Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Predicción de la Demanda de Movilidad Espacio-Temporal del Transporte Público mediante Transformers

Autor(a): Koldo Moya Iratxeta
Tutor(a): Damiano Zanardini

Madrid, 07 - 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Predicción de la Demanda de Movilidad Espacio-Temporal del Transporte Público mediante Transformers

07 - 2024

Autor(a): Koldo Moya Iratxeta
Tutor(a): Damiano Zanardini
Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En el 2017, Ashish Vaswani junto a otros tanto profesionales introdujeron los *Transformers* en el ámbito del Procesamiento de Lenguaje Natural. Estos avances mejoraron los resultados de los modelos de dicho ámbito en cuanto a rapidez de entrenamiento, capacidad de trabajar con dependencias a corto plazo y los limites de memoria que tenían las Redes Neuronales Recurrentes. Los *Transformers* estaban principalmente compuestos de mecanismos de atención que ayudaban al modelo a centrarse en las partes mas importantes de las secuencias de entrada. Estos modelos *Transformers* fueron principalmente desarrollados para realizar tareas relacionadas con el campo de la traducción automática sin embargo, debido a su gran capacidad para tratar con dependencias a largo plazo, se empezaron a utilizar para tareas relacionadas con la predicción de series temporales. En [Prado-Rujas *et al.*, 2023] se introdujo un problema basado en la predicción de la demanda de movilidad del transporte público en diferentes series temporales ya que, desde hace décadas que ciudades como Nueva York o Chicago sufrían el problema de congestión de tráfico. Por lo tanto, este trabajo se lleva a cabo mediante un *Transformer* por dos motivos diferentes. El primero, se basa en que este tipo de modelo ha sido previamente utilizado para tareas de predicción similares. El segundo de los motivos es que la predicción de la demanda de movilidad del transporte público podría ayudar a gestionar las calles de mejor manera. Es decir, se va a realizar dicha predicción en diferentes series temporales mediante un modelo *Transformer* desarrollado específicamente para esta tarea denominado “Spatio Temporal Transformer for Mobility Demand Forecasting” (STT-MDF).

Abstract

In 2017, Ashish Vaswani along with other professionals introduced *Transformers* in the field of Natural Language Processing. These advances improved the results of the models in this field in terms of speed of training, ability to work with short-term dependencies and the memory limits that Recurrent Neural Networks had. The *Transformers* were mainly composed of attention mechanisms that helped the model focus on the most important parts of the input sequences. These *Transformers* models were mainly developed to perform tasks related to the field of machine translation, however, due to their great ability to deal with long-term dependencies, they began to be used for tasks related to time series prediction. In [Prado-Rujas *et al.*, 2023] a problem was introduced based on the prediction of public transport mobility demand in different time series since, for decades, cities such as New York or Chicago have suffered from the problem of traffic congestion. Therefore, this work is carried out using a *Transformer* for two different reasons. The first one is based on the fact that this type of model has been previously used for similar prediction tasks. The second reason is that predicting public transport mobility demand could help manage streets in a better way. That is, said prediction will be made in different time series using a *Transformer* model developed specifically for this task called "Spatio Temporal Transformer for Mobility Demand Forecasting" (STT-MDF).

Tabla de contenidos

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La movilidad urbana es un desafío creciente para las grandes ciudades que sufren congestiones de tráfico, contaminación e ineficiencias en el transporte público. Los atascos se han convertido en un hecho cotidiano para muchos, y el medio ambiente ha estado pidiendo cambios desde hace tiempo. Se debe tener en cuenta que en las grandes ciudades, el medio de transporte más usado para los desplazamientos de trabajo u ocio es el coche personal [Fiorello *et al.*, 2015], cuya ocupación media es de 1,7 personas por coche. Por lo tanto, sumando este dato a la gran cantidad de vehículos de transporte publico que circulan por la ciudad, se obtiene uno del los motivos por el que las carreteras de las grandes ciudades están habitualmente colapsadas. Para abordar estos problemas e intentar evitarlos, es fundamental contar con las herramientas para planificar y administrar adecuadamente el sistema de transporte. En este sentido, la previsión de las necesidades de movilidad es una tarea crítica que puede ayudar a optimizar la oferta de transporte público y mejorar la calidad de vida de los ciudadanos.

Entre estas herramientas, se encuentra la utilizada a lo largo de este trabajo de fin de máster, los *Transformers*. Estos, son una arquitectura de red neuronal diseñada originalmente para el sector del procesamiento del lenguaje natural que han demostrado una gran eficiencia en la predicción de secuencias en varios dominios. A diferencia de las redes neuronales recurrentes tradicionales que procesan flujos uno tras otro, los *Transformers* utilizan atención multidimensional para procesar todas las entradas de flujo simultáneamente. El termino *atención* en este trabajo es muy importante, ya que la propia estructura de los modelos *Transformers* evita la recurrencia y se basa por completo en un mecanismo de atención para establecer dependencias globales entre la entrada y la salida. Estas dependencias creadas entre las secuencias de entrada y de salida son una herramienta muy poderosa debido a que son de gran ayuda a la hora de llevar a cabo la predicción de secuencias espacio-temporales, como la demanda del transporte publico, y este trabajo estará basado en un modelo de *Transformer* espacio-temporal (STT-MDF).

El problema que se busca resolver a lo largo de este trabajo puede ser formulado como una previsión de series temporales múltiples, es decir, dada una secuencia temporal de datos históricos de demanda de movilidad, predecir la demanda futura para determinados horizontes. Para esto, se han escogido diferentes datos sobre el uso de taxis y bicicletas y sus viajes en la ciudad de Chicago ubicaba en Estados Unidos en el estado de Chicago. Dentro de la ciudad de Chicago se escogerán los

datos dentro de una "mesh-grid" o rejilla de malla la cual sera una cuadricula que limitará el tamaño de la ciudad.

1.1. Motivación

La principal motivación por la que se ha llevado a cabo este trabajo es el reto de adaptar la investigación "Combining heterogeneous data sources for spatio-temporal mobility demand forecasting" [Prado-Rujas *et al.*, 2023] a un modelo *Transformer*. En este estudio, se busca predecir la demanda del transporte público de la ciudad de Chicago a través de un modelo espacio temporal llamado "Spatio-Temporal Mobility Demand Forecaster" (ST-MDF). Por lo tanto, sabiendo que los *Transformers* trabajan de una manera diferente frente a los datos, esto sirve como gran motivación y como reto para llevar a cabo este trabajo.

Este tipo de investigaciones puede tener una gran relevancia en la sociedad actual, ya que predecir la demanda de movilidad del transporte público es un problema crucial en el diseño y planificación del propio transporte urbano. Al obtener una buena predicción de demanda de movilidad, se pueden alcanzar importantes beneficios como la mejora de la calidad del servicio del transporte público, y la reducción de la congestión vehicular en las carreteras en grandes ciudades como Chicago.

1.2. Objetivos

El objetivo principal de este trabajo de fin de máster es llevar a cabo la implementación de un modelo *Transformer* espacio-temporal para conseguir una predicción lo más precisa posible de la demanda de movilidad del transporte público de una ciudad en particular, en este caso Chicago. Para poder llegar a dicha implementación, se utilizaran datos del uso y movilidad de los taxis y bicicletas en la ciudad de Chicago.

Para llevar a cabo este objetivo principal, existen diferentes pasos a seguir que podrían considerarse objetivos secundarios:

- El primero de los objetivos secundarios para llevar a cabo el objetivo principal previamente mencionado y por tanto, llevar a cabo este trabajo de fin de máster, es la recopilación de datos con los que va a trabajar el modelo STT-MDF. Este objetivo tiene gran importancia ya que existen innumerables conjuntos de datos sobre la movilidad del transporte público de las ciudades grandes como Chicago y es necesario entender estos conjuntos de datos y saber interpretarlos para más adelante obtener buenos resultados en los experimentos a realizar.
- Otro objetivo secundario es el desarrollo de un primer modelo STT-MDF de prueba que sea capaz de soportar ejecuciones con los datos anteriormente escogidos.
- Una vez el objetivo de la recopilación de datos se ha completado y a su vez, un primer modelo de prueba desarrollado, un objetivo sumamente vital es el análisis de los resultados. De esta manera, se podrán ver y modificar los puntos débiles del modelo STT-MDF y se podrán ajustar los parámetros o hiperparámetros pertinentes.
- En relación con el objetivo anterior, es necesario comparar los resultados obtenidos con los resultados de modelos desarrollados por otros investigadores.

En este caso, será útil la comparación con modelos de *Transformers* como los modelos que no tengan que ver con los *Transformers*.

1.3. Contextualización

El objetivo principal de este apartado es detallar que las palabras en Ingles y los tecnicismos que van a aparecer a lo largo de la memoria se van a escribir entre comillas para que la lectura de este documento sea lo más sencilla y amena posible. El motivo de esta decisión se debe a que se comprende que los lectores de la memoria son personas cualificadas en sus respectivos campos por lo que están acostumbrados a leer una infinidad de *papers* diariamente y la gran mayoría de estas investigaciones se encuentran en Inglés. Por lo tanto, es entendible que dichos lectores normalicen estas palabras en Inglés antes que en Castellano. En cuanto a los tecnicismos, se realizará una breve explicación junto a cada uno de estos para que el lector no pierda el hilo del apartado que se encuentra leyendo en cada momento.

1.4. Estructura del documento

En este apartado del capítulo de la introducción se va a explicar brevemente el contenido de cada uno capítulos que aparecerán a lo largo de la memoria.

- El primer capítulo, es decir, el capítulo actual, está basado en una introducción al tema a tratar para poner en contexto del trabajo al lector. Dentro se verán diferentes apartados como los objetivos, los trabajos relacionados o la motivación de llevar a cabo este proyecto.
- El segundo capítulo, constará del estado del arte del trabajo en el que se mencionarán los trabajos más relevantes de los diferentes apartados que de una forma u otra están relacionados con los *Transformers*, las series temporales, las dependencias espacio-temporales, etc.
- El tercer capitulo, describe detalladamente el modelo STT-MDF, mencionando su arquitectura y la importancia de cada una de las partes de este. Además, se menciona la implementación llevada a cabo para poder desarrollar dicho modelo y se explicarán los experimentos realizados junto a los resultados obtenidos.

Capítulo 2

Estado del arte

2.1. Introducción

En este capítulo se mencionarán las características más importantes de los *Transformers*, *Transformers* espacio-temporales y de la predicción de la demanda de movilidad del transporte público que han ido surgiendo a lo largo de los últimos años en diferentes investigaciones relacionadas con el tema a tratar.

La predicción de la demanda de movilidad del transporte público en ciudades grandes trata en estimar condiciones futuras de tráfico o espaciales (la densidad de vehículos) y condiciones temporales (vacaciones, meteorología). Predecir estas condiciones de tráfico desde siempre ha sido una tarea sumamente difícil debido a la complejidad de modelar las dependencias espacio-temporales [[Davis *et al.*, 2019]; [Wu *et al*, 2018]] y es por ello que la congestión de tráfico ha sido un problema desde la existencia de los vehículos en las grandes ciudades. En el caso de las ciudades más grandes de Australia, en 2015 se calculó que para evitar estas congestiones el coste era de 16.500 millones de dólares, superando la estimación del año 2010 de 12.800 millones de dólares. Además, en un investigación realizada por [David, 2015] se estimó que este coste podía subir a 30 billones de dólares en 2030. Por ello, Singapur implementó ciertas regulaciones en el número de vehículos en las carreteras [BBC News, 2017], pese a que era algo totalmente inviable para los países que tenían sistemas de transporte público deficientes.

Más adelante, con el objetivo de ponerle fin a dichas congestiones de tráfico y de poder tratar con las diferentes dependencias espacio-temporales, aparecieron modelos basados en "Recurrent Neuronal Networs" (RNNs) [Schmidt, 2019]. Entre estos se encontraban tanto la unidad recurrente controlada (GRU) [Jin *et al.*, 2018] y la memoria a largo plazo (LSTM), que podían ser utilizados de manera efectiva para capturar dependencias temporales [[Cui *et al.*, 2018]; [Jin *et al.*, 2018]]. De hecho, [Shi *et al.*, 2015] propusieron un modelo LSTM convolucional aplicado a la predicción de tráfico en el que el flujo de tráfico en cada paso del tiempo se iba alimentado recursivamente en una arquitectura LSTM [Cui *et al.*, 2018]. Un par de años después, se propuso una nueva forma de implementar un modelo RNN convolucional en el que los operadores convolucionales de difusión gráfica modelaban las dependencias espaciales, mientras que se utilizaba una GRU para capturar las dependencias temporales. Sin embargo, pese a que los modelos RNN podían capturar eficientemente

las dependencias temporales, estos modelos tenían varias deficiencias internas como las dificultades para guardar información secuencial a largo plazo, lo cual producía perdidas de las dependencias temporales o también el hecho de que las RNNs eran difíciles de paralelizar lo cual hacia que el proceso de entrenamiento fuese menos eficiente.

Los investigadores introdujeron la arquitectura *Transformer* con el fin de reemplazar las RNNs en la traducción automática [Vaswani *et al.*, 2017]. Esta nueva estructura basada simplemente en mecanismos de atención reemplazaba las "Convolutional Neural Networks" (CNNs) y las RNNs. Como se ha comentado, los *Transformers* aparecieron con el fin de aplicarlos a la traducción automática, sin embargo, frente a la dificultad de tratar con las dependencias espacio-temporales para resolver la congestión de tráfico, los investigadores se dieron cuenta que estos *Transformers* podían ser útiles también para la predicción del flujo de tráfico, por lo que consecuentemente también para la predicción de demanda de movilidad del transporte público. Esto ocurría porque la en la traducción automática [Wu *et al.*, 2016] el objetivo es traducir una frase origen escrita en una lengua a una frase objetivo en otra lengua utilizando un sistema de aprendizaje secuencia a secuencia, y la predicción de tráfico se podía formular de una forma similar. Esta forma se basa en utilizar datos sobre las condiciones históricas del tráfico de tal manera que se conviertan en indicativos de las condiciones futuras en las que la secuencia origen consiste en una serie de datos de tráfico del pasado y la secuencia destino se compone de una serie de condiciones de tráfico en futuros pasos temporales. Dicho de otra manera, cada paso temporal en las secuencias de tráfico origen y objetivo se puede comparar con el índice de posición de cada palabra en las frases de entrada y salida en la tarea de traducción automática.

2.2. Selección de artículos

En cuanto a las plataformas escogidas para la búsqueda de información para llevar a cabo este capítulo del estado del arte, han sido de gran ayuda algunas como arXiv, Google Académico, Researchgate, Science Direct o Institute of Electrical and Electronics Engineers (IEEE)

Por otro lado, a la hora de llevar a cabo la búsqueda de experimentos realizados en diferentes investigaciones relacionadas con el tema a tratar en este trabajo de fin de máster, y por lo tanto también la búsqueda del código de los modelos desarrollados, ha sido de gran ayuda la herramienta GitHub.

2.3. Predicción de la demanda de movilidad del transporte público

La predicción de la demanda de movilidad del transporte público está influenciada por multitud de factores [T. Tsekeris and C. Tsekeris, 2011], es por ello que a lo largo de los últimos años esta predicción ha sido tratada de diferentes maneras e implementando diferentes modelos:

- En [[Zhang *et al.*, 2011]; [Li *et al.*, 2012]; [Moreira-Matias *et al.*, 2012]] se trata como una tarea de predicción de series temporales. En estas tres investigaciones se implementa de formas diferentes el modelo ARIMA, el cual es un modelo es-

tadístico que utiliza variaciones y regresiones de datos estadísticos con el fin de encontrar patrones para una predicción hacia el futuro. En [Zhang *et al.*, 2011] se utiliza una extensión de la versión tradicional llamada ARIMA estacional (SARIMA) con el fín de predecir el flujo de tráfico a corto plazo. Más adelante, en [Li *et al.*, 2012] se llevo a cabo la implementación del modelo ARIMA para la predicción de la recogida de taxis en puntos calientes individuales a partir de trayectorias GPS.

- Otros estudios como [[Ma *et al.*, 2013]; [Miao *et al.*, 2016]; [Ma *et al.*, 2016]], realizados unos años después, basaban la futura demanda de movilidad en una distribución de probabilidad. Por ejemplo, en [Ma *et al.*, 2013] se propuso un sistema de programación de taxis que responda solicitudes en tiempo real a las solicitudes enviadas por peatones simulados con una distribución de Poisson.

- Las investigaciones mencionadas hasta el momento están basadas en métodos estadísticos, sin embargo, a lo largo de los años y a través de los resultados obtenidos en las investigaciones realizadas, los profesionales del sector empezaron a utilizar algoritmos basados en el aprendizaje automático [[Hoang *et al.*, 2016]; [Lippi *et al.*, 2013]; [Habtemichael and Cetin, 2016], [Roos *et al.*, 2017]] para llevar a cabo la tarea de predicción de la demanda de movilidad del transporte público. En [Roos *et al.*, 2017], se utilizan las redes bayesianas para predecir el flujo de pasajeros en el metro. Por otro lado, en estudios como [Hoang *et al.*, 2016] en el que se buscaba predecir los flujos de personas en la ciudad a partir de big data, se descubrió que los campos aleatorios de Markov pueden capturar algunas dependencias que las BN no pueden. También se acabó entrenando el algoritmo "k-nearest neighbors" (k-NN) para el problema de la previsión del flujo de tráfico [Habtemichael and Cetin, 2016].

- Más adelante, se utilizaron los métodos basados en el aprendizaje profundo o *Deep learning* para la predicción de la demanda de taxis [[Yao *et al.*, 2018]; [Xu *et al.*, 2017]; [Rodrigues *et al.*, 2018], [Liu *et al.*, 2020]]. En [Xu *et al.*, 2017] se implementaron redes LSTM junto con redes de densidad mixta con el objetivo de predecir dicha demanda de taxis en diferentes zonas de la ciudad de Nueva York, teniendo en cuenta los datos de demanda del pasado y otro tipo de información como el tiempo meteorológico, la hora del día, festvios, etc. En cuanto a la investigación de [Yao *et al.*, 2018], consiguieron capturar las dependencias espaciales utilizando capas convolucionales en varios pasos temporales y estas mismas capas eran enviadas a la red LSTM. Para terminar con los métodos basados en el *Deep learning* para la predicción de la demanda de taxis, es importante mencionar que en [Liu *et al.*, 2020] se implementó un mecanismo de atención que integraba tres predicciones diferentes: un módulo instantáneo espacio-temporal, un módulo a corto plazo y un módulo periódico a largo plazo.

- En relación con el punto anterior, se llevaron a cabo diferentes investigaciones de nuevo basadas en métodos *Deep Learning* para la predicción de demanda de bicicletas [[Lin *et al.*, 2018]; [Jian, 2022]; [Chai *et al.*, 2018], [Li *et al.*, 2021]] desarrollaron un grafo del sistema de bicicletas compartidas y lo implementaron para el sistema de intercambio de bicicletas públicas llamado *Divvy* en Chicago y en la ciudad de Nueva York. Para conseguir resultados competentes utilizaron convoluciones multi-gráficas con una ventana de entrada que leía las últimas 6 horas para predecir el flujo de bicicletas en la hora siguiente. También se

predijo la demanda de cada hora de bicicletas en las diferentes estaciones de la ciudad de Nueva York mediante los grafos de redes neuronales o "Graph Neural Networks" (GNNs) [Lin *et al.*, 2018].

- Dentro de todas las diferentes formar de tratar el problema de predicción de la demanda de movilida del transporte público, finalmente, se desarrolaron métodos basados en el aprendizaje por refuerzo o *Reinforcement learning* como [Jiang *et al.*, 2018], en cual utilizaba diferentes técnicas para optimizar el número de pasajeros que utilizan el metro de Shanghai en horas punta. Un par de años después, en [Khaidem *et al.*, 2020] se utilizaron estos métodos basados en el *Reinforcement Learning* para predecir la movilidad humana en tres ciudades reales y una sintética.

Gracias a los resultados obtenidos de todas la investigaciones mencionadas se pueden llevar a cabo nuevas investigaciones basadas explícitamente en la predicción de la demanda de movilidad del transporte público [Prado-Rujas *et al.*, 2023].

2.4. Transformers

En la ultima década, los *Transformers* han revolucionado el "Natural Language Processing" (NLP) teniendo una gran capacidad para comprender y generar texto. Los *Transformers* han dejado atrás las arquitecturas recurrentes convencionales y utilizan una forma totalmente nueva para tratar con las secuencias de texto.

La revolución empezó en el año 2017, cuando en [Vaswani *et al.*, 2017] se presentó el modelo llamado *Transformer* basado en mecanismos de atención. El objetivo de este modelo era capturar relaciones a largo plazo entre las palabras que formaban una la secuencia de texto de entrada.

Este primer *Transformer* [Vaswani *et al.*, 2017], también conocido como *vanilla Transformer*, es un modelo secuencia-a-secuencia formado por una estructura codificador-decodificador la cual va a ser explicada a lo largo de este apartado. Cada uno de los bloques de codificador tiene dos subcapas que están formadas por un módulo *multi-head self-attention* y una "position-wise fully connected feed-forward network" (FFN). Entonces, se lleva a cabo una conexión residual alrededor de cada una de las dos subcapas, seguida de una normalización de capas de la cual se obtendrá un *output* Por otro lado, cada uno de lo bloques de decodificador están compuestos por las mismas dos subcapas que los bloques codificador, además de una tercera subcapa la cual realiza un *multi-head attention* sobre el *output* proporcionado por el decodificador. Al igual que en la parte del codificador, se lleva a cabo una conexión residual alrededor de cada una de la subcapas, seguido también de una capa de normalización. Entonces, se modifica la subcapa *self-attention* para evitar que las posiciones se fijen en las posiciones posteriores. Este enmascaramiento, combinado con el hecho de que los *embeddings* de salida están desplazados una posición, garantiza que las predicciones para la posición i sólo pueden depender de los *output* en las posiciones inferiores a i.

En cuanto a las funciones de atención, son la asignación de una consulta y un conjunto de pared clave-valor a un *output*, donde la consulta, las claves, los valores y el propio *output* son vectores. Dicho *output* se calcula a través de una suma ponderada de valores, en la que el paso asignado a cada valor está calculado mediante

una función de compatibilidad de la consulta con su clave correspondiente. Dentro de los tipos de mecanismos de atención aplicados al modelo, se encuentra por un lado el "Scaled Dot-Product Attention" y por otro lado está el conocido mecanismo de atención denominado como *multi-head attention*.

A continuación, en la figura 2.1 podemos ver la estructura del *Transformer* desarrollada por [Vaswani *et al.*, 2017] de forma detallada.

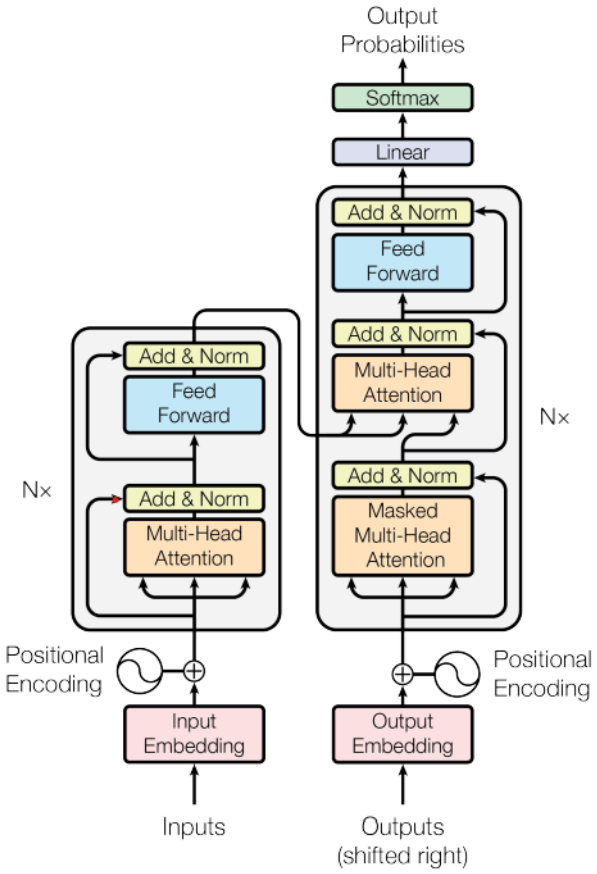


Figura 2.1: Estructura codificador-decodificador del *Transformer* introducido en [Vaswani *et al.*, 2017] página 3.

2.4.1. Embeddings

Los *embeddings* se han convertido en parte integrante de los modelos de inteligencia artificial, especialmente en el ámbito del "Natural Language Processing" (NLP). Su introducción revolucionó la representación de datos textuales y mejoró notablemente el rendimiento de diversas tareas. En los primeros tiempos del NLP, las palabras se representaban normalmente como vectores unidimensionales, en los que a cada palabra se le asignaba un índice único en un espacio de alta dimensión. Sin embargo, la escasez de información en estas representaciones dificultaba la captura de las relaciones semánticas entre palabras.

Para solucionar este problema, surgió el concepto *embeddings* de palabras en el innovador modelo *Word2Vec*, propuesto en [Mikolov *et al.*, 2013]. Este concepto se basaba en la idea de aprender representaciones distribuidas de palabras entrenando redes

neuronales para predecir palabras vecinas dada una palabra objetivo. Las representaciones de la capa oculta de estas palabras objetivo se utilizaron entonces como *embeddings* de palabras. Este método proporcionaba representaciones continuas y densas que captaban las relaciones semánticas y las similitudes entre las palabras. *Word2Vec* demostró mejoras significativas en varias tareas de NLP, como la analogía de palabras y la similitud semántica. Entonces, tras el éxito de estos *embeddings* de palabras, los investigadores extendieron sus aplicaciones más allá del NLP a otros dominios de la inteligencia artificial y empezaron a aplicarse para representar entidades, características o incluso puntos de datos enteros en un espacio vectorial continuo de dimensiones inferiores. Estos *embeddings* pretendían captar información significativa y permitir a los modelos aprender patrones y relaciones complejas a partir de los datos.

El momento clave del avance de los *embeddings* fue cuando se introdujo el primer *Transformer* en [Vaswani *et al.*, 2017]. La arquitectura *Transformer* revolucionó por completo el panorama de las tareas de NLP, incluida la traducción automática, el análisis de sentimientos y la comprensión del lenguaje. En el corazón del *Transformer* se encuentra la capa de *embeddings*, que desempeña un papel vital en la asignación de elementos de entrada discretos, como palabras o tokens, a representaciones vectoriales continuas.

2.4.2. Codificador

La introducción del codificador en la inteligencia artificial y a los *Transformers* ha revolucionado diversos campos, como el del "Natural Language Processing" (NLP) y el de la visión por ordenador. Un codificador es un componente crucial de muchos modelos de aprendizaje profundo, ya que sirve de base para el aprendizaje de representaciones a partir de datos de entrada sin procesar.

Una de las investigaciones influyentes que introdujo el concepto de los codificadores de redes neuronales es [Kingma and Welling, 2013]. Este artículo propone el "variational auto-encoder" (VAE), el cual es un modelo generativo que utiliza un codificador para asignar datos a un espacio latente de baja dimensión, lo que facilita el aprendizaje y la generación de representaciones eficientes.

En los *Transformers*, se introdujeron los mecanismos *self-attention* y *multi-head attention* los cuales supusieron un avance significativo. Esto fue presentado en el estudio [Vaswani *et al.*, 2017] junto a la arquitectura del *Transformer*, que revolucionó las tareas de modelado de secuencias. Dichos mecanismos de atención eran el componente principal del codificador en los *Transformers* y su objetivo era para capturar dependencias entre diferentes posiciones en la secuencia de entrada, eliminando la naturaleza secuencial de los modelos recurrentes como las RNN [Schmidt, 2019]. Este artículo demostró la eficacia de los *Transformers* en tareas de traducción automática y sentó las bases para avances posteriores.

2.4.3. Decodificador

La introducción del decodificador en la inteligencia artificial y los *Transformers* ha desempeñado un papel crucial en diversos dominios, como el "Natural Language Processing" y la generación de secuencias. Un decodificador es un componente clave

en muchos modelos de aprendizaje profundo, responsable de generar secuencias de salida basadas en representaciones codificadas.

En el contexto de los *Transformers*, el decodificador es una parte esencial de la arquitectura general que consiste en una pila de capas idénticas, similar a la del codificador. Cada capa del decodificador incorpora mecanismos de *self-attention* para captar las dependencias dentro de la secuencia de salida que se está generando. El mecanismo de *self-attention* permite al decodificador centrarse en las partes relevantes de la secuencia de salida y modelar eficazmente las relaciones entre las distintas posiciones. Dicho concepto del decodificador en los *Transformers* se introdujo en el mismo artículo que el codificador [Vaswani *et al.*, 2017].

Investigaciones como [Dai *et al.*, 2018] han supuesto avances sumamente importantes para el decodificador en los *Transformers* ya que, introdujo técnicas para manejar secuencias más largas en el decodificador. Este componente, ha sido fundamental para lograr un rendimiento de vanguardia en varias tareas del ámbito "Natural Language Processing", incluida la traducción automática, la generación de lenguaje y el resumen de texto.

2.4.4. Mecanismos de atención

Los mecanismos de atención permiten que los modelos se centren en partes específicas de los datos de entrada, lo que permite un procesamiento más complejo y consciente del contexto. En este apartado, se van a explorar los orígenes de los mecanismos de atención y su integración con los *Transformers* que se han convertido en la columna vertebral de muchas aplicaciones de inteligencia artificial de próxima generación.

En [Bahdanau *et al.*, 2014] se introdujeron por primera vez los mecanismos de atención a las tareas de traducción automática. En los mecanismos propuestos en dicho estudio, la atención permite que el modelo se centre en partes relevantes de la oración de origen mientras genera la palabra correspondiente en el idioma de destino. Este mecanismo de atención utiliza una suma ponderada de estados ocultos del flujo de origen, calculando dinámicamente los pesos en función de su importancia para el paso de decodificación actual. Al centrarse en diferentes partes del flujo de entrada durante la traducción, el modelo logra una mayor precisión y mejora la calidad de la traducción.

En el trabajo de [Vaswani *et al.*, 2017], mencionado anteriormente, se llevaron los mecanismos de atención a la vanguardia del "Natural Language Processing". El *Transformer* empleó un mecanismo *self-attention* conocido como "scaled dot-product attention", que permitió al modelo capturar dependencias entre diferentes palabras en la secuencia de entrada. A diferencia de las RNN, los *Transformers* procesaban toda la secuencia en paralelo, lo que los hacía muy eficientes y escalables. El mecanismo de *self-attention* calculaba los pesos de atención de cada palabra de la secuencia comparándola con todas las demás, lo que permitía captar eficazmente las dependencias locales y globales.

2.4.4.1. Self-attention

Como se ha mencionado recientemente, este mecanismo de atención se presentó por primera vez en el estudio de [Vaswani *et al.*, 2017] con el objetivo de que se pudie-

sen abordar las limitaciones de las "Recurrent Neuronal Networs" (RNN) a la hora de capturar dependencias de largo alcance en secuencias. Mientras que estas RNN procesaban secuencias de forma secuencial (de una a una), el mecanismo *self-attention* permitía a los *Transformers* capturar dependencias entre distintas posiciones en paralelo, lo que los hace más eficientes y eficaces para tareas que implican grandes cantidades de datos.

