UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingenierías Informática y Telecomunicación

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

TRABAJO DE FIN DE MÁSTER

Estudio sobre la Efectividad del Positional Encoding en Transformers para Series Temporales y Diseño de Mecanismos Adaptados

Presentado por:
Cristhian Moya Mota

Curso académico 2024-2025

Estudio sobre la Efectividad del Positional Encoding en Transformers para Series Temporales y Diseño de Mecanismos Adaptados

Cristhian Moya Mota

Cristhian Moya Mota *Estudio sobre la Efectividad del Positional Encoding en Transformers para Series Temporales y Diseño de Mecanismos Adaptados*.

Trabajo de fin de Grado. Curso académico 2024-2025.

Responsable de tutorización

Julian Luengo Martín
DECSAI

Diego Jesús García Gil
LSI

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

Escuela Técnica Superior de Ingenierías Informática y Telecomunicación

Universidad de Granada

Declaración de originalidad

D./Dña. Cristhian Moya Mota

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2024-2025, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 6 de agosto de 2025

Fdo: Cristhian Moya Mota

Dedicatoria (opcional)

Agradecimientos

Agradecimientos (opcional, ver archivo `preliminares/agradecimiento.tex`).

Summary

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.tex`

Índice general

1 Introducción

En este primer punto, describiremos de forma breve la motivación a realizar este TFM. Analizaremos la importancia de poseer herramientas capaces de realizar una predicción a largo plazo en series temporales (en inglés, *Long-term Time Series Forecasting*, LSTF), pero, sobre todo, centrándonos en un aspecto clave: la codificación posicional (*Positional Encoding*, en adelante, PE), común a prácticamente todas las alternativas actuales del estado del arte.

Procederemos a justificar su importancia, así como a formular los objetivos que se cubren con la realización de este trabajo.

1.1. Motivación

En la actualidad, los datos son uno de los bienes más preciados. Las telecomunicaciones nos han permitido alcanzar un volumen inimaginable de información digital, la cual no somos prácticamente capaces de procesar, y extraer conocimiento útil se convierte en una tarea complicada. Su gran variedad y modalidad hace necesario disponer de modelos multimodales cada vez más complejos para procesarlos, como los modelos fundacionales [1], para tratar así de disponer de una herramienta cercana a ser capaz de procesar todo tipo de información.

Hemos podido apreciar grandes avances en el procesamiento de texto, con los grandes modelos de lenguaje como GPT, el cual se ha convertido en una herramienta que usamos habitualmente para resolver nuestras dudas. Ahora, incluso permite generar imágenes, video y audio, y pensar profundamente las respuestas.

Sin embargo, estos no son los únicos tipos de datos que podemos emplear para aprender. Los datos basados en flujos, y las series temporales, son un recurso clave que podemos emplear para resolver multitud de problemáticas. Podemos predecir a largo plazo, clasificar fenómenos, o bien, incluso detectar anomalías. En este trabajo, nos centraremos sobre todo en la primera tarea y las dificultades que existen en este ámbito.

1.1.1. Importancia de las Series Temporales

Las series temporales son un tipo de datos caracterizados por una secuencia organizada en el tiempo. Generalmente, consisten en una o varias variables observadas a intervalos regulares, con el objetivo de registrar la evolución de un fenómeno. Suelen recogerse de forma periódica mediante sensores o procesos automatizados (por ejemplo, la temperatura de un motor medida por un termostato electrónico), aunque también pueden originarse de forma irregular, como en el caso de registros manuales. Es importante que, cuando se trata de múltiples variables, estas se muestreen con la misma frecuencia para facilitar su análisis y evitar la necesidad de técnicas de imputación, que podrían introducir distorsiones en los resultados.

El objetivo: utilizar estos datos para aprender qué ocurrirá en instantes futuros, realizando predicciones en un horizonte concreto sobre el que es desconocido su comportamiento. Sin embargo, es importante destacar que no todos los fenómenos pueden ser predichos aunque dispongamos de un conjunto de datos suficiente y adecuado. Es esencial que, para que la técnica sea efectiva, estemos ante una tarea de forecasting, es decir, de predicción de elementos futuros, pero pudiéndonos apoyar en información pasada para comprender el fenómeno que estamos estudiando. Por ejemplo, tratar de predecir la temperatura que hará en las próximas 2 horas, en intervalos de 15 minutos,es una tarea viable: disponemos de gran cantidad de recursos históricos, como la tendencia de las horas anteriores, y los valores históricos de días previos, o incluso, el comportamiento en años anteriores, que pueden ser útiles.

Pero también podríamos enfrentarnos a escenarios más complejos, como el que gestiona diariamente Red Eléctrica para estimar el consumo de electricidad y responder con los tipos de energía más adecuados, evitando tanto el exceso como el déficit en la red. Para ello, se apoyan no solo en datos históricos de consumo, sino también en la información en tiempo real proveniente de sensores y mediciones realizadas en distintas estaciones eléctricas, lo que permite anticipar fluctuaciones y minimizar problemas de sincronización. A esto se suman otros factores externos que pueden influir, como las condiciones meteorológicas o incidencias en redes interconectadas de países vecinos. Tras la experiencia vivida en abril de 2025, queda en evidencia la importancia de esta tarea, ya que en caso de error, las consecuencias pueden ser muy graves: personas atrapadas en ascensores, hospitales en emergencia, problemas de tráfico, interrupción de la cadena de frío en los alimentos... Las pérdidas económicas se estimaron entre 1.600 y 2.500 millones de euros.

Por tanto, disponer de buenas técnicas de estudio para series temporales es una herramienta clave en multitud de aplicaciones. Manualmente, podemos realizar aproximaciones posibles para casos sencillos, donde apreciamos un comportamiento repetitivo en el tiempo de la serie (estacionalidad) o un comportamiento estrictamente creciente o decreciente linealmente (tendencia). Sin embargo, cuando el problema se vuelve más complejo o involucra múltiples variables, recurrir a matemáticas simples o técnicas intuitivas ya no es suficiente. En estos casos, es necesario contar con herramientas más sofisticadas que permitan modelar el comportamiento de la serie de forma aproximada, pero lo más fiel posible a la realidad.

Es aquí donde entran en juego métodos ampliamente utilizados, como la descomposición de la serie en componentes fundamentales: tendencia, estacionalidad y ruido. Una de las técnicas estadísticas más reconocidas en este ámbito es ARIMA (AutoRegressive Integrated Moving Average) [2]. Se basa en utilizar una ventana deslizable sobre la que ir calculando una media móvil, es decir, un intervalo de amplitud q en torno a un valor central, el cual se va moviendo de izquierda a derecha para evaluar con todas las instancias de la serie, y en una componente autoregresiva, que trata de incorporar información de instantes anteriores al que se está evaluando (hasta p valores). También se integra la serie tantas veces como sea necesario para suavizar su comportamiento. De este modo, es posible generar predicciones razonables, siempre y cuando el problema cumpla con ciertas condiciones que abordaremos más adelante.

No obstante, incluso técnicas como ARIMA presentan limitaciones importantes, especialmente cuando se enfrentan a horizontes de predicción amplios. A medida que intentamos anticipar valores más alejados en el tiempo, el modelo se ve obligado a basarse cada vez más

en sus propias predicciones previas en lugar de los datos observados, lo que provoca una acumulación progresiva de errores.

Esto plantea la necesidad de modelos más robustos, capaces de capturar dependencias de largo alcance y manejar múltiples variables de forma conjunta, tratando de minimizar el error evitando su acumulación. Podemos recurrir, como alternativa, a una de las herramientas ya empleadas en los grandes modelos de lenguaje que mencionamos anteriormente: los Transformers [17]. Pero su uso no es directo; debemos de adaptar su funcionamiento a las series temporales, ya que estos modelos, originalmente diseñados para tareas de lenguaje natural, no conservan de manera adecuada la información temporal del orden de los datos.

1.1.2. La dificultad de predicción a largo plazo: información posicional

Para poder adaptar modelos basados en Transformers para lenguaje natural a series temporales, es imprescindible introducir mecanismos de codificación posicional (positional encoding) que permitan al modelo interpretar correctamente la secuencia en el tiempo [10]. Sin este componente, los Transformers serían incapaces de distinguir el orden de la entrada, lo cual es esencial para predecir correctamente la evolución de un fenómeno.

Encontrar una codificación eficiente, sencilla y coherente con la estructura de los datos se encuentra en continua búsqueda en el estado del arte, pero la mayoría de ellas presentan los siguientes inconvenientes:

- **Falta de captura de la estructura**. En muchas ocasiones, se opta por utilizar codificaciones sencillas basadas en senos y cosenos, de la misma forma que se hace en procesamiento de lenguaje, pero de esta forma, estamos perdiendo información de patrones estacionales e información local de interés. Esto ocurre, por ejemplo, en modelos como Informer [18], uno de los primeros en adaptar los Transformers a series temporales.

- **Dificultad para adaptarse a diferentes escalas temporales y falta de semántica**. Propuestas simples, como la ya mencionada codificación sinusoidal, pueden ser ineficientes cuando se busca trabajar con distintas granularidades temporales. Al tratarse de una codificación fija, sus valores no se ajustan al cambiar la frecuencia de muestreo (por ejemplo, de segundos a minutos) , ya que mantienen una amplitud y periodicidad predefinidas. Por otro lado, aproximaciones más complejas, como las basadas en convoluciones, intentan capturar patrones locales dentro de la secuencia, pero tienden a centrarse en un corto plazo, lo que dificulta la detección de relaciones de largo alcance o multiescala. Además, este tipo de codificación no incorpora explícitamente la semántica temporal (como la hora del día o el día de la semana), lo que limita su interpretabilidad y generalización en tareas donde esa información es relevante.

- **Complejidad**. Para paliar las dificultades de los métodos más simples, se han evaluado alternativas más complejas, que permiten aumentar el rendimiento: es el caso de Autoformer, que incorpora información autoregresiva al modelo, tomando así cierta similaridad con la componente AR de ARIMA; o bien, se proponen encodings basados en transformadas de Fourier. Pero estas aumentan la necesidad de cómputo y el tiempo necesario para su entrenamiento, lo cual hace surgir otra vertiente alternativa, basada en modelos más sencillos, como Reformer [12], que trata de convertir la

eficiencia cuadrática de los Transformers, a lineal. O Informer, que persigue lograr eficiencia $n \log n$ mediante su ProbSparse Attention [18]. Pero, a veces puede conseguirse el efecto contrario: empeorar en exceso los resultados, a consta de un menor tiempo de entrenamiento e inferencia. Por ejemplo, en Informer, al muestrearse en atención solo ciertos vectores de entrada, perdemos información local que puede ser clave.

La misión de este trabajo se centra en tratar de encontrar una alternativa que sea capaz de mantenerse cercana a la semántica del problema, permitiendo adaptabilidad en cuanto a parámetros, y tratando de alcanzar un equilibrio entre eficiencia y rendimiento.

1.2. Justificación

Tras estudiar el problema, surge la necesidad de crear una alternativa adecuada a los modelos de codificación existentes, incorporando dentro de ella la semántica del propio problema, e información local que permita una mayor adaptabilidad de los modelos creados, reduciendo las tasas de error. Aunque este objetivo ya es tratado por diversos modelos y líneas de investigación del estado del arte, en este trabajo buscamos replantear el positional encoding para que sea adaptable y capaz de incorporar información global y local, dando mayor importancia a una u otra en función de la entrada dada.

Todo esto se plantea aprovechando el conocimiento adquirido a partir de técnicas más clásicas, como ARIMA, e incorporando información estadística que resulte comprensible para prácticamente cualquier persona que utilice el modelo. De este modo, aunque se recurra posteriormente a modelos profundos como los Transformers, la codificación posicional introducida al inicio en los datos puede ser ajustable y fácilmente interpretable en función del contexto del problema.

Hasta ahora, gran parte de los esfuerzos se han centrado en adaptar soluciones desarrolladas originalmente para otros dominios, principalmente el procesamiento de lenguaje natural, al contexto de las series temporales. Sin embargo, estas adaptaciones no siempre se alinean completamente con las particularidades de los datos temporales, lo que puede derivar en pérdidas de información relevantes o en un rendimiento subóptimo. Esta desconexión pone de relieve la necesidad de desarrollar codificaciones específicamente diseñadas para capturar la estructura temporal propia de este tipo de datos. En este documento, abordaremos esta problemática desde una perspectiva crítica, comparando distintos métodos existentes y proponiendo nuevas soluciones, algunas de ellas híbridas, que buscan mejorar la representación temporal en modelos basados en Transformers.

1.3. Objetivos

Teniendo en cuenta los conceptos descritos anteriormente, y analizando el estado del arte en la actualidad, vemos justificada la necesidad de encontrar nuevas codificaciones posicionales con el objetivo de conseguir:

1. Realizar una revisión exhaustiva del estado del arte sobre positional encoding y su aplicación en modelos Transformer para series temporales, estudiando el funcionamiento

de las diferentes técnicas y extrayendo el conocimiento útil para crear nuevas alternativas.

