción secuencial. Se trata de un tipo específico de arquitectura de redes neuronales que consiste en predecir los valores futuros a partir de los pasados d valores, contemplando la idea de que el comportamiento en el pasado de una serie temporal tiene impacto en su desempeño en el futuro.

Dada una serie temporal $\{x_1, x_2, \dots, x_t\}$, donde x_i representa una muestra en el tiempo i , una ARN puede predecir el valor futuro \hat{x}_{t+1} basado en las observaciones pasadas $x_{t-1}, x_{t-2}, \dots, x_{t-d}$, con $t, d \in \mathbb{N}$

Una ARN puede expresarse como una función f que mapea las observaciones pasadas a la predicción futura:

$$\hat{x}_{t+1} = f(x_{t-1}, x_{t-2}, \dots, x_{t-d})$$

En la práctica, las ARN suelen tener una estructura recurrente ³ (que trataremos más adelante), hecho que no implica el desuso de arquitecturas básicas con propagación hacia adelante (*feedforward*) como unidad base.

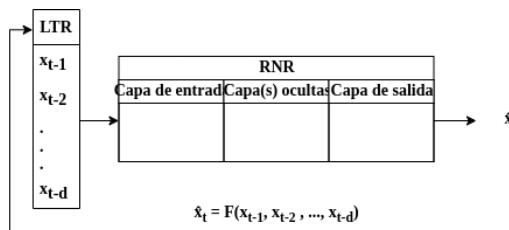


Figura 3.1: Esquema general de una RNN

Sin embargo, implementar capas recurrentes permite que la red capture dependencias o relaciones implícitas a lo largo de la serie temporal y realice predicciones

³Según Haykin, podemos interpretar a una ARN como una Red Neuronal Recurrente en si misma, sin embargo para fines de este trabajo, trataremos a ambas como entidades independientes

aun más precisas (como también veremos más adelante). Las ARN también pueden incorporar capas convolucionales o capas completamente conectadas, dependiendo de la naturaleza de los datos y la complejidad del problema de predicción.

3.3. Redes Neuronales LSTM

Las Redes Neuronales Recurrentes (*Recurrent Neural Networks*) (RNNs) son un sistema dinámico no lineal. A diferencia de las redes neuronales tradicionales, las RNN tienen conexiones retro-alimentadas, lo que significa que las salidas de algunos de sus nodos se incluyan como parte de la entrada del mismo nodo en siguientes iteraciones del proceso de predicción y la información que fluye a través de estas pueda hacerlo cíclicamente. Esto permite retener y aprovechar información sobre estados anteriores a lo largo de la secuencia de datos (persistencia). Están diseñadas para modelar datos secuenciales, como palabras en un texto o muestras en una serie temporal cuyos datos en cierto espacio de tiempo δt guardan relación con algún intervalo $\delta(t - d)$ en el pasado.

Dada una secuencia de entrada $\{x_1, x_2, \dots, x_t\}$, y una secuencia de salida correspondiente $\{y_1, y_2, \dots, y_t\}$, una RNN se define de forma general como sigue:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ y_t &= g(h_t) \end{aligned}$$

Donde:

- h_t es el estado oculto o estado interno en el tiempo t . Representa la memoria de la red en ese momento y se calcula utilizando una función de activación f que toma como entrada el elemento actual x_t y el estado oculto anterior h_{t-1} .
- y_t es la salida de la red en el tiempo t , calculada utilizando una función de activación g aplicada al estado oculto h_t ⁴.

⁴También se suele ver a $g = f$ de manera que $y_t = h_t$

- f y g son funciones de activación no lineales, como la función sigmoide (σ) o la función tangente hiperbólica ($tanh$), que introducen la no linealidad en el modelo y permiten a la red capturar relaciones complejas en los datos secuenciales.

Una de las arquitecturas más simples de las RNNs son los Perceptrones Multicapa Recurrentes (*Recurrent Multilayer Perceptron*) RMLP que se caracteriza por que cada una de sus capas recibe como entrada la salida que esta que genero en su en un tiempo $t - 1$. Se define como sigue:

$$\begin{aligned} x_{1,n+1} &= \Psi_I(x_{1,n}, U_n) \\ x_{1,n+1} &= \Psi_I I(x_{2,n}, x_{1,n+1}) \end{aligned}$$

.

$$x_{o,n+1} = \Psi_o(x_{o,n}, x_{M,n+1})$$

A pesar de que las RNNs hacen uso de información en el pasado para modelar el comportamiento actual, la manera cíclica de actuar de estas sólo influye cuando los datos que necesitamos pertenecen al pasado inmediato anterior de la predicción actual. Si por el contrario, lo que intentamos predecir está ligado a muestras que se presentaron con mayor anterioridad, la red perderá su capacidad de predicción con base a esta información. Es aquí donde se nos presenta el llamado problema de Dependencias a Largo Plazo (*Long-term dependencies*) [3].

Las redes neuronales con células de Memoria de Corto y Largo Plazo o LSTM (*Long Short-Term Memory*) son un tipo especializado de RNNs, presentadas por primera vez por Hochreiter y Schmidhuber (1991) [10], diseñadas para manejar con mayor eficacia los problemas de dependencias entre datos secuenciales. Las LSTM tienen unidades de memoria llamadas “células de memoria” que les permiten retener y actualizar información durante largos períodos, a diferencia de las RNNs comunes.

La idea central de una célula LSTM es su estado (*cell state*) y sus diferentes compuertas. La primera actúa como una banda transportadora o autopista que transfiere información a lo largo de la cadena de secuencia de la célula, mientras que las compuertas actúan como redes neuronales con la capacidad de aprender qué información se olvidará o recordará en iteraciones siguientes. Dada una secuencia de entrada $\{x_1, x_2, \dots, x_t\}$ y una secuencia de salida correspondiente $\{y_1, y_2, \dots, y_t\}$, una célula de memoria LSTM realiza la siguiente operación:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [x_t, h_{t-1}] + b_i) \\ f_t &= \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \\ o_t &= \sigma(W_o \cdot [x_t, h_{t-1}] + b_o) \\ g_t &= \tanh(W_g \cdot [x_t, h_{t-1}] + b_g) \\ C_t &= f_t \cdot C_{t-1} + i_t \cdot g_t \\ h_t &= o_t \cdot \tanh(C_t) \end{aligned}$$

Donde:

- i_t , compuerta de entrada (*input gate*): determina qué valores serán los candidatos para formar parte del nuevo estado de la célula C_t .
- f_t , compuerta de olvido (*forget gate*): mediante la función sigmoide se decide que datos olvidar (0) o mantener (1) del estado oculto (*hidden state*) anterior h_{t-1} .
- g_t es la candidata de celda.
- o_t compuerta de salida (*output gate*).
- C_t es el estado de la célula en el tiempo t , que almacena y actualiza la información relevante a largo plazo. Se obtiene al multiplicar g_t por f_t obteniendo los valores candidatos omitiendo aquellos que se decidieron olvidar, al sumar $i_t \cdot g_t$ tenemos todas las actualizaciones de los valores de C_t .
- h_t es la salida en el tiempo t , que se calcula utilizando una versión filtrada del estado de la célula.

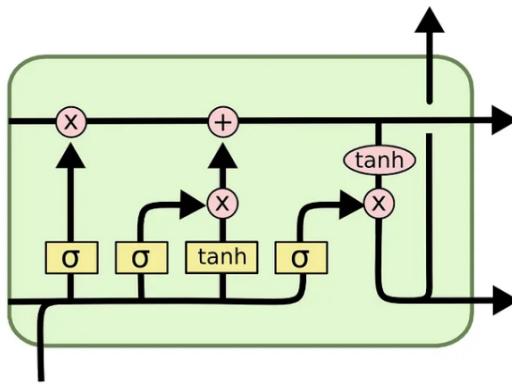


Figura 3.2: Esquema general de una célula LSTM (Recuperado de [3])

3.3.1. Entrenamiento de una red LSTM

El entrenamiento de una red con células LSTM parte del mismo principio que el de un perceptrón multicapa: un paso de propagación hacia adelante, evaluación del error por medio de la función E , la retro-propagación con el cálculo de sus respectivos gradientes y la optimización según el algoritmo que se aplique.

Para obtener los gradientes de E con respecto de cada uno de los componentes con ayuda de la regla de la cadena. Por ejemplo, se sabe que durante la propagación hacia adelante el flujo de la información de o_t :

$$o_t \text{ a } E: o_t \longrightarrow h_t \longrightarrow E$$

Por lo que el cálculo del respectivo gradiente es:

$$\frac{\partial E}{\partial o_t} = \frac{\partial E}{\partial h_t} \cdot \tanh(c_t)$$

Y para $a_o = W_o \cdot [x_t, h_{t-1}] + b_o$:

$$a_o \text{ a } E: a_t \longrightarrow o_t \longrightarrow h_t \longrightarrow E$$

Y entonces:

$$\frac{\partial E}{\partial a_o} = \frac{\partial E}{\partial h_t} \cdot \tanh(c_t) \cdot o_t(1 - o_t)$$

Y consecuentemente para los demás compuertas de la célula [11] ⁶ [12]

Los gradientes de la célula LSTM:

$$\frac{\partial E}{\partial h_t} = y_t - h_t$$

$$\frac{\partial E}{\partial o_t} = \frac{\partial E}{\partial h_t} \cdot \tanh(C_t)$$

$$\frac{\partial E}{\partial a_o} = \frac{\partial E}{\partial h_t} \cdot \tanh(c_t) \cdot o_t(1 - o_t)$$

$$\frac{\partial E}{\partial C_t} = \frac{\partial E}{\partial h_t} \cdot o_t \cdot (1 - \tanh^2(C_t))$$

$$\frac{\partial E}{\partial g_t} = \frac{\partial E}{\partial C_t} \cdot i_t$$

$$\frac{\partial E}{\partial a_C} = \frac{\partial E}{\partial C_t} \cdot i_t \cdot (1 - g_t^2)$$

$$\frac{\partial E}{\partial i_t} = \frac{\partial E}{\partial C_t} \cdot g_t$$

$$\frac{\partial E}{\partial a_i} = \frac{\partial E}{\partial C_t} \cdot g_t \cdot i_t(1 - i_t)$$

$$\frac{\partial E}{\partial f_t} = \frac{\partial E}{\partial C_t} \cdot C_{t-1}$$

$$\frac{\partial E}{\partial a_f} = \frac{\partial E}{\partial C_t} \cdot C_{t-1} \cdot f_t(1 - f_t)$$

$$\frac{\partial E}{\partial C_{t-1}} = \frac{\partial E}{\partial C_t} \cdot f_t$$

⁵El desarrollo de los anteriores se puede ver reflejado en A

⁶Aquí podemos encontrar la derivación completa de cada una de las compuertas

⁷Solo si se usa a $E = ECM(y_t, h_t)$

Definimos $Z_t = [x_t, h_{t-1}]$. Durante la propagación hacia adelante Z_t tiene flujo por medio de las compuertas de entrada, salida, de olvido y el estado de la célula:

$$\frac{\partial E}{\partial Z_t} = W_f^T \frac{\partial E}{\partial a_f} + W_i^T \frac{\partial E}{\partial a_i} + W_o^T \frac{\partial E}{\partial a_o} + W_C^T \frac{\partial E}{\partial a_C}$$

Por último, para los pesos y sesgos:

$$\begin{aligned}\frac{\partial E}{\partial W_t} &= \frac{\partial E}{\partial a_f} \cdot z_t^T, \quad \frac{\partial E}{\partial b_t} = \frac{\partial E}{\partial a_f} \\ \frac{\partial E}{\partial W_i} &= \frac{\partial E}{\partial a_i} \cdot z_t^T, \quad \frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial a_i} \\ \frac{\partial E}{\partial W_o} &= \frac{\partial E}{\partial a_o} \cdot z_t^T, \quad \frac{\partial E}{\partial b_o} = \frac{\partial E}{\partial a_o}\end{aligned}$$

3.4. Redes Neuronales GRU

Las redes neuronales con Unidades Recurrentes Cerradas (*Gated Recurrent Unis*) o GRU, propuestas por Cho, et. al [13], son una arquitectura de RNR diseñada para modelar y procesar eficientemente datos secuenciales. Las GRU, al igual que las LSTM, tienen como objetivo resolver el problema de las dependencias a largo plazo en datos secuenciales. No obstante, las GRU resultan más sencillas que las LSTM y poseen menos parámetros, lo cual acelera el proceso de entrenamiento y disminuye la susceptibilidad al sobreajuste en conjuntos de datos reducidos.

Una GRU consta de compuertas de actualización (*update gate*) y reinicio (*reset gate*) que controlan el flujo de información dentro de la unidad. Dada una secuencia de entrada x_t , una GRU se define a partir de las siguientes ecuaciones:

$$\begin{aligned}z_t &= \sigma(W_z \cdot x_t + U_z \cdot h_{t-1}) \\ r_t &= \sigma(W_r \cdot x_t + U_r \cdot h_{t-1}) \\ g_t &= \tanh(W_g \cdot x_t + r_t \odot U_g \cdot h_{t-1}) \\ h_t &= (1 - z_t) \odot g_t + z_t \odot h_{t-1}\end{aligned}$$

Donde:

- z_t es la puerta de actualización, que controla cuánto de la información previa h_{t-1} debe ser actualizada.
- r_t es la puerta de reinicio, que controla cuánto de la información previa h_{t-1} debe ser olvidada.
- g_t es el candidato para el nuevo estado oculto h_t , que se basa en la información actual x_t y el estado anterior h_{t-1} ponderado por la puerta de reinicio r_t .
- σ es la función sigmoide y \odot denota la multiplicación elemento por elemento.

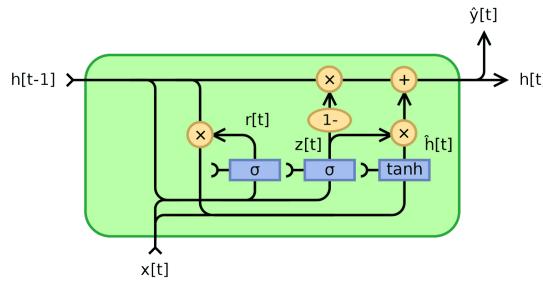


Figura 3.3: Esquema general de una célula GRU (Recuperado de Jeblad, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=66234713>

3.4.1. Entrenamiento de una red GRU

En un sentido similar al desarrollo de entrenamiento de las redes LSTM, los gradientes de cada uno de los componentes de la célula son:

$$\frac{\partial E}{\partial g_t} = \frac{\partial E}{\partial h_t} (1 - z_t)$$

⁸También podemos encontrar la implementación $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot g_t$ como podemos ver en [14]

$$\frac{\partial E}{r_t} = \frac{\partial E}{\partial h_t}(1 - z_t)W_g h_{t-1}[1 - g_t^2]$$

$$\frac{\partial E}{z_t} = \frac{\partial E}{\partial h_t}(h_{t-1} - g_t)$$

De la misma manera, los parámetros de la red quedan⁹:

$$d1 = \frac{\partial E}{\partial h_t}(1 - z_t)(1 - g_t^2)$$

$$d2 = [(d1) \odot W_h^T]h_{t-1}(r_t(1 - r_t))$$

$$d3 = [h_{t-1} \frac{\partial E}{\partial h_t} - g_t \frac{\partial E}{\partial h_t}](z_t(1 - z_t))$$

$$\frac{\partial E}{W_r} = h_{t-1}^T \odot (d2)$$

$$\frac{\partial E}{U_r} = x_t^T \odot (d2)$$

$$\frac{\partial E}{W_z} = h_{t-1}^T \odot (d3)$$

$$\frac{\partial E}{U_Z} = x_t^T \odot (d3)$$

$$\frac{\partial E}{W_h} = h_{t-1}^T \odot (d1)$$

$$\frac{\partial E}{U_h} = x_t^T \odot (d1)$$

⁹Estos gradientes fueron recuperados de [15]

Capítulo 4

Construcción del Modelo

Los modelos que a continuación se presentan son compuestos por la DWT y cada una de las redes que estudiamos en el capítulo anterior: la NARNN, con la arquitectura propuesta por Asmaa Y. Fathi, et. al. [4] por un lado, y la estructura planteada por Adusumilli, R. [16] para la la red con celulas LSTM (Long Short-Term Memory neural network) (LSTMnn) y la red con células GRU (GRUun), así como las redes que no cuentan con el pre-procesamiento de datos de la DWT.

Para la implementación, se usará *Python 3.11* ¹ ² pues su sintaxis simple y legible facilita la programación y comprensión del código, lo que agiliza el desarrollo de los modelos de redes neuronales. Además cuenta con una cantidad importante de módulos y marcos de trabajo específicamente diseñados para el desarrollo de aprendizaje profundo. Algunos de los más populares incluyen TensorFlow, PyTorch, Keras, y scikit-learn (que revisaremos más adelante). Cada uno de ellos los iremos mencionando a medida que se requieran. Otra ventaja de su versatilidad es que permite combinar el uso de bibliotecas específicas con otras funcionalidades propias del lenguaje, como manipulación de datos y visualización de resultados en un mismo entorno.