Al incorporar este mecanismo, el *Transformer* puede capturar las dependencias entre todas las posiciones de la secuencia de entrada, independientemente de su distancia. Esta capacidad de modelar eficazmente las dependencias de largo alcance ha sido decisiva para lograr un rendimiento de vanguardia en diversas tareas de "Natural Language Processing". De hecho, desde su introducción, este mecanismo se ha explorado y perfeccionado en investigaciones posteriores como [[Child *et al.*, 2019]; [Choromanski *et al.*, 2020]; [Wang *et al.*, 2020]; [Kitaev *et al.*, 2020]]. Se han propuesto distintas variantes y extensiones de la *self-attention*, como el mecanismo *multi-head attention* que permite al modelo atender simultáneamente a distintos subespacios de la secuencia de entrada, y los patrones de atención dispersa, que mejoran la eficiencia computacional atendiendo sólo a un subconjunto de posiciones.

2.4.4.2. Multi-Head attention

El *multi-head attention* es una extensión del mecanismo de *self-attention* y igual que este, fue introducido en [Vaswani *et al.*, 2017]. Permite al modelo atender simultáneamente a distintos subespacios de la secuencia de entrada, lo que posibilita un aprendizaje de la representación más completo y expresivo.

La introducción del *multi-head attention* ha sido decisiva para mejorar la capacidad expresiva y el rendimiento de los *Transformers*. Permite un modelado más matizado de las dependencias y proporciona un mecanismo para capturar simultáneamente distintos tipos de información. El *multi-head attention* ha demostrado un rendimiento superior en diversas tareas de "Natural Language Processing", como la traducción automática, la comprensión del lenguaje y la generación de textos.

En conclusión, el *multi-head attention* proporciona a los *Transformers* la capacidad de captar diversas dependencias, modelar distintos tipos de información simultáneamente y lograr un rendimiento superior en tareas de "Natural Language Processing". Su capacidad para atender a múltiples subespacios mejora la potencia expresiva y la comprensión contextual de los modelos, lo que conduce a una generación y comprensión de secuencias más precisas, sólidas y ricas en contexto.

A continuación, podemos observar en la figura 2.2 la arquitectura que forma la extensión del mecanismo *self-attention* (*multi-head attention*). Después, se encuentra un ejemplo de las matrices que proporciona dicho mecanismo *self-attention* como output cada vez que las secuencias de entrada pasan por el. Esto se puede encontrar en la figura 2.3.

2.4.5. Modelos de Transformers

En este apartado, se van a presentar algunos de los *Transformers* más relevantes de la última década que se desarrollaron tras la revolución provocada por la pionera investigación de [Vaswani *et al.*, 2017].

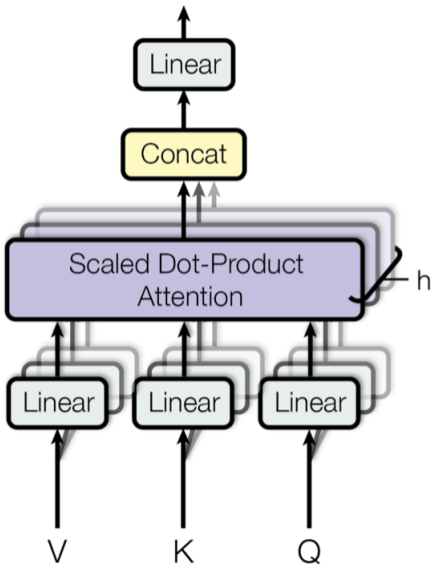


Figura 2.2: Estructura del mecanismo *multi-head attention* de [Vaswani *et al.*, 2017] página 4.

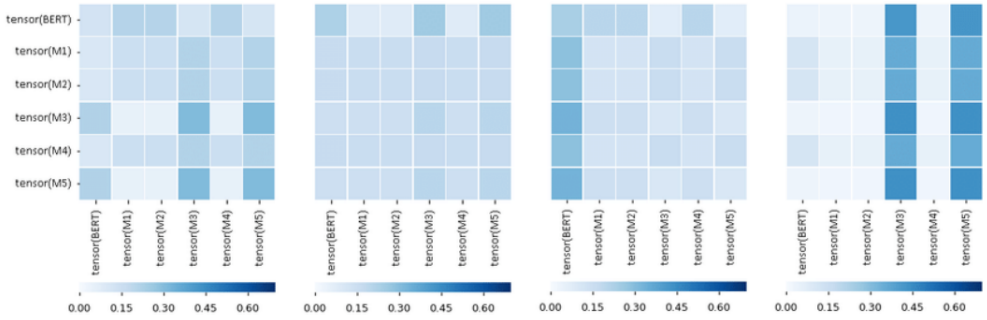


Figura 2.3: Ejemplo de la matriz resultante de un mecanismo *multi-head attention* de [Bugueño *et al.*, 2019]

2.4.5.1. Bidirectional Encoder Representations from Transformers (BERT)

"Bidirectional Encoder Representations from Transformers" (BERT) es un modelo basadao en el "Natural Language Processing" (NLP) de última generación desarrollado por Google en 2018 [Jacob *et al.*, 2019]. Revolucionó el campo de la NLP al introducir un enfoque novedoso para el pre-entrenamiento de las representaciones lingüísticas. BERT se basa en la arquitectura *Transformer*, que es una arquitectura de modelo de aprendizaje profundo introducida por [Vaswani *et al.*, 2017]. El *Transformer* consta de un codificador y un decodificador, donde ambos se componen de capas apiladas de *self-attention* y *feed-forward*. Sin embargo, este nuevo modelo se centra únicamente en la parte del codificador y utiliza un enfoque bidireccional.

Los modelos lingüísticos tradicionales, como los basados en "Recurrent Neuronal Networs" (RNN), procesan el texto de forma secuencial, leyendo las palabras de izquierda a derecha o viceversa. BERT, en cambio, emplea un método de entrenamiento bidireccional, es decir, tiene en cuenta tanto el contexto izquierdo como el derecho de una palabra a la hora de hacer predicciones. Esta naturaleza bidireccional permite

a BERT captar información contextual más completa y comprender los matices del lenguaje.

La fase de pre-entrenamiento de BERT consiste en entrenar el modelo con grandes cantidades de datos de texto sin etiquetar. Durante esta fase, BERT aprende a predecir las palabras que faltan en las frases conociendo el contexto tanto del lado izquierdo como del derecho de la frase. Se entrena en tareas como el modelado de lenguaje enmascarado, en el que se enmascaran palabras aleatorias del texto de entrada y el modelo se entrena para predecir esas palabras enmascaradas basándose en el contexto circundante. BERT también se entrena en la tarea de predicción de la siguiente frase, en la que aprende a predecir si dos frases aparecen consecutivamente o no en los datos de entrenamiento. Tras la fase de pre-entrenamiento, BERT se perfecciona en tareas posteriores específicas como la clasificación de textos, el reconocimiento de entidades con nombre, la respuesta a preguntas, el análisis de sentimientos, etc. Durante la puesta a punto, BERT se entrena con datos etiquetados específicos de la tarea, lo que le ayuda a adaptarse a los requisitos específicos de la tarea de destino.

2.4.5.2. Generative Pre-trained Transformer 2 (GPT-2)

Generative Pre-trained Transformer 2 (GPT-2) es un modelo de lenguaje muy influyente desarrollado por la empresa OpenAI e introducido en [Radford *et al.*, 2019], basado en el éxito de su predecesor, GPT [Radford *et al.*, 2018], y ha ampliado los límites del modelado generativo de texto. GPT-2 se basa en la arquitectura *Transformer*, que consiste en una pila de capas de *self-attention* y *feed-forward*. Se trata de un modelo generativo, lo que significa que es capaz de generar un texto coherente y contextualmente relevante a partir de una indicación dada. A diferencia de BERT, que se centra en la formación bidireccional, GPT-2 es unidireccional y genera texto de forma secuencial, palabra por palabra.

La innovación clave de GPT-2 reside en su enfoque de pre-entrenamiento. Durante esta fase, el modelo se entrena en un gran corpus de texto disponible públicamente en Internet, aprendiendo a predecir la siguiente palabra de una frase dado el contexto precedente. Este proceso ayuda al modelo a desarrollar una rica comprensión del lenguaje y le permite generar textos coherentes y adecuados al contexto.

GPT-2 introdujo varios avances en el modelado del lenguaje. Tiene una arquitectura a gran escala con 1.500 millones de parámetros (aunque también existen versiones más pequeñas con menos parámetros), lo que lo convierte en uno de los mayores modelos lingüísticos en el momento de su lanzamiento. El gran tamaño del modelo le permite captar y aprender patrones complejos del lenguaje, lo que se traduce en una impresionante capacidad de generación de textos. Otra de las características más notables del GPT-2 es su capacidad para generar textos diversos y creativos. Ha demostrado la capacidad de generar párrafos coherentes, relatos completos, poesía e incluso imitar el estilo de escritura de autores o géneros específicos. Sin embargo, es importante señalar que GPT-2 genera texto basándose en patrones que aprende de los datos de entrenamiento, y no posee verdadera comprensión o conciencia. Es decir, depende totalmente de los datos con los que se ha entrenado por lo que puede generar textos sobre un tema muy especifico que esten parcialmente incorrectos.

Debido a la preocupación por el posible uso indebido de los potentes modelos lingüísticos, OpenAI lanzó inicialmente GPT-2 con algunas restricciones. Sin embargo, en

versiones posteriores puso a disposición del público el modelo completo y sus parámetros. Desde entonces, GPT-2 ha sido ampliamente adoptado por investigadores y entusiastas, contribuyendo a avances en diversas tareas de NLP, escritura creativa e interacción persona-ordenador.

2.4.5.3. Text-to-Text Transfer Transformer (T5)

"Text-to-Text Transfer Transformer" (T5) es un modelo de lenguaje versátil presentado por Google Research en [Raffel *et al.*, 2020]. T5 se basa en la arquitectura *Transformer* y está diseñado para manejar una amplia gama de tareas de "Natural Language Processing" (NLP) utilizando un marco unificado. T5, a diferencia de otros modelos,destaca por su capacidad para realizar múltiples tareas NLP a través de un enfoque de texto a texto. En este paradigma, todas las tareas se plantean como problemas de generación de texto, en los que la entrada y la salida se representan como cadenas de texto. Esto permite a T5 realizar diversas tareas, como clasificación de textos, traducción, resumen, respuesta a preguntas, etc., proporcionando los formatos de entrada y salida adecuados.

Durante la fase de pre-entrenamiento, T5 se entrena en un gran corpus de datos de texto utilizando un objetivo de "masked language modeling". En este caso, al igual que en BERT, se enmascaran palabras o espacios aleatorios del texto de entrada y el T5 se entrena con el objetivo de predecir los fragmentos que faltan. Sin embargo, a diferencia de BERT, lo que se busca en la propia fase entrenamiento de T5 incluye el hecho de generar el texto de salida completo, en lugar de predecir tokens enmascarados individuales. Esto convierte a T5 en un modelo generativo que puede generar frases completas o incluso párrafos de texto.

T5 se entrena simultáneamente en diversas tareas utilizando una arquitectura de modelo compartida. Al entrenarse conjuntamente en varias tareas, T5 aprende a captar la comprensión general del lenguaje y puede transferir conocimientos entre tareas. Este enfoque de aprendizaje multitarea permite a T5 obtener un gran rendimiento en diversas pruebas de NLP. Además, T5 funciona sobre un formato texto a texto, lo cual es algo muy característico del modelo. Por otro lado, para aplicar T5 a una tarea concreta, el texto de entrada se formula como una pregunta o una indicación, y la salida deseada se genera como la respuesta o la finalización de la indicación. Al enmarcar todas las tareas en una forma coherente de texto a texto, T5 proporciona un marco unificado y flexible para las tareas de NLP.

2.4.5.4. GPT-3 (2020): Generative Pre-trained Transformer 3 (GPT-3)

"Generative Pre-trained Transformer" 3 (GPT-3) es un modelo de lenguaje muy avanzado desarrollado por OpenAI, lanzado en [Brown *et al.*, 2020]. Al igual que su predecesor se baso en el existe de [Radford *et al.*, 2018], en el caso de GPT-3 ha ocurrido lo mismo, ya que GPT-2 [Radford *et al.*, 2019] obtuvo unos resultados muy buenos por lo que se busco mediante este nuevo actualizarlo y mejorarlo aún más con lo que respecta a su tamaño, capacidades y rendimiento.

GPT-3 se basa en la arquitectura *Transformer*, que consta de capas apiladas de *self-attention* y *feed-forward*. Al igual que sus predecesores, se trata de un modelo generativo ya que es capaz de generar texto similar al humano a partir de un estímulo o contexto determinado. Por otro lado, GPT-3 tiene una escala sin precedentes, con la

asombrosa cifra de 175.000 millones de parámetros, lo que lo convierte en uno de los mayores modelos lingüísticos jamás creados en el momento de su lanzamiento.

La principal característica de GPT-3 es su capacidad para realizar una amplia gama de tareas de "Natural Language Processing" (NLP) con un entrenamiento mínimo específico. Es capaz de comprender y generar textos coherentes, responder preguntas, traducir, resumir y analizar sentimientos, entre otras funciones. GPT-3 ha logrado esta notable versatilidad aprovechando su amplio entrenamiento previo en un corpus diverso de textos de Internet. Además, a diferencia de los modelos anteriores que requerían un ajuste preciso para tareas específicas, GPT-3 puede realizar varias tareas sin necesidad de realizar ningún ajuste o con unos pocos ejemplos. Al condicionar el modelo con una indicación o instrucción, GPT-3 puede generar texto que muestre el comportamiento o resultado deseado, dado el contexto. Esta capacidad de *zero-shot* o ningún ajuste pone de manifiesto el poder de generalización y transferencia de aprendizaje de GPT-3.

GPT-3 ha demostrado un rendimiento excepcional en una gran variedad de pruebas y tareas. Es capaz de generar textos creativos, responder a preguntas con precisión, proporcionar explicaciones detalladas e incluso componer poesía o relatos. Sin embargo, es importante tener en cuenta que los resultados de GPT-3 se generan a partir de patrones aprendidos de sus datos de entrenamiento y no siempre son perfectamente precisos. Es decir, como se ha comentado en el caso del GPT-2 existe la posibilidad de que genere textos parcialmente incorrectos si en los datos utilizados para entrenar al modelo existe algún tipo de fallo o hay algún dato incompleto.

2.4.6. Transformers de largo alcance

Los *Transformers*, como se ha explicado anteriormente, están basados en mecanismos de atención los cuales implican la correspondencia de cada consulta con toda la secuencia de clave, y esto provoca que el tiempo de ejecución y uso de memoria crezca de forma cuadrática con la longitud de los datos de entrada. Por lo tanto, los investigadores desarrollaron y evaluaron variantes de *Transformer* para secuencias más largas [Tay *et al.*, 2020]. En la mayoría de los métodos desarrollados en dicha investigación por la comunidad de investigadores introducían heurísticas para dispersar la matriz de atención. Por ejemplo, en [Li *et al.*, 2019] se centraba la atención principalmente a los tokens de entrada adyacente, en [[Child *et al.*, 2019];[Ye *et al.*, 2019]] tokens cada vez más lejanos o algunos estudios como [[Zaheer *et al.*, 2020]; [Zhang *et al.*, 2021]] eran una combinación de algunos de estos. Sin embargo, pese a que dichos métodos eran efectivos, sus sesgos inductivos sobre la estructura de la matriz de atención entrenada no siempre son compatibles con tareas ajenas al campo del procesamiento del lenguaje natural (NLP).

2.4.7. Series temporales en Transformers

Originalmente, la gran mayoría de investigaciones basadas en el aprendizaje profundo para la predicción de series temporales trabajaban sobre un marco *Seq2Seq* (secuencia a secuencia, ejemplo en la figura 2.4) en el que se mapea una *context window* de los c pasos temporales más recientes a una *target window* de predicciones para un horizonte temporal de h pasos hacia el futuro. Siendo x_t un vector de valores *timestamp* (día, mes, año, etc.) en un tiempo t y y_t sea un vector de valores espaciales en el mismo tiempo t. Entonces, dada una *context sequence* de *inputs* de

tiempo $(x_{T-c}, ..., x_T)$ y *inputs* espaciales $(y_{T-c}, ..., y_T)$ hasta el tiempo *T*, se genera una secuencia de salida de valores de espaciales $(\hat{y}_{T+1}, ..., \hat{y}_{T+h})$ correspondientes a las predicciones en los pasos de tiempo futuros $(x_{T+1}, ..., x_{T+h})$.

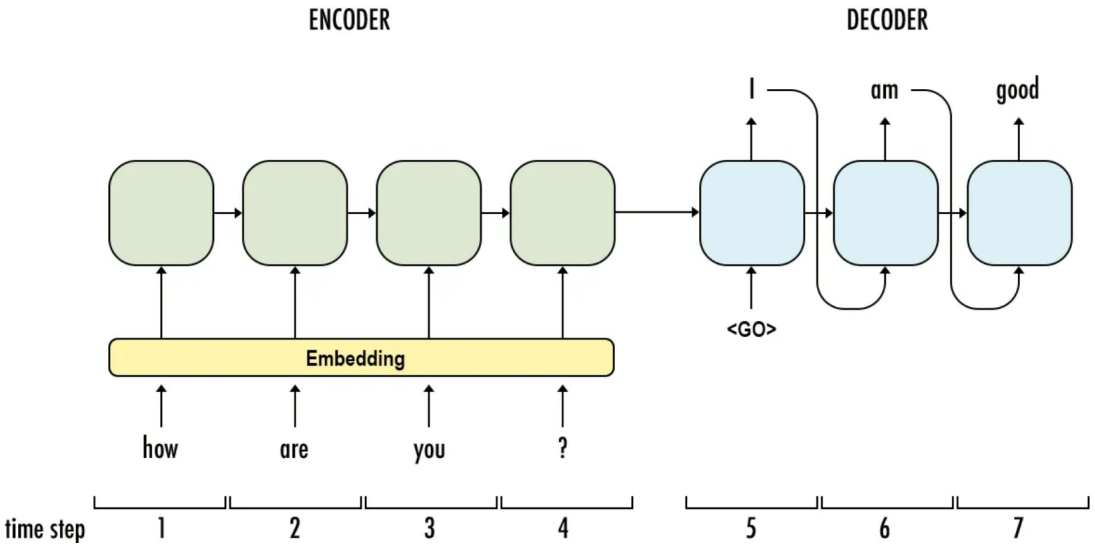


Figura 2.4: Ejemplo del modelo Seq2Seq y su estructura de [Manish, 2017]

La clase más común entre los modelos de predicción de series temporales profundos es una combinación entre las RNN y las convoluciones unidimensionales (Conv1Ds) [[Borovykh *et al.*, 2017]; [Smyl, 2020]; [Flunkert *et al.*, 2017];[Lai *et al.*, 2017];]. Aunque, la clase que más se asemeja al problema propuesto en este trabajo fin de máster está basada en mecanismos de atención que tienen como objetivo superar el entrenamiento auto-regresivo de las RNN y la interpretación de dependencias a largo plazo [[Iwata and Kumagai, 2020]; [LI *et al.*, 2019]; [Zerveas *et al, 2020*]; [Oreshkin *et al.*, 2020]]. Entre estas investigaciones existen dos que destacan más en cuanto a semejanza con el objetivo de este proyecto que son el *Spacetimeformer* de [Grigsby *et al.*, 2021] y, como es entendible, el estudio en el que está basado el *Spacetimeformer* denominado *Informer* de [Zhou *et al.*, 2020]. Estos dos modelos tienen una arquitectura general de codificador-decodificador *Transformer* capaz de ocuparse del problema de la predicción de series temporales.

En el modelo denominado *Informer* toma los *timestamps* de secuencia $(x_{T-c}, ..., x_{T+h})$ y los incrusta en una dimensión superior *d*. Los valores espaciales (se utilizan ceros para reemplazar la secuencia objetivo desconocida $(y_{T-c}, ..., y_T, 0_{T+1}..., 0_{T+h})$) se mapean a dicha misma dimensión d. Las variables de tiempo (*x*) y espaciales (*y*) se suman para crear una secuencia de entrada de tokens $c + h$, escritos en forma de matriz como $Z \in R^{(c+h)Xd}$. Dentro de la propia estructura del *Informer* (figura 2.5), el codificador procesa la subsecuencia $Z[0...c]$ proveniente de la matriz $Z \in R^{(c+h)Xd}$ mientras el decodificador observa la *target secuence* $Z[c + 1...c + h]$. Las salidas del decodificador se tratan como predicciones y el error se minimiza mediante la regresión a los valores de secuencia verdadera. Se debe tener en cuenta que el *Informer* genera toda la secuencia de predicción directamente en una sola pasada hacia adelante, a diferencia de los modelos generativos, que están formados tan solo de la parte del decodificador, que generan cada token de forma iterativa (como los modelos lingüísticos grandes). Esto tiene la ventaja de reducir el cálculo en el momento de re-

utilizar la representación del codificador en cada capa del decodificador y minimizar la acumulación de errores en las predicciones autorregresivas.

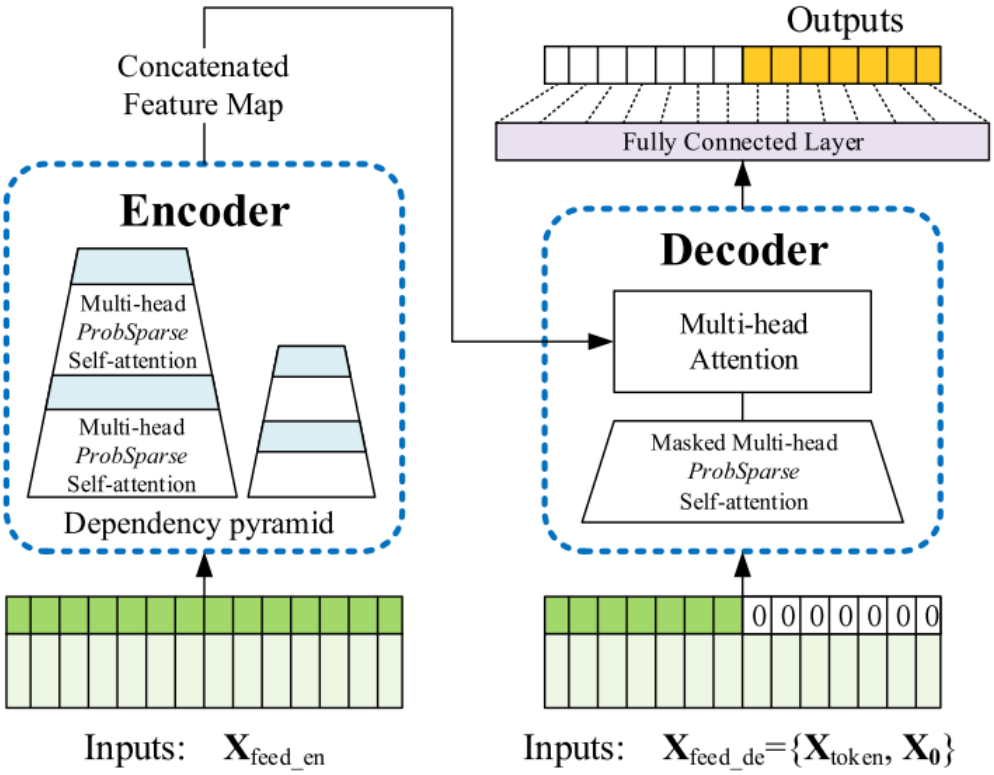


Figura 2.5: Arquitectura codificador-decodificador del modelo propuesto en *Informer* [Zhou *et al.*, 2020]

Después del*Informer*, del *Spacetimeformer* y todas las investigaciones parecidas citadas, surgió una nueva oleada de trabajos e investigaciones que tenían como objetivo mejorar los resultados de referencia que habían dejado los Transformers en el problema de predicción de series temporales. En estos trabajos se introducían métodos que ajustaban la arquitectura del modelo [[Madhusudhanan *et al.*, 2021]; [Liu *et al.*, 2022]] y que volvían a introducir el conocimiento clásico del dominio de series temporales, como la descomposición en serie, la autocorrelación o el cálculo en el espacio de frecuencias [[Zhou *et al, 2022*]; [Woo *et al.*, 2020]]. Estos sesgos de series temporales se utilizan a menudo como una forma específica de dominio de atención eficiente [[Wu *et al.*, 2021]; [Liu *et al.*, 2022]; [Du *et al.*, 2022]], ya que muchas tareas relacionada al problema de predicción de series temporales a largo plazo se extienden naturalmente más allá de los límites de secuencia de los *Transformers* predeterminados.

Es importante tener en cuenta que las series temporales de *Transformers* utilizan un token de entrada por cada paso temporal, de modo que el *embedding* de cada token en el tiempo t representa N variables diferentes en ese momento. Esto contrasta con dominios como el "Natural Language Processing" (NLP) en el que cada token representa una sola idea unificada como, por ejemplo, una sola palabra [Shih *et al.*, 2018]. Otro caso a mencionar es el del gráfico de paso de mensajes el cual es el resultado

de aplicar la técnica *attention* sobre una secuencia multivariante aprendiendo patrones a lo largo del tiempo mientras mantiene las variables espaciales agrupadas. Esto se puede ver reflejado en la figura 2.6 en la que las variables espaciales aun están agrupadas y tan solo se diferencian las variables de tiempo tras aplicar la técnica *attention*, marcando con un azul más oscuro las relaciones a las que se debe prestar más atención. En la parte izquierda de la línea discontinua negra se encuentran las variables temporales y espaciales de contexto mientras que a la derecha las variables objetivo. Esta configuración obliga a las variables dentro de cada token a recibir la misma cantidad de información de otros pasos de tiempo, a pesar del hecho de que las variables pueden tener patrones o relaciones distintas entre sí. Idealmente, tendríamos un forma de modelar este tipo de relaciones variables.

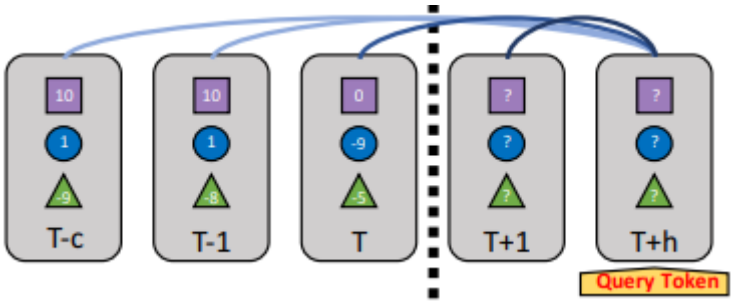


Figura 2.6: Representación de series temporales con las variables espaciales agrupadas ([Grigsby *et al.*, 2021], página 1).

2.4.8. Dependencias espaciales

Los modelos estadísticos y basados en redes neuronales se desarrollaron para la previsión del flujo de tráfico. Los modelos estadísticos como el "Autoregressive Integrated Moving Average" (ARIMA) [Min and Wynter, 2011] y las "Bayesian Networks" [Want *et al.*, 2014] modelan las dependencias espaciales desde una visión probabilística. Pese a que estos modelos quieren ayudar a analizar la incertidumbre dentro de la movilidad del tráfico en las ciudades grandes, su naturaleza lineal les impide modelar eficazmente la alta no linealidad dentro de la propia movilidad de tráfico. Entonces, las redes neuronales se introdujeron para capturar la no linealidad de dicha movilidad de tráfico, pero sus estructuras totalmente conectadas son intensivas en computación y consumen memoria. Además, la falta de suposiciones hace imposible capturar los complicados patrones espaciales en los flujos de tráfico.

Después, se introdujeron las "Convolutional Neural Networks" (CNN) en el estudio de [O'Shea and Nash, 2015] y se emplearon en tareas de predicción de la movilidad del tráfico teniendo en cuenta sus poderosas capacidades de extracción de características en muchas aplicaciones [[Gehring *et al.*, 2017]; [Krizhevsky *et al.*, 2012]; [Shelhamer *et al.*, 2014]]. Estas nuevas redes neuronales convolucionales se utilizaron para extraer las características espaciales en las que las redes de tráfico se convierten en cuadrículas regulares,también conocidas como *mesh-grids* [[Ma *et al.*, 2017]; [Zhang *et al.*, 2016]; [Zhang *et al.*, 2020]]. Sin embargo, esta conversión de la red conduce a la pérdida de información topológica inherente que caracteriza las redes de tráfico irregular por lo que se introdujeron las "Graph Neural Networks" (GNN) [[Scarselli *et al.*, 2009]; [Gilmer *et al.*, 2017]] para generalizar el aprendizaje profundo a dominios no euclidianos. Y como variante de dichas GNN, aparecieron las "Graph

Convolution Networks" (GCN) [[Atwood and Towsley, 2015]; [Defferrard *et al.*, 2016]; [Kipf and Welling, 2016]] que generalizan las convoluciones clásicas al dominio del grafo. Más adelante, se propuso que las GCN modelasen las dependencias espaciales de los flujos de tráfico para explorar la topología de tráfico inherente.

Por otro lado, las "Spatio-Temporal Graph Convolutional Networks" (STGCN) se anunciaron en [Yu *et al.*, 2017] y eran capaces de modelar las dependencias espaciales con convoluciones de grafos espectrales definidos en un grafo no dirigido, mientras que las "Diffusion Convolutional Recurrent Neural Network" (DCRNN) [Li *et al.*, 2017] empleaban convoluciones de grafos de difusión en un grafo dirigido para acomodar las direcciones de la movilidad de tráfico. Sin embargo, los modelos mencionados ignoraban los cambios dinámicos de las condiciones del tráfico como las horas punta y los accidentes de tráfico, ya que las dependencias espaciales se fijaban una vez eran entrenadas, es decir, no es posible cambiarlas una vez han sido entrenadas con dichos datos. Luego, en [Guo *et al.*, 2019] , se generaban las dependencias espaciales dinámicamente con la profundidad de los bloques espaciales y temporales, en lugar de los pasos de tiempo reales.

Por último, las dependencias espaciales dinámicas se consiguen modelar incorporando las "Graph Attention Networks" (GAT) [Velickovic *et al.*, 2017] y las características geográficas integradas resumidas por un *meta-learner* adicional de [Pan *et al.*, 2019]. Sin embargo, la topología de grafos predefinida que usa k vecinos más cercanos se limita a descubrir patrones ocultos de dependencias espaciales en varias escalas más allá de los nodos locales. Por esto, *Graph WaveNet* [Wu *et al.*, 2019] mejora la precisión de la previsión de tráfico con patrones espaciales ocultos a través de un *embedding* que se puede aprender para cada nodo en el gráfico, pero sus dependencias espaciales se siguen fijando una vez han sido entrenados.

2.4.9. Dependencias temporales

Como se indica en [[Ma *et al.*, 2015]; [Wu and Tan, 2016]], las RNN están limitadas para modelar dependencias temporales, debido a gradientes explosivos o de desaparición en el entrenamiento y la determinación inexacta de las longitudes de secuencia. Para poder tratar con estos inconvenientes, se desarrollaron las "Gated Recurrent Units" (GRU) [Chung *et al.*, 2014] y la "Long-Short Term Memory" (LSTM) [Hochreiter and Schmidhuber, 1997]. Estos nuevos modelos eran capaces de modelar dependencias de largo alcance para la previsión de tráfico [[Guo *et al.*, 2019]; [Pan *et al.*, 2019]; [Ma *et al.*, 2015]; [Wu and Tan, 2016]]. Sin embargo, estos modelos secuenciales aún sufren de un gran problema ya que, el proceso de entrenamiento consume demasiado tiempo y tienen una escalabilidad limitada para poder modelar secuencias largas.