2. Analizar las propuestas originales de positional encoding adaptadas a series temporales, realizando un análisis crítico y explorando las diferentes vías alternativas.
3. Evaluar sistemáticamente la efectividad de los positional encoding estándar en tareas de forecasting sobre benchmarks de series temporales, haciendo uso de varios conjuntos de datos de diferente procedencia y estructura.
4. Establecer criterios claros para el diseño de nuevas codificaciones que permitan encontrar nuevas formas de incorporar información útil, pero además sirvan como guía para investigaciones futuras, identificando buenas prácticas, limitaciones comunes y condiciones bajo las cuales cada enfoque resulta más adecuado.
5. Comparar el impacto de las distintas codificaciones posicionales no solo en la precisión de las predicciones, sino también en aspectos como la capacidad de generalización y su eficiencia computacional.

En este documento, se recoge el estudio del estado del arte actual, realizando un estudio crítico y comparativo de las diferentes propuestas, haciendo también hincapié en el propio funcionamiento de los Transformer y como afecta a la codificación y procesado de la información. Se buscará, como ya se ha mencionado, crear un resumen con diferentes técnicas, nuevas y otras conocidas, que por separado o en su conjunto, de forma híbrida, propongan soluciones adecuadas a diferentes conjuntos de datos provenientes del mundo real.

Adicionalmente, se proporciona un repositorio en GitHub[1] que contiene todo el código desarrollado, así como las referencias a los conjuntos de datos utilizados. Este recurso se mantiene accesible con el objetivo de facilitar su análisis, reutilización y posible mejora en trabajos futuros.

[1] https://github.com/hexecoded/TFM

2 Tendencias y Estado del arte

Antes de adentrarnos de lleno en el análisis del problema y la realización de nuevas metodologías, es necesario estudiar y comprender el estado actual de esta temática en el estado del arte. Debido a que no se ha visto tan explotada como otros ámbitos de la IA, como es el caso de los modelos convolucionales o el aprendizaje automático supervisado tabular, podemos encontrar continuos cambios y nuevas vertientes que pueden inspirarnos a la hora de abordar el problema.

Hasta hace una década, buena parte de las técnicas diseñadas especialmente para series temporales y predicción a largo plazo, se basan principalmente en la descomposición las mismas en subcomponentes mucho más sencillas de procesar, las cuales pueden seguir un enfoque similar a divide y vencerás: descomponer la red en diferentes elementos, y mediante un mecanismo de agregación (aditivo, multiplicativo, o estadístico), recomponer la solución a la tarea. Sin embargo, se trata de una estrategia últimamente menos utilizada, en decadencia a favor de nuevas técnicas

Actualmente, una de las principales tendencias consiste en adaptar modelos originalmente desarrollados para otras modalidades. Un ejemplo de ello es el uso de convoluciones, ampliamente utilizadas en visión por computador, con el objetivo de capturar patrones locales en las secuencias y reducir la dimensionalidad del modelo. Previamente, también se han explorado arquitecturas recurrentes, como las Recurrent Neural Networks (RNN) y sus variantes más avanzadas, como las Long Short-Term Memory (LSTM) [7], aunque estas presentaban ciertas limitaciones en cuanto a capacidad de paralelización y en la modelización de relaciones temporales de mayor alcance, ya que para lograrlo aumentaba en exceso el tamaño de la red para incorporar más conexiones.

Más recientemente, ha ganado protagonismo la adaptación de mecanismos provenientes del procesamiento de lenguaje natural, en particular los Transformers, debido a su capacidad para modelar dependencias a largo plazo de forma eficiente. Además, comparten una motivación estructural con las series temporales: la necesidad de preservar una secuencia ordenada y coherente en la entrada, lo que permite aprovechar su arquitectura basada en atención para tareas de predicción.

A continuación, nos adentraremos de lleno en dichas propuestas, con el fin de aclarar, en primer, los conceptos clave acerca de las series temporales, y además, entender las problemáticas que surgen durante su procesado.

2.1. Procesamiento básico de Series Temporales

En la introducción, se han presentado brevemente las características fundamentales de las series temporales: su estructura temporal, caracterizada por muestreo, la importancia del orden de medición, y la necesidad de mantener las relaciones temporales para lograr aprender

de forma efectiva.

Pero, en esta sección, profundizaremos en los aspectos clave del preprocesamiento necesario para su análisis de manera más formal, ya que a diferencia de otros tipos de datos, las series temporales requieren una atención especial a la dimensión temporal, y cualquier alteración en su estructura puede afectar directamente la capacidad del modelo para aprender los patrones subyacentes en ella. Por ello, es fundamental centrarnos en cuestiones como la frecuencia de muestreo, la consistencia temporal, el tratamiento de valores faltantes, y la normalización de las variables.

2.1.1. Modelos basados en descomposición

Las series temporales pueden exhibir una gran cantidad de patrones y comportamientos, los cuales son interesantes de estudiar y visualizar con claridad para estudiar que posible enfoque seguir a la hora de resolver el problema. Diferenciando cada una de las partes, podemos tratar de resolver cada una de ellas por separado, y posteriormente, construir un modelo agregado capaz de solucionar nuestro problema.

Frecuentemente, podemos encontrar 3 comportamientos, los cuales son fácilmente identificables incluso gráficamente, que nos permiten obtener información bastante valiosa acerca del problema que estamos estudiando, y nos facilitan la decisión de escoger la técnica adecuada:

- **Tendencia**. Es el movimiento de los valores de serie a largo plazo, es decir, el comportamiento mostrado por los datos a la hora de estudiar su progresión desde el inicio de la serie hasta su final. El objetivo es encontrar si esta mantiene una dirección, de manera general, en su periodo de muestreo, buscando si sigue un comportamiento creciente, decreciente o constante, al igual que en el caso de estudio de monotonía clásico en funciones. Normalmente, se representan mediante comportamientos sencillos, y no se ven afectadas por grandes variaciones o el impacto de otras componentes de la serie, como variables aleatorias u otros factores externos. Normalmente, podemos modelarla haciendo uso de funciones elementales, como ecuaciones lineales o funciones exponenciales, logarítmicas o polinomiales.

  A veces, se requieren procesos de suavizado para así poder eliminar el efecto de otras componentes ruidosas que rodean la tendencia real de la serie. Esto se puede conseguir, de manera similar a la convolución, desplazando una ventana a lo largo de la serie, de manera que los valores dentro de ella sean suavizado. La forma más habitual de lograrlo es con una media móvil, la cual, cuanto mayor amplitud tenga, mayor reducción de ruido conseguiremos.

  En la figura 2.1, podemos ver un ejemplo de su cálculo representado gráficamente sobre el dataset Air Passengers [4].

- **Estacionalidad**. Son fluctuaciones periódicas y predecibles dentro de la serie temporal, las cuales siguen una determinada frecuencia y que pueden ser fácilmente modelables una vez se observa al menos un periodo. Normalmente, coinciden con unidades de medida de calendario, pudiendo encontrar así frecuecias semanales, mensuales, trimestrales, anuales... etc. Normalmente, son un factor visualmente sencillo de identificar, el

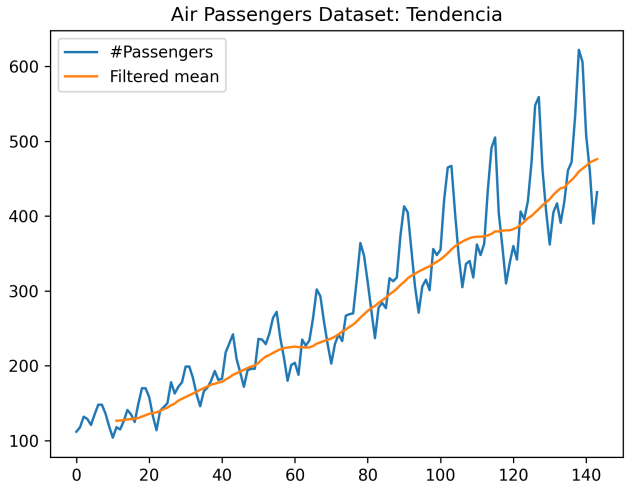

Figura 2.1: Estudio de la tendencia en el dataset Air Passengers

cual apenas varía ente períodos, y que nos permite obtener información muy valiosa para el modelado.

Podemos encontrar este comportamiento, por ejemplo, en los desplazamientos realizados durante las épocas vacionales: podremos observar un patrón de crecimiento en estas en Navidad y las vacaciones de verano, principalmente julio y agosto; y en el resto de meses el comportamiento será a la baja. Y eso ocurrirá con un período anual, ya que dichas fechas se ubican siempre en el mismo lugar. Diferente caso sería con Semana Santa, ya que su fecha no es coincidente todos los años y no cumple al 100 % de estacionalidad como las otras dos, ya que aunque es acotable en calendario, no es exactamente coincidente anualmente.

En el caso del dataset anterior, se puede observar esta estacioalidad claramente (figura 2.2)

- **Ciclo**. Son también fluctuaciones de la serie temporal, pero, a diferencia de ser regulares, siguen un período irregular de logitud fija. Normalmente, está vinculado a eventos que no están especialmente vinculados a fenómenos de calendario. El ejemplo más representativo podría ser la tendencia de crecimiento económico de un país, o el comportamiento de las acciones en bolsa de una sociedad anónima.

  En la figura 2.3 podemos un ejemplo de este comportamiento con los permisos concedidos para la construcción de viviendas en EEUU [11], donde no podemos encontrar patrones claros como en Air Passengers.

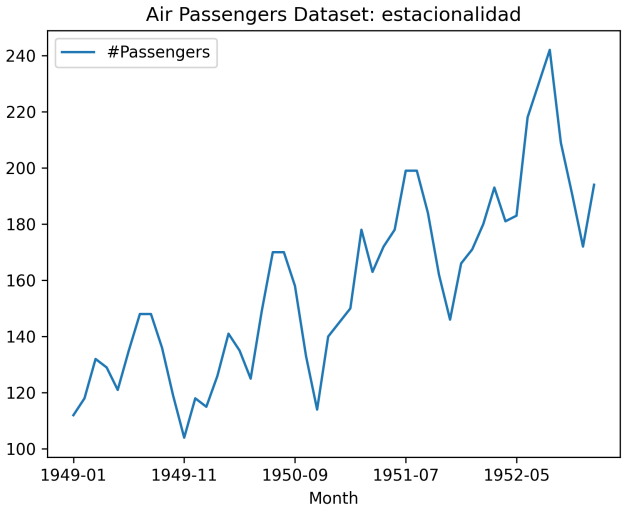

Figura 2.2: Estudio de la estacionalidad en el dataset Air Passengers

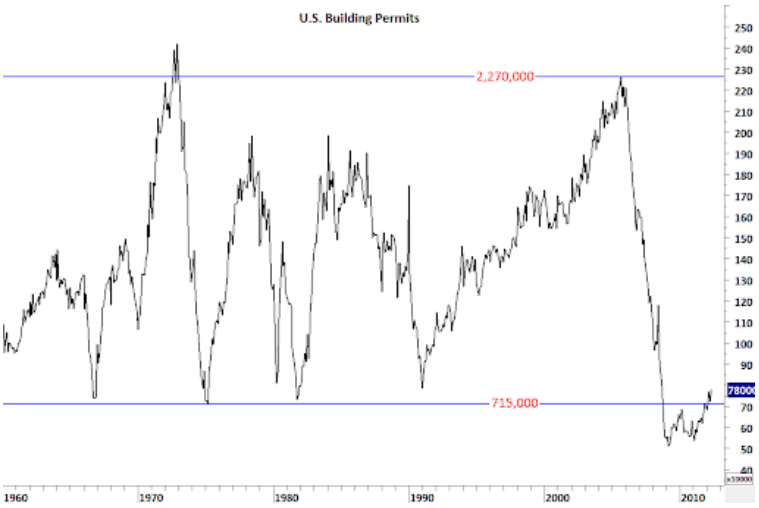

Figura 2.3: Estudio de ciclos en USA building permits

Una vez comprendidos estos conceptos, podremos utilizar su información para decidir qué modelo se adapta mejor a nuestro problema y además tratar de modelar la tendencia y la estacionalidad. Anteriormente, mencionamos la gran utilidad de modelos como ARIMA, que funciona bajo este concepto. Pero no se trata del único modelo existente bajo este paradigma, sino uno de los más utilizados. En función del mecanismo de agregación de la solución, podemos clasificar las técnicas en dos grupos:

- **Descomposición aditiva**. Realiza una separación de la serie temporal varias componentes, las cuales son modeladas por separado y sumadas entre si para dar lugar a

la función predicha final (ecuación 2.1.1). Es la más sencilla de usar en casos simples. Es indicada cuando las fluctuaciones estacionales por las variaciones entorno a la tendencia no varían con el valor de la serie temporal, es decir, pueden prácticamente mantenerse entre a lo largo de toda la serie sin apenas cambios. En general, se usa cuando los elementos no depende los unos de otros.

$$y(t) = T(t) + S(t) + C(t) + E(t)$$

- **Descomposición multiplicativa**. Se utiliza esta alternativa cuando las diferentes componentes dependen en nivel general de la serie (ecuación 2.1.1). Es el caso de las series en las cuales los aumentos en la tendencia provocan aumentos también los picos de las tendencias estacionales o los ciclos. Por ejemplo, en el dataset de Air Passengers, el comportamiento que se da es de este tipo.

$$y(t) = T(t) \times S(t) \times C(t) \times E(t)$$

Donde, en ambos casos:

- *T(t)*: Componente de la tendencia
- *S(t)*: Componente estacional
- *C(t)*: Componente cíclica
- *E(t)*: Componente aleatoria y error

La elección de cada método depende, por tanto, de los datos, y nos beneficiaremos, como ya hemos visto a lo largo de la definición, de una adecuada visualización de los mismos cuando sea posible.