¹El repositorio con el código del proyecto se puede ver en https://github.com/MiguelAngelLiera/Stock_Exchange_NN_PP

²Las especificaciones del software utilizado se pueden ver en [C](#)

4.1. Descomposición de datos por DWT

A partir de este punto, se usarán los datos de **ACTINVRB** para cada uno de los pasos en la construcción de este y el siguiente capítulo. En este sentido, es un buen conjunto para la evaluación de los modelos desde que vemos la caída del valor de las acciones desde finales de 2019 (A partir de las doscientas semanas), hecho que se acentúa durante el primer año de la pandemia causada por la COVID-19. En palabras de Pablo Duarte, Director de Análisis y Estrategia de ACTINVER: "...la pandemia agregó mayor incertidumbre a las expectativas de crecimiento nacionales, mermando el ánimo de los inversionistas ya cautelosos por la crisis sanitaria." [17] Esta característica nos permite evaluar como reaccionan las arquitecturas ante eventos anómalos.

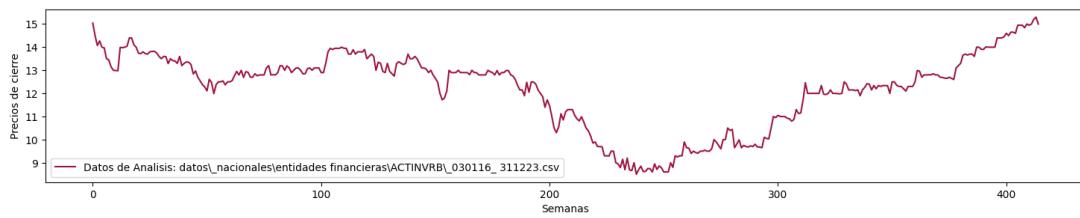


Figura 4.1: Precios de cierre semanal de **ACTINVRB** del 1 de enero de 2016 al 31 de diciembre de 2023

La biblioteca *Pywavelets* [18] cuenta con una implementación sólida, eficiente y fácil de usar de algoritmos basados en ondículas tanto para análisis en tiempo continuo como en tiempo discreto. Proporciona una amplia gama de funciones para realizar transformaciones de ondícula y manipular coeficientes. Entre estas podemos encontrar tanto para la descomposición de datos por DWT (función `dwt`) como para la reconstrucción de estos a partir de sus coeficientes (`idwt`, `upcoef`).

El primer paso de la descomposición es la elección de la ondícula madre. Se opta por una u otra dependiendo de las características de la serie de tiempo que se va a analizar. En general, debe ser en función del comportamiento de la serie original para que esta pueda ser reconstruida o analizada. Para series que impliquen cambios no suaves y repentinos es recomendable usar *bior3.5* [4]. Se trata de una ondícula madre que pertenece a la familia de ondículas biortogonales. La notación *bior3.5* indica que tiene un filtro de paso bajo de longitud tres y un filtro de paso alto de longitud cinco en la descomposición, es decir, un filtro de tres coeficientes

para la suavización (filtro de paso bajo) y un filtro de cinco coeficientes para la detección de detalles (filtro de paso alto). Así, nos da una buena representación de señales con cambios abruptos, lo que la hace útil en aplicaciones donde se requiere una alta capacidad de representación.

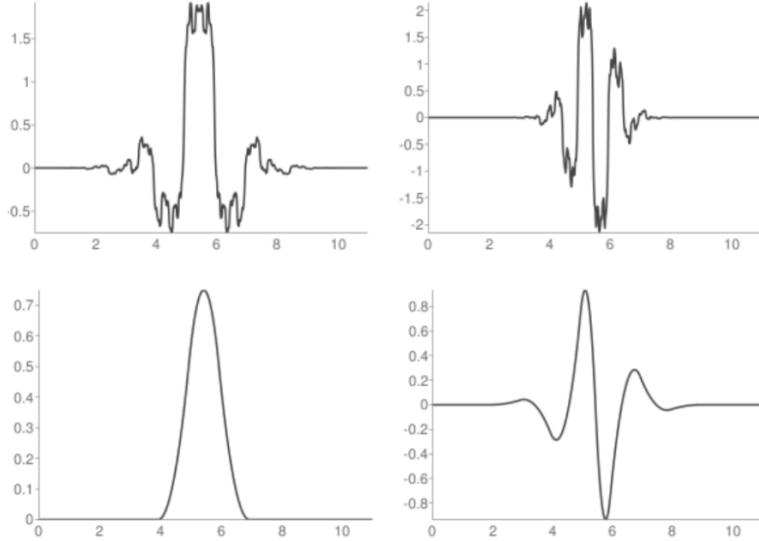


Figura 4.2: Funciones de descomposición (superior) y reconstrucción (inferior) de escala ϕ y de ondícula ψ

Para aplicar la DWT sobre los datos, es necesario realizar una extrapolación de los datos para que la descomposición en los extremos de la señal sea lo más precisa y limpia posible. Para nuestros fines, se escoge el modo *symmetric* ya que refleja de mejor manera la descomposición en componentes de baja frecuencia.

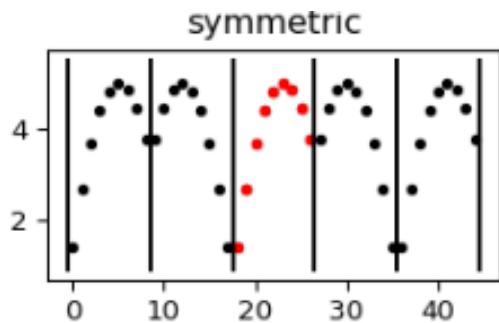


Figura 4.3: Extrapolación con modo *Symmetric*

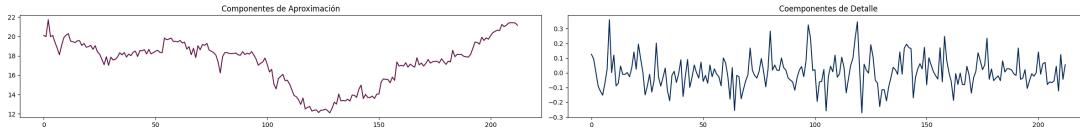


Figura 4.4: Componentes de Detalle y Aproximación de los datos de **ACTINVRB**

Antes de continuar necesitamos dividir el conjunto de datos en entrenamiento (70 %) y prueba (30 %), para la fase del entrenamiento. Más adelante, durante el capítulo cinco [5](#) se explica a detalle esta división.

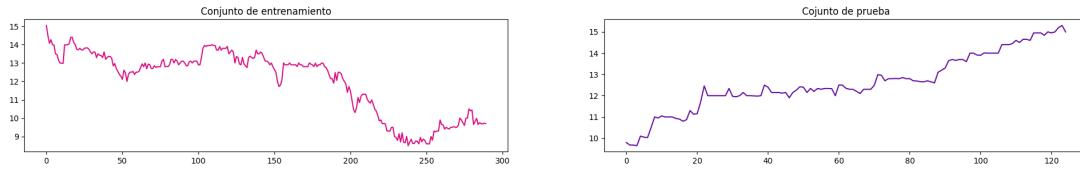


Figura 4.5: Conjunto de entrenamiento y prueba de los datos de **ACTINVRB**

Para la eliminación del ruido de la señal, usaremos una descomposición en cinco niveles empleando el algoritmo piramidal que se menciono en capítulos anteriores, atendiendo al hecho de suavizar la serie sin que pierda sus propiedades ^{[3](#)}.

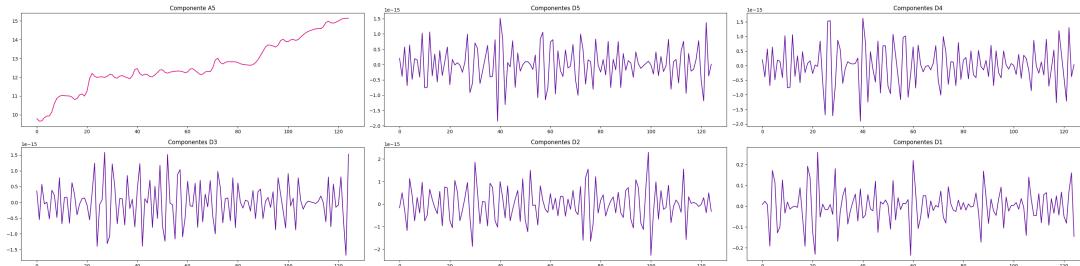


Figura 4.6: Componentes de detalle y aproximación en cinco niveles del conjunto de datos de prueba de **ACTINVRB**

³La implementación de la descomposición multinivel de la señal por la DWT se puede encontrar en [B](#)

Para cada uno de estos componentes se creará una red NARNN, LSTMnn y GRUnn con el objetivo de pronosticar su comportamiento.

4.2. Modelo NARNN

La arquitectura del modelo DWT-NARNN

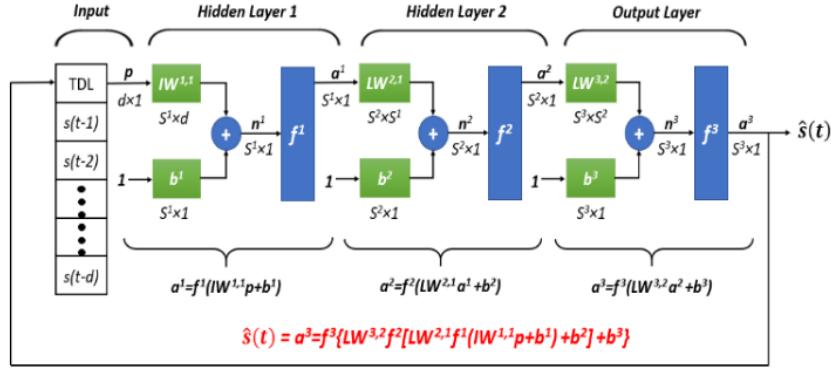


Figura 4.7: Arquitectura NARNN (Recuperado de [4])

En la figura 4.7 podremos encontrar la arquitectura de la NARNN. Se trata de RNN con una Linea de Tiempo Retrasado (LTR) que representa los datos cierre de las ocho semanas anteriores a la predicción. También encontramos dos capas ocultas cada una con diez neuronas típicas y una última capa de salida con una sola neurona. La función de activación en cada capa es la *logarítmica-sigmoide*, *tangente hiperbólica sigmoide* y una función lineal respectivamente.

Para su implementación se utilizó la biblioteca *pytorch* B:

4.3. Modelo LSTMnn

La red LSTMnn se toma la arquitectura:

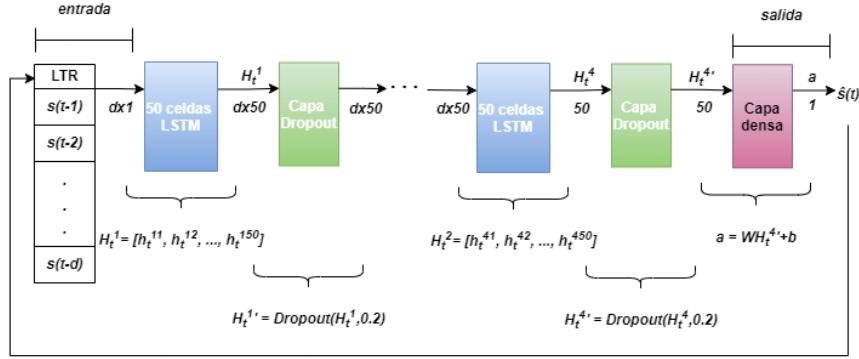


Figura 4.8: Arquitectura LSTMnn

La LSTMnn, al igual que la NARNN, presenta una LTR. Los ocho datos de entrada serán procesados por cuatro capas con cincuenta células LSTM cada una, y entre ellas cuatro capas de exclusión aleatorio o *dropout* al veinte por ciento, de manera que las salidas de esta cantidad de células de la capa inmediata anterior se trunquen a cero para evitar el sobre-ajuste. Finalmente encontramos una capa densamente conectada con una sola neurona para la salida.

La implementación fue resuelta a partir de la biblioteca *keras*. Se implementa la LSTMnn en B:

4.4. Modelo GRUUnn

Es una arquitectura par a la LSTMnn con la diferencia de que las capas con células LSTM son remplazadas por capas GRU, esto para mostrar una comparación equivalente y transparente entre el par de redes recurrentes. La red GRUUnn se toma la arquitectura:

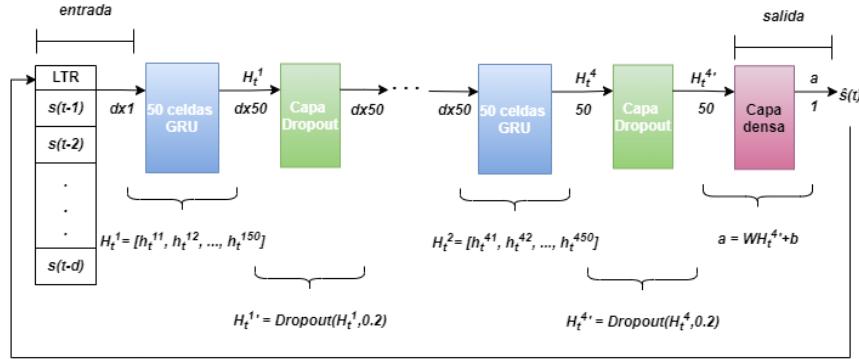


Figura 4.9: Arquitectura GRUUnn

La implementación fue resuelta a partir de la biblioteca *keras*. Se implementa la GRUUnn en [B](#):

Capítulo 5

Proceso de Entrenamiento

El método de entrenamiento aplicado a un modelo es a la par de importante que la elección de la arquitectura y paradigma de la red. El algoritmo de optimización, sumado a los parámetros que se usarán: tasa de aprendizaje, épocas, tamaño de lote son determinantes en el resultado del experimento. En el caso de las NARNNs, es la propagación hacia atrás partiendo del algoritmo Levenberg-Marquardt ¹ el proceso empleado. Mientras que ADAM es utilizado en el aprendizaje de las RNNs. Aunado a lo anterior, nos valdremos de metodologías sobre el proceso: el entrenamiento por reforzamiento del profesor y el entrenamiento auto-predictivo, además de un análisis exhaustivo para hallar los parámetros ideales.

Para estos dos enfoques se emplearán dos tipos de muestreos. Primero, procederemos partiendo el conjunto de datos en dos: una para el entrenamiento que comprende el setenta por ciento de este, tomando las muestras de forma aleatoria y lo mismo para el restante conjunto de prueba. Nos referiremos a esta propuesta como *muestreo aleatorio*. Luego se evaluará el desempeño de los modelos sobre estos datos en diferentes combinaciones de parámetros (tasa de aprendizaje y épocas). Posterior a ello, dividiremos nuevamente los conjuntos en la misma proporción, pero con la diferencia que separaremos el conjunto de manera temporal, es decir, el último treinta por ciento de la serie de tiempo, que serán nuestras últimas semanas de análisis formarán parte del conjunto de prueba, llamémosle *muestro temporal*, y de igual manera se examinarán los resultados.

¹Cuya implementación se ve reflejada en [B](#)

Para comparar el rendimiento de los modelos tomaremos como métricas de comparación al error cuadrático medio (*Mean Square Error*) (MSE) pues es tanto para las NARNNs y como para las RNNs su función de error, además que es una medida simple para medir el error en la predicción a priori. Esta medida se verá reflejada en las gráficas del presente capítulo.

Por otro lado para tratar a mayor profundidad el análisis nos valdremos de otras tres métricas que se verán reflejadas en el siguiente capítulo: la raíz del error cuadrático medio (*Rooted Mean Squared Error*) (RMSE) como una versión alterna al MSE. El porcentaje promedio de error absoluto (*Mean Absolute Percentage Error*) (MAPE) que describe la magnitud promedio del error absoluto del modelo. La Simetría Direccional (*Directional Symmetry*) (DS) es una medida que evalúa la dirección del pronóstico respecto a la dirección real de cambio en la serie de tiempo.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100$$

$$\text{DS} = \frac{100}{n} \sum_{i=1}^n d_i$$

Donde:

- n : Número total de observaciones.
- y_i : Valor real o verdadero en la observación i .
- \hat{y}_i : Valor predicho por el modelo para la observación i .
- $d_i = \begin{cases} 1, & \text{si } (y_i - y_{t-1})(\hat{y}_i - \hat{y}_{t-1}) \geq 0 \\ 0, & \text{de otra manera} \end{cases}$

Como se mencionó, para cada método de entrenamiento se encontraron los parámetros adecuados a partir de una búsqueda exhaustiva de la mejor combinación de

estos para lograr una predicción aceptable y evitando el sobre-ajuste. Dicho procedimiento se ve reflejado en las tablas de referencia que se ven a continuación (tasa de aprendizaje contra número de épocas) en las cuales se vera reflejado el MSE total de la predicción y en el caso de los modelos que cuentan una descomposición con la DWT se presenta MSE de los componentes / MSE de la reconstrucción.