Entonces, como alternativa aparecieron algunos modelos de aprendizaje de secuencias basados en la convolución [Gehring *et al.*, 2017]. Estos nuevos modelos requerían de múltiples capas ocultas para cubrir grandes contextos bajo un tamaño limitado de campos receptivos. *WaveNet* con convolución de dilatación se adopta en [Wu *et al.*, 2019] para ampliar los campos receptivos y, en consecuencia, reducir el número de capas ocultas. Sin embargo, la escalabilidad del modelo está restringida para secuencias de entrada largas, ya que el número de capas ocultas aumenta linealmente con las longitudes de las secuencias de entrada. Además, la eficiencia para capturar dependencias de largo alcance se vería afectada por capas más pro-

fundas, debido a las crecientes longitudes de caminos entre los componentes de la secuencia [[Vaswani *et al.*, 2017]; [Wu and Tan, 2016]]. Estos hechos implican que sería prohibitivo encontrar las longitudes óptimas de las secuencias de entrada, ya que el modelo necesita ser rediseñado para secuencias de entrada con diferentes *self-attention* altamente paralelizables. Las dependencias variables en el tiempo de largo alcance se pueden capturar de forma adaptativa a partir de secuencias de entrada con varias longitudes con una sola capa.

2.4.10. Transformers espacio-temporales

Una de las aplicaciones más importantes de los *Transformers* es el modelado espacio-temporal, en el cual destacan en el manejo de datos secuenciales y espaciales con dependencias temporales como se puede ver en la figura 2.7. En dicha figura, a la izquierda se encuentran las variables contextuales tanto espaciales como temporales, mientras que a la derecha las objetivo. En cuanto a la aplicación de la técnica *attention*, se puede observar como diferentes zonas de las líneas azules tienen un azul más o menos oscuro lo que significa que cuanto más oscuro más atención hay que prestar a esa parte.

Los *Transformers* espacio-temporales surgieron como una potente herramienta para tareas como la comprensión de vídeo, el reconocimiento de acciones y la predicción de movimiento. A continuación se van a mencionar y describir detalladamente algunos de los modelos de *Transformers* espacio-temporales más relevantes desarrollados hasta la fecha.

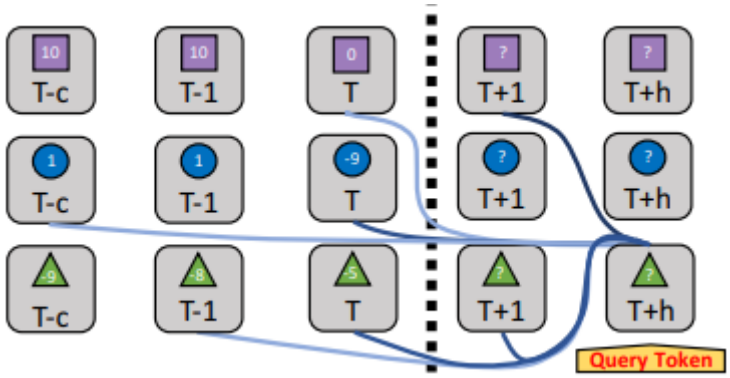


Figura 2.7: Variables espacio-temporales cuyas variables espacial en cada paso en el tiempo son un tokens separado ([Grigsby *et al.*, 2021], página 1).

2.4.10.1. Video Vision Transformers (ViViT)

"Video Vision Transformers" (ViViT) introducida en [Arnab *et al.*, 2021] es una arquitectura de modelos diseñada para tareas de comprensión de vídeo, centrada específicamente en el reconocimiento visual y el análisis en vídeos. ViViT amplía la arquitectura *Transformer*, que ha demostrado un gran éxito en tareas de "Natural Language Processing" e imágenes, al dominio de los vídeos. La idea central de ViViT es aprovechar el mecanismo *self-attention* de los *Transformers* para modelar dependencias de largo alcance y capturar relaciones espacio-temporales dentro de los fotogramas de vídeo. Los enfoques tradicionales de análisis de vídeo suelen basarse en "Recurrent

Neuronal Networs" (RNN) o convolucionales (de diferentes dimensiones) y sufren notables limitación esa la hora de captar las dependencias a largo plazo entre fotogramas. ViViT aborda esta limitación utilizando la *self-attention* para permitir el modelado del contexto global.

Por el lado de las dependencias espaciales, en ViViT, cada fotograma de vídeo se trata como una secuencia de parches, de forma similar a como se dividen las imágenes en parches en el "Vision Transformer" (ViT) [Dosovitskiy *et al.*, 2020], los cuales son posteriormente procesados por una pila de capas *self-attention* y *feed-forward*, similar a la arquitectura *Transformer* original [Vaswani *et al.*, 2017]. En los mecanismos *self-attention* se busca que cada parche atienda a otros parches, capturando las relaciones espaciales dentro de un mismo fotograma. Por el otro lado, en cuanto a las dependencias temporales, dependencias temporales entre fotogramas de vídeo, ViViT incorpora un mecanismo de incrustación temporal. Introduce una codificación temporal adicional para captar el orden y la secuencia de los fotogramas, lo que permite al modelo aprender la dinámica temporal del vídeo. Esta incrustación temporal se combina con la información espacial de cada parche para formar la entrada a las capas de *self-attention*.

Para entrenar el ViViT de forma supervisada se utilizan conjuntos de datos de vídeo con las anotaciones apropiadas para la tarea específica en cuestión, como el reconocimiento de acciones o la clasificación de vídeos. Durante el entrenamiento, el modelo aprende a predecir las etiquetas o anotaciones correctas basándose en los fotogramas de vídeo de entrada. Las capas de *self-attention* y la combinación de información espacial y temporal permiten a ViViT aprender representaciones ricas para tareas de análisis de vídeo.

2.4.10.2. Spatial-Temporal Transformer Networks for Traffic Flow Forecasting (STTN)

El modelo denominado "Spatial-Temporal Transformer Networks" (STTN) fue introducido en [Mingsing Xu *et al.*, 2020] y utiliza una arquitectura diseñada específicamente para la previsión del flujo de tráfico. Tarea cuyo objetivo es predecir las condiciones del tráfico en varios lugares e intervalos de tiempo. STTN utiliza la arquitectura *Transformer*, que ha demostrado un gran éxito en tareas de "Natural Language Processing", y la adapta para capturar dependencias espaciales y temporales en los datos de tráfico. La idea clave de STTN es modelizar las relaciones espacio-temporales entre distintos sensores de tráfico o ubicaciones a lo largo del tiempo teniendo en cuenta tanto la proximidad geográfica de los sensores como las dependencias temporales entre pasos temporales consecutivos. Al incorporar estos factores, la STTN pretende captar las complejas pautas y dinámicas del flujo de tráfico.

El modelo consta de dos componentes principales: el *Transformer* espacial y el *Transformer* temporal. La parte espacial se centra en captar las dependencias espaciales entre distintas ubicaciones. Utiliza mecanismos *self-attention* para modelar las relaciones entre los sensores de tráfico basándose en su proximidad espacial y en los patrones de tráfico. Mientras que la parte temporal, por su parte, se centra en modelar las dependencias temporales de los datos de tráfico y aprovecha los mecanismos *self-attention* para captar las relaciones temporales entre pasos temporales consecutivos. Atendiendo a los pasos temporales históricos relevantes, el *Transformer* temporal puede aprender los patrones temporales y las tendencias del flujo de tráfico.

Entonces, tanto el modelo espacial como el temporal se combinan para formar la arquitectura global de la STTN.

El propio modelo se entrena utilizando datos históricos de tráfico, donde los datos de entrada son una secuencia de observaciones del flujo de tráfico en varios lugares y pasos temporales, y la salida es la predicción del flujo de tráfico perteneciente a algunos pasos temporales más avanzados.

Al utilizar la arquitectura *Transformer*, STTN puede modelar eficazmente las dependencias de largo alcance y capturar las relaciones no lineales en los datos de flujo de tráfico. De hecho, ha demostrado un rendimiento prometedor en tareas de previsión de flujos de tráfico, superando a los métodos tradicionales y alcanzando resultados de vanguardia en varios conjuntos de datos de referencia. Debido a estos buenos resultados, se ha acabando aplicando el conocido modelo STTN en la gestión del tráfico, la planificación urbana y los sistemas de transporte inteligentes ya que, una predicción precisa del flujo de tráfico, como la que facilita este modelo, puede ayudar a optimizar el control de los semáforos, gestionar la congestión y mejorar la eficiencia general del transporte.

2.4.10.3. Transformer-XL (TRXL)

"Transformer-XL" (TRXL) es una arquitectura que amplía el *Transformer* original para manejar secuencias más largas y capturar dependencias a más largo plazo. Fue introducido en [Dai *et al.*, 2018] como solución a la limitación que sufría el *Transformer* original de [Vaswani *et al.*, 2017] de no poder procesar secuencias muy largas, como las que se encuentran en tareas de modelado de lenguaje o comprensión de documentos.

El principal desafío con el procesamiento de secuencias largas utilizando *Transformers* tradicionales son los requisitos computacionales y de memoria, ya que el mecanismo de *self-attention* escala cuadráticamente con la longitud de la secuencia. Además, los *Transformers* adolecen de falta de información posicional cuando se trata de secuencias más largas que las observadas durante el entrenamiento. TRXL aborda estos problemas introduciendo dos innovaciones principales: el mecanismo de recurrencia a nivel de segmento y el esquema de codificación posicional relativa.

Por un lado, el mecanismo de recurrencia a nivel de segmento permite a TRXL capturar dependencias más largas dividiendo la secuencia de entrada en segmentos y manteniendo una memoria recurrente de estados ocultos dentro de cada segmento. Esto permite al modelo recordar y reutilizar representaciones de segmentos anteriores, ampliando de forma efectiva la ventana de contexto más allá de la longitud fija del mecanismo de atención de *Transformer*. Por el otro lado, el esquema de codificación posicional relativa ayuda a TRXL a superar la limitación de la codificación posicional absoluta utilizada en los *Transformers* tradicionales. En lugar de utilizar incrustaciones de posición fija, TRXL incorpora codificaciones posicionales relativas que codifican las distancias relativas entre diferentes posiciones dentro de un segmento. Esto permite al modelo aprender representaciones que tienen en cuenta la posición y que se generalizan mejor a secuencias más largas.

Entonces, al combinar el mecanismo de recurrencia a nivel de segmento y el esquema de codificación posicional relativa, TRXL puede procesar secuencias largas con mayor eficacia y capturar dependencias que se extienden más allá de la ventana de contexto

fija de los *Transformers* tradicionales. Esto hace que TRXL sea idóneo para tareas que implican dependencias a largo plazo, como el modelado del lenguaje, la comprensión de documentos y la generación de textos.

2.5. Conclusiones

En este estudio, se explora la aplicación de *Transformers* espacio-temporales en la predicción de la demanda de transporte público. Sin embargo, antes de adentrarse en estos modelos espacio-temporales, se ha llevado a cabo una contextualización del problema, es decir, se ha seguido el siguiente orden: contextualizar el problema $\rightarrow$ cómo solucionar dicho problema.

En primer lugar, se realiza una descripción detallada de como a través de diferentes investigaciones se ha ido introduciendo el problema de la predicción de la demanda de movilidad del transporte público en el campo de la inteligencia artificial. Después, se lleva a cabo la introducción de los *Transformers* por comprender el origen y la introducción de los *Transformers*, centrándonos en el trabajo [Vaswani *et al.*, 2017]. Esta investigación pionera revolucionó el campo del "Natural Language Processing" y allanó el camino para la adopción generalizada de los *Transformers* en varios campos. Dentro de dicha introducción de los *Transformers* se profundiza en diferentes apartados de su estructura como los *embeddings*, la codificación posicional, el codificador, el decodificador y los mecanismos de atención (*self-attention* y *multi-head attention*).

A continuación, se ha profundizado en la evolución de los diferentes *Transformers* que se han ido desarrollando hasta la fecha, destacando algunas de las arquitecturas más influyentes hasta la fecha como [Jacob *et al.*, 2019], [Radford *et al.*, 2018], [Radford *et al.*, 2019] y [Raffel *et al.*, 2020], todas logrando un rendimiento de vanguardia en una variedad de tareas que incluyen comprensión, generación y traducción de idiomas. Además, se ha mencionado otro tipo de *Transformer* denominados *Transformers* de largo alcance.

Sobre la base de estos avances, se ha explorado el campo de los *Transformers* espacio-temporales mencionando y explicando la introducción de las dependencias espaciales y temporales y la aplicación de las series temporales en este tipo de modelos. Luego, al igual que para los *Transformers* más originales, se han mencionado trabajos notables que integran arquitecturas de *Transformers* con datos espacio-temporales, lo que permite un modelado eficiente de dinámicas e interacciones dentro de dimensiones espaciales y temporales. Estos modelos han mostrado resultados prometedores en campos tan variados como el pronóstico del tiempo, el pronóstico del tráfico y, ahora, el pronóstico de la demanda del transporte público.

En conclusión, el estado actual de los *Transformers* es increíblemente prometedor, ya que han demostrado su versatilidad y eficacia en diversos ámbitos. Su notable éxito en tareas de procesamiento del lenguaje natural los ha catapultado a la vanguardia de la investigación en la inteligencia artificial. Además, la adaptación de los *Transformers* al tratamiento de datos espacio-temporales ha abierto nuevas vías para su aplicación en la previsión del transporte y campos afines. Esta integración de información espacial y temporal tiene el potencial de revolucionar la precisión y eficacia de las predicciones en los sistemas de transporte. A medida que la investigación y el desarrollo sigan avanzando en los *Transformers*, cabe esperar arquitecturas aún más eficientes, una mayor interpretabilidad y la integración de técnicas de aprendizaje

por transferencia, lo que aumentará aún más sus capacidades e impacto. El futuro de los *Transformers* es prometedor y ofrece grandes oportunidades de avance en el análisis del transporte y en otros ámbitos.

Capítulo 3

Desarrollo

En este capítulo se van a describir la tareas realizadas en la parte técnica de este trabajo de fin de máster. En primer lugar se va a describir el problema a resolver, mencionando los conjuntos de datos seleccionados para llevar a cabo dicha resolución del problema. Después, se explicará el modelo STT-MDF ("Spatio Temporal Transformer for Mobility Demand Forecasting") junto a una descripción de su arquitectura. Luego, se mencionará la implementación que se ha llevado a cabo tanto para desarrollar el código como para realizar los diferentes experimentos. Finalmente, se mencionarán dichos experimentos realizados en la fase de desarrollo del trabajo.

3.1. Descripción del problema

La predicción de la demanda del transporte público en ciudades grandes como Chicago es de vital importancia para poder evitar el gran problema de congestión de las carreteras que sufren este tipo de ciudades. Por lo tanto, esta predicción, en este trabajo fin de máster, se ha llevado a cabo mediante el uso de un *Transformer*.

Sin embargo, dicha predicción se puede llevar a cabo de otras formas y así ha quedado reflejado en [Prado-Rujas *et al.*, 2023], investigación en la cual está basado este trabajo. Explorar y comprender diferentes formas de llevar a cabo dicha predicción permite un análisis exhaustivo del problema en cuestión y posibilita una decisión más informada sobre la selección del método más adecuado.

Para conseguir la resolución al problema planteado se han recompilado y combinado diferentes conjuntos de datos heterogéneos de la ciudad de Chicago para entrenar y evaluar el STT-MDF. Estos conjuntos de datos contienen información valiosa relacionada con el uso del transporte público, como datos históricos de usuarios, información demográfica, condiciones meteorológicas y otros factores relevantes que pueden influir en los patrones de demanda. Combinando y utilizando estos conjuntos de datos, el STT-MDF puede entrenarse y evaluarse eficazmente, capturando la confusa dinámica de la demanda de transporte público en la ciudad.

3.2. Conjuntos de Datos

Los conjuntos de datos seleccionados para ser utilizados en el STT-MDF, con el objetivo de brindar la información necesaria a dicho modelo y poder llevar a cabo una

predicción precisa, son conjuntos de datos que contienen una información precisa y actualizada de los diferentes transportes públicos de la ciudad. En el caso de este trabajo, nos vamos a fijar tanto en las bicicletas como en los taxis ya que son dos de los transportes públicos de la ciudad de Chicago más utilizados y sobretodo porque se puede obtener una gran cantidad de información sobre estos de una manera sencilla. Para encontrar dichos conjuntos de datos debemos buscar tanto en el portal de datos del ayuntamiento de Chicago [1] para el caso de los taxis, como en la web de la compañía de bicicletas Divvy[2] para el caso de las bicis. A continuación, se muestra el *mesh-grid* de la ciudad de Chicago de 500 metros x 500 metros cuya función será explicada más adelante. En esta figura (3.1) se representan las 801 zonas de taxis (puntos rojos) y las 684 zonas para dejar las bicicletas en Chicago (puntos azules). La imagen ha sido obtenida de [Prado-Rujas *et al.*, 2023], investigación que utiliza datos meteorológicos, el cual no es el caso de este trabajo de fin de máster por lo que es necesario ignorar el centro de la meteorología (punto verde).

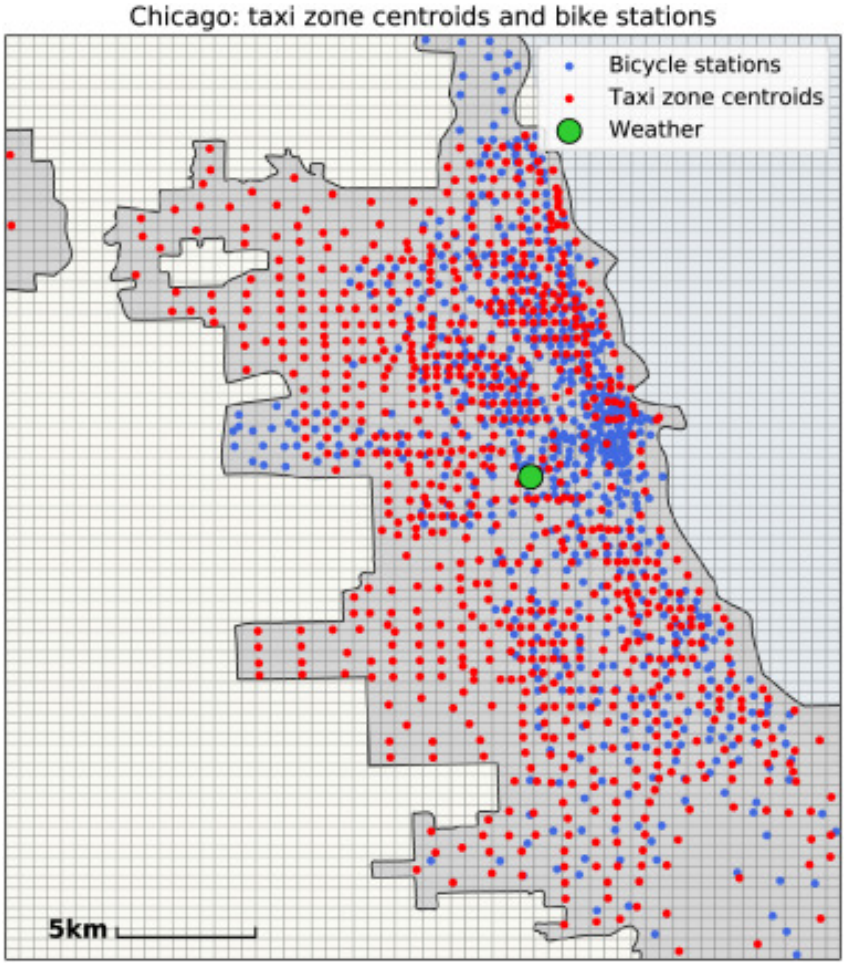


Figura 3.1: Mesh-grid de la ciudad de Chicago extraido de [Prado-Rujas *et al.*, 2023], página 4

.

[1] https://data.cityofchicago.org/
[2] https://divvybikes.com/

3.2.1. Taxis

Entre estos conjuntos de datos facilitados por el propio ayuntamiento de Chicago, se encuentran los conjuntos de datos que tienen la información sobre las zonas y horarios de recogida y bajada de taxis a partir del año 2013. Estos conjuntos de datos han sido utilizados para realizar los experimentos pertinentes para obtener la predicción de viajes de taxis. Los datos se pueden descargar en archivos CSV, uno por año. Dentro de estos archivos cada una de las filas representa un viaje de taxi y en cada una de estas filas aparece la siguiente información:

- Zona de recogida: Se basan en los distritos censales de la ciudad de Chicago, que en los últimos cien años no han cambiado o apenas han cambiado. Como se puede observar en la figura 3.1, existen 801 zonas de taxi (los puntos rojos de la imagen).
- Hora de recogida: La frecuencia con la que se obtienen los puntos de recogida de las zonas de taxis es de 15 minutos. Esta frecuencia es crucial para poder calcular el número de intervalos que existen en un día: $|T| = 4 \cdot 24 = 96$, ya que en 1 hora existen 4 intervalos de 15 minutos. Los conjuntos de datos mencionados en el apartado de *Taxis* son de 8 años diferentes, es por ello que a la hora de calcular los intervalos que existen todo ese tiempo se lleva a cabo esta operación: $|T| = 4 \cdot 24 \cdot (365 \cdot 8 + 2) = 280512$, sumando los 2 días de los años bisiestos correspondientes.

Originalmente, a través de los datos facilitados por el ayuntamiento de Chicago se obtienen 195 millones de viajes de taxis sin embargo, una gran cantidad de estos viajes tenían datos incompletos por lo que se han eliminado y el número de viajes ha terminado en 172 millones. Estos viajes han sido procesados e incorporados al conjunto de datos final que alimentará de información al STT-MDF mediante el código desarrollado en la investigación de [Prado-Rujas *et al.*, 2023]. En este procesamiento de viajes, en primer lugar, se han agrupado los viajes de taxis en diferentes contenedores dependiendo de la zona e intervalo de tiempo, de esta manera, obteniendo un conjunto de datos que registra el número de viajes que se han realizado durante un intervalo de tiempo $t \in T$ y zona $l \in L$ detallada: $X_{taxi} = \{x_t^l, t \in T, l \in L\}$.

Como se puede ver en la figura 3.1, los datos van a tratarse en una *mesh-grid* originalmente de 90x60 aunque dependiendo de la capacidad computacional será reducida en los diferentes experimentos realizados. Para poder crear esta *mesh-grid* tenemos que saber que $r = 90$ y $c = 60$. Entonces, se fueron insertando linealmente las *mesh-grid* en la malla de 90 x 60, utilizando el conjunto de datos X_{taxi} como entrada con sus respectivas geolocalizaciones. De esta manera, se construyó la *mesh-grid* de taxi con la forma de $|T|$ ***x 90 x 60***.

Finalmente, se obtiene el conjunto de datos con los viajes de taxis realizados dentro de la *mesh-grid* de 90 x 60 y posteriormente normalizados mediante la ecuación de:

$$x' = \frac{x - min(X)}{max(X) - min(X)} \tag{3.1}$$

Consiguiendo de esta manera el archivo **15m_map_90_60_taxi_norm_abs.h5** de tipo H5 que es un archivo de datos en Formato de Datos Jerárquicos (HDF) generalmente utilizados para almacenar grandes cantidades de datos números, gráficos y de texto.

En este caso, se utiliza este tipo de archivos debido a que se busca guardar los viajes realizados en 280512 intervalos de 15 minutos por lo que serán grandes cantidades de información. Una vez obtenido este tipo de archivo, será fácil de manejar dentro del código para alimentar al STT-MDF con toda su información.

3.2.2. Bicicletas

En cuanto a los conjuntos de datos de las bicicletas, a diferencia de los taxis que fueron obtenidos en el portal del ayuntamiento de la ciudad de Chicago, la información necesaria de las bicicletas ha sido obtenida entre los diferentes conjuntos de datos de la compañía de alquiler de bicicletas Divvy. Dicha compañía facilita a todos los usuarios datos anónimos de los viajes realizados, eliminando los trayectos con una duración menor a 1 minuto. Al igual que con los taxis, los conjuntos de datos se pueden descargar en archivos CSV individuales los cuales corresponden a periodos de duración variada. En cada una de las filas de estos CSV se muestra un viaje realizado y dentro de estas filas se encuentra la siguiente información:

- Estación de bicis: Algunas estaciones incluyen un identificador junto con un archivo de busqueda para poder conocer la ubicación de la estación, otras estaciones no tienen este archivo de búsqueda por lo que la geolocalización se basaba en la información obtenida en diferentes periodos y por último, existen otras estaciones las cuales incluyen una geolocalización exacta del lugar donde empezaba el viaje. Como se puede ver en la figura 3.1, existen 684 estaciones de bicicletas (puntos azules) dentro de la *mesh-grid* establecida de la ciudad de Chicago. La información de ubicación es dada en cada una de las filas del archivo mediante la longitud y latitud.

- Hora de inicio del viaje: Se toma de la misma manera que en los conjuntos de datos de los taxis, teniendo intervalos de 15 minutos y 8 años de información separada año por año en archivos CSV. Por lo tanto para calcular el número de intervalos que hay en estos 8 años se lleva a cabo el siguiente cálculo: $|T| = 4 \cdot 24 \cdot (365 \cdot 8 + 2) = 280512$.

Debido a que se está trabajando con grandes cantidades de información, existe la posibilidad de que existan datos incompletos, como es el caso del 0,35 % de los viajes en bicicletas que aparecen en el archivo sin la información de latitud y longitud en la que se ha empezado el viaje. Quedando restantes 24 millones de viajes en bicicleta. A diferencia de los conjuntos de datos de los taxis que facilitaban un conjunto de zonas como como posibles lugares de inicio del trayecto, la información que se obtiene de las bicicletas nos permite conocer el lugar desde que se empieza el viaje con exactitud.

De nuevo, mediante un código de la investigación de [Prado-Rujas *et al.*, 2023] se agrupan los viajes en bicicleta temporal y espacialmente obteniendo, de esta manera, el conjunto de datos $X_{bike} = \{x_t^l, t \in T, l \in L\}$. En el que L corresponde a la *mesh-grid* de 90 x 60 mencionada hasta el momento. Es decir, el conjunto de datos X_{bike} va insertando uno a uno los viajes en bicicleta que ocurren durante cada intervalo de tiempo $t \in T$ en cada zona de la *mesh-grid* $l \in L$. Entonces, se normaliza el conjunto de datos X_{bike} mediante la ecuación 3.1 obteniendo el archivo H5 final **15m_map_90_60_bike_norm_abs.h5** con una forma de $|T|$ *x 90 x 60* que alimentará al STT-MDF.

3.2.3. Tiempo

El STT-MDF es un *Transformer* espacio-temporal, por lo que una vez conocemos cómo se van a tratar los datos espaciales (bicicletas y taxis), es necesario conocer cómo se va trabajar la parte temporal. En este caso, el conjunto de datos tendrá una forma de $|T|$ *x 8* siendo esos 8 valores los siguientes:

- Coseno del tiempo del día en un periodo de 24 horas pasadas a segundos, estando el valor entre [-1, 1]. Se calcula mediante la siguiente formula en la que t es el numero de instantes correspondientes al conjunto de datos y *day* el número de instantes correspondientes a un día.

$$cos(\theta) = cos\frac{2t\pi}{day} \tag{3.2}$$

- Seno del tiempo del día en un periodo de 24 horas pasadas a segundos, estando el valor entre [-1, 1]. Se calcula mediante la siguiente formula en la que t es el numero de instantes correspondientes al conjunto de datos y *day* el número de instantes correspondientes a un día.

$$sin(\theta) = sin\frac{2t\pi}{day} \tag{3.3}$$

- Coseno del tiempo del día en un periodo de 7 días pasados a segundos, estando el valor entre [-1, 1]. Se calcula mediante la siguiente formula en la que t es el numero de instantes correspondientes al conjunto de datos y *week* el número de instantes correspondientes a una semana.

$$cos(\theta) = cos\frac{2t\pi}{week} \tag{3.4}$$

- Seno de la hora del día en un periodo de 7 días pasados a segundos, estando el valor entre [-1, 1]. Se calcula mediante la siguiente formula en la que t es el numero de instantes correspondientes al conjunto de datos y *week* el número de instantes correspondientes a una semana.

$$sin(\theta) = sin\frac{2t\pi}{week} \tag{3.5}$$

- Coseno de la hora del día en un periodo de 1 año pasado a segundos, estando el valor entre [-1, 1]. Se calcula mediante la siguiente formula en la que t es el numero de instantes correspondientes al conjunto de datos y *year* el número de instantes correspondientes a un año.

$$cos(\theta) = cos\frac{2t\pi}{year} \tag{3.6}$$

- Seno de la hora del día en un periodo de 1 año pasado a segundos, estando el valor entre [-1, 1]. Se calcula mediante la siguiente formula en la que t es el numero de instantes correspondientes al conjunto de datos y *year* el número de instantes correspondientes a un año.

$$sin(\theta) = sin\frac{2t\pi}{year} \tag{3.7}$$

- Diferenciador entre día de la semana y día de fin de semana, representando el valor 0 los días de la semana, mientras que el valor 1 los días del fin de semana.
- Diferenciador entre vacaciones y no vacaciones, representando el valor 0 los días no vacacionales, mientras que el valor 1 los días vacacionales.

Por ejemplo, el día 15 de diciembre del año 2000 con un conjunto de datos correspondiente a un año ($365 * 24 * 4 = 35040$ instancias) se reflejaría de la siguiente forma: $[0.234, 0.567, 0.890, 0.876, 0.543, 0.210, 0, 0]$. Los primeros seis valores se obtienen a través de las formulas 3.2, 3.3, 3.4, 3.5, 3.6, 3.7 mientras que, los últimos están señalando que es un día entre semana y no correspondiente a la época de vacacional.

3.3. Modelo propuesto STT-MDF

Partiendo del conocimiento expuesto en el capítulo 2, en este apartado del capitulo del desarrollo se va a explicar el STT-MDF tanto describiendo su arquitectura como mencionando la implementación que se ha llevado a cabo para desarrollarlo y realizar las experimentos pertinentes.

3.3.1. Arquitectura

En este trabajo fin de máster, se empleó una arquitectura basada en el *Transformer* codificador-decodificador con el objetivo mencionado anteriormente de predecir la demanda del transporte público (taxis y bicicletas) de la ciudad de Chicago. La elección de esta arquitectura, la cual se puede observar en la figura 3.2, estuvo motivada por su éxito en diversas tareas de procesamiento del lenguaje natural y visión por ordenador, así como por su capacidad para capturar eficazmente dependencias de largo alcance. Esto último es la clave de los *Transformers*, porque pese a que estos modelos sean actualmente algo novedosos y no se sabe mucho sobre estos, se conoce que trabajan mejor cuando trabajan con una gran cantidad de información y conjuntos de datos de gran tamaño, como es el caso.

La arquitectura utilizada para desarrollar el STT-MDF está principalmente basada en la investigación de [Grigsby *et al.*, 2021]. Este estudio presenta un novedoso enfoque que aborda los retos de la previsión de datos espacio-temporales dinámicos, como los patrones meteorológicos, el flujo de tráfico y la dinámica de la población, que presentan dependencias temporales y espaciales complejas.

