UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR

INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Predicción de Tráfico de Red Usando Arquitecturas Neuronales Avanzadas

Autor:

Dña. Alicia Román Antolín

Tutor:

Dr. D. Miguel Luis Bote Lorenzo

Dr. D. Pablo Casaseca De La Higuera

Valladolid, 19 de septiembre de 2025

TÍTULO:	**Predicción de Tráfico de Red Usando Arquitecturas Neuronales Avanzadas**
AUTOR:	**Dña. Alicia Román Antolín**
TUTOR:	**Dr. D. Miguel Luis Bote Lorenzo** **Dr. D. Pablo Casaseca De La Higuera**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

PRESIDENTE:	**Dr. D. Pablo Casaseca De La Higuera**
VOCAL:	**Dr. D. Javier Aguiar Pérez**
SECRETARIO:	**Dr. D. Miguel Luis Bote Lorenzo**
FECHA:	**19 de septiembre de 2025**
CALIFICACIÓN:	

Resumen

La predicción del tráfico de red es una actividad de gran relevancia en el ámbito de las telecomunicaciones, ya que permite anticipar patrones de congestión, mejorar la planificación de infraestructuras y la distribución eficiente de los recursos disponibles, entre otras cosas. Dicha predicción puede ser abordada como un problema de series temporales. En este trabajo se estudian y comparan el rendimiento de la predicción de tráfico de red de distintas arquitecturas de redes neuronales que se consideran adecuadas para la predicción de series temporales: Long Short-Term Memory (LSTM) y su evolución reciente, Extended Long Short-Term Memory (xLSTM) junto con otras arquitecturas avanzadas.

Todos los modelos fueron entrenados adaptando su estructura a las características de los datos de entrada. Una vez completado el entrenamiento, se llevó a cabo un análisis comparativo centrado especialmente en el rendimiento, la precisión y complejidad computacional de cada modelo. En el caso de xLSTM y LSTM, se exploraron diferentes configuraciones con el objetivo de contrastar los resultados obtenidos. Además, para realizar un estudio más completo, se incluyen en el análisis otras arquitecturas avanzadas como Time-series Dense Encoder (TiDE), Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS), Transformadores de Fusión Temporal (TFT), Interpolación Jerárquica Neuronal para la Predicción de Series Temporales (N-HiTS) y Redes Convolucionales Temporales (TCN). Los resultados muestran un gran rendimiento por parte de TiDE y MLP, además son arquitecturas con poca carga computacional lo que es muy favorable para nuestro problema, la predicción de píxeles. Sin embargo, los resultados no ofrecen una primera aproximación exploratoria, dado que

no se han obtenido bajo condiciones homogéneas. Para obtener conclusiones más certeras sería necesario repetir los experimentos bajo un marco experimental uniforme.

Palabras clave

Tráfico de red, redes neuronales, series temporales, predicción, métricas de evaluación, carga computacional

Abstract

Network traffic prediction is a highly relevant task in the field of telecommunications, as it enables the anticipation of congestion patterns, improved infrastructure planning, and efficient allocation of available resources, among other benefits. This type of prediction can be approached as a time series problem. In this work, we study and compare the performance of various neural network architectures considered suitable for time series forecasting: Long Short-Term Memory (LSTM), its recent evolution Extended Long Short-Term Memory (xLSTM), and other advanced architectures.

All models were trained by adapting their structure to the characteristics of the input data. Once training was completed, a comparative analysis was conducted, focusing particularly on performance, accuracy, and computational complexity. In the case of xLSTM and LSTM, various configurations were explored in order to contrast the results obtained. Furthermore, to carry out a more comprehensive study, the analysis includes other advanced architectures such as Time-series Dense Encoder (TiDE), Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS), Temporal Fusion Transformers (TFT), Neural Hierarchical Interpolation for Time Series Forecasting (N-HiTS), and Temporal Convolutional Networks (TCN). The results show strong performance from TiDE and MLP, and both are architectures with low computational load, which is highly favorable for our problem—pixel prediction. However, the results do not provide an initial exploratory approximation, as they were not obtained under homogeneous conditions. To draw more reliable conclusions, it would be necessary to repeat the experiments within a consistent experimental framework.

Keywords

Network traffic, neural networks, time series, prediction, evaluation metrics, computational cost

Agradecimientos

Quiero dar las gracias a todas las personas con las que he compartido estos cuatros años de mi vida, por todas las experiencias que hemos vivido juntos y por convertirme en quien soy ahora.

En especial, quiero agradecer a mis padres, hermana, familia y amigos porque sin su apoyo y cariño nada de esto no hubiera sido posible.

También me gustaría agradecer a mis tutores Pablo y Miguel por sus explicaciones, consejos y correcciones.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

1. Introducción

La predicción de tráfico de red consiste en la estimación del flujo de datos futuros de una red de telecomunicaciones a partir de los datos de tráfico pasados y actuales [1]. Algunas de las utilidades de la predicción del tráfico de red es la de evitar la congestión de esta, la optimización de rutas de reenvío y la correcta distribución de los recursos de la infraestructura de telecomunicaciones para ajustarlos al volumen de tráfico generado en la misma.

Desde un punto de vista más técnico, la predicción de tráfico de red se puede plantear como un problema de series temporales dado que el volumen de datos trasmitidos puede representarse y medirse cronológicamente. En nuestro caso particular, la red se dividide en píxeles usando una cuadrícula, debido a que partimos de un mapa de localizaciones espaciales. Una vez hecha esta división, se realizan las mediciones del tráfico de la red, obteniendo de esta manera las series temporales. “Una serie temporal es una secuencia de puntos de datos recopilados, registrados o medidos en intervalos de tiempo sucesivos y uniformemente espaciados” [2]. La predicción de estas resulta fundamental en numerosos campos como el climático, financiero o comercial, donde existe la necesidad de anticipar tendencias y comportamientos.

Existen diferentes algoritmos de predicción de series temporales, los cuales podemos clasificar en 4 grandes grupos: modelos lineales, modelos no lineales, modelos híbridos y modelos descompuestos. Los modelos lineales asumen relacionen lineales entre los datos pasados y futuros, en cambio los modelos no lineales capturan patrones más complejos sin necesidad de ser estos lineales. Los modelos híbridos son el resultado de la combinación de dos o más modelos lineales y no lineales. Y por último los modelos descompuestos se caracterizan por el desglose de la serie temporal en cuatro componentes, de forma que cada componente es tratado de manera independiente por el modelo durante la predicción [3].

En la predicción de tráfico destacan los modelos no lineales sobre los lineales, debido a que son capaces de detectar relaciones complejas. Dentro de este grupo, se encuentran los modelos estadísticos clásicos, ARIMA y ARMA, referentes en tareas de predicción. Sin embargo, en la actualidad, se ha avanzado hacia técnicas más sofisticadas de predicción de dependencias no lineales como las Redes Neuronales Recurrentes (RNN, por sus siglas en inglés Recurrent Neural Network).

Las redes neuronales recurrentes han evidenciado ser realmente eficientes en la captura de dependencias a largo plazo en series temporales y en particular, la arquitectura

Long-Term Memory (LSTM) ha demostrado ser muy eficaz en la predicción de tráfico. Sin embargo, la arquitectura LSTM tradicional presenta una serie de limitaciones en aspectos como la paralelización durante el entrenamiento y la capacidad de almacenamiento de información contextual. Ante estas limitaciones, recientemente ha surgido una nueva variante llamada xLSTM, esta variante se compone de bloques mLSTM y sLSTM, que introducen mejoras como la memoria matricial o la compuerta exponencial que, otorgan mayor capacidad de almacenamiento y de revisión de la información almacenada [4]. Esta variante se ha presentado como una mejora de LSTM sin embargo, no ha probado su utilidad en el escenario de la predicción de tráfico.

En los últimos años, han ido apareciendo distintos modelos no lineales de predicción de series temporales entre los que sobresalen: Time-series Dense Encoder (TiDE), modelo basado en perceptrón multicapa (MLP) que salió a la luz en 2024 formado por capas densas [5]; y Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS), modelo introducido en 2020 capaz de captar fluctuaciones tanto a corto como a largo plazo [6]. No obstante, hasta el momento no se ha probado la utilidad de ninguno de estos modelos en el contexto de la predicción de tráfico.

Dentro del ámbito de la predicción de series temporales existen dos enfoques importantes de aprendizaje: el aprendizaje en línea y el aprendizaje por lotes. El aprendizaje por lotes consiste en entrenar el modelo a partir de un conjunto de entrenamiento previamente definido y disponible. Mientras que el aprendizaje en línea permite entrenar el modelo a medida que se van incorporando nuevas muestras. Modelos como el Perceptrón Multicapa (modelo no lineal), Regresión Lineal (modelo lineal) o Regresor Pasivo Agresivo (modelo no lineal) son modelos que pueden ser entrenados bajos ambos enfoques.

1.1. Objetivos

El principal objetivo de este trabajo es **analizar y comparar la capacidad predictiva de diferentes modelos de redes neuronales aplicándolos específicamente para predecir el tráfico de red** recogido en series temporales.

Dentro de este objetivo principal, se establecen los siguientes objetivos parciales:

- Comprender los fundamentos teóricos de los algoritmos utilizados.
- Estudiar las características de las series temporales empleadas, de forma que nos ayude en la interpretación de los resultados.
- Evaluar las mejoras introducidas por la arquitectura xLSTM respecto al modelo clásico.
- Analizar otras arquitecturas avanzadas para enriquecer la comparativa como Time-series Dense Encoder (TiDE), Linear Regression, Temporal Convolutional Network (TCN) o Multi-Layer Perceptron (MLP).
- Realizar un estudio integral que permita determinar qué arquitectura es la más adecuada identificando ventajas e inconvenientes.

1.2. Metodología

Este Trabajo de Fin de Grado sigue los principios del método de ingeniería propuestos en [7], pero en este caso con el objetivo de abordar el problema de la predicción de series temporales. Este método utilizado en el desarrollo del proyecto se organiza en distintas fases que se detallan a continuación.

En primer lugar, abordamos la **fase de idea,** normalmente esta fase comienza por la necesidad de resolver un problema, en nuestro caso el problema es la predicción de tráfico de red. En esta fase exploramos sobre diferentes conceptos como las redes neuronales y el aprendizaje profundo leyendo libros, artículos científicos, recursos formativos y cuadernos de Jupyter Notebook como [8], [9], [10], [11], [12] o [13]. Posteriormente, centramos la investigación en comprender la arquitectura LSTM, y en especial, su variante más actual, xLSTM. Esta variante fue diseñada para mejorar la retención de dependencias a largo plazo y la capacidad de generalización, es decir, la habilidad de utilizar lo aprendido con nuevos datos [4]. Para ello consultamos artículos científicos y ejemplos prácticos de proyectos como repositorios en GitHub[14].

En la **fase de concepto**, consiste en una lluvia de ideas en la que se propongan varias soluciones al problema planteado. De entre las posibles soluciones de este problema, se optó por emplear los modelos previamente probados en [15], como xLSTM, además de incorporar otras arquitecturas adicionales.

La **fase de planificación,** está consiste en la división en tares del trabajo y la estimación de la duración de dichas tareas. Algunas de estas tareas fueron: la modificación del código proporcionado en el repositorio[1] del artículo [15] o la realización de experimentos.

A continuación, en la **fase de diseño** se debía comprender el funcionamiento e implementación del código proporcionado por dicho repositorio, con el objetivo de identificar los cambios que se querían hacer sobre este proyecto para adaptarlo nuestros objetivos.

Durante la **fase de desarrollo** nos encargamos de realizar cambios específicos en el código de forma que se ajustase a nuestros datos, además de adaptarlo a las diferentes configuraciones de xLSTM que se incluyeron. Por otro lado, llevamos a cabo diferentes experimentos con la intención de obtener los resultados que usaremos para comparar los modelos de aprendizaje profundo aplicados en la predicción de series temporales. Los resultados de los experimentos corresponden al cálculo de las siguientes métricas: la Raíz del Error Cuadrático Medio (RMSE, por sus siglas en inglés Root Mean Squared Error), con la que evaluaremos la precisión de las predicciones de series temporales; y el Error Absoluto Medio (MAE, Mean Absolute Error), que cuantifica la magnitud del error promedio del modelo.

Finalmente, la sexta **fase de lanzamiento.** En esta fase se deben hacer entrega de la documentación y del producto final. Como en nuestro caso no existe un producto final,

[1] Repositorio disponible en: https://github.com/gonzalopezgil/xlstm-ts

esta fase solo incluye la entrega de este Trabajo de Fin de Grado donde se documentan los hallazgos encontrados.

1.3. Estructura de la memoria

La memoria se organiza en cinco capítulos, con la intención de guiar el lector para conseguir una mejor comprensión del documento. En el Capítulo 1 – Introducción se explica de una manera breve, el contexto general del trabajo. El Capítulo 2 – Estado del arte, alberga las primeras pinceladas sobre la predicción de series temporales y el uso de las redes neuronales en este ámbito. En particular nos centramos en la arquitectura LSTM, tanto en el modelo clásico como en su variante más reciente. En el Capítulo 3 – Entorno experimental, se describen los datos utilizados, las arquitecturas implementadas, las métricas de evaluación y los recursos computacionales empleados. El Capítulo 4 – Resultados y análisis, expone la recopilación de los resultados y su análisis crítico, presentando comparaciones de rendimiento entre las diferentes arquitecturas de redes neuronales avanzadas. Finalmente, el Capítulo 5 – Conclusiones y trabajo futuro, recoge las principales conclusiones extraídas del estudio realizado y se plantean posibles líneas de mejora para trabajos futuros.

Capítulo 2

2. Ascendentes y estado del arte

2.1. Introducción

El uso de redes neuronales en la predicción de tráfico de red tiene un gran interés en la actualidad. La motivación principal para seguir avanzando en el desarrollo de esta herramienta radica en su capacidad de anticipar eventos, lo que nos permite ajustar dinámicamente los parámetros de la red, como, por ejemplo, el ancho de banda o distribuir de manera adecuada los recursos estructurales de la red. En nuestro caso, el objetivo es conseguir anticiparnos de la manera más acertada posible a la cantidad de tráfico transmitidos por un sistema de telecomunicaciones.

A lo largo de este capítulo se hablará sobre series temporales y diferentes técnicas utilizadas para su análisis y predicción, junto con la aplicación de redes neuronales avanzadas en la predicción de series temporales. Asimismo, se hará una breve introducción al aprendizaje automático y se describirán de manera teórica las arquitecturas de aprendizaje profundo utilizadas en este trabajo. Entre ellas, se encuentran las redes LSTM, TiDE y N-BEATS. Y, por último, cerraremos este capítulo con la exposición de las diferentes conclusiones a las que hemos llegado.

2.2. Predicción de series temporales

Una serie temporal se puede definir como un conjunto de datos cuyos valores han sido medidos de manera secuencial y equiespaciada en el tiempo. La forma de representar una variable temporal es colocando el tiempo en el eje de abscisas y la métrica de estudio en el eje de ordenadas. [2]

Clásicamente, las series temporales se analizan descomponiéndolas en tres componentes básicos: tendencia, efecto estacional y componente aleatoria (también denominada ruido). Esta descomposición se expresa mediante la siguiente ecuación:

$$X_t = T_t + E_t + I_t \qquad (1)$$

donde X_t representa el valor de la serie, T_t la tendencia, E_t el efecto estacional y I_t el ruido. [16]

Esta técnica de expresar la descomposición de componentes como una suma se denomina descomposición aditiva. Otra forma habitual de expresar esta descomposición es a través de una multiplicación. Siendo, por tanto, la expresión de la descomposición

multiplicativa la siguiente:

$$X_t = T_t * E_t * I_t \qquad (2)$$

La tendencia refleja la dirección hacia la que evolucionan los datos a largo plazo. Las tendencias se pueden clasificar en lineales y no lineales. El efecto estacional consiste en patrones periódicos que se repiten en intervalos regulares, es decir, esta componente representa los eventos repetitivos. La componente aleatoria, por otra parte, representa las fluctuaciones correspondientes a eventos fortuitos. [17][18]

Esta división permite aislar los distintos comportamientos de la serie, evitando de esta manera confusiones entre, por ejemplo, fluctuaciones aleatorias y patrones repetitivos.

Sin embargo, algunos enfoques incorporan a esta descomposición un componente adicional, las variaciones cíclicas. Las variaciones cíclicas son fluctuaciones a largo plazo que no siguen un periodo fijo.

Otra forma de analizar las series temporales aplicando el análisis espectral. Esta técnica de análisis de series temporales se basa en la idea de que una serie temporal se puede representar como una combinación lineal de funciones trigonométricas:

$$x_t = \sum_{k=0}^{n} a_k * \cos w_k t + b_k * \sin w_k t \qquad (3)$$

donde a_k y b_k son amplitudes y $w_k = 2\pi k/T$ es la frecuencia. Por tanto, para poder determinar que funciones trigonométricas componen la serie necesitamos transformarla al dominio de la frecuencia para lo que aplicamos la Trasformada de Fourier Discreta (TFD). Esta técnica permite detectar patrones periódicos o cíclicos. [19]

También se puede aplicar el método de autocorrelación para el análisis de series temporales. Este método mide el parecido entre los diferentes puntos de una serie. Para aplicar este método debemos calcular la matriz de autocorrelación, esta matriz está formada por los diferentes coeficientes de autocorrelación. El coeficiente de autocorrelación en el retardo k se calcula con la siguiente fórmula:

$$\rho_k = \frac{Cov[y_t, y_{t+k}]}{\sqrt{Var[y_t]} * \sqrt{Var[y_{t+k}]}} \qquad (4)$$

Una vez calculada la matriz de autocorrelación, se procede al análisis, un mayor nivel de autocorrelación indica la presencia de ciclos o tendencias. Este método presenta una limitación, el coeficiente de correlación mide una relación lineal entre las muestras, de forma que no se puede usar para capturar relaciones complejas.[20], [21]

Por otro lado, como se mencionó anteriormente, existen diversos algoritmos de predicción de series temporales. Estos algoritmos se pueden clasificar en cuatro grandes grupos: modelos lineales, modelos no lineales, modelos híbridos y modelos descompuestos.

Los modelos lineales suponen una relación de linealidad entre los datos pasados y futuros, es decir, asumen una estructura de covarianza en la serie temporal. Dentro de esta categoría los modelos autorregresivos (AR) utilizan en sus predicciones combinaciones lineales de valores pasados, de esta forma, el valor actual de la serie

temporal es una función de *p* valores previos más un error aleatorio. El orden *p* de la combinación lineal se determina aplicando el análisis de autocorrelación, en particular, se determina utilizando la función de autocorrelación parcial. Por otro lado, los modelos de media móvil (MA) en lugar de utilizar valores pasados para realizar las predicciones, usan combinaciones lineales de los errores pasados, lo que es útil para capturar irregularidades o fluctuaciones aleatorias. En este caso, para calcular el orden *q* de la combinación lineal el modelo MA emplea la función de autocorrelación (ecuación 4). [2]

Los modelos autorregresivos de media móvil (ARMA) combinan las estrategias AR y MA para modelar las dependencias usando tanto de valores pasados como los errores pasados, siendo este modelo adecuado para series estacionarias. En el caso de que la serie no sea estacionaria, los modelos más adecuados son los modelos autorregresivos integrados de media móvil (ARIMA), que incluyen un término de diferenciación para manejar tendencias y patrones más complejos. Para capturar patrones estacionales periódicos, se utilizan modelos estacionales ARIMA (SARIMA), que incorporan componentes estacionales adicionales. Tanto ARIMA como SARIMA y ARMA emplean el análisis de autocorrelación, en concreto la función de autocorrelación y la función de autocorrelación parcial, para calcular los órdenes de los componentes autorregresivos *p* y de media móvil *q*.

Por otra parte, los modelos no lineales se utilizan cuando las relaciones temporales no se pueden interpretar como una combinación lineal dado su complejidad. Entre estas técnicas se encuentran las redes neuronales y otros enfoques como el GARCH. El GARCH es un modelo estadístico de análisis de series temporales capaz de capturar patrones no lineales y relaciones dinámicas complejas.

Los modelos híbridos integran estrategias tanto lineales como no lineales aprovechando las ventajas de ambos modelos para mejorar la fiabilidad y exactitud de la predicción. Un ejemplo de este tipo de modelos es la combinación del modelo ARIMA, encargado de modelar las dependencias lineales, con GARCH, encargado de capturar la variabilidad en la varianza.

Finalmente, los modelos descompuestos facilitan la identificación y modelado individual de cada componente de la serie temporal, lo que mejora la precisión en el pronóstico y la interpretación de los datos. Estos modelos utilizan técnicas como la descomposición aditiva y multiplicativa mencionadas anteriormente.

2.3. Introducción al aprendizaje automático

El aprendizaje automático es un subconjunto de la inteligencia artificial que consiste en desarrollar programas, denominados modelos, que sin necesidad de ser programados explícitamente para una tares sean capaces de resolverla a partir de los datos. Dentro del aprendizaje automático encontramos dos grandes grupos: el aprendizaje supervisado y no supervisado. El aprendizaje supervisado es aquel que se entrena con datos etiquetados, es decir, utilizando la relación entre la entrada y la salida con el objetivo de predecir la etiqueta correcta. Sin embargo, un modelo no supervisado se entrena usando datos sin etiquetar, de forma que no sabe cuál es la respuesta correcta de antemano y por tanto su objetivo es encontrar relaciones dentro de los datos. El aprendizaje

supervisado se divide en dos categorías principales: la clasificación y la regresión. Los modelos de regresión son los que realizan predicciones de valores continuos mientras que los de clasificación son aquellos que asignan categorías discretas. En el caso de los modelos de regresión, el objetivo es estimar un valor numérico a partir de un conjunto de datos de entrada. Los modelos de regresión son fundamentales para el análisis predictivo, especialmente en el análisis de series temporales. [22]

Dentro del aprendizaje automático, se encuentra el aprendizaje profundo, que utiliza redes neuronales con múltiples capas para extraer y aprender patrones complejos [13]. Para poder entender qué son las redes neuronales avanzadas necesitamos comprender tres conceptos clave: nodo, funciones de activación y capas.

- Un nodo es la unidad básica de procesamiento de la red neuronal, que a través de la función de activación imita el comportamiento de una neurona.
- La función de activación es una función matemática que sirve para decidir si el nodo genera salida o no. La función de activación introduce no linealidad a la red.
- Las capas son conjuntos de nodos procesados en paralelo. Hay tres tipos de capas: de entrada, oculta y de salida. La capa de entrada es la encargada de recibir la información que llega desde fuera de la red neuronal. Su función es procesar esos datos y prepararlos para enviarlos a la siguiente capa: la capa oculta. La capa oculta puede recibir información tanto de la capa de entrada como de otras capas oculta anteriores. Su misión es analizar, combinar y transformar la información para mejorar la información que se envía a la siguiente capa. Por último, la capa de salida proporciona el resultado final obtenido por la red tras el procesamiento de los datos. Cada capa puede tener uno o varios nodos.