5.1. Entrenamiento por reforzamiento del profesor

Cada característica del conjunto de datos, es decir, la entrada de toda red en cualquier lote de cualquier época del entrenamiento, forma parte de los datos originales: el precio de cierre semanal durante ocho semanas ($[t_{n-1}, t_{n-2}, \dots, t_{n-8}]$), a partir de las cuales dará como salida el valor de la novena semana (t_n). Luego, se emplea un corrimiento temporal del valor de una semana durante la evaluación de la siguiente característica (tomando $[t_{(n+1)-1}, t_{(n+1)-2}, \dots, t_{(n+1)-8}]$), para predecir la consecuente (t_{n+1}).

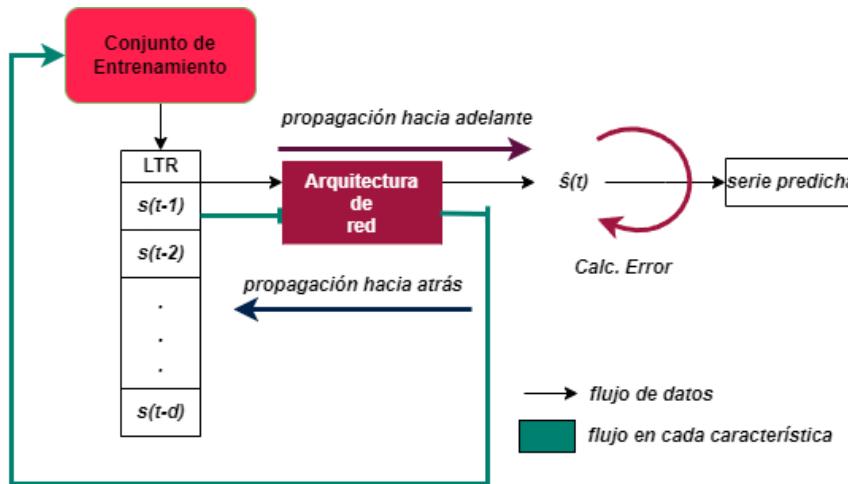


Figura 5.1: Diagrama del reforzamiento del profesor

5.1.1. NARNN

Cuando todas las características del lote cuentan con su correspondiente predicción es cuando LM ejecuta un paso, modificando los pesos y sesgos de la red a partir del gradiente y matriz hessiana de la función de error.

Se analizó únicamente la combinación de una tasa de aprendizaje contra entrenamiento, manteniendo un tamaño de lote de uno. Comenzamos arbitrariamente en cualquier punto de la tabla, y se direcciona la búsqueda hacia donde el error en cada época parezca disminuir lo suficiente.

T. Aprendizaje	Estandar			
	Epocas			
0.001	10	15	20	30
0.01	1.6			1.59
0.03	2.35			
0.04		4.81		0.261
0.05		56.89		
0.06			14.59	
0.08		77.23	37.53	
0.2		334.53		
0.5	18.53			

Figura 5.2: Tabla comparativa entre combinaciones de parámetros de la NARNN: tasa de aprendizaje contra número épocas

Como vemos, la mejor intersección se da en [0.04,30], obteniendo las siguientes predicciones en ambos muestreos:

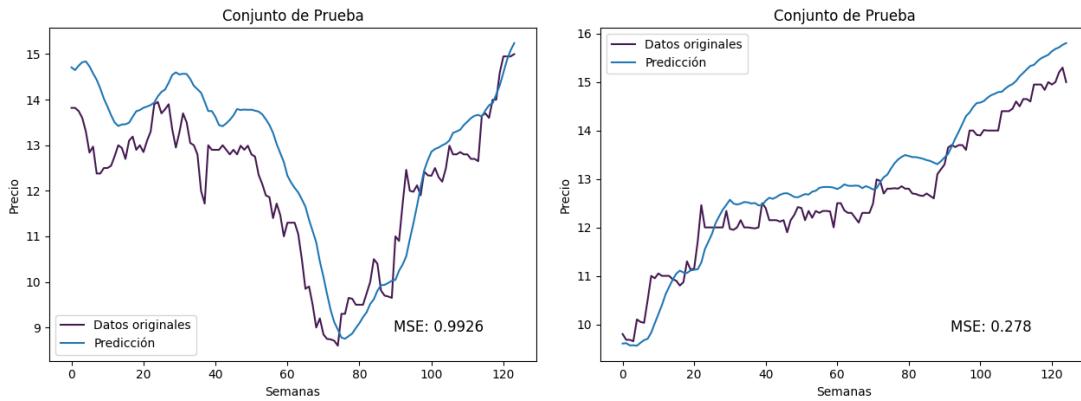


Figura 5.3: Desempeño de la NARNN en conjunto de prueba: Muestro aleatorio y temporal

Cabe mencionar que encontrar los parámetros adecuados depende directamente de como estén inicializados estos. En general, para tanto la NARNN y la DWT-NARNN y para la longitud de datos de estudio representa un reto importante a la hora de encontrarlos ².

²Como se ve reflejado en la tabla, el número de intentos en comparación a los de las RNNs es mayor

5.1.2. DWT-NARNN

Se procedió de manera similar que la NARNN, con la diferencia que se encontraron diferentes combinaciones de parámetros para las redes dedicadas a los componentes de detalle y de aproximación.

T. Aprendizaje	Estandar							
	5	10	15	30	40	50	60	100
0.015	7.96				431.37			
0.017			0.16/5.93			0.12/4.84		
0.017			0.14/5.62					
0.017			0.11/6.86					
0.018			0.6/2.45			0.23/8.29		0.11/4.93
0.0185			0.12/4.38					
0.02				0.08/3.72		0.10/4.21		
0.025				0.1/4.59				
0.025				0.16/6.86				
0.03		2.15/4.65						
0.04			4.73 2.34/5.49	0.17/5.92		4.98	0.16/4.08	
0.05				0.3/7.57				
0.06				13.3/413.86				
0.08				12.52/407.76				
0.1	27.57		0.67/24.35					
0.2			96.11					
0.5	3.87							

Figura 5.4: Tabla comparativa entre combinaciones de parámetros de la DWT-NARNN: tasa de aprendizaje contra número épocas. (Los datos resaltados en verde representan la mejor combinaciones para las componentes de alta frecuencia y azul para los de baja).

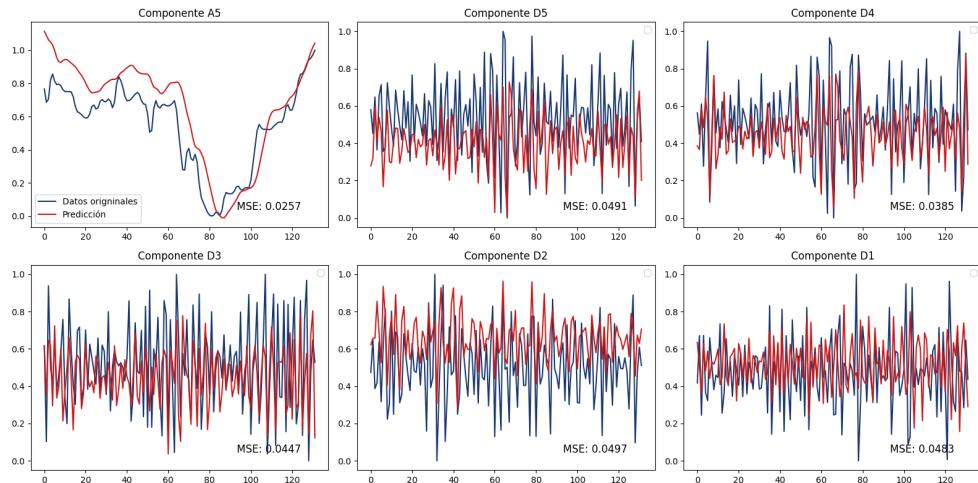


Figura 5.5: Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro aleatorio.

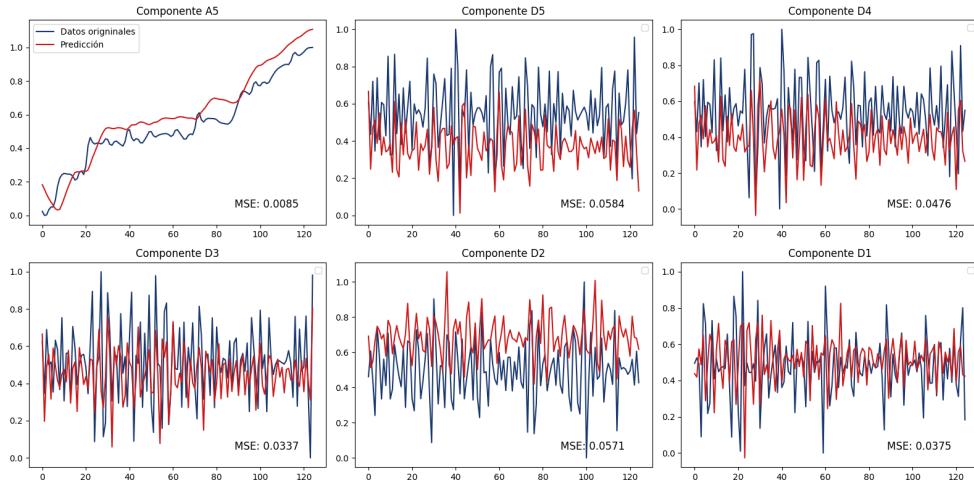


Figura 5.6: Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro temporal.

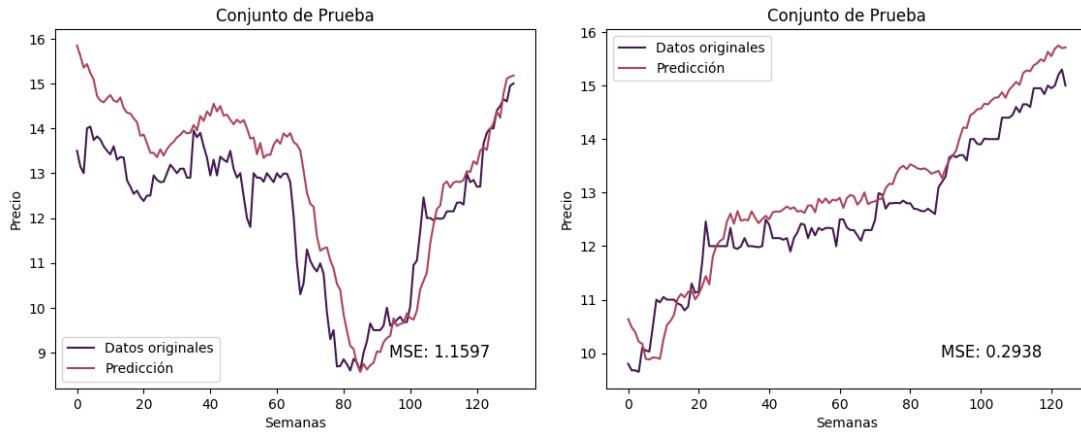


Figura 5.7: Desempeño de la DWT-NARNN en la reconstrucción del conjunto de prueba: Muestro aleatorio y temporal.

5.1.3. LSTMnn

Encontrar los parámetros ideales en este caso empleó un menor esfuerzo. Partiendo de los datos que nos proporciona Adusumilli, Roshan, una tasa de aprendizaje de 0.0001, un tamaño de lote de 32 y 60 épocas, se obtuvieron muy buenos resultados en ambos muestreos:

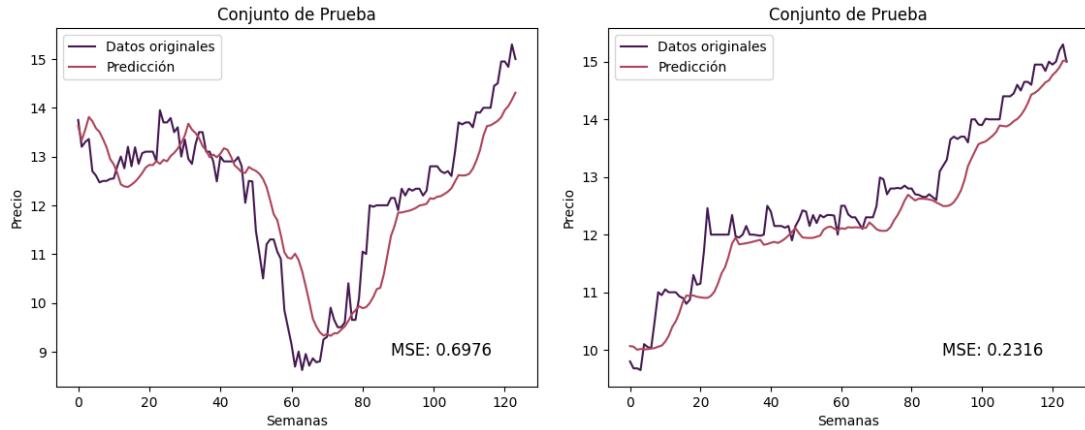


Figura 5.8: Desempeño de la LSTMnn en conjunto de prueba: Muestro aleatorio y temporal.

5.1.4. DWT-LSTMnn

Se usaron los mismos parámetros que para la LSTMnn en la predicción de la componente $A5$ de baja frecuencia. Mientras que para las demás se procedió de manera similar a la NARNN y a la DWT-NARNN.

		Estandar	
		Componente	Epcas
T. Aprendizaje 0.01	D5	60	90
	D4	0.0171	0.016
	D3	0.0094	0.0099
	D2	0.0099	
	D1	0.0083	

Figura 5.9: Tabla comparativa entre parámetros para componentes de alta frecuencia de la DWT-LSTMnn: tasa de aprendizaje contra número épocas por cada componente.

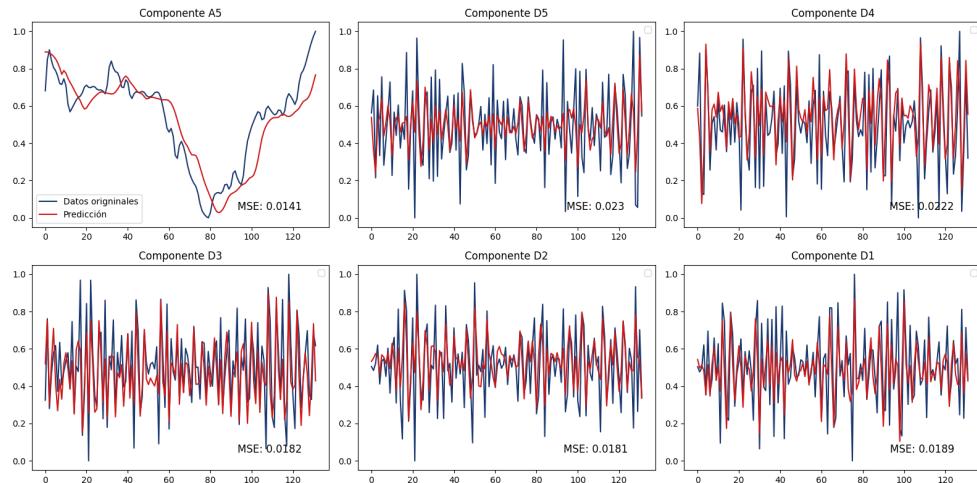


Figura 5.10: Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro aleatorio.

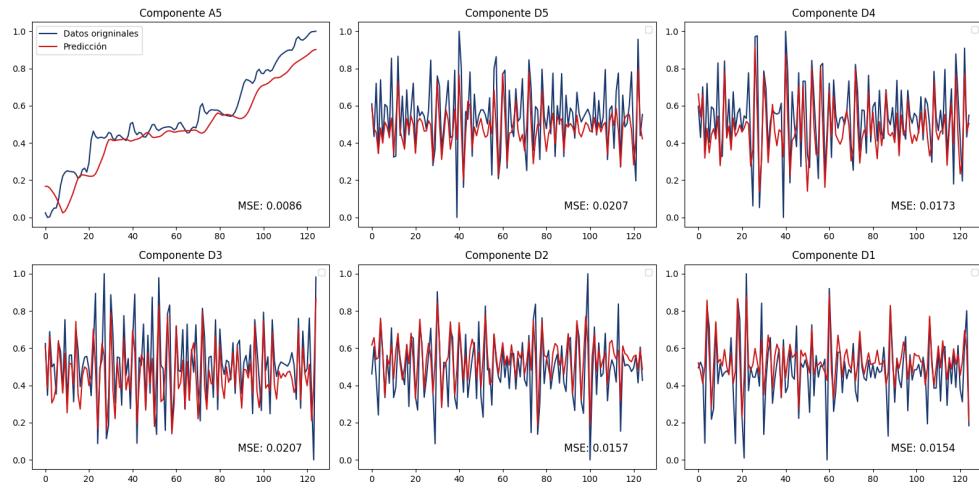


Figura 5.11: Predicciones de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro temporal.