En cuanto a la propia arquitectura, su idea principal reside en su capacidad para aprender representaciones contextuales de las secuencias de entrada, capturando al mismo tiempo las dependencias locales y globales. Dicha arquitectura está compuesta de capas *self-attention* apiladas, redes neuronales *feed-forward* y conexiones residuales, que en conjunto permiten a la red modelar eficazmente las interacciones de largo alcance dentro de los datos de entrada. Su estructura general consta de una red codificadora encargada de captar los patrones espacio-temporales de los datos de entrada y una red decodificadora que genera predicciones futuras basadas en la información codificada. Tanto el codificador como el decodificador se componen de varias capas, cada una de las cuales consta de un mecanismo de *self-attention* y redes neuronales *feed-forward* en función de la posición.

Para poder alimentar las capas del codificador y las del decodificador es necesario que

los conjuntos de datos pasen por la fase de embedding. Por el lado del codificador para mejorar aún más la capacidad del modelo de captar dependencias de largo alcance, la arquitectura *Transformer* incorpora la codificación posicional. La idea principal de esta codificación posicional es añadir un conjunto de vectores de longitud fija a los *embeddings* de entrada, donde cada vector corresponde a una posición en la secuencia de entrada. Estos vectores de codificación posicional se crean con el fin de que contengan información tanto sobre la posición como sobre la frecuencia del token/palabra en la secuencia. Para ello se utilizan las siguientes funciones seno (*sin*) (figura 3.8) y coseno (*cos*) (figura 3.9) de distintas frecuencias y fases.

$$PE_{(pos,2i)} = sin(\frac{pos}{1000^{2i/d_{model}}}) \tag{3.8}$$

$$PE_{(pos,2i+1.0)} = cos(\frac{pos}{1000^{2i/d_{model}}}) \tag{3.9}$$

En estas dos formulas, *pos* es la posición del token/palabra correspondiente en la secuencia, *i* es el índice del vector de codificación posicional (*PE*) y d_{model} es la dimensión de los *embeddings* de entrada. Al añadir los vectores de codificación posicional a los *embeddings* de entrada, el modelo es capaz de diferenciar los tokens en función de su posición en la secuencia. De este modo, los mecanismos *self-attention* atienden tanto al contenido de dichos tokens como a su información posicional, lo que permite al modelo captar el orden y las relaciones entre los distintos elementos de la secuencia. En otras palabras, de esta manera se obtenía el contexto de cada token. Cabe destacar que la codificación posicional sólo se añade una vez a los mencionados *embeddings* de entrada y permanece constante durante todas las capas del modelo, ayudando de esta manera a manejar mejor la información secuencial.

Por otro lado, el lado del decodificador, se utiliza un mecanismo adicional *cross-attention* que le permite atender a las representaciones codificadas generadas por el codificador. Este mecanismo de atención cruzada ayuda al decodificador a centrarse en la información espacio-temporal relevante de los datos de entrada, al tiempo que genera predicciones precisas para los futuros pasos temporales.

3.3.1.1. Embedding

En el STT-MDF, el *embedding* desempeña un papel crucial en la transformación de los datos de entrada en una representación continua que pueda ser procesada por las capas posteriores del modelo. El proceso *embedding* asigna cada elemento discreto de entrada, como palabras o tokens, a una representación vectorial continua, también conocida como *embedding*. Las propias capas de *embedding* sirven de paso inicial en la arquitectura del STT-MDF y se aplica por separado a las secuencias de entrada tanto en el codificador como en el decodificador.

Al tratarse de un caso en el que la información con la que se trabaja es espacial y temporal, cada uno de estos tipos de datos incluyen características diferentes (ubicaciones geográficas o información temporal). Cada característica suele representarse como una variable categórica o un valor numérico. Para incorporar estas diversas características de entrada al modelo, se utilizan capas de *embedding* específicas para convertirlas en representaciones continuas. A continuación, se van a explicar las diferentes métodos aplicados a través de las capas *embedding* que forman parte de la arquitectura del STT-MDF:

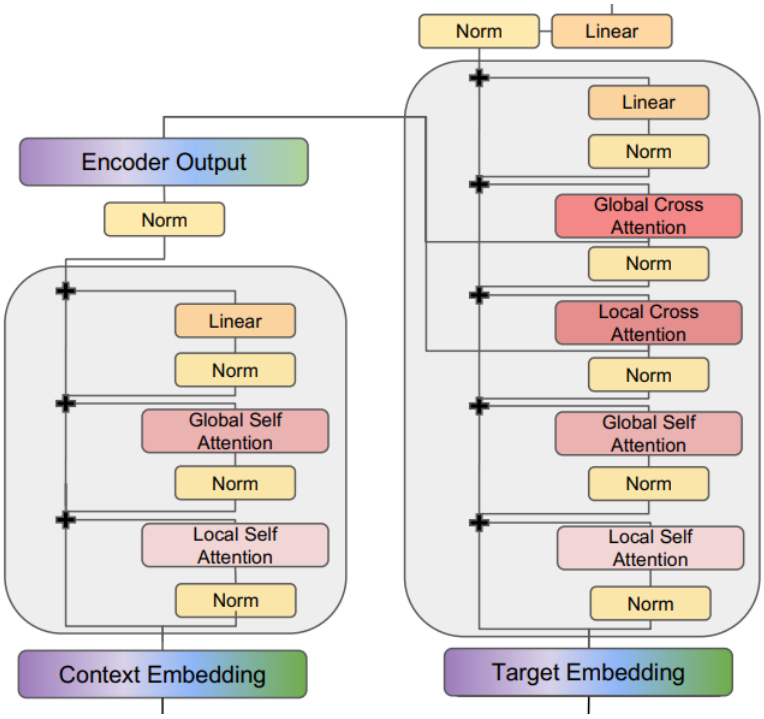


Figura 3.2: Estructura codificador-decodificador del STT-MDF extraído de [Grigsby *et al.*, 2021], página 4.

- **Secuencias espacio-temporales:** Para poder leer las secuencias de entrada, estas son pasadas por un *embedding* de tokens utilizando la técnica de *flattening* basada en la conversión de una estructura de datos multidimensional en una estructura unidimensional. Esta técnica se utiliza sobre cada vector y_t (vector de las dependencias espaciales *target* con una copia de su *timestamp* x_t, obteniendo de esta manera una nueva secuencia *Z*. Entonces, se realiza un *embedding* de dicha nueva secuencia, obteniendo *Z'* que al ser pasada por el modelo, la matriz de atención *A* representa un grafo espacio-temporal reflejando en él un camino directo entre cada variable en cada paso de tiempo. Al output de este *embedding* se le llama valor+tiempo.

- **Tiempo:** Las variables de tiempo, junto a las espaciales, son una de los dos tipos de variables que se encuentran en los conjuntos de datos utilizados. Para tratar con estas, es necesario utilizar el método *Time2Vec* de [Kazemi *et al.*, 2019] basado en capturar y representar las características temporales en los datos de series de tiempo de una manera más efectiva. Time2Vec evita las representaciones tradicionales como las fechas o los índices de tiempo y utiliza una representación vectorial de tiempo que se aprende durante el proceso de entrenamiento. Otros métodos como el seq2seq no pueden ser utilizados para tratar este tipo de información porque generalmente descartan información temporal explicita. En cuanto a la función del Time2Vec en este trabajo, asigna x_t a patrones sinusoidales de desplazamientos y longitudes de onda aprendidos. Esto es de gran ayuda a la hora de representar relaciones periódicas que se extienden más allá de la longitud limitada de la secuencia contextual. Entonces, los valores de las variables concatenados y los *embeddings* temporales se proyectan a la dimensión de entrada del modelo con una capa *feed-forward*.

- **Embedding de posición**: Es necesario añadir este *embedding* de posición ya que los *Transformers*, por defecto, no pueden interpretar el orden de los tokens y añadiendo esta técnica posicional se inicializan vectores de *embeddings* d-dimensionales para cada paso en el tiempo hasta alcanzar la longitud máxima de la secuencia.
- **Espacio:** Primero, se han creado las secuencias espacio-temporales aplicando la técnica del *flattenig* sobre N variables en tokens separados. Después, cada uno de estos tokens recibe una copia de la información temporal x asociada a cada una de dichas variables para su *embedding* de valor+tiempo. Finalmente, se le asigna un índice mediante el *embedding* de posición para marcar el orden en el paso del tiempo original, de manera que cada posición aparece ahora N veces. Entonces, una vez tenemos todo ese proceso realizado, se diferencian las variables en cada paso temporal con un *embedding* de variable adicional. Se inicializan N vectores *embedding* d-dimensionales para cada índice de variable, de forma muy similar que en el *embedding* de posición. Por lo tanto, esto conlleva que las representaciones de las variables se inicialicen aleatoriamente y se aprendan de principio a fin durante la fase de entrenamiento del STT-MDF. Como se ha podido observar a lo largo del documento, y especialmente en este apartado de, existen dos conjuntos de *embeddings* (espacial y temporal) que crean interesantes paralelismos entre los modelos de series temporales (como el propuesto) y los *Transformers* especializados en visión por computador.
- **Datos incompletos:** Pese a que los conjuntos de datos han sido previamente preparados específicamente para evitar que dentro de estos existiese información incompleta, es necesario aplicar una técnica de *embedding* que nos asegure que no existe ningún tipo de dato vació entre los que van a pasar por las diferentes capas del modelo. Además, en los conjuntos de datos tan grandes relacionados con el transporte público es muy común que algún dato como la localización inicial del viaje o la hora de salida estén incompletos. Por lo tanto, aplicando el *embedding* a cada variable sobre su propio token separado proporciona una flexibilidad de dejar valores perdidos en la canalización de datos, reemplazarlos por ceros en el pase hacia adelante o *forward pass*, y luego decirle al modelo cuándo faltaban valores originalmente con una *given embedding* binaria. Una vez conseguido esto, los *embeddings* de valor+tiempo, variable, posición y *given embedding* se suman para crear la secuencia de entrada espacio-temporal final.

3.3.1.2. Mecanismos de atención

El objetivo principal de los mecanismos de atención que se han utilizado para formar la arquitectura del STT-MDF es capturar las dependencias entre las diferentes posiciones de la secuencia espacio-temporal de entrada. Como se ha comentado en diferentes apartados del capítulo 2, los mecanismos de atención son los componentes esenciales de la arquitectura estándar de los *Transformers* y son sumamente importante a la hora de capturar eficazmente dependencias de largo alcance. En el STT-MDF, aparecen dos tipos diferentes de mecanismos de atención:

- **Self-attention**: También conocido como *intra-attention*, permite al modelo analizar y entender las relaciones entre los diferentes elementos de una misma secuencia y de esta manera, centrarse en parte especificas de dicha secuencia capturando dependencias de largo alcance de manera efectiva. Para poder

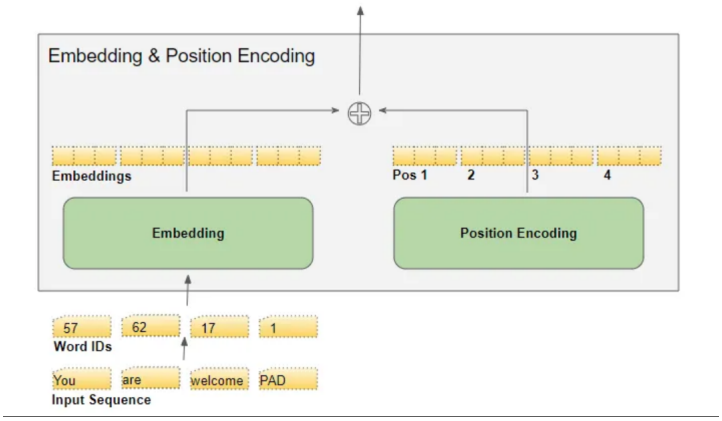


Figura 3.3: Ejemplo del funcionamiento de las capas *embedding* y el *positional encoding* de [Manish, 2017].

implementar el mecanismo *self-attention* o *intra-attention*, la secuencia espacio-temporal de entrada se transforma en tres representaciones obtenidas tras aplicar una transformación linear de los elementos dee entrada: Consulta o *Query* (Q), Clave o *Key* (K) y Valor o *Value* (V). La idea es que cada elemento en la secuencia genere sus propias consultas, claves y valores. Entonces, los pesos de atención o puntuaciones de atención, valores numéricos que representan la importancia o la relevancia asignada a los diferentes elementos en los mecanismos de atención, se calculan midiendo la similitud entre la consulta de un elemento concreto y las claves de todos los demás elementos de la secuencia. Este cálculo de similitud puede realizarse utilizando distintos métodos, como el *dot product* o el *scaled dot product*. El resultado es una puntuación de similitud para cada par clave-elemento. Estas puntuaciones de similitud se transforman un pesos de atención mediante una función *softmax*, que normaliza las puntuaciones y garantiza que entre todas sumen 1. Los elementos con mayor peso de atención serán los más relevantes de cada elemento. Una vez obtenidos los pesos de atención, se aplican a los elementos de valor correspondientes. Cada elemento de valor se multiplica por su peso de atención y se suman los valores ponderados. Esta suma de valores ponderados representa el resultado final del mecanismo *self-attention* para el elemento actual. Al considerar las contribuciones de todos los elementos relevantes, el modelo capta las relaciones y dependencias contextuales dentro de la secuencia.

A continuación, podemos observar en primer lugar en la figura 3.4 la arquitectura que forma el mecanismo recientemente introducido. Después, se encuentra un ejemplo de la matriz que proporciona dicho mecanismo *self-attention* como output cada vez que las secuencias de entrada pasan por el. Esto se puede encontrar en la figura 3.5.

- **Cross-attention**: Amplía las capacidades del mecanismo *self-attention* recientemente mencionado al permitir que el modelo capte las relaciones entre distintas secuencias o contextos. Permite al modelo incorporar información de múltiples fuentes o modalidades, mejorando su capacidad para comprender relaciones complejas. A diferencia del mecanismo anterior, en el *cross-attention* hay dos conjuntos de representaciones: Consulta o *Query*(*Q*) y Clave-Valor o *Key-Value*

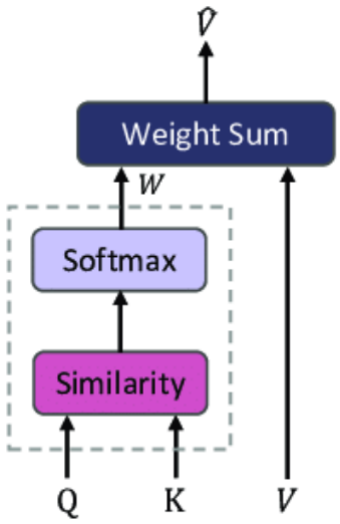


Figura 3.4: Estructura del mecanismo *self-attention* utilizado en el STT-MDF y obtenido de [Vaswani *et al.*, 2017].

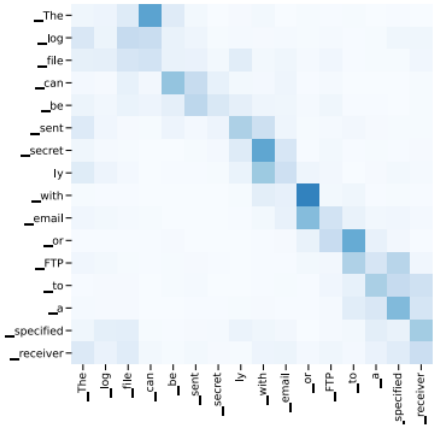


Figura 3.5: Ejemplo de la matriz resultante de un mecanismo *self-attention* extraído de [Wand and Li, 2018]

(KV). Las consultas se derivan de elementos de una secuencia, mientras que los pares clave-valor se derivan de elementos de otra secuencia. Este diseño permite al modelo atender a los elementos de la secuencia clave-valor basándose en las consultas. En cuanto a los pesos de atención el mecanismo *cross-attention*, al igual que el mecanismo *self-attention*, consiste en calcular los pesos de atención. Para cada elemento de la consulta, el modelo mide la similitud entre la consulta y las claves de la secuencia clave-valor. Este cálculo de similitud se realiza utilizando las mismas funciones de similitud que en el primer mecanismo mencionado. Las puntuaciones de similitud resultantes se transforman en pesos de atención mediante una función softmax. Estos pesos de atención determinan la importancia relativa de cada elemento clave-valor con respecto a cada elemento de consulta. A continuación, los pesos de atención se aplican a los elementos de valor correspondientes de la secuencia clave-valor y cada uno de estos elementos de valor se multiplican por su peso de atención y se suman los valores ponderados la cual sera el resultado final de este segundo mecanis-

mo de atención para cada elemento de consulta. Al incorporar la información de los elementos relevantes de la secuencia clave-valor, el modelo capta las interdependencias y relaciones entre las dos secuencias.

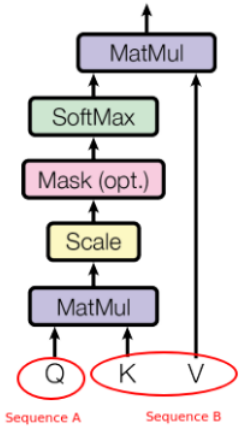


Figura 3.6: Estructura del mecanismo *cross-attention* utilizado en el STT-MDF introducido en [Vaswani *et al.*, 2017].

La potencia de estos dos mecanismos de atención reside en su capacidad para captar dependencias de largo alcance, considerar el contexto global y modelar eficazmente las relaciones dentro de las secuencias y entre ellas. Estos mecanismos han transformado el campo del *deep learning* y se han convertido en fundamentales en diversas aplicaciones, como la traducción automática, el análisis de sentimientos y el reconocimiento de imágenes, etc. Comprender e implementar eficazmente los mecanismos de *self-attention* y *cross-attention* es esencial para avanzar en las capacidades de los *Transformers*. Cabe mencionar que estos mecanismos de atención aparecen en el STT-MDF unos junto a otros, es decir, no hay un solo mecanismo de atención de cada tipo en el codificador o en el decodificador. Por lo tanto, se forma la extensión de los mecanismos de atención llamada *multi-head attention* la cual se basa en que en lugar de tener tan solo un conjunto de matrices como salida de la mecanismo de atención, se van a obtener uno por cada uno de los mecanismos de aparezcan en la estructura. De esta manera, se obtendrán múltiples conjuntos de estas matrices.

De nuevo, basándome en la investigación de [Grigsby *et al.*, 2021], se ha tenido en cuenta que la secuencia más larga puede complicar el aprendizaje de los problemas con N (longitud de la secuencia) más grandes. Entonces, se ha añadido a la arquitectura general un sesgo estructural hacia la secuencia de cada variable propia del token con módulos de atención **local** en cada capa del codificador y decodificador. En la capa **local**, cada token responde a los pasos temporales de su propia variable espacial. Por otro lado, esto no significa que la capa de atención **global** vaya a ser simplificada separando la atención temporal de la espacial, sino que, los tokens atienden a cada token en la secuencia de su propia variable y, además, luego atenderán a cada token en toda la secuencia **global** espacio-temporal. Todo esto se ve reflejado en la arquitectura del modelo (figura 3.2) como *Global Self Attention*, *Local Self Attention*, *Local Cross Attention* y *Global Cross Attention*.

Los mecanismos *self-attention* aparecerán tanto en las capas de codificador como en las del decodificador, mientras que los mecanismos *cross-attention* tan solo aparecerán en las capas del decodificador debido a que este comparará la secuencia de salida

del codificador y buscará capturar la relación entre dicha secuencia codificada y la secuencia *target* o objetivo.

3.3.1.3. Codificador

En la arquitectura del STT-MDF, el codificador tiene un rol bastante importante a la hora de capturar y codificar la información espacio-temporal de la información de datos de entrada. El codificador procesa la secuencia de entrada, que consiste en observaciones históricas, y genera representaciones codificadas que capturan las características y patrones relevantes. Después, estas representaciones o secuencias codificadas son enviadas al decodificador. Esta arquitectura del codificador combina la potencia de los *Transformers*, que han demostrado un rendimiento excepcional en el procesamiento del lenguaje natural (NLP) y la generación de imágenes, con modificaciones específicas para poder abordar el reto propuesto en este trabajo fin de máster de realizar predicciones espacio-temporales.

En primer lugar, la entrada del codificador es una secuencia de datos de entrada que son observaciones espacio-temporales. Estos elementos de entrada se transforman primero en representaciones vectoriales continuas, conocidos como *embeddings*, mediante una capa de *embedding* que asigna cada elemento de entrada a un espacio vectorial continuo de dimensiones inferiores, lo que permite al modelo procesar eficazmente los datos de entrada. Una vez se han pasado estos datos de entrada por las capas de *embedding*, se debe aplicar la codificación posicional mencionada en anteriores apartados con el objetivo principal de incorporar información sobre el orden y la posición de los elementos de entrada dentro de cada secuencia. La codificación posicional es necesaria para poder captar el aspecto temporal de los datos, lo cual permite al modelo comprender la dinámica temporal y las dependencias entre los distintos pasos temporales. Generalmente, esto se consigue añadiendo funciones sinusoidales de diferentes frecuencias y fases a los *embeddings* de entrada obtenidos en el paso anterior, proporcionando de esta manera una señal posicional del modelo.

En segundo lugar, una vez los datos de entrada han pasado por las capas de *embedding* y se ha realizado su pertinente codificación posicional, es necesario conocer la estructura de este codificador en especifico. El codificador consiste de un número determinado de capas de codificadores que puede variar dependiendo del resultado que se quiera obtener en cada uno de los experimentos, en el caso de este trabajo generalmente son entre 2 y 4 capas. Cada una de estas capas está formada por dos sub-capas de mecanismos de atención *self-attention* siendo una *Local Self Attention* y la otra *Global Self Attention*, una sub-capa de normalización antes de las dos sub-capas de mecanismos *self-attention* y de la sub-capa "Feed-Forward Neural Network" (FFNN) siguiendo la arquitectura *Pre-Norm* de los *Transformers* [Xiong *et al.*, 2020] y utilizando la técnica de normalización *Batch-Norm* [Ioffe and Szeged, 2015]. La FFNN mencionada al final aparece en la arquitectura del STT-MDF (figura 3.2) como *Linear*. A continuación, se van a explicar cada uno de los componentes de la estructura de cada capa del codificador:

- **Input** y *Output*: Pese a que se ha explicado recientemente como se trabajan los datos de entrada de la primera cada del codificador, también es importante saber que *output* de cada capa del codificador es pasado como *input* de la siguiente capa del codificador hasta que se alcanza el numero final de capas. En este momento, los datos de salida obtenidos tras pasar todas las sub-capas de

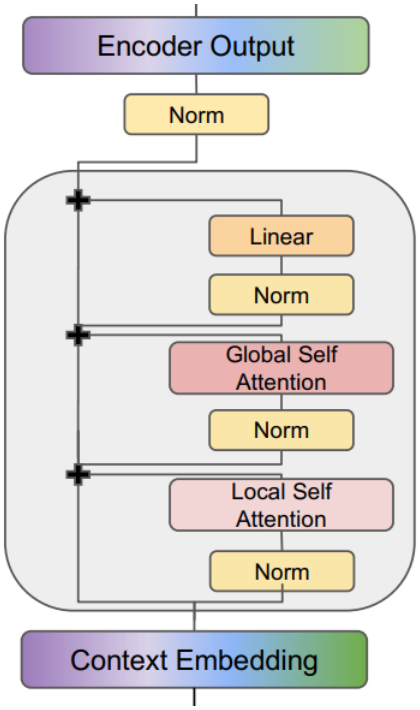


Figura 3.7: Estructura del codificador del STT-MDF obtenida de [Grigsby *et al.*, 2021].

cada capa del codificador se han almacenado en la variable x_1 a la cual se le añade el valor de la variable *x* original, obteniendo de esta manera la variable *output* la cual se va a pasar por una última capa de normalización para alimentar estos datos obtenidos y normalizados al decodificador. Esta ultima capa de normalización no se encuentra dentro de las capas de codificador ya que, como se acaba de explicar, tan solo entra en acción una vez se han superado todas estas capas.

- **Normalización:** Para las capas de normalización, se sigue la arquitectura mencionada recientemente de *Pre-Norm* [Xiong *et al.*, 2020] la cual es una alternativa eficaz para situaciones como la de este trabajo, ya que es capaz de mitigar problemas como la desaparición o explosión de gradientes en redes neuronales profundas. Como su propio nombre indica y como se ha aclarado anteriormente, la normalización de se aplica al input de cada sub-capa antes de que la operación de esta sub-capa se realice, es decir, se lleva a cabo la normalización del input *x* de los mecanismos de atención *Local Self Attention* y *Global Self Attention* y de la sub-capa *Feed Forward* antes de que cada una de estas sub-capas realicen su función. En cuanto a la técnica de normalización utilizada, se ha utilizado la *Batch-Norm*. Fue introducida en [Ioffe and Szeged, 2015] y lo que hace es normalizar las activaciones de cada capa basándose en las estadísticas calculadas sobre un pequeño lote de ejemplos de entrenamiento.

- **Local Self Attention**: Tras realizar la normalización *Batch-Norm* de la secuencia de entrada *x*, se obtiene x_1. Entonces, entra en acción el mecanismo *Local Self Attention* empezando por una localización de los tokens de la secuencia de entrada x_1 cuyo objetivo es centrar o restringir la atención a tokens específicos dentro de la secuencia normalizada. Después, se ejecuta la propia función de

Local Self Attention que ha sido explicada anteriormente con x_1 como consulta, clave y valor y una máscara de atención *self_mask_seq*. La secuencia de salida de este mecanismo de atención se almacena de nuevo en x_1, mientras que las puntuaciones de atención son ignoradas. Dicha secuencia de salida se pasa por una función de localización inversa, que como su propio nombre indica lo que hace es invertir los efectos de la función localización anteriormente explicada, almacenando el nuevo valor obtenido en la variable x_1. Finalmente, a esta variable se le aplica un *dropout* y se le suma el valor original de la variable *x* dando como resultado la secuencia de salida que se va a almacenar de nuevo en esta variable *x*.

- **Global Self Attention**: Tras realizar la normalización *Batch-Norm* de la secuencia de salida *x* del anterior mecanismo de atención, se obtiene x_1. A esta variable se le aplica una función *Window Time* para dividir la secuencia de entrada de este mecanismo de atención, la secuencia de salida del mecanismo de atención anterior, en ventanas o incluso segmentos más pequeños donde cada ventana representa un subconjunto de tokens consecutivos. El objetivo de esta división es encontrar dependencias locales o información contextual dentro de ventanas específicas de la secuencia. La función *Window Time* también se debe aplicar sobre la variable *self_mask_seq* ya que, a la hora de llamar el método *Global Self Attention* se utiliza x_1 como clave, consulta y valor y *self_mask_seq* como máscara de atención por lo que, en caso de aplicar la división en diferentes ventanas, se debe aplicar en ambas variables para que cumplan con los mismos requisitos. Los resultados del método *Global Self Attention* se almacenan en la variable x_1 y las puntuaciones de atención se asignan a la variable *attn*. Después, se aplica la funcion *ReverseWindowTime* a las variables x_1 y *self_mask_seq* para volver a agrupar las diferentes ventanas divididas. Finalmente, a este resultado se le aplica el *dropout* establecido para los mecanismos de atención y se le añade el valor original de *x*, almacenando el resultado de esta operación en la propia variable *x*.

- **Feed Forward**: Tras obtener el resultado de la función *Global Self Attention x*, este se pasa por la sub-capa de normalización obteniendo de esta manera x_1. Dicha variable entonces, se pasa por una sub-capa *feed forward* compuestas de dos capas convolucionales de una dimensión (Conv1D). La primera de estas capas se aplica sobre la variable normalizada x_1 la cual pasa por dicha capa convolucional primero, entrando por el canal de entrada cuyo valor de dimensión es el de la variable *d_model* y segundo, saliendo por el canal de salida con el valor de dimensión que contenga la variable *d_ff*. Una vez pasada la variable por dicha sub-capa, se lleva a cabo una función de activación (*relu* o *gelu*) y finalmente, un *dropout* con un valor especifico para esta sub-capa *feed forward*. El valor obtenido tras la primera de las dos Conv1D que forman esta última sub-capa del codificador, se almacena en la variable x_1 la cual será pasada por la segunda Conv1D que contiene los valores de dimensiones de los canales de entrada y salida al revés que en la primera Conv1D de manera que x_1 proporcionado en la salida mantendrá el mismo valor de dimensión que al entrar por la primera capa convolucional. Además, al valor obtenido después de aplicar dicha segunda convolución, se le aplicará de nuevo el mismo valor *dropout* que a la primera convolución y se almacenará el resultado en la variable x_1.

3.3.1.4. Decodificador

El decodificador dentro de la arquitectura propuesta es sumamente importante a la hora de generar predicciones espacio-temporales precisas y dinámicas. Toma las representaciones codificadas de la secuencia de entrada, producidas por el codificador, y genera predicciones para futuras tramas espacio-temporales. El decodificador consta de varias capas, denominadas capas decodificadoras, cuya cantidad es un hiperparámetro que dependiendo de los objetivos del experimentos y de los resultados que se desean obtener será una u otra. Cada capa decodificadora consta de cuatro subcapas que son diferentes mecanismos de atención, una sub-capa *feed forward* y una sub-capa de normalización basada en la técnica *Batch-Norm* [Ioffe and Szeged, 2015] antes de cada una de las demás sub-capas siguiendo la estructura *Pre-Norm* de los *Transformers* [Xiong *et al.*, 2020]. Los cuatro mecanismos diferentes mencionados son: *Local Self Attention*, *Global Self Attention*, *Local Cross Attention* y *Global Cross Attention*. De esta manera, el decodificador combina los mecanismos de *self-attention* y *cross-attention*, lo que permite al modelo prestar atención a distintas partes de la entrada y aprender las relaciones contextuales.

En el caso del decodificador, también se trabaja con la capa de *embedding* ya que recibe las secuencias de entrada espacio-temporales por dos sitios diferentes (salida del codificador y datos *target*). Como se ha comentado, el objetivo del decodificador es generar predicciones futuras basándose en los datos de entrada espacio-temporales codificados y para ello necesita comparar dichos datos con los datos pasados por la capa de *embedding* que los cuales se llamarán objetivo o *target*. Por lo tanto, los datos *target* deben ser tratados de la misma manera que los datos de entrada del codificador puesto que se trata del mismo tipo de secuencias espacio-temporales pero en este caso, secuencias de un paso temporal más avanzado que se compararán con los datos espacio-temporales de salida del decodificador. Es decir, para que se traten de la misma manera, se deben convertir en primer lugar en representaciones vectoriales y después se les aplicará una codificación posicional. Estos mismos pasos se han explicado en el apartado del codificador ya que, se podría decir que dentro de cada capa del decodificador existe un capa codificadora que abarca desde que los datos *target* se pasan por las capas *embedding* hasta que los dos mecanismos *self-attention* proporcionan sus pertinentes datos trabajados a los dos mecanismos restantes denominados *Local Cross Attention* y *Global Cross Attention*.