Las redes neuronales al entrenar actualizan sus parámetros utilizando un gradiente. Cuando se entrenan redes muy profundas puede surgir un problema conocido como el desvanecimiento del gradiente. Este problema consiste en que el valor de dicho gradiente se va haciendo cada vez más pequeño de forma que el modelo deja de aprender de manera efectiva.

Para el entrenamiento de las redes neuronales es necesario dividir el conjunto de datos en tres conjuntos:

- Conjunto de entrenamiento: compuestos por los datos que el modelo utiliza para aprender y ajustar parámetros.
- Conjunto de validación: este conjunto se utiliza para refinar la los hiperparámetros.
- Conjunto de test: este conjunto de datos se utiliza para evaluar el rendimiento del modelo.

2.4. Redes neuronales aplicadas a series temporales

Para entender cómo podemos utilizar las redes neuronales en la predicción de series temporales vamos a recurrir a explicar el funcionamiento de las redes neuronales

recurrentes puesto que este tipo de redes son expertas en el tratamiento de series temporales.

Las Redes Neuronales Recurrentes (RNN) son "un tipo de red neuronal artificial diseñada específicamente para procesar secuencias de datos, donde la salida de un paso de tiempo se utiliza como entrada para el siguiente" [23]. Son capaces de manejar hábilmente tanto información pasada como actual, comprender esta información e ir actualizando continuamente su conocimiento a medida que se incorporan nuevos datos.

Las redes RNN modelan la salida $\hat{y}_t$ como una función de los valores pasados:

$$\hat{y}_{t+1} = f(y_t, y_{t-1}, \dots, y_1; \theta) \qquad (5)$$

donde $(y_t, y_{t-1}, \dots, y_1)$ es la secuencia de valores pasados, θ son los parámetros aprendidos por la red y $f(\cdot)$ es una función que modela la evolución de la serie temporal. Esta formulación puede interpretarse como una generalización de las técnicas clásicas de análisis de series temporales. Con el uso del estado oculto h_t las redes RNN incorporan la idea de autocorrelación, debido a que este estado oculto se calcula a partir de información de estado anterior. Por otro lado, no implementa directamente una descomposición aditiva o multiplicativa o un análisis espectral, pero sí es capaz de aprender de manera implícita tendencias, efectos estacionales, periodicidad y ruido.

La arquitectura de una RNN está compuesta por nodos organizados en capas, las cuales pueden ser de tres tipos: de entrada, oculta o de salida. La red puede tener varias capas ocultas, pero solo puede tener una capa de salida y otra de entrada. En las redes neuronales convencionales, las capas suelen conectarse de manera secuencial. Sin embargo, en el caso de una RNN existen conexiones adicionales llamadas recurrentes, que permiten la retroalimentación de información dentro de la red, es decir, la salida de un nodo en un instante de tiempo se convierte en la entrada del mismo nodo o de otro nodo de la capa en el siguiente instante.

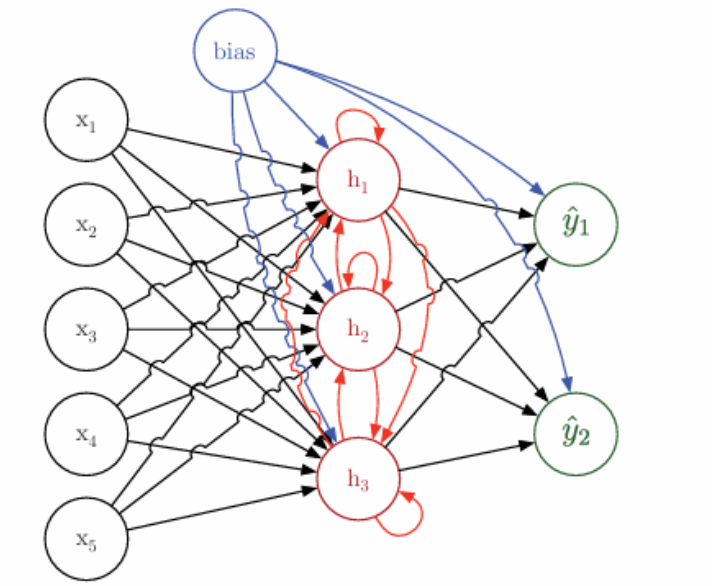

Figura 1: Conexiones de una red neuronal recurrente. Figura tomada de [24]

Las RNN están diseñadas para trabajar con datos que llegan en orden, como ocurre en las series temporales. A diferencia de otras redes que procesan todo de golpe, las RNN procesan la información paso a paso. En cada paso, la red realiza tres tareas: entrada, procesamiento y salida.

La entrada está formada por dos elementos, la entrada actual, compuesta por el dato que llega en ese instante; y el estado interno anterior que contiene la información aprendida por la red hasta ese instante de tiempo. El estado interno anterior llega como entrada a través de una conexión recurrente, que se encuentra en los nodos de las capas ocultas.

El procesamiento consiste en la combinación de la entrada actual con el estado anterior para obtener el nuevo estado interno, para ello utilizan funciones de activación. Estas funciones de activación pueden ser: la función sigmoide, que normaliza los valores entre 0 y 1; la función tangente hiperbólica, que centra los valores entre -1 y 1; la función ReLU, que deja pasar solo los valores positivos; etc.

Una vez calculado el nuevo estado interno, se genera la salida, que puede ser una predicción del valor siguiente, una decisión, una clasificación, etc. Además, ese nuevo estado se envía al siguiente paso de tiempo, permitiendo que la red recuerde lo que ha aprendido.

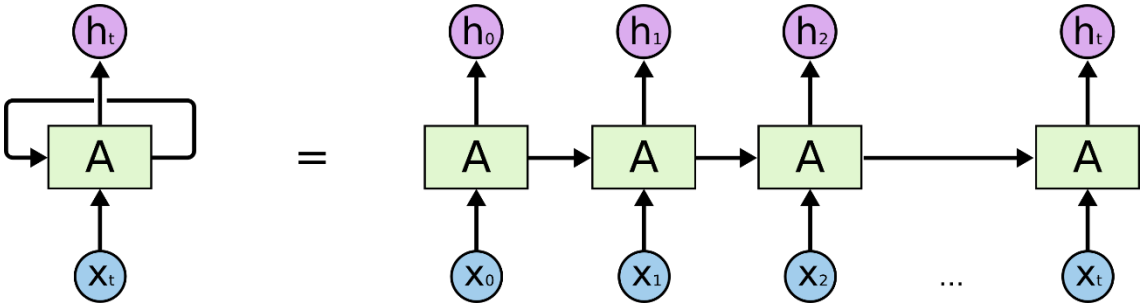

Figura 2: Esquema de una celda recurrente (LSTM) y su representación desplegada en el tiempo. Figura tomada de [25]

Las redes neuronales recurrentes tradicionales son un tipo de aprendizaje profundo que ha sido ampliamente utilizadas para entender secuencias. Sin embargo, estas redes no eran capaces de retener información a largo plazo, provocando que la información aprendida en instantes lejanos se olvidara. Este problema se conoce como gradiente desvaneciente. La aparición de LSTM revolucionó las redes neuronales recurrentes, gracias a su novedosa propuesta que solucionaría la falta de manejo de datos secuenciales, la introducción de puertas que controlan el flujo de información.

2.5. LSTM y sus limitaciones

Las redes LSTM (Long Short-Term Memory, Memoria a Largo y Corto Plazo) son redes neuronales recurrentes, lo que significa que utilizan lo aprendido en instantes pasados para realizar las predicciones.

Las LSTM fueron diseñadas a finales de los años noventa por los investigadores alemanes, Sepp Hochreiter y Jürgen Schmidhuber. Como se mencionó en el apartado

anterior, estas redes surgieron para solucionar el problema del decaimiento del gradiente, un problema común en las redes neuronales recurrentes tradicionales. Para ello, se introdujeron tres puertas, también llamadas compuertas, que permiten elegir de forma inteligente qué información almacenar y cuál olvidar.

La clave del LSTM es el mecanismo de celda de memoria que simula la capacidad del ser humano de catalogar información como importante o irrelevante, permitiendo decidir qué información recordar y cuál descartar. Por cada entrada, existe una celda de estado y el mecanismo de la celda de memoria es la que controla y gestiona estas celdas de estado.

Las celdas de estado constituyen el eje central de la arquitectura LSTM. La información de la serie temporal recorre la celda de estado, la cual modifica esta información usando las compuertas. Como se observa en la Figura 4, esta celda contiene tres puertas: de olvido, de entrada y de salida.

La puerta de olvido determina qué información del estado anterior se debe descartar utilizando como función de activación la función sigmoide que normaliza los valores entre 0 y 1. Si la salida de la puerta de olvido es 0 o cercana a 0, esto significa que dicha información debe ser olvidar por completo y si la salida es 1 o próxima a 1 significa que se debe recordar dicha la información [25]. La puerta de entrada elige nueva información importante, la cual será almacenada en la celda de memoria. Realiza dos pasos, primero, se decide que valores se van a actualizar a través de una capa sigmoide, mientras que paralelamente, se crea una lista de posibles valores para almacenar utilizando una capa tanh, y posteriormente, se combinan las salidas de ambas capas obteniendo la información a almacenar. La puerta de salida filtra y selecciona la información que debe enviarse a la siguiente celda de estado, dispone de dos operaciones, la primera es una función tanh que transforma el estado de la celda y la segunda es una función sigmoide que se encarga de filtrar la información que se convertirá en salida de la celda.[26][27]

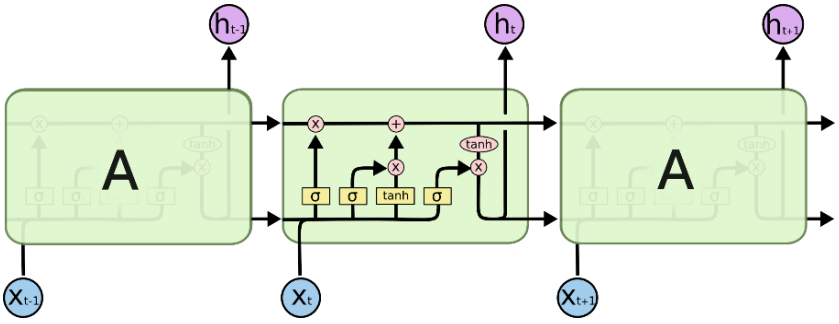

Figura 3: Representación esquemática de una capa recurrente LSTM. Figura tomada de [25]

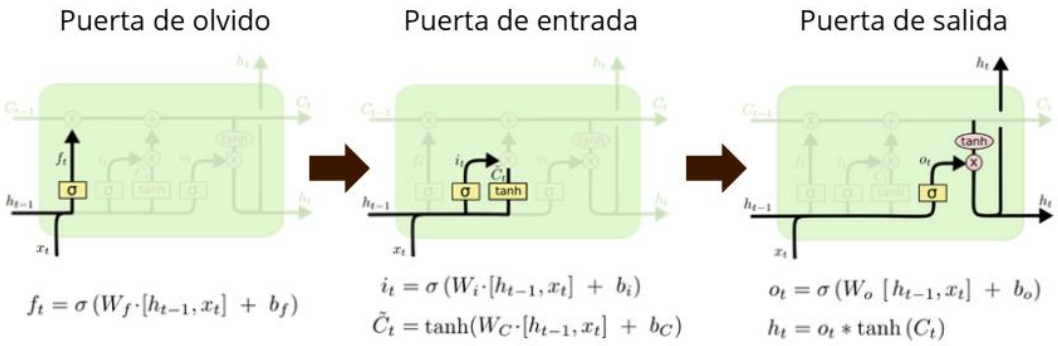

Figura 4: Diagrama detallado de las tres puertas fundamentales en una celda LSTM. [25]

Las redes LSTM han supuesto una mejora respecto a las redes neuronales recurrentes tradicionales, resolviendo problemas el gradiente desvaneciente, mejorando capacidades de memoria y aumentando el rendimiento en múltiples aplicaciones. Estas redes han demostrado ser una herramienta muy eficaz en la predicción de series temporales según diversos estudios. Sin embargo, presentan algunas limitaciones importantes. Por ejemplo, no permiten revisar las decisiones de almacenamiento una vez tomadas. Además, tienen una capacidad de almacenamiento limitada y su diseño dificulta la paralelización debido a la mezcla de memoria.

En resumen, las redes LSTM supusieron un avance fundamental respecto a las redes recurrentes tradicionales, al mitigar el problema del decaimiento del gradiente y mejorar la capacidad de retención de información en la memoria a largo plazo. A pesar de ello, sus limitaciones de capacidad de almacenamiento y la falta de paralelización han impulsado el desarrollo de nuevas variantes y arquitecturas que buscan superar estas barreras.

2.6. Evolución de LSTM

Aunque el modelo LSTM ha sido ampliamente utilizado para capturar dependencias a largo plazo, sus limitaciones impulsaron el desarrollo de una nueva versión de esta arquitectura que salió a la luz en mayo del 2024. Esta nueva variante denominada xLSTM fue introducida por un grupo de investigadores entre los que se encuentra Sepp Hochreiter, quien junto a Jürgen Schmidhuber publicaron en 1997 la versión original de LSTM.

xLSTM propone una serie de mejoras interesantes para solventar ciertas limitaciones de la versión original. En primer lugar, para resolver el problema de la imposibilidad de revisar las decisiones de almacenamiento introduce una compuerta exponencial. Esta compuerta permite a la red examinar los valores almacenados y en el caso de que se encuentren valores más relevantes permite modificar estos valores. De esta forma, se evita almacenar información innecesaria. En segundo lugar, para superar la limitación de almacenamiento, xLSTM incorpora una memoria matricial que mejora la capacidad de retención de la información. En tercer y último lugar, soluciona la falta de paralelización a través de la introducción de una variante de LSTM llamada mLSTM.

Esta variante elimina la mezcla de memoria y permite la paralelización de las operaciones dentro del bloque mLSTM. En lugar de depender de conexiones ocultas-ocultas, mLSTM utiliza una memoria matricial con una regla de covarianza para conseguir que los cálculos se realicen en paralelo sin perder información relevante.

En la Figura 5 podemos ver de forma visual los soluciones que propone xLSTM las limitaciones de LSTM.

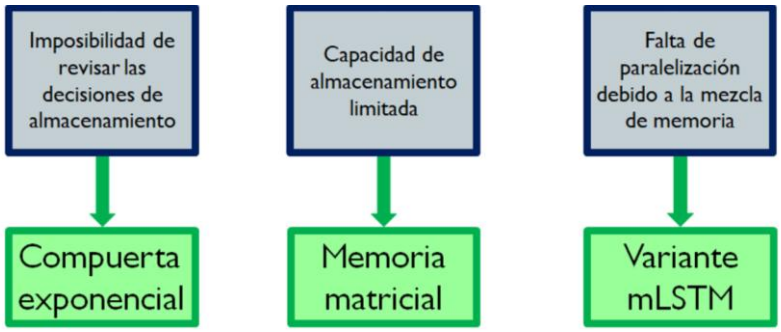

Figura 5: Evolución de LSTM a xLSTM

2.6.1. Arquitectura xLSTM

La arquitectura xLSTM está compuesta por dos variantes del modelo LSTM denominadas sLSTM y mLSTM, que se combinan de forma alterna y secuencial en bloques residuales. Un bloque residual es aquel que dispone de una conexión que permite sumar la entrada del bloque con la salida.

Por un lado, el bloque sLSTM incorpora una memoria escalar, una actualización escalar y una técnica novedosa de mezcla de memoria. Sin embargo, la mezcla de memoria en sLSTM no admite procesamiento en paralelo debido a las conexiones recurrentes entre estados ocultos, lo que limita su rendimiento computacional al requerir un procesamiento secuencial.

Por otro lado, el bloque mLSTM incluye una memoria matricial junto con una regla de actualización basada en la covarianza, lo que permite eliminar la mezcla de memoria, logrando así la paralelización de las operaciones. La memoria del mLSTM no requiere parámetros adicionales, pero tiene una alta complejidad computacional debido a su memoria matricial y su actualización.

En conjunto, el uso de estas estas dos variantes en la arquitectura xLSTM permite aprovechar las ventajas de ambas. xLSTM utiliza la capacidad de mezcla y actualización de memoria de sLSTM junto con la paralelización y eficiencia computacional de mLSTM. Por tanto, al mezclar estas dos variantes, xLSTM consigue un mayor manejo de secuencias complejas, pero sin verse perjudicado por la excesiva cantidad de información almacenada ni por la falta de procesamiento.

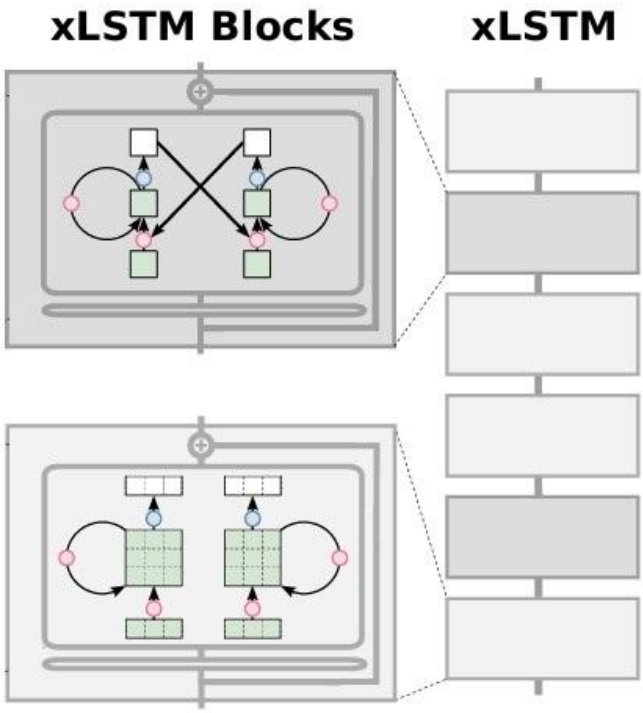

Figura 6: Representación de la estructura de bloques de la arquitectura xLSTM. Figura tomada de [4]

2.6.2. Bloque mLSTM

El bloque mLSTM forma parte de la nueva arquitectura xLSTM. Este bloque aporta a esta arquitectura una mayor capacidad para capturar patrones complejos en series temporales mediante el uso de una estructura matricial. Su misión es transformar y recordar información paso a paso usando una estructura sofisticada y eficiente.

A continuación, procederemos a explicar el funcionamiento del bloque mLSTM, para ayudar a la compresión de la explicación se emplea la Figura 7 que ilustra perfectamente el comportamiento de mLSTM.

mLSTM emplea una prenormalización por capa, es decir, antes de pasar realizar operaciones con la información de entrada, se aplica esta normalización, de esta forma los datos pasan a estar en la misma escala lo que nos ayuda en el aprendizaje del modelo. Además, justo antes de la prenormalización, se utilizada una conexión residual para reservar la entrada original y poder sumarla más adelante a la salida procesada del bloque, facilitando el entrenamiento de modelos profundos.

Luego, con la entrada ya normalizada, se realiza una proyección ascendente con un factor de proyección de valor 2, de forma que la dimensión de la entrada se duplica. Una parte de esta entrada proyectada se manda a la celda mLSTM y la otra va a una puerta de salida y se usará para decidir qué información se convertirá en salida del bloque.

La parte de la entrada proyectada que se envía directamente a la celda mLSTM será utilizada para calcular los tensores de Consulta, Clave y Valor (QKV) mediante transformaciones lineales. Estos tensores se usan para optimizar la búsqueda y uso de la memoria. El tensor de Consulta contiene la pregunta; el tensor Clave contiene las

etiquetas para buscar la información necesaria para responder a la pregunta del tensor Consulta y el tensor Valor es la respuesta.

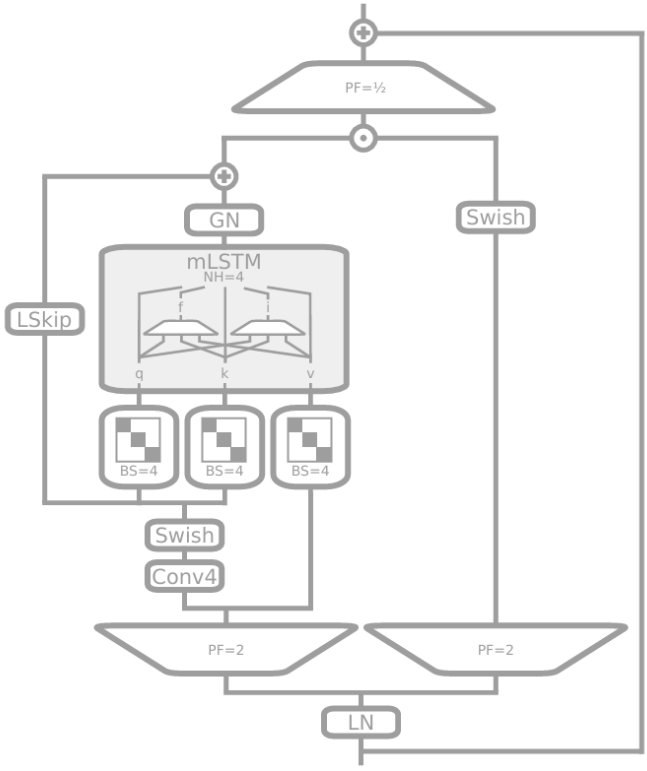

Figura 7: Representación esquemática de la estructura del bloque mLSTM. Figura tomada de [4]

Los tensores de Consulta y Clave se generan a partir de la entrada convolucionada mediante matrices de proyección bloque-diagonales. Estas matrices tienen bloques de tamaño 4, lo que implica utilizar cuatro cabezas (NH=4). La convolución, por su parte, emplea un tamaño de ventana de 4. Por otro lado, el tensor Valor se obtiene a partir de una conexión residual que se salta la convolución.

También, aunque no aparece en la figura, se calculan las entradas de las puertas de entrada, olvido y salida mediante transformaciones lineales basadas en la entrada actual, la entrada proyectada. Y posteriormente, se aplican funciones de activación para obtener los valores finales de las compuertas.

La mLSTM tiene una memoria especial, esta memoria se actualiza usando los tensores de Consulta, Clave, y Valor, junto con las compuertas de entrada y olvido, que controlan la mezcla secuencial. Esta operación actualiza el estado de memoria matricial siguiendo una regla de actualización de covarianza.[4]

A continuación, se actualiza también el estado normalizador y se realiza una recuperación y normalización de información. Para la recuperación de información se utiliza el tensor de Consulta que accede a la información relevante, una vez recuperada, la información se normaliza con el estado normalizador y se emplea como preactivación

del estado oculto, es decir, la información normalizada se usa como entrada de la puerta de salida que es quien calcula el estado oculto.

La compuerta de salida regula esta preactivación mediante una multiplicación elemento a elemento, filtrando los valores y generando el estado oculto final.

La salida de la celda mLSTM se obtine mediante una capa de normalización por grupos (Group Normalization), que aplica una normalización independiente a cada una de las cuatro cabezas. A esta salida se le suma una conexión residual y acto seguido pasa por una puerta de salida externa.

Finalmente, la salida se somete a una post-proyección que reduce su dimensionalidad para que coincida con la dimensión original de la entrada y se le suma la señal de entrada original que se había reservado previamente.