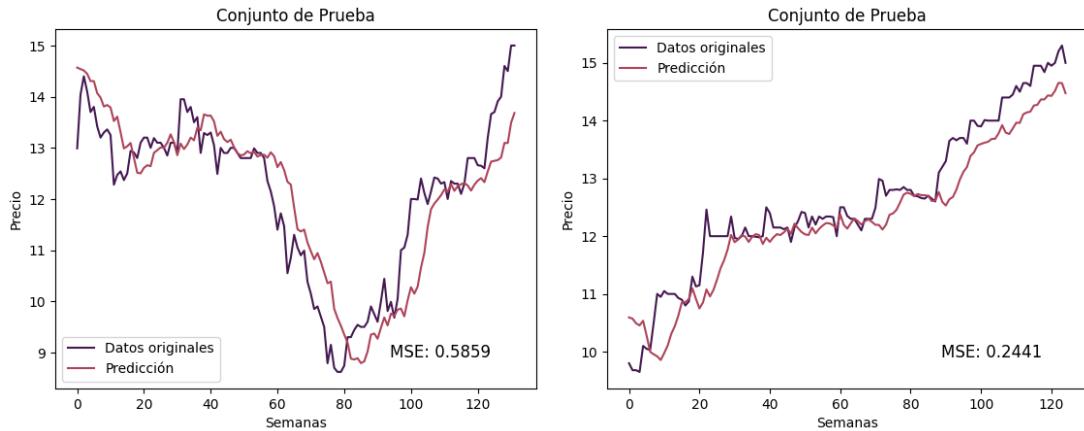


Figura 5.12: Desempeño de la DWT-LSTMnn en la reconstrucción del conjunto de prueba: Muestro aleatorio y temporal.

5.1.5. GRUUnn

Al poseer una arquitectura paralela a las LSTMnn y la DWT-GRUUnn a la DWT-LSTMnn, se hicieron uso de los mismos datos de entrenamiento. Obteniendo así:

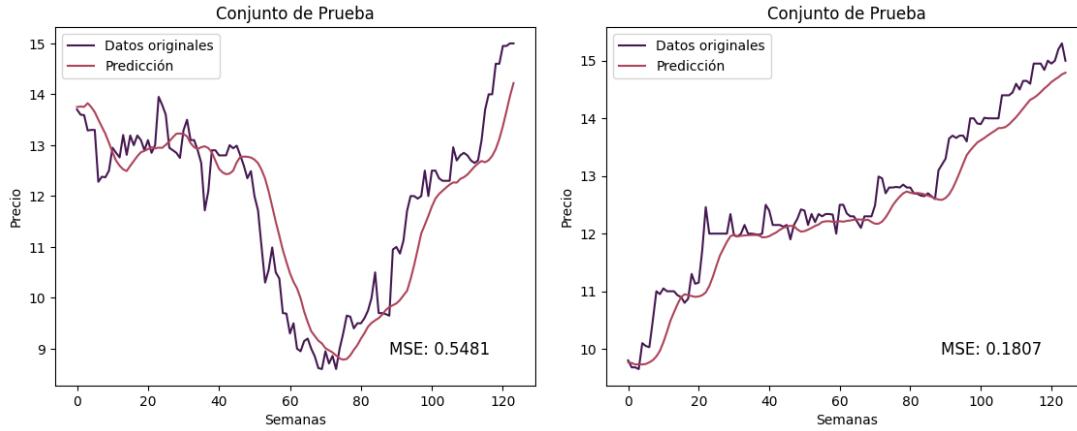


Figura 5.13: Desempeño de la GRUUnn en conjunto de prueba: Muestro aleatorio y temporal

5.1.6. DWT-GRUun

Y también:

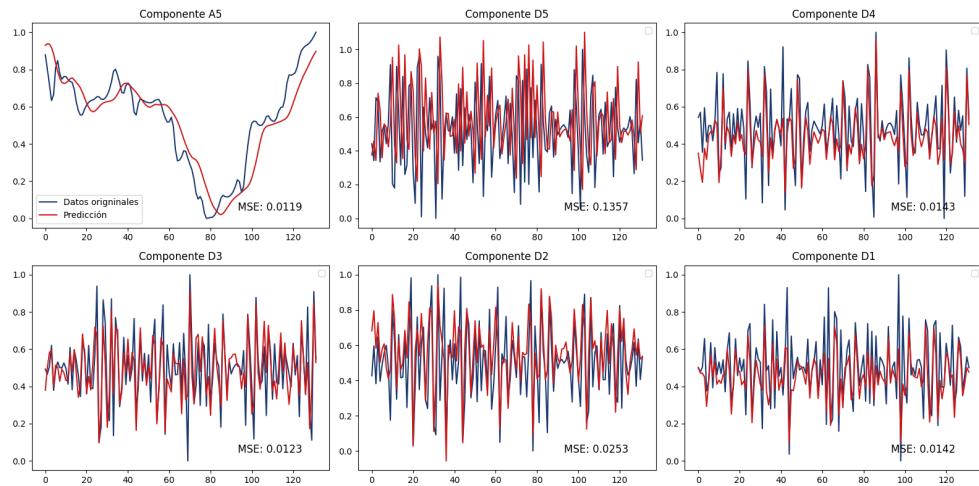


Figura 5.14: Predicciones de la DWT-GRUun de los componentes del conjunto de pruebas de muestro aleatorio

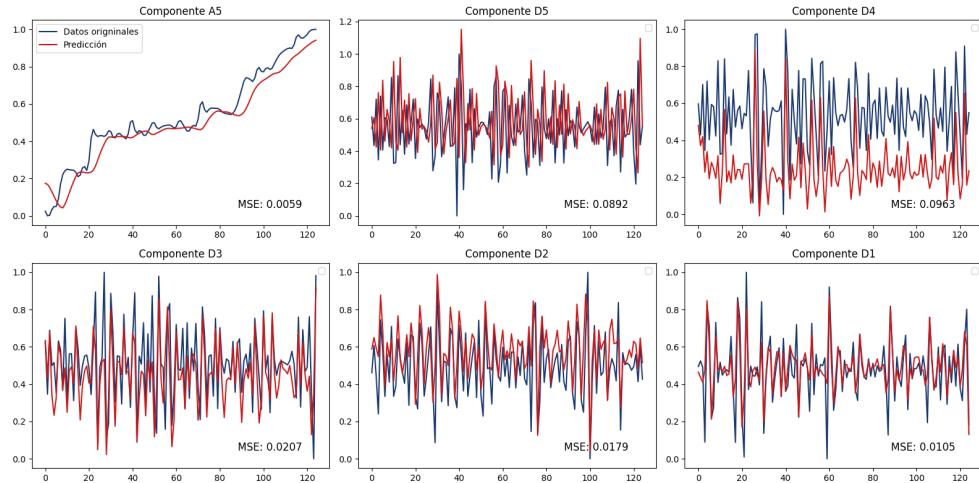


Figura 5.15: Predicciones de la DWT-GRUun de los componentes del conjunto de pruebas de muestro temporal

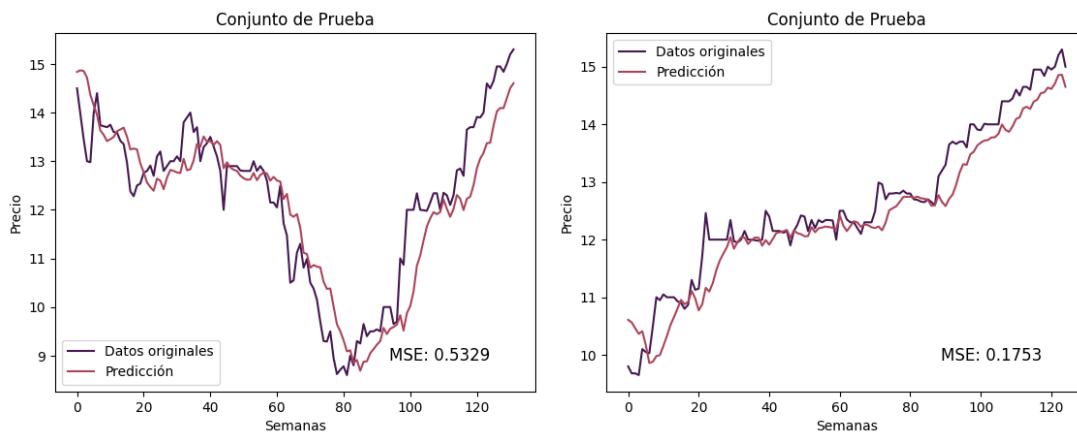


Figura 5.16: Desempeño de la DWT-GRU_nn en la reconstrucción del conjunto de prueba: Muestro aleatorio y temporal.

5.2. Entrenamiento Auto-predictivo

De la misma manera que se vio durante la sección anterior la entrada del conjunto de datos serán los precios de las primeras ocho semanas del conjunto de entrenamiento, para predecir el valor de la novena semana. Esa novena semana se concatenará como parte de la entrada en la predicción del precio siguiente semana, ocupando el lugar del octavo valor y repitiendo esto en las siguientes iteraciones. De esta forma, durante el cálculo de cada predicción se tomará como entrada a $[\hat{t}_{n-1}, \hat{t}_{n-2}, \dots, \hat{t}_{n-8}]$ donde \hat{t}_i es la predicción i -ésima de la red, obteniendo \hat{t}_n .

Es importante recalcar que este experimento es de gran importancia para nuestro propósito, pues evaluará la autonomía de las redes para identificar tendencias de largo plazo. A continuación se muestra el desempeño de cada una de las redes bajo este concepto.

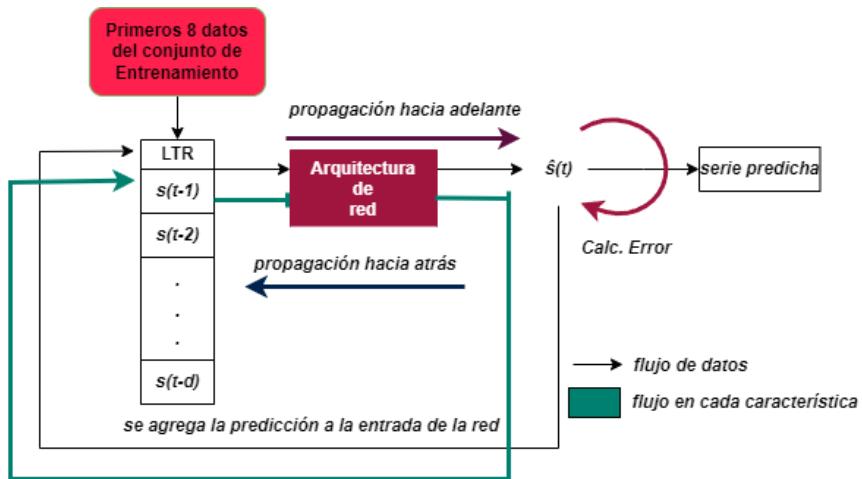


Figura 5.17: Diagrama del entrenamiento auto-predictivo

5.2.1. NARNN

La obtención de los parámetros ocurre de la misma manera analítica.

		Auto-predictivo		
		Epochas		
T. Aprendizaje		20	30	40
	0.0005			34.27
0.0008			2.09	12.71
0.001			23.52	26.13
0.01				34.55
	0.04			33.27
0.05				35
	0.06		3.87	2.86
0.1		37.53		
	0.2		38.25	

Figura 5.18: Tabla comparativa entre parámetros de la NARNN: tasa de aprendizaje contra número épocas durante entrenamiento auto-predictivo.

El mejor resultado que se pudo encontrar es el siguiente:

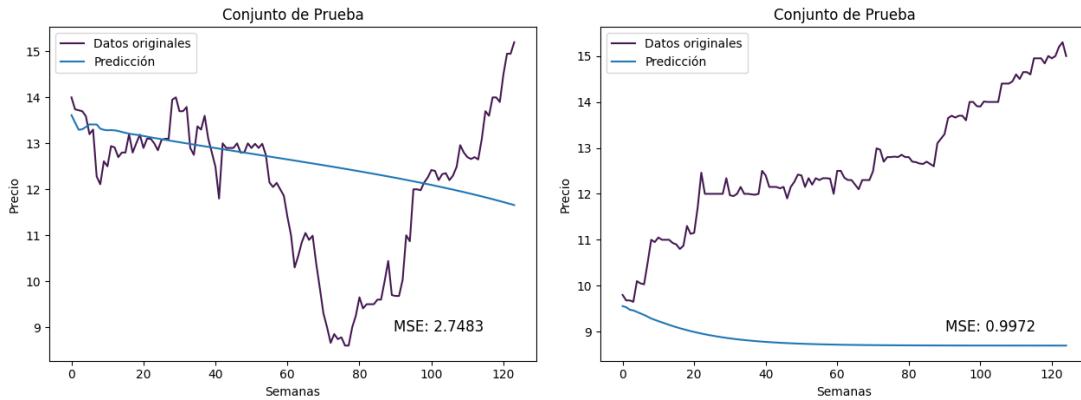


Figura 5.19: Desempeño auto-predictivo de la NARNN en conjunto de prueba: Muestro aleatorio y temporal.

Vemos que en lo que respecta a la predicción, no encaja bien con los valores esperados hecho que surge desde que los primeros resultados de predicción arrastran consigo cierto error y que, a la larga, este error aumenta dándonos un resultado

cada vez más inexacto. Por esta razón, se decidió agregar un nuevo parámetro, con el fin de estabilizar las primeras predicciones que la red nos da, de manera que los valores subsecuentes a estas se encuentren no tan alejadas al valor real.

Se empleó una técnica de decaimiento en la tasa de aprendizaje durante una época, de manera que en los lotes más próximos al inicio de la serie serán para los cuales la red aprenda mejor, garantizando que las predicciones en un inicio sean más acertadas y en consecuencia las siguientes arrastren una cantidad menor de error. *El factor de decaimiento (FD)*:

Algoritmo 3: Ajuste del factor de decaimiento del aprendizaje

```

1 alComenzarLote (lote_actual, perdida_actual):
2   si perdida_actual ≤ tolerancia y lote_designado == lote_actual entonces
3     factor_de_decaimiento = factor_de_decaimiento × 0.8
4     lote_designado = lote_designado + 1
      // El lote actual indica a partir de qué lote se empezará
      // a tomar en cuenta el FD
5   fin
6   tasa_actual ←  $\frac{\text{tasa\_inicial}}{1 + \text{factor\_de\_decaimiento} \times \text{iteracion}}$ ; // Calcula la nueva tasa
      // de aprendizaje
7   iteracion++ // Incrementa el número de iteración
8   regresa tasa_actual ;
  
```

Así, durante cada época si la pérdida actual es menor a la tolerancia y el lote designado es el actual, el FD cae, y la penalización de la tasa de aprendizaje en los siguientes lotes es menor. De cualquier forma se disminuye la tasa actual en función de la tasa inicial, el FD y el número de época o lote en el que nos encontramos.

Aplicando lo anterior, se tiene:

		Auto-predictivo (Versión 2)	
		Epocas	
T. Aprendizaje	FD	30	60
		0.05	62.86
0.06		1.37	
0.1	0.5	63.26	
0.4		3.71	
0.5		189.83	
0.8		1.9	

Figura 5.20: Tabla comparativa entre parámetros de la NARNN: tasa de aprendizaje contra número épocas y factor de decaimiento durante entrenamiento auto-predictivo.

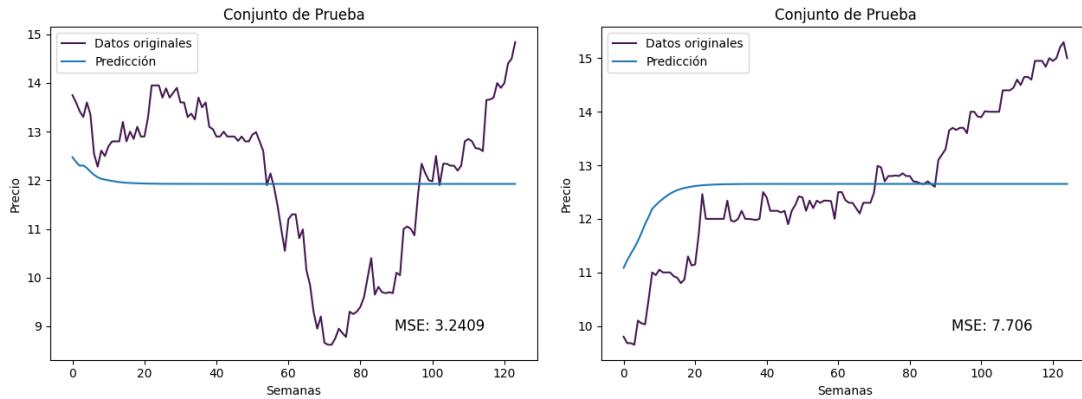


Figura 5.21: Desempeño auto-predictivo de la NARNN en conjunto de prueba bajo la implementación del factor de decaimiento: Muestro aleatorio y temporal.

Encontrándonos así una mejor predicción en general ³.

³Notemos que esta técnica 'obliga' a la red a aprender mejor los primeros datos del conjunto en decremento de entendimiento que tiene de los subsecuentes. Hecho que perjudica a la predicción cualquier otro tipo de predicción que no sea enteramente auto-predictiva.

5.2.2. DWT-NARNN

En el mismo sentido que la sección anterior. Se usaron los mismos parámetros que para la NARNN para el componente de baja frecuencia. Para los demás se encontró que un enfoque sin FD daba buen resultado.