A continuación, definiremos en mayor detalle 3 de los algoritmos basados en descomposición más usados: STL [5], ARIMA [3] y Prophet [16].

2.1.1.1. STL

Esta técnica, llamada *Seasonal Time-Series Decomposition*, nos permite descomponer la serie en partes más sencillas de modelar, separando la serie en tres componentes básicas: tendencia, estacionalidad y resto, siguiendo así el esquema visto anteriormente. De esta forma, podemos comprender en mayor detalle cómo evolucionan los valores a lo largo del tiempo, y comprender mejor cómo modelar su comportamiento.

Funcionamiento

Para obtener un modelo para cada una de ellas, normalmente se sigue un proceso de 3 pasos que nos permite aislar la información.

1. **Extracción de la tendencia**. Se comienza extrayendo la tendencia de la serie para así obtener el comportamiento subyacente de la misma, y saber si es creciente, decreciente o se mantiene constante. En este algoritmo, se consigue mediante un proceso de suavizado, llamado *Locally Estimated Scatterplot Smoothing* (Loess) de manera iterativa. Para ello, primero se comienza aplicando Loess, de manera que se suavizan los valores realizando una regresión no parámetrica, donde se observan los valores cercanos

a cada timestamp de manera que se realiza un promedio que reduce las diferencias en el entorno. Si la serie es muy compleja o ruidosa, es posible que sea necesario repetir este proceso varias veces, por lo que se convierte en un proceso iterativo, utilizando tamaños mayores de ventana entorno a cada valor.

Dicha ventana se estima a través de pesos, los cuales se ven reducidos cuanto más alejados al valor están. Estos se rigen por el valor de amplitud h, que recogen las ecuaciones clásicas de la regresión de Loess. Dado un conjunto de datos (t_i, y_i), el valor suavizado en t_0 se calcula como:

$$\hat{y}(t_0) = \mathbf{x}_0^\top \hat{\boldsymbol{\beta}}(t_0)$$

donde $\hat{\boldsymbol{\beta}}(t_0)$ se obtiene minimizando la suma ponderada de errores:

$$\hat{\boldsymbol{\beta}}(t_0) = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} w_i(t_0) \left(y_i - \mathbf{x}_i^\top \boldsymbol{\beta}\right)^2$$

Los pesos $w_i(t_0)$ dependen de la distancia temporal y se definen, habitualmente, con el kernel tricúbico:

$$w_i(t_0) = \left(1 - \left|\frac{t_i - t_0}{h}\right|^3\right)^3, \quad \text{para} \quad |t_i - t_0| < h$$

El valor arrojado será 0 fuera del intervalo, y un valor entre 0 y 1 dentro de él, seleccionando así la relevancia de cada punto respecto a sus valores del entorno.

2. **Extracción de la estacionalidad**. Para extraer la estacionalidad, debemos aislarla de la tedencia que modelamos en el paso anterior. Para ello, basta con restar dicha componente a la serie original, y así disponer ahora únicamente del los patrones estacionales y el ruido.
$$y'(t) = y(t) - trend(t)$$

   A partir de aquí, debemos dividir la serie sin tendencia en subseries estacionales, agrupando los datos según la posición que ocupan dentro del ciclo; por ejemplo, si identificamos patrones con frecuencia mensual, se agruparían todos los valores correspondientes al mismo mes en años distintos, y sobre cada una de estas subseries se aplica un suavizado Loess de forma individual. Así, podremos capturar con precisión el patrón que se repite en cada estación, de manera similar a un promedio. Gracias a este proceso, podremos reconstruir una componente estacional coherente con las variaciones periódicas observadas en nuestro conjunto de datos, al que denominaremos *s(t)*.

3. **Obtención del resto**. Una vez extraídas tanto la tendencia como la estacionalidad, la componente residual se obtiene simplemente como la diferencia entre la serie original y la suma de las dos componentes anteriores:
$$r(t) = y(t) - trend(t) - s(t)$$

   Esta parte representa la variabilidad no explicada por los patrones sistemáticos de largo plazo ni por los ciclos periódicos, e incluye tanto el ruido aleatorio como cualquier

información no capturada por el modelo. Si bien no es modelable, es muy útil para identificar si la serie es correctamente modelable por este método, ya que si observamos demasiada información en esta componente, nos está indicando que una parte importante de la información no está siendo recogida. Lo deseable es la que la varianza descrita sea la mínima posible. Sin embargo, hay algunos casos en los que encontrar puntos extremos es de utilidad, y se trata de la búsqueda de valores anómalos. Si la serie restante permanece estable y con poca varianza, a excepción de ciertos puntos, puede ser de interés estudiar qué ocurrió en dichos instantes, ya que su comportamiento se sale del comportamiento habitual modelado.

En su publicación original, podemos encontrar un ejemplo de ejecución sobre el conjunto de datos *Daily Carbon Dioxide Data* (ver figura 2.4), que recoge información sobre las emisiones de CO_2 desde el 17 de abril de 1974 hasta el 31 de diciembre de 1986, y permite observar la información recogida por cada una de las componentes.

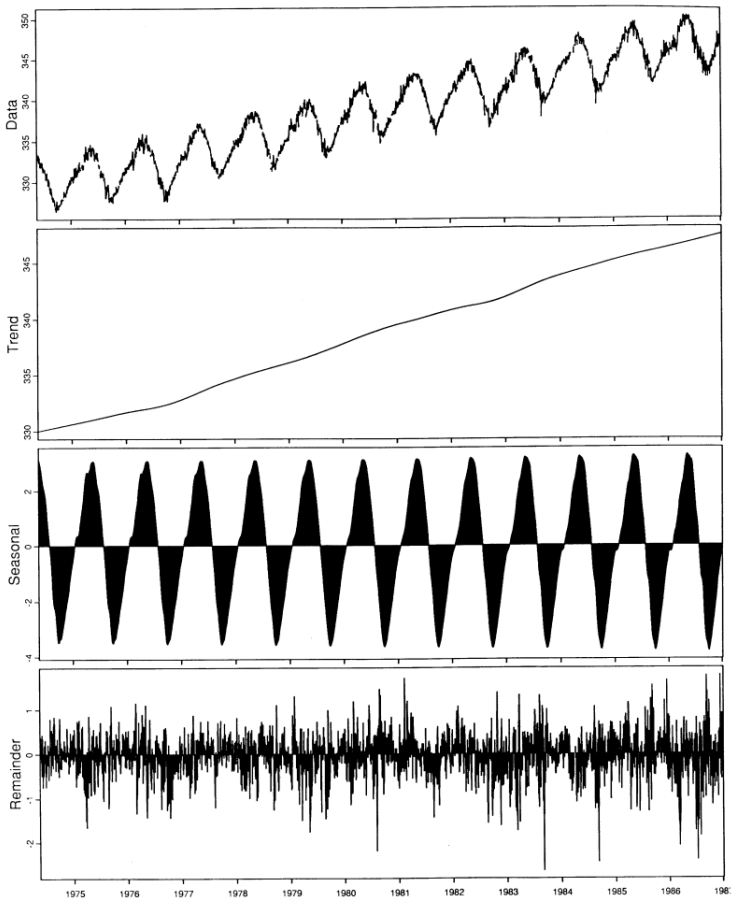

Figura 2.4: STL: Ejemplo de caso de estudio, Daily Carbon Dioxide Data [5]

En este caso, podemos apreciar visualmente una serie fácilmente modelable por este enfoque, ya que la curva describe una tendencia creciente bastante clara, y los periodos estacionales están bastante bien delimitados y no se ven influenciados por el crecimiento

recogido en la tendencia. Es un ejemplo perfecto para el uso de los modelos basados en descomposición aditiva.

Conclusiones
STL supuso una buena base para modelos posteriores, pero esta forma de modelado presenta varias limitaciones, que acotan su funcionamiento a problemas simples:

- Sólo se limita a descomponer series de naturaleza aditiva, por lo que aquellas en las que las diferentes componentes se vean influenciados entre sí requerirán técnicas propias de modelos multiplicativos.
- Pueden perderse gran cantidad de detalles explicados por los datos, debido a la configuración del parámetro de suavizado. Si alisamos de más la serie, podemos estar perdiendo información clave acerca de nuestro problema, dejando completamente de lado ciertos comportamientos sutiles, lo cual aumenta la tasa de error. Debemos escoger adecuadamente los valores al aplicar Loess.
- Es costoso computacionalmente en caso de series de larga duración, ya que el cálculo de aplicar el kernel se hace para cada punto en cada iteración sobre el conjunto de datos, tanto para encontrar la tendencia como para luego generalizar la estacionalidad.

Pero, sin lugar a dudas, la principal desventaja de este método es que no consigue por sí solo lo que deseamos resolver en este trabajo: predecir, sobre todo, a largo plazo. STL es un procedimiento muy útil que nos permite entender el comportamiento de los datos, y descomponerlo para comprender el funcionamiento de cada componente, pero no es posible por sí solo realizar predicciones. Dependemos del apoyo de otros procedimientos que nos permitan estimar , basándonos en este historial, la progresión de las componentes extraídas como la tendencia (modelable por regresión), y la estacional, la cual podría ser evaluada de manera simplificada por su valor medio histórico por intervalo.

Esto lo hace en un método interesante desde el punto de vista de análisis, y que permite dar lugar a modelos que sí son capaces de estimar predicciones futuras, pero no es una herramienta por sí sola que nos permita resolver la tarea que nos concierne en este trabajo.

2.1.1.2. ARIMA

Basándonos de nuevo en el método de descomposición de series, podemos encontrar el modelo **ARIMA** (*AutoRegressive Integrated Moving Average*) una técnica estadística ampliamente utilizada para el modelado de series temporales univariantes. Como ya adelantamos en la introducción del proyecto, su estructura se basa en la combinación de tres componentes principales: una parte autoregresiva (AR), una parte integrada (I) y una parte de media móvil (MA). Cada una de estas partes permite capturar diferentes aspectos del comportamiento temporal de los datos.

Componente de Integración
Por integración, entendemos el número de veces que es necesario integrar la serie temporal original para que esta se vuelva estacionaria. De manera resumida, una serie estacionaria es aquella cuyas propiedades estadísticas, como la media, la varianza y la autocorrelación entre valores, se mantienen constantes a lo largo del tiempo, como ya vimos en la definición

de los modelos aditivos.

La diferenciación nos permite eliminar tendencias o patrones sistemáticos de crecimiento o decrecimiento, facilitando el modelado de las componentes autoregresivas (AR) y de media móvil (MA). A pesar de que su nombre nos pueda insinuar un proceso complejo de integración analítica, al realizarse sobre valores numéricos, no es más que una diferencia de valores:

$$y'_t = y_t - y_{t-1}$$

Este proceso puede aplicarse tantas veces como sea necesario para lograr estacionariedad. Dicho número de repeticiones viene controlado por el valor de orden d, que indica el número de veces que se ha diferenciado la serie. En general, el modelo ARIMA supone que tras aplicar d diferencias, la serie resultante $y^{(d)}$ puede ser modelada por un proceso estacionario ARMA(p, q). Por tanto, el componente I de ARIMA es un procesamiento indispensable de la serie cuando no se cumpla el requisito de estacionariedad, y permitirá calcular sobre la serie resultante las componentes AR y MA con mayor facilidad.

Para saber cuando deja de ser necesario aplicar el proceso iterativo de integración, podemos hacer uso de dos posibles mecanismos: uno gráfico, basado en el gráfico de autocorrelación de valores, y otro estadístico, basado en el test de Dickey-Fuller aumentado (ADF).

- El análisis gráfico consiste en observar el comportamiento de la función de autocorrelación (ACF) tras aplicar una o varias diferenciaciones. Si la serie es no estacionaria, las valores siguientes de autocorrelación decaen lentamente, indicando que queda información relevante en dichas componentes. Por el contrario, si la serie es estacionaria, la ACF suele caer rápidamente a cero, estableciendo que no es necesario continuar diferenciando (ver figura 2.5).