2.6.3. Bloque sLSTM

El bloque sLSTM es una variante de LSTM diseñada para gestionar información mediante de un novedoso mecanismo de mezcla de memoria, que permite distinguir relaciones complejas y dependencias a largo plazo dentro de una secuencia. Este nuevo mecanismo introduce mejoras en la forma en que se actualiza y mezcla la memoria, permitiendo controlar y revisar las relaciones complejas que el modelo aprende y que usa en las predicciones. Además, incorpora mecanismos como normalización por capa y conexiones residuales, que estabilizan el aprendizaje y facilitan el entrenamiento de modelos.

A continuación, nos apoyaremos la Figura 8, imagen proporcionada por los autores de xLSTM, para describir el funcionamiento del bloque sLSTM. En la figura podemos apreciar una representación clara y detallada del comportamiento de este bloque.

El bloque sLSTM integra una arquitectura residual con capa de prenormalización, esto quiere decir que existen ciertos “atajos” que permiten que la señal de entrada se sume a la salida del bloque, evitando que se pierda la información aprendida en pasos anteriores. Cuando la señal entra al bloque sLSTM lo primero que se encuentra es con esta capa de prenormalización. Sin embargo, justo antes de esta capa existe una conexión residual que guarda la señal de entrada para sumarla posteriormente a la salida de la celda sLSTM.

Una vez la señal esta normalizada, tenemos dos caminos, en uno de ellos a la señal de entrada normalizada se le aplica una convolución causal de tamaño de ventana 4 junto con una función de activación Swish. Este camino tiene como destino las puertas de entrada y olvido. El otro camino conduce a la señal hacia la entrada de la celda y hacia la entrada de la compuerta de salida.

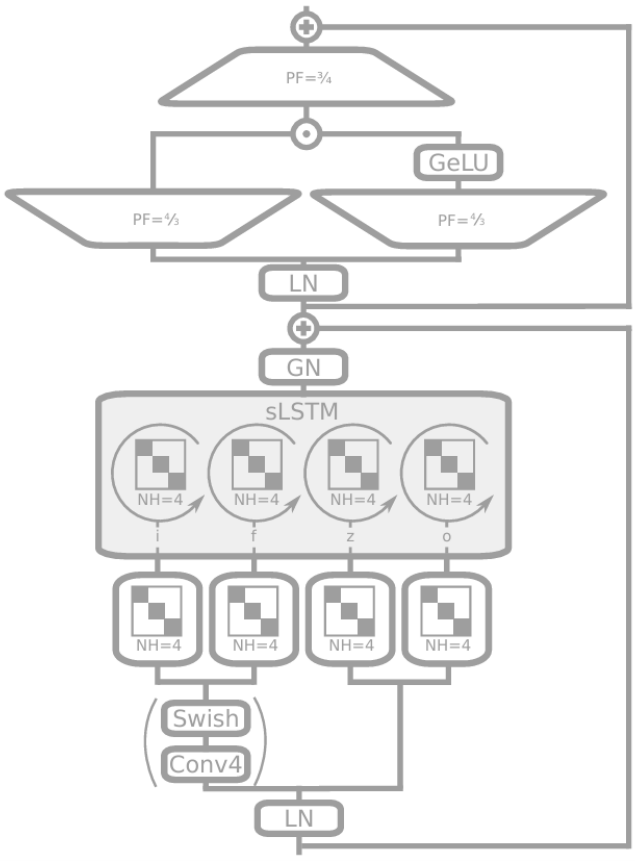

Figura 8: Representación esquemática de la estructura del bloque sLSTM. Figura tomada de [4]

El siguiente paso es la capa bloque-diagonal utilizada para la actualización de la celda. Esta capa lineal tiene cuatro bloques diagonales, también denominados cabezas. Estos bloques diagonales calculan tanto las entradas de las compuertas como la entrada de la celda sLSTM, que es la información candidata a ser almacenada.

Con las preactivaciones para la entrada de la celda y para las compuertas de entrada, olvido y salidas calculadas, ya tenemos las entradas de la celda. Una vez dentro de la celda sLSTM, se aplican funciones de activación a estas entradas.

Después, se actualiza el estado de memoria escalar, aplicando al estado de memoria anterior, la puerta de olvido, y a la entrada de la celda, la puerta de entrada. Esta actualización presenta dependencias con el estado oculto anterior ya que, tanto la puerta de olvido como la puerta de entrada y la entrada candidata se combinan linealmente con el estado oculto anterior en la fase de mezcla secuencial. Esta manera se obtiene el nuevo contenido de la memoria.

A continuación, seguimos con la actualización del estado normalizador, que se actualiza recursivamente y se utiliza para normalizar el estado de memoria escalar. Seguidamente, se calcula el estado oculto combinando el estado de memoria escalar normalizado con la compuerta de salida. El estado de memoria es modulado por la compuerta de salida, que regula la información que pasa hacia el estado oculto final.

Posteriormente, se normaliza el estado oculto mediante una normalización por grupos (Group Normalization), que se comporta igual que una capa de normalización aplicada individualmente a cada una de las cuatro cabezas.

Finalmente, la salida normalizada es procesada por un MLP (Multi-Layer Perceptron) con compuertas. Este MLP realiza una proyección ascendente seguida de una proyección descendente. Y utiliza una función de activación GeLU y un factor de proyección de 4/3 para que su dimensionalidad para que coincida con la dimensión original de la entrada. Con la salida del bloque sLSTM ya calculada, se suma la entrada original que habíamos reservado al inicio del bloque.

2.7. Perceptrón multicapa

El perceptrón multicapa (MLP) es una arquitectura sencilla y clásica dentro de las redes neuronales. Está formado por tres o más capas: una capa de entrada, una o varias ocultas y una de salida. Su principal característica es la alta conectividad entre nodos, donde cada nodo está conectado a todos los nodos de la capa anterior y posterior.

La información dentro de esta red fluye en un solo sentido, hacia delante, desde la capa de entrada hasta la de salida. A este comportamiento es conocido como una feedforward. Cada nodo procesa la entrada convirtiéndola en una combinación lineal utilizando los pesos y el sesgo, después para calcular su salida a esta combinación lineal se le aplica una función de activación no lineal como la función sigmoide. Posteriormente, después de que se haya generado la salida final, durante el entrenamiento, el modelo recalcula los pesos buscando minimizar el error obtenido en la predicción. [28]

Esta arquitectura presenta una gran eficiencia computacional debido a su sencillez. Modelos como TiDE, N-BEATS o N-HiTS, los cuales se van a explicar a continuación, están basados en este modelo.

2.8. TiDE

TiDE es una nueva arquitectura de aprendizaje automático muy reciente, pero de gran interés. El modelo TiDE fue desarrollado por Google Research y Google Cloud, junto con la Universidad de California, San Diego. El artículo [5] presenta esta nueva herramienta.

Este modelo se basa en la arquitectura de perceptrón multicapa (MLP), lo que le otorga una estructura sencilla y eficiente. A diferencia de modelos más complejos como LSTM, TiDE es mucho más sencillo, esta sencillez le convierte en un modelo mucho más rápido. A pesar de ser un modelo muy simple computacionalmente, puede igualar o superar en rendimiento a modelos más complejos. [5]

TiDE toma como entrada todos los datos de la serie temporal y junto con información almacenada en variables adicionales para realizar las predicciones de futuros valores. Su arquitectura se basa en un esquema codificador-decodificador compuesto por redes densas, concretamente por redes MLP. El codificador genera un resumen de la serie

temporal completa y luego este resumen es interpretado por el decodificador para realizar la predicción del horizonte completo en una única operación. Esto significa que, a diferencia de modelos autorregresivos, este paso no se va haciendo con cada predicción individualmente, sino que TiDE está diseñado para predecir el horizonte completo de una vez, lo que aumenta significativamente su eficiencia y velocidad. [5]

Una propiedad interesante del modelo TiDE es el uso de covariables dinámicas y estáticas. En la arquitectura de este modelo, antes del codificador-decodificador, se encuentra una proyección de características (Feature Projection). Esta proyección de características trabaja exclusivamente con covariables dinámicas. Las covariables dinámicas son información adicional que cambia con el tiempo y se utilizada para contextualizar tanto en la ventana temporal (lookback) como en el horizonte de predicción. La misión de la proyección de características es comprimir la información de estas covariables dinámicas de forma que el volumen de datos con el que deba trabajar el modelo sea mucho más pequeño. Al reducir la dimensionalidad de las covariables dinámicas conseguimos almacenar exclusivamente la información relevante que posteriormente será utilizada para la predicción.

Cabe destacar que, al no ser un modelo autorregresivo, TiDE no usa la predicción anterior como entrada para la siguiente predicción, esto provoca que TiDE tenga una complejidad computacional inferior frente a los modelos autorregresivos. Por ejemplo, no hay una repetición del ciclo completo de codificación-decodificación para cada valor a predecir, sino un proceso eficiente que produce el pronóstico completo en una única operación hacia adelante. Todo esto explica la alta eficiencia del modelo TiDE, además de su velocidad y escalabilidad en tareas de pronóstico a largo plazo. [5]

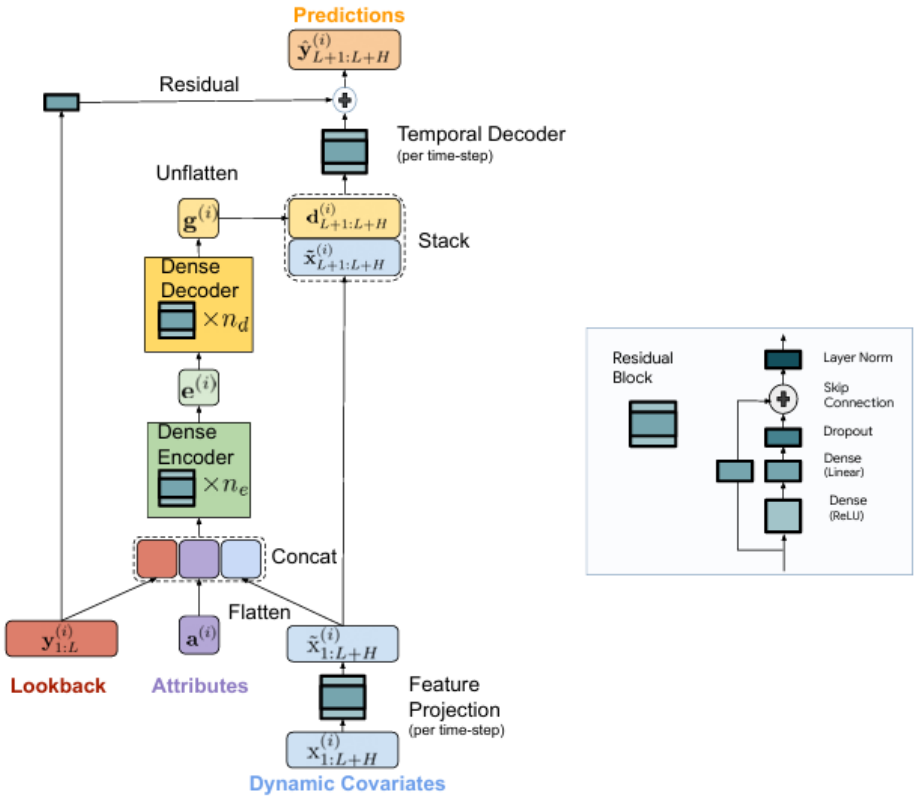

Figura 9: Arquitectura del modelo TiDE. Figura tomada de [5]

La imagen anterior (Figura 9) muestra la arquitectura general de TiDE donde podemos observar lo que se acaba de explicar, primero, las covariables dinámicas y los datos históricos son procesados a través de una capa de proyección de características que reduce su dimensionalidad. Seguidamente, el codificador sintetiza esta información haciéndola más compacta. Y a continuación, el decodificador utiliza la información compactada para generar simultáneamente el pronóstico para todo el horizonte temporal. Además, se incluyen bloques residuales que facilitan el entrenamiento profundo y mejoran la estabilidad del modelo.

TiDE ha demostrado ser un modelo robusto y competitivo en la predicción de series temporales a largo plazo para diversos conjuntos de datos, como aquellos con características no lineales, patrones estacionales complejos y otras particularidades. Todo esto convierte al TiDE en una alternativa prometedora, especialmente para escenarios con grandes volúmenes de datos o donde se requieren resultados rápidos.

2.9. N-BEATS

N-BEATS (Análisis de Expansión de Bases Neurales para Pronóstico de Series Temporales) fue una propuesta innovadora presentada por Boris Oreshkin, Dmitri Carpov, Nicolas Chapados y Yoshua Bengio en 2020. Esta modelo propone un nuevo enfoque en la forma con la que se trabajan los pronósticos de series temporales. A diferencia de otros modelos basados en redes recurrentes, N-BEATS utiliza únicamente redes neuronales feedforward, exactamente redes perceptrón multicapa (MLP) organizadas en bloques residuales. Una característica que lo hace tan especial es que no necesita escalado de entrada, ya que fue diseñada para trabajar con el valor real de los datos sin necesidad de normalizarlos entre 0 y 1. Además, el modelo N-BEATS no emplea la ingeniería de características, es decir, no utiliza información adicional para contextualizar la serie.

La estructura de N-BEATS es personalizable y flexible, permitiendo realizar diferentes combinaciones mediante el uso de múltiples bloques para ajustarse a nuestras necesidades. La variante de N-BEATS interpretable, es capaz de interpretar la información, cualidad poco habitual en las redes neuronales. N-BEATS-I puede dar repuesta a qué componentes de datos influyen en las predicciones, gracias a que se adapta a las diferentes peculiaridades del conjunto de datos utilizado. Su arquitectura está compuesta por pilas (conjunto de bloques) que se encargan de capturar uno de los componentes que caracteriza el conjunto de datos, como puede ser la tendencia o estacionalidad. Al tener una estructurarse dividida en bloques, cada bloque realiza una predicción del pasado de forma separada, intentando capturar y explicar qué partes de los datos pasados están influenciadas por la característica que le corresponde analizar. Esto recibe el nombre de backcast o mirada hacia atrás.[29], [6]

Por otro lado, el modelo genérico N-BEATS (N-BEATS-G) está diseñado de manera simple y buscando una mayor rapidez en el entrenamiento. Esta variante al ser genérica está orientada a no descomponer la serie en componentes como la tendencia o la estacionalidad. Además, requiere de poca información para realizar las predicciones ya

que aprende directamente de la serie temporal. Es una variante caracterizada por su gran versatilidad y capacidad de generalización. [6]

La arquitectura de N-BEATS está formada por pilas y cada pila está compuesto de diferentes bloques, haciendo posible la propagación hacia delante. El apilamiento en bloques es una cualidad que permite refinar el proceso de predicción. La primera capa de bloques hace un "dibujo inicial" del futuro de la serie temporal y las siguientes capas estudian el "dibujo" y tratan de corregir los errores. [6]

Durante el entrenamiento se alterna la predicción de valores futuros (forecast) con la reconstrucción de valores pasados (backcast) lo que permite minimizar el error de sus predicciones. Por otro lado, durante el pronóstico, se combinan todos los resultados de los diferentes bloques, de forma que las deducciones obtenidas de los componentes con influencia en la serie temporal que han sido tratados por los bloques se integran en la predicción final. [6]

N-BEATS es un modelo diseñado específicamente para la predicción de series temporales. Esta red es robusta, versátil y veloz en el entrenamiento, especialmente su variante genérica. Sin duda es una red de gran interés por su capacidad para ofrecer pronósticos precisos y eficientes en una amplia variedad de series temporales.

2.10. N-HiTS

N-HiTS es un modelo presentado en [30] que pretende ampliar el enfoque propuesto por N-BEATS pero buscando reducir la carga computacional y mejorar la precisión en la captura de dependencias a largo plazo. En la figura 10 podemos observar la organización de la arquitectura de N-HiTS. Este modelo al igual que N-BEATS está formado por una estructura jerárquica basada en pilas (stacks) y cada pila contiene varios bloques que a su vez esto bloques están formados por redes MLP con una función de activación ReLU. Los perceptrones multicapa tienen dos salidas: previsión y retroproyección. La retroproyección se utiliza para mejorar las predicciones de los bloques anteriores, mientras que la previsión se utiliza para generar la salida de la pila. La salida de cada pila es la suma de las previsiones de cada bloque.

N-HiTS emplea una técnica denominada muestreo de datos multitasa, esta técnica se aplica usando la capa MaxPool que submuestrea la señal, esta capa proporciona a los bloques diferentes versiones de la señal y cada versión es interpretada por un bloque. De esta forma cada pila recibe una señal con diferente grado de muestreo. Con esto lo que N-HiTS consigue es descomponer la señal de tal forma que en lugar de descomponer la señal en tendencia, estacionalidad y ruido se descompone en diferentes escalas de muestreo. [30]

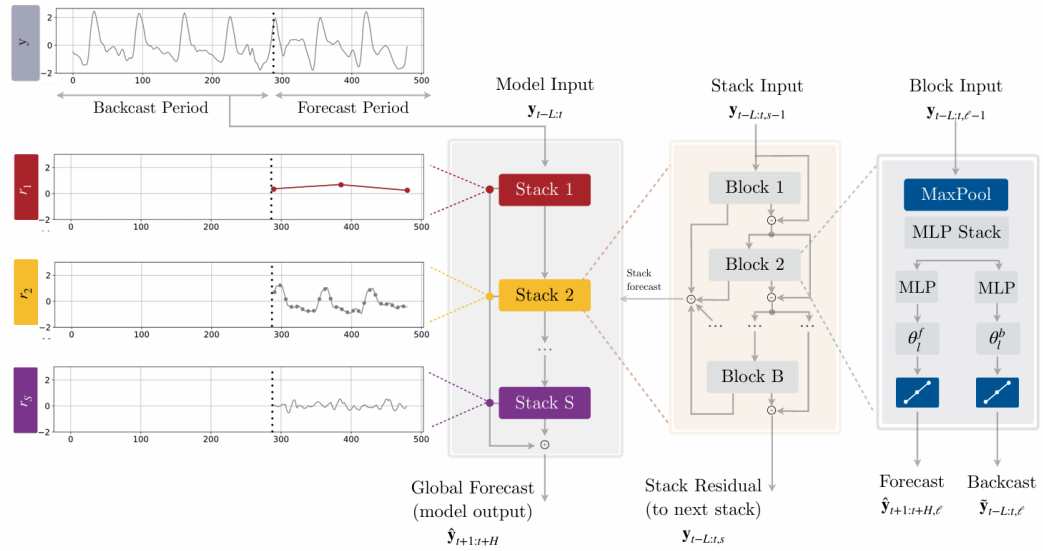

Figura 10: Arquitectura del modelo N-HiTS. Figura tomada de [30]

2.11. TSMixer

TSMixer [31] presenta una arquitectura basada exclusivamente en el perceptrón multicapa. Esta arquitectura está compuesta por dos tipos de bloques: time-mixing y feature-mixing. Los bloques time-mixing se encargan de capturar las dependencias temporales, en cambio los bloques feature-mixing se encargan de integrar la información de las covariables para deducir correlaciones entre estas y el comportamiento de la serie temporal. TSMixer incorpora mecanismo para estabilizar el entrenamiento como las conexiones recurrentes y la normalización.

Este modelo presenta una arquitectura simple con gran capacidad de generalización. Sin embargo, su rendimiento requiere de una buena configuración de lo hiperparámetros, como número de bloques o la dimensión de las capas ocultas. Aun así, TSMixer es una arquitectura eficiente que combina simplicidad y rendimiento, con gran versatilidad en la predicción de series temporales.

2.12. TFT

El Transformador de Fusión Temporal (TFT) fue propuesto por Bryan Lim, Sercan Ö. Arık, Nicolas Loeff y Tomas Pfister en [32]. TFT es un modelo no lineal basado en atención, es decir, no trata todos los pasos temporales con la misma importancia, este tipo de modelos buscan aprender que partes del pasado deben prestan más atención. TFT, al igual que TiDE, fue diseñado para predecir el horizonte temporal. Para ello, el modelo utiliza tanto los datos de la serie temporal como la información de las covariables estáticas y dinámicas.

Sin embargo, una particularidad de TFT es la distinción de dos subtipos de covariables dinámicas. TFT divide las covariables dinámicas en covariables del horizonte de predicción denominadas entradas conocidas y en covariables del pasado temporal denominadas entradas observadas. Para procesar esta información, TFT incorpora bloques de selección de características para identificar las variables más importantes y

emplea codificadores de variables estáticas que resume la información estática y que luego se usará para interpretar la información dinámica.

Este modelo implementa un esquema codificador-decodificador. El codificador tiene como entradas la ventana temporal junto con las entradas observadas y las entradas conocidas que contiene la información del pasado temporal, mientras que el decodificador utiliza las entradas conocidas correspondientes al horizonte temporal para generar las predicciones. Después del codificador-decodificador utiliza un mecanismo de atención interpretable de múltiples cabezales que le permite fusionar la información pasada con la información futura. [32]

2.13. TCN

Las Redes Neuronales Convolucionales Temporales es un modelo basado en convoluciones que fue presentado en [33]. Este modelo utiliza convoluciones causales apiladas, de forma que la salida en cada instante de tiempo depende únicamente de los datos de la serie temporal previos a ese instante. El uso de este tipo de convoluciones le permite la paralelización del procesamiento. Además, aplica la técnica del padding que cosiste en rellenar con ceros para mantener una longitud de secuencia fija y así evitar perder la información de los extremos.

Para capturar dependencias a largo plazo, TCN utiliza convoluciones dilatadas que le permiten ampliar de manera eficiente la cantidad de pasos temporales de la secuencia de entrada que influyen en la predicción de cada salida. Estas capas convolucionales dilatadas son integradas por TCN en bloques residuales, que permiten sumar la entrada del bloque a la salida del mismo. Estos bloques permiten estabilizar el entrenamiento. [33]

En resumen, TCN es un modelo no lineal, simple y eficiente que, gracias al uso de convoluciones causales y a su capacidad de paralelización, puede capturar dependencias a largo plazo de manera eficaz, lo que le convierte en una herramienta adecuada para trabajar con series temporales.

2.14. DeepTCN

El modelo DeepTCN es una extensión de la arquitectura TCN que fue propuesta en [34]. Al igual que TCN incorpora convoluciones causales y dilatadas para procesar series temporales. Con las convoluciones causales consigue que solo se use información pasada en cada instante temporal. Y con las convoluciones dilatadas busca ampliar la cantidad de pasos temporales procesados sin aumentar en exceso coste computacional.

DeepTCN se caracteriza por su arquitectura profunda debido a la incorporación de bloques residuales apilados. Al incorporar esto bloques residuales evita el problema del desvanecimiento pese al haber aumentado la profundidad de la red. Además, el aumento de la profundidad en comparación con TCN fue buscando mejorar la captura de dependencias a largo plazo para poder realizar predicciones del multihorizonte. [34]

Una de las principales mejoras que introduce DeepTCN es la capacidad de realizar pronósticos probabilísticos. DeepTCN busca modelar la distribución de probabilidad de observaciones futuras. Para realizar esta estimación de la densidad de probabilidad utiliza dos enfoques: el paramétrico y el no paramétrico. El enfoque paramétrico busca predecir los parámetros de una distribución hipotética como puede ser por ejemplo la gaussiana. Para realizar esta estimación utiliza el método de máxima verosimilitud. Por el contrario, el enfoque no paramétrico se basa en la regresión de cuantiles, para realizar predicciones de puntos de interés. La estimación de cuantiles, además, permite medir la incertidumbre de la predicción.