Auto-predictivo				
T. Aprendizaje	Epocas			
	30	40	50	60
0.000001	0.17/6.44		0.15/5.82	
0.00001			0.14/5.39	
0.00002			0.13/5.28	
0.0001	0.18/6.93			
0.001	0.15/6.21			
0.01	0.16/6.39		0.04/1.41	
0.025	0.17/6.62			
0.05	0.18/7.04			
0.1	0.17/6.64			

Figura 5.22: Tabla comparativa entre parámetros de la DWT-NARNN para componentes de alta frecuencia: tasa de aprendizaje contra número épocas durante entrenamiento auto-predictivo.

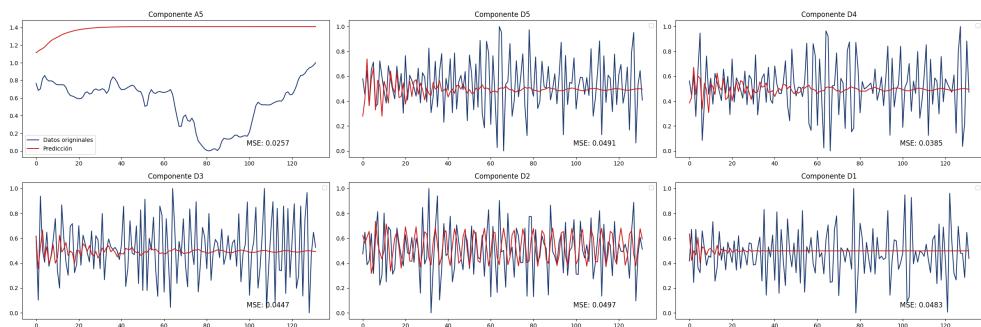


Figura 5.23: Predicciones auto-predictivas de la DWT-NARNNnn de los componentes del conjunto de pruebas de muestro aleatorio.

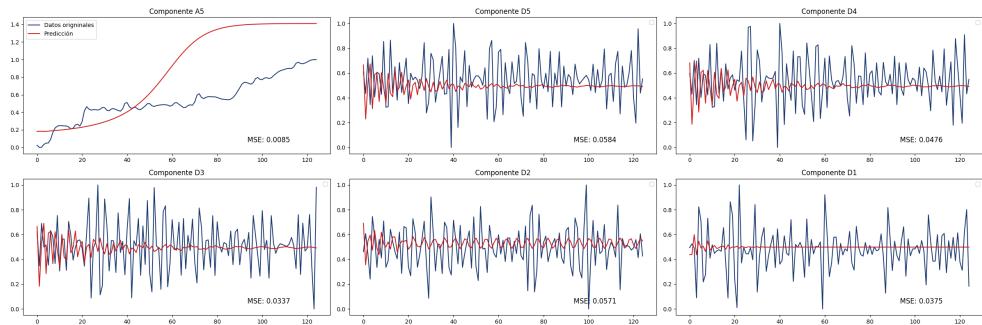


Figura 5.24: Predicciones auto-predictivas de la DWT-NARNNnn de los componentes del conjunto de pruebas de muestro temporal.

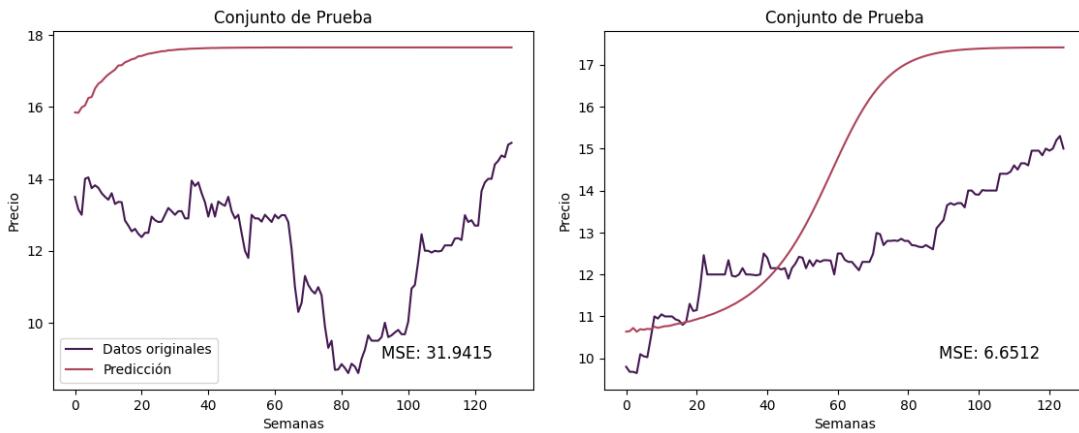


Figura 5.25: Desempeño auto-predictivo de la DWT-NARNNnn en conjunto de prueba: Muestro aleatorio y temporal.

5.2.3. LSTMnn

En el mismo sentido que la NARNN, los resultados del entrenamiento auto-predictivo sin implementar el factor de decaimiento no son muy alentadores.

Auto-predictivo		
	Epochas	
T. Aprendizaje	60	100
0.000005	4.26	
0.000008	4.36	
0.00001	4.87	
0.00003	3.17	
0.00005	2.34	
0.00006	2.09	1.92
0.0001	2.03	
0.00012	2.02	
0.005	34.27	
0.015	6.6	
0.05	5.85	

Figura 5.26: Tabla comparativa entre parámetros de la LSTMnn: tasa de aprendizaje contra número épocas en un entrenamiento auto-predictivo.

Aplicando un estudio de la misma manera que la NARNN. Tenemos:

Auto-predictivo							
T. Aprendizaje	FD	Epochas	FD	Epochas	FD	Epochas	
0.0001		60		60		100	
0.01	1.2						1.28
0.08		1.8111			1.76		
0.5			0.5		1.7484		0.1
					1.9777		

Figura 5.27: Tabla comparativa entre parámetros de la LSTMnn: tasa de aprendizaje contra número épocas en un entrenamiento auto-predictivo con factor de decaimiento.

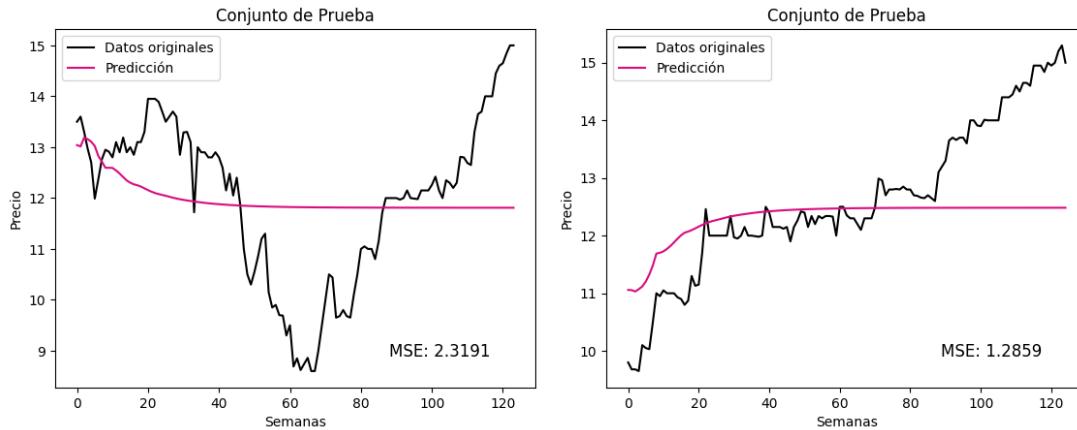


Figura 5.28: Desempeño auto-predictivo de la LSTMnn en conjunto de prueba: muestreo aleatorio y temporal.

5.2.4. DWT-LSTMnn

Para este entrenamiento, se obtuvieron los siguientes resultados.

Auto-predictivo					
	Epochas				
T. Aprendizaje	30	40	50	60	
0.000001	0.17/6.44			0.15/5.82	
0.00001				0.14/5.39	
0.00002				0.13/5.28	
0.0001	0.18/6.93				
0.001	0.15/6.21				
0.01	0.16/6.39			0.04/1.41	
0.025	0.17/6.62				
0.05	0.18/7.04				
0.1	0.17/6.64				

Figura 5.29: Tabla comparativa entre parámetros de la DWT-LSTMnn: tasa de aprendizaje contra número épocas y factor de decaimiento durante entrenamiento auto-predictivo

Tomando la misma combinación de parámetros para las redes destinadas a cada componente.

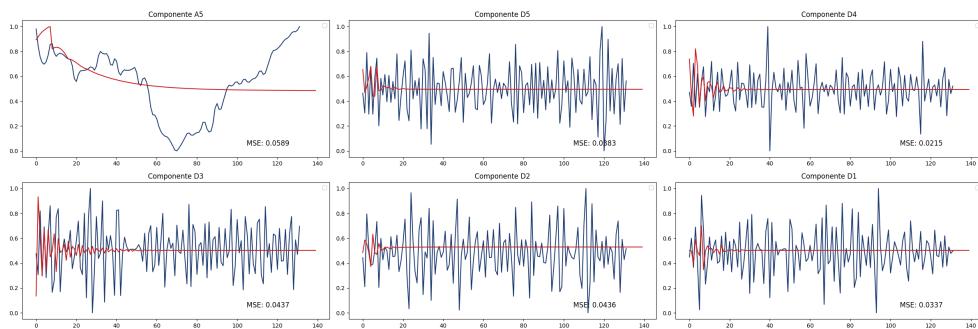


Figura 5.30: Predicciones auto-predictivas de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestro aleatorio.

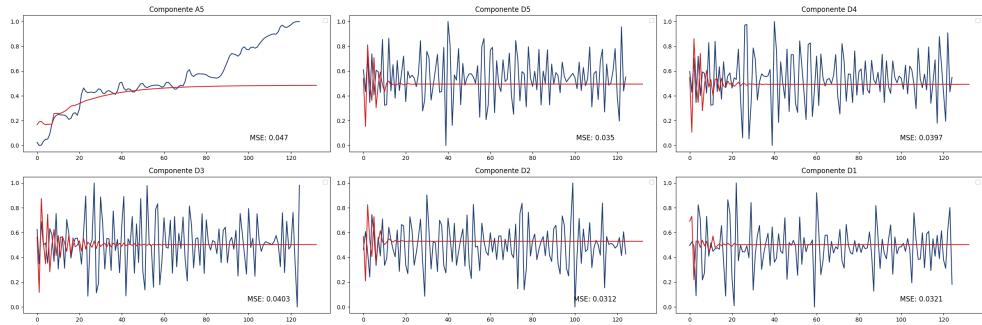


Figura 5.31: Predicciones auto-predictivas de la DWT-LSTMnn de los componentes del conjunto de pruebas de muestreo temporal.

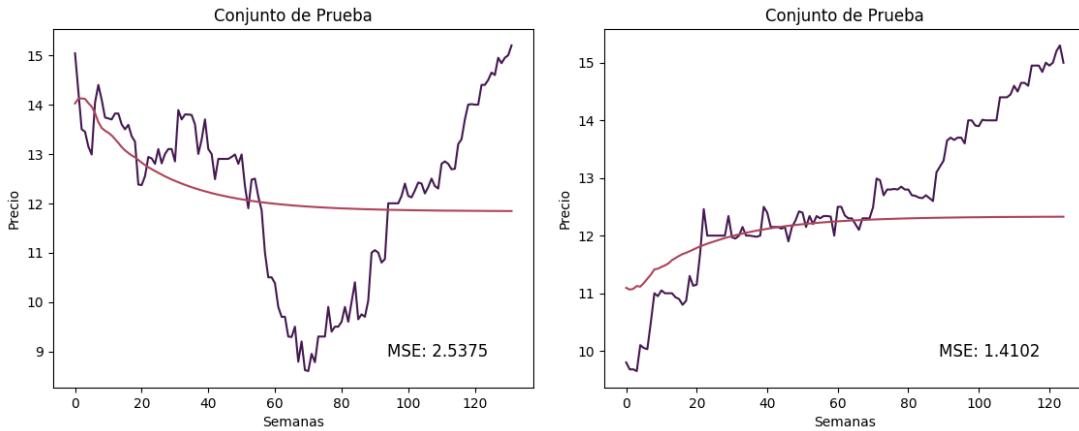


Figura 5.32: Desempeño auto-predictivo de la DWT-LSTMnn en conjunto de prueba: muestreo aleatorio y temporal.

5.2.5. GRUun

De la misma manera y parámetros que la LSTMnn:

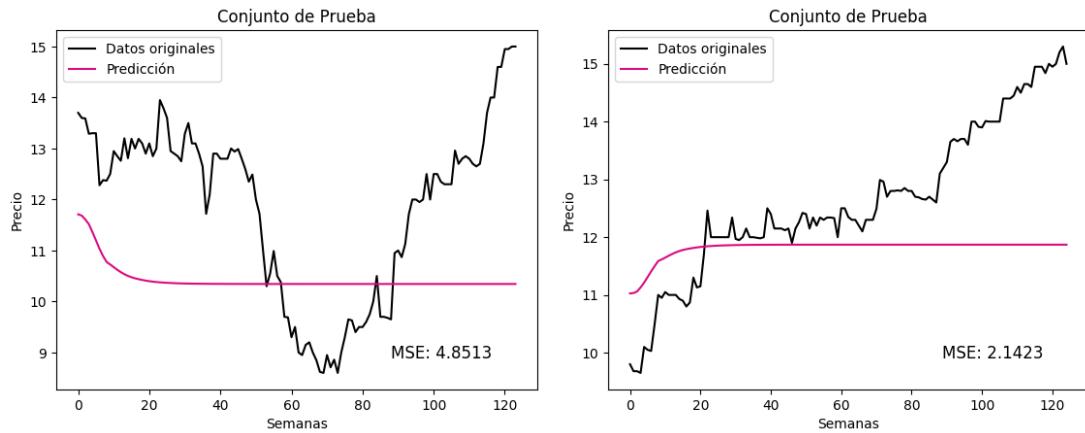


Figura 5.33: Desempeño auto-predictivo de la GRUUn en conjunto de prueba.

5.2.6. DWT-GRU_{nn}

De la misma manera y parámetros que la DWT-LSTM_{nn}:

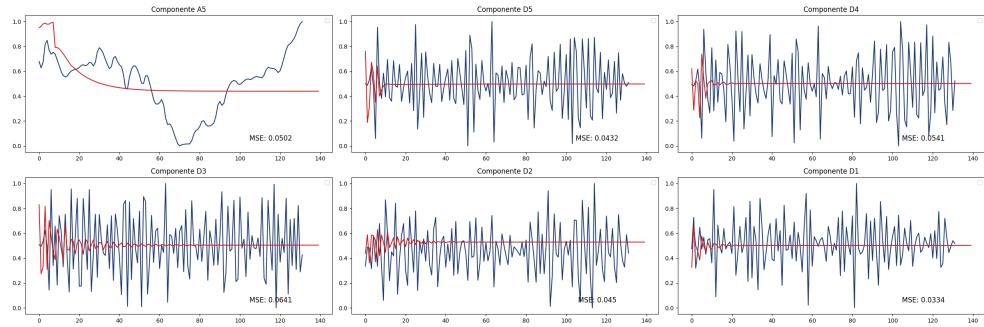


Figura 5.34: Predicciones auto-predictivas de la DWT-GRU_{nn} de los componentes del conjunto de pruebas de muestro aleatorio.

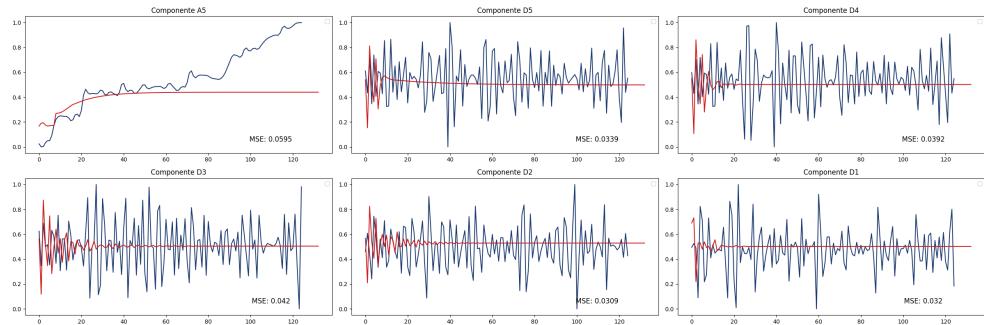


Figura 5.35: Predicciones auto-predictivas de la DWT-LSTM_{nn} de los componentes del conjunto de pruebas de muestro temporal.