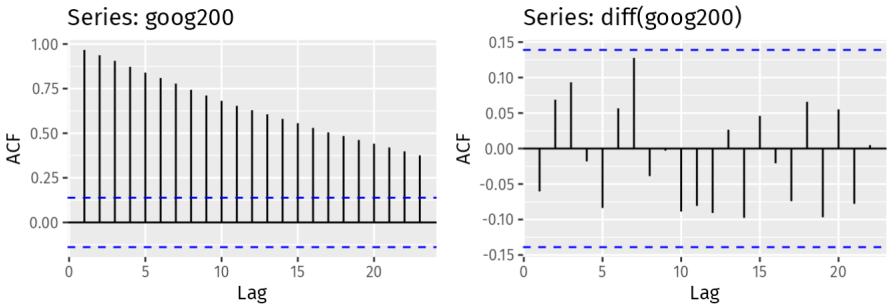

Figura 2.5: Gráfico ACF: no estacionario (izquierda) vs estacionario (derecha) [9]

Numéricamente, la autocorrelación en un retardo k equivale a:

$$\rho_k = \frac{\text{Cov}(y_t, y_{t-k})}{\sigma^2}$$

donde σ^2 es la varianza de la serie y $\text{Cov}(y_t, y_{t-k})$ es la covarianza entre los valores actuales y los desplazados k periodos atrás.

- El test ADF, por su parte, nos proporciona la misma conclusión de manera numérica, para lograr un mayor rigor estadístico en la solución. Su funcionamiento radica en la búsqueda de una raíz unitaria en la serie, en cuyo caso significa que la serie presenta comportamiento no estacionario.

  De forma general, el test se basa en ajustar una regresión del tipo:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \epsilon_t$$

  donde $\Delta y_t = y_t - y_{t-1}$ representa la primera diferencia de la serie, γ es el parámetro clave para detectar la raíz unitaria, y ϵ_t es el error.

  Sin embargo, lo más relevante para su uso es comprender cómo funciona el test de hipótesis. Como hipótesis nula (H_0), define que la serie no es estacionaria, es decir, que presenta una raíz unitaria, y como (H_1) podremos aceptar que la serie sea estacionaria. Para ello, basta con observar el valor de p:

  - Si el **p-valor** del test es significativo se puede rechazar la hipótesis nula. En ese caso, la serie puede considerarse estacionaria, y podemos parar el proceso iterativo de diferenciación.
  - Si el **p-valor** es mayor al valor de significación, no podemos rechazar la hipótesis nula, lo que indica que la serie sigue siendo no estacionaria, y debemos continuar el proceso de integración.

Componente autoregresiva (AR)

Esta parte del modelo representa la relación entre el valor actual de la serie y sus valores pasados. Supone que el valor presente puede explicarse como una combinación lineal de observaciones anteriores, los cuales se denominan retardos o lags. Normalmente, el número de lags a emplear es un hiperparámetro clave del modelo, el cual es controlado por p.

Matemáticamente, podemos expresarlo como:

$$z_t = w_1 y_{t-1} + w_2 y_{t-2} + \cdots + wi_p y_{t-p} + \varepsilon_t$$

donde:

- $w_1, w_2, \ldots, w_p$ son los coeficientes autoregresivos que indican el peso o importancia de cada valor pasado. Cuando usamos el valor p, establecemos el número de valores con peso 1 que tomará nuestro modelo, mientras que el resto se mantienen a 0. Pero, podríamos tener variantes en las que dichos pesos fueran valores flotantes entre 0 y 1 que permitan una ponderación más refinada de los valores.
- ε_t es el término asociado al error en el instante t.

Gracias a la componente AR, podemos capturar relaciones de corto plazo y patrones de dependencia temporal, siempre que la serie sea estacionaria. Es habitual que muchas series temporales muestren este tipo de estructura, donde los valores pasados son predictivos del comportamiento futuro inmediato, como ya vimos con el caso de las temperaturas y el consumo eléctrico. Por tanto, la elección del parámetro p es clave: un valor pequeño implica que sólo los valores más recientes tengan influencia, mientras que un valor mayor permite

capturar relaciones más amplias en el tiempo, pero puede aumentar el riesgo de sobreajuste y se pierda localidad. En definitiva, se trata de un elemento esencial del rendimiento de ARIMA, por lo que es clave elegir adecuadamente su valor.

Componente de media móvil (MA)

Por último, nos queda describir el funcionamiento de la media móvil. Esta tiene como objetivo modelar la parte estocástica de la serie temporal, es decir, los errores de predicción cometidos en instantes anteriores. Anteriormente, vimos que la parte autoregresiva se basaba en realizar la media sobre sus mismos valores, mientras que aquí tratamos de extraer información de los residuos.

Formalmente, un modelo MA de orden q se define como:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

donde:

- y_t es el valor observado de la serie en el instante t,
- μ es la media de la serie (si está centrada, puede ser cero),
- ε_t es el término de error en el instante t,
- $\theta_1, \ldots, \theta_q$ son los coeficientes del modelo MA.

Esto nos permitirá capturar patrones aleatorios que no pueden explicarse mediante una simple tendencia o correlación temporal, y por tanto, mejorando los resultados y el rendimiento del modelo, al suavizarse los errores pasados para acercanos más a la serie real. Por tanto, el número de errores a considerar q debe escoger de manera adecuada. Para facilitar esta decisión, podemos basarnos de nuevo en dos enfoques: en emplear la información del gráfico ACF, o bien, de manera experimental:

- El análisis gráfico mediante ACF nos permite analizar el comportamiento de los lags; en un modelo MA puro, los primeros q retardos (*lags*) suelen mostrar autocorrelación significativa, pero luego suelen decrecer de manera repentina a cero. Una buena estimación puede ser tomar como valor de q el último lag con autocorrelación significativa antes del corte. Es decir, si a partir del punto q+1 se produce un descenso abrupto, tomamos q.
- Si la gráfica no nos proporciona una respuesta clara, y la serie es modelable en un tiempo razonable, se puede realizar una búsqueda hiperparámetros, probando diferentes valores de q, y entrenando un modelo ARIMA para cada combinación posible (junto a valores de p y d), y evaluar su rendimiento utilizando métricas como el *Mean Squared Error* (MSE). Para facilitar el proceso, podemos usar técnicas clásicas como grid search, que no es más que realizar todas las combinaciones posibles de parámetros p,q y d. Pero, de manera más avanzada, existen adaptaciones de ARIMA en frameworks como auto_arima, que usan una estimación automática de los hiperparámetros.

Conclusiones

En resumen, ARIMA nos permite modelar gran cantidad de problemas de manera simple, cumplendo como única restricción la estacionariedad. Pero principalmente esta característica, si bien facilita enormemente el proceso, también impacta sobre la calidad del resultado;

si la serie tiene una estructura muy compleja y no estacionaria, convertirla para que lo sea puede provocar una pérdida masiva de información, ya que al diferenciar estamos suprimiendo información que podría ser potencialmente relevante. En casos extremos, la pérdida de expresividad podría ser tal hasta lograr una serie muy plana, sin comportamientos aparentes. Por tanto, es necesario emplear técnicas más sofisticadas que no requieran dicha condición para extraer modelos más generalizables.

Por otro lado, las predicciones a largo plazo también son problemáticas: como se buscan relaciones lineales entre variables, con estaciones definidas, y variables sencillas de representar, cuando se usa ante series complejas puede tenderse a formular una predicción bastante plana y poco informativa que tienda a la media, ante la imposibilidad del modelo a adaptarse a los cambios. De manera directa, esto influencia también ante la capacidad de adaptación de los outliers: ante su presencia, se podría ver alterada la predicción real, y se deben estudiar y analizar en fases previas de preprocesado.

Por último, en caso de que se disponga de información adicional, como intervalos de tiempo, localización en calendario, habitualmente usada cuando se dispone de datos de larga extensión para facilitar su estructuración y la detección de períodos, no es posible incorporarlo de forma directa. Habría que realizar modificaciones sobre este, como por ejemplo, las integradas en ARIMAX, que se basa en la implementación original [3], pero añadiendo de manera aditiva operaciones adicionales aprovechando la información externa.

2.1.1.3. Prophet

Como alternativa más reciente de los modelos aditivos, disponemos de Prophet [16]. Desarrollado por Meta, apareció teniendo como objetivo facilitar la generación de predicciones de calidad, incluso si es empleado por analistas sin experiencia en series temporales. Según su blog oficial [15], Prophet fue creado para responder a la necesidad de escalar la producción de pronósticos precisos en entornos empresariales sin necesidad de altos conocimientos para poder extraer buenas conclusiones.

Funcionamiento

A diferencia de los modelos anteriores, que definían la necesidad de cumplir ciertas restricciones, como la estacionariedad, Prophet nos permite trabajar con entornos más cambiantes, al estar diseñado para recibir datos de al menos varios meses de duración, donde si bien existen patrones estacionales claros por la estructura de tipo calendario, es capaz de adaptarse a cambios e imprevistos. Es decir, es capaz de modelar días festivos o de caracter especial sobre el impacto del desarrollo de la serie (como el lanzamiento de un nuevo servicio o producto de una empresa) y es más robusto frente a outliers que los modelos que vimos anteriormente. (ver figura 2.6).

La descomposición se basa en una suma de tres componentes, de manera similar a los modelos anteriores:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Donde:

- $g(t)$ representa la tendencia a largo plazo, que puede ser lineal o logística, y puede

contener cambios de tendencia que se ajustan de manera automática.

- $s(t)$ modela la estacionalidad periódica mediante funciones basadas en series de Fourier. Permite capturar estacionalidades diarias, semanales o anuales, aunque las implementaciones disponibles permiten también especificar intervalos definidos por el usuario.
- $h(t)$ representa efectos de festividades o eventos especiales, introducidos manualmente por el usuario.
- ε_t representa el error asociado a la predicción realizada, que se asume con distribución normal.

Entrando en mayores detalles de implementación, una de las principales novedades reside en algo tan sencillo de visualizar como la tendencia. Hasta ahora, normalmente disponíamos de un modelado de única ecuación, comúnmente lineal, que era común a toda la serie. Pero Prophet es más flexible gracias a la detección automática de cambios de pendiente, los cuales denomina *changepoints*. Además, para reflejar comportamientos y cambios de tendencia de manera más suave y progresiva, permite ajustar la suavidad en torno a los changepoints mediante un parámetro de escala.

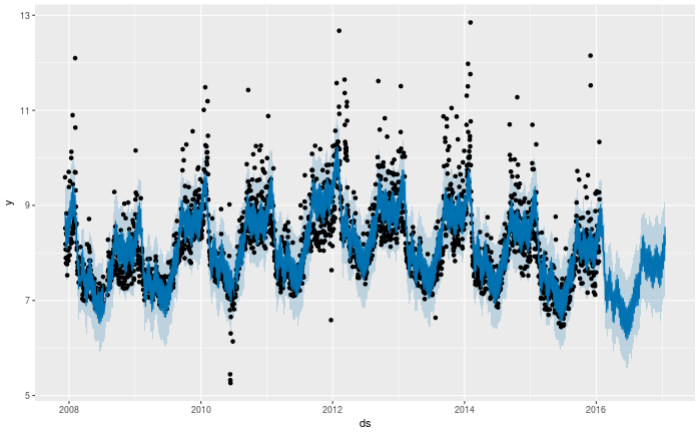

Figura 2.6: Prophet: Ejemplo de predicción en entorno ruidoso [15]

Adicionalmente, la modelización de la estacionalidad también recibe un formulación más sofisticada. Usando Fourier, podemos introducir la duración del período de cada ciclo (P) e ir ajustando su precisión mediante el parámetro n (ver ecuación 2.1.1.3). Pero lo más interesante es que se facilita la inclusión explícita de festividades o eventos que pueden afectar significativamente la serie. Para ello, el modelo permite especificar ventanas alrededor de esos puntos para así aprender también su comportamiento.

$$s(t) = \sum_{n=1}^{N} \left[a_n \cos\left(\frac{2\pi n t}{P}\right) + b_n \sin\left(\frac{2\pi n t}{P}\right) \right]$$

Conclusiones

En resumen, Prophet supone un salto considerable en cuanto a adaptación y cercanía a los datos extraídos de casos reales. Nos permite insertar la información explícita que mencionabamos en ARIMA, ya que nos permite añadir de manera muy intuitiva información adicional del problema. Además, en su publicación original se destaca la eficacia incluso cuando los datos presentan imperfecciones, como valores atípicos, datos faltantes o cambios abruptos en la tendencia, lo cual facilita enormemente el proceso de preprocesado, el cual puede reducirse al mínimo. Esto aporta considerablemente a su faceta relacionada con la facilidad de uso.