2.15. Regresor Pasivo agresivo

El Modelo regresor pasivo agresivo (Passive Agressive Regressor), es un modelo lineal como el modelo de regresión lineal diseñado para ser entrenado en línea, es decir, está diseñado para ser entrenado a medida que se van incorporando nuevas muestras. Utiliza una estrategia de actualización pasiva agresiva en la que si el error cometido en la predicción es pequeño mantiene los pesos, sin embargo, cuando el error cometido excede cierto umbral reajusta radicalmente los pesos. La función de pérdida que utiliza el modelo para comprobar si debe o no reajustar sus pesos es la siguiente:

$$L\big(w;(x,y)\big) = \begin{cases} 0 & |w * x - y| \leq \varepsilon \\ |w * x - y| - \varepsilon & resto \end{cases} \qquad (6)$$

donde w son los pesos, x es el vector de entrada, y es el valor real de salida y ε es el umbral de tolerancia. [35]

Este modelo presenta varias ventajas, como la eficiencia computacional dado que es un modelo bastante sencillo; y el entrenamiento en línea ya que le permite entrenar a medida que llegan los datos, lo que le convierte en una buena opción para nuestro escenario de predicción de tráfico de red.

2.16. Regresión lineal

El modelo de regresión lineal (Linear Regresión), es determinista por lo que no presenta variabilidad en sus resultados. Este modelo está diseñado para capturar dependencias lineales, por lo tanto, tiene una limitación clara que es la dificultad para aprender patrones no lineales.

Este modelo busca aproximar la serie temporal a una combinación lineal de las variables de entrada $(x_0, x_1, x_2, ...)$, para ello utiliza la siguiente ecuación:

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \cdots \qquad (7)$$

donde y es la salida y $(b_0, b_1, b_2, ...)$ son los coeficientes de la combinación lineal. Para conseguir ajustar la combinación lineal el modelo emplea el método de mínimos cuadrados, que consiste en minimizar la suma de los errores cuadráticos entra los valores reales y las predicciones. De esta forma, se obtienen los coeficientes que mejor se ajustan a la serie temporal.[36]

Otro rasgo destacable de este modelo es que presenta una estructura compuesta únicamente por una capa de entrada y una de salida, al no disponer de capas ocultas, lo que lo convierte en un algoritmo simple y de baja carga computacional.

2.17. Naive

El algoritmo Naive es un modelo lineal, muy sencillo y eficiente que se caracteriza utilizar para predecir el valor actual ($\hat{y}_t$), el valor de la serie temporal del instante previo ($\hat{y}_{t-1}$).

$$\hat{y}_t = \hat{y}_{t-1} \tag{8}$$

El modelo Naive es excelente como baseline en la evaluación de predicción de series temporales. De forma que, cualquier modelo que obtenga un rendimiento inferior al modelo Naive carece de utilidad práctica debido a que este no supera a la aproximación más simple posible. [37]

2.18. Conclusiones

Durante el capítulo se ha introducido el concepto de series temporales, así como distintas técnicas empleadas para su análisis y predicción. Se han presentado diferentes redes neuronales, desde los modelos más sencillos como Naive o regresión lineal hasta modelos más recientes como TIDE o N-BEATS, donde se han mostrado una gran diversidad de estrategias de predicción como convoluciones temporales o mecanismos de atención.

Dentro de los distintos modelos analizados, TiDE y N-BEATS al estar basados principalmente en la arquitectura perceptrón multicapa (MLP), ofrecen equilibrio entre rendimiento y eficiencia computacional. Esta característica les permite obtener resultados precisos y robustos con una carga computacional significativamente menor en comparación con las redes recurrentes tradicionales.

Por otro lado, tanto LSTM como su nueva variante xLSTM, aunque son modelos mucho más pesados computacionalmente debido a su naturaleza recurrente, resultan ser de gran interés gracias a su capacidad de capturar dependencias a largo plazo, aspecto crucial en la predicción de series temporales.

En definitiva, la conclusión que extraemos de este capítulo es que la elección del modelo debe basarse no solo en la precisión de las predicciones, sino que también se deben tener en cuenta otros aspectos como eficiencia computacional. Por tanto, la decisión debe ser un equilibro entre precisión y eficiencia computacional.

Capítulo 3

3. Entorno experimental

3.1. Introducción

En este capítulo se detalla el entorno experimental empleado para realizar de este trabajo. El objetivo principal del capítulo es proporcionar detalles sobre la estructura del conjunto de datos utilizado, las arquitecturas implementadas, el diseño experimental, algoritmos de referencia con los que comparar los modelos propuestos, así como la infraestructura utilizada durante los experimentos.

En primer lugar, en el apartado 3.2, se describe los distintos conjuntos de datos utilizados, Esta descripción incluye un análisis de las principales características de las series temporal, ayudando a comprender por qué se han elegido estos datos y cómo contribuyen al desarrollo del trabajo.

A continuación, en el apartado 3.3, se presentan las arquitecturas implementadas, mostrando de esta manera la variedad del estudio. Dentro de este apartado se incluye varios subapartados entre los que se encuentra, un subapartado denominado “Algoritmos de referencia”, donde se exponen los modelos clásicos utilizados en la comparación. Estos modelos han sido previamente descritos en el capítulo anterior.

En el apartado 3.4, se detalla el diseño experimental. Se explica cómo se realiza la división del conjunto de datos en entrenamiento y test. Además, se describe el procedimiento seguido para la selección de hiperparámetros.

Seguidamente, en el apartado 3.5, se expone la infraestructura utilizada. Se especifican los lenguajes de programación y bibliotecas empleadas durante el desarrollo del código, así como el hardware y software donde se han ejecutado los experimentos, con el objetivo de proporcionar una visión completa del entorno computacional.

Por último, los apartados 3.6 y 3.7 están dedicados a las métricas de evaluación y a las conclusiones finales de este capítulo 3. En ellos se explican las métricas utilizadas para evaluar el rendimiento de los modelos y se presentan las observaciones más relevantes sobre el entorno experimental con el que se ha trabajado.

3.2. Descripción de los datos

Hemos trabajado con dos conjuntos de datos diferentes, buscando cubrir un rango amplio de características presentes en estos con el fin de asegurar la validez de los

resultados. Ambos conjuntos de datos pertenecen al conjunto de datos de tráfico de Milán [38]. El conjunto de datos de Milán recopila registros tanto meteorológicos (información de humedad, temperatura y viento) como de telecomunicaciones (actividad en internet, llamadas y SMS) de la cuidad de Milán desde el 1 de noviembre de 2013 hasta el 1 enero de 2014. Estos datos se encuentran agregados por horas y agrupados en bloques de 300 m^2 del interior de la ciudad de Milán. Estos bloques están divididos en celdas de 1x1 metro. Estas celdas de 1x1 metro se denominan píxeles. Del conjunto de datos de Milán se utilizan los datos de la actividad en internet correspondientes a las celdas identificadas como píxeles 4259, 4456, 5060, 4757, 4758, 4857 y 4858.

El primer conjunto con el que se ha trabajado está compuesto por tres series temporales que corresponden de tres píxeles disjuntos identificados como 4259, 4456 y 5060. Para este trabajo se han acortado el número de muestra utilizadas a 1400, por lo que la serie temporal utilizada abarca desde el 1 de noviembre del 2013 a las 00:00 hasta el 29 de diciembre del 2013 a las 07:00.

En la Figura 11, podemos observar que las tres series temporales presentan un patón claro y repetitivo, con fluctuaciones regulares lo que nos sugiere que existe una componente periódica en los datos. Además, en las tres series se aprecia una tendencia decreciente hacia el final. La variabilidad en el rango de valores es notable en las tres series, presentando los valores más altos el píxel 5060, seguido por el 4456 y finalmente el píxel 4259. También se observa una correlación relativamente fuerte entre las series temporales de los píxeles 4259 y 4456, así como entre los píxeles 4259 y 5060. Sin embargo, esta correlación es menos evidente entre los píxeles 4456 y 5060.

Cabe destacar que esta correlación sugiere una dependencia espacial entre ciertos píxeles del grid, probablemente influenciada por la proximidad o características comunes del tráfico en esas zonas.

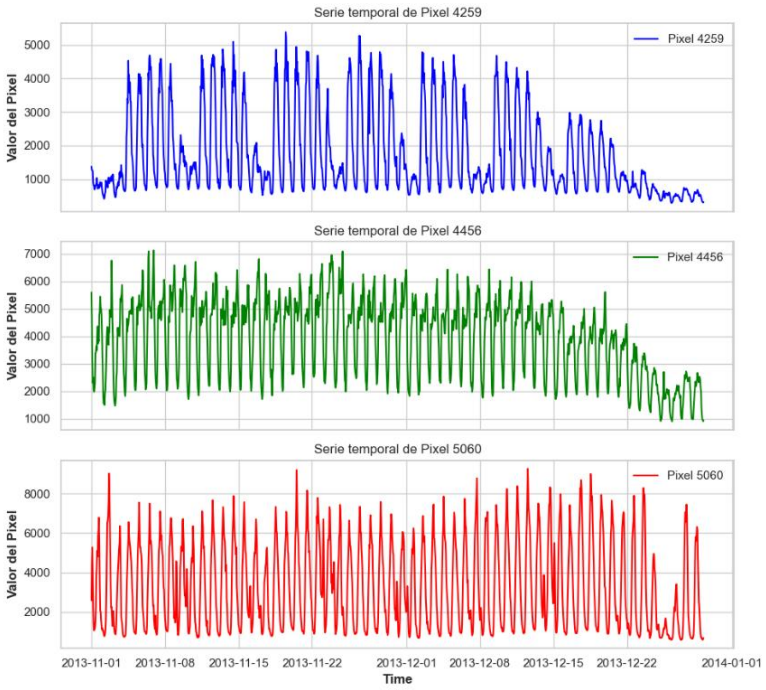

Figura 11: Representación del conjunto de datos inicial

Para el segundo conjunto hemos utilizado cuatro píxeles dispuestos en una cuadrícula 2x2 identificados como 4757, 4758, 4857, 4858. Este segundo conjunto de datos es interesante porque los píxeles están correlados espacialmente. Las series temporales correspondientes a estos píxeles comparten ciertas similitudes con las anteriores, como la presencia de un decrecimiento en la parte final, una alta variabilidad y una clara periodicidad en los patrones de tráfico. Al igual que en el primer conjunto, se seleccionaron las primeras 1400 muestras, que cubren desde el 1 de noviembre del 2013 a las 00:00 hasta el 29 de diciembre del 2013 a las 07:00.

Se puede observar que las cuatro series temporales de los pixeles 4758, 4857 y 4858 presentan patrones muy similares. Sin embargo, la serie temporal 4757 comparte similitudes principalmente con la 4857 pero tanto no con las otras dos y pasa lo mismo con las series 4758 y 4858, entre ambas existen mayores similitudes. En este caso, las similitudes tienden a darse en la dirección vertical dentro del grid, es decir, entre píxeles que están alineados verticalmente. La serie temporal correspondiente al píxel 4857 destaca por tener picos de mayor intensidad en comparación con las demás. Además, las cuatro series muestran un crecimiento del valor de la intensidad de tráfico al inicio de la serie, siendo este crecimiento más marcado en las series de los píxeles 4758 y 4858.

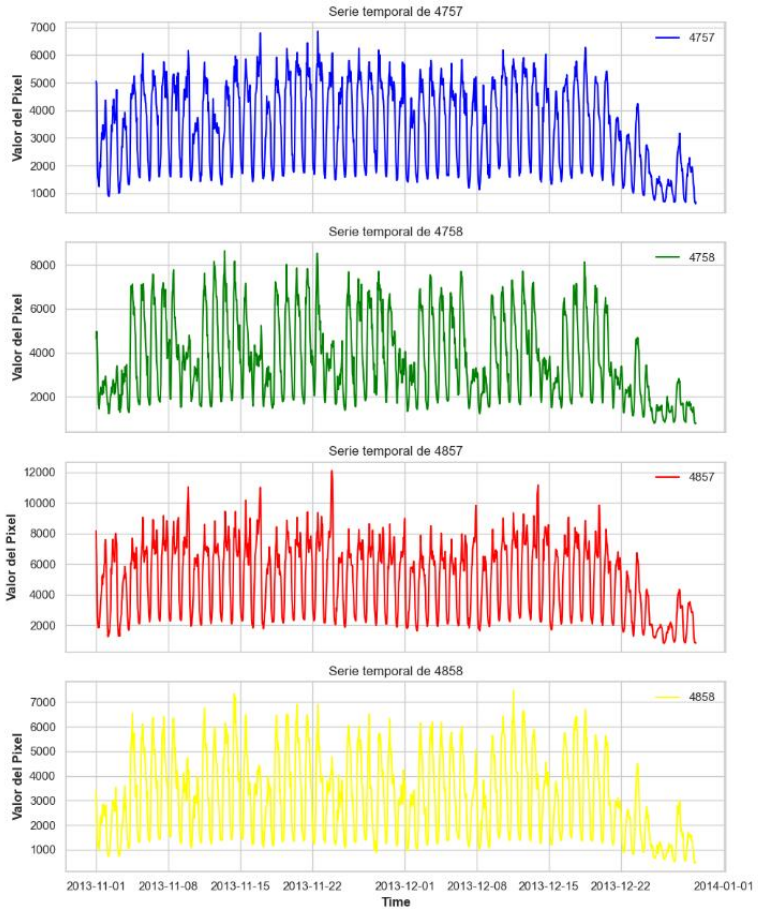

Figura 12: Representación del segundo conjunto de datos utilizado

Cabe destacar que ambos conjuntos de datos disponen unas características favorables para su predicción, dado que no presentan valores atípicos o ruidos aislados. Las series cuentan con una componente periódica bien definida y muestran una clara relación entre ellas.

En resumen, estas características convierten a estos dos conjuntos de datos en un buen punto de partida para la evaluación de modelos de predicción de series temporales, permitiendo observar las capacidades de estos para capturar componentes como la estacionalidad o las tendencias y patrones.

3.3. Arquitecturas implementadas

En esta sección se describen las diferentes arquitecturas que se han utilizado a lo largo del trabajo para la predicción de series temporales. Dentro de estas arquitecturas se encuentran las arquitecturas basadas en xLSTM, que van a ser comparadas tanto entre sí como junto al resto de arquitecturas.

Cabe destacar que algunas de estas arquitecturas empleadas son incrementales, dato que debemos tener en cuenta en la evaluación de los modelos. Los modelos incrementales son aquellos que actualizan sus parámetros durante la fase de prueba con cada muestra. Esto puede ser una ventaja frente a modelos que no son incrementales ya que les permite adaptarse a los nuevos datos. Sin embargo, también puede ser una desventaja porque puede ocurrir que el modelo llegue a sobreajustarse.

Además, se incluirá un subapartado dedicado a los algoritmos clásicos como perceptrón multicapa, regresión lineal y regresor pasivo agresivo, que, a pesar de ser modelos más tradicionales, continúan siendo de gran importancia y referentes en el campo de la predicción de series temporales. Estos métodos clásicos nos ayudarán en la comparación y evaluación del rendimiento de las nuevas arquitecturas implementadas, sirviendo como punto de partida para validar y contextualizar los resultados obtenidos.

3.3.1. Arquitecturas basadas en xLSTM

Para este trabajo se han implementado cuatro arquitecturas diferentes basadas en la variante xLSTM. Dos de estas arquitecturas están formadas exclusivamente por uno de los dos bloques que componen xLSTM, es decir, una utiliza únicamente el bloque mLSTM y la otra el bloque sLSTM. El objetivo de esta elección es comparar modelos con diferente carga computacional, y evaluar de manera individual las habilidades predictivas de cada uno de los bloques fundamentales.

Por otro lado, las otras dos arquitecturas combinan ambos bloques fundamentales, pero de formas diferentes. La primera, que denominaremos simplemente xLSTM, está compuesta por un bloque mLSTM junto con un bloque sLSTM, integrando de esta manera las funcionalidades de ambos bloques. La segunda arquitectura recibe el nombre de xLSTM-TS, nombre que se usó para denominar a esta arquitectura en [15]. Esta versión utiliza un total de cuatro bloques: tres bloques mLSTM y un bloque sLSTM. Ambas configuraciones buscan aprovechar tanto la capacidad para manejar la memoria matricial como la paralelización del bloque mLSTM; junto con la habilidad de gestionar información del bloque sLSTM, con el fin de maximizar el rendimiento predictivo sin comprometer excesivamente la eficiencia computacional.

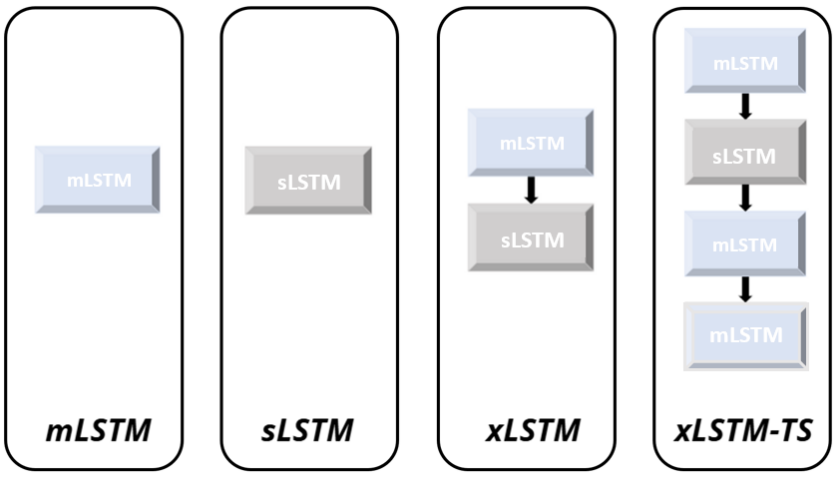

Figura 13: Representación de las diferentes arquitecturas de la variante xLSTM implementadas

3.3.2. Otras arquitecturas empleadas

Además de las arquitecturas basadas en xLSTM, en este trabajo se ha utilizado diversas arquitecturas adicionales para ampliar la comparación y evaluar el desempeño de distintos enfoques de predicción de series temporales. Estas arquitecturas se pueden agrupar en tres bloques principales según su origen y características.

El primer grupo está formado por las arquitecturas empleadas en [15] que incluye modelos recientes como TiDE, DeepTCN, N-HiTS, TCN, TFT, N-BEATS, concretamente su versión genérica; y TSMixer. Estos modelos aplican diferentes enfoques, como pueden ser el perceptrón multicapa, convoluciones temporales, apilamiento de bloques residuales, mezcladores de series temporales o mecanismos de atención para capturar patrones complejos.

El segundo bloque compuesto por el perceptrón multicapa (MLP), regresión lineal (Linear Regression) y regresor pasivo agresivo (Passive Agressive Regressor), agrupa modelos más clásicos y sencillos. Estas arquitecturas son menos complejas y han sido muy utilizadas como referencia por su simplicidad y eficiencia computacional, por tanto, son una buena elección como modelos de referencia.

El tercer grupo está constituido por seis modelos de LSTM: LSTM Ind., LSTM Ind. Online, LSTM Ind. Online Incremental, LSTM Multi., LSTM Multi. Online y LSTM Multi. Online Incremental. Estos modelos incluyen diferentes configuraciones y modos de entrenamiento, algunos de estos han sido entrenados por lotes (batch) y otros por entrenamiento en línea (online). Y algunos utilizan una versión unidimensional (Ind.) y otros una versión multidimensional (Multi.). Además, los modelos LSTM que en este caso son entrenados en línea también son modelos stateful por lo que la actualización del estado oculto se realiza entre muestras en lugar de entre lotes. En cambio, los entrenados por lotes son modelos stateless. Estos modelos stateless reinician el estado oculto con cada nuevo lote de datos, por tanto, la información del estado oculto solo se utilizada en ese lote. Por otro lado, el ajuste de parámetros de los modelos de online LSTM se realiza usando la siguiente función:

$$\theta_t = \underset{\theta_t}{\operatorname{argmin}}(x_t - g(x_{t-1}, x_{t-2}, \dots, x_1, \theta_{t-1}))^2 \qquad (9)$$

donde x_t es el valor de la serie es en instante t, θ_t son los nuevos parámetros y $g(\cdot)$ es una función continua que modela la evolución de la serie temporal.[39]

Incluir todas estas arquitecturas en nuestro trabajo permite que la comparación sea más fiable y robusta al poder evaluar el rendimiento de modelos con características muy diferentes.

3.3.3. Algoritmos de referencia

La importancia de los algoritmos de referencia radica en el hecho de que son una herramienta vital para comparar y evaluar el rendimiento de los modelos avanzados de predicción de series temporales. Estos algoritmos son el punto de partida para valorar el grado de mejora que ofrecen enfoques más modernos como LSTM, xLSTM, TiDE o N-BEATS.

El primer algoritmo de referencia utilizado es el algoritmo Naive, este algoritmo es un modelo muy básico cuyo interés radica en su uso como baseline, de forma que los modelos que no superen en rendimiento predictivo de Naive carecen de interés.

El segundo algoritmo de referencia utilizado es el perceptrón multicapa (MLP) en su versión batch. Esta arquitectura presenta una gran eficiencia computacional debido a su sencillez. Además, modelos como TiDE o N-BEATS, presentes en este trabajo, están basados en este modelo.

El tercer algoritmo de referencia utilizado es el modelo de regresión lineal (Linear Regresión), también en su versión batch. Es un algoritmo simple y de baja carga computacional, lo cual lo convierte en un modelo de referencia adecuado.

El cuarto y último algoritmo de referencia utilizado es el algoritmo regresor pasivo agresivo (Passive Agressive Regressor), en su versión en línea e incremental.

Estos modelos han sido elegidos como algoritmos de referencia por su simplicidad y eficacia, respaldadas por un amplio historial de estudios que validan sus capacidades y serán utilizados como punto de partida para la evaluación de los modelos más recientes. A pesar de las limitaciones que presentan los modelos clásicos para capturar dependencias temporales complejas, permiten realizar comparaciones objetivas con modelos más avanzados. Los modelos más avanzados, por el contrario, aunque son capaces de capturar estas dependencias, pueden ser mucho más pesados computacionalmente.

3.4. Diseño experimental

Para realizar predicciones precisas es fundamental construir un conjunto de datos adecuado que permita al modelo identificar y aprender patrones temporales relevantes. El conjunto de datos debe estar formado por secuencias de datos ordenadas cronológicamente, en nuestro caso estas secuencias tienen periodicidad horaria, lo que

significa que cada punto de la serie temporal corresponde a una medición hecha cada hora del día.

Para entrenar a los modelos necesitamos definir previamente dos parámetros denominados lookback y lookforward. El lookback o ventana temporal es un parámetro numérico que determina la cantidad de pasos temporales previos que se emplean como entrada para realizar la predicción. Por ejemplo, un valor de lookback igual a cinco implica que el modelo usa como entrada los cinco últimos datos de la serie temporal. El lookforward, en cambio, es el número de pasos futuros que el modelo intentará predecir en cada iteración. Si el lookforward tiene un valor de 3, el modelo generará como salida los próximos 3 valores de la serie temporal.