- **Input** y *Output*: A diferencia del codificador, los datos de entrada en el decodificador se obtienen de dos sitios diferentes. Por un lado, se encuentran los datos proporcionados por el codificador, los cuales han sido pasados por sus múltiples capas hasta finalmente ser normalizados y pasados al primer mecanismo *cross-attention*. Por el otro lado, se encuentran los datos mencionados recientemente como datos *target* los cuales, como se ha explicado, pasan por las capas de *embedding* y los mecanismos *self-attention* hasta encontrarse con los datos codificados en el primer mecanismo *cross-attention*. Estos datos se pasan por cada una de las capas del decodificador y una vez pasadas todas, se normalizan y se pasan por una capa *Feed Forward* obteniendo de esta manera los resultados o predicciones finales.
- **Normalización:** Las sub-capas de normalización en el decodificador siguen la misma estructura que en el codificador conocida como *Pre-Norm* introducida en [Xiong *et al.*, 2020] y en cada una de estas sub-capas también se utiliza la

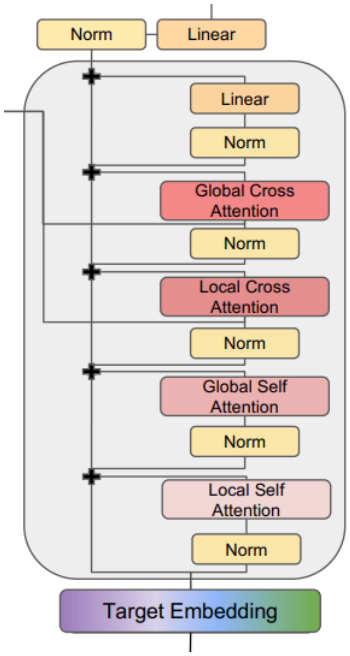


Figura 3.8: Estructura del decodificador del STT-MDF obtenida de [Grigsby *et al.*, 2021].

misma técnica de normalización que en el codificador denominada *Batch-Norm* de [Ioffe and Szeged, 2015].

- **Local Self Attention** y *Global Self Attention*: Ambos mecanismos *self-attention* no necesitan ser explicados puesto que llevan a cabo exactamente la misma función que en el codificador. La única diferencia es que los datos que reciben estos mecanismos pertenecen a pasos temporales más avanzados que los del codificador y que los datos de salida que proporcionen, no serán los datos de salida del decodificador sino que deberán pasar primero por los mecanismos *cross-attention* que se van a explican a continuación.

- **Local Cross Attention**: Como se ha explicado en punto de normalización, se sigue la estructura basada en normalizar los datos antes de que entren a los diferentes mecanismos. Por lo tanto, se normaliza la variable *x* la cual ha sido obtenida tras aplicar las diferentes funciones de los mecanismos *Local Self Attention* y *Global Self Attention* sobre los datos *target*, obteniendo así x_1. Entonces, se lleva a cabo una localización de dicha variable ya normalizada y de la variable *cross* la cual contiene los datos de salida del codificador. Esta localización se lleva a cabo para ayudar al mecanismo de atención a elegir y centrarse en los tokens más importantes de ambas variables. Una vez realizada la localización, se lleva a cabo el método del mecanismo *Local Cross Attention* que recibe como par de clave-valor la variable que contiene los datos codificados y recientemente localizados *cross*, como consulta la variable que contiene los datos *target* x_1 y como mascará de atención la variable *cross_mask_seq*. El resultado de este mecanismo almacenado en la variable x_1 se pasa por una localización inversa para mantener el formato inicial, se le aplica el *dropout* correspondiente y se le suma el valor de la variable original *x*. El resultado de esta operación se guarda de nuevo en la variable *x*.

- **Global Cross Attention**: Antes llamar al método de este mecanismo, al igual que en cada uno de estos mecanismos, se normalizan los datos de salida almacenados en *x* tras ejecutar el *Local Cross Attention* obteniendo de está manera la variable normalizada x_1. Al tratarse de un mecanismo *cross-attention* tienen influencia dos tipos de datos diferentes: x_1 (datos de salida normalizados del mecanismo anterior) y *cross* (datos codificados). Entonces, al igual que con el mecanismo Global Self Attention a la hora de realizar la división en un número de ventanas determinado mediante el método *Window Time*, se debe llevar a cabo en este caso para las variables x_1, *cross* y la propia máscara de atención cuya variable tiene el nombre *cross_mask_seq* (la máscara de atención cumple la misma función que para los mecanismos *self-attention*). Una vez realizada dicha división se llama al método *Global Self Attention* con la variable x_1 como consulta y la variable *cross* como par de clave-valor. El resultado de dicho mecanismo se guarda en la variable x_1 y las puntuaciones de atención en *attn*. Después, se invierten los efectos de la división de ventanas mediante el método *ReverseWindowTime* para las tres variables involucradas. Finalmente, se ejecuta el *dropout* correspondiente a los mecanismos de atención a los datos de salida obtenidos tras la ejecución de estos (x_1) y, además, se le suma al resultado del dicho *dropout* el valor original de la variable *x*. El resultado de esta operación se guarda de nuevo en la variable *x*.
- **Feed Forward**: Por último, esta capa *feed forward* tiene la misma función que en el codificador por lo que no es necesaria su explicación en este punto. Sin embargo, existe una diferencia la cual ha sido mencionada en el punto de **input** y *output* basada en que una vez finalizadas las capas de decodificador, esta capa *feed forward* vuelve a ser utilizada para finalmente obtener los resultados deseados.

3.3.2. Implementación

El trabajo realizado está principalmente basado en dos investigaciones diferentes, por lo que el código del STT-MDF también lo está. Por un lado, se encuentra la parte del código que lleva a cabo el procesamiento de los datos y que los alimenta a las capas de *embedding* del STT-MDF. Dicha parte está basada en la primera de las dos investigaciones, es decir, en [Prado-Rujas *et al.*, 2023] cuyo código es un código abierto que se puede encontrar en el siguiente **repositorio** de GitHub. Por otro lado, se encuentra la parte más importante del proyecto, el modelo. Este modelo está principalmente basado en el modelo desarrollado en investigación de [Grigsby *et al.*, 2021] cuyo código también es un código abierto de GitHub y se puede encontrar en el siguiente **repositorio**.

El código desarrollado y modificado para este trabajo fin de máster está disponible en el siguiente **repositorio**, el cual básicamente es una rama nueva del código de [Grigsby *et al.*, 2021]. Respecto al lenguaje de programación utilizado, el modelo está desarrollado en Python [Python S.F., 2023], exactamente utilizando las bibliotecas Pytorch [Paszke *et al.*, 2019] y Pytorch Lightning. PyTorch es un marco de *Deep Learning* de código abierto desarrollado principalmente por el laboratorio de investigación de IA de Facebook (FAIR). Proporciona una forma flexible y eficiente de construir, entrenar y desplegar redes neuronales profundas. PyTorch se utiliza ampliamente para diversas tareas de *Machine Learning*, como la visión por ordenador, el procesamiento

del lenguaje natural y el aprendizaje por refuerzo. En el caso de este trabajo, Pytorch ha sido utilizado para desarrollar el modelo. Pytorch Lightning, es una envoltura ligera de PyTorch que simplifica el entrenamiento y la organización de modelos complejos de *Deep Learning*. Su objetivo es reducir el código repetitivo y estandarizar las mejores prácticas para el entrenamiento de modelos, lo que facilita la creación de proyectos de *Deep Learning* escalables y reproducibles. Y en el caso de este trabajo, su uso ha sido específicamente para el proceso de capacitación del modelo y ejecución del mismo. Como se ha mencionado anteriormente, el código de [Prado-Rujas *et al.*, 2023] se ha utilizado como base de la parte de procesamiento de datos, sin embargo, este código estaba en desarrollado en la biblioteca de Keras por lo que se han debido realizado cambios bastante notables de manera que el la estructura del código origen es casi inapreciable. Keras es un marco de *Deep Learning* de código abierto escrito en Python. Proporciona una API de alto nivel que facilita la construcción y el entrenamiento de redes neuronales profundas. Keras se desarrolló centrándose en la facilidad de uso, la modularidad y la extensibilidad, por lo que es accesible tanto para principiantes como para investigadores experimentados.

3.3.2.1. Procesado de datos

En los modelos *Transformers*, al igual que en cualquier tipo de modelo de cualquier ámbito de la Inteligencia Artificial la separación de datos en conjuntos de entrenamiento (*train*), validación (*val'*) y prueba (*test*) es una práctica esencial para valorar y evaluar el rendimiento del modelo. Cada uno de estos conjuntos tiene su finalidad:

- Conjunto de entrenamiento o *Training set*: El *training set* obtiene la mayor parte de los datos y, como su propio nombre indice, se utiliza para entrenar. Es el conjunto de datos en el que el modelo aprende patrones, características y relaciones entre entradas y salidas. El modelo ajusta sus parámetros internos basándose en los ejemplos del conjunto de entrenamiento para optimizar su rendimiento.

- Conjunto de validación o *Validation set*: El *validation set* es la porción más pequeña de los datos que se utiliza para afinar (*fine-tuning*) el modelo durante el proceso de entrenamiento. Ayuda a controlar el rendimiento del modelo y a tomar decisiones sobre el ajuste de los hiperparámetros o la selección del modelo. El *validation set* no se utiliza para entrenar directamente el modelo, sino para evaluar su capacidad de generalización e identificar posibles problemas, como un ajuste excesivo o insuficiente.

- Conjunto de prueba *Test set*: El *test set* es una porción separada de los datos que se utiliza para evaluar el rendimiento final del modelo entrenado. Sirve como medida imparcial de la capacidad del modelo para generalizar a datos no vistos. El *test set* debe ser representativo de los escenarios reales a los que se enfrentará el modelo. Es crucial mantener el conjunto de pruebas aislado durante todo el proceso de entrenamiento y validación para evitar el conocido *overfitting* (cuando el modelo funciona muy bien con los datos de entrenamiento, pero no generaliza bien con datos nuevos que no se han visto).

Generalmente, a la hora de crear estos tres diferentes conjuntos de datos las divisiones más habituales son el 60 % para el entrenamiento, el 20 % para la validación y el 20 % para las pruebas. Sin embargo, en el caso de este proyecto, debido a la poca capacidad computacional para ejecutar el modelo, el número de instancias de estos

tres conjuntos sumadas no alcanzará el numero máximo de instancias que existen en los archivo proporcionados por el estudio [Prado-Rujas *et al.*, 2023], denominados **15m_map_90_60_taxi_norm_abs.h5** y **15m_map_90_60_bike_norm_abs.h5**, explicados en el apartado 3.2.2 y 3.2.1. El número de instancias que completarán cada uno de estos tres conjuntos se van a calcular teniendo en cuenta el número de intervalos que ocurren en un día (intervalos de 15 minutos, es decir, 4 en una hora y $4 \cdot 24$ en un día) y a partir de este dato se calcularán cuantos intervalos ocurren en unos días, una semana, un mes o incluso en un año. Al llevar a cabo este cálculo, se tendrá en cuenta que conjunto de datos debe tener un porcentaje mayor de instancias que otro.

Estos conjuntos de datos de entrenamiento, validación y prueba se separan en dos tipos de datos los datos contextuales *X* y los datos objetivo *Y*. Estos dos tipos de datos están separados en otras dos ramas (espaciales y temporales) ya que, el trabajo está basado en un problema espacio-temporal. En cuanto a los datos contextuales, por un lado, se encuentran los datos temporales que se van a almacenar en la variable (Y_{time}) los cuales han sido previamente explicados en el apartado 3.2.3. En dicho apartado, se menciona que la forma del conjunto de datos temporales es de $|T|$ *x 8*, sin embargo, esta forma se va a modificar ya que uno de los objetivos del modelo es recibir datos de entrada de n_x pasos temporales diferentes. Esta variable n_x es un hiperpárametro que se ajustará dependiendo del experimento. De esta manera, la forma del conjunto de datos temporales (Y_{time}) será de $|T|$ *x* n_x *x 8*. Por el otro lado de los datos contextuales, están los datos espaciales que se van a almacenar en la variable (Y_{time_series}), que al igual que el conjunto de datos temporales, han sido explicados previamente pero en este caso en los apartados 3.2.2 y 3.2.1. Repitiendo el patrón que se ha seguido con los datos temporales, la forma del conjunto de datos espaciales cambiará con respecto a la mencionada en dichos apartados ya que para entrar al modelo y poder compararlo con los datos temporales deben tener formatos parecidos. Por lo tanto, este conjunto pasara de $|T|$ *x 90 x 60* a $|T|n_x$ *x 90 x 60*, extrayendo de esta manera los datos espaciales correspondientes a los n_x pasos temporales. Cabe destacar que la forma de este segundo tipo de datos es de $90x60$ debido a que la *mesh-grid* original es de ese tamaño, sin embargo, para poder realizar ejecuciones completas en cada uno de los experimentos se limitará dicha *mesh-grid* a un tamaño algo más reducido.

Respecto al otro tipo de datos guardados, los datos objetivo, utilizarán la misma forma que los datos contextuales pero se almacenarán en las variables (Y_{time}) y (Y_{time_series}) respectivamente. Además, los pasos temporales de los que se extraerán los datos serán pasos más avanzados en el tiempo, como es lógico, y los determinará el valor del hiperpárametro n_y cuyo valor generalmente es equivalente al de n_x. Entre los pasos temporales de los datos contextuales y objetivo existe una variable llamada*shift* la cual hace referencia al número de intervalos que existen entre los dos tipos de datos mencionados. El objetivo de dicha variable es que los datos contextuales no se solapen ni los datos objetivo ni con las predicciones correspondientes. En la figura 3.9 se puede observar la imagen extraída de la investigación de [Prado-Rujas *et al.*, 2023] que hace referencia a lo recientemente explicado. En esta figura se realiza la explicación de los pasos temporales utilizados para los datos contextuales y objetivo o output ya que, el objetivo de las predicciones es que estos dos últimos tipos de datos ocurran en los mismos pasos temporales e incluso en el momento más parecido posible. En la figura *t* hace referencia al tiempo, n_x y n_y son los mencionados pasos temporales, *r* y *c* se refieren a las filas y columnas de la *mesh-grid*, *s* es el *shift* (ventana) entre

los pasos temporales y la f es la técnica/método/función utilizada en el modelo para llevar a cabo las predicciones. Es decir, $f(X) \approx Y$.

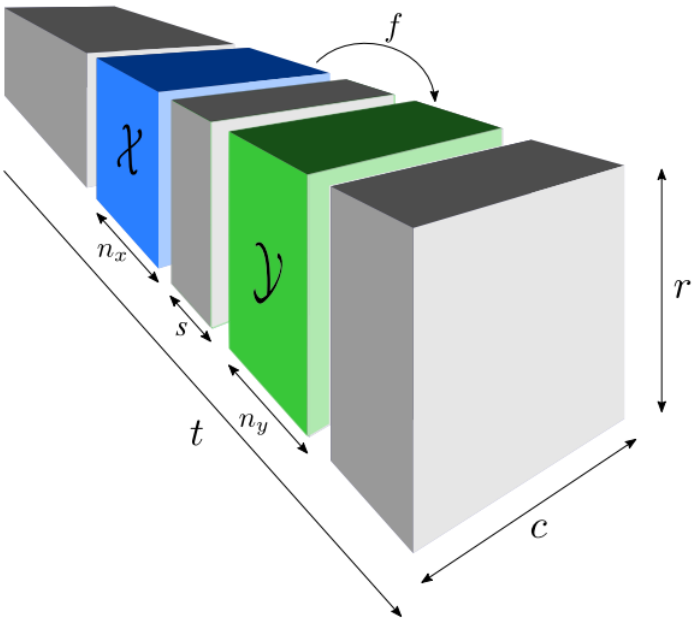


Figura 3.9: Representación del problema de predicciones de series temporales ([Prado-Rujas *et al.*, 2023])

Por último, para poder alimentar las primeras capas *embedding* del modelo con estos conjuntos de datos, estos se deben pasar a tensores Torch (PyTorch) y hasta este momento se había trabajado con estos conjuntos como *Numpy arrays*. Una vez convertidos los conjuntos de datos de entrenamiento, validación y prueba en tensores Torch, los cuales son muy útiles para este tipo de problemas espacio-temporales ya que ayudan a la aceleración de la GPU en el entrenamiento, estos se pasan por la clase *DataModule* proporcionada también por PyTorch. En este proyecto, dicha clase se ha utilizado como una forma de encapsular la carga de datos y los pasos de preprocesamiento necesarios para poder llevar a cabo las fases de entrenamiento, validación y prueba del STT-MDF. Además, proporciona una interfaz estandarizada para gestionar y organizar los datos, lo que facilita compartir y reproducir experimentos. Tras pasar por la clase *DataModule* y esta misma devolver los datos de entrenamiento, validación y prueba en el formato correspondiente, se da por finalizada la fase de procesamiento de datos, puesto que está todo listo para alimentar correctamente al modelo.

3.3.2.2. Tipos de atención en entrenamiento

A la hora de asignar una arquitectura de normalización asignada al modelo, existen las opciones: *Pre-Norm* [Xiong *et al.*, 2020], *BatchNorm* [Ioffe and Szeged, 2015], *PowerNorm* [Shen *et al.*, 2020] y *ScaleNorm* [Nguyen and Salazar, 2019]. Cada una de estas técnicas han sido extraídas de múltiples estudios relacionados con los *Transformers* [[Huang *et al.*, 2020]; [Liu *et al.*, 2020]; [Wu *et al.*, 2021]; [Zhou *et al.*, 2020]; [Araabi and Monz, 2020]; [Fan *et al.*, 2019]] y se han incorporado al STT-MDF con el objetivo de aumentar el rendimiento y la robustez de los hiperparámetros al tiempo que conservamos la simplicidad que destaca en los este tipo de modelos.

En primer lugar, *Pre-Norm* es la arquitectura utilizada a la hora de implementar las normalización de capas, basado en la mejora del entrenamiento y la convergencia del STT-MDF. Para entender mejor la técnica *Pre-Norm*, se deben mencionar tanto el concepto de conexiones residuales como el concepto de la normalización de capas. Las conexiones residuales, introducidas en la arquitectura *ResNet* [He *et al.*, 2015], permiten a la red aprender funciones residuales, lo que facilita la optimización de las redes profundas. La normalización de capas es una técnica que normaliza las entradas de una capa para estabilizar el proceso de entrenamiento. Al aplicar la técnica *Pre-Norm* en la red, cada capa de la misma se le aplica la normalización de capa antes de la función de activación (almacenada en la variable *activation*). De esta manera, se garantiza que la entrada a la función de activación esté normalizada, lo que puede ayudar a aliviar el problema del gradiente evanescente y mejorar la estabilidad general de la red.

En el modelo propuesto existe la posibilidad de aplicar una de las siguientes funciones de activación:

- **Rectified Linear Unit (RELU)**: Es una de las funciones de activación más sencillas y utilizadas. Es eficiente desde el punto de vista computacional y se ha demostrado que funciona bien en las tareas relacionadas al *Machine Learning* y todas sus ramas. Sin embargo, *ReLU* sufre el problema del *dying ReLU*, en el que las neuronas pueden quedarse atascadas en un estado de salida cero, lo que hace que no puedan recuperarse durante el entrenamiento.

$$\text{Para } x < 0 \rightarrow \text{ReLU}(x) = 0 \tag{3.10}$$

$$\text{Para } x > 0 \rightarrow \text{ReLU}(x) = x \tag{3.11}$$

- **Exponential Linear Unit (ELU)**: Es similar a la función *ReLU* pero tiene una curva más suave para los valores negativos, lo que puede ayudar a aliviar el problema del *dying ReLU*. El hiperparámetro *aplha* que se puede observar en las fórmula 3.12 y 3.13 determina la pendiente de ambas funciones para entradas negativas, controlando cuánto se aproxima la función al lado negativo.

$$\text{Para } x < 0 \rightarrow \text{ELU}(x) = \alpha(\exp(x) - 1) \tag{3.12}$$

$$\text{Para } x > 0 \rightarrow \text{ELU}(x) = x \tag{3.13}$$

- (ELU) (GELU): Es una aproximación suave de la función de activación *ReLU* y ha ganado popularidad por su eficacia en modelos de *Deep Learning*. Presenta características de saturación similares a las funciones sigmoidales, pero con una tasa de crecimiento más rápida. La función *GELU* puede ser útil para reducir el problema del gradiente evanescente:

$$GELU(x) = xP(X \leq x) = x\phi(x) = x \cdot \frac{1}{2}[1 + erf(x/\sqrt{2})] \tag{3.14}$$

Después, con el objetivo de reemplazar la técnica *LayerNorm*, se encuentra *BatchNorm* que proporciona grandes ventajas en el dominio de series temporales. A la hora de introducir *BatchNorm*, se argumentaba que era más popular en aplicaciones de visión artificial porque la varianza de entrenamiento reducida mejora el rendimiento

sobre las *LayerNorms* que son las predeterminadas en NLP. Sin embargo, mediante los experimentos realizados en [Grigsby *et al.*, 2021] y en este trabajo, se ha demostrado que es de gran utilidad para las tareas relacionadas con el problema de predicción de series temporales.

Por último, también se han añadido las *ScaleNorm* y *PowerNorm*. *ScaleNorm* es una técnica de preprocesamiento utilizada en el análisis de datos y el *Machine Learning* para estandarizar el rango o la escala de las características de entrada. El propósito de la normalización de escala es poner todas las características en una escala similar, asegurando que ninguna característica en particular domine el algoritmo de aprendizaje o análisis debido a su mayor magnitud. *PowerNorm* es una técnica de normalización que Consiste en calcular la potencia de la señal o del conjunto de datos y, a continuación, escalarla por un factor que la lleve al nivel de potencia deseado. Para ello se pueden utilizar varios métodos, como dividir por la raíz cuadrada de la potencia de la señal, escalarla por un factor específico o utilizar técnicas estadísticas como la normalización *z-score*.

El modelo STT-MDF constará de un hiperparámetro en el que se especificará cual de estas cuatro técnicas mencionadas se utilizará en cada experimento.

La elección de la implementación de la atención es flexible en toda la arquitectura, lo que permite avanzar en el subcampo de la atención de largo alcance para mejorar la escalabilidad futura. En la gran mayoría de los experimentos realizados, se ha utilizado la versión *ReLU* de *Performer* [Choromanski *et al.*, 2020] debido a su ahorro de memoria y compatibilidad al trabajar con los mecanismos de atención más protagonistas en la arquitectura del STT-MDF (*self-attention* y *cross-attention*). Además, también se han realizado experimentos utilizando el tipo de atención de *ProbSparse* [Wang *et al.*, 2021] como los experimentos realizados en el modelo propuesto en [Zhou *et al.*, 2020].

- **Performer**: Ofrece una aproximación eficiente a los mecanismos *self-attention* que aparecen a lo largo del modelo puesto que, estos mecanismos tradicionales tienen una complejidad cuadrática, lo que los hace computacionalmente caros al trabajar con entradas grandes. Entonces, este tipo de atención aborda dicho problema empleando un algoritmo de atención rápida basado en el "kernelized attention framework". Además, sustituye el *dot-product attention* por una técnica de aproximación que utiliza proyecciones aleatorias y la transformada rápida de Fourier (FFT). Gracias a estas técnicas, consigue una complejidad lineal tanto en el número de fichas como en la dimensión del espacio de características, lo que reduce significativamente el coste computacional de la autoatención. Se ha demostrado que el *Performer* mantiene un rendimiento comparable al de los mecanismos de *self-attention* estándar, al tiempo que es más eficiente.
- **ProbSparse**: ProbSparse, abreviatura de "Probabilistic Sparse Attention", es otro mecanismo de atención que pretende mejorar la eficiencia y escalabilidad de la autoatención. Introduce la dispersión en el mecanismo de atención mediante la imposición de patrones dispersos en los pesos de atención. Esto significa que sólo un subconjunto de fichas se atienden entre sí, lo que reduce la carga computacional.ProbSparse consigue la dispersión entrenando los pesos de atención para que sigan una distribución probabilística. Durante el entrenamiento, fomenta que la mayoría de los pesos de atención sean cercanos a cero, ignorando de forma efectiva las interacciones irrelevantes entre fichas. El resultado

es un patrón de atención más estructurado e interpretable. Se ha comprobado que ProbSparse ofrece ventajas computacionales, especialmente en escenarios en los que las secuencias de entrada son largas o los recursos computacionales son limitados.

3.3.2.3. Métricas de evaluación

A la hora de obtener los resultados más específicos y detallados posibles se han utilizado diferentes métricas de evaluación para poder valorar el rendimiento o la eficacia de diversos modelos, algoritmos, sistemas o procesos en una amplia gama de campos. Estas métricas proporcionan medidas cuantitativas que ayudan a comparar diferentes enfoques, optimizar parámetros, tomar decisiones informadas y evaluar la calidad general de un sistema o solución. En concreto, las métricas utilizadas en este trabajo han sido las siguientes:

- **Mean Squared Error (MSE)**: Es una de las métricas más utilizada en los modelos similares al propuesto en este trabajo. Ha sido sumamente util para calcular la media de las diferencias al cuadrado entre los valores de las predicciones obtenidas de la demanda de movilidad del transporte publico (taxis y bicicletas) y los valores objetivo o reales a los que se deberían asemejar lo maximo posible dichas predicciones. Para calcular el *MSE* primero se obtienen los valores de las predicciones realizadas y los valores objetivos o reales y se calcula la diferencia entre cada uno de estos valores elevando al cuadrado la resta entre sí. Se suman cada uno de estos cálculos realizados y se divide dicha suma por el número total de instancias en h pasos temporales, obteniendo de esta manera la *Mean Squared Error*. Este procedimiento se puede observar en la siguiente fórmula (3.15:)

$$MSE := \frac{1}{hN} \sum_{n=1}^{N} \sum_{t=T}^{T+h} (y_t^n - \hat{y}_t^n)^2 \tag{3.15}$$

  Esta métrica denominada *MSE*, pone un mayor énfasis en los errores más grandes debido a la operación de elevar al cuadrado, por lo que es más sensible a los valores atípicos.

- **Mean Absolute Error (MAE)**: Es otra métrica popular para los modelos de regresión. Mide la diferencia absoluta media entre los valores predichos y los valores reales u objetivo. En este caso, para calcular esta otra métrica, se debe llevar a cabo el siguiente procedimiento (representado en la fórmula 3.16): Obtener los valores de las predicciones y los valores objetivo o reales, después, calcular la diferencia absoluta entre cada uno de estos valores y sumar todas estas diferencias. Finalmente, se divide esta suma por le número total de instancias en h pasos temporales, al igual que ocurría en la métrica *MSE*, obteniendo la "Mean Absolute Error".

$$\text{MAE} := \frac{1}{hN} \sum_{n=1}^{N} \sum_{t=T}^{T+h} |y_t - \hat{y}_t| \tag{3.16}$$

  El *MAE* ofrece una interpretación más directa, ya que no se ve afectado por la operación de elevación al cuadrado.

- **Mean Absolute Percentage Error (MAPE)**: Es una métrica utilizada habitualmente en este tipo de tareas de predicción. Mide la diferencia porcentual media

entre los valores previstos y los reales. Los pasos para calcular el *MAPE* son los siguientes (representados en la fórmula 3.17): Obtener los valores de las predicciones y los valores objetivo o reales correspondientes, calcular la diferencia absoluta entre cada uno de estos valores, dividir cada diferencia absoluta por el valor objetivo o real correspondiente, sumar todos los porcentajes resultantes de dicha división y, finalmente, dividir la suma por el número total de instancias en h pasos temporales para obtener la "Mean Absolute Percentage".

$$\text{MAPE} := \frac{1}{hN} \sum_{i=0}^{N-1} |\frac{(y_t - \hat{y}_t)}{y_t}| \tag{3.17}$$

El *MAPE* proporciona una medida relativa del error, lo cual resulta útil a la hora de comparar la precisión entre distintos conjuntos de datos o variables. Sin embargo, tiene algunas limitaciones, como la sensibilidad a los valores cero y a los valores no limitados.

- **Symmetric Mean Absolute Percentage Error (SMAPE)**: Es una variación del *MAPE* que aborda el problema de los errores asimétricos. En el caso de está última métrica implementada en el modelo denominada *SMAPE*, se debe realizar un procedimiento similar al resto de las métricas el cual se verá reflejado en la fórmula 3.18. Dicho procedimiento comienza con la obtención de los valores de las predicciones realizadas y los valores reales u objetivo correspondientes, después, se calcula la diferencia absoluta entre cada uno de estos dos tipos de valores. Una vez calculada la diferencia, se divide por la mitad de la suma de los valores absolutos de las predicciones y los valores absolutos reales u objetivo. Se suman todos los valores resultantes de esta operación realizada hasta el momento y se divida esa suma total por el número de instancias en h pasos temporales, obteniendo de esta manera el "Symmetric Mean Absolute Percentage Error".

$$\text{SMAPE} := \frac{1}{hN} \sum_{n=1}^{N} \frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|)/2} \tag{3.18}$$

SMAPE proporciona una medida equilibrada que es simétrica y evita los problemas de división por cero. Se ha decido utilizarla en este caso, porque es muy común entre los problemas de predicción de la demanda de series temporales.

En cada una de estas formulas, y_t^n y $\hat{y}_t^n$ corresponden a los valores reales u objetivo y a los valores obtenidos de las predicciones realizadas de la variable número n en el paso temporal t, respectivamente. El detalle principal a tener en cuenta en las fórmulas de las siguientes métricas es que se está calculando el promedio a lo largo de la secuencia de predicción de h pasos temporales.

Como se ha mencionado en el apartado del STT-MDF 3.3, los datos de salida se basan en una secuencia de predicciones más o menos parecidas a la secuencia objetivo. Esta secuencia de salida se restaura de manera que obtiene el mismo formato que la secuencia de entrada del decodificador, es decir, la secuencia objetivo. Esto es lo que hace posible calcular las diferencias mencionadas en cada una de las métricas y por ello se pueden utilizar en este modelo. En la mayoría de los modelos cuyo objetivo es la predicción de series temporales se utilizan principalmente las métricas "Mean Absolute Error (MAE)" y "Mean Squared Error (MSE)". Sin embargo, en el caso del *STT-MDF* se han incluido también, como se ha podido ver en la reciente descripción

de las mismas, las métricas "Mean Absolute Percentage Error" (MAPE) y "Symmetric Mean Absolute Percentage Error (SMAPE)".

3.3.2.4. Hiperparámetros

A la hora de realizar los experimentos correspondientes, existen una gran cantidad de posibilidades y esto se debe a los múltiples hiperparámetros que forman el modelo. Gracias a estos hiperparámetros, se puede incluir una configuración totalmente detallada especificando en variables o detalles más pequeños que no han sido explicados a lo largo de la memoria, pero que se van a explicar a continuación:

- **Batch size**: Almacenado en la variable *batch_size*, se refiere al número de muestras de datos que se introducen en el modelo durante cada iteración de la fase de entrenamiento. Determina cuántas instancias se procesan simultáneamente antes de actualizar los parámetros del modelo.
- **Learning rate** inicial: Almacenado en la variable *init_lr*, representa la tasa de aprendizaje inicial al comienzo de la fase de entrenamiento. Es el punto de partida de la programación de la tasa de aprendizaje y determina la magnitud de las actualizaciones de los parámetros durante las iteraciones iniciales.
- **Learning rate** base: Almacenado en la variable *base_lr*, se refiere a la tasa de aprendizaje base o por defecto utilizada durante el proceso de entrenamiento. Es la tasa de aprendizaje primaria que se aplica para actualizar los parámetros del modelo durante la optimización.
- **Warmup steps**: Almacenado en la variable *warmup_steps*, indican el número de pasos iniciales o iteraciones durante los cuales el *learning rate* aumenta gradualmente desde un valor muy pequeño hasta su valor base. Ayuda a estabilizar el proceso de aprendizaje al principio del entrenamiento.
- **Decay factor**: Almacenado en la variable *decay_factor*, determina la velocidad a la que decae o se reduce el *learning rate* durante el entrenamiento. Normalmente, el *learning rate* se multiplica por el *decay factor* para disminuir su valor. Al reducir gradualmente el *learning rate*, el modelo puede refinar sus actualizaciones de parámetros y converger hacia una solución óptima. Estos últimos cuatro hiperparámetros mencionados están relacionados de manera que el *init_lr* se ira descomponiendo en cada *decay_step* ($decay\ step = init\_lr \cdot decay\_factor$) a lo largo de todos los *warmup_steps* hasta alcanzar el valor base asignado a *base_lr*.
- **Dimensiones:** Aparecen diferentes tipos de dimensiones a lo largo del modelo almacenados en las variables *d_model*, *d_queries_keys*, *d_values* y *d_ff* que definen la dimensión de los *embeddings* de entrada y salida, de los vectores *Query*, *Key* y *Value* en los mecanismos de atención y de las capas *FFNN* respectivamente. Establecer dimensiones a las entradas y salidas de las diferentes capas del modelo determina la capacidad y expresividad del propio modelo y es un factor crucial en su rendimiento global.
- **Capas codificador:** Almacenado en la variable *e_layers*, se refiere al número de capas codificadoras del modelo. Las capas codificadoras procesan los datos de entrada y extraen las características relevantes. Aumentar el número de capas codificadoras puede aumentar la capacidad del modelo para captar patrones intrincados y mejorar su poder de representación.