Sin embargo, aunque supone un gran salto con lo disponible anteriormente, aún se siguen manteniendo varias de las limitaciones que venimos analizando a lo largo de los modelos. En primer lugar, podemos observar que su enfoque podría verse más implicado en el ajuste de curvas que en una modelación como tal de las dependencias temporales, que nos permitan realizar forecasting a largo plazo. Si bien el modelo destaca por su sencillez al usuario, la elección de parámetros sigue siendo un aspecto delicado, ya que el modelo puede incurrir en sobreajuste si no se calibran adecuadamente. Debemos asegurarnos de sus valores estudiando previamente el conjunto de datos.

Finalmente, Prophet no incluye de manera nativa mecanismos autoregresivos ni permite modelar series multivariantes, lo cual representa una limitación cuando trabajamos con series que miden varios factores al mismo tiempo, y que incluso pueden estar correlacionados entre sí.

2.1.1.4. Comparativa final y limitaciones

A lo largo de este apartado, hemos analizado tres de los enfoques más relevantes para la descomposición de series temporales: STL, ARIMA y Prophet. Aunque todos comparten el mismo objetivo, que es separar la serie en componentes interpretables como tendencia, estacionalidad y ruido, cada uno lo aborda desde una perspectiva distinta, lo que se traduce en ventajas e inconvenientes específicos (ver tabla 2.1).

- El método **STL** nos ha resultado especialmente útil en las fases de análisis exploratorio. Su uso de suavizado local mediante Loess le permite adaptarse a estacionalidades no necesariamente fijas y proporciona una descomposición visualmente clara e interpretable. Sin embargo, al tratarse de un enfoque puramente descriptivo, su utilidad para la predicción es limitada, y se requieren herramientas adicionales.

- Por su parte, **ARIMA** representa una aproximación estadística clásica basada en la modelización de las diferentes componentes temporales de la serie. Hemos establecido que resulta eficaz en series estacionarias con comportamientos lineales, gracias a su capacidad para capturar tanto relaciones autoregresivas como de medias móviles. Pero la necesidad de afinar los hiperparámetros, la precondición de estacionareidad y su naturaleza lineal limita su rendimiento ante patrones no lineales o estacionalidades más complejas.

- Por último, **Prophet** es una solución más moderna y accesible, especialmente diseñada para aplicaciones empresariales. A diferencia de ARIMA, no exige estacionariedad y su estructura facilita su uso por usuarios sin conocimientos especiales sobre series tem-

porales. Sin embargo, carece de componentes autoregresivos y no modela relaciones multivariantes de forma nativa, lo que puede comprometer su versatilidad.

Característica	**STL**	**ARIMA**	**Prophet**
Enfoque	Suavizado Loess	Modelo estadístico autoregresivo	Modelo aditivo estructurado
Requiere estacionariedad	No	Sí	No
Componente de tendencia	Suavizado local	Parte del modelo (AR/I)	Curva lineal o logística con puntos de cambio
Modelado de estacionalidad	Suavizado periódico	Diferenciación estacional o SARIMA	Series de Fourier
Eventos especiales	No contemplado	No los contempla	Incluidos explícitamente
Capacidad predictiva	Limitada	Buena si la serie es estacionaria y lineal	Alta en escenarios empresariales comunes
Robustez ante valores atípicos	Alta	Baja	Alta
Facilidad de uso	Media	Baja	Alta
Interpretabilidad	Muy alta	Media	Muy alta

Tabla 2.1: Comparativa entre métodos de descomposición de series temporales

En resumen, podemos afirmar que estos modelos constituyen una base sólida para el análisis inicial de series temporales, pero enfrentamos varias dificultades con datos dinámicos, dependencias de largo plazo, series multivariantes o patrones no lineales. Es en estos contextos donde se vuelve necesaria la incorporación de modelos más avanzados, como las redes neuronales recurrentes (RNN) o incluso los Transformers, que permiten modelar dependencias temporales profundas de manera eficiente y escalar a volúmenes de datos mayores. En los apartados siguientes, analizaremos estos nuevos modelos.

2.1.2. Redes Neuronales Recurrentes

Hasta ahora, la mayoría de soluciones estudiadas se han basado en el procesamiento de un conjunto de datos de entrada, que es procesado y modelado de manera más o menos sofisticada, y se otorga una salida, como, por ejemplo, los modelos de descomposición que hemos abordado hasta ahora. No obstante, en escenarios donde las series temporales presentan relaciones no lineales complejas, múltiples escalas de dependencia y estructuras difíciles de modelar de forma explícita, resulta necesario recurrir a modelos más avanzados.

Podríamos pensar en emplear modelos tradicionales vistos en otros ámbitos, como las redes neuronales. Pero en dichos problemas, las entradas y salidas no guardan una relación entre sí; son independientes. En los conjuntos de entrenamiento habituales, como problemas tabulares de regresión o clasificación, lo que nos importa es asignar el valor adecuado, y normalmente, cada instancia es independiente del resto. Pero cuando disponemos de información secuencial, como el lenguaje natural, o en nuestro caso, las series temporales, esto no es así.

En los métodos basados en descomposición, hemos podido apreciar que nos basamos continuamente en los valores pasados para ser capaces de modelar perdiendo la mínima información posible, y que por ejemplo, encontrar patrones como la estacionalidad es de gran importancia para minimizar el error y a su vez simplificar el modelado. Esto descarta la utilización de perceptrones multicapa, o capas densamente conectadas para el modelado de series temporales con un mínimo de complejidad, debido a dicha dependencia. Solo quizás, podríamos modelar series simples con una clara tendencia y forma, debido a que podrían asemejarse a una regresión. Pero si queremos obtener resultados escalables a redes más compleja, debemos tener en cuenta las conexiones temporales entre cada timestamp, y modelar dicho orden dentro de la secuencia global de los datos. Esto lo podemos conseguir relacionando cada salida con la anterior y es así como surge el concepto de las Redes Neuronales Recurrentes.

2.1.2.1. Funcionamiento de las RNN

Las RNN, al igual que las redes neuronales tradicionales, son capaces de procesar una entrada y arrojar una salida, modelando los datos internamente mediante sus capas ocultas. Pero la diferencia radica en la necesidad de establecer conexiones entre la salida anterior y la entrada actual. Esto se consigue con conexiones recurrentes que permiten que la información fluya de un instante al siguiente dentro de la propia red (ver figura 2.7).

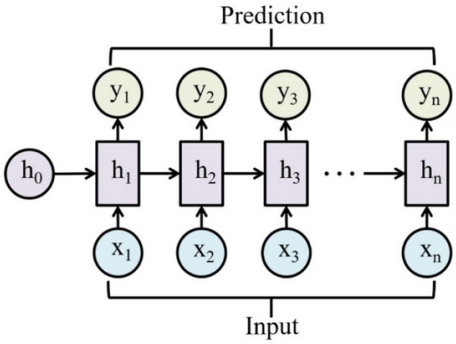

Figura 2.7: RNN: arquitectura del modelo [14]

Estructuralmente, una RNN clásica está compuesta de tres capas principales [14]: una capa de entrada, una capa oculta con conexiones recurrentes, y una capa de salida. El comportamiento de la red se define en función de cómo se actualiza su estado interno y cómo se generan las salidas. En su definición original, en cada instante de tiempo t, la red recibe un vector de entrada $\mathbf{x}_t$ y actualiza su estado oculto $\mathbf{h}_t$ según la siguiente ecuación:

$$\mathbf{h}_t = \sigma_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

donde:

- $\mathbf{W}_{xh}$ es la matriz de pesos entre la entrada y la capa oculta,
- $\mathbf{W}_{hh}$ representa la conexión recurrente entre los estados ocultos,
- $\mathbf{b}_h$ es el vector de sesgo asociado,
- σ_h es la función de activación, normalmente la tangente hiperbólica (`tanh`) o la rectificada lineal (`ReLU`).

A partir del estado oculto actualizado, se calcula la salida correspondiente mediante:

$$\mathbf{y}_t = \sigma_y(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y)$$

donde:

- $\mathbf{W}_{hy}$ conecta la capa oculta con la capa de salida,
- $\mathbf{b}_y$ es el vector de sesgo en la salida,
- σ_y es la función de activación de la salida.

Gracias principalmente al estado oculto, podemos integrar la información de entrada actual con las salidas previas, permitiendo que la red exhiba un comportamiento ordenado temporalmente en sus salidas. Pero, sin lugar a dudas, otro de los elementos clave para el funcionamiento adecuado es escoger una correcta función de activación σ_h, ya que se encarga de añadir la no linearidad a la salida. Una de las más empleadas es la tangente hiperbólica, *tanh*, que permite ajustar los valores de los pesos al intervalo [-1,1], el cual es bastante intuitivo gracias al estar centrado en cero, y además, permite la existencia de pesos tanto positivos como negativos.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

También es posible encontrar ReLU o sus variantes, ya que permiten ajustar los pesos de manera computacionalmente sencilla, pero solo permite el paso de valores positivos.

$$ReLU(x) = max(0, x)$$

Elegir una u otra opción depende del tipo de problema que tratemos de resolver o incluso del conjunto de datos de entrada, pero es un elemento clave que afecta directamente al aprendizaje de los estados ocultos y la propagación de dicha información.

2.1.2.2. Limitaciones de las RNN estándar

Aunque las RNN suponen un gran avance en el aprendizaje de datos secuenciales y permitieron una mejora considerable en los resultados, no están exentas de problemas durante el aprendizaje. Al igual que otros modelos que usan la propagación mediante gradientes, debemos tener especial cuidado con su gestión para evitar la explosión de gradientes y, en

su caso, el extremo contrario, el desvanecimiento.

Estos efectos se ven potenciados debido al particular funcionamiento del algoritmo de retropropagación entre los diferentes pasos temporales. A diferencia de una red neuronal clásica, necesitamos que los pesos tengan relación también con los pasos anteriores de la secuencia, y para lograrlo, debemos modificar el proceso, de manera que podamos emplear el procedimiento habitual con la nueva estructura oculta de pesos. En cada etapa de actualización:

1. La RNN se desenrolla (*unroll*) en el tiempo durante un número finito de pasos (por ejemplo, 10).
2. Se aplica retropropagación en la red desenrollada de la manera habitual.
3. Se acumulan los gradientes de todos los pasos temporales hacia los pesos compartidos.
4. Los parámetros se actualizan con una optimización como SGD o Adam.

De esta forma, implementamos el mecanismo conocido como *Backpropagation Through Time* (BPTT). Durante el entrenamiento con este mecanismo, calculamos los gradientes de la función de pérdida respecto a los pesos de la red a lo largo de varios pasos temporales, y es al propagar cuando podemos enfrentarnos a los dos problemas ya mencionados: la explosión y el desvanecimiento del gradiente.

El desvanecimiento del gradiente se produce si los términos derivados que aparecen en el cálculo (concretamente, los productos sucesivos de matrices Jacobianas) tienen valores propios menores que uno, su multiplicación sucesiva tiende a cero. Esto provoca que los gradientes disminuyan exponencialmente a medida que retrocedemos en el tiempo, impidiendo que la red actualice adecuadamente los pesos de las capas o pasos más antiguos, y como consecuencia, la red pierde la capacidad de aprender dependencias a largo plazo, ya que el error no alcanza a modificar significativamente las entradas lejanas.

Por otro lado, la explosión del gradiente es justo el caso opuesto; si los valores propios de las Jacobianas son mayores que uno, la multiplicación produce gradientes que crecen exponencialmente. Esto puede llevar a actualizaciones de pesos excesivas, provocando inestabilidad numérica. En ambos casos, por tanto, la capacidad de aprendizaje de la red se ve comprometida, y puede perjudicar gravemente a los resultados obtenidos.

Formalmente, podemos ver reflejado este comportamiento mediante el desarrollo de los estados ocultos. A partir de la fórmula recurrente del estado oculto:

$$\mathbf{h}_t = \sigma_h \left(\mathbf{W}_{xh} \mathbf{x}_t + \mathbf{W}_{hh} \, \sigma_h \left(\mathbf{W}_{xh} \mathbf{x}_{t-1} + \mathbf{W}_{hh} \mathbf{h}_{t-2} + \mathbf{b}_h \right) + \mathbf{b}_h \right),$$

el cálculo del gradiente de $\mathbf{h}_t$ respecto a un estado anterior $\mathbf{h}_{t-n}$ implica el producto de Jacobianas:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-n}} = \prod_{k=t-n}^{t-1} \mathbf{J}_k,$$

donde $\mathbf{J}_k$ representa la matriz Jacobiana del estado oculto en el instante k. La estabilidad de estos productos depende críticamente de los valores propios de $\mathbf{J}_k$, por lo que se trata del

elemento clave causante de las dos problemáticas descritas.