La división del conjunto de datos en subconjuntos de entrenamiento y test es otro aspecto clave. En nuestro caso, la serie temporal consta de 1400 muestras; de estas, el 50% se utiliza para entrenar al modelo (700 muestras) y el 50% restante son las muestras empleadas para el conjunto de test (700 muestras). En los experimentos realizados, no se incluye una fase específica de validación.

3.4.1. Parámetros utilizados para la comparación de las arquitecturas xLSTM

Respecto al proceso de entrenamiento de las arquitecturas xLSTM, el número de épocas utilizado, es decir, las iteraciones sobre el conjunto de datos de entrenamiento, es de 200. Asimismo, el número de ejecuciones fijado para estas arquitecturas es de cinco, de modo que el valor de las métricas de error corresponde a la media obtenida tras repetir cinco veces la ejecución del modelo.

Modelos	Dropout	Épocas	Tamaño de salida	Nº ejecuciones
xLSTM-TS	-	200	1	5
xLSTM	-	200	1	5
mLSTM	-	200	1	5
sLSTM	-	200	1	5

Tabla 1: Parámetros utilizados en los experimentos de las arquitecturas xLSTM

3.4.2. Parámetros utilizados para la comparación con otros modelos de predicción

En la Tabla 2 se presentan los parámetros utilizados en los experimentos para cada modelo. En el caso de las arquitecturas xLSTM, los parámetros fueron elegidos por nosotros, mientras que en los modelos descritos en [15] se adoptaron los valores reportados por los autores, dado que dichos parámetros fueron seleccionados con el objetivo de alcanzar un error de predicción específico.

Modelos	Dropout	Épocas / iteraciones	Tamaño de salida	Nº ejecuciones
xLSTM-TS	-	200	1	5
xLSTM	-	200	1	5
mLSTM	-	200	1	5
sLSTM	-	200	1	5
N-BEATS	-	40	1	1
N-HiTS	-	40	1	1
TiDE	-	200	1	1
TFT	0.1	40	1	1
TCN	0	20	1	1
DeepTCN	0.2	20	1	1
TSMixer	-	40	1	1
MLP	-	Máximo 1000	1	1
Pasive Agression Regression	-	1 x Muestra de entrenamiento	1	1
Linear Regression	-	-	1	1
Naive	-	-	1	1
LSTM Ind.	-	200	1	1
LSTM Ind. Online	-	5 x Muestra de entrenamiento	1	1
LSTM Ind. Online Inc.	-	5 x Muestra de entrenamiento	1	1
LSTM Multi.	-	200	1	1
LSTM Multi. Online	-	5 x Muestra de entrenamiento	1	1
LSTM Ind. Online Inc.	-	5 x Muestra de entrenamiento	1	1

Tabla 2: Parámetros utilizados en los experimentos de las diferentes arquitecturas

3.5. Infraestructura utilizada

Para el desarrollo de este trabajo se ha utilizado como lenguaje de programación el lenguaje Python[2]. Python es un lenguaje de programación muy popular actualmente gracias a su fácil aprendizaje. Este lenguaje es empleado tanto por programadores expertos como por aquellos sin experiencia previa. Se trata de un lenguaje de propósito general, lo que significa que puede ser utilizado para infinidad de aplicaciones como el aprendizaje profundo, la programación web, entre otras.

Python es un lenguaje interactivo, interpretado y orientado a objetos, características que contribuyen a su versatilidad y facilidad de uso. Su popularidad en la programación de aprendizaje profundo se debe en gran parte a la multitud de bibliotecas o librerías especializadas que ofrece, como Pandas, NumPy y PyTorch. El uso de estas librerías facilita la programación haciéndola más ágil y eficiente.

Las principales ventajas de programar en Python son su sintaxis sencilla, la versatilidad que ofrece, su fácil aprendizaje para principiantes, los módulos y bibliotecas que oferta y el hecho de que es un lenguaje de código abierto, lo que permite su uso de

[2] Disponible en: https://www.python.org/downloads/

manera gratuita [40]. Además, Python es compatible con múltiples sistemas operativos, incluyendo Windows y Linux.

Para la edición y ejecución del código, se ha empleado Visual Studio Code[3], un editor de código adoptado por un montón de usuarios, gratuito, de código abierto y multiplataforma. Fue desarrollado por Microsoft y está diseñado para adaptarse a un montón de lenguajes de programación. Es una herramienta ligera y altamente personalizable.

Una de sus características más destacadas es la gran variedad de extensiones con las que cuenta y que se pueden instalar para aumentar sus funcionalidades. De esta forma el usuario puede adaptar la herramienta a sus necesidades desde soporte de lenguajes hasta control de versiones o depuración. VSCode también brinda la posibilidad de establecer conexiones remotas vía SSH, permitiendo la edición de código en servidores remotos. [41]

Para el desarrollo de este proyecto, se han utilizado archivos con extensión .ipynb que corresponden a documentos de Jupyter Notebook[4]. Jupyter Notebook es una aplicación web de código abierto que permite crear y compartir documentos con textos, código ejecutable, imágenes y gráficos. Originalmente, esta aplicación se conocía como IPython Notebook debido a que estaba enfocada exclusivamente en la programación con Python. Posteriormente, se incluyeron nuevos lenguajes de programación como Julia y R, y fue entonces cuando pasó a denominarse como lo conocemos actualmente, Jupyter Notebook, nombre que refleja las iniciales de los tres lenguajes principales. [42]

Un documento de Jupyter Notebook está compuesto por celdas de texto y código. Las celdas de código pueden ejecutarse de forma independiente y generar gráficos o imágenes haciendo nuestra experiencia más interactiva. Además, la incorporación de celdas textos permite agregar explicaciones que facilitan la comprensión tanto del código como del documento en conjunto.

Por otro lado, Jupyter Notebook ofrece la posibilidad de trabajar de manera colaborativa en línea, lo cual facilita que múltiples usuarios pueden editar y ejecutar el mismo notebook de manera simultánea. Pero en nuestro caso los documentos han sido editado y ejecutados a través de VSCode.

Para gestionar las dependencias y entornos de desarrollo, se creó un entorno virtual de Python mediante una carpeta llamada .venv. Este entorno permite aislar la instalación de Python y sus librerías, facilitando la gestión de versiones específicas de paquetes y evitando conflictos con otros proyectos.

El hardware utilizado ha sido un ordenador personal con las siguientes características principales:

- Procesador: AMD Ryzen 7 4700U con gráficos Radeon, 8 núcleos físicos y 8 hilos.
- Memoria física instalada (RAM) 12GB.

[3] Dispinible en: https://code.visualstudio.com/
[4] https://jupyter.org/

- Sistema operativo: Windows.

El hardware empleado es accesible y realista, lo que demuestra que es posible llevar a cabo tareas de aprendizaje profundo y predicción de series temporales sin necesidad de disponer de una infraestructura especializada ni grandes recursos computacionales.

3.6. Métricas de evaluación

Para la comparación de los modelos de aprendizaje profundo se ha optado por el empleo de dos métricas de error, RMSE y MAE. La elección de estas métricas es una práctica habitual para evaluar las capacidades predictivas de los modelos en series temporales. Ambas métricas juntas proporcionan una perspectiva más completa de la precisión de los modelos: la métrica MAE es una medida fácil de interpretar porque representa el error medio mientras que la métrica RMSE castiga los casos en lo que las predicciones se alejan de la media, aportándonos información sobre las dificultades del modelo para ajustarse a los valores reales.

La Raíz del Error Cuadrático Medio (RMSE, Root Mean Squared Error) es una medida muy sensible, ya que no es proporcional a los valores de los errores. Para calcularlo, primero se elevan al cuadrado las diferencias entre los valores reales y las predicciones, luego se obtiene el promedio de estos valores cuadrados y, por último, se extrae la raíz cuadrada de ese promedio. Por tanto, su fórmula es la siguiente:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad (10)$$

donde n es el número de muestras del conjunto de test, y_i representa el valor real y $\hat{y}_i$ el valor predicho.

El Error Absoluto Medio (MAE, Mean Absolute Error) es una medida que nos proporciona el valor real del error cometido por el modelo en la predicción, lo que la hace una métrica de fácil interpretación. Cuanto menor sea el valor del MAE mejor será la capacidad predictiva del modelo. La fórmula para calcular el MAE es la siguiente:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \qquad (11)$$

donde n es el número de muestras del conjunto de test, y_i representa el valor real y $\hat{y}_i$ el valor predicho. [43]

MAE en comparación con RMSE no penaliza en exceso los errores grandes ya que cada error contribuye de forma lineal al cálculo del error promedio. En cambio, RMSE se calcula elevando al cuadrado las diferencias, por lo que los errores que se desvían de la media tienen mayor impacto en el resultado final.

3.7. Conclusiones

En este capítulo se ha definido el enfoque metodológico y técnico de este trabajo. Primero, se han presentado el conjunto de datos, dando a conocer sus características y dependencias entre sí. Con este conjunto busca tener un número suficiente de resultados como para poder sacar conclusiones significativas.

Se han implementado distintas arquitecturas, desde modelos más simples hasta propuestas más recientes, lo que aporta una visión más amplia al comparar tanto el rendimiento como el coste computacional de queda modelo.

Después, procedemos a detallar los diferentes aspectos del diseño experimental, centrándonos en parámetros como el lookback y lookforward, la división del conjunto de datos en entrenamiento y test, y en los mecanismos utilizados para garantizar la reproductibilidad de los resultados. Todo esto ayuda a la fiabilidad de los resultados y a la validez de las conclusiones. En cuanto a las herramientas, se ha trabajado en entornos accesibles y conocidos como Python, Jypiter Notebook o Visual Studio Code.

Por último, se han presentado las métricas de error, MAE y RMSE, que usaremos en el siguiente capítulo para analizar los modelos de redes neuronales, ya que nos proporcionan información sobre el error medio y el nivel de las desviaciones que de las predicciones.

Capítulo 4

4. Resultados y análisis

4.1. Introducción

En este capítulo 4, se analizan y evalúan los resultados de los experimentos realizados. Previamente, se han presentado las diferentes arquitecturas que vamos a comparar, así como las métricas seleccionadas para la evaluación del rendimiento de los modelos. Cabe destacar que, para la comparación de los modelos, el resultado numérico de las métricas es fundamental. Sin embargo, la complejidad computacional de cada arquitectura también es un aspecto que tendremos en cuenta durante la evaluación, ya que afecta directamente a la eficiencia del modelo y a su adecuación para distintas aplicaciones.

Los experimentos realizados corresponden a dos bloques diferentes de series temporales, los cuales han sido presentados en el capítulo anterior. El motivo para ello es que, tras los experimentos del primer conjunto de series temporales, los resultados no fueron concluyentes. Por ello, se decidió ampliar el número de experimentos incorporando un nuevo conjunto de series temporales, con el fin de obtener conclusiones más robustas y generalizables.

En definitiva, este capítulo constituye una pieza clave del trabajo, ya que en él se evalúa la efectividad de las arquitecturas propuestas. A partir del análisis realizado se formularán las hipótesis de las que luego se obtendrán las conclusiones finales y las futuras líneas de investigación.

4.2. Análisis de la comparación entre arquitecturas xLSTM

En este apartado presentamos en la Tabla 3 los resultados obtenidos en las predicciones de cada píxel por las distintas arquitecturas xLSTM. Recordemos que el resultado de estos experimentos son el promedio de 5 experimentos. Este apartado nos va a permitir sacar conclusiones sobre que rasgos de las series temporales son mejor anticipados por cada arquitectura y las posibles dificultades a las que se pueden enfrentar estas cuando el tamaño de ventana utilizado es de mayor o menor longitud.

Modelo	Lookback	Pixel 4259		Pixel 4456		Pixel 5060		Pixel 4757		Pixel 4758		Pixel 4857		Pixel 4858	
		MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
XLSTM-TS	**12**	**128.09**	**171.23**	333.89	439.24	322.32	445.89	261.72	342.78	389.9	518.36	**393.81**	**547.24**	265.29	349.13
XLSTM-TS	**24**	130.8	177.77	**281.67**	**371.95**	353.27	491.87	279.07	366.09	372.95	495.06	460.02	593.67	259.25	351.66
XLSTM-TS	**36**	156.07	204.10	430.14	589.09	336.87	449.32	253.21	334.23	450.02	601.85	532.43	766.68	270.52	357.7
XLSTM	**12**	143.16	186.10	327.74	441.84	389.7	503.12	284.27	383.75	371.86	476.93	425.11	580.44	260.41	344.74
XLSTM	**24**	156.0	208.74	340.06	450.20	**309.18**	**424.90**	**223.07**	**299.38**	**327.62**	**424.1**	433.81	568.95	**233.57**	**313.83**
XLSTM	**36**	144.62	196.52	401.80	524.66	341.30	460.54	265.42	346.46	456.43	629.13	538.84	726.1	253.08	330.13
SLSTM	**12**	183.81	230.91	348.29	437.81	360.03	492.85	257.67	341.43	343.04	439.28	471.85	597.95	264.08	352.55
SLSTM	**24**	173.34	227.00	345.38	440.41	334.36	472.31	248.09	327.82	342.98	437.98	442.11	570.82	243.28	327.81
SLSTM	**36**	167.17	217.50	336.80	414.45	315.77	445.14	262.02	344.66	376.69	473.19	432.32	569.86	245.41	329.84
MLSTM	**12**	168.18	219.89	345.36	458.17	374.63	501.69	259.33	337.27	333.38	429.33	435.72	566.41	264.55	351.94
MLSTM	**24**	170.61	220.02	320.93	404.18	342.48	481.10	312.46	415.36	437.38	568.24	445.35	563.74	260.67	341.57
MLSTM	**36**	157.78	210.71	343.38	440.56	334.08	454.07	304.89	383.88	373.77	490.68	484.6	623.68	295.71	383.1
Naive	**NA**	171.45	281.80	311.59	432.53	572,00	780.65	306.56	420.28	360.07	534.29	510.16	712.75	312.79	448.85

Tabla 3: Resultados de las métricas MAE y RMSE obtenidos por las arquitecturas xLSTM

En la tabla podemos observar que, en las arquitecturas que combinan los dos bloques sLSTM y mLSTM obtienen siempre un mejor resultado que los que solo utilizan uno de estos bloques. En cuatro ocasiones es xLSTM la que obtiene los mejores resultados y en las otras tres es xLSTM-TS. En la primera serie temporal, píxel 4259, las arquitecturas combinadas destacan claramente frente a sLSTM y mLSTM, tanto en MAE como en RMSE, sin embargo, en los demás píxeles esta superioridad no es tan marcada. Observando las tablas podemos distinguir cuatro grupos según los resultados obtenidos por las arquitecturas.

- El primer grupo está formado por la serie temporal del píxel 4259. Los resultados de este grupo muestran que la arquitectura xLSTM-TS sobresalen por encima de las otras tres tanto en MAE como en RMSE. Lo cual puede indicar que esta arquitectura presenta un buen manejo de series con amplitud moderada, sin picos abruptos, con una forma de onda suave y redondeada, tendencia marcada, picos regulares y poca presencia de ruido.

- El segundo grupo está compuesto por las series temporales de los píxeles 4456 y 4857. Estas series son bastante diferentes entre sí. La presencia de ruido en ambas es evidente y tienen menor estabilidad que otros píxeles. Sin embargo, el píxel 4456 presenta oscilaciones irregulares, pero en un rango más acotado, mientras que el píxel 4857 presenta algunos picos más alejados de la media, mostrando una mayor variabilidad con valores que cambian drásticamente y oscilaciones irregulares. Para estas series, la arquitectura con mejor resultado en RMSE y en MAE es xLSTM-TS. En la serie 4456 por detrás de esta arquitectura se encuentra mLSTM, pero en la serie 4857 mLSTM obtiene el segundo mejor resultado de RMSE, pero es sLSTM la que obtiene el segundo mejor resultado de MAE.

- En el tercer grupo se encuentran las series de los píxeles 5060 y 4757. Este agrupamiento puede resultar un poco confuso puesto que las dos series temporales que se encuentran en él tienen características bastante distintas. El píxel 5060 presenta rasgos más suaves, oscilaciones más amplias y regulares alrededor de su media, y con valores más repetitivos. En cambio, el píxel 4757 presenta rasgos más puntiagudos, con oscilaciones de menor amplitud, pero con mayor irregularidad, sugiriendo mayor presencia de ruido. Sin embargo, ambas presentan picos altos, patrón periódico marcado y gran variabilidad. En consecuencia, los resultados suguieren que para series periódicas con gran variabilidad las arquitecturas más adecuadas son xLSTM y sLSTM, al ser estos los modelos con mejor en rendimiento. xLSTM-TS tampoco realiza malas predicciones para este grupo pudiendo reafirmar que esta arquitectura se adapta bien a series con oscilaciones contenidas en un rango definido.

- Y, por último, el cuarto grupo está formado por las series 4758 y 4858. En general, ambas series son muy similares entre sí. Ambas presentan variaciones abruptas con picos pronunciados, alta estacionalidad y un patrón periódico muy marcado y regular, aunque con presencia de ruido. En ambas series la arquitectura con mejores resultados es xLSTM, lo cual puede indicar que este modelo puede

manejar mejor series con un patrón periódico muy marcado con alta variabilidad y ruido. Además, xLSTM-TS, por el contrario, obtiene un rendimiento bajo con errores altos tanto en RMSE como en MAE en ambas series, lo cual también puede sugerir que no se adapta bien a este tipo de series.

En resumen, podríamos decir que las arquitecturas xLSTM y xLSTM-TS son las más adecuadas en general, dado que xLSTM obtuvo los mejores resultados en cuatro de los píxeles analizados, mientras que xLSTM-TS se destacó en los tres restantes. xLSTM-TS realiza mejores predicciones en comparación con el resto de las arquitecturas cuando la serie es suave, estable y periódica o en series con oscilaciones contenidas en un rango acotado, mientras que xLSTM consigue menores errores de predicción con series más variables, con ruido y rasgos puntiagudos. Por otro lado, la variante sLSTM tiende a funcionar mejor con datos de alta variabilidad y ciclos marcados, mientras que mLSTM parece adaptarse bien a series con alta variabilidad y bastante ruido. Además, mLSTM muestra algunas discrepancias entre RMSE y MAE, al contrario que sLSTM.

4.2.1. Evaluación de la influencia del tamaño de ventana en las arquitecturas xLSTM

También es interesante destacar la influencia del tamaño de ventana en el rendimiento de las arquitecturas. En las Figuras 18 a 24 del Apéndice C se muestra una comparación con los valores de la métrica RMSE obtenidos por cada arquitectura (xLSTM-TS, xLSTM, mLSTM y sLSTM) para diferentes longitudes de ventana temporal (lookback) en cada serie temporal. Estas gráficas permiten observar de manera visual qué arquitectura ofrece mejores resultados en cada caso y cómo varía su rendimiento con el tamaño de la ventana. En las figuras se puede apreciar que:

En el píxel 4259, las arquitecturas xLSTM y xLSTM-TS destacan consistentemente sobre mLSTM y sLSTM, consiguiendo los mejores valores de RMSE para todas las longitudes de ventana, especialmente con ventana más cortas.

En los píxeles 4456 y 4857, la arquitectura xLSTM-TS obtienen buenos resultados cuando el tamaño de ventana es pequeño, 12 o 24 muestras. Sin embargo, cuando aumentamos dicho tamaño, la diferencia de xLSTM-TS con los otros modelos se reduce, siendo incluso superada por mLSTM y sLSTM cuando la longitud de ventana es lo suficientemente grande, 36 muestras.

Para los píxeles 5060 y 4757, xLSTM presenta mejores resultados cuando el tamaño de ventana es de 24 muestras, lo que sugiere que esta arquitectura trabaja mejor con ventanas no muy grandes.

En las series de los píxeles 4758 y 4858, xLSTM vuelve a obtener mejores resultados cuando el tamaño de ventana es de 24 muestras. Podríamos decir que el modelo xLSTM se adapta bien a tamaños de ventana pequeños.

xLSTM y xLSTM-TS generalmente alcanzan su mejor rendimiento con ventanas cortas, de 12 o 24 muestras, aunque en estas arquitecturas la elección de la longitud de ventana depende en mayor medida de las características de la serie temporal. En cambio,

mLSTM y sLSTM muestran mayor estabilidad frente a variaciones de la longitud de la ventana.

4.3. Comparación con otros modelos de predicción

En este apartado se analizan en las Tablas 4 a 10 los resultados de las métricas evaluadas (MAE y RMSE) obtenidos en las predicciones de cada píxel realizadas por las distintas arquitecturas objeto de comparación. Estas arquitecturas son TiDE, DeepTCN, N-HiTS, TCN, TFT, N-BEATS, TSMixer, LSTM: LSTM Ind., LSTM Ind. Online, LSTM Ind. Online Incremental, LSTM Multi., LSTM Multi. Online y LSTM Multi. Online Incremental, MLP, Linear Regression y Passive Agressive Regressor, Nauve, mLSTM, sLSTM, xLSTM y xLSTM-TS. En estas tablas también se incluyen resultados para diferentes valores de longitud de ventana, lo que permite comparar el rendimiento de las arquitecturas bajo distintas longitudes. Además, en las tablas podemos ver que algunas de las arquitecturas se encuentran subrayadas, estas corresponden a los modelos incrementales. Al tener estos modelos la capacidad de actualizar sus parámetros con cada predicción, la comparación con el resto de las arquitecturas puede ser un poco injusta. Por esa razón remarcamos esta diferencia de condiciones y, además, se tendrá en cuanta a los ahora de sacar conclusiones sobre los resultados.

El objetivo de este apartado es identificar patrones de comportamiento por arquitectura y serie temporal además de determinar si hay una arquitectura que en general proporcione mejores resultados. También se busca encontrar un comportamiento generalizado y determinar si la elección de la longitud de ventana depende fundamentalmente de la arquitectura o si, por el contrario, esta elección debe ajustarse tanto a la serie temporal como al modelo utilizado. Para ello, primero se analizan los resultados para cada píxel recogidos en las tablas, comentando las características de la serie temporal junto con las arquitecturas que han conseguido errores más pequeños. Y posteriormente, se analizará el rendimiento de las arquitecturas para los distintos tamaños de ventana. Una vez identificado el comportamiento de las arquitecturas, lo usaremos para extraer las conclusiones.

Durante el análisis se hará referencia a las Figuras 25-31 del Apéndice E que muestran una comparación por píxel de los resultados para cada longitud de ventana por arquitectura.