- **Capas decodificador:** Almacenado en la variable *d_layers*, se refiere al número de capas codificadoras del modelo. Las capas codificadoras procesan los datos de entrada y extraen las características relevantes. Aumentar el número de capas codificadoras puede aumentar la capacidad del modelo para captar patrones intrincados y mejorar su poder de representación.

- **Dropout**: Existen diferentes *dropouts* para las diferentes capas del modelo que están almacenados en las variables *dropout_emb*, *dropout_attn_out*, *dropout_attn_matrix*, *dropout_qkv*y *dropout_ff*, se refiere a la tasa de *dropout* aplicada a las capas *embeddings*, a las matrices y a los vectores de salida de los mecanismos de atención, a los vectores *Query*, *Key* y *Value* de estos mismos mecanismos de atención y a la salida de las capas *FFNN* respectivamente. La técnica *dropout* es una técnica de regularización utilizada en modelos como el propuesto para evitar el *overfitting*. Pone a cero aleatoriamente una fracción de las unidades de entrada durante la fase de entrenamiento.

- **Coeficiente *L2*:** Almacenado en la variable *l2_coeff*, controla la fuerza de la regularización L2 aplicada a los parámetros del modelo durante la fase de entrenamiento. La regularización L2 es una técnica muy utilizada para evitar el famoso *overfitting* en este tipo de modelos. Asignando un valor más alto al hiperpárametro "l2_coeff.[el] efecto de regularización es más fuerte, ya que amplifica la penalización en valores de parámetros más grandes. Por el contrario, un valor más bajo reduce el efecto de regularización.

- **Número de *attention heads*:** Almacenado en la variable *n_heads*, indica el número de cabezas de atención del modelo transformador. Las cabezas de atención operan en paralelo y capturan diferentes aspectos o patrones en los datos de entrada. Un mayor número de cabezas de atención permite al modelo capturar relaciones y dependencias más complejas

- **Longitud máxima de la secuencia:** Almacenado en la variable *max_seq_len*, se refiere a la longitud máxima de secuencia permitida o considerada en el STT-MDF. Este hiperparámetro nace de la suma de otros dos hiperparámetros denominados *context_points* y *target_points* que se encargan de almacenar el valor de la longitud de la secuencia de entrada contexto y la secuencia de entrada objetivo respectivamente.

- **GPUs**: Almacenado en la variable *devices*, se refiere al número de unidades de procesamiento gráfico (*GPU*) utilizadas en la fase de entrenamiento del modelo. Las *GPU* son dispositivos de hardware especializados que destacan en la realización de cálculos paralelos, lo que las hace muy eficaces para entrenar y acelerar modelos de este tipo. En este caso, se ajusta el valor de esta variable a la hora de crear el *trainer* de la librería *Pytorch Lightning* que se encarga principalmente de entrenar y realizar pruebas del modelo.

  Como se ha mencionado, existen una gran cantidad de posibilidades dentro de cada uno de los hiperparámetros pero también hay que destacar que existen bastantes más hiperpárametros sin embargo, en este apartado tan solo se han explicado los que no han sido detalladamente descritos o mencionados a lo largo de la memoria y entre los que no se habían descrito o mencionados, los que más influencia pueden llegar a tener al cambiar sus valores en cada uno de los experimentos.

3.4. Experimentos

En este apartado se va a explicar de manera detallada la composición de cada uno de los los experimentos realizados a lo largo del desarrollo del modelo. Esta composición está formada principalmente por la configuración de los hiperparámetros, el tratamiento de datos y el equipo utilizado.

3.4.1. Equipo y capacidad computacional

Respecto al equipo, a lo largo del desarrollo del modelo ha sido una gran limitación puesto que, los conjuntos de datos con los que se ha trabajado son inmensamente grandes y la unidad de procesamiento utilizada durante gran parte del tiempo, que ha conllevado la fase de ejecución del proyecto, ha sido una *GPU* personal cuyas especificaciones son detalladas más adelante. El motivo por el que los conjuntos de datos escogidos para este trabajo son tan grandes es porque los modelos *Transformers* proporcionan mejores resultados cuando son alimentados con grandes cantidades de datos. Debido a esta falta de capacidad computacional, se ha intentado encontrar diferentes alternativas a la unidad de procesamiento personal como las proporcionadas por Google Colab o magerit en colaboración con cesvima.

Características de las tres unidades de procesamiento utilizadas:

1. **GPU** personal: Los resultados obtenidos por esta unidad de procesamiento han sido notablemente peores debido a la limitación computacional mencionada anteriormente. Se llama *GeForce RTX 2060* y cuenta con 1920 núcleos CUDA, 240 *Tensor Cores* que pueden ofrecer 52 *TFLOPS* de potencia de *deep learning*, 30 núcleos RT que pueden transmitir 5 gigarrayos por segundo, 6 GB de la memoria GDDR6 más reciente que se ejecuta a 14 Gbps y una frecuencia acelerada de GPU de 1,68 GHz.

2. **"GPU" de Google Colab**: Conocida como *NVIDIA Tesla T4* es un componente totalmente adecuado para ejecuciones que relacionadas con los modelos de inteligencia artificial y tiene 16 GB de memoria compartida en función del modo asignado. Esta memoria se comparte entre todos los usuarios de la misma máquina física, por lo que la memoria disponible puede variar en función del uso en cada momento.

3. **GPU** de magerit: Esta unidad de procesamiento se denomina *NVIDIA A100* y tiene una capacidad mayor que las dos primeras unidades mencionadas. Las características de esta *GPU* son: 6912 núcleos CUDA, 512 *Tensor Cores*, 40 GB de memoria dedicada a una velocidad de 1.6 TB/s y está acelerada por una arquitectura Ampere.

3.4.2. Configuración

La configuración de los experimentos realizados se basa mayormente en la inicialización de los hiperparámetros. Aunque, existen otros parámetros que tienen un gran impacto en cuanto al rendimiento de los experimentos como el tamaño de la *meshgrid* o el de los conjuntos de datos de entrenamiento, valoración y test. A continuación se van a mostrar los valores asignados a los hiperparámetros y parámetros a la hora de llevar a cabo los experimentos más exitosos:

- Tamaño del mapa o *mesh-grd*: El motivo principal de disminuir el tamaño inicial de la *mesh-grid* de Chicago ha sido la limitación computacional ya que, se ha priorizado invertir gran parte de la capacidad computacional de las *GPUs* NVIDIA A100 en los conjuntos de datos de entrenamiento, validación y test. Por lo tanto, esto ha hecho que el tamaño de mañana utilizado en los experimentos llevados a cabo sea de 40 x 40, mientras que el original es de 90 x 60 como se puede observar en el apartado | 3.2
- Los conjuntos de datos de entrenamiento, validación y test, se han tenido que limitar considerablemente puesto que, los conjuntos de datos originales de taxis y bicicletas constan de una información diaria de 8 años diferentes lo cual es una cantidad de datos enorme. Como bien se explica en 3.2.1, cada día consta de 96 instancias ($inst\_per\_day = 24 * 4$), medida que se va a utilizar para calcular el numero de instancias que tendrán los conjuntos de datos con los que se alimentará al modelo.

  1. Entrenamiento (0, 70176): Desde el primer día guardado en el dataset general (0) hasta 0 hasta el número de instancias correspondientes a 2 años, teniendo en cuenta que uno de ellos es bisiesto (70176). La formula para obtener el límite de instancias es la siguiente:

     $$(2 * 365 + 1) * ints\_in\_day \tag{3.19}$$

     La formula es la siguiente

  2. Validación (70176, 74976): Desde la última instancia del conjunto de datos de entrenamiento (70176), hasta el sumatorio del número de instancias de entrenamiento y las instancias correspondientes a 50 días (74976). En total son 4800 instancias de validación. La formula para obtener el límite de instancias es la siguiente:

     $$train\_lim + 50 * ints\_in\_day \tag{3.20}$$

  3. Test (74976, 84576): Desde la última instancia del conjunto de datos de validación (74976) hasta el sumatorio del número de instancias de validación y las instancias correspondientes a 100 días (84576). En total son 9600 instancias de test. La formula para obtener el límite de instancias es la siguiente:

     $$val\_lim + 100 * ints\_in\_day \tag{3.21}$$

- Respecto a los hiperparámetros, se han asignado los siguientes valores:

3.4.3. Resultados

En este apartado, se explicarán los resultados obtenidos en los experimentos realizados con la unidad de procesamiento proporcionada por magerit utilizando tablas e imágenes. Estos experimentos se realizarán sobre los conjuntos de datos de bicicletas 3.2.2 y taxis 3.2.1. El objetivo principal de llevar a cabo diferentes ejecuciones es comparar los resultados obtenidos en estas dependiendo de la cantidad de horizontes contextuales *n_x* y de predicción *n_y* asignados y del espacio temporal entre estos dos tipos de horizontes *shift*. Visualmente estos valores afectarán al aspecto de las tablas

Hiperparámetros en Taxis			
Batch_size	8	epoch	50
Enc layers	1	Dec layers	1
d_model	10	d_queries_keys	2
d_values	2	n_heads	2
d_ff	10	Activation	gelu
init_lr	0.0001	base_lr	0.001
l2_coeff	0.001	decay_factor	0.8
dropout_attn_matrix	0.01	dropout_attn_out	0.1
dropout_emb	0.1	dropout_ff	0.1
dropout_qkv	0.1	embed_method	spatio-temporal
loss	mae	local_cross_attn	performer
local_self_attn	performer	max_seq_len	8

Tabla 3.1: Valores establecidos a los hiperparámetros más relevantes en los experimentos realizados sobre los conjuntos de datos de taxis

ya que, las tablas tendrán el número de filas correspondientes al valor asignado a dichas variables.

Con el objetivo de contextualizar y facilitar la comprensión de los resultados se van a facilitar dos tablas diferentes en cada uno de los experimentos junto a una detallada descripción y valoración de estos. En la primera de las dos tablas aparecerán los mapas de calor con la información correspondiente a los pasos temporales de contexto. Después, en la segunda tabla se mostrarán los mapas de calor correspondientes a los datos objetivo y a las predicciones. Estos mapas objetivo se sitúan justo encima de las predicciones con el objetivo de que la comparación sea lo más sencilla y visual posible. Cada una de las columnas de ambas tablas corresponde a un paso temporal diferente (se detallan los minutos que han pasado hasta ese momento en concreto junto al paso temporal h_i siendo $i \in \{0, ..., 7\}$.).

Respecto a los mapas de calor, reflejarán con un color más llamativo las zonas donde ocurre o se prevé que ocurra algún tipo de actividad, mientras que las zonas con un color más apagado harán referencia a los lugares donde no ocurre o se prevé que no ocurra ninguna actividad o casi ninguna actividad. El tamaño asignado a cada uno de estos mapas se corresponderá a un fragmento de la *mesh-grid* original del mapa de Chicago debido a que, aplicar el tamaño completo del mapa en cada uno de los experimentos no ha sido posible por las limitaciones computacionales.

Por último, se van a comparar los valores obtenidos de las métricas de error *MAE*, *MSE*, *MAPE*, *SMAPE* correspondientes a la fase test de cada uno de los experimentos llevados a cabo.

3.4.3.1. Resultados taxis

Los experimentos realizados utilizando el conjunto de datos que contiene la información espacio-temporal de la demanda de taxis en Chicago durante diferentes años (**15m_map_90_60_taxi_norm_abs.h5**) han tenido los siguientes resultados:

- Los resultados del primer experimento se muestran en la tabla 3.3, mientras que los datos contextuales aparecen en la tabla 3.2. Ambas tablas constan de

diferentes mapas de calor correspondientes a cada uno de los pasos temporales contextuales y de predicción. A la hora comparar las predicciones obtenidas en este primer experimento con los mapas de calor de los datos objetivo, rápidamente se detectan diferentes similitudes. Entre estas similitudes se encuentra el hecho de que el modelo ha sido capaz de prever donde no ocurre o casi no ocurre actividad y el hecho de que el modelo ha previsto correctamente las zonas donde se centra la actividad. Respecto a los aspectos negativos de los resultados, el más notable es que el modelo pese a detectar las zonas donde ocurre actividad no ha sido capaz de acertar en la intensidad de dicha actividad. Es decir, en las predicciones obtenidas la diagonal que aparece por la esquina inferior izquierda de os mapas es notablemente más discreta que en los mapas objetivo. La conclusión de este primer experimento es que el modelo es capaz de predecir casi certeramente lo que ocurre en un tiempo cercano.

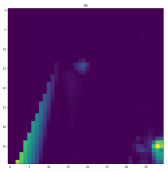
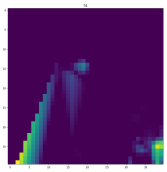
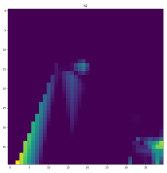
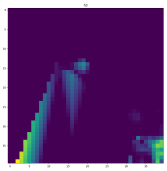
Datos contextuales del primer experimento de taxis			
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)
			

Tabla 3.2: Mapas de calor contextuales del primer experimento con el conjunto de datos de taxis donde x_n=4, x_y=4 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

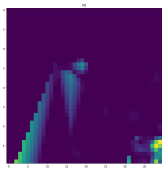
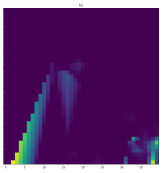
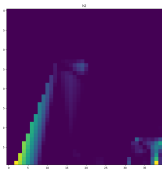
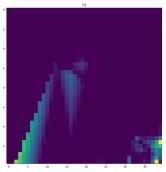
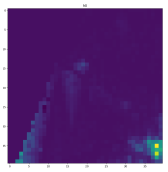
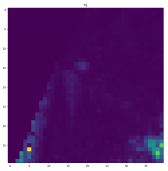
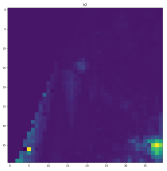
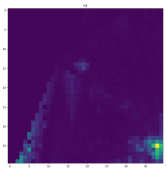
Datos objetivo del primer experimento de taxis			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			
Predicciones del primer experimento de taxis			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			

Tabla 3.3: Resultados del primer experimento con el conjunto de datos de taxis donde x_n=4, x_y=4 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(\dot{i} + s))$ siendo $i \in \{0, ..., n_y\}$.

- Los resultados del segundo experimento se muestran en la tabla 3.5, mientras que los datos contextuales aparecen en la tabla 3.4. Ambas tablas constan de

diferentes mapas de calor correspondientes a cada uno de los pasos temporales contextuales y de predicción. La particularidad de este experimento se encuentra en que los pasos temporales a predecir son 8. Respecto a los mapas de calor de las predicciones, son muy similares a los mapas objetivo. Sin embargo, a estos resultados se les suma un nuevo aspecto negativo basado en que el mapa esta ligeramente cubierto por un color azulado. Esto significa que el modelo no ha sido capaz de eliminar un pequeño porcentaje de actividad a lo largo del mapa por lo que el mapa esta cubierto por una ligera "niebla" azul. Esto no hace que las predicciones sean totalmente malas pero, si peores a las del primer experimento. En conclusión, los resultados de este experimento es son peores por el hecho de que el modelo esta obteniendo la misma cantidad de contexto que en el primer experimento pero, esta teniendo que predecir lo que ocurre en un tiempo más lejano.

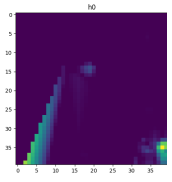
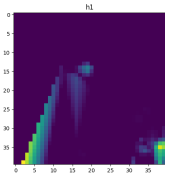
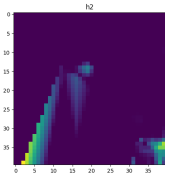
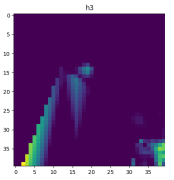
Datos contextuales del segundo experimento de taxis			
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)
			

Tabla 3.4: Mapas de calor contextuales del segundo experimento con el conjunto de datos de taxis donde x_n=4, x_y=8 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

- El tercer experimento se detalla en las tablas 3.6 y 3.7 de la misma forma que el resto. El detalle a destacar de esta ejecución es que se ha alimentado al modelo con 8 pasos temporales contextuales, buscando predecir la actividad de 4 futuros pasos temporales. En este caso, las predicciones obtenidas son prácticamente iguales a las del primer experimento, es decir, muy buenas. Una conclusión coherente podría ser que el modelo está siendo alimentado con bastante contexto y no tiene que predecir lo que ocurre en un tiempo muy lejano lo cual puede ser la causa de los malos resultados en el anterior experimento.
- El cuarto experimento se detalla en la tablas 3.8 y 3.9 de la misma forma que el resto. La particularidad de este experimento se encuentra en que se va a alimentar al modelo con información de 8 pasos temporales y se va a predecir la actividad de otros 8 futuros pasos temporales. En este caso, las predicciones obtenidas son peores en comparación con los primeros tres experimentos. Esta valoración negativa se debe a que la "niebla" mencionada en el segundo experimento aparece en casi todos los pasos temporales con una intensidad mayor. Además, el hecho de que en algunos de los pasos temporales no aparezca dicha "niebla" indica que el modelo está siendo irregular a la hora de mostrar los resultados. Respecto a los aspectos positivos, el modelo continua acertando el lugar en el que ocurre el foco principal de actividad. lo que indica una irregularidad a la hora de proporcionar los resultados. En conclusión, el modelo vuelve a tener peores resultados (al igual que en el apartado 2) cuando al experimento se le asignan 8 pasos temporales de predicción.

- El quinto experimento se detalla en las tablas 3.10 3.11 de la misma forma que el resto. En comparación al experimento anterior, consta de la misma cantidad de pasos temporales contextuales y de predicción. Sin embargo, la diferencia se encuentra en los pasos temporales que se encuentran entre los contextuales y los de predicción (variable *shift*) ya que, a diferencia del resto de experimentos en los que estaban asignados 4 pasos temporales, en este caso son 8. Esto hace que el tiempo entre el contexto y la predicción sea de 2 horas en lugar de tan solo 1. Las predicciones obtenidas son similares a las del cuarto experimento manteniendo el aspecto negativo de la "niebla" con la particularidad de la intensidad que prevé el modelo en los focos de actividad es más parecida a la mostrada en los mapas objetivo. En conclusión, los resultados siguen sin estar a la altura de los experimentos con 4 pasos temporales de predicción sin embargo, este experimento mejora el anterior pese a que el tiempo entre el último mapa contextual y el primero es mayor.

3.4.3.2. Resultados bicicletas

Los experimentos realizados utilizando el conjunto de datos que contiene la información espacio-temporal de la demanda de bicicletas en Chicago durante diferentes años (**15m_map_90_60_bike_norm_abs.h5**) han tenido los siguientes resultados:

- Los resultados del primer experimento se muestran en la tabla 3.13 mientras que los datos contextuales aparecen en la tabla 3.12 a través de diferentes gráficos e imágenes. Respecto a las predicciones, son bastante similares a los objetivo puesto que, el modelo esta siendo capaz de detectar las zonas principales de actividad y, además, también prevé que en gran parte del mapa no ocurre ningun tipo de actividad. Sin embargo, un aspecto negativo es la poca variación de las predicciones a lo largo de los pasos temporales. Esto es algo negativo por el hecho de que a lo largo de los pasos temporales de los mapas objetivo existe una variación bastante notable. La conclusión de este experimento es que el modelo es capaz de prever donde ocurre actividad pero, no diferenciar en que paso temporal ocurre.
- El segundo experimento se detalla en las tablas 3.14 3.15 de la misma forma que el primero, con la diferencia de los pasos temporales de predicción asignados a este segundo experimento son 8. Las predicciones obtenidas son totalmente similares a las del primer experimento, con los mismos aspectos negativos y positivos. Algo a destacar es que el modelo ha sido capaz de eliminar la característica "niebla" que aparecía en los resultados de varios experimentos de taxis. En conclusión, el modelo sigue siendo bastante certero pese a tener que predecir lo que ocurre en un tiempo más lejano que en el primer experimento.
- El tercer experimento se detalla en las tablas 3.16 3.17 de la misma forma que el resto. Las predicciones obtenidas nos exactamente similares a las de los primeros dos experimentos pero, sus aspectos negativos y positivos coinciden. Lo que las diferencia de estos dos experimentos es que la zona en la que el modelo detecta el foco de actividad es más grande por lo que, no es tan certero. En conclusión, el modelo está siendo capaz de detectar la zona donde ocurre la actividad pero no el lugar exacto. Por lo tanto, el resultado es un poco más genérico.
- Este cuarto experimento se detalla en las tablas 3.18 3.19 de la misma forma

que el resto. La particularidad de este experimento se encuentra en que se va a alimentar al modelo con información de 8 pasos temporales y se va a predecir la actividad de otros 8 futuros pasos temporales. Respecto a las predicciones, ocurre exactamente lo mismo que en el experimento anterior pero, con algo más de exactitud. La conclusión de este cuarto experimento es que al tener más contexto el modelo ha podido mejorar la precisión del resultado. Sin embargo, se mantiene el aspecto negativo de que el modelo no es capaz de diferenciar correctamente entre los pasos temporales. Esto se ve reflejado, por ejemplo, en la forma diagonal que aparece en la esquina inferior derecha que tan solo aparece en 3 pasos temporales objetivo mientras que, en los pasos temporales de predicción aparece en todos.

- El quinto experimento se detalla en las tablas 3.20 3.21 de la misma forma que el resto. En comparación al experimento anterior, consta de la misma cantidad de pasos temporales contextuales y de predicción. Sin embargo, la diferencia se encuentra en los pasos temporales que se encuentran entre los contextuales y los de predicción (variable *shift*) ya que, a diferencia del resto de experimentos en los que estaban asignados 4 pasos temporales, en este caso son 8. Esto hace que el tiempo entre el contexto y la predicción sea de 2 horas en lugar de tan solo 1. En cuanto a los mapas de calor de predicción, vuelven a tener los mismos aspectos negativos y positivos de los experimentos anteriores. En conclusión, pese a que el modelo no detecta de forma certera los diferentes focos de actividad, sigue siendo capaz de prever la nula actividad en el resto del mapa.

3.4.3.3. Métricas de error bicicletas y bicicletas

Este apartado tiene como objetivo comparar los resultados proporcionados por las métricas de error *MAE, MSE, MAPE, SMAPE* de la fase test de cada una de las ejecuciones realizadas tanto con los conjuntos de datos de taxis como con los conjuntos de datos de bicicletas.

Estos resultados se comparan en una misma tabla (3.22) y se obtienen las siguientes conclusiones:

- En el primer experimento comparando tanto los resultados de las métricas como las predicciones mostradas en mapas de calor se puede observar que los experimentos de los taxis son minimamente mejores.
- Los valores resultantes de las métricas del segundo, cuarto y quinto experimentos muestran una clara ventaja de los experimentos de taxis. Esto también se observaba en las predicciones mostradas.
- En el tercer experimento los valores resultantes de los experimentos realizados con ambos conjuntos de datos son muy similares.

En general, se podría decir que los resultados obtenidos en los experimentos realizados utilizando los conjuntos de datos de bicicletas son mejores.

3.4.4. Comparación de resultados

Este apartado se van a comparar los resultados obtenidos por el modelo propuesto con los modelos mencionados en [Prado-Rujas *et al.*, 2023]. Concretamente, se compararán los valores resultantes de la métrica *RMSE* correspondientes a cada uno de los horizontes de predicción de cada una de los experimentos. Cabe destacar que no es una comparación exacta por el hecho de que el tamaño de mapa asignado a las ejecuciones del modelo **STT-MDF** es diferente al de los demás modelos. Esto puede influir en que los experimentos a los que se ha asignado un tamaño de mapa más pequeño tengan mejor resultados puesto que, cuanto más pequeño el modelo tiene menos lugares donde poder fallar.

Los modelos escogidos para comparar sus resultados con el STT-MDF en este modelo son los siguientes:

- ST-MDF [Prado-Rujas *et al.*, 2023]: Es un modelo basado en un LSTM convolucional compuesto por los bloques de movilidad, tiempo y meteorología. Se concatenan los vectores de salida de cada uno de estos bloques y se pasan por diferentes *fully-connected layers* produciendo un vector unidimensional. Después, se convierte en un vector 4D proporcionando las cuatro dimensiones de salida correspondientes a la predicción: horizonte temporal, latitud, longitud y tipo de transporte (bicicleta o taxi).
- LSTM [Cui *et al.*, 2018]: Utiliza una estructura muy similar a la propuesta en la investigación de [Prado-Rujas *et al.*, 2023] basada en sus tres módulos, con la diferencia que en vez de estar compuesta de una capa convolucional, se compone por una capa LSTM de 16 neuronas y, además, sustituye la deconvolución por una capa densa adicional.
- BiLSTM: Tiene la misma estructura que el LSTM básico, pero en lugar de basarse en una capa LSTM que devuelve 6 neuronas, se basa en una capa LSTM bidireccional que devuelve 16 (8 neuronas en el *forward pass* y las otras 8 en el *backward pass*).
- Persistence: Está basado en la suposición de que la variable predicha no sufrirá ningun tipo de cambio en el futuro. Por lo tanto, se puede expresar como $f(\{M_{t_1}, ..., M_{t_{n_x}}\}) = M_{t_{n_x}}$.
- Naive: La predicción de este modelo es la media puntual de la ventana de entrada, la cual puede ser representada de la siguiente manera: $f(\{M_{t_1}, ..., M_{t_{n_x}}\}) = \frac{1}{n_x} \sum_{i=1}^{n_x} M_{t_i}$
- ST-GAT*: Es una mejora del modelo original ST-GAT basado en las *Graph Neural Networks*. Esta mejora está compuesta por tres bloques: ST-GAT (*input* taxis y bicicletas), LSTM (*input* metereologico), Linear (*input* series temporales). Estos tres bloques se agrupan mediante una feed-forward neural network (FFNN) de un tamaño de 128 neuronas.

Al comparar los valores resultantes del modelo propuesto que se muestran en las tablas 3.24 y 3.23 con los demás modelos, se ve claramente que dicho modelo propuesto no ha sido capaz de mejorar los resultados de los otros modelos. Aunque, cabe destacar, que en ejecuciones como la detallada en la fila 6 de la tabla 3.23 o la de la fila 18 de la tabla 3.24 los resultados son muy similares. Estos malos resultados se deben en gran parte por la limitación computacional ya que, a la hora de

realizar las ejecuciones correspondientes no se han podido asignar los valores idóneos a algunos hiperparametros. Por ejemplo, el modelo constaba tan solo de 1 capa codificador-decofidicador.

Tabla 3.5: Resultados del segundo experimento con el conjunto de datos de taxis donde x_n=4, x_y=8 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15\dot{(}i + s))$ siendo $i \in \{0, ..., n_y\}$.

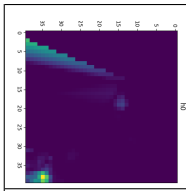
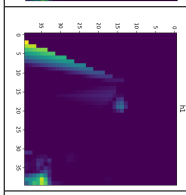
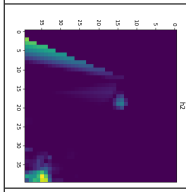
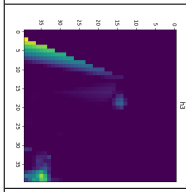
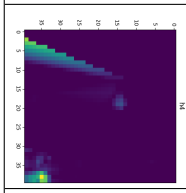
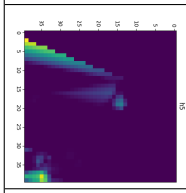
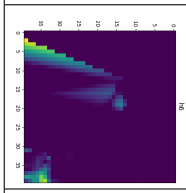
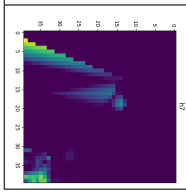
Datos contextuales del tercer experimento de taxis							
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)	h_4 (75 minutos)	h_5 (90 minutos)	h_6 (105 minutos)	h_7 (120 minutos)
							

Tabla 3.6: Mapas de calor contextuales del tercer experimento con el conjunto de datos de taxis donde x_n=8, x_y=4 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

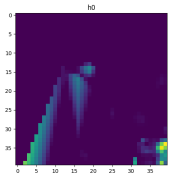
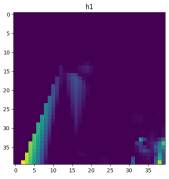
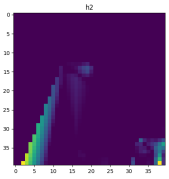
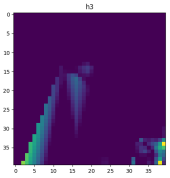
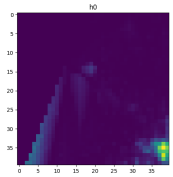
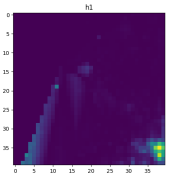
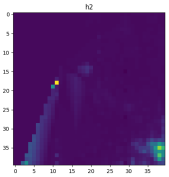
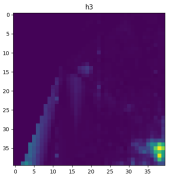
Datos objetivo del tercer experimento de taxis			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			
Predicciones del tercer experimento de taxis			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			

Tabla 3.7: Resultados del tercer experimento con el conjunto de datos de taxis donde x_n=8, x_y=4 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(i + s))$ siendo $i \in \{0, ..., n_y\}$.