Para tratar de solucionar este problema, podemos encontrar variantes de este algoritmo en el estado del arte. Algunas, tratan de simplemente modificar el funcionamiento de la propagación, con técnicas como el *gradient clipping*, que permiten limitar el tamaño máximo del gradiente durante la retropropagación, evitando explosiones incontroladas. Pero también podemos encontrar arquitecturas específicas como las LSTM (Long Short-Term Memory), que introducen mecanismos de control del flujo de información mediante puertas, facilitando la retención de información relevante a largo plazo y mitigando el problema desvanecimiento de gradientes de manera estructural.

2.1.2.3. LSTM

Para resolver los problemas de desvanecimiento del gradiente asociados a las RNN clásicas, Hochreiter y Schmidhuber propusieron en 1997 las *Long Short-Term Memory networks* (LSTM) [8]. La innovación clave de estas redes es la introducción de mecanismos de puertas que regulan el flujo de información a lo largo del tiempo, permitiendo mantener una memoria a largo plazo de forma más estable.

Cada célula LSTM incorpora tres compuertas principales: la compuerta de entrada, la compuerta de olvido y la compuerta de salida. Estas compuertas permiten decidir qué información se debe incorporar, qué parte del estado anterior se debe olvidar y qué parte del nuevo estado se debe transmitir a la siguiente etapa. Gracias a este diseño, las redes LSTM son especialmente eficaces para modelar dependencias temporales de largo alcance.

El comportamiento interno de una célula LSTM se puede describir mediante las siguientes ecuaciones:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) && \text{(compuerta de entrada)} \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) && \text{(compuerta de olvido)} \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) && \text{(compuerta de salida)} \\
g_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) && \text{(nuevo contenido candidato)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(estado de la celda)} \\
h_t &= o_t \odot \tanh(c_t) && \text{(estado oculto)}
\end{aligned}
$$

Donde σ representa la función sigmoide, tanh la tangente hiperbólica, y $\odot$ la multiplicación elemento a elemento. Esta arquitectura permite mantener y actualizar información relevante a lo largo del tiempo, mitigando los problemas habituales en el aprendizaje de secuencias largas.

Cada compuerta dentro de una celda LSTM desempeña una función específica en la regulación del flujo de información. La compuerta de entrada controla cuánta información del nuevo vector de entrada x_t se incorpora al estado de la celda c_t; la compuerta de olvido determina qué parte del estado anterior c_{t-1} se retiene, y, finalmente, la compuerta de salida decide cuánta información del estado actual c_t se utiliza para calcular el estado oculto h_t (ver figura 2.8).

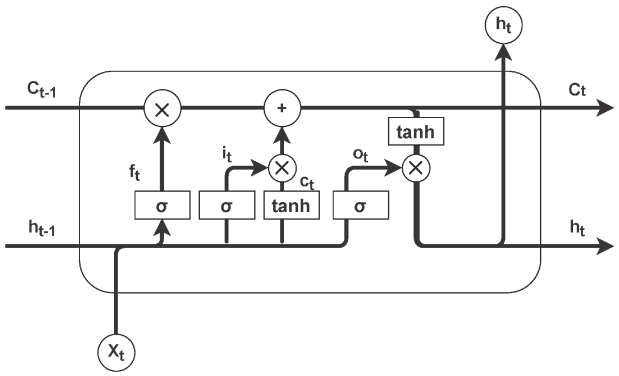

Figura 2.8: LSTM: funcionamiento de la arquitectura de puertas

Además, la entrada candidata de la celda, denotada como g_t, representa nueva información que podría añadirse al estado de la celda tras ser modulada por la compuerta de entrada.

Gracias a estos mecanismos de control, las redes LSTM pueden decidir dinámicamente qué información conservar o descartar en cada momento, lo que les permite gestionar dependencias temporales a largo plazo de manera mucho más eficaz que las RNN convencionales. En esta arquitectura, la recurrencia se mantienen a través del estado de la celda c_t, que actúa como una cinta transportadora que permite el traspaso continuo de información entre pasos temporales. Este flujo sostenido de memoria facilita la conservación de patrones relevantes a lo largo de secuencias de larga extensión.

Las multiplicaciones elemento a elemento entre compuertas y vectores asociados aseguran una interacción eficiente entre componentes, posibilitando transformaciones complejas sin comprometer la estabilidad durante el entrenamiento. En conjunto, las LSTM mantienen un equilibrio entre flexibilidad, memoria y estabilidad en el aprendizaje.

Sin embargo, no suponen la solución definitiva a los problemas, pues, si bien conseguimos solucionar el problema de los gradientes, hemos incorporado complejidad adicional al modelo que puede afectar en diferentes aspectos:

- **Complejidad computacional:** se requieren más recursos que modelos más simples debido al uso de múltiples compuertas y operaciones por paso temporal, incrementando el número de parámetros y el tiempo de entrenamiento. Además, su estructura recurrente impide aprovechar eficientemente las GPU, y aprovechar su potencia para lograr un buen escalado.
- **Dificultad para capturar dependencias muy largas:** Aunque superan a las RNN básicas, si tratamos de modelar predicciones a largo plazo, como buscamos en este trabajo, podemos apreciar una pérdida de efectividad.
- **Sensibilidad a la configuración:** el rendimiento depende de manera sensible de los parámetros escogidos, cuya selección no es una tarea trivial y puede afectar considerablemente a los resultados.

- **Interpretabilidad:** aunque su arquitectura muestra el flujo que sigue la información, interpretar de forma precisa el funcionamiento interno es un proceso complicado al agregar elementos adicionales como las puertas.

A modo de conclusión, aunque las LSTM representaron un avance fundamental en el ámbito del forecasting, en aplicaciones modernas suelen ser superadas por arquitecturas más flexibles y escalables como los Transformers, los cuales estudiaremos a continuación, y serán nuestro principal foco de atención.

2.2. Transformers

Los Transformes son una de las mayores invenciones de los últimos años en el tratamiento de datos secuenciales, como el lenguaje natural, y en series temporales. Su estructura (ver figura 2.9), la cual propone el uso de mecanismos basados en atención, ha permitido prescindir de los algoritmos basados en recurrencia, y simplificar el proceso complejo de aprendizaje que teníamos debido a los problemas con la propagación y los gradientes.

A diferencia de las redes RNN o LSTM, que procesan la información de forma secuencial y requieren mantener estados ocultos a lo largo del tiempo, los Transformers permiten procesar todas las posiciones de la secuencia en paralelo, lo que mejora drásticamente la eficiencia computacional y la capacidad de modelar relaciones a largo plazo. Esto nos permite también aprovechar arquitecturas de computación con gran capacidad para la paralelización, como el uso de GPUs, y así reducir tiempos de ejecución, o bien, aprovechar la capacidad extra de paralelización para enfrentarnos a problemas mucho más complejos, con mayor número de parámetros y dependencias a largo plazo.

Su funcionamiento se basa en cuatro elementos, los cuales estudiaremos a continuación: su arquitectura basada en bloques encoder-decoder, la posiblidad de parelelizado mediante múltiples heads, el mecanismo de *self-attention*, y la incorporación de información adicional mediante *embeddings*, para así introducir información de orden, siendo este último el elemento que nos permitirá mejorar de manera drástica los resultados obtenidos al aplicar Transformers en series temporales.

A partir de ahora, nos centraremos en estudiar a fondo su aplicación y ajuste para tratar de mejorar los resultados obtenidos en series temporales.

2.2.1. Conceptos previos: token y embedding

En los modelos basados en Transformers, uno de los pasos principales antes de dotar la información a la arquitectura es transformar los datos en un formato numérico comprensible para la red neuronal. Esto es especialmente importante con textos, ya que tenemos la información codificada en caracteres.

Para construir las entradas, se emplea el concepto de *token*, que representa la unidad mínima de entrada al modelo. Su tamaño depende del problema, y no existe una unidad cocnreta: podría ser una secuencia completa, una subpalabra, un carácter o incluso un símbolo de puntuación.

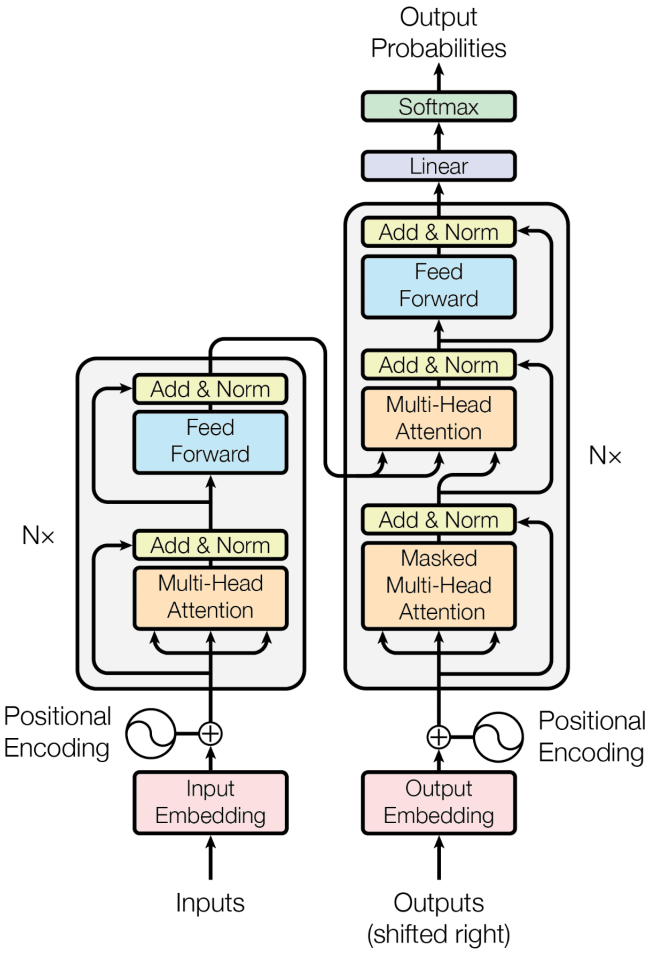

Figura 2.9: Transformer: modelo de arquitectura encoder-decoder [17]

En su definición original para textos, el proceso de tokenización convierte una secuencia de texto en una lista de tokens, cada uno de los cuales es mapeado a un identificador entero a través de un vocabulario predefinido. Dicho mapeo sirve como base para generar una representación vectorial de cada token, conocida como **embedding**. Esta representación no se limita a un único vector, sino que resulta de la combinación de distintos componentes que aportan información contextual adicional.

En los modelos Transformer clásicos, cada token es transformado en un vector de entrada única de manera aditiva:

- **Token embedding:** representa el significado semántico del token, aprendido durante el entrenamiento.
- **Positional encoding:** añade información sobre la posición del token en la secuencia, necesaria dado que los Transformers no procesan los datos de manera secuencial.

- **Segment embedding:** es un elemento opcional, el cual indica a qué segmento o frase pertenece cada token, útil en tareas de relación entre pares de oraciones.

La suma de estos tres vectores produce la entrada final que el modelo utiliza en la capa de atención, y permite que el modelo tenga acceso simultáneo tanto al contenido semántico como a la estructura secuencial del texto.

En el caso de las series temporales, el término token puede ser algo menos intuitivo de entender, ya que no tenemos palabras como tal. Pero cada punto temporal o ventana de datos puede considerarse una unidad equivalente. Aquí, el *token embedding* suele derivarse directamente de los valores numéricos observados, y es la codificación posicional el elemento que cobra relevancia, ya que los valores de tiempo no están implícitos en la secuencia misma, sino que deben incorporarse explícitamente mediante un mecanismo que represente la fecha u hora.

De esta forma, podemos apreciar el paralelismo entre ambos dominios: mientras que en texto los embeddings representan palabras y estructuras lingüísticas, en series temporales encapsulan magnitudes numéricas y patrones temporales, pero dejando al margen la forma en la que construimos el embedding, en ambos casos buscamos representar y modelar las relaciones contextuales a lo largo del tiempo o de la secuencia.

2.2.2. Arquitectura: encoder y decoder

La arquitectura Transformer se construye sobre un esquema de tipo *encoder–decoder*. Para comprenderlo mejor, podemos tomar como referencia los autoencoders, pero teniendo en cuenta que ahora nos centramos en el modelado de secuencias; mientras que un autoencoder clásico codifica una entrada en una representación latente para luego reconstruirla, el Transformer aplica esta idea para tareas de transformación de secuencia a secuencia: el encoder transforma la secuencia de entrada en una representación intermedia rica en contexto, y el decoder genera la secuencia de salida basada en dicha representación.

2.2.2.1. Encoder

El encoder es la parte del modelo que toma como entrada una secuencia de vectores (habitualmente embeddings de palabras o características de entrada), y produce una secuencia de vectores de igual longitud que contienen información contextualizada. Para aprovechar el paralelismo, está compuesto por una pila de N bloques idénticos, donde cada uno incluye:

- Un mecanismo de *self-attention*, que permite a cada elemento de la secuencia atender a todos los demás y capturar relaciones globales, y cuyo funcionamiento profundizaremos posteriormente.
- Una red neuronal totalmente conectada, aplicada de manera independiente en cada posición.

