UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Máster del
Máster Universitario en Ingeniería y Ciencia de Datos

Predicción de tráfico mediante redes neuronales basadas en grafos

Luis Manuel Correas Ramos

Dirigido por: Agustín Carlos Caminero Herráez

María del Rocío Muñoz Mansilla

Curso: 2021-2022: 1ª Convocatoria

Resumen

La predicción del tráfico ha tornado a ser una tarea crucial con multitud de intereses tanto en aspectos económicos como de salud, más aún con el rápido crecimiento en el nivel de circulación debido al incremento del tamaño de las grandes ciudades. En este trabajo se propone un sistema de predicción del nivel de tráfico mediante el aprendizaje basado en datos históricos y la propagación de información en una red de carreteras compleja, como lo es la ciudad de Madrid.

A través de una simple entrada que permita identificar un instante en el futuro y condiciones básicas de la naturaleza de ese día, el sistema ofrecerá una estimación del nivel de densidad circulatoria para cada uno de los arcos que conforman la red.

El sistema propuesto es sencillo y extensible, permitiendo añadir nuevas características tanto a la entrada como a la salida, simplemente incluyéndolas en la fase de procesado de la información y ajustando los parámetros que marcan la estructura de la red.

Constará de dos partes principales y diferenciadas por su cometido y estructura: la primera encargada de procesar los valores que identifican un punto en el tiempo y ofrecer una estimación del uso de vía para cada arco en el sistema; la segunda, en su lugar, llevará a cabo la propagación espacial del estado de los arcos usando una red neuronal aplicada a grafos, con el fin de estabilizar las estimaciones incluso en los arcos donde no existe un medidor físico de tráfico. De este modo, es posible aprender de las tendencias y relaciones de ocupación de vía entre nodos y arcos vecinos, condicionadas por las características físicas de cada carretera.

La fase de aprendizaje donde se hace uso de una red neuronal aplicada a grafos ha sido diseñada mediante el uso de un *framework* que puede adaptarse a varios *backends* de aprendizaje automático, otorgando una mayor versatilidad a la aplicación y la posibilidad de computación mediante unidades de procesamiento de gráficos o sistemas distribuidos. Este sistema permite el paso de mensajes entre nodos y arcos según su conectividad y, de este modo, lograr el aprendizaje basado en la cercanía entre elementos. Gracias a esto, tiene la habilidad de ofrecer estimaciones dadas en concordancia para el resto de puntos de la red, aunque no dispongan de un medidor y por lo tanto no haya un valor de referencia con el que entrenarlos.

El sistema ha sido capaz de demostrar una precisión razonablemente buena, con un error medio inferior a 10 puntos, aun contando con información muy básica como entrada al sistema y, por tanto, pocos datos con los que tomar decisiones. Además, en pruebas de validación de la propagación espacial, tras ser retiradas la mediciones del 10 % de los arcos de la red, el incremento del error global es muy pequeño, subiendo hasta los 12-13 puntos de separación media para los puntos de medición inhibidos.

Palabras clave

Predicción de tráfico, Madrid, redes neuronales, python, OpenStreetMap, Deep Graph Library, MXNet, Pytorch, grafo, GNN, PEMS-BAY.

Índice general

1. Introducción **1**

1.1. Motivación y objetivos 1
1.2. Alcance 2
1.3. Planificación 3
1.4. Estructura de la memoria 4
1.5. Código fuente 4
1.6. Resumen 5

2. Estado de la cuestión **1**

2.1. Redes neuronales 2
2.2. Redes neuronales basadas en grafos 3
2.3. Aprendizaje del tráfico usando GNN 4
2.4. Comparativa 6
2.5. Resumen 7

3. Análisis de información **1**

3.1. Datos espaciales 1
3.2. Ubicación puntos de medida 1
3.3. Histórico de tráfico 2
3.3.1. Arcos con mediciones 3
3.3.2. Incompletitud de muestras 3
3.4. Tipología del día y festividades 4
3.5. Resumen 6

4. Modelo propuesto **1**

4.1. Modelado del problema 1
4.1.1. Entradas del sistema 1
4.1.2. Salida de datos 2
4.2. Procesamiento de información 3
4.2.1. Características de entrada 3

4.2.2. Características grafo 4
4.3. Modelo aprendizaje 4
4.3.1. Red reuronal de estimación de tráfico 4
4.3.2. GNN para propagación espacial de datos 5
4.3.3. Modelo de regresión 7
4.3.4. Función de pérdida 8
4.4. Bucle de entrenamiento 10
4.5. Optimización consumo memoria GPU 10
4.6. Resumen 11

5. Arquitectura y diseño **1**
5.1. Definición de clases 1
5.1.1. Núcleo 1
5.1.2. Red 2
5.1.3. Datos 3
5.1.4. Modelo 5
5.1.5. Constantes definidas 7
5.2. Visión global del sistema 9
5.3. Interfaz de consulta 9
5.3.1. /traffic/point 10
5.3.2. /traffic/info 11
5.4. Resumen 12