Modelo	Lookback	MAE	RMSE
MLP	12	**105.48**	**158.55**
MLP	24	112.40	159.45
MLP	36	116.70	166.00
XLSTM-TS	12	128.09	171.23
TiDE	24	117.90	177.67
XLSTM-TS	24	130.8	177.77
TiDE	12	124.66	179.92
XLSTM	12	143.16	186.10
Linear Regression	36	143.20	186.35
TiDE	36	125.78	190.05
XLSTM	36	144.62	196.52
LSTM Multi. Online Inc.	24	139.09	199.23
N-BEATS	36	134.92	200.58
Linear Regression	24	145.84	200.90
XLSTM-TS	36	156.07	204.10
LSTM Ind.	24	145.85	204.46
N-BEATS	24	137.41	207.33
XLSTM	24	156.0	208.74
MLSTM	36	157.78	210.71
SLSTM	36	167.17	217.50
MLSTM	12	168.18	219.89
MLSTM	24	170.61	220.02
LSTM Ind. Online Inc.	24	140.88	223.78
LSTM Multi.	24	174.71	226.05
SLSTM	24	173.34	227.00
N-BEATS	12	146.96	229.38
SLSTM	12	183.81	230.91
Linear Regression	12	187.35	239.00
TCN	12	183.45	256.23
TCN	24	203.62	257.51
DeepTCN	24	211.67	258.88
Passive Agressive Regressor	24	164.71	259.32
N-HiTS	24	172.55	263.60
N-HiTS	12	171.74	266.07
TCN	36	202.50	266.39
Passive Agressive Regressor	36	179.44	266.78
N-HiTS	36	175.73	268.41
LSTM Ind. Online	24	211.45	272.51
Naive	NA	171.45	281.80
LSTM Multi. Online	24	244.34	330.28
DeepTCN	36	307.37	351.39
TSMixer	12	243.81	362.05
TFT	12	254.48	391.60
Passive Agressive Regressor	12	261.40	398.31
TFT	24	255.77	416.74
TSMixer	24	285.02	464.93
TFT	36	302.17	483.54
DeepTCN	12	374.71	496.16
TSMixer	36	321.48	520.90

Tabla 4: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 4259

4.3.1. Comparación de los resultados de las diferentes arquitecturas

Observando los resultados podemos determinar que la comparación de estos al incluir solo las arquitecturas no incrementales frente a si incluimos los modelos incrementales es bastante similar. En ambos casos, la arquitectura con mejores resultados en general para todas las series temporales es sin duda TiDE, seguida de cerca por MLP. TiDE presenta el mejor rendimiento en cinco de los siete píxeles y MLP en los otros dos. Luego, arquitecturas como xLSTM-TS, Passive Agressive Regressor, xLSTM, sLSTM, LSTM Multi. Online Incremental y no incremental, presentan un rendimiento inconsistente al conseguir buenos resultados solo para algunos píxeles. Por su parte, los diferentes modelos de LSTM consiguen resultados ambiguos al no poder determinar un rendimiento consistente entre las series ya que, dependiendo de la serie temporal un modelo obtienen mejor rendimiento que otro. También resulta destacable que en las series del segundo conjunto de datos los modelos LSTM multidimensionales no siempre consigan mejores resultados que los modelos unidimensionales al estar estas series correladas espacialmente.

En la Tabla 4, se muestran los resultados correspondientes a las predicciones de la serie del píxel 4259, una serie a priori más fácil de predecir, sin apenas ruido, con una forma de onda suave y con un patrón claro y repetitivo. En la tabla, se puede observar un rendimiento superior de la arquitectura MLP, que ocupa la primera posición en términos de MAE y RMSE para todos los valores de longitud de ventana. Las arquitecturas TiDE y xLSTM-TS muestran resultados competitivos, pero un poco por detrás de MLP. Modelos más clásicos como Linear Regression obtienen también buenos resultados, aunque inferiores. Sin embargo, modelos complejos como TFT, TSMixer y DeepTCN tienen errores significativamente altos.

En la siguiente tabla, Tabla 5, TiDE logra superar a MLP, consiguiendo un valor de RMSE de 287.81 y un valor de MAE de 214, mientras MLP consigue un valor de RMSE de 293.43 y 218.06 de MAE. Para casi todos los tamaños de ventana TiDE obtiene errores más bajos que MLP, aunque la diferencia de rendimiento entre ambos modelos es reducida. Otros modelos como Passive Agressive Regressor y LSTM Multi. Online Inc., ambas arquitecturas incrementales y N-BEATS, no incremental, han alcanzado resultados aceptables. La serie temporal predicha en este caso es una serie con bastante ruido, irregular con rango definido y no presenta un patrón tan claro. Los modelos TFT, TSMixer y DeepTCN vuelven a situarse en las últimas posiciones de la tabla con altos errores.

Los resultados de la serie 5060, Tabla 6, muestran que TiDE vuelve a tener un rendimiento superior al resto de arquitecturas tanto en RMSE y MAE. Por detrás del TiDE se encuentran las cuatro arquitecturas xLSTM, con xLSTM a la cabeza. En esta serie MLP no ha obtenido buenos resultados consiguiendo grandes errores lo que puede indicar que no se adapta tan bien a series con picos regulares, formas puntiagudos y gran variabilidad. Con buenos resultados en MAE pero no tanto en RMSE encontramos a las arquitecturas N-BEATS y LSTM Ind. Online Incremental. Además, por tercera vez

consecutiva, TFT, TSMixer y DeepTCN vuelven a situarse en las últimas posiciones de la tabla.

Los resultados de la serie temporal del píxel 4757, serie con forma puntiaguda, irregular y gran variabilidad; posicionan a la arquitectura TiDE en la primera y segunda posición de la Tabla 11. La arquitectura MLP muestran resultados competitivos, pero un poco por detrás de TiDE. Modelos como LSTM Multi. Online Inc., N-BEATS y xLSTM tienen también buenos resultados. Los modelos TFT, TSMixer y DeepTCN siguen teniendo errores significativamente altos.

La Tabla 8 correspondiente al píxel 4758 presenta los resultados obtenidos por las distintas arquitecturas al aplicarse sobre una serie con alta estacionalidad y variabilidad. En la tabla, TiDE aparece como líder seguido de MLP, aunque la diferencia en este caso es muy reducida entre ambas arquitecturas. Los modelos Linear Regression y LSTM Multi. Online Incremental obtienen buenos resultados. De nuevo, los modelos TFT, TSMixer y DeepTCN obtienen errores altos.

Seguimos con la Tabla 9, donde vuelven a ganar TiDE y MLP tanto en MAE como en RMSE. Por detrás de estas dos arquitecturas se sitúan en cuanto al RMSE los modelos LSTM Ind. y N-BEATS y en cuanto al MAE el modelo LSTM Multi. Online Incremental. Y los modelos que presentan rendimientos más bajos son TFT, TSMixer y DeepTCN.

Por último, en la Tabla 10, nuevamente se sitúan en primeras posiciones MLP y TiDE, pero para esta serie, es MLP quien consigue los mejores resultados. Podemos observar que, en series con un patrón periódico marcado y variabilidad, propiedades que caracterizan a esta serie, como a la serie del píxel 4758, la diferencia entre MLP y TiDE en ambas series es mínima. Siguiendo con los resultados, más atrás en la tabla se encuentran las tres arquitecturas LSTM multidimensionales, llama la atención que en este caso la arquitectura no incremental este por delante de la incremental, sin embargo, la diferencia entre las tres es mínima. Por otro lado, una vez más, los modelos TFT, TSMixer y DeepTCN vuelven a estar en las últimas posiciones de la tabla.

En definitiva, podemos observar que las series con picos más altos como las de los píxeles 5060, 4758 y 4857 son las que tienen errores más elevados. Además, tanto TiDE como MLP han mostrado una buena capacidad de adaptación a las diferentes características de las series temporales utilizadas. TiDE ha alcanzado mejores resultados en general. Sin embargo, MLP la supera en rendimiento en series sin apenas ruido, con una forma de onda suave y con un patrón claro y repetitivo y la iguala en series de gran variabilidad. Los modelos xLSTM, xLSTM-TS, Passive Agressive Regressor, Linear Regression, N-BEATS y LSTM Multi. Online Incremental obtienen resultados competitivos, pero inconsistentes, situándose en varias ocasiones en las diez primeras posiciones de las tablas. Por otro lado, TFT, TSMixer y DeepTCN han obtenido en todas las series los peores resultados, lo que indica una capacidad predictiva limitada. También cabe destacar que la variante xLSTM no siempre supera a la arquitectura LSTM tradicional.

En cuanto a la adaptación a los diferentes tamaños de ventana, en las Figuras 24-30 se observa que, por norma general, TiDE y MLP presentan una gran estabilidad frente a variaciones en la longitud de la ventana, por lo que la elección del tamaño parece no

tener un impacto significativo en el rendimiento de estas arquitecturas. Por otro lado, tanto Passive Agressive Regressor como Linear Regression y N-BEATS obtienen siempre peores resultados con un tamaño de ventana de 12 muestras, por tanto, podemos determinar que estas arquitecturas trabajan mejor con tamaños de ventana grande. Esto puede deberse en el caso de Linear Regression y Passive Agressive Regressor a la simplicidad de estas arquitecturas y en el caso de N-BEATS este comportamiento puede estar asociado al hecho de que esta arquitectura no emplea mecanismos de mezcla de memoria ni información adicional, el modelo aprende únicamente con la información de la ventana. Asimismo, como se ha mencionado anteriormente, las arquitecturas xLSTM y xLSTM-TS no muestran un mejor rendimiento con un tamaño específico, sino que esta elección depende en mayor medida de las características de la serie temporal. También arquitecturas como DeepTCN, TCN y TSMixer se ven muy afectadas por la elección de ventana.

4.4. Discusión

En la sección anterior se han comentado los resultados obtenidos, sin embargo, es importante recordar, como ya se señaló en capítulos previos, que la evaluación de los modelos no debe basarse solo su rendimiento, sino también deben considerarse otros aspectos como la carga computacional o la forma de entrenar del modelo. Un modelo que utiliza una actualización incremental tiene una mayor capacidad de adaptación al problema que nos planteamos de la predicción de tráfico de red que un modelo que se entrena en modo batch. Los modelos incrementales son capaces de actualizarse progresivamente a medida que llegan nuevos datos al contrario que los modelos batch que requieren de ser rentrenados. Por tanto, pese a que TiDE y MLP han obtenidos el mejor rendimiento y son modelos de baja carga computacional puede que no sean tan adecuados para el problema que nos planteamos. Por otro lado, el modelo LSTM Multi. Online Incremental es un modelo con un rendimiento competitivo, entrenado en línea, pero con una elevada complejidad lo que lo convierte en un modelo pesado computacionalmente, además requiere de mayor tiempo de ejecución en comparación con modelos más sencillos. Otro modelo entrenado en línea es Passive Agressive Regressor, este modelo es adecuado para nuestro problema por su capacidad de actualización continua y a su baja carga computacional, sin embargo, su capacidad predictiva es inferior a TiDE y MLP. Los modelos xLSTM y xLSTM-TS también muestran una capacidad de predicción aceptable, pero no son modelos entrenados en línea, además presentan una elevada carga computacional, mayor que la versión LSTM clásica, no obstante el tiempo de ejecución de ambos modelos es bastante alto, por estas razones su aplicación en nuestro problema de predicción de tráfico de red presenta bastantes limitaciones. Modelos como Linear Regression, o N-BEATS también obtienen resultados competitivos, sin embargo, son modelos entrenados en modo batch. En el caso de N-BEATS, la carga computacional resulta comparable a la del modelo TiDE, mientras que Linear Regression, presenta una carga computacional inferior tanto a TiDE como a MLP, sin embargo, esta ventaja es eclipsada por su menor capacidad predictiva. Por tanto, N-BEATS y Linear Regression carecen de ventajas frente a TiDE y MLP.

Cabe señalar que la comparación de los distintos modelos se ha realizado bajos condiciones heterogéneas en cuanto número de ejecuciones y épocas de entrenamiento. Esta circunstancia limita la validez de los resultados a la hora de comparar estrictamente la capacidad predictiva de cada arquitectura. Pese a ello, el análisis realizado resulta valioso como una primera aproximación, ya que permite identificar tendencias generales de rendimiento y orientar futuras evaluaciones.

4.5. Conclusiones

En conclusión, el análisis realizado en este capítulo ha permitido identificar la capacidad predictiva de las diferentes arquitecturas evaluadas. En un primer momento, al comparar únicamente las diferentes arquitecturas de la nueva variante de LSTM, se comprobó que la combinación de los bloques mLSTM y sLSTM ofrece mejores resultados que su uso individual. xLSTM y xLSTM-TS obtuvieron las mejores predicciones, mostrando mayor facilidad de adaptación tanto en series suaves y periódicas como en aquellas más irregulares y ruidosas.

No obstante, al incluir en el análisis otros modelos de aprendizaje profundo, se observó que TiDE y MLP alcanzan un rendimiento superior al resto de arquitecturas. Ambas arquitecturas además destacan por su estabilidad frente a variaciones en la longitud de la ventana y su capacidad para adaptarse a distintas tendencias y patrones temporales. Las dos arquitecturas requieren baja carga computacional, un aspecto que debemos tener en cuenta a la hora de seleccionar y compararlas. Por esta razón pueden ser la opción más adecuada para nuestro problema, no obstante, esta conclusión debe interpretarse con cautela, ya que la comparación entre modelos no se realizó bajo condiciones heterogéneas.

Otras arquitecturas, como Passive Aggressive Regressor, Linear Regression, LSTM Multi. Online Inc., xLSTM, xLSTM-TS y N-BEATS aunque logran resultados competitivos, su rendimiento es más inconsistente y dependiente de factores como el tamaño de ventana. En particular, Passive Agressive Regressor y N-BEATS obtienen siempre mejores resultados con un tamaño de ventana grande. Por otro lado, modelos como TFT, TSMixer y DeepTCN presentan una capacidad predictiva limitada en todas las series, por debajo de Naive, modelo que hemos utilizado como baseline.

Cabe destacar que Linear Regression, Passive Aggressive Regressor y N-BEATS presentan baja carga computacional, sin embargo, la diferencia en rendimiento observada con MLP y TiDE resulta determinante. Por el contrario, xLSTM, xLSTM-TS y LSTM Multi. Online Incremental requieren un mayor coste computacional, lo cual representa una limitación para su aplicación en predicción de tráfico de red, dado que este se caracteriza por cambios rápidos y dinámicos.

Capítulo 5

5. Conclusiones y trabajo futuro

5.1. Conclusiones

La predicción del tráfico en red es una actividad de gran relevancia en el ámbito de las telecomunicaciones, ya que permite anticipar patrones de congestión, la optimización de rutas de reenvío, mejorar la planificación de infraestructuras y la distribución eficiente de los recursos disponibles. El principal objetivo de este trabajo es analizar y comparar la capacidad predictiva de diferentes modelos de redes neuronales aplicándolos específicamente para predecir el tráfico de red. Para alcanzar este objetivo, se recurrió a diferentes arquitecturas, algunas de ellas propuestas en [15] así como a un conjunto de datos procedente del tráfico de red en la ciudad de Milán [38], del cual se seleccionaron determinados píxeles para los experimentos. Este Trabajo de Fin de Grado es un estudio exploratorio centrado en el análisis de los resultados obtenidos. Dicho análisis busca proporcionar una primera aproximación del comportamiento de las distintas arquitecturas evaluadas en el contexto de la predicción de tráfico en red. Para ello, no solo se han considerado el rendimiento alcanzado por los modelos, sino también otros aspectos como el tipo de entrenamiento del modelo y la carga computacional de cada arquitectura.

El análisis realizado se estructuró en dos fases. En la primera, se comparan bajo condiciones homogéneas las diferentes arquitecturas implementadas de la variante xLSTM, concretamente las arquitecturas xLSTM, xLSTM-TS, mLSTM y sLSTM. En la segunda fase, se amplió el estudio incluyendo un conjunto más diverso de modelos, tanto modelos más clásicos como algunos más recientes con el objetivo de conseguir unas primeras aproximaciones del rendimiento de estas arquitecturas.

Se observó que modelos como TiDE y MLP destacan frente al resto por su rendimiento global, logrando los mejores resultados en las series evaluadas; y baja carga computacional, lo que las convierte en buenas candidatas para nuestro problema. En contraste, arquitecturas como Passive Aggressive Regressor, Linear Regression y N-BEATS presentan baja carga computacional, pero su rendimiento limita su utilidad práctica. Por otro lado, xLSTM, xLSTM-TS y LSTM Multi. Online Incremental pese a conseguir resultados competitivos, requieren de un coste computacional elevado, lo que restringe su utilidad en la predicción de tráfico de red. Finalmente, modelos como TFT, TSMixer y DeepTCN presentan un comportamiento menos favorable en todas las series analizadas.

5.2. Trabajo futuro

Para próximos trabajos a futuro, se plantea mejorar la metodología de este trabajo para aumentar la validez de los resultados. En primer lugar, sería recomendable equilibrar el número de repeticiones y de épocas de entrenamiento para todos los modelos. En segundo lugar para mejorar la robustez de la comparación se debería incluir un análisis estadístico más exhaustivo, incorporando por ejemplo intervalos de confianza.

También se plantea la posibilidad de ampliar el conjunto de datos, incorporando series temporales más complejas. Las series utilizadas en este estudio presentan patrones y tendencias bastante parecidos, de forma que seria interesante incorporar series con distintos rasgos. La incorporación de series con mayor variabilidad, ruido o formas más irregulares permitirá evaluar la capacidad predictiva de los modelos propuestos de manera más generalizada debido a la inclusión de escenarios más diversos y heterogéneos.

Otra posible vía para próximos trabajos que sería interesante explorar, es la inclusión de nuevas arquitecturas avanzadas que hayan demostrado un alto rendimiento en tareas de predicción de series temporales. En particular, los modelos basados en Transformers, como LogTrans [44] o FEDformer [45], resultan de gran interés, ya que son ampliamente utilizados en el ámbito del procesamiento secuencial por su capacidad para interpretar y capturar dependencias a largo plazo, así como para predecir patrones de elevada complejidad. También, sería interesante adaptar estos modelos para incorporar sus versiones online o incrementales, de modo que puedan actualizar sus parámetros con cada muestra.

Por otro lado, se propone investigar técnicas de preprocesamiento orientadas a la reducción de ruido, como las aplicadas en [15]. En este artículo, se emplea un filtrado wavelet para suavizar la forma de onda de las señales temporales, lo que mejora la calidad de los datos de entrada consiguiendo así una mayor precisión en las predicciones. Aplicar este tipo de técnicas al conjunto de datos utilizado en este estudio permitiría analizar su impacto en el rendimiento de los modelos y valorar su utilidad como paso previo al entrenamiento.

Referencias

[1] A. Azzouni and G. Pujolle, “A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction,” Jun. 2017, [Online]. Available: http://arxiv.org/abs/1705.05690

[2] “Time Series Analysis and Forecasting,” GeeksforGeeks. Accessed: Aug. 07, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/time-series-analysis-and-forecasting/

[3] M. R. Joshi and T. H. Hadi, “A Review of Network Traffic Analysis and Prediction Techniques,” Jul. 2015. Accessed: Sep. 07, 2025. [Online]. Available: https://arxiv.org/abs/1507.05722

[4] M. Beck *et al.*, “xLSTM: Extended Long Short-Term Memory,” Dec. 2024, Accessed: Sep. 15, 2025. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3737916.3741333

[5] A. Das, W. Kong, A. Leach, S. Mathur, R. Sen, and R. Yu, “Long-term Forecasting with TiDE: Time-series Dense Encoder,” Apr. 2024, [Online]. Available: http://arxiv.org/abs/2304.08424

[6] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, “N-BEATS: Neural basis expansion analysis for interpretable time series forecasting,” Feb. 2020, [Online]. Available: http://arxiv.org/abs/1905.10437

[7] R. Lasser, “Engineering Method, Electrical and Computer Engineering Design Handbook,” Tufts University School of Engineering. Accessed: Sep. 16, 2025. [Online]. Available: https://sites.tufts.edu/eeseniordesignhandbook/2013/engineering-method/

[8] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Second Edition. O’Reilly, 2019.

[9] F. Chollet, *Deep Learning with Python*, Second Edition. New York: Manning Publications Co. LLC, 2021.

[10] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are Transformers Effective for Time Series Forecasting?,” 2023. Accessed: Sep. 08, 2025. [Online]. Available: https://arxiv.org/abs/2205.13504

[11] E. Lanza, “How to Apply Transformers to Time Series Models,” Intel Tech, Medium. Accessed: Sep. 08, 2025. [Online]. Available: https://medium.com/intel-tech/how-to-apply-transformers-to-time-series-models-spacetimeformer-e452f2825d2e

[12] Z. Wang, J. Hu, G. Min, Z. Zhao, Z. Chang, and Z. Wang, “Spatial-Temporal Cellular Traffic Prediction for 5G and Beyond: A Graph Neural Networks-Based Approach,” *IEEE*

Trans Industr Inform, vol. 19, no. 4, pp. 5722–5731, Apr. 2023, doi: 10.1109/TII.2022.3182768.

[13] "Aprendizaje Profundo (Deep Learning)." Accessed: Sep. 08, 2025. [Online]. Available: https://pglez82.github.io/DeepLearningWeb/

[14] "GitHub - NX-AI/xlstm: Official repository of the xLSTM." Accessed: Sep. 08, 2025. [Online]. Available: https://github.com/NX-AI/xlstm/

[15] G. L. Gil, P. Duhamel-Sebline, and A. McCarren, "An Evaluation of Deep Learning Models for Stock Market Trend Prediction," Aug. 2024, [Online]. Available: http://arxiv.org/abs/2408.12408

[16] A. Gómez, "Series temporales," Matemáticas y sus fronteras. Accessed: Aug. 10, 2025. [Online]. Available: https://blogs.mat.ucm.es/agomez-corral/2021/08/12/series-temporales/

[17] J. M. Marín, "Series Temporales Introducción." Accessed: Aug. 27, 2025. [Online]. Available: https://halweb.uc3m.es/esp/personal/personas/jmmarin/esp/edescrip/tema7.pdf

[18] "Time Series Decomposition Techniques," GeeksforGeeks. Accessed: Aug. 29, 2025. [Online]. Available: https://www.geeksforgeeks.org/python/time-series-decomposition-techniques/

[19] Instituto Nacional de Estadística, "Análisis espectral y ajuste estacional," Madrid, España, Oct. 2012. Accessed: Sep. 07, 2025. [Online]. Available: https://www.ine.es/clasifi/analisisyajuste.pdf

[20] Universidad de Sevilla, "Autocorrelación en series temporales." Accessed: Sep. 08, 2025. [Online]. Available: http://innoevalua.us.es/files/perpage/@disenoslongitudinalesdeinvestigacionaplicados@autocorrelacion.pdf

[21] J. A. Mauricio, "Introducción al Análisis de Series Temporales." Accessed: Sep. 07, 2025. [Online]. Available: https://www.ucm.es/data/cont/docs/518-2013-11-11-JAM-IAST-Libro.pdf

[22] "Aprendizaje automático: qué es, tipos y cómo se aplica," Blog UE. Accessed: Sep. 20, 2025. [Online]. Available: https://universidadeuropea.com/blog/que-es-aprendizaje-automatico/

[23] "Introducción a las Redes Neuronales Recurrentes (RNN)." Accessed: Aug. 31, 2025. [Online]. Available: https://imaster.academy/contenidos-tematicos/talentotech/TalentoTech/M3unidades/Inteligencia%20artificial/Innovador/Unidad1/assets/files/L2_1_Introduccion-aRedesNeuronalesRecurrentesRNN.pdf

[24] A. Profundo, P. González, and P. Pérez, "Tema 4: Arquitecturas y aplicaciones de las redes neuronales profundas."