Ambas subcapas están envueltas en una conexión residual seguida de una normalización por capas, de manera que no se produzca desvanecimiento de gradientes cuando tomamos largas secuencias de entrada, y además, se mejore la convergencia gracias al efecto de la normalización.

2.2.2.2. Decoder

El decoder también consta de N bloques, pero su estructura es ligeramente más compleja, ya que debe generar una secuencia salida de manera autoregresiva, que mantega la información contextual del orden. Cada bloque del decoder contiene:

- Un *self-attention* enmascarado. Su pricipal función es que el modelo, en un instante del proceso de predicción, pueda acceder a la información de tokens anteriores o a sí misma, pero no ver la información de tokens futuros.
- Un mecanismo de *cross-attention*, que permite que cada token del decoder acceda a toda la información producida por el encoder, lo cual es esencial cuando existe una fuerte relación entre la información de entrada y la salida que se desea producir.
- Una red totalmente conectada de tipo feed-forward, similar a la del encoder.

Cada paso de generación en el decoder depende tanto de los tokens generados hasta el momento como de la representación completa de la entrada, por lo que se está aprovechando la información de orden existente en los datos y el conocimiento disponible, siendo este proceso totalmente compatible con la aplicación en series temporales.

Teniendo en cuenta el anterior símil del Transformer con los Autoencoders, es importante tener clara la diferencia para no entrar en cofusión. Aunque los Transformers no buscan reconstruir directamente la entrada como en los autoencoders clásicos, la estructura encoder–decoder cumple un propósito análogo: transformar información de entrada en una forma útil y compacta que el decoder pueda explotar para generar la salida deseada. A esta diferencia, se le suma también la utilidad de la información contextual: para mejorar el rendimiento, y facilitar el proceso de aprendizaje, el modelo hace uso de información que permita identificar relaciones de orden y facilitar el proceso de recurrencia de manera implícita en los datos.

2.2.3. Mecanismo de atención

El núcleo del Transformer es el mecanismo de *self-attention*. Este se basa en la definición de 3 subespacios: Queries (Q), Keys (K) y Values (V). Profundizaremos en ver cómo se relacionan estos elementos desde un punto de vista formal.

La atención entre posiciones se calcula como una similitud entre queries y keys, seguida de una ponderación de los values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Es frecuente encontrar en el denominador el término $\sqrt{d_k}$, ya que su principal función es evitar que los productos escalares sean excesivamente grandes, y actúa a modo de normalización para evitar problemas con los gradientes.

De por sí solo, este mecanismo es el principal elemento en el funcionamiento de los transformers. Pero podemos mejorar su potencial si lo paralelizamos. Es aquí cuando entra en

acción el mecanismo de *multi-head attention*, donde se realizan múltiples operaciones de atención en paralelo, cada una con sus propias proyecciones aprendidas, permitiendo al modelo aprender distintas relaciones entre tokens en diferentes subespacios de representación.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Posteriormente, estos valores se concatenan, y podemos obtener el resultado que buscábamos de manera paralela.

2.2.4. Importancia del Positional Encoding

Dado que el Transformer no incorpora mecanismos recurrentes ni convolucionales, carece de una forma intrínseca para captar el orden relativo o absoluto de los elementos que recibe como entrada. Para superar esta limitación, los autores del modelo original introdujeron el concepto de *positional encodings*, vectores que se suman a las representaciones embebidas de cada token en función de su posición dentro de la secuencia. Estos vectores proporcionan una señal explícita de la ubicación de cada elemento, permitiendo que el modelo incorpore información posicional en el cálculo de la atención.

En el modelo original de transformer, el cual denominaremos en futuras secciones como Transformer *vanilla*, el positional encoding se definió mediante funciones periódicas basadas en seno y coseno con distintas frecuencias, cuya formulación es la siguiente:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Este diseño ofrece varias ventajas: al ser funciones periódicas y deterministas, no requieren parámetros adicionales a entrenar y permiten que el modelo generalice a secuencias más largas que las vistas durante el entrenamiento. Además, facilitan que la red aprenda a reconocer patrones posicionales y relaciones relativas entre elementos, aprovechando las propiedades de las ondas seno y coseno para captar distancias y ciclos.

No obstante, aunque esta técnica es adecuada para procesar texto, donde el orden es fundamental, su aplicabilidad en series temporales es más compleja y no siempre ha de producir un buen resultados. En las series temporales, la posición en la secuencia no es solo un índice ordinal, sino que representa una dimensión temporal real, ligada a un instante específico. Esto implica que los elementos de la secuencia deben ordenarse dado su timestamp, pero también contienen información contextual inherente al tiempo: estacionalidad, tendencias que evolucionan a lo largo del tiempo, y eventos que ocurren en momentos concretos.

En las series temporales, cuando queremos detectar sucesos concretos o anomalías, para posteriormente modelar y crear predicciones largo plazo, toda la información no suele estar concentrada en único timestamp. Por ejemplo, si tenemos un dataset muestreado por horas, puede ser frecuente y viable que para comprender el fenómeno, debamos mirar no sólo la información de una hora concreta, sino del día completo. Por tanto, la codificación posicional tradicional basada en seno y coseno no captura explícitamente estas características, pues si bien es capaz de establecer un orden en el punto de vista global de los datos, no

es capaz de contener y explicar información local de estos. Estamos perdiendo información contenida en patrones temporales más complejos.

Esto plantea un desafío fundamental: la necesidad de diseñar o adaptar mecanismos de *positional encoding* que sean capaces de integrar no solo la información ordinal, sino también las propiedades temporales explícitas que caracterizan a estos datos. En el estado del arte, podemos encontrar ya algunos enfoques en esta dirección:

- **Encodings aprendibles**. Permiten al modelo optimizar vectores posicionales específicos para las características del conjunto de datos. Tienen la ventaja de facilitar el proceso de entrenamiento sin necesidad de especificar parámetros de forma explícita, pero se debe tener especial cuidado con la inicialización y la normalización de valores, para evitar problemas de crecimiento desproporcionado en ciertos parámetros.
- **Codificaciones basadas en atributos del timestamp**. Consiste en incluir explícitamente variables de tiempo (hora, día de la semana, mes, año) que añaden señales temporales al embedding, de forma que se pueda usar su información para el orden, pero también como un atributo más del problema.
- **Transformaciones basadas en series de Fourier o wavelets**. Capturan patrones periódicos más complejos y multi-escalares, aunque, dada su naturaleza basada también en funciones trigonométricas, acercan la codificación a los datos pero sigue siendo un enfoque similar al original con sus correspondientes limitaciones.

Aunque se han logrado mejoras y avances sobre la definición vanilla del encoding posicional gracias a estas propuestas, la mayoría de estas aproximaciones comparten una limitación estructural: la escasa vinculación semántica y local entre los datos de entrada y su codificación posicional.

De manera más detallada,frecuentemente los vectores que representan la posición suelen ser generados de forma independiente al contenido real de la secuencia. Esto significa que dos tokens o timestamp con significado similar pero ubicados en posiciones diferentes reciben representaciones posicionales completamente distintas, sin ningún tipo de correspondencia semántica entre ellas. En series temporales, puede suponer un grave problema: encontrar patrones como la estacionalidad pueden ser claves en el modelado, y con este enfoque, este tipo de ciclos sería prácticamente imposible de detectar.

Un ejemplo sencillo podría ser la estimación del consumo eléctrico. En una serie que recoge datos meteorológicos diarios durante varios años, los valores correspondientes a una hora concreta de diferentes días podrían compartir patrones similares; es habitual encontrar por ejemplo consumos superiores en las horas cercanas al almuerzo o la cena, y de menor uso en altas horas de la madrugada. Pero, si la codificación posicional no reconoce esta regularidad por tratarse de posiciones globalmente distintas en la secuencia, el modelo no podrá aprovechar dicha repetición estructural, limitando su capacidad de modelado.

Además, en muchos contextos, los eventos relevantes están más determinados por la posición relativa y el entorno local inmediato que por una posición absoluta dentro de la secuencia. La falta de mecanismos que conecten directamente el significado de los datos con su ubicación limita el poder expresivo de la red y su capacidad para realizar generalizaciones

útiles. Esta problemática refuerza la necesidad de proponer codificaciones posicionales con mayor semántica, más contextualizadas, que integren explícitamente el contenido de los datos y su estructura temporal.

A lo largo de este trabajo, exploraremos esta línea de mejora como eje central de nuestra nueva metodología de codificación.

2.2.5. Modelos aplicados al forecasting de series temporales

En el estado del arte, podemos encontrar multitud de modelos que proponen adaptaciones y soluciones alternativas al uso de Transformers para series temporales. Algunas, son soluciones propias de lenguaje natural adaptadas a su uso en series temporales, mientras que en otros casos tratan de basarse en conceptos básicos anteriormente explotados en series, como la descomposición o el uso de la autocorrelación, pero empleando Transformers como vía de aprendizaje.

Concretamente, comenzaremos analizando variantes del modelo Transformer original que tuvieron un gran impacto en los modelos adaptados, como TransformerXL [6], Reformer [12] y LogTrans [13], para posteriormente analizar el funcionamiento de Informer [18], Autoformer, FEDFormer, y modelos más recientes como PatchTST y TimesNet.

2.2.5.1. Variantes de Transformer y mejoras previas

El modelo original de Transformer propuesto en *Attention is All You Need* [17] ha sido objetivo de muchas investigaciones que han buscado optimizar tanto la capacidad de modelado como la eficiencia computacional. Si bien su principal campo de aplicación siguió siendo el lenguaje natural, dichas innovaciones han permitido posteriores mejoras a las aplicaciones de Transformers en series temporales.

De entre las características deseables, se han buscado continuamente mejoras de rendimiento para dominios de larga secuencia, relaciones de dependencia a múltiples escalas y mejoras a nivel computacional, tratando de reducir la complejidad de ejecución pero sacrificando el mínimo rendimiento posible.

Transformer-XL (2019)
Una de las primeras propuestas realizadas es el Transformer-XL [6]. Esta profundiza sobre el concepto de positional encoding relativo, reemplazando el esquema absoluto original por uno que codifica distancias relativas entre tokens. Esto no solo permite capturar mejor dependencias a largo plazo, sino que también hace posible extender el contexto más allá de la ventana de entrenamiento, reutilizando estados ocultos de segmentos previos y patrones locales.

En series temporales, esta técnica tiene una gran relevancia en la detección de patrones estacionales, ya que nos permite encontrar patrones, donde el significado de la relación temporal radica sobre todo en dónde se ubica un evento concreto dentro del período estacional, y no tanto en su posición absoluta en la secuencia.

LogTrans (2019)

La variante LogTrans[13] se centra en reducir el coste computacional del cálculo de atenciones, proponiendo un mecanismo de atención diseñado para reducir el coste computacional del *self-attention* tradicional, denominado *log-sparse attention*. La idea busca su fundamento en secuencias de larga extensión, donde se argumenta que no todas las interacciones entre elementos son igualmente relevantes: mientras que ciertas posiciones requieren una atención densa hacia elementos cercanos, por razones como lazos locales estrechos, otras mantienen dependencias más espaciadas que solo necesitan ser consideradas de forma ocasional. Para lograrlo, el mecanismo *log-sparse* selecciona las posiciones a las que cada token presta atención siguiendo una distribución logarítmica. Esto significa que las posiciones cercanas se incluyen de forma continua y densa, asegurando que las dependencias locales estén bien representadas, mientras que las posiciones más lejanas se muestrean con menor densidad, manteniendo aún así una cobertura global de la secuencia pero con un número significativamente reducido de interacciones.

Empleando el muestreo, la complejidad se reduce de $O(n^2)$ a aproximadamente $O(n \log n)$, lo que permite manejar secuencias mucho más largas sin que el coste computacional o el consumo de memoria GPU se disparen. Sin embargo, esta estrategia tiene ciertos riesgos, sobre todo si lo aplicamos a series temporales: usar un patrón de muestreo conlleva el riesgo de perder información relevante, ya que dicha relación no está siendo recogida. Este problema puede resultar especialmente crítico en series temporales donde ciertas fluctuaciones clave puedan ser omitidas y provoquen una pérdida de rendimiento considerable, o bien, durante la búsqueda de anomalías en las series, tarea la cual está bastante vinculada también con la predicción.

A pesar de esta limitación, *log-sparse attention* demostró que es posible reducir el coste de la atención a cambio de un pequeño incremento del error en series, siendo más notable en series de larga extensión. De hecho, esta idea abrió la puerta a modelos como *Informer*, uno de los modelos de *forecasting* basados en Transformers más influyentes en el estado del arte, en el que se adopta el enfoque probabilístico para seleccionar las claves más relevantes. Más adelante, examinaremos su funcionamiento en detalle, centrándonos en su aplicación específica a la predicción de series temporales.