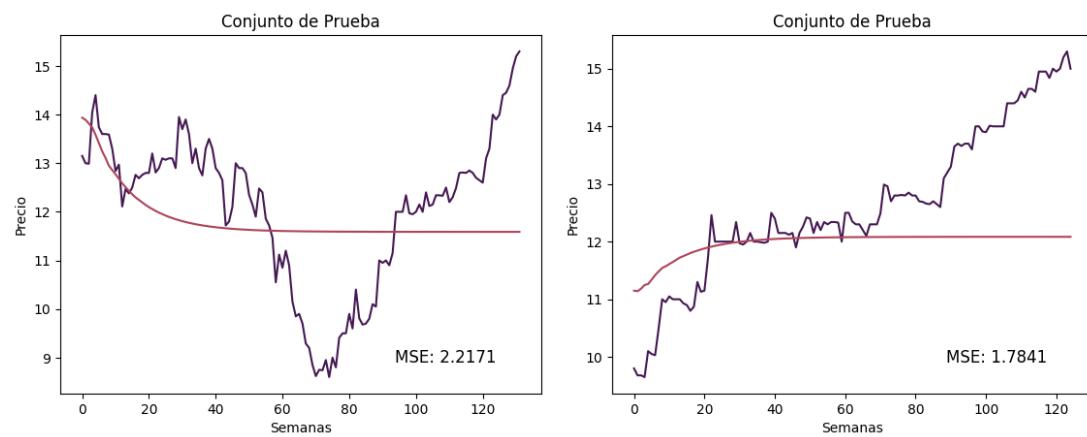


Figura 5.36: Desempeño auto-predictivo de la DWT-GRU_{nn} en conjunto de prueba: muestreo aleatorio y temporal.

Capítulo 6

Evaluación de desempeño

En este capítulo se muestran los resultados que se obtuvieron en las métricas mencionadas. Para los los modelos sin DWT se muestra únicamente el RMSE, MAPE y DS de la predicción del conjunto de prueba contra los precios reales. Para los restantes se pinta un desglose de cada métrica por cada componente y la reconstrucción total del pronóstico del conjunto para pruebas.

6.1. Predicción Estándar

	RMSE	MAPE (%)	DS (%)
NARNN — muestreo aleatorio			
Predicción de c_prueba	1.014	7.21322	63.7097
DWT-NARNN — muestreo aleatorio			
Reconstrucción de c_prueba	1.0769	7.397269	65.1515
Componente A5	1.0502	7.186337	62.8788
Componente D5	0.0000	133.993170	68.9394
Componente D4	0.0000	109.625883	62.8788
Componente D3	0.0000	102.676865	74.2424
Componente D2	0.0000	154.036830	70.4545
Componente D1	0.1402	315.594656	58.3333
LSTM — muestreo aleatorio			
Predicción de c_prueba	0.8335	6.015341	62.0968
DWT-LSTM — muestreo aleatorio			
Reconstrucción de c_prueba	0.8689	5.887960	68.1818
Componente A5	0.8488	5.699098	58.3333
Componente D5	0.0000	62.915075	81.0606
Componente D4	0.0000	95.984062	84.8485
Componente D3	0.0000	68.514078	79.5455
Componente D2	0.0000	69.674597	78.7879
Componente D1	0.1195	1924.748875	81.8182
GRU — muestreo aleatorio			
Predicción de c_prueba	0.7595	4.990504	68.5484
DWT-GRU — muestreo aleatorio			
Reconstrucción de c_prueba	0.7087	4.932105	70.4545
Componente A5	0.6824	4.713478	61.3636
Componente D5	0.0000	129.506225	40.9091
Componente D4	0.0000	125.145635	87.1212
Componente D3	0.0000	69.453592	82.5758
Componente D2	0.0000	62.306607	87.1212
Componente D1	0.1439	264.549962	83.3333

	RMSE	MAPE (%)	DS (%)
NARNN — muestreo temporal			
Predicción de c_prueba	0.5272	3.728504	57.6
DWT-NARNN — muestreo temporal			
Reconstrucción de c_prueba	0.5421	3.895320	77.6
Componente A5	0.5234	3.788553	57.6
Componente D5	0.0000	174.387157	55.2
Componente D4	0.0000	154.766175	65.6
Componente D3	0.0000	107.227901	68.0
Componente D2	0.0000	204.159326	66.4
Componente D1	0.0987	445.400429	67.2
LSTM — muestreo temporal			
Predicción de c_prueba	0.4807	3.023597	58.4
DWT-LSTM — muestreo temporal			
Reconstrucción de c_prueba	0.4940	3.052687	71.2
Componente A5	0.5104	3.194822	50.4
Componente D5	0.0000	82.476074	77.6
Componente D4	0.0000	84.153618	80.8
Componente D3	0.0000	85.620667	76.0
Componente D2	0.0000	78.178528	80.0
Componente D1	0.0614	232.752878	76.0
GRU — muestreo temporal			
Predicción de c_prueba	0.4149	2.425552	58.4
DWT-GRU — muestreo temporal			
Reconstrucción de c_prueba	0.4186	2.538661	74.4
Componente A5	0.4200	2.560362	52.8
Componente D5	0.0000	165.702431	28.8
Componente D4	0.0000	271.978322	84.0
Componente D3	0.0000	84.694473	80.8
Componente D2	0.0000	101.649577	83.2
Componente D1	0.0507	162.795013	76.0

6.2. Predicción Auto-regresiva

	RMSE	MAPE (%)	DS (%)
NARNN — muestreo aleatorio			
Predicción de c_prueba	1.5902	11.690531	83.0645
DWT-NARNN — muestreo aleatorio			
Reconstrucción de c_prueba	1.0769	7.397269	65.1515
Componente A5	5.6517	47.323432	59.8485
Componente A5	5.6362	47.178851	56.0606
Componente D5	0.0000	100.315032	51.5152
Componente D4	0.0000	93.483034	47.7273
Componente D3	0.0000	87.791170	45.4545
Componente D2	0.0000	114.652779	55.3030
Componente D1	0.1212	126.322933	58.3333
LSTM — muestreo aleatorio			
Predicción de c_prueba	1.5654	11.118145	56.4516
DWT-LSTM — muestreo aleatorio			
Reconstrucción de c_prueba	1.4242	10.028115	50.7576
Componente A5	1.4135	9.956542	45.4545
Componente D5	0.0000	89.239022	56.8182
Componente D4	0.0000	86.142969	40.1515
Componente D3	0.0000	97.384352	55.3030
Componente D2	0.0000	86.135574	58.3333
Componente D1	0.1619	562.130456	46.9697
GRU — muestreo aleatorio			
Predicción de c_prueba	1.8843	14.017671	72.5806
DWT-GRU — muestreo aleatorio			
Reconstrucción de c_prueba	1.5201	11.193495	56.0606
Componente A5	1.5034	11.071481	45.4545
Componente D5	0.0000	79.033183	50.7576
Componente D4	0.0000	86.834054	70.4545
Componente D3	0.0000	88.958365	47.7273
Componente D2	0.0000	79.117372	53.0303
Componente D1	0.1516	137.442302	52.2727

	RMSE	MAPE (%)	DS (%)
NARNN — muestreo temporal			
Predicción de c_prueba	1.1734	7.394705	76.8
DWT-NARNN — muestreo temporal			
Reconstrucción de c_prueba	2.5790	16.083149	60.8
Componente A5	2.5705	16.032415	58.4
Componente D5	0.0000	103.209729	45.6
Componente D4	0.0000	101.234477	43.2
Componente D3	0.0000	92.928121	44.8
Componente D2	0.0000	106.261789	48.0
Componente D1	0.0867	166.880758	61.6
LSTM — muestreo temporal			
Predicción de c_prueba	1.134	6.435196	60.0
DWT-LSTM — muestreo temporal			
Reconstrucción de c_prueba	1.1875	6.243048	59.2
Componente A5	1.1900	6.227096	59.2
Componente D5	0.0000	94.481289	75.2
Componente D4	0.0000	92.125017	46.4
Componente D3	0.0000	79.400759	55.2
Componente D2	0.0000	96.781732	72.0
Componente D1	0.0885	193.398147	47.2
GRU — muestreo temporal			
Predicción de c_prueba	1.4637	8.468626	68.0
DWT-GRU — muestreo temporal			
Reconstrucción de c_prueba	1.3357	7.353046	60.0
Componente A5	1.3388	7.358327	60.0
Componente D5	0.0000	91.487306	46.4
Componente D4	0.0000	90.523456	57.6
Componente D3	0.0000	81.710255	46.4
Componente D2	0.0000	98.222451	44.0
Componente D1	0.0884	188.670092	53.6

6.3. Predicción Auto-predictiva con Corrección

Hasta el momento, hemos trabajado con los datos de ACTINVRB durante 2016 a 2020, siendo el faltante el análisis a los demás conjuntos. Por ello y para poner a prueba la capacidad de predicción de nuevas entradas de los modelos, emplearemos una esquema híbrido de predicción sobre los datos de las demás entidades financieras seleccionadas. Llamémosle *Predicción Auto-predictiva con Corrección* al alimentar a los modelos con los valores de las últimas ocho semanas de cada conjunto de entrenamiento, generando ocho predicciones auto-predictivas, esto es, de la semana nueve hasta la dieciséis. Para la predicción de la semana diecisiete se toma nuevamente ocho entradas correspondientes que pertenecen a los datos reales y así sucesivamente en intervalos de ocho semanas. De este modo, generaremos predicciones auto-predictivas y predicciones estándar alternadas. Así logrando un esquema 'justo' en el cual los modelos se enfrentan a predicciones lo suficientemente autónomas para su evaluación. Entonces, se obtuvo lo siguiente para cada conjunto de datos.