[25] C. Olah, "Understanding LSTM Networks," colah's blog. Accessed: Aug. 10, 2025. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[26] “What is LSTM - Long Short Term Memory?,” GeeksforGeeks. Accessed: Aug. 10, 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/

[27] “Introducción a la memoria a corto-largo plazo (LSTM),” MathWorks. Accessed: Aug. 10, 2025. [Online]. Available: https://es.mathworks.com/discovery/lstm.html

[28] R. A. García and A. F. Lepera, “El Perceptrón Multicapa - Modelo Matemático e Implementación en Python,” *Revista de Investigación en Modelos Matemáticos Aplicados a la Gestión y la Economía*, vol. 1, pp. 31–57, 2024.

[29] N. Bhatnagar, “N-BEATS: The Unique Interpretable Deep Learning Model for Time Series Forecasting,” Medium. Accessed: Sep. 08, 2025. [Online]. Available: https://medium.com/@captnitinbhatnagar/n-beats-the-unique-interpretable-deep-learning-model-for-time-series-forecasting-8dfdefaf0e34

[30] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and A. Dubrawski, “N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting,” Nov. 2022, [Online]. Available: http://arxiv.org/abs/2201.12886

[31] S.-A. Chen, C.-L. Li, N. Yoder, S. O. Arik, and T. Pfister, “TSMixer: An All-MLP Architecture for Time Series Forecasting,” Sep. 2023, [Online]. Available: http://arxiv.org/abs/2303.06053

[32] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, “Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting,” Sep. 2020, [Online]. Available: http://arxiv.org/abs/1912.09363

[33] S. Bai, J. Z. Kolter, and V. Koltun, “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,” Apr. 2018, [Online]. Available: http://arxiv.org/abs/1803.01271

[34] Y. Chen, Y. Kang, Y. Chen, and Z. Wang, “Probabilistic forecasting with temporal convolutional neural network,” *Neurocomputing*, vol. 399, pp. 491–501, Jul. 2020, doi: 10.1016/j.neucom.2020.03.011.

[35] K. Crammer, O. Dekel, and J. Keshet, “Online Passive-Aggressive Algorithms Shai Shalev-Shwartz Yoram Singer †,” 2006.

[36] Esri, “Cómo funciona el algoritmo Regresión lineal,” ArcGIS Pro Documentation. Accessed: Sep. 20, 2025. [Online]. Available: https://pro.arcgis.com/es/pro-app/3.3/tool-reference/geoai/how-linear-regression-works.htm

[37] R. J. Hyndman and G. Athanasopoulos, “3.1 Some simple forecasting methods,” in *Forecasting: Principles and Practice*, 2nd Edition., 2018. Accessed: Sep. 17, 2025. [Online]. Available: https://otexts.com/fpp2/simple-methods.html?utm_source=chatgpt.com

[38] M. Abdullah, “Milan Dataset,” Jun. 2022. doi: https://dx.doi.org/10.21227/7gzy-6552.

[39] P. Casaseca-De-La-Higuera *et al.*, “Online Active Learning For LSTM-Based Network Traffic Prediction,” en Proc. 16th Int. Conf. Softw., Knowl., Inf. Manage. & Appl. (SKIMA). 2025. En prensa.

[40] “¿Qué es Python y para qué se usa? Guía para principiantes,” Coursera. Accessed: Aug. 07, 2025. [Online]. Available: https://www.coursera.org/mx/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[41] F. de Z. García, “¿Qué es Visual Studio Code y cuáles son sus ventajas? | Blog de Arsys | Blog de Arsys,” Blog de Arsys. Accessed: Aug. 08, 2025. [Online]. Available: https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas

[42] M. Driscoll, “Jupyter Notebook: An Introduction,” Real Python. Accessed: Aug. 08, 2025. [Online]. Available: https://realpython.com/jupyter-notebook-introduction/

[43] P. González and P. Pérez, “Tema 3: Entrenamiento de redes neuronales profundas.”

[44] S. Li *et al.*, “Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting.”

[45] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting.” [Online]. Available: https://github.com/MAZiqing/FEDformer.

Apéndice A

Código Python de la configuración de las arquitecturas xLSTM

En esta sección se presentan las líneas de código utilizadas para definir las diferentes configuraciones de la arquitectura xLSTM. La estructura de estos modelos se comenta en el Capítulo 3, en concreto en el apartado 3.3.1.

```python
# Define the xLSTM configuration
if arquitectura=='XLSTM':
    cfg = xLSTMBlockStackConfig(
        mlstm_block=mLSTMBlockConfig(
            mlstm=mLSTMLayerConfig(
                conv1d_kernel_size=4, qkv_proj_blocksize=2, num_heads=2  # Reduced parameters to save memory
            )
        ),
        slstm_block=sLSTMBlockConfig(
            slstm=sLSTMLayerConfig(
                #backend="cuda",
                backend="vanilla",
                num_heads=2,  # Reduced number of heads to save memory
                conv1d_kernel_size=2,  # Reduced kernel size to save memory
                bias_init="powerlaw_blockdependent",
            ),
            feedforward=FeedForwardConfig(proj_factor=1.1, act_fn="gelu"),  # Reduced projection factor to save memory
        ),
        context_length=seq_length,
        num_blocks=2,  # Reduced number of blocks to save memory
        embedding_dim=embedding_dim,
        slstm_at=[1],
    )
elif arquitectura=='MLSTM':
    cfg = xLSTMBlockStackConfig(
        mlstm_block=mLSTMBlockConfig(
            mlstm=mLSTMLayerConfig(
                conv1d_kernel_size=4, qkv_proj_blocksize=2, num_heads=2  # Reduced parameters to save memory
            )
        ),
        context_length=seq_length,
        num_blocks=1,  # Reduced number of blocks to save memory
        embedding_dim=embedding_dim,
    )
elif arquitectura=='SLSTM':
    cfg = xLSTMBlockStackConfig(
        slstm_block=sLSTMBlockConfig(
            slstm=sLSTMLayerConfig(
                #backend="cuda",
                backend="vanilla",
                num_heads=2,  # Reduced number of heads to save memory
                conv1d_kernel_size=2,  # Reduced kernel size to save memory
                bias_init="powerlaw_blockdependent",
            ),
            feedforward=FeedForwardConfig(proj_factor=1.1, act_fn="gelu"),  # Reduced projection factor to save memory
        ),
        context_length=seq_length,
        num_blocks = 1,
        embedding_dim=embedding_dim,
        slstm_at = [0],
    )
```

Figura 14: Código Python de la configuración de las arquitecturas xLSTM. Parte 1

```python
elif arquitectura=='XLSTM-TS':
    cfg = xLSTMBlockStackConfig(
    mlstm_block=mLSTMBlockConfig(
        mlstm=mLSTMLayerConfig(
            conv1d_kernel_size=4, qkv_proj_blocksize=2, num_heads=2  # Reduced parameters to save memory
        )
    ),
    slstm_block=sLSTMBlockConfig(
        slstm=sLSTMLayerConfig(
            #backend="cuda",
            backend="vanilla",
            num_heads=2,  # Reduced number of heads to save memory
            conv1d_kernel_size=2,  # Reduced kernel size to save memory
            bias_init="powerlaw_blockdependent",
        ),
        feedforward=FeedForwardConfig(proj_factor=1.1, act_fn="gelu"),  # Reduced projection factor to save memory
    ),
    context_length=seq_length,
    num_blocks=4,  # Reduced number of blocks to save memory
    embedding_dim=embedding_dim,
    slstm_at=[1],
)
```

Figura 15: Código Python de la configuración de las arquitecturas xLSTM. Parte 2

Apéndice B

Código Python del entrenamiento y evaluación de las arquitecturas xLSTM

Los modelos xLSTM obtienen resultados diferentes por cada ejecución, por tanto, para que los resultados sean representativos de la eficiencia y rendimiento de cada uno de los cuatro modelos xLSTM, se realizan cinco ejecuciones por cada uno de ellos. La capacidad predictiva se calcula haciendo la media de dichas ejecuciones.

```python
arquitecturas = ['XLSTM-TS', 'XLSTM', 'MLSTM', 'SLSTM']
for arquitectura in arquitecturas:
    print(f"Ejecutando {arquitectura}")
    plot_architecture_xlstm(arquitectura)
    metrics_accumulator[arquitectura] = {}
    for i in range(5):
        results_df, metrics = run_xlstm_ts(train_X, train_y, val_X, val_y, test_X, test_y, scaler, "Pixel "+pixel, 'Pixel', test_dates, arquitectura)
        metrics_accumulator[arquitectura][i] = metrics
```

Figura 16: Código Python del bucle de entrenamiento y evaluación de las arquitecturas xLSTM

```python
final_metrics_accumulator = {}

for arquitectura, ejecuciones in metrics_accumulator.items():
    if all(isinstance(k, int) for k in ejecuciones.keys()):
        df_media = pd.DataFrame.from_dict(ejecuciones, orient="index")
        media = df_media.mean(numeric_only=True).round(4)
        final_metrics_accumulator[arquitectura] = media
    else:
        final_metrics_accumulator[arquitectura] = pd.Series(ejecuciones).round(4)
```

Figura 17: Código Python para cálculo promedio de las métricas de evaluación

Apéndice C

Gráficas de comparación de los resultados de la métrica RMSE de las arquitecturas xLSTM

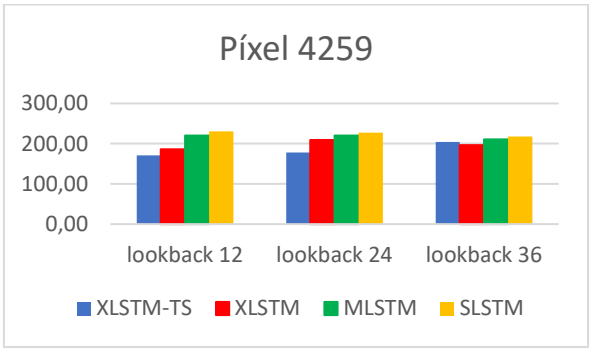

Figura 18: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 4259 organizados por valores de lookback

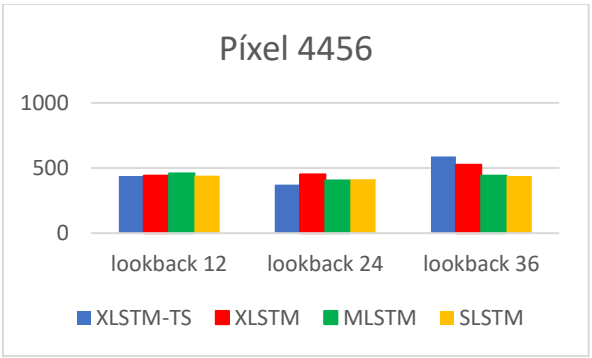

Figura 19: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 4456 organizados por valores de lookback

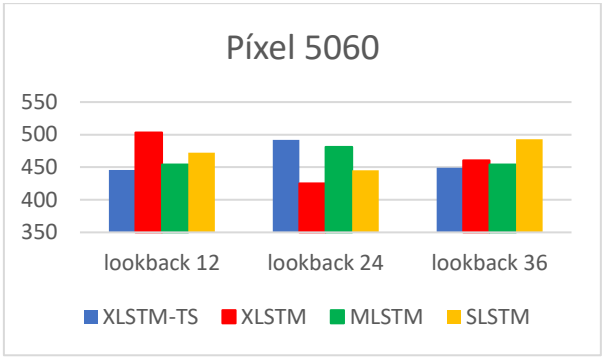

Figura 20: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 5060 organizados por valores de lookback

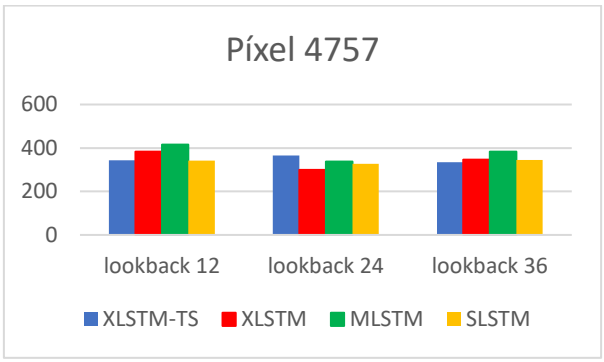

Figura 21: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 4757 organizados por valores de lookback

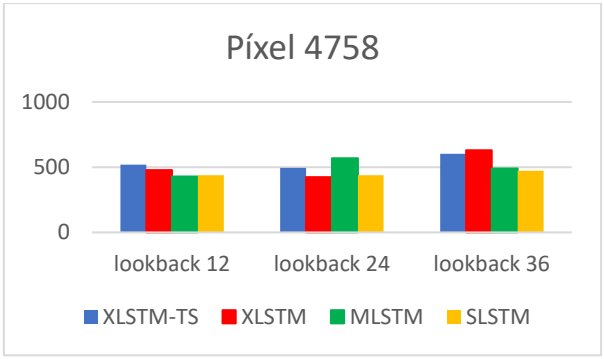

Figura 22: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 4758 organizados por valores de lookback

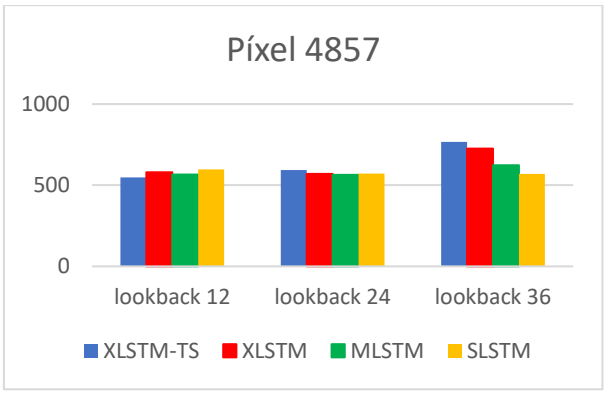

Figura 23: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 4857 organizados por valores de lookback

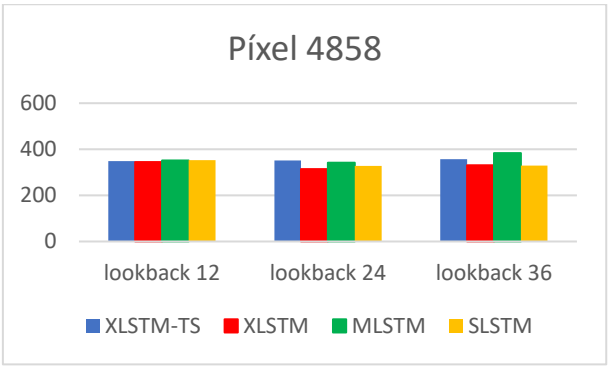

Figura 24: Comparación de los resultados de la métrica RMSE obtenidos por las arquitecturas xLSTM para la serie temporal del píxel 4858 organizados por valores de lookback

Apéndice D

Tablas de resultados de todas las series temporales

En el Capítulo 4 se presenta la tabla con los resultados correspondientes a la serie temporal del píxel 4259, mientras que el resto de las tablas se incluyen en el apéndice.

Modelo	Lookback	MAE	RMSE
TiDE	24	**214.00**	**287.81**
MLP	24	218.06	293.43
TiDE	36	223.58	297.53
MLP	36	222.35	298.87
Passive Agressive Regressor	24	234.40	327.67
MLP	12	237.70	328.54
LSTM Multi. Online Inc.	24	251.3	330.74
Passive Agressive Regressor	36	241.94	332.67
TiDE	12	246.46	346.36
LSTM Ind. Online Inc.	24	260.37	346.69
N-BEATS	24	266.39	353.37
Linear Regression	24	288.93	356.29
N-BEATS	36	275.60	363.24
XLSTM-TS	24	281.67	371.95
N-HiTS	24	303.61	399.00
N-HiTS	36	304.27	399.69
Linear Regression	36	332.81	402.94
MLSTM	24	320.93	404.18
N-BEATS	12	298.45	409.85
SLSTM	24	336.80	414.45
TCN	12	333.96	420.78
Naive	NA	311.59	432.53
SLSTM	36	348.29	437.81
XLSTM-TS	12	333.89	439.24
SLSTM	12	345.38	440.41
MLSTM	36	343.38	440.56
XLSTM	12	327.74	441.84
XLSTM	24	340.06	450.20
DeepTCN	24	369.70	454.77
MLSTM	12	345.36	458.17
LSTM Ind. Online	24	355.43	464.43
LSTM Multi.	24	353.65	475.92
Passive Agressive Regressor	12	369.30	490.21
N-HiTS	12	364.32	505.35
LSTM Ind.	24	348.48	524.44
XLSTM	36	401.80	524.66
TFT	24	393.42	530.58
LSTM Multi. Online	24	418.9	546.02
TSMixer	36	430.21	558.16
TSMixer	12	433.88	560.23
TFT	36	421.62	562.88
XLSTM-TS	36	430.14	589.09
TSMixer	24	445.93	593.27
TCN	36	498.97	613.27
TFT	12	452.70	630.01
Linear Regression	12	539.03	642.98
TCN	24	520.52	644.34
DeepTCN	12	699.73	824.47
DeepTCN	36	741.84	950.50

Tabla 5: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 4456

Modelo	Lookback	MAE	RMSE
TiDE	24	**292.86**	**412.18**
XLSTM	24	309.18	424.90
TiDE	12	311.17	432.04
SLSTM	24	315.77	445.14
XLSTM-TS	12	322.32	445.89
XLSTM-TS	36	336.87	449.32
MLSTM	36	334.08	454.07
TiDE	36	330.27	456.77
XLSTM	36	341.30	460.54
N-BEATS	36	327.97	461.46
LSTM Ind. Online Inc.	24	330.51	461.79
LSTM Multi. Online Inc.	24	329.88	471.00
N-BEATS	24	332.45	471.19
SLSTM	12	334.36	472.31
LSTM Ind.	24	341.84	479.24
MLSTM	24	342.48	481.10
Linear Regression	36	317.60	482.88
MLP	24	312.84	487.16
XLSTM-TS	24	353.27	491.87
SLTM	36	360.03	492.85
Linear Regression	24	324.31	493.18
MLSTM	12	374.63	501.69
XLSTM	12	389.7	503.12
LSTM Ind. Online	24	365.95	512.7
N-BEATS	12	368.40	513.18
DeepTCN	24	376.67	516.25
MLP	36	353.71	522.11
MLP	12	309.76	523.44
Linear Regression	12	386.69	549.33
LSTM Multi.	24	409	553.59
TCN	12	410.57	556.29
TCN	36	419.63	556.97
TCN	24	461.43	583.78
N-HiTS	36	438.91	597.52
Passive Agressive Regressor	36	443.99	601.72
LSTM Multi. Online	24	440.3	607.28
N-HiTS	24	458.19	635.13
Passive Agressive Regressor	24	485.59	657.27
N-HiTS	12	494.55	664.89
DeepTCN	36	539.52	712.51
Naive	NA	572,00	780.65
Passive Agressive Regressor	12	672.59	944.26
TFT	36	723.19	1011.04
TSMixer	12	796.29	1069.14
TFT	12	798.06	1074.05
TFT	24	772.45	1092.57
TSMixer	36	825.32	1159.65
DeepTCN	12	907.84	1166.79
TSMixer	24	879.46	1261.31

Tabla 6: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 5060

Modelo	Lookback	MAE	RMSE
TiDE	24	**208.69**	**279.58**
TiDE	36	217.33	285.53
MLP	12	214.49	293.33
MLP	36	221.27	294.19
LSTM Multi. Online Inc.	24	221.68	297.59
XLSTM	24	223.07	299.38
N-BEATS	36	233.86	303.89
LSTM Multi.	24	233.12	305.79
MLP	24	240.45	310.12
Linear Regression	24	247.44	314.00
Passive Agressive Regressor	24	235.87	316.22
Linear Regression	36	254.06	318.77
Passive Agressive Regressor	36	238.82	320.49
TiDE	12	230.41	320.57
SLSTM	24	248.09	327.82
LSTM Multi. Online	24	249.64	330.09
N-BEATS	24	249.12	332.08
XLSTM-TS	36	253.21	334.23
MSLTM	24	259.33	337.27
SLSTM	12	257.67	341.43
XLSTM-TS	12	261.72	342.78
SLSTM	36	262.02	344.66
XLSTM	36	265.42	346.46
LSTM Ind. Online Inc.	24	330.51	361.79
XLSTM-TS	24	279.07	366.09
LSTM Ind.	24	284.81	367.4
N-HiTS	36	297.62	382.99
XLSTM	12	284.27	383.75
MLSTM	36	304.89	383.88
DeepTCN	24	317.74	389.5
TCN	12	306.11	398.14
TCN	36	334.03	410.03
N-HiTS	24	308.78	411.61
N-BEATS	12	313.85	412.01
MLSTM	12	312.46	415.36
Naive	NA	306.56	420.28
TCN	24	343.61	423.38
Linear Regression	12	357.63	433.28
Passive Agressive Regressor	12	360.01	473.03
N-HiTS	12	363.69	485.93
LSTM Ind. Online	24	398.9	549.14
TSMixer	12	470.39	629.4
DeepTCN	36	525.52	652.48
TFT	36	488.52	655.5
TSMixer	36	497.99	678.4
TFT	12	509.19	695.68
TFT	24	531.22	724.55
TSMixer	24	531.03	736.44
DeepTCN	12	705.76	806.78

Tabla 7: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 4757

Modelo	Lookback	MAE	RMSE
TiDE	24	**283.67**	**385.76**
MLP	12	284.10	386.76
MLP	36	289.76	387.46
TiDE	36	292.64	388.84
MLP	24	287.44	395.55
Linear Regression	36	315.29	398.17
LSTM Multi. Online Inc.	24	298.38	403.1
Linear Regression	24	322.25	414.49
TiDE	12	303.49	415.06
N-BEATS	24	311.93	419.66
XLSTM	24	327.62	424.1
MLSTM	12	333.38	429.33
N-BEATS	36	330.44	437.4
SLSTM	24	342.98	437.98
SLSTM	12	343.04	439.28
LSTM Multi.	24	352.02	446.41
Passive Agressive Regressor	24	348.22	457.26
Passive Agressive Regressor	36	354.68	461.18
SLSTM	36	376.69	473.19
XLSTM	12	371.86	476.93
Linear Regression	12	389.48	486.88
MLSTM	36	373.77	490.68
XLSTM-TS	24	372.95	495.06
LSTM Ind. Online Inc.	24	370.45	500.5
N-BEATS	12	366.45	503.31
TCN	12	366.07	504.57
LSTM Multi. Online	24	345.63	505.03
XLSTM-TS	12	389.9	518.36
DeepTCN	24	416.79	519.12
N-HiTS	36	392.76	519.2
TCN	36	405.32	521.29
Naive	NA	360.07	534.29
LSTM Ind.	24	403.3	534.82
LSTM Ind. Online	24	412.49	544.38
TCN	24	435.53	547.83
MLSTM	24	437.38	568.24
N-HiTS	24	419.51	572.51
XLSTM-TS	36	450.02	601.85
N-HiTS	12	436.44	604.64
XLSTM	36	456.43	629.13
DeepTCN	36	606.67	727.58
Passive Agressive Regressor	12	552.07	761.58
TFT	12	581.1	785.6
TSMixer	12	590.7	785.67
TFT	36	639.36	862.01
TFT	24	633.0	882.36
TSMixer	24	637.52	891.36
TSMixer	36	655.11	904.56
DeepTCN	12	775.44	960.97