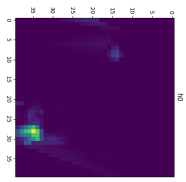
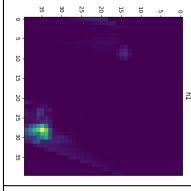
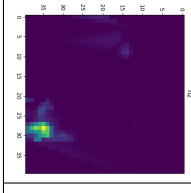
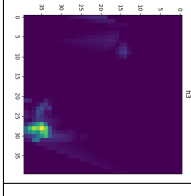
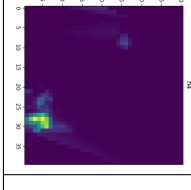
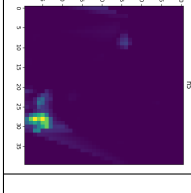
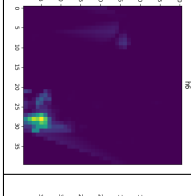
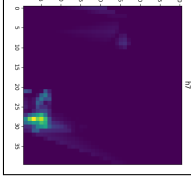
Datos contextuales del cuarto experimento de taxis							
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)	h_4 (75 minutos)	h_5 (90 minutos)	h_6 (105 minutos)	h_7 (120 minutos)
							

Tabla 3.8: Mapas de calor contextuales del cuarto experimento con el conjunto de datos de taxis donde x_n=8, x_y=8 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

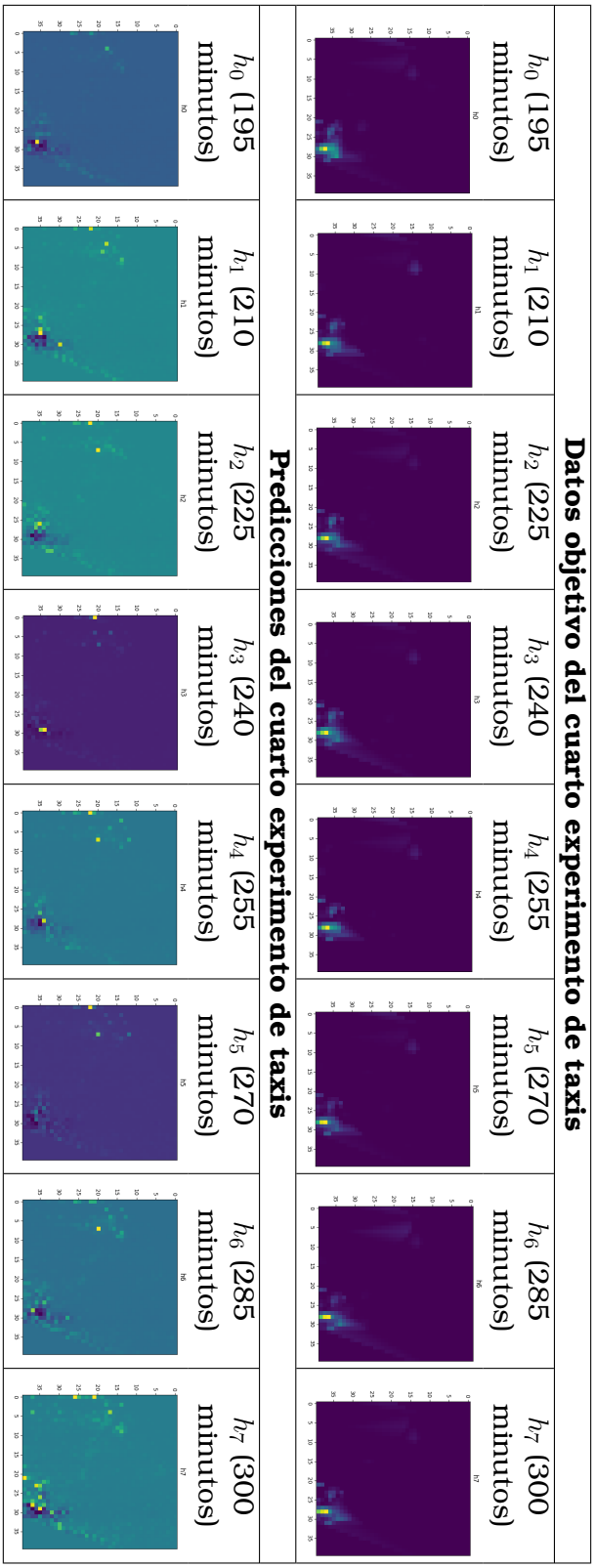


Tabla 3.9: Resultados del cuarto experimento con el conjunto de datos de taxis donde x_n=8, x_y=8 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(i + s))$ siendo $i \in \{0, ..., n_y\}$.

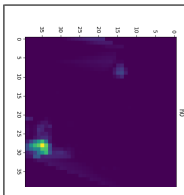
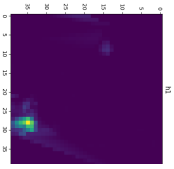
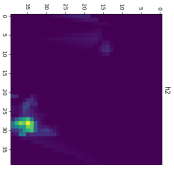
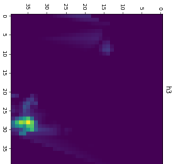
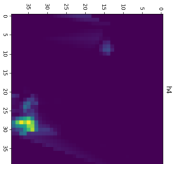
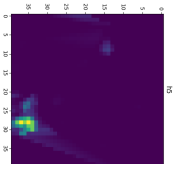
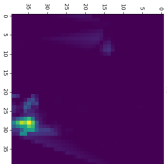
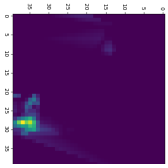
Datos contextuales del quinto experimento de taxis							
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)	h_4 (75 minutos)	h_5 (90 minutos)	h_6 (105 minutos)	h_7 (120 minutos)
							

Tabla 3.10: Mapas de calor contextuales del quinto experimento con el conjunto de datos de taxis donde x_n=8, x_y=8 y shift=8. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

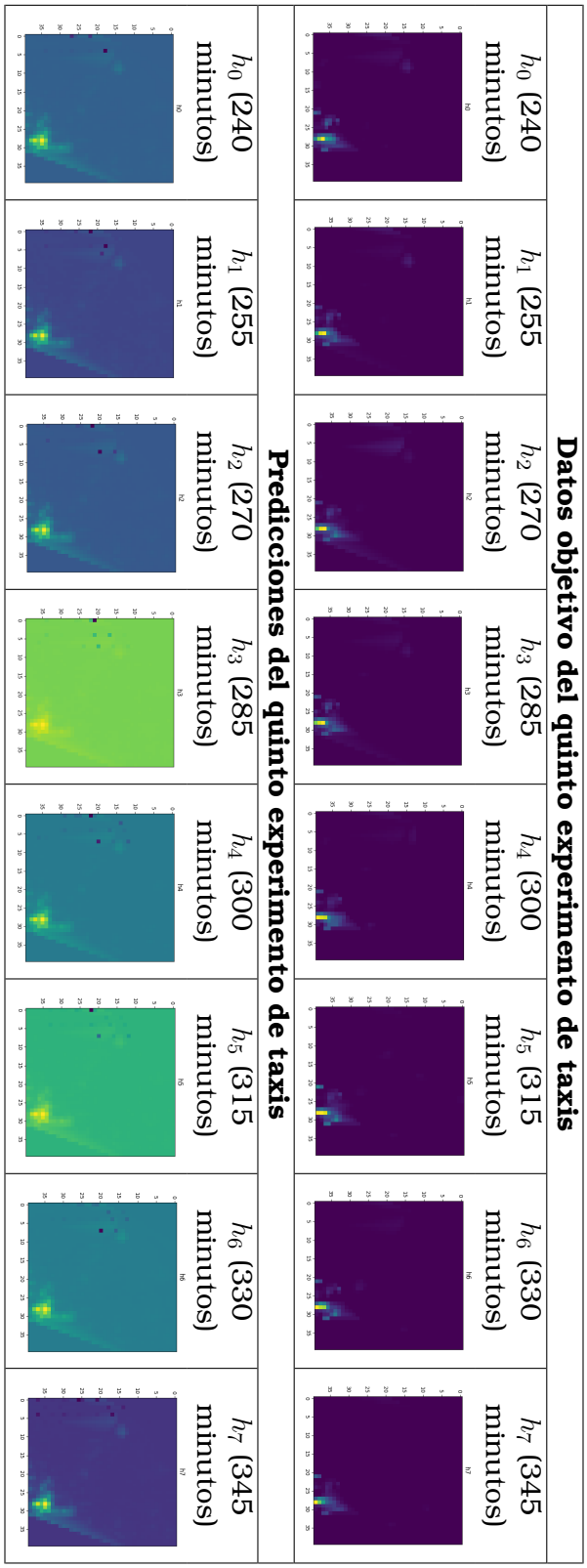


Tabla 3.11: Resultados del quinto experimento con el conjunto de datos de taxis donde x_n=8, x_y=8 y shift=8. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(\dot{i} + s))$ siendo $i \in \{0, ..., n_y\}$.

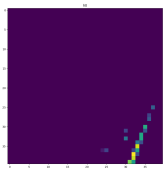
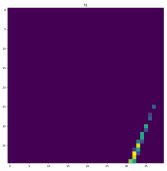
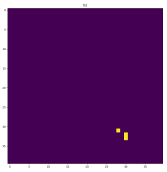
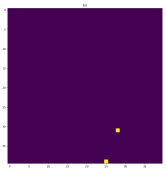
Datos contextuales del primer experimento de bicicletas			
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)
			

Tabla 3.12: Mapas de calor contextuales del primer experimento con el conjunto de datos de bicicletas donde x_n=4, x_y=4 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

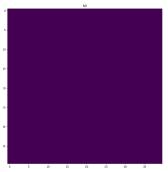
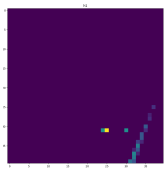
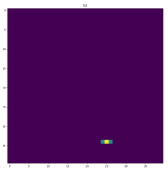
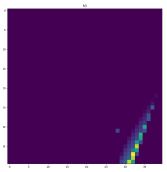
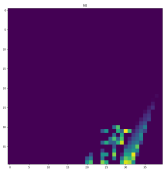
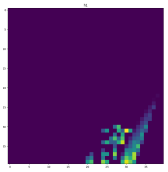
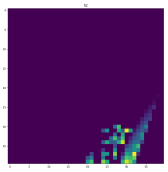
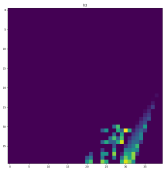
Datos objetivo del primer experimento de bicicletas			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			
Predicciones del primer experimento de bicicletas			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			

Tabla 3.13: Resultados del primer experimento con el conjunto de datos de bicicletas donde x_n=4, x_y=4 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(\dot{i} + s))$ siendo $i \in \{0, ..., n_y\}$.

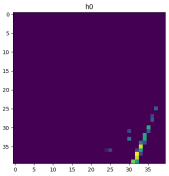
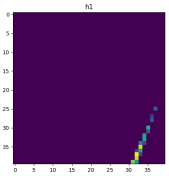
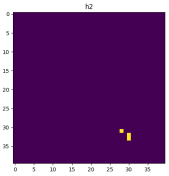
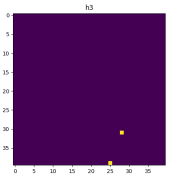
Datos contextuales del segundo experimento de bicicletas			
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)
			

Tabla 3.14: Mapas de calor contextuales del segundo experimento con el conjunto de datos de bicicletas donde x_n=4, x_y=8 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

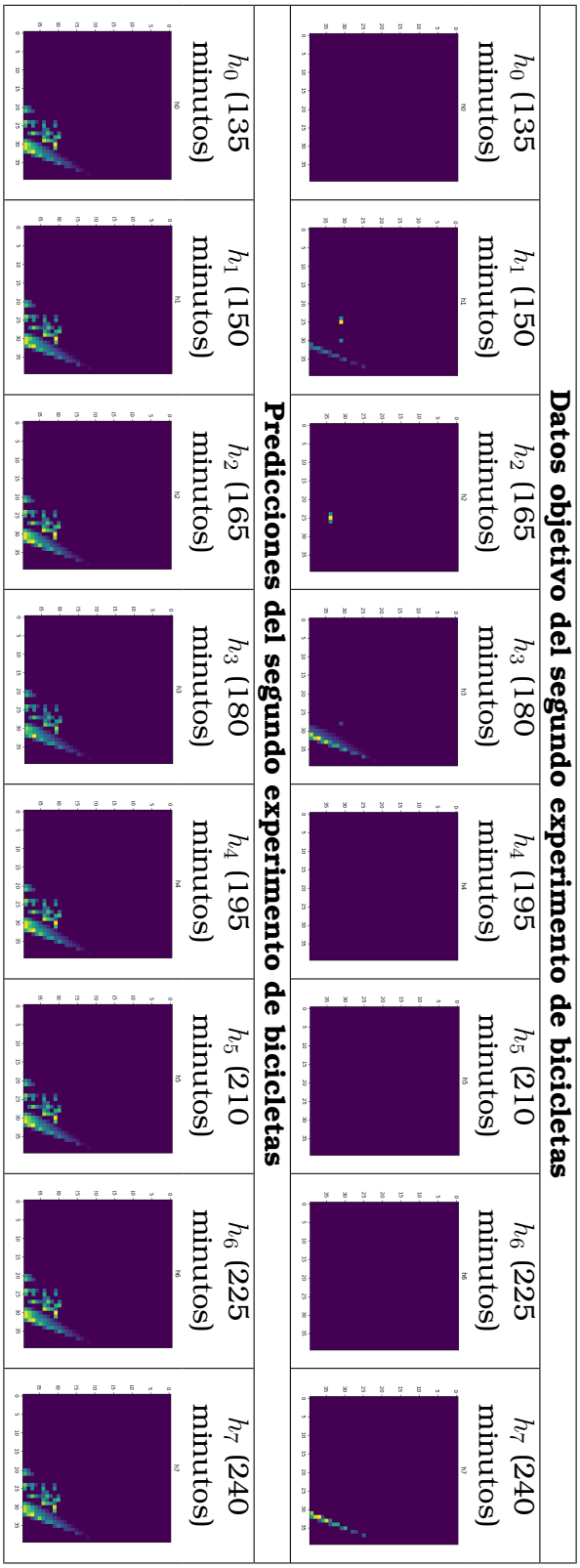


Tabla 3.15: Resultados del segundo experimento con el conjunto de datos de bicicletas donde x_n=4, x_y=8 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(i + s))$ siendo $i \in \{0, ..., n_y\}$.

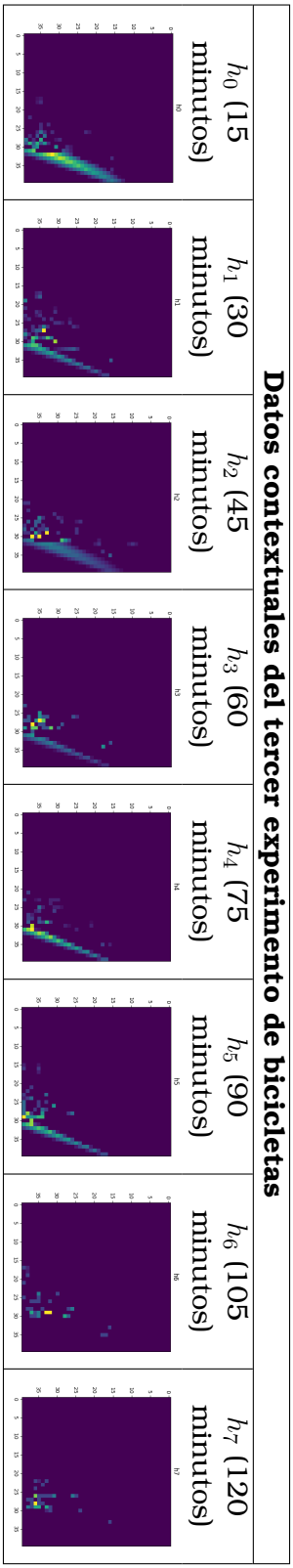


Tabla 3.16: Mapas de calor contextuales del tercer experimento con el conjunto de datos de bicicletas donde x_n=8, x_y=4 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

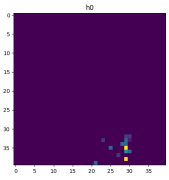
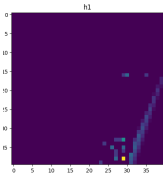
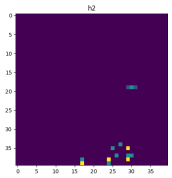
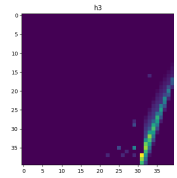
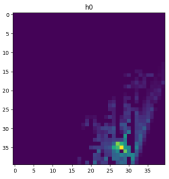
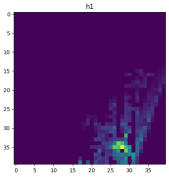
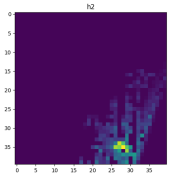
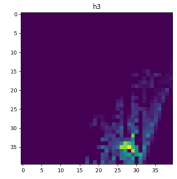
Datos objetivo del tercer experimento de bicicletas			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			
Predicciones del tercer experimento de bicicletas			
h_0 (135 minutos)	h_1 (150 minutos)	h_2 (165 minutos)	h_3 (180 minutos)
			

Tabla 3.17: Resultados del tercer experimento con el conjunto de datos de bicicletas donde x_n=8, x_y=4 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(i + s))$ siendo $i \in \{0, ..., n_y\}$.

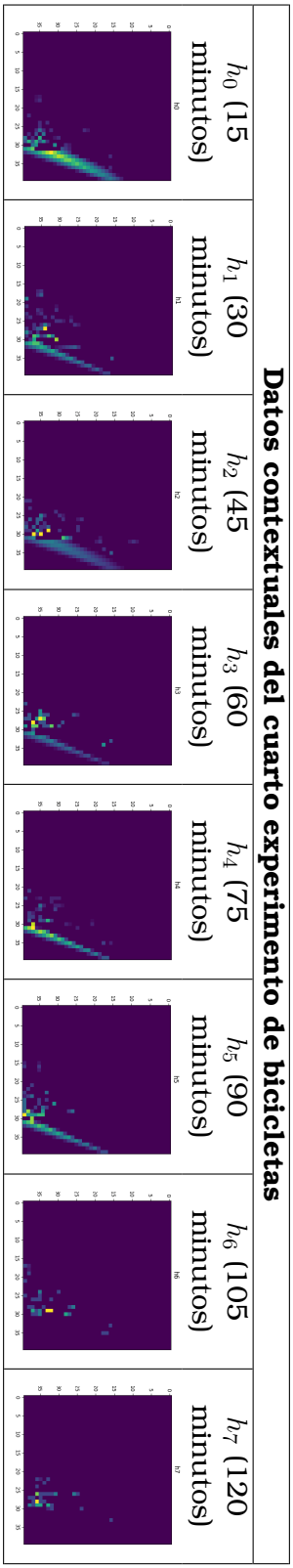


Tabla 3.18: Mapas de calor contextuales del cuarto experimento con el conjunto de datos de bicicletas donde x_n=8, x_y=8 y shift=4. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

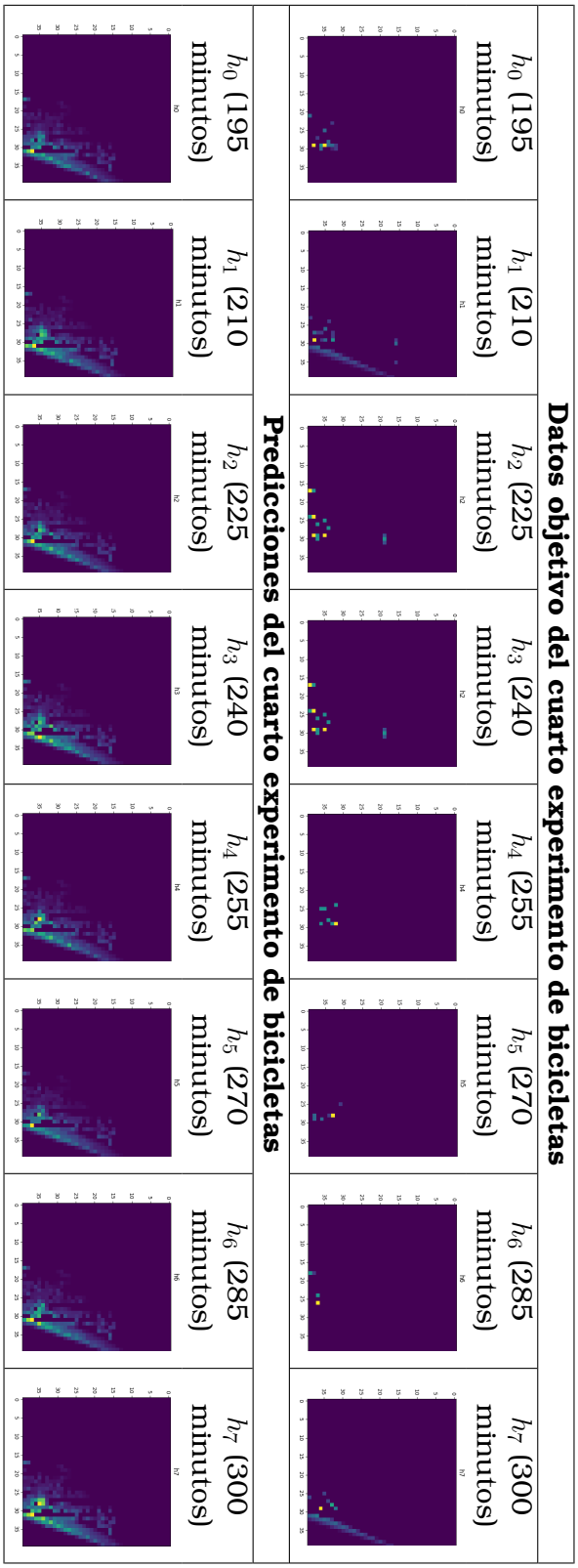


Tabla 3.19: Resultados del cuarto experimento con el conjunto de datos de bicicletas donde x_n=8, x_y=8 y shift=4. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15\dot{(}i + s))$ siendo $i \in \{0, ..., n_y\}$.

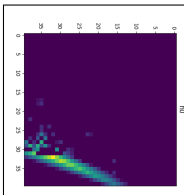
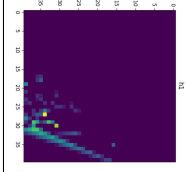
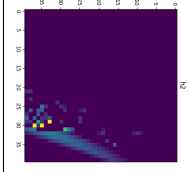
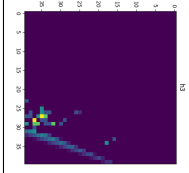
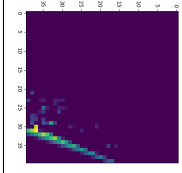
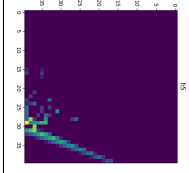
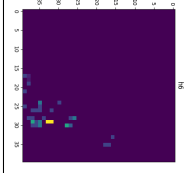
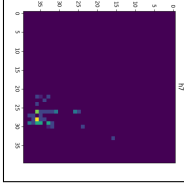
Datos contextuales del quinto experimento de bicicletas							
h_0 (15 minutos)	h_1 (30 minutos)	h_2 (45 minutos)	h_3 (60 minutos)	h_4 (75 minutos)	h_5 (90 minutos)	h_6 (105 minutos)	h_7 (120 minutos)
							

Tabla 3.20: Mapas de calor contextuales del quinto experimento con el conjunto de datos de bicicletas donde x_n=8, x_y=8 y shift=8. Estos mapas de calor pertenecen a diferentes horizontes (h_i) a lo largo del tiempo. Un tiempo definido por horizontes h en los que: $h_i = 15\dot{i}$ minutos siendo $i \in \{0, ..., 7\}$.

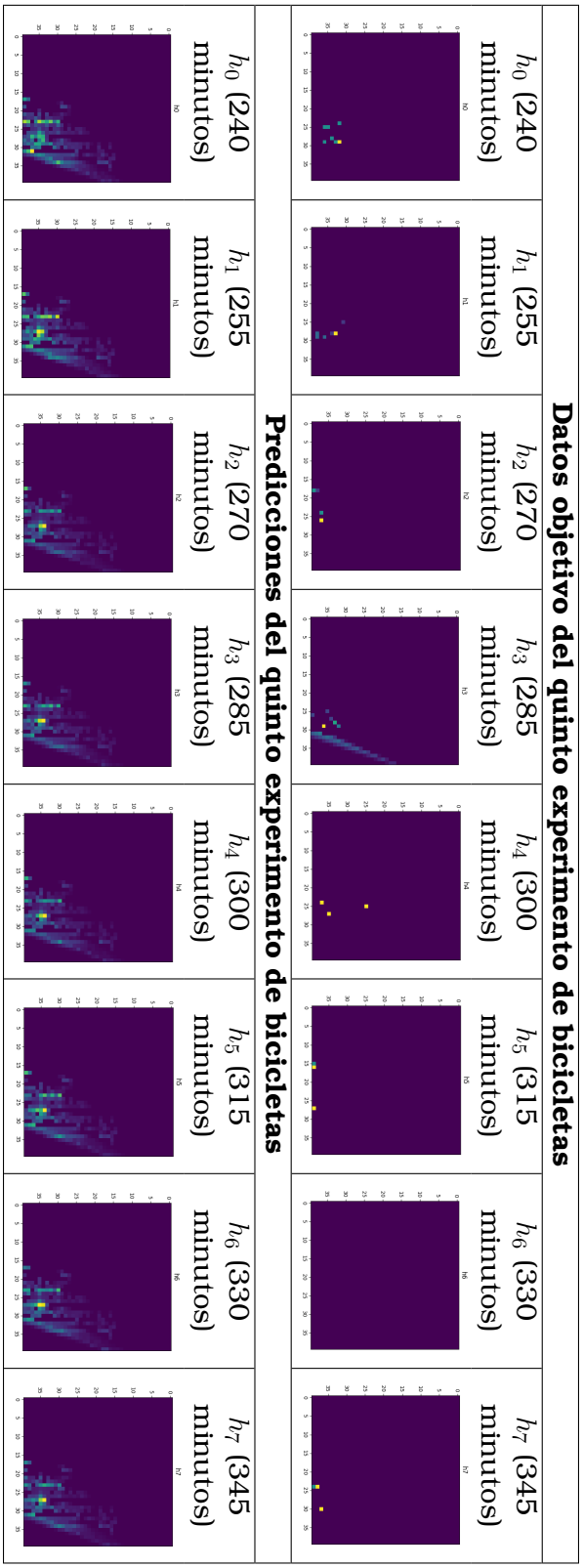


Tabla 3.21: Resultados del quinto experimento con el conjunto de datos de bicicletas donde x_n=8, x_y=8 y shift=8. Los mapas de calor correspondientes a los objetivos y las predicciones obtenidas pertenecen a diferentes horizontes (h_i) a lo largo del tiempo partiendo del último horizonte contextual. Un tiempo definido por horizontes h en los que: $h_i = (n_x * 15) + (15(\dot{i} + s))$ siendo $i \in \{0, ..., n_y\}$.

Valores resultantes métricas de error						
$n_x, n_y, shift$	**Transporte**	**MAE**	**MSE**	**MAPE**	**SMAPE**	**RMSE**
4,4,4	Taxis	0.0181	0.0100	1.6771	0.8557	0.1000
	Bicicletas	0.0856	0.0188	1.9127	0.9210	0.1371
4,8,4	Taxis	0.1863	0.0706	1.1334	1.3067	0.2657
	Bicicletas	0.0923	0.0213	1.6582	1.0588	0.1459
8,4,4	Taxis	0.0218	0.0248	1.180	0.4823	0.1574
	Bicicletas	0.0788	0.0137	1.0682	0.9433	0,1170
8,8,4	Taxis	0.1552	0.0651	1.3115	1.6008	0.2551
	Bicicletas	0.0870	0.0181	1.1682	0.9932	0.1345
8,8,8	Taxis	0.1725	0.2827	3.5703	1.3701	0,5357
	Bicicletas	0.0978	0.0213	1.1846	1.2445	0,1459

Tabla 3.22: Resultados obtenidos de las métricas de error en los experimentos de taxis.

Tabla 3.23: Comparación de los resultados obtenidos de la métrica *RMSE* en experimentos de diferentes modelos sobre los conjuntos de datos de taxis a lo largo del tiempo. Un tiempo definido por intervalos *h* en los que: $h_i = 15(\dot{i+s})$ siendo $i \in \{0, ..., 7\}$. El tiempo de entrenamiento se muestra en formato *hh:mm*.

#	Model	n_x, n_y, s	Tamaño mapa	Training time	RMSE							
					h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
1	ST-MDF	4,4,4	90x60	07:35	0.146	0.147	0.150	0.153	-	-	-	-
2	LSTM	4,4,4	90x60	04:24	0.225	0.219	0.222	0.219	-	-	-	-
3	BiLSTM	4,4,4	90x60	05:29	0.194	0.191	0.190	0.190	-	-	-	-
4	Persistence	4,4,4	90x60	-	0.143	0.151	0.158	0.165	-	-	-	-
5	Naive	4,4,4	90x60	-	0.151	0.158	0.165	0.172	-	-	-	-
6	**STT-MDF**	4,4,4	40x40	71:39	0.179	0.178	0.188	0.205	-	-	-	-
7	ST-MDF	4,8,4	90x60	09:40	0.154	0.153	0.153	0.154	0.156	0.158	0.162	0.166
8	LSTM	4,8,4	90x60	05:44	0.196	0.195	0.195	0.195	0.196	0.193	0.194	0.193
9	BiLSTM	4,8,4	90x60	07:50	0.175	0.175	0.174	0.173	0.173	0.173	0.174	0.174
10	Persistence	4,8,4	90x60	-	0.143	0.151	0.158	0.165	0.171	0.178	0.184	0.190
11	Naive	4,8,4	90x60	-	0.151	0.158	0.165	0.172	0.178	0.184	0.189	0.195
12	**STT-MDF**	4,8,4	40x40	72:11	0.431	0.393	0.389	0.339	0.308	0.337	0.238	0.358
13	ST-MDF	8,4,4	90x60	13:05	0.145	0.145	0.147	0.150	-	-	-	-
14	LSTM	8,4,4	90x60	06:42	0.186	0.188	0.191	0.196	-	-	-	-
15	BiLSTM	8,4,4	90x60	08:18	0.192	0.193	0.192	0.193	-	-	-	-
16	Persistence	8,4,4	90x60		0.143	0.151	0.158	0.165	-	-	-	-
17	Naive	8,4,4	90x60	-	0.163	0.169	0.175	0.181	-	-	-	-
18	**STT-MDF**	8,4,4	40x40	77:29	0.240	0.224	0.209	0.217	-	-	-	-
19	ST-MDF	8,8,4	90x60	14:52	0.153	0.151	0.150	0.151	0.152	0.155	0.158	0.162
20	LSTM	8,8,4	90x60	07:44	0.199	0.202	0.205	0.201	0.210	0.212	0.210	0.220
21	BiLSTM	8,8,4	90x60	11:02	0.683	0.701	0.706	0.710	0.709	0.694	0.675	0.658
22	Persistence	8,8,4	90x60	-	0.143	0.151	0.158	0.165	0.171	0.178	0.184	0.190
23	Naive	8,8,4	90x60	-	0.163	0.169	0.175	0.181	0.187	0.193	0.198	0.203
24	**STT-MDF**	8,8,4	40x40	93:29	0.381	0.370	0.340	0.317	0.318	0.293	0.273	0.249
25	ST-MDF	8,8,8	90x60	14:44	0.162	0.160	0.158	0.158	0.158	0.160	0.162	0.165
26	LSTM	8,8,8	90x60	07:48	0.170	0.166	0.162	0.161	0.160	0.160	0.159	0.160
27	BiLSTM	8,8,8	90x60	11:18	0.139	0.141	0.142	0.144	0.145	0.146	0.148	0.150
28	Persistence	8,8,8	90x60	-	0.171	0.178	0.184	0.190	0.194	0.200	0.205	0.210
29	Naive	8,8,8	90x60	-	0.187	0.193	0.198	0.203	0.208	0.213	0.218	0.222
30	**STT-MDF**	8,8,8	40x40	92:50	0.311	0.284	0.268	0.244	0.258	0.272	0.255	0.266

Tabla 3.24: Comparación de los resultados obtenidos de la métrica *RMSE* en experimentos de diferentes modelos sobre los conjuntos de datos de las bicicletas a lo largo del tiempo. Un tiempo definido por intervalos *h* en los que: $h_i = 15\dot{(}i + s)$ siendo $i \in \{0, ..., 7\}$. El tiempo de entrenamiento se muestra en formato *hh:mm*.