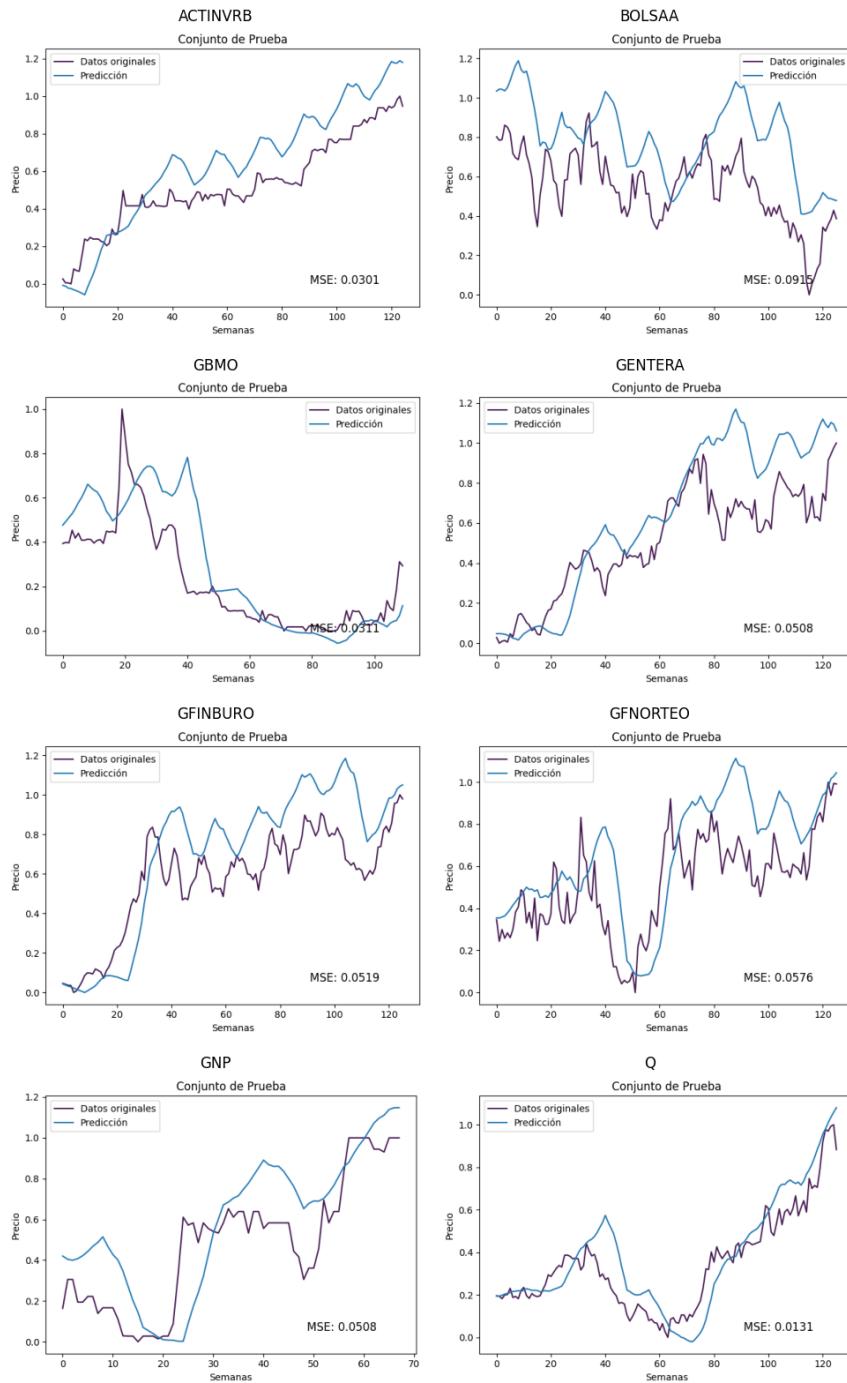


Figura 6.1: Desempeño del modelo NARNN

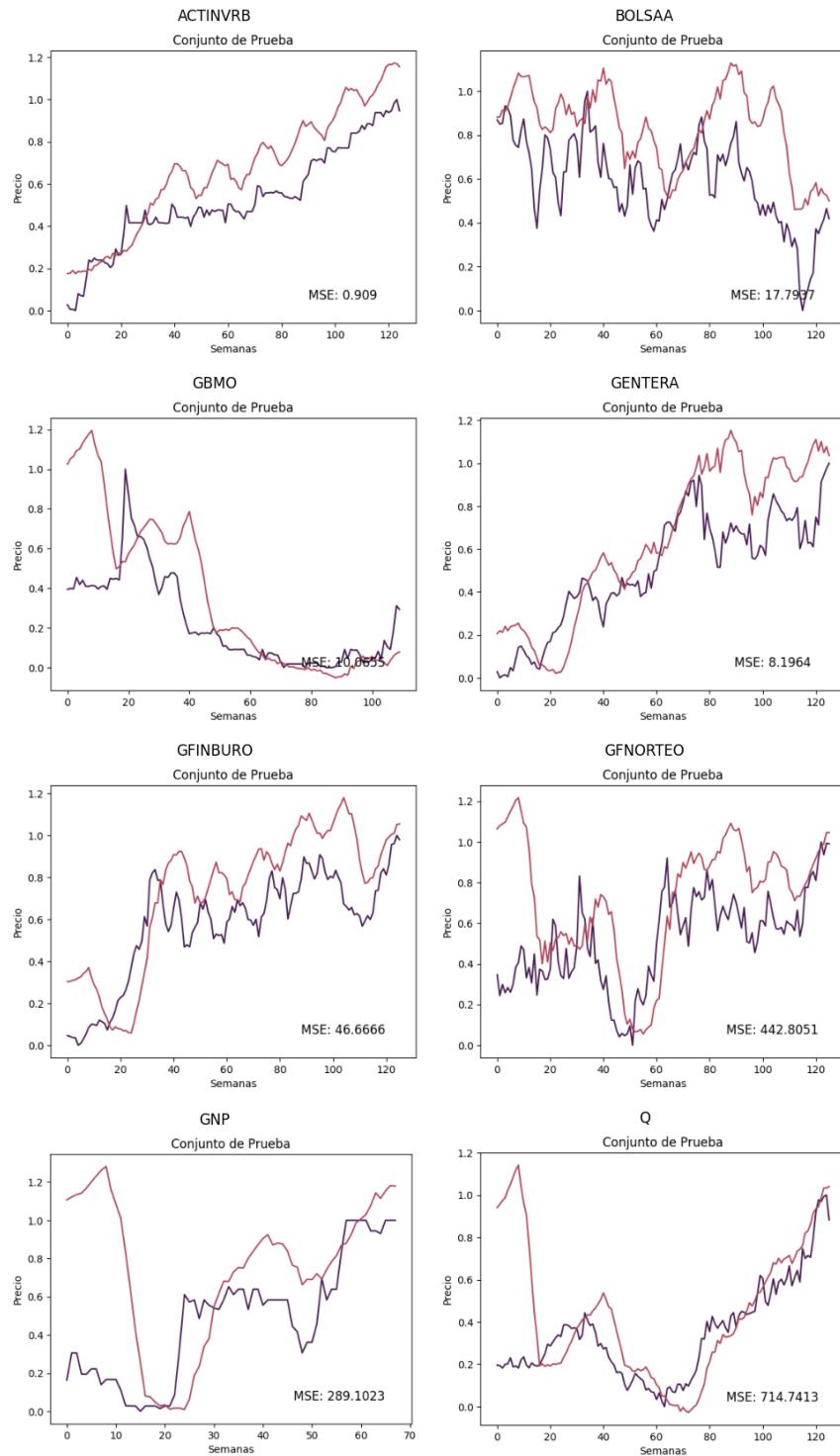


Figura 6.2: Desempeño del modelo DWT-NARNN

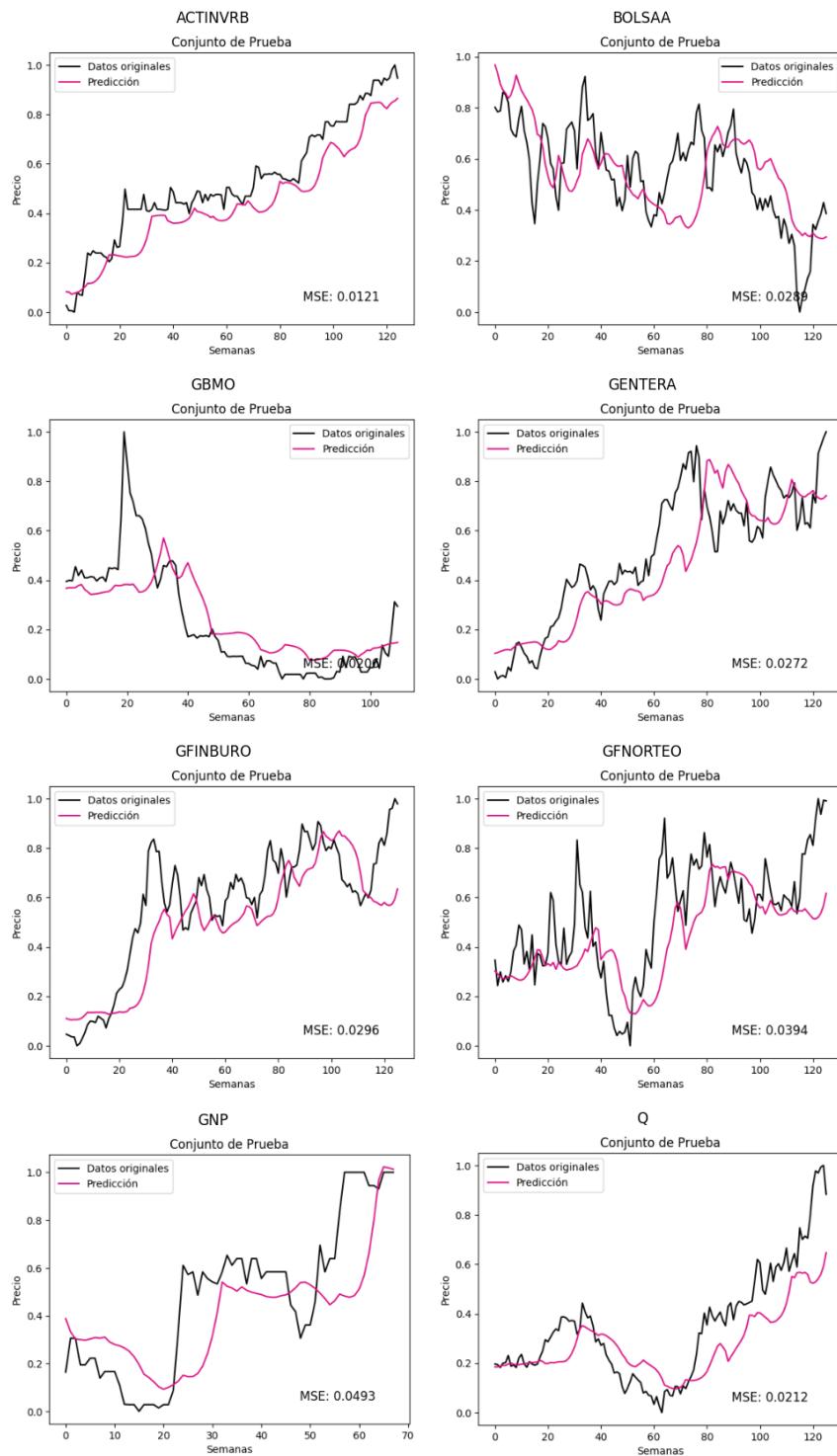


Figura 6.3: Desempeño del modelo LSTM

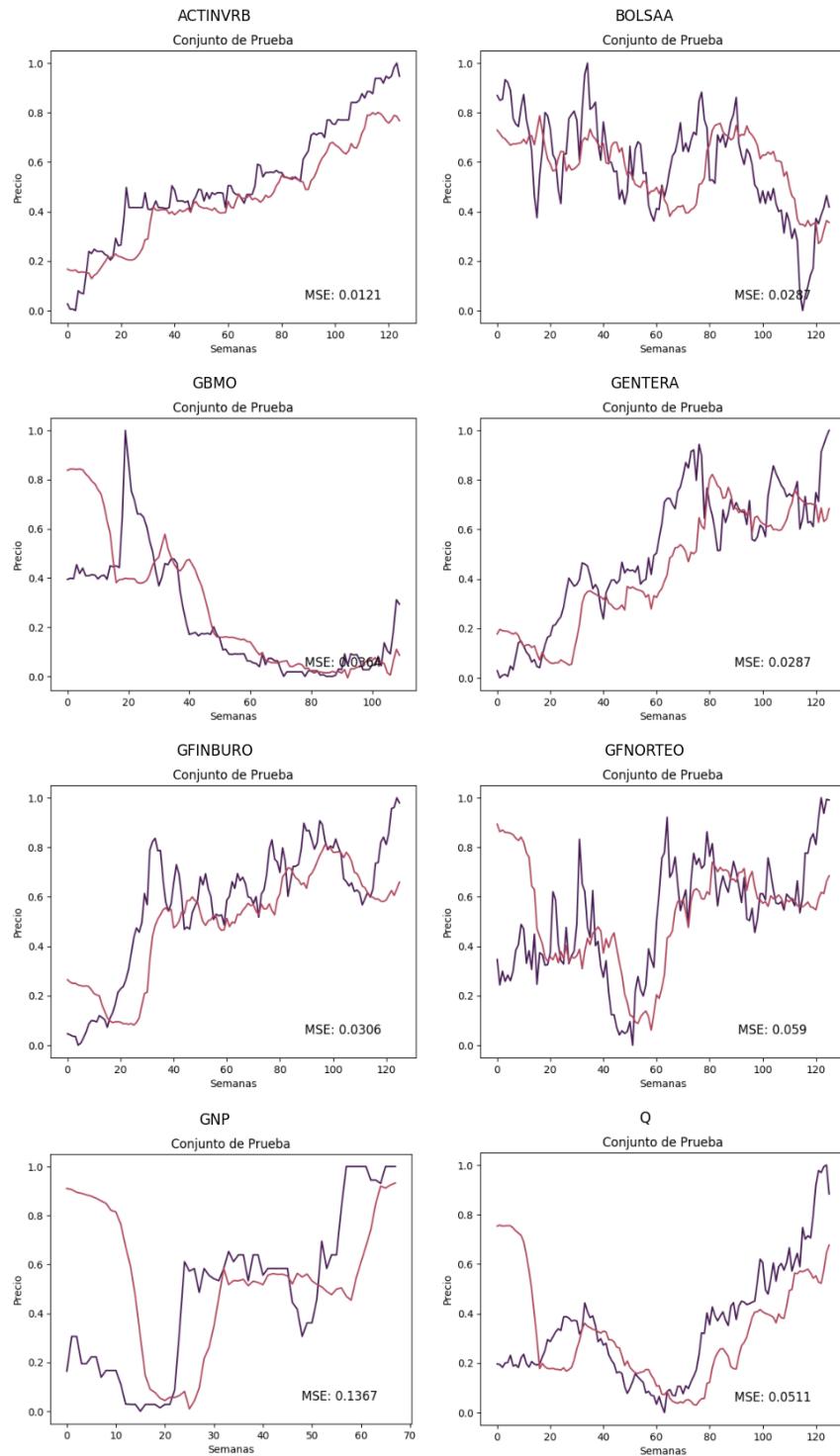


Figura 6.4: Desempeño del modelo DWT-LSTM

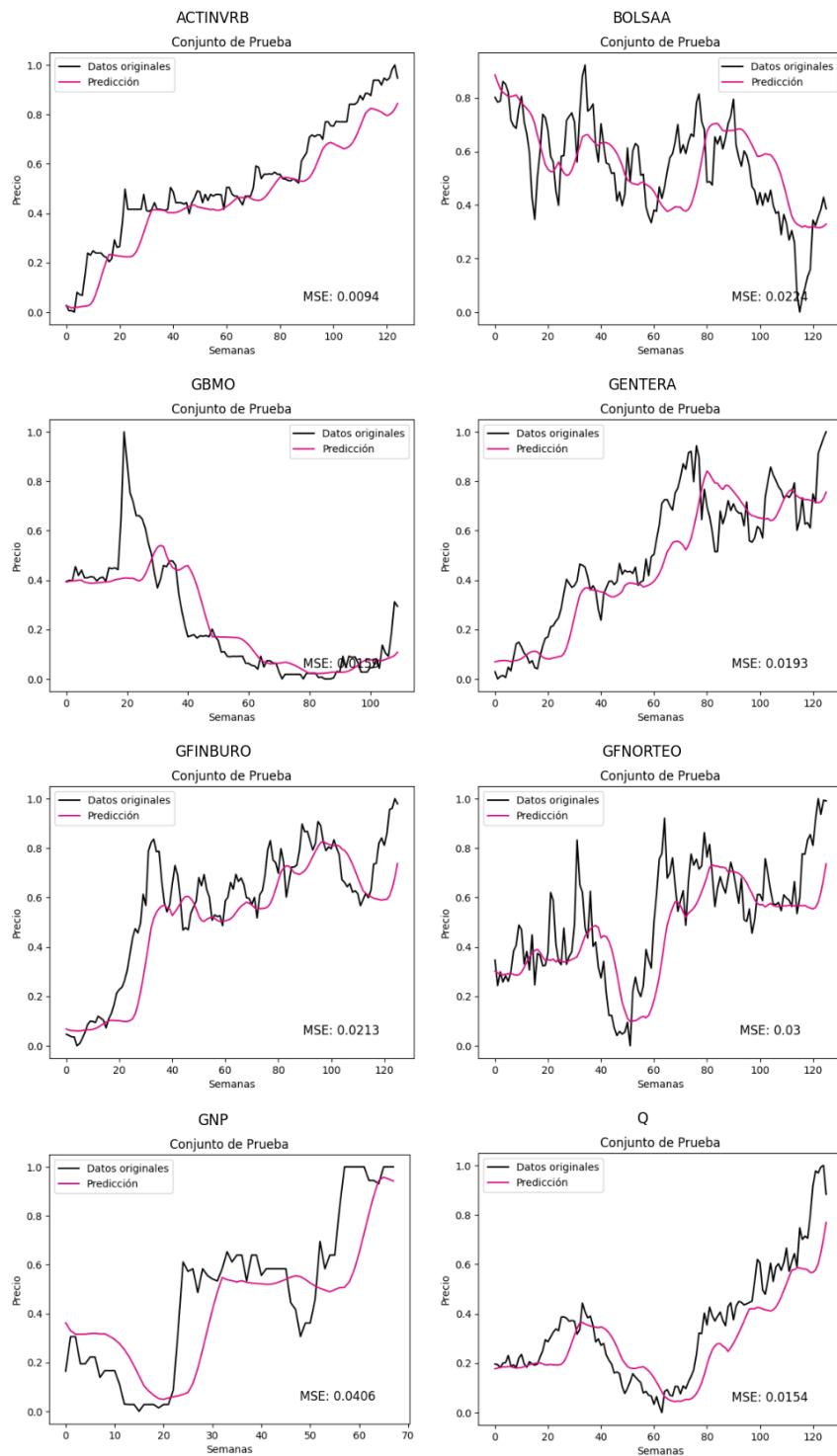


Figura 6.5: Desempeño del modelo GRU

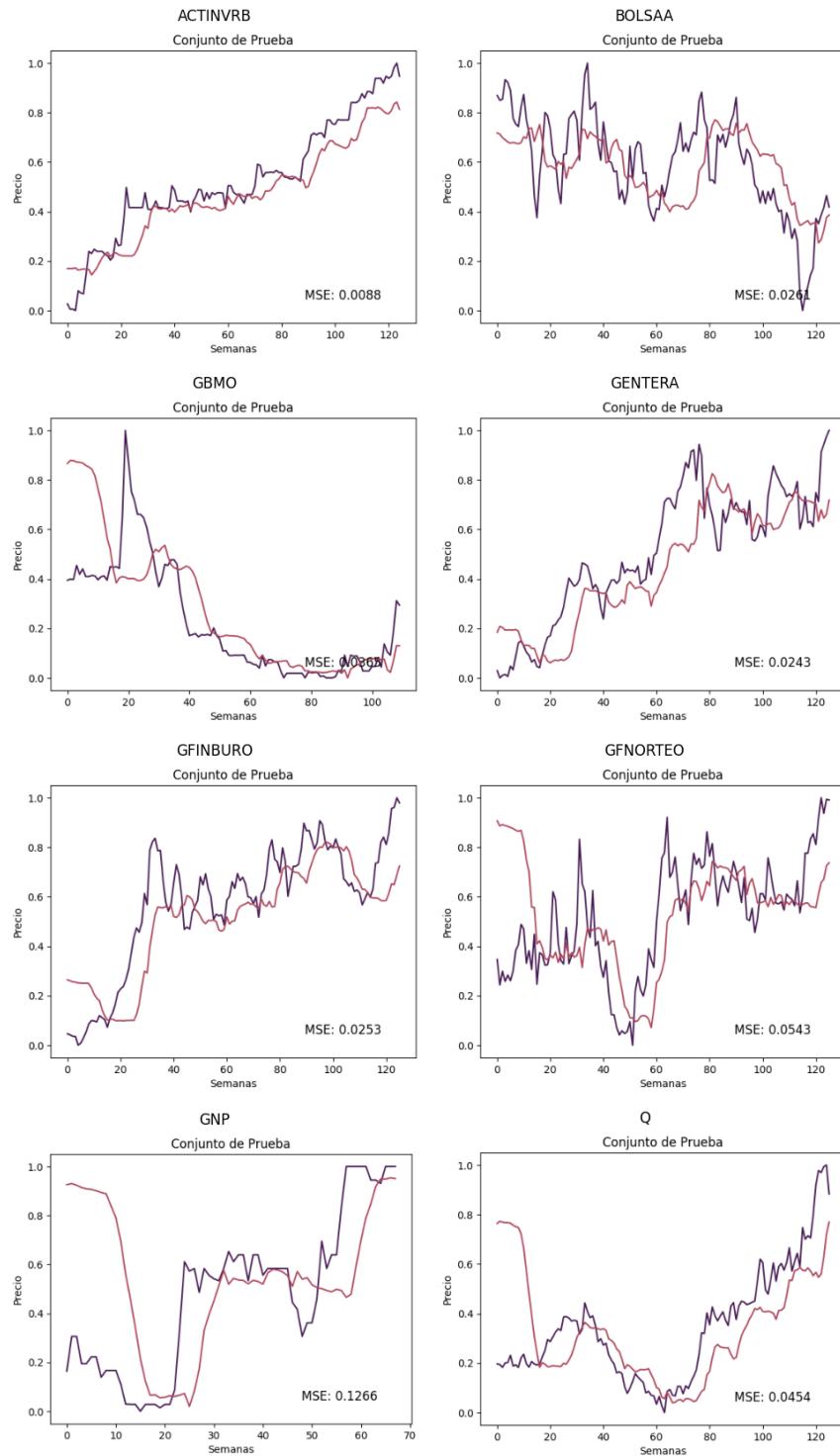


Figura 6.6: Desempeño del modelo DWT-GRU

Organizando los resultados de las métricas para las predicciones anteriores:

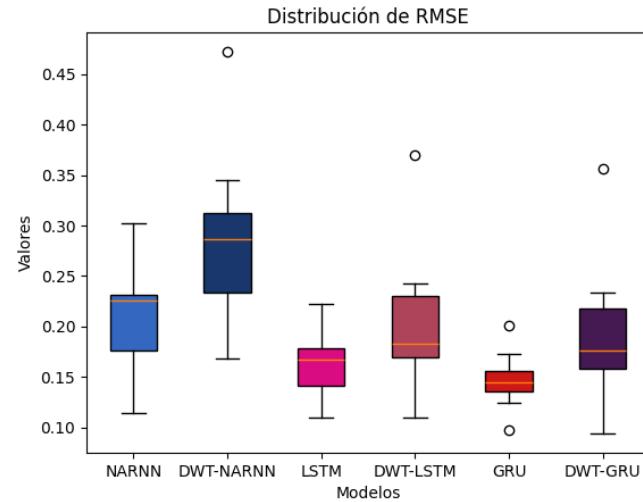


Figura 6.7: Diagrama de caja para métrica RMSE de las predicciones de todos los conjuntos de datos.

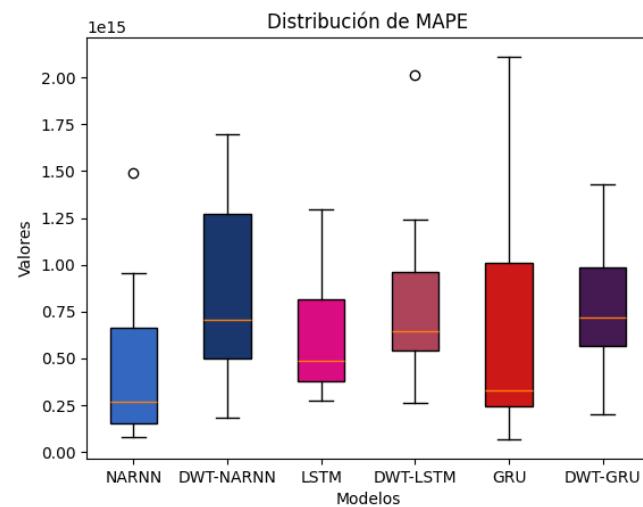


Figura 6.8: Diagrama de caja para métrica MAPE de las predicciones de todos los conjuntos de datos.