Tabla 8: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 4758

Modelo	Lookback	MAE	RMSE
TiDE	24	**321.25**	**436.49**
MLP	24	344.99	451.84
MLP	12	345.19	469.94
MLP	36	350.70	476.57
TiDE	36	353.08	477.43
LSTM Ind.	24	366.86	486.49
N-BEATS	24	365.24	492.82
N-BEATS	36	358.98	494.77
LSTM Multi. Online Inc.	24	365.26	505.21
TiDE	12	359.17	507.49
Linear Regression	36	410.22	514.78
Passive Agressive Regressor	36	381.00	519.67
Linear Regression	24	419.04	526.13
Passive Agressive Regressor	24	374.02	526.59
LSTM Ind. Online Inc.	24	396.58	531.88
XLSTM-TS	12	393.81	547.24
MLSTM	24	445.35	563.74
MLSTM	12	435.72	566.41
XLSTM	24	433.81	568.95
SLSTM	36	432.32	569.86
SLSTM	24	442.11	570.82
N-BEATS	12	429.73	574.51
XLSTM	12	425.11	580.44
LSTM Multi.	24	443.62	580.74
XLSTM-TS	24	460.02	593.67
SLSTM	12	471.85	597.95
N-HiTS	24	455.12	605.24
LSTM Multi. Online	24	471.71	621.16
MLSTM	36	484.6	623.68
N-HiTS	36	477.57	629.59
TCN	12	512.96	662.24
DeepTCN	24	542.35	685.17
LSTM Ind. OLINE	24	504.72	693.73
Naive	NA	510.16	712.75
XLSTM	36	538.84	726.1
N-HiTS	12	558.77	761.81
Linear Regression	12	635.22	763.21
XLSTM-TS	36	532.43	766.68
TCN	36	664.55	825.85
TCN	24	678.93	839.48
Passive Agressive Regressor	12	605.77	864.42
TFT	24	743.77	1019.63
TFT	36	776.03	1040.65
DeepTCN	36	862.52	1094.1
TSMixer	12	844.54	1131.36
TSMixer	36	841.33	1133.22
TFT	12	838.85	1155.82
TSMixer	24	847.42	1173.54
DeepTCN	12	1146.85	1314.38

Tabla 9: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 4857

Modelo	Lookback	MAE	RMSE
MLP	24	**204.73**	**280.56**
MLP	12	212.42	286.04
TiDE	24	207.58	286.06
LSTM Multi. Online	24	215.58	294.17
TiDE	36	224.59	296.18
LSTM Multi. Online Inc.	24	217.39	297.17
LSTM Multi.	24	225.08	298.7
MLP	36	230.86	305.53
XLSTM	24	233.57	313.83
Linear Regression	36	250.92	320.09
TiDE	12	230.57	320.97
Linear Regression	24	250.63	326.90
SLSTM	24	243.28	327.81
SLSTM	36	245.41	329.84
XLSTM	36	253.08	330.13
MLSTM	24	260.67	341.57
XLSTM	12	260.41	344.74
XLSTM-TS	12	265.29	349.13
Passive Agressive Regressor	24	257.70	349.54
XLSTM-TS	24	259.25	351.66
MLSTM	12	264.55	351.94
SLSTM	12	264.08	352.55
N-BEATS	36	260.85	354.39
N-BEATS	24	259.31	356.81
XLSTM-TS	36	270.52	357.7
Passive Agressive Regressor	36	270.57	358.30
LSTM Ind.	24	262.1	372.81
LSTM Ind. Online Inc.	24	278.28	376.22
MLSTM	36	295.71	383.1
N-HiTS	36	296.37	392.52
DeepTCN	24	312.51	401.54
TCN	12	300.68	409.54
Linear Regression	12	325.95	410.23
TCN	24	340.45	417.0
TCN	36	338.91	420.22
LSTM Ind. Online	24	319.21	421.38
N-BEATS	12	314.69	435.99
N-HiTS	24	318.72	443.01
Naive	NA	312.79	448.85
N-HiTS	12	378.95	520.02
Passive Agressive Regressor	12	412.93	555.31
DeepTCN	36	513.25	614.28
TFT	36	478.73	640.53
TFT	12	471.01	645.57
TSMixer	12	501.02	678.4
TSMixer	36	542.24	742.02
TFT	24	563.7	796.23
TSMixer	24	568.35	817.65
DeepTCN	12	709.09	864.17

Tabla 10: Resultados de las métricas MAE y RMSE obtenidos por cada arquitectura en la serie temporal del píxel 4858

Apéndice E

Gráficas de comparación de los resultados de la métrica RMSE de todas las arquitecturas

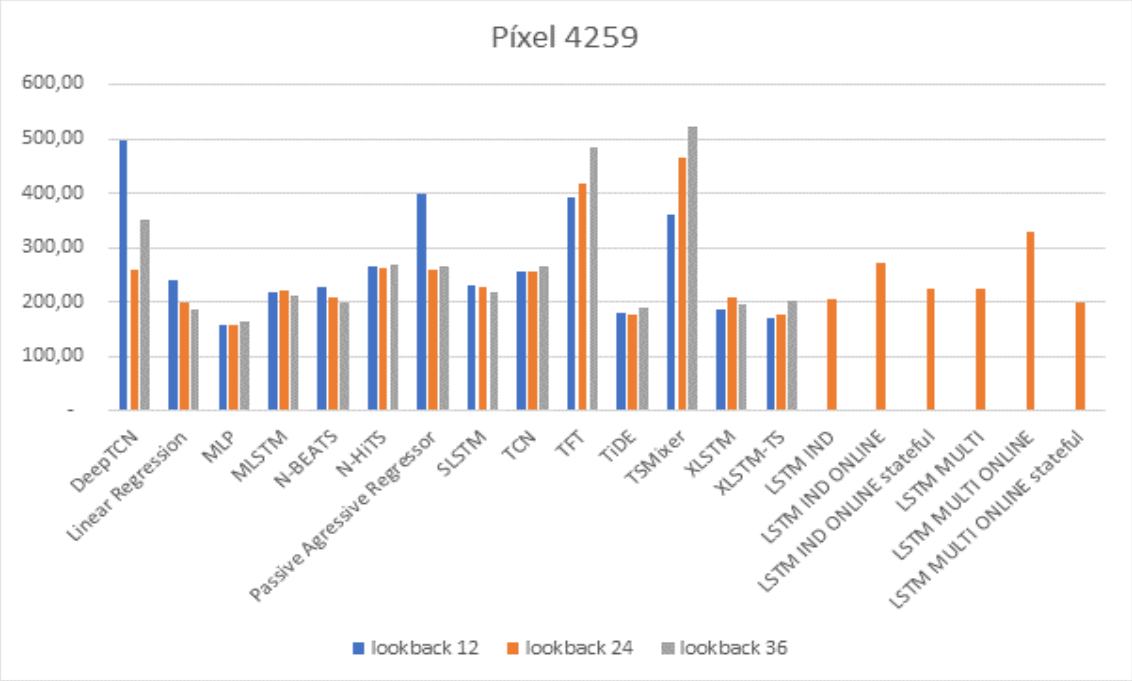

Figura 25: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 4259 para cada valor de lookback organizados por arquitectura

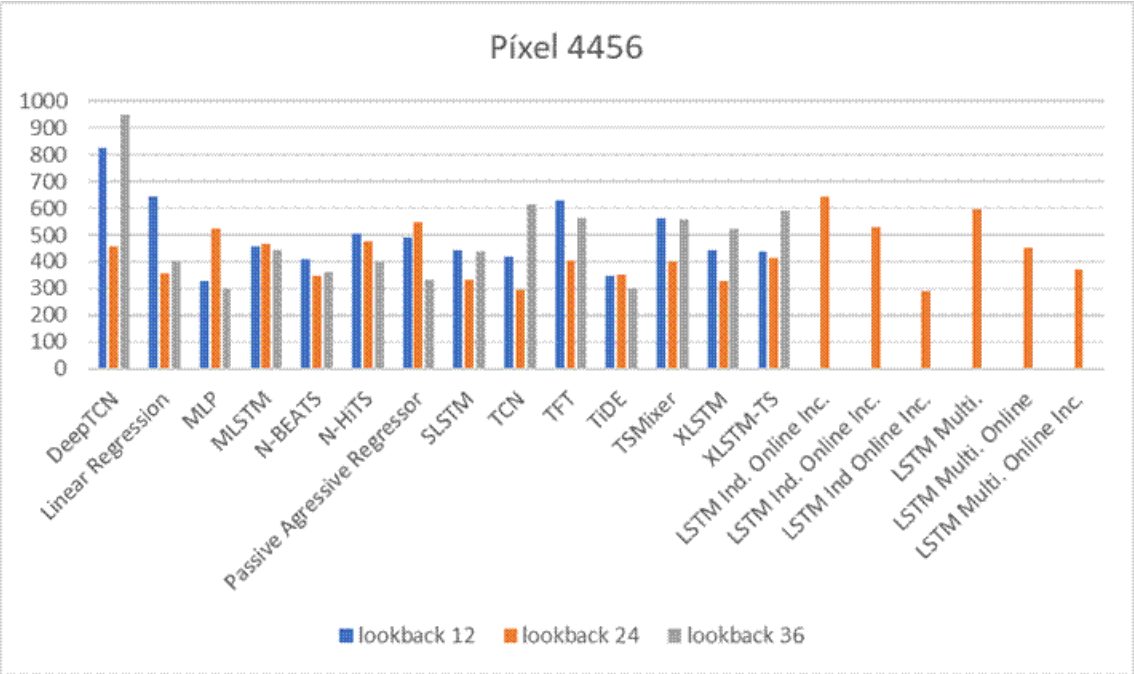

Figura 26: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 4456 para cada valor de lookback organizados por arquitectura

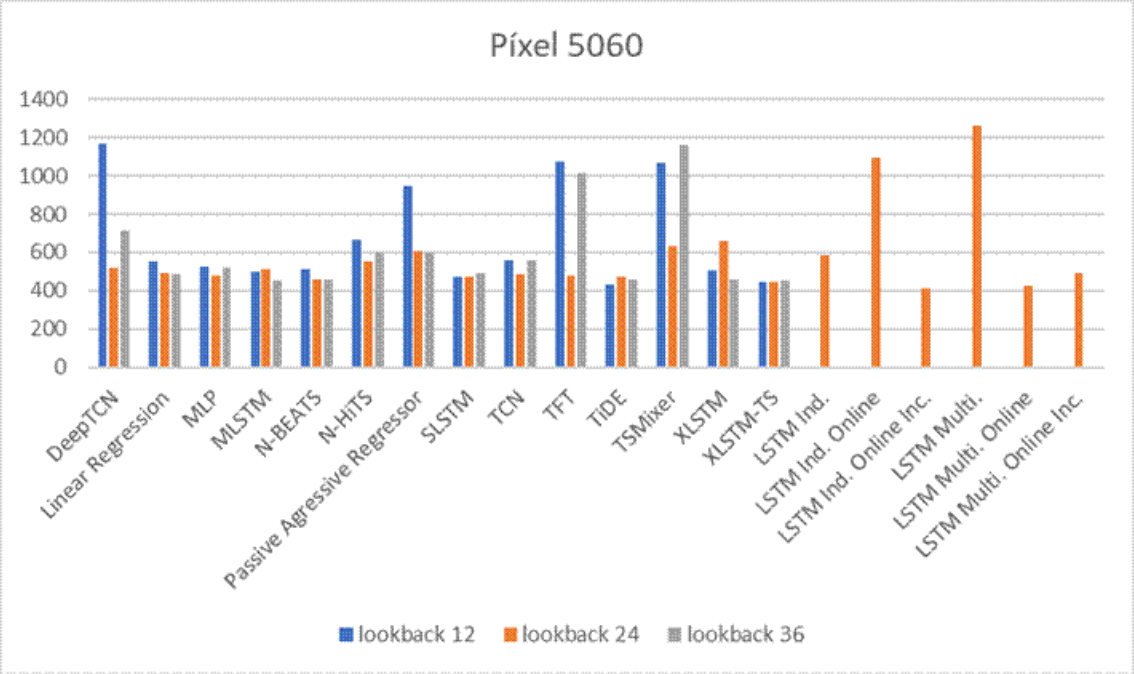

Figura 27: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 5060 para cada valor de lookback organizados por arquitectura

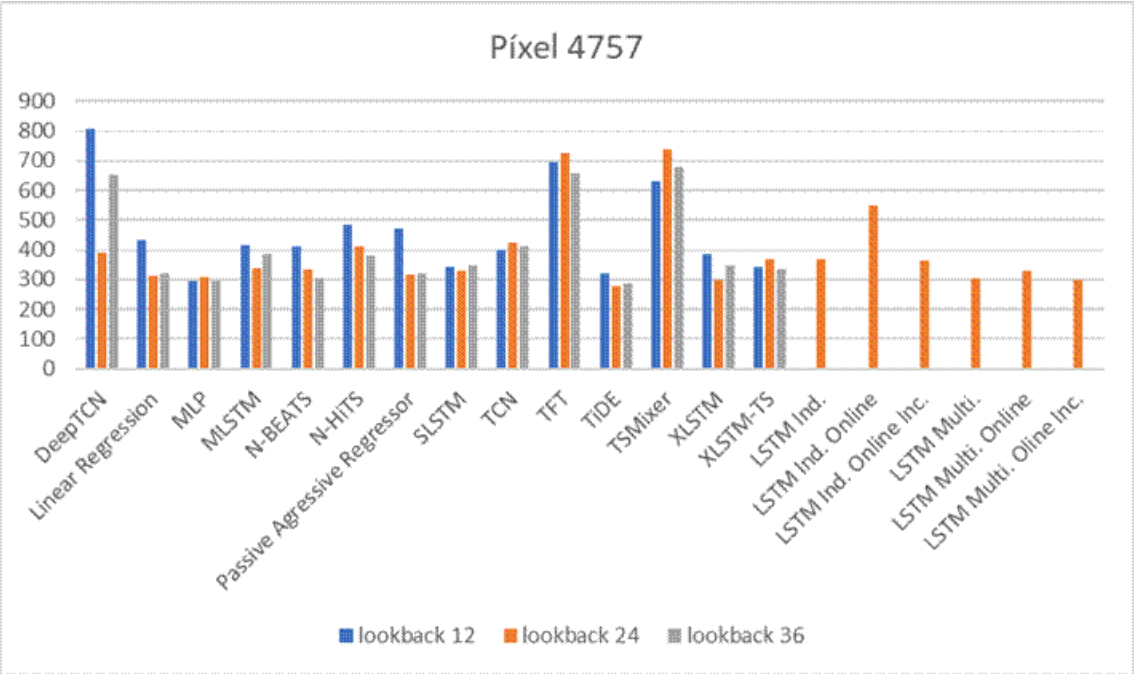

Figura 28: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 4757 para cada valor de lookback organizados por arquitectura

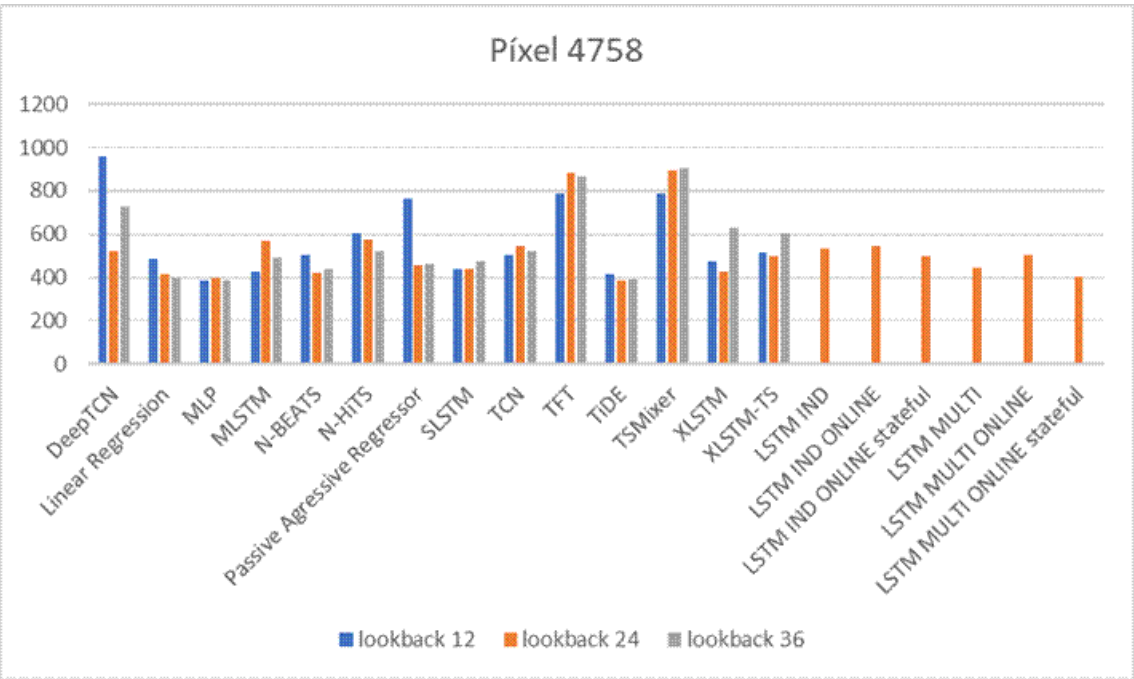

Figura 29: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 4758 para cada valor de lookback organizados por arquitectura

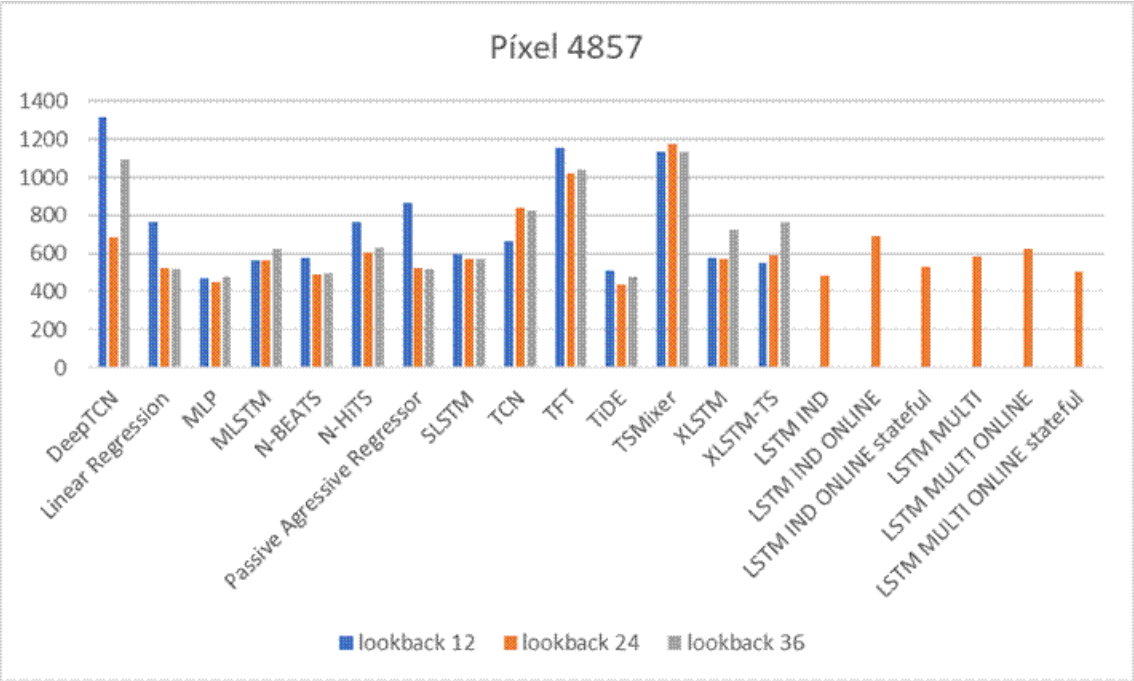

Figura 30: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 4857 para cada valor de lookback organizados por arquitectura

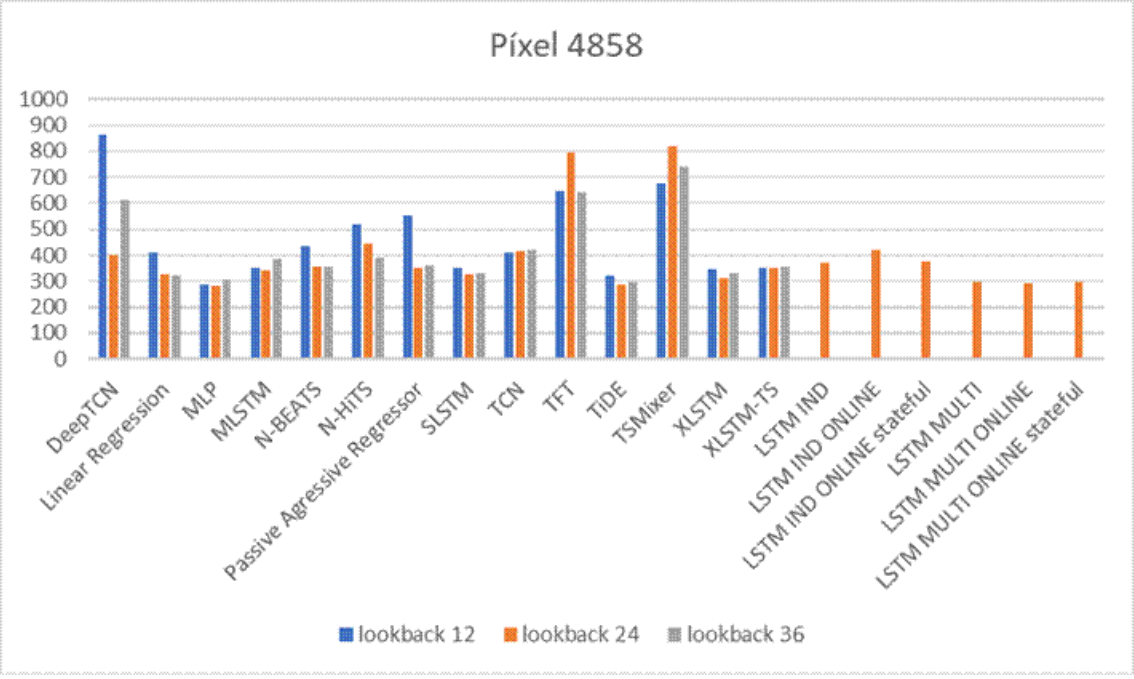

Figura 31: Comparación de los resultados de la métrica RMSE obtenidos en la serie temporal del píxel 4858 para cada valor de lookback organizados por arquitectura