#	Model	n_x, n_y, s	Tamaño mapa	Training time	RMSE							
					h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7
1	ST-MDF	4,4,4	90x60	07:35	0.085	0.086	0.086	0.086	-	-	-	-
2	LSTM	4,4,4	90x60	04:24	0.136	0.135	0.144	0.142	-	-	-	-
3	BiLSTM	4,4,4	90x60	05:29	0.080	0.082	0.082	0.082	-	-	-	-
4	Persistence	4,4,4	90x60	-	0.079	0.084	0.088	0.092	-	-	-	-
5	Naive	4,4,4	90x60	-	0.075	0.079	0.083	0.086	-	-	-	-
6	**STT-MDF**	4,4,4	40x40	70:16	0.229	0.104	0.201	0.119	-	-	-	-
7	ST-MDF	4,8,4	90x60	09:40	0.085	0.085	0.086	0.086	0.086	0.086	0.086	0.087
8	LSTM	4,8,4	90x60	05:44	0.080	0.083	0.081	0.080	0.080	0.086	0.084	0.082
9	BiLSTM	4,8,4	90x60	07:50	0.082	0.082	0.083	0.084	0.085	0.086	0.087	0.087
10	Persistence	4,8,4	90x60	-	0.079	0.084	0.088	0.092	0.095	0.097	0.099	0.101
11	Naive	4,8,4	90x60	-	0.075	0.079	0.083	0.086	0.088	0.09	0.092	0.093
12	**STT-MDF**	4,8,4	40x40	71:13	0.219	0.179	0.230	0.111	0.333	0.180	0.201	0.290
13	ST-MDF	8,4,4	90x60	13:05	0.085	0.086	0.086	0.086	-	-	-	-
14	LSTM	8,4,4	90x60	06:42	0.091	0.092	0.092	0.092	-	-	-	-
15	BiLSTM	8,4,4	90x60	08:18	0.099	0.096	0.096	0.096	-	-	-	-
16	Persistence	8,4,4	90x60	-	0.080	0.084	0.088	0.092	-	-	-	-
17	Naive	8,4,4	90x60	-	0.078	0.081	0.083	0.086	-	-	-	-
18	**STT-MDF**	8,4,4	40x40	82:11	0.179	0.098	0.140	0.091	-	-	-	-
19	ST-MDF	8,8,4	90x60	14:52	0.085	0.085	0.086	0.086	0.086	0.086	0.086	0.087
20	LSTM	8,8,4	90x60	07:44	0.097	0.094	0.099	0.093	0.092	0.092	0.090	0.090
21	BiLSTM	8,8,4	90x60	11:02	0.301	0.309	0.306	0.294	0.280	0.261	0.248	0.243
22	Persistence	8,8,4	90x60	-	0.080	0.084	0.088	0.092	0.095	0.097	0.099	0.101
23	Naive	8,8,4	90x60	-	0.078	0.081	0.083	0.086	0.088	0.089	0.090	0.092
24	**STT-MDF**	8,8,4	40x40	95:37	0.201	0.170	0.193	0.180	0.221	0.263	0.284	0.222
25	ST-MDF	8,8,8	90x60	14:44	0.087	0.087	0.087	0.087	0.087	0.087	0.088	0.088
26	LSTM	8,8,8	90x60	07:48	0.087	0.084	0.081	0.080	0.078	0.077	0.078	0.079
27	BiLSTM	8,8,8	90x60	11:18	0.062	0.061	0.061	0.062	0.062	0.063	0.064	0.065
28	Persistence	8,8,8	90x60	-	0.095	0.097	0.099	0.101	0.102	0.103	0.104	0.105
29	Naive	8,8,8	90x60	-	0.088	0.089	0.090	0.092	0.093	0.094	0.094	0.095
30	**STT-MDF**	8,8,8	40x40	93:14	0.236	0.312	0.197	0.117	0.249	0.331	0.371	0.368

Capítulo 4

Conclusiones

Este trabajo fin de máster ha consistido en desarrollar un modelo "Transformer" para poder llevar a cabo la tarea de predicción de demanda de movilidad del transporte público en la ciudad de Chicago, concretamente de las bicicletas y los taxis. En el modelo propuesto, ha sido necesario tratar con dependencias temporales y espaciales, de tal manera que se han capturado las diferentes relaciones entre dichas dependencias con el objetivo de conseguir las predicciones de las series temporales pertinentes. Estas relaciones se percibían a través de los múltiples mecanismos de atención que formaban cada una de las capas del codificador y decodificador, concretamente, los mecanismos de tipo "global" y "local" conocidos en los modelos "Transformer" como "self-attention" y "cross-attention".

Respecto a esta memoria, cabe destacar la extensa contextualización del problema a resolver y de los "Transformers", incluso mencionando las múltiples investigaciones similares hasta la fecha. Esta información ha sido totalmente necesaria para más adelante poder llevar a cabo el desarrollo del modelo, ya que está basado en diferentes modelos de las investigaciones mencionadas.

Los resultados obtenidos tras realizar los experimentos pertinentes con ambos conjuntos de datos son bastante similares. Aunque se podría decir que los experimentos correspondientes a los conjuntos de datos de bicicletas tienen unos resultados ligeramente mejores. Estos resultados se han visto afectados por una limitación computacional que ha hecho que a diferentes hiperparametros como el número de capas codificador-decodificador, el número de cabezas de atención o el propio tamaño de la "mesh-grid" se les asignase valores mas pequeños que los deseados. Por ejemplo, los conjuntos de datos de bicicletas y taxis se basaban en una "mesh-grid" de 90x60 (representación del mapa de Chicago de 500mx500m), mientras que en los experimentos se ha limitado a 40x40. La conclusión obtenida tras finalizar la fase de ejecuciones es, que pese a sufrir esta limitación computacional, las predicciones son bastante buenas. Es cierto que no son totalmente precisas pero, tienen una semejanza a la realidad muy notable y seguramente con algo mas de capacidad los resultados serían muy buenos.

En cuanto a las posibles mejoras, la primera de estas sería la más clara de todas, ya que se basaría en poder finalizar este trabajo realizando los experimentos necesarios con la capacidad computacional adecuada. Otra posible modificación, la cual ya está incluida en el trabajo de [Prado-Rujas *et al.*, 2023] sería poder tratar dentro

del modelo con las dependencias meteorológicas lo cual cambiaría los resultados de las predicciones por completo, puesto que en los días con condiciones meteorológicas adversas o peligrosas (rachas de viento, lluvia, niebla, hielo, nieve, etc.) la demanda de la movilidad del transporte público probablemente sería menor. Además, una idea relacionada con esta última posible futura modificación sería tener en cuenta las 4 estaciones del año a la hora de procesar los datos. Respecto a posibles futuros trabajos que obtengan este como base, sería la implementación de las "Graph Neural Networks" (GNN) para tratar principalmente con las dependencias espaciales. Otra opción sería añadir diferentes puntos de interés teniendo en cuanta las dependencias temporales, es decir, un concierto en un día en concreto podría revolucionar por completo las predicciones.

En conclusión, realizar este trabajo fin de máster ha sido bastante interesante debido a que el objetivo principal es solucionar un problema actual como es el la congestión de trafico. en las grandes ciudades como Chicago. Además, hacerlo mediante un tipo de modelo como son los"Transformers" del cual no se conoce tanto como de otros tipos de modelos ya que, su nacimiento fue tan solo hace 6 años ha sido un reto interesante. Al igual que la parte de alimentarme de información sobre el problema a resolver y el modelo con el que se busca resolver el problema sin la necesidad de viajar muchos años hacia el pasado para poder encontrar la información idónea.

Bibliografía

[Agarap, 2018] Agarap, A.F. (2018). *Deep Learning using Rectified Linear Units (ReLU).* ArXiv, abs/1803.08375.

[Araabi and Monz, 2020] Araabi, A., and Monz, C. (2020). *Optimizing Transformer for Low-Resource Neural Machine Translation.* ArXiv, abs/2011.02266.

[Arnab *et al.*, 2021] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić and C. Schmid. (2021). *ViViT: A Video Vision Transformer*. 2021. IEEE/CVF International Conference on Computer Vision (ICCV), 6816-6826.

[Atwood and Towsley, 2015] Atwood, J., and Towsley, D.F. (2015). *Diffusion-Convolutional Neural Networks.* NIPS.

[BBC News, 2017] BBC News. (2017)*Singapur: Singapore to freeze car numbers.* Accessed: 20.

[Bahdanau *et al.*, 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate.*arXiv preprint arXiv:1409.0473.

[Borovykh *et al.*, 2017] Borovykh, A., Bohté, S.M., and Oosterlee, C.W. (2017). *Conditional Time Series Forecasting with Convolutional Neural Networks.* https://arxiv.org/pdf/1703.04691.pdf

[Brown *et al.*, 2020] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T.J., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). *Language Models are Few-Shot Learners.* ArXiv, abs/2005.14165.

[Bahdanau *et al.*, 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate.*arXiv preprint arXiv:1409.0473.

[Bugueño *et al.*, 2019] Bugueño, Margarita and Mendoza, Marcelo. (2019). Applying Self-attention for Stance Classification. 10.1007/978-3-030-33904-3_5.

[Cai *et al.*, 2020] Cai, L, Janowicz, K, Mai, G, Yan, B and Zhu, R. (2020) *Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting.* Transactions in GIS. 24: 736– 755.

[Chai *et al.*, 2018] Chai, D., Wang, L., and Yang, Q. (2018). *Bike flow prediction with multi-graph convolutional networks.* Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 397-400, 10.1145/3274895.3274896.

[Child *et al.*, 2019] Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). *Generating Long Sequences with Sparse Transformers.* ArXiv, abs/1904.10509.

[Child *et al.*, 2019] Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). *Generating Long Sequences with Sparse Transformers.* ArXiv, abs/1904.10509.

[Choromanski *et al.*, 2020] Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L.J., and Weller, A. (2020). *Rethinking Attention with Performers.* ArXiv, abs/2009.14794.

[Chung *et al.*, 2014] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* https://arxiv.org/pdf/1412.3555.pdf

[Clevert *et al.*, 2015] Clevert, D., Unterthiner, T., and Hochreiter, S. (2015). *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* arXiv: Learning.

[Cui *et al.*, 2018] Cui, Z., Ke, R., and Wang, Y. (2018). *Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction.* International Workshop on Urban Computing (UrbComp). arXiv:1801.02143.

[Dai *et al.*, 2018] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov. (2019). ACL (1), page 2978-2988. Association for Computational Linguistics.

[Davis *et al.*, 2019] Davis, N., Raina, G., and Jagannathan, K. (2019). *Grids versus graphs: Partitioning space for improved taxi demand-supply forecasts. Preprint.* arXiv:1902.06515

[David, 2015] David, C. (2015) *Traffic and congestion cost trends for Australian capital cities.* Department of Infrastructure and Regional Development, Bureau of Infrastructure, Transport and Regional Economics, Canberra.

[Defferrard *et al.*, 2016] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.* NIPS.

[Dosovitskiy *et al.*, 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.* https://arxiv.org/pdf/2010.11929.pdf

[Du *et al.*, 2022] Du, D., Su, B., and Wei, Z. (2022). *Preformer: Predictive Transformer with Multi-Scale Segment-wise Correlations for Long-Term Time Series Forecasting.* ArXiv, abs/2202.11356.

[Falcon *et al.*, 2020] Falcon, W., Borovec, J., Eggert, N.S., Bereznyuk, V., dXD, I., Wälchli, A., Jordan, J., Præsius, S.K., Murrell, T., Harris, E., Bapat, S.V., Schröter, H., Kulkarni, A., Haunschmid, V., Lipin, D., Singh, A.P., Fan, T.J.,

Skafte, N., Mary, H., Eyzaguirre, C., cinjon, Bakhtin, A., Zh, Z., Jo, Y., Izsak, P., Rangel, O.A., Ling, J., Sharma, H., Waite, E., and Aydin, A. (2020). *PyTorchLightning/pytorch-lightning: TPU support and profiling.* Github - Lightning AI

[Fan *et al.*, 2019] Fan, A., Grave, E., and Joulin, A. (2019). *Reducing Transformer Depth on Demand with Structured Dropout.* ArXiv, abs/1909.11556.

[Fiorello *et al.*, 2015] Joint Research Centre, Institute for Prospective Technological Studies, Fiorello, D., Zani, L. (2015), *EU survey on issues related to transport and mobility*. Publications Office

[Flunkert *et al.*, 2017] Flunkert, V., Salinas, D., and Gasthaus, J. (2017). *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks.* ArXiv, abs/1704.04110.

[Gehring *et al.*, 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. (2017). *Convolutional Sequence to Sequence Learning.* International Conference on Machine Learning.

[Gilmer *et al.*, 2017] Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., and Dahl, G.E. (2017). *Neural Message Passing for Quantum Chemistry.* ArXiv, abs/1704.01212.

[Grigsby *et al.*, 2021] Grigsby, J., Wang, Z., and Qi, Y. (2021). *Long-Range Transformers for Dynamic Spatiotemporal Forecasting*. arXiv:2109.12218.

[Guo *et al.*, 2019] Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. (2019). *Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting*. AAAI Conference on Artificial Intelligence.

[Habtemichael and Cetin, 2016] Habtemichael, F.G., and Cetin, M. (2016). *Short-term traffic flow rate forecasting based on identifying similar traffic patterns.* Transportation Research Part C-emerging Technologies, 66, 61-78. 10.1016/j.trc.2015.08.017.

[He *et al.*, 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). *Deep Residual Learning for Image Recognition.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[Hendrycks and Gimpel, 2016] Hendrycks, D., and Gimpel, K. (2016). *Gaussian Error Linear Units (GELUs).* arXiv: Learning.

[Hoang *et al.*, 2016] Hoang, M.X., Zheng, Y., and Singh, A. (2016). *FCCF: forecasting citywide crowd flows based on big data.* Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 1-10, 10.1145/2996913.2996934.

[Hochreiter and Bengio, 2001] Hochreiter, S., and Bengio, Y. (2001). *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.* IEEE, 2001, pp.237-243

[Hochreiter and Schmidhuber, 1997] Hochreiter, S., and Schmidhuber, J. (1997). *Long Short-Term Memory*. Neural Computation, 9, 1735-1780.

[Huang *et al.*, 2020] Huang, X.S., Perez, F., Ba, J. and Volkovs, M.. (2020). *Improving Transformer Optimization Through Better Initialization*. Proceedings of the 37th International Conference on Machine Learning in Proceedings of Machine Learning Research 119:4475-4483.

[Ioffe and Szeged, 2015] Ioffe, S., and Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* International Conference on Machine Learning.

[Iwata and Kumagai, 2020] Iwata, T., and Kumagai, A. (2020). *Few-shot Learning for Time-series Forecasting.* ArXiv, abs/2009.14379.

[Jacob *et al.*, 2019] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019).*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* ArXiv, abs/1810.04805.

[Jian, 2022] Jiang, W. (2022). *Bike sharing usage prediction with deep learning: a survey.* Neural Computing and Applications, 34, 15369 - 15385.

[Jiang *et al.*, 2018] Jiang, Z., Fan, W.D., Liu, W., Zhu, B., and Gu, J. (2018). *Reinforcement Learning Approach for Coordinated Passenger Inflow Control of Urban Rail Transit in Peak Hours.* Transportation Research Part C: Emerging Technologies

[Jin *et al.*, 2018] Jin, W., Lin, Y., Wu, Z., and Wan, H. (2018). *Spatio-temporal recurrent convolutional networks for citywide short-term crowd flows prediction.* In Proceedings of the Second International Conference on Compute and Data Analysis, DeKalb, IL (pp. 28–35). New York, NY: ACM.

[Kazemi *et al.*, 2019] Kazemi, S.M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., and Brubaker, M.A. (2019). *Time2Vec: Learning a Vector Representation of Time.* ArXiv, abs/1907.05321.

[Khaidem *et al.*, 2020] Khaidem, L., Luca, M., Yang, F., Anand, A., Lepri, B., and Dong, W. (2020). *Optimizing Transportation Dynamics at a City-Scale Using a Reinforcement Learning Framework.* . IEEE Access, 8, 171528-171541

[Kingma and Welling, 2013] Kingma, D.P., and Welling, M. (2013). *Auto-Encoding Variational Bayes.* CoRR, abs/1312.6114.

[Kipf and Welling, 2016] Kipf, T., and Welling, M. (2016). *Semi-Supervised Classification with Graph Convolutional Networks.* ArXiv, abs/1609.02907.

[Kitaev *et al.*, 2020] Kitaev, N., Kaiser, L., and Levskaya, A. (2020). *Reformer: The Efficient Transformer.* ArXiv, abs/2001.04451.

[Krizhevsky *et al.*, 2012] Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012). *ImageNet classification with deep convolutional neural networks.* Communications of the ACM, 60, 84 - 90.

[Lai *et al.*, 2017] Lai, G., Chang, W., Yang, Y., and Liu, H. (2017). *Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks.* The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval.

[Li *et al.*, 2012] Li, X., Pan, G., Wu, Z., Qi G.,Li S., Zhang D., Zhang W. and Wang Z. (2012) *Prediction of urban human mobility using large-scale taxi traces and its applications*. Front. Comput. Sci. 6, 111–121.

[Li *et al.*, 2017] Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2017). *Graph Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting*. arxiv:1707.01926

[Li *et al.*, 2019] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y. Wang, and X. Yan. (2019). *Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting*. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 471, 5243–5253.

[Li *et al.*, 2021] Li, X., Xu, Y., Chen, Q., Wang, L., Zhang, X., and Shi, W. (2021). *Short-Term Forecast of Bicycle Usage in Bike Sharing Systems: A Spatial-Temporal Memory Network.* IEEE Transactions on Intelligent Transportation Systems, 23, 10923-10934.

[Lin *et al.*, 2018] Lin, L., He, Z., Peeta, S., and Wen, X. (2017). *Predicting Station-level Hourly Demands in a Large-scale Bike-sharing Network: A Graph Convolutional Neural Network Approach*. ArXiv, abs/1712.04997.

[LI *et al.*, 2019] LI, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y., and Yan, X. (2019). *Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting.* ArXiv, abs/1907.00235.

[Lippi *et al.*, 2013] Lippi, M., Bertini, M., and Frasconi, P. (2013). S*hort-Term Traffic Flow Forecasting: An Experimental Comparison of Time-Series Analysis and Supervised Learning.* IEEE Transactions on Intelligent Transportation Systems, 14, pp. 871-882, 10.1109/TITS.2013.2247040.

[Liu *et al.*, 2020] Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. (2020). *Understanding the Difficulty of Training Transformers. Conference on Empirical Methods in Natural Language Processing*. arXiv:2004.08249

[Liu *et al.*, 2020] Liu, T., Wu, W., Zhu, Y., and Tong, W. (2020). *Predicting taxi demands via an attention-based convolutional recurrent neural network.* Knowl. Based Syst., 206, 106294. 10.1016/j.knosys.2020.106294.

[Liu *et al.*, 2022] Liu, S., Yu, H., Liao, C., Li, J., Lin, W., Liu, A.X., and Dustdar, S. (2022). *Pyraformer: Low-Complexity Pyramidal Attention for Long-Range Time Series Modeling and Forecasting.* International Conference on Learning Representations.

[Ma *et al.*, 2013] Ma S., Zheng Y. and Wolfson O. (2013). *T-share: A large-scale dynamic taxi ridesharing service*. IEEE 29th International Conference on Data Engineering, ICDE pp. 410-421, 10.1109/ICDE.2013.6544843.

[Ma *et al.*, 2015] Ma, X., Tao, Z., Wang, Y., Yu, H., and Wang, Y. (2015). *Long short-term memory neural network for traffic speed prediction using remote microwave sensor data*. Transportation Research Part C-emerging Technologies, 54, 187-197.

[Ma *et al.*, 2016] Ma, Y., Lin, T., Cao, Z., Li, C., Wang, F., and Chen, W. (2016). *Mobility Viewer: An Eulerian Approach for Studying Urban Crowd Flow.* IEEE Transactions on Intelligent Transportation Systems, 17, 2627-2636.

[Ma *et al.*, 2017] Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., and Wang, Y. (2017). *Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction.* Sensors (Basel, Switzerland), 17.

[Madhusudhanan *et al.*, 2021] Madhusudhanan, K., Burchert, J., Duong-Trung, N., Born, S., and Schmidt-Thieme, L. (2021). *Yformer: U-Net Inspired Transformer Architecture for Far Horizon Time Series Forecasting.* ArXiv, abs/2110.08255.

[Manish, 2017] Manish, C. (2017). *Sequence to sequence model: Introduction and concepts* Towards Data Science

[Miao *et al.*, 2016] Miao, F., Han, S., Lin, S., Stankovic, J.A., Huang, H., Zhang, D., Munir, S., He, T., and Pappas, G.J. (2015). *Taxi Dispatch With Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach.* IEEE Transactions on Automation Science and Engineering, 13, 463-478.

[Min and Wynter, 2011] Min, W., and Wynter, L. (2011). *Real-time road traffic prediction with spatio-temporal correlations.* Transportation Research Part C-emerging Technologies, 19, 606-616.

[Mingsing Xu *et al.*, 2020] Xu, M., Dai, W., Liu, C., Gao, X., Lin, W., Qi, G., and Xiong, H. (2020). *Spatial-Temporal Transformer Networks for Traffic Flow Forecasting.* arXiv:2001.02908.

[Mikolov *et al.*, 2013] Mikolov, T., Chen, K., Corrado, G.S., and Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space.* International Conference on Learning Representations.

[Moreira-Matias *et al.*, 2012] Moreira-Matias, L., Gama, J., Ferreira, M., and Damas, L. (2012). *A predictive model for the passenger demand on a taxi network.* 15th International IEEE Conference on Intelligent Transportation Systems, 1014-1019.

[Nguyen and Salazar, 2019] Nguyen, T.Q., and Salazar, J. (2019).*Transformers without Tears: Improving the Normalization of Self-Attention.* ArXiv, abs/1910.05895.

[OpenAI, 2023] OpenAI (2023). *GPT-4 Technical Report.* ArXiv, abs/2303.08774.

[Oreshkin *et al.*, 2020] Oreshkin, B.N., Carpov, D., Chapados, N., and Bengio, Y. (2020). *Meta-learning framework with applications to zero-shot time-series forecasting.* ArXiv, abs/2002.02887.

[O'Shea and Nash, 2015] O'Shea, K., and Nash, R. (2015). *An Introduction to Convolutional Neural Networks.* ArXiv, abs/1511.08458.

[Pan *et al.*, 2019] Pan, Z., Liang, Y., Wang, W., Yu, Y., Zheng, Y., and Zhang, J. (2019). *Urban Traffic Prediction from Spatio-Temporal Data Using Deep Meta Learning.* Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[Paszke *et al.*, 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). *PyTorch: An Imperative Style, HighPerformance Deep Learning Library* ArXiv, abs/1912.01703.

[Prado-Rujas *et al.*, 2023] Prado-Rujas, I., Serrano, E., Dopico, A.G., Córdoba, M.L., and Pérez, M.S. (2023).*Combining heterogeneous data sources for spatio-temporal mobility demand forecasting.* Information Fusion, vol. 91, pp 1-12, ISSN 1566-2535.

[Python S.F., 2023] Python Software Foundation, 2023. *Python Programming Language.* Python-org

[Radford *et al.*, 2018] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018) *Improving language understanding by generative pre-training.* language-understanding-paper.pdf

[Radford *et al.*, 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners.* unsupervised-multitask-learners.pdf

[Raffel *et al.*, 2020] Raffel, C., Shazeer, N.M., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P.J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. ArXiv, abs/1910.10683.

[Rodrigues *et al.*, 2018] Rodrigues, F., Markou, I., and Pereira, F.C. (2018).*Combining time-series and textual data for taxi demand prediction in event areas: a deep learning approach.* ArXiv, abs/1808.05535.

[Roos *et al.*, 2017] Roos, J., Gavin, G., and Bonnevay, S. (2017). *A dynamic Bayesian network approach to forecast short-term urban rail passenger flows with incomplete data.* Transportation research procedia, 26, pp. 53-61, 10.1016/j.trpro.2017.07.008.

[Scarselli *et al.*, 2009] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., and Monfardini, G. (2009). *The Graph Neural Network Model.* IEEE Transactions on Neural Networks, 20, 61-80.

[Schmidt, 2019] Schmidt, R.M. (2019). *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview.* ArXiv, abs/1912.05911.

[Shelhamer *et al.*, 2014] Shelhamer, E., Long, J., and Darrell, T. (2014). *Fully convolutional networks for semantic segmentation.* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3431-3440.

[Shen *et al.*, 2020] Shen, S., Yao, Z., Gholami, A., Mahoney, M.W., and Keutzer, K. (2020). *PowerNorm: Rethinking Batch Normalization in Transformers.* International Conference on Machine Learning.

[Shi *et al.*, 2015] Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-C. (2015). *Convolutional LSTM network: A machine learning approach for precipitation nowcasting.* In Proceedings of the 28th Conference on Neural Information Processing Systems, Montreal, Quebec, Canada (pp. 802–810). San Diego, CA: NIPS.

[Shih *et al.*, 2018] Shih, S., Sun, F., and Lee, H. (2018). *Temporal pattern attention for multivariate time series forecasting.* Machine Learning, 1-21.

[Smyl, 2020] Smyl, S. (2020). *A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting.* arXiv: Machine Learning.

[Sun *et al.*, 2021] Wang, Xun and Zhang, Haozhi and Huang, Weilin and Scott, and Matthew R. (2020). *CBT: Cross-Batch Memory for Embedding Learning.* arXiv:1912.06798.

[Tay *et al.*, 2020] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder and D. Metzler. (2020). *Long Range Arena: A Benchmark for Efficient Transformers*. arXiv: 2011.04006.

[T. Tsekeris and C. Tsekeris, 2011] T. Tsekeris and C. Tsekeris (2011) *Demand Forecasting in Transport: Overview and Modeling Advances*, Economic Research-Ekonomska Istraživanja, 24:1, 82-94, DOI: 10.1080/1331677X.2011.11517446.

[Vaswani *et al.*, 2017] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). *Attention is all you need.* Proceedings of NeurIPS. pp. 5998–6008.

[Velickovic *et al.*, 2017] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio', P., and Bengio, Y. (2017). *Graph Attention Networks.* ArXiv, abs/1710.10903.

[Wand and Li, 2018] Wang, J., and Li, S. (2018). *SELF-ATTENTION MECHANISM BASED SYSTEM FOR DCASE 2018 CHALLENGE TASK 1 AND TASK 4.* Semanticscholar-Corpus:52501342

[Wang *et al.*, 2020] Wang, S., Li, B.Z., Khabsa, M., Fang, H., and Ma, H. (2020). *Linformer: Self-Attention with Linear Complexity.* ArXiv, abs/2006.04768.

[Wang *et al.*, 2021] Wang, X., Sun, S., Xie, L., and Ma, L. (2021). *Efficient Conformer with Prob-Sparse Attention Mechanism for End-to-EndSpeech Recognition.* ArXiv, abs/2106.09236.

[Want *et al.*, 2014] J. Wang, W. Deng, and Y. Guo, 2014. *New bayesian combination method for short-term traffic flow forecasting.* Transp. Res. Part Emerg. Technol., vol. 43, pp. 79–94.

[Woo *et al.*, 2020] Woo, G., Liu, C., Sahoo, D., Kumar, A., and Hoi, S.C. (2022). *ETSformer: Exponential Smoothing Transformers for Time-series Forecasting.* ArXiv, abs/2202.01381.

[Wu *et al.*, 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W. and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. Preprint, arXiv:1609.08144.

[Wu and Tan, 2016] Wu, Y., and Tan, H. (2016). *Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework.* ArXiv, abs/1612.01022.

[Wu *et al*, 2018] Wu, Y., Tan, H., Qin, L., Ran, B., and Jiang, Z. (2018). *A hybrid deep learning based traffic flow prediction method and its understanding. Transportation Research Part.* C: Emerging Technologies, 90, 166–180.

[Wu *et al.*, 2019] Wu, Z., Pan, S., Long, G., Jiang, J., and Zhang, C. (2019). *Graph WaveNet for Deep Spatial-Temporal Graph Modeling.* ArXiv, abs/1906.00121.

[Wu *et al.*, 2021] Wu, H., Xu, J., Wang, J., and Long, M. (2021). *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. Neural Information Processing Systems.* arXiv:2106.13008

[Wu *et al.*, 2021] Wu, Z., Wu, L., Meng, Q., Xia, Y., Xie, S., Qin, T., Dai, X., and Liu, T. (2021). *UniDrop: A Simple yet Effective Technique to Improve Transformer without Extra Cost.* ArXiv, abs/2104.04946.

[Xiong *et al.*, 2020] Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. (2020). *On Layer Normalization in the Transformer Architecture*. International Conference on Machine Learning.

[Xu *et al.*, 2017] Xu, J., Rahmatizadeh, R., Bölöni, L., and Turgut, D. (2017). *Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks.* IEEE Transactions on Intelligent Transportation Systems, 19, pp. 2572-2581, 10.1109/TITS.2017.2755684.

[Yao *et al.*, 2018] Yao H., Wu F., Ke J., Tang X., Jia Y., Lu S., Gong P., Ye J. and Li Z. (2018). *Deep multi-view spatial-temporal network for taxi demand prediction.* Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32.

[Ye *et al.*, 2019] Ye, Z., Guo, Q., Gan, Q., Qiu, X., and Zhang, Z. (2019). BP-Transformer: Modelling Long-Range Context via Binary Partitioning. ArXiv, abs/1911.04070.

[Yu *et al.*, 2017] Yu, T., Yin, H., and Zhu, Z. (2017). *Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting*. International Joint Conference on Artificial Intelligence.

[Zaheer *et al.*, 2020] Zaheer, M., Guruganesh, G., Dubey, K.A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. (2020). *Big Bird: Transformers for Longer Sequences*. ArXiv, abs/2007.14062.

[Zerveas *et al, 2020*] Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A., and Eickhoff, C. (2020). *A Transformer-based Framework for Multivariate Time Series Representation Learning.* Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.

[Zhang *et al.*, 2011] Zhang N., Zhang Y. and Lu H. (2011) *Seasonal autoregressive integrated moving average and support vector machine models: prediction of short-term traffic flow on freeways* Transp. Res. Rec., 2215 (1), pp. 85-92, 10.3141/2215-09.

[Zhang *et al.*, 2016] Zhang, J., Zheng, Y., and Qi, D. (2016). *Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction.*AAAI Conference on Artificial Intelligence.

[Zhang *et al.*, 2020] Zhang, J., Zheng, Y., Sun, J., and Qi, D. (2020). *Flow Prediction in Spatio-Temporal Networks Based on Multitask Deep Learning.*IEEE Transactions on Knowledge and Data Engineering, 32, 468-478.

[Zhang *et al.*, 2021] Zhang, H., Gong, Y., Shen, Y., Li, W., Lv, J., Duan, N., and Chen, W. (2021). *Poolingformer: Long Document Modeling with Pooling Attention.* arXiv: 2105.04371

[Zhou *et al.*, 2020] Zhou, W., Ge, T., Xu, K., Wei, F., and Zhou, M. (2020). *Scheduled DropHead: A Regularization Method for Transformer Models.* Findings.

[Zhou *et al.*, 2020] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. (2020). *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting.* ArXiv, abs/2012.07436.

[Zhou *et al, 2022*] Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., and Jin, R. (2022). *FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting.* ArXiv, abs/2201.12740.

Anexo

Este capítulo es opcional, y se escribirá de acuerdo con las indicaciones del Tutor.