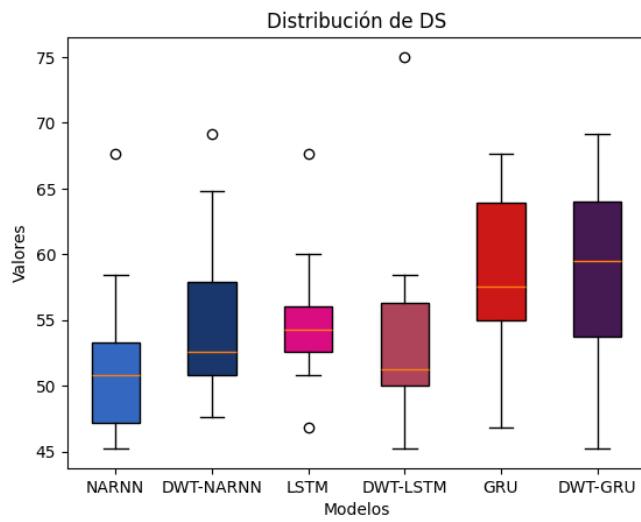


Figura 6.9: Diagrama de caja para métrica DS de las predicciones de todos los conjuntos de datos.

Capítulo 7

Conclusiones

Como vemos en el diagrama RMSE del capítulo anterior, las RNNs ya sea con DWT o no tienen un mejor desempeño de predicción que las NARNNs, un resultado que se esperaba ante el hecho de que las RNNs están capacitadas especialmente para resolver problemas de datos con dependencias a largo plazo como vimos en durante el estudio de redes neuronales.

Es importante destacar que a pesar de que la DWT sea una herramienta fundamental en el desempeño de una red neuronal, la arquitectura y métodos de entrenamiento de esta son igualmente importantes. Como vimos a lo largo del estudio, la combinación de un pre-procesamiento de datos con una red recurrente nos brinda mejores resultados:

- Podemos ver en el diagrama de cajón de RMSE que esta distribución es mayor cuando se usa la DWT (se equivoca más), pero es precisamente debido a la predicción de 'ruido' adicional en la serie, es decir, que al emplear la descomposición y la respectiva predicción en cada una de sus componentes y la posterior recomposición aumenta el ruido en esta y así consecuentemente el error o diferencia entre la serie original y esta. Hecho que no necesariamente significa que la predicción con DWT sea menor acertada, al contrario, la tendencia se ve mejor reflejada como podemos ver con el diagrama de DS.
- Los resultados de la predicción respecto a la dirección se adecuan más cuando existe la descomposición a diferencia de las redes que no cuentan

con ella, es decir que cuando se usa la DWT, las tendencias de pérdida o ganancia son mejor captadas. Véase que los modelos con mejor desempeño con respecto a esta métrica son el par de GRUnns.

A lo largo del entrenamiento, se pudo comprobar que como bien señala Prudhvira-ju Srivatsavaya [19], gracias a que los parámetros y complejidad en una LSTMnn son mayores que en una GRUnn, el tiempo de entrenamiento es lógicamente mayor, hecho que suma un punto a favor de esta arquitectura. Con este análisis, podemos decir que el mejor paradigma fue el recurrente, empleando la DWT sobre la GRUnn.

Sin embargo, no olvidemos que la predicción de precios de instrumentos financieros a lo largo del tiempo conlleva un reto importante y es vital no mantenerse atados a una sola tecnología cuando realizamos una investigación de este tipo. Existe una cantidad abismal de herramientas y técnicas que nos acercan cada día más a nuestros objetivos. Además de la creciente demanda de procesamiento de datos no dejemos de lado el análisis de otras variables sociales, políticas y culturales que impactan en estos estudios. Es por esta razón que este estudio no representa un cierre a la temática, si no que podemos encontrar en herramientas como la atención o modelos como los transformadores, buenos temas para continuar la investigación a futuro.

Apéndice A

Desarrollo del gradiente o_t y a_o

Para o_t :

$$\frac{E}{o_t} = \frac{\partial E}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t}, \text{ por regla de la cadena}$$

$$\implies \frac{E}{o_t} = \frac{\partial E}{\partial h_t} \cdot \frac{\partial}{\partial o_t}(o_t \cdot \tanh(c_t)), \text{ sustituyendo el valor de } h_t$$

$$\implies \frac{E}{o_t} = \frac{\partial E}{\partial h_t} \cdot \tanh(c_t)$$

Y para a_o :

$$\frac{E}{\partial a_o} = \frac{\partial E}{\partial o_t} \cdot \frac{\partial o_t}{\partial a_o}$$

$$\implies \frac{E}{\partial a_o} = \frac{\partial E}{\partial o_t} \cdot \frac{\partial}{\partial a_o}(\sigma(a_o))$$

$$\implies \frac{E}{\partial a_o} = \frac{\partial E}{\partial o_t} \cdot \sigma(a_o)(1 - \sigma(a_o))$$

$$\implies \frac{E}{\partial a_o} = \frac{\partial E}{\partial h_t} \cdot \tanh(c_t) \cdot o_t(1 - o_t)$$

Apéndice B

Implementación de algoritmos

B.1. DWT multinivel

```
1 import pywt
2
3 def multilevel_dwt(data, wavelet, levels, mode = 'constant'):
4     """
5         Calcula la transformada de ondícula (wavelet) de un conjunto
6         de datos o serie de tiempo y devuelve sus componentes
7         [IMPORTANTE] cAn, cDn hacen referencia a los coeficientes de
8         aproximación y detalle al nivel n respectivamente, que surgen
9         de la descomposición de la serie por
10        la transformada, mientras que An y Dn son los componentes de
11        aproximación y detalle al nivel n, se generan al aplicar una
12        reconstrucción parcial (upcoef) de
13        los coeficientes y son útiles para, al sumar cada uno de
14        estos componentes, obtener la señal original.
15        Args:
16            data: serie o señal a descomponer
17            wavelet: función con la cual se realizará la convolución,
18            y en consecuencia la descomposición
19            levels: nivel de la descomposición
20            mode:
21        """
22
23    n = len(data)
24    componentes = []
25    At = data
```

```

19     for l in range(1,levels+1):
20         # descompone la serie de tiempo en un siguiente nivel y
21         # obtiene los coeficientes de detalle y aproximacion
22         (cAt, cDt) = pywt.dwt(At, wavelet, mode)
23         # componentes de Aproximacion
24         At = pywt.upcoef('a', cAt, wavelet, take = n)
25         # componentes de Detalle
26         Dt = pywt.upcoef('d', cDt, wavelet, take = n)
27         componentes[:0] = [Dt]
28         if (l == levels):
29             componentes[:0] = [At]
    return componentes

```

B.2. Implementación NARNN

```

1 import torch.nn as nn
2 import torch.nn.functional as F
3
4 class NARNN(nn.Module):
5     """
6     Red Neuronal no Lineal Auto-regresiva
7
8     Estructura de la red:
9         Entrada: los n valores anteriores de un instante de la
10        serie.
11        Arquitectura: 3 capas densamente conectadas con 10
12        neuronas cada una. La primera con la funcion tangente-sigmode
13        y la segunda con logaritmo-sigmoide como funciones de
14        activacion. Luego la capa de salida,
15        con una funcion lineal como activacion comprende una sola
16        neurona.
17        Salida: un solo valor que representa la semana
18        consecuente a las n de entrada.
19        """
20
21     def __init__(self, t_entrada, t_salida, nombre = 'NARNN'):
22         super(NARNN, self).__init__()
23         self.nombre = nombre
24         self.fc1 = nn.Linear(t_entrada,10)
25         self.fc2 = nn.Linear(10,10)
26         self.fc3 = nn.Linear(10,t_salida)
27
28     def forward(self, x):
29         """

```

```

24     Paso de propagacion hacia adelante de la red
25 """
26     tan_sigmoid = lambda a : F.tanh(F.sigmoid(a))
27     x = tan_sigmoid(self.fc1(x))
28     x = F.logsigmoid(self.fc2(x))
29     x = self.fc3(x)
30     return x

```

B.3. Implementación LSTMnn

```

1 from keras import Model
2 from keras.layers import LSTM
3 from keras.layers import Dropout
4 from keras.layers import Dense
5
6 class red_LSTM(Model):
7 """
8     Red Neuronal con celdas de Memoria de Corto y Largo Plazo
9
10    Estructura de la red:
11        Entrada: los n valores anteriores de un instante de la
12        serie.
13        Arquitectura: 4 capas con 50 celdas LSTM e intercaladas 4
14        capas de desactivacion (dropout)
15        del 20% y finalmente una ultima capa densamente conectada
16        que comprende una sola neurona.
17        Salida: un solo valor que representa la semana
18        consecuente a las n de entrada.
19 """
20    def __init__(self, input_dim, output_dim):
21        super().__init__()#red_LSTM, self
22        self.LSTM1 = LSTM(units=50, return_sequences=True,
23                          input_shape=(input_dim, 1))
24        self.dropout1 = Dropout(0.2)
25        self.LSTM2 = LSTM(units=50, return_sequences=True)
26        self.dropout2 = Dropout(0.2)
27        self.LSTM3 = LSTM(units=50, return_sequences=True)
28        self.dropout3 = Dropout(0.2)
29        self.LSTM4 = LSTM(units=50)
30        self.dropout4 = Dropout(0.2)
31        self.dense = Dense(units=output_dim)
32
33    def call(self, inputs):

```

```

29         """
30     Define el comportamiento del modelo cuando se llama.
31     """
32     x = self.LSTM1(inputs)
33     x = self.dropout1(x)
34     x = self.LSTM2(x)
35     x = self.dropout2(x)
36     x = self.LSTM3(x)
37     x = self.dropout4(x)
38     x = self.LSTM4(x)
39     x = self.dropout4(x)
40     return self.dense(x)

```

B.4. Implementación GRUun

```

1 from keras import Model
2 from keras.layers import GRU
3 from keras.layers import Dropout
4 from keras.layers import Dense
5
6 class red_GRU(Model):
7     """
8         Red Neuronal con Unidades Recurrentes Cerradas
9
10        Estructura de la red:
11            Entrada: los n valores anteriores de un instante de la
12            serie.
13            Arquitectura: 4 capas con 50 celdas LSTM e intercaladas 4
14            capas de desactivaciOn (dropout)
15            del 20% y finalmente una Ultima capa densamente conectada
16            que comprende una sola neurona.
17            Salida: un solo valor que representa la semana
18            consecuente a las n de entrada.
19        """
20        def __init__(self,input_dim,output_dim):
21            super().__init__#red_GRU,self
22            self.GRU1 = GRU(units=50,return_sequences=True,
23            input_shape=(input_dim, 1))
24            self.dropout1 = Dropout(0.2)
25            self.GRU2 = GRU(units=50,return_sequences=True)
26            self.dropout2 = Dropout(0.2)
27            self.GRU3 = GRU(units=50,return_sequences=True)
28            self.dropout3 = Dropout(0.2)

```

```

24         self.GRU4 = GRU(units=50)
25         self.dropout4 = Dropout(0.2)
26         self.dense = Dense(units=output_dim)
27
28     def call(self, inputs):
29         """
30             Define el comportamiento del modelo cuando se llama.
31         """
32         x = self.GRU1(inputs)
33         x = self.dropout1(x)
34         x = self.GRU2(x)
35         x = self.dropout2(x)
36         x = self.GRU3(x)
37         x = self.dropout4(x)
38         x = self.GRU4(x)
39         x = self.dropout4(x)
40         return self.dense(x)

```

B.5. Algoritmo Levenberg-Marquardt

```

1     def step(self):
2         """
3             Paso de la optimizacion del algoritmo LM
4         """
5         x_n = self.aux_convierte_parametros()
6
7         #calculamos la matriz hessiana
8         h = torch.autograd.functional.hessian(self.
calcula_perdida, x_n)
9
10        #calculamos el gradiente de la funcion
11        grad_f = torch.autograd.grad(self.calcula_perdida(x_n),
x_n)[0]
12
13        # calculamos la transpuesta del gradiente
14        grad_f = torch.transpose(torch.unsqueeze(grad_f, 0), 0, 1)
15
16        # multiplica un escalar por la matriz identidad del
17        # tamanio de h y se lo sumamos a h
18        h_p = h+self.lambda*torch.eye(h.size(1))
19        #producto punto entre - la inversa de h_p y el gradiente
# de la red
20        x_n1 = torch.matmul(-torch.inverse(h_p), grad_f)

```

```
20      #se le da la forma adecuada para que se pueda sumar con
21      #el vector de nuevos pesos
22      x_n = x_n.reshape(h.size(1), 1)
23
24      #se realiza la actualizacion a los parametros
25      x_n = x_n + self.lr*x_n1
26      return self.asigna_parametros(torch.transpose(x_n,0,1)
27          [0],reasignar=False)
```

Apéndice C

Especificaciones de software

Tabla C.1: Paquetes y sus versiones

Paquete	Versión
keras	2.13.1
matplotlib	3.8.3
matplotlib-inline	0.1.6
numpy	1.24.3
pandas	2.2.1
PyWavelets	1.5.0
scikit-learn	1.4.1.post1
scipy	1.12.0
tensorboard	2.13.0
tensorboard-data-server	0.7.2
tensorflow	2.13.1
tensorflow-estimator	2.13.0
tensorflow-intel	2.13.1
tensorflow-io-gcs-filesystem	0.31.0
torch	2.2.1
torch-tb-profiler	0.4.3
torchaudio	2.2.1
torchvision	0.17.1

Tabla C.2: Especificaciones del dispositivo

Procesador	13th Gen Intel(R) Core(TM) i7-13700HX @ 2.10 GHz
RAM instalada	32.0 GB (31.6 GB utilizable)
Tipo de sistema	Sistema operativo de 64 bits, procesador x64

Bibliografía

- [1] Ingrid Daubechies. *1. The What, Why, and How of Wavelets*, pages 1–16. doi: 10.1137/1.9781611970104.ch1. URL <https://pubs.siam.org/doi/abs/10.1137/1.9781611970104.ch1>.
- [2] Amara Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2):50–61, 1995. doi: 10.1109/99.388960.
- [3] Christopher Colah. Understanding lstm networks. colah’s blog, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/?source=post_page-----b3996e6a0296-----.
- [4] Asmaa Y. Fathi, Ihab A. El-Khodary1, and Muhammad Saafan. A hybrid model combining discrete wavelet transform and nonlinear autoregressive neural network for stock price prediction: An application in the egyptian exchange. *Revue d’Intelligence Artificielle*, 37(1):15–21, 2 2023. doi: 10.18280/ria.370103.
- [5] M Ahmadizadeh. An introduction to short-time fourier transform (stft), 2014. URL <https://sharif.edu/~ahmadizadeh/courses/advstdyn/Short-Time%20Fourier%20Transform.pdf>. Sharif University of Technology.
- [6] Robi Polikar. The wavelet tutorial. part iv. multiresolution analysis: The discrete wavelet transform, 1999. URL <https://web.archive.org/web/20180430090402/http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html>.
- [7] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989. doi: 10.1109/34.192463.

- [8] Simeon Kostadinov. Understanding backpropagation algorithm. Medium, 2019. URL <https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd>.
- [9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.
- [11] Rahuljha. Lstm gradients: Detailed mathematical derivation of gradients for lstm cells. Medium, 2020. URL <https://towardsdatascience.com/lstm-gradients-b3996e6a0296>.
- [12] Arun Mallya. Lstm forward and backward pass. Medium. URL <https://arunmallya.github.io/writeups/nn/lstm/index.html#/>.
- [13] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Holger Schwenk Fethi Bougares, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Association for Computational Linguistics*, 2014.
- [14] Dimitri Fichou. Gru units. The Comprehensive R Archive Network, 2023. URL https://cran.r-project.org/web/packages/rnn/vignettes/GRU_units.html.
- [15] Mihir. Forward and backpropagation in grus — derived | deep learning. Medium, 2019. URL <https://medium.com/@mihirkhandekar/forward-and-backpropagation-in-grus-derived-deep-learning-5764f374f3f5>.
- [16] Roshan Adusumilli. Machine learning to predict stock prices. *Towards Data Science*, 2019. URL <https://towardsdatascience.com/predicting-stock-prices-using-a-keras-lstm-model-4225457f0233>.
- [17] Claudia Tejeda. Existen oportunidades en bolsa, dice actinver. El Economista, 9 2020. URL <https://www.eleconomista.com.mx/mercados/Existen-oportunidades-en-Bolsa-dice-Activer-20200924-0115.html>.
- [18] Gregory R. Lee, Ralf Gommers, Filip Waselewski, KaiWohlfahrt, and Aaron O’Leary. Pywavelets: A python package for wavelet analysis. *Journal of*

- Open Source Software*, 4(36):1237, 2019. doi: 10.21105/joss.01237. URL <https://doi.org/10.21105/joss.01237>.
- [19] Prudhviraju Srivatsavaya. Lstm vs gru. *Towards Data Science*, 2023. URL <https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a>.