Reformer (2020)

Reformer [12] aborda otro de los cuellos de botella principales de los Transformer: el coste $O(n^2)$ del cálculo de atenciones. En secuencias muy largas, el orden cuadrático puede aumentar considerablemente los tiempos de ejecución, hasta el punto de que en datasets de gran tamaño, puede suponer un obstáculo de cara a poder ser ejecutado. Dicho orden de complejidad proviene de realizar el producto entre los diferentes tokens de entrada, pues se debe realizar el producto de cada elemento por el resto, y esto depende directamente del tamaño de los datos.

Para tratar de reducir dicha complejidad, Reformer introduce varios conceptos fundamentales que mejoran la eficiencia sin sacrificar la calidad del aprendizaje.

- **Locality-Sensitive Hashing (LSH):** Esta técnica permite aproximar el cálculo de la atención agrupando tokens similares mediante el uso de funciones hash, diseñadas para preservar la proximidad en el espacio vectorial. En lugar de calcular la atención

entre todos los pares de tokens, el modelo solo la calcula dentro de unos grupos, llamados *buckets*, que están formados por tokens cercanos en su representación (ver figura 2.10).

Gracias a ello, podemos reducir el coste de $O(n^2)$ a aproximadamente $O(n \log n)$, al limitar las interacciones a subconjuntos relevantes. Además, LSH favorece que la atención se concentre en relaciones locales y semánticamente relevantes, lo cual favorece también colateralmente a su aplicación en series temporales y la detección de estacionalidad.

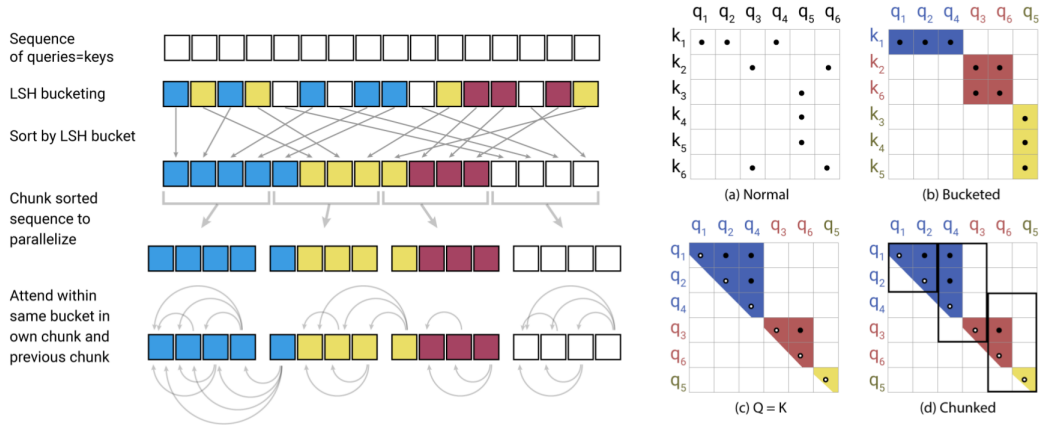

Figura 2.10: Reformer: funcionamiento del mecanismo de creación de *buckets* [12]

- **Codificación reversible:** Continuando con las mejoras de eficiencia, Reformer modifica la arquitectura para emplear redes residuales reversibles que permiten reconstruir los estados intermedios de cada capa durante *backpropagation*, en lugar de almacenarlos durante el *forward*. Esto reduce drásticamente la memoria requerida para entrenar modelos profundos y secuencias largas. La invertibilidad de las capas permite recuperar el input de cada capa a partir de su output, lo que implica una compensación entre tiempo computacional (por el costo de recalcular estados) y memoria. Su impacto puede apreciar en mayor medida cuando el conjunto de entrada dispone miles o millones de instancias o pasos a analizar.

Ambas mejoras nos permiten realizar un mejor aprovechamiento de la memoria de video, mejorando la reutilización de cálculos y permitiendo así usar conjuntos de datos de mayor tamaño, aunque debemos de tener en cuenta que agrupar por similaridad introduce errores numéricos en el cálculo de la atención.

2.2.5.2. Informer (2020)

El modelo **Informer** es una de las primeras propuestas que adapta de forma específica la arquitectura Transformer al dominio de las series temporales, tratando de modificar su funcionamiento para adaptarse a las limitaciones detectadas en el contexto de series temporales. Su diseño se orienta al problema de *Long Sequence Time-Series Forecasting* (LSTF), para predecir horizontes temporales largos a partir de entradas igualmente extensas, lo que plantea

dificultades de escalabilidad y eficiencia tanto en entrenamiento como en inferencia.

En el uso directo de Transformers canónicos sobre series temporales, surgen tres problemas fundamentales, algunos de ellos ya estudiados en el apartado anterior:

- **Alta complejidad computacional y de memoria**. La implementación estándar del mecanismo de atención presenta un coste de $O(L^2)$ en secuencias de longitud L, debido a la necesidad de calcular interacciones entre todos los pares de posiciones. Esto limita su uso en horizontes de predicción largos o datos de alta frecuencia, ya que limita su escalabilidad.
- **Cuellos de botella con la memoria**. En arquitecturas encoder–decoder profundas, donde tenemos varias capas, podemos tener problemas con la complejidad espacial, ya que disponiendo de J capas, la complejidad total será de $O(J \cdot L^2)$, limitando así la posibilidad de procesar largas series.
- **Baja velocidad de inferencia**. En la inferencia original propuesta por Transformer, el decoding se realiza paso a paso, aumentando los tiempos de inferencia hasta el punto de ser tan lenta como los modelos basados en RNN. Por tanto, debemos de encontrar un nuevo mecanismo capaz de simplificar y adaptar este proceso.

Para mitigar estas limitaciones, Informer introduce tres innovaciones clave:

1. **ProbSparse Self-Attention**. Se trata de un mecanismo de atención dispersa basado en la hipótesis de que, en muchas tareas de series temporales, solo unas pocas *queries* concentran la mayor parte de la relevancia en la atención. Al muestrear y priorizar dichas *queries*, se reduce el coste computacional de $O(L^2)$ a $O(L \log L)$ sin degradar la capacidad de modelar dependencias a largo plazo. Para ello, se emplea el mismo procesamiento visto en LogTrans [13].
2. **Self-Attention Distilling**. Se trata de una operación jerárquica que filtra las atenciones dominantes y reduce progresivamente la dimensión temporal entre capas. Este proceso actúa como una mecanismo de compresión, donde se preservan los patrones relevantes y disminuyendo el consumo de memoria, permitiendo que modelos más profundos procesen secuencias largas sin sobrecargar la GPU, logrando una complejidad reducida de $O((2-\epsilon)L \log L)$.
3. **Generative Decoder**. Para solucionar el problema de la inferencia paso a paso, se propone un decodificador capaz de generar toda la secuencia de salida en un único paso (*one-forward*). Este enfoque no solo acelera la inferencia, sino que también reduce la acumulación de errores entre pasos, mejorando la estabilidad en horizontes de predicción extensos.

En conjunto, estas mejoras convierten a **Informer** en un modelo especialmente interesante para escenarios de predicción multivariados y datos de gran volumen, manteniendo la capacidad de capturar tanto dependencias locales como globales reduciendo el impacto en memoria y mejorando la eficiencia.

A continuación, haremos énfasis en cada una de estas 3 propuestas, y analizaremos por partes su arquitectura (ver figura 2.11).

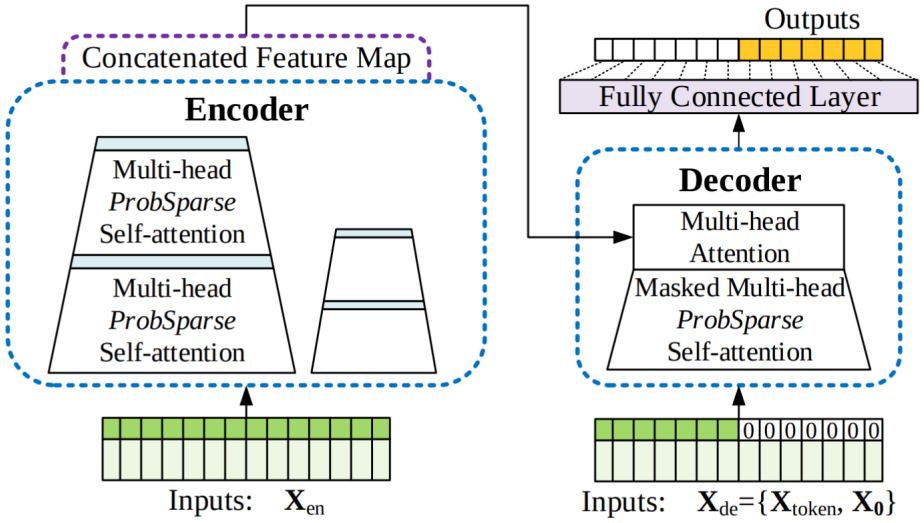

Figura 2.11: Informer: arquitectura completa del modelo

Mecanismo de atención

El mecanismo de *self-attention* tradicional se definía como:

$$A(Q,K,V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V,$$

Donde la implementación canónica requiere computar todas las similitudes par a par $QK^\top$, con coste temporal y de memoria $O(L^2)$.

Informer parte de la observación empírica de que la distribución de probabilidades de atención suele mostrar una *cola larga*: pocas consultas concentran la mayor parte de la energía de atención, mientras que la mayoría producen señales débiles. Para aprovechar esta propiedad, se define una medida de "dispersión de consulta" que cuantifica la propensión de una consulta q_i a generar picos de atención:

$$M(q_i,K) = \max_j \left\{\frac{q_i k_j^\top}{\sqrt{d}}\right\} - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{q_i k_j^\top}{\sqrt{d}},$$

donde L_K es el número de claves (en encoder–decoder L_Q y L_K pueden diferir). Intuitivamente, $M(q_i,K)$ mide la diferencia entre el máximo score y el score medio: valores grandes indican consultas con picos pronunciados (es decir, consultas "informativas").

Esencialmente, este mecanismo consta de dos fases:

1. **Selección de consultas relevantes:** se estima $M(q_i,K)$ mediante muestreo parcial de claves con LogSparse y se conservan solo las $u = c \cdot \ln L_Q$ consultas con mayor valor. Para ello, para cada consulta q_i, se calcula una estimación de su medida de dispersión

$$\hat{M}(q_i,K) = \max_{j\in\mathcal{S}_i}\left\{\frac{q_i k_j^\top}{\sqrt{d}}\right\} - \frac{1}{|\mathcal{S}_i|}\sum_{j\in\mathcal{S}_i}\frac{q_i k_j^\top}{\sqrt{d}},$$

donde $\mathcal{S}_i \subset \{1, \ldots, L_K\}$ es un subconjunto de claves obtenido mediante muestreo *log-sparse*, es decir, seleccionando índices de forma que la densidad de puntos crece logarítmicamente con la distancia temporal. Este muestreo permite aproximar $M(q_i, K)$, tratando de mantener con alta probabilidad las claves más informativas.

Finalmente, se ordenan las consultas según $\hat{M}(q_i, K)$ y se conservan solo las $u = c \cdot \ln L_Q$ de mayor valor, donde c establece qué proporción seleccionar. A mayor valor, mayor precisión, pero mayor coste.

2. **Cálculo selectivo de atención:** solo para esas u consultas se computa la atención completa frente a todas las claves, mientras que las no seleccionadas reciben una aproximación suave o interpolada. Esta podría ser un valor medio, ya calculado anteriormente al realizar la búsqueda de las instancias informativas.

Con $u = O(\log L)$, el coste se reduce de $O(L^2)$ a $O(L \log L)$ por cada head del modelo, consiguiendo una reducción considerable de la complejidad.

3 Conjuntos de datos: selección y preprocesado

4 Modelos de encoding posicional y entorno de trabajo

5 Análisis y estudio de encodings sobre las bases de datos

6 Conclusiones y trabajos futuros

Bibliografía

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models, 2022.

[2] G. E. P. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Inc., USA, 1990.

[3] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 1970.

[4] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976. AirPassengers dataset.

[5] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3–73, 1990.

[6] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

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

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[9] R. J. Hyndman and G. Athanasopoulos. Forecasting: Principles and practice - stationarity and differencing, 2021. Accedido el 31 de julio de 2025.

[10] H. Irani and V. Metsis. Positional encoding in transformer-based time series models: A survey, 2025.

[11] A. Kibar. U.s. housing starts. u.s. building permits. https://blog.techcharts.net/index.php/2012/06/19/u-s-housing-starts-u-s-building-permits/, 2012. Consultado el 25 de julio de 2025.

[12] N. Kitaev, Łukasz Kaiser, and A. Levskaya. Reformer: The efficient transformer, 2020.

[13] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[14] I. D. Mienye, T. G. Swart, and G. Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9), 2024.

[15] S. J. Taylor and B. Letham. Prophet: Forecasting at scale, 2017. Facebook Research Blog. Revisado el 30/07/2025.

[16] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[18] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting, 2021.