6. Resultados **1**
6.1. Juego de datos de tráfico de Madrid 1
6.1.1. Especificación de pruebas 2
6.1.2. Resultados del modelo 2
6.2. Juego de datos PEMS-BAY 12
6.2.1. Detalles del dataset 13
6.2.2. Resultados del modelo 15
6.3. Resumen 18

7. Desarrollo **1**
7.1. Resumen tiempo empleado 1
7.2. Evolución prototipado 2
7.3. Estructura de directorios 3
7.4. Tecnologías y productos empleados 4
7.5. Resumen 5

8. Conclusiones y trabajos futuros **1**
8.1. Conclusiones . . . 1
8.2. Trabajos futuros . . . 2
8.2.1. Nuevas características de entrada . . . 2
8.2.2. Alteraciones red de carreteras . . . 2
8.2.3. Mejoras en procesamiento de puntos de detección . . . 3
8.2.4. Importancia de puntos de medida . . . 3
8.2.5. Cálculo de rutas óptimas . . . 3
8.2.6. Aplicación sobre *benchmarks* del sector . . . 4
8.3. Contribuciones . . . 4
8.3.1. Componentes GNN desarrollados . . . 4
8.3.2. Histórico de tráfico . . . 8
8.4. Resumen . . . 8

Bibliografía **10**

A. Glosario de términos **15**

Índice de figuras

3.1. Arcos con mediciones asignadas 4
3.2. Mediciones por muestra febrero 2022 5

4.1. Estructura global de la red neuronal 4
4.2. Arquitectura Deep Graph Library 5
4.3. Estructura GNN desarrollada 6
4.4. Estructura redes neuronales internas 7
4.5. Flujo bucle entrenamiento 11

5.1. Flujo procesamiento juego de datos 4
5.2. Arquitectura global del sistema 9

6.1. Boxplot error por medición datos validación 4
6.2. Boxplot error por medición datos pruebas 4
6.3. Error absoluto salida intermedia medio según número de mediciones por muestra . . 5
6.4. Error absoluto salida GNN medio según número de mediciones por muestra 6
6.5. Caso 1. Original vs. Resultado. 31/01/2022 20:30 7
6.6. Caso 1. Diferencia absoluta entre original y predicción final 8
6.7. Caso 2. Original vs. Resultado. 19/03/2022 17:30 9
6.8. Caso 2. Diferencia absoluta entre original y predicción final 10
6.9. Puntos de medición anulados en datos de entrenamiento 12
6.10. Distribución puntos de medida PEMS-BAY 13
6.11. Red PEMS-BAY simplificada y puntos de medición 14
6.12. Boxplot error por medición de datos conjunto de validación (PEMS-BAY) 16
6.13. Boxplot error por medición de datos conjunto de pruebas (PEMS-BAY) 17

Índice de tablas

1.1. Desglose estimaciones por fases 4

2.1. Tabla comparativa métodos de aprendizaje basados en redes neuronales 6

5.1. Interfaz de consultas al modelo 10

6.1. Tamaño conjuntos validación y pruebas 3
6.2. Datos evaluación modelo 3
6.3. Caso 1. Distribución del error 9
6.4. Caso 2. Distribución del error 11
6.5. Datos evaluación modelo entrenamiento restringido 11
6.6. Datos evaluación sobre PEMS-BAY 16
6.7. Comparativa entre aproximaciones sobre PEMS-BAY 17

7.1. Desglose trabajo efectivo por fases 2

Capítulo 1

Introducción

El rápido desarrollo de la urbanización ha ocasionado que el transporte en las ciudades esté bajo una gran presión, especialmente cuando tiene que lidiar con grandes crecimientos de población y vehículos en circulación. Por fortuna, actualmente es posible recopilar enormes cantidades de datos del estado del tráfico que, mediante los avances en inteligencia artificial, hacen viable su tratamiento automático para realizar análisis de conducta.

La predicción de tráfico es un aspecto de gran importancia en el transporte por carretera, puesto que permite anticipar en la fase de planificación situaciones de congestión que conllevarían un mayor retraso y/o coste, mejorando a su vez la calidad de vida de los ciudadanos.

Existen dos posibles enfoques a la hora de llevar a cabo estos estudios, o dicho de forma equivalente, tratando de inferir dos valores: velocidad de la circulación o cantidad de tráfico en cada punto, pudiendo ser complementarios ambos.

En los inicios de la investigación en este campo, las predicciones se basaron en la extracción de series temporales mediante el uso de métodos estadísticos ([Drew, 1968], [Han & Moutarde, 2012]). Sin embargo, la mayoría de estos métodos aceptan la asunción de la existencia de estacionaridad en los comportamientos. Estos procedimientos, además de necesitar de expertos para su aplicación, no cuentan con suficientes parámetros para ajustar los datos a la perfección. El aprendizaje profundo puede tratar de paliar estos problemas y presentar unas predicciones mucho más próximas a la realidad.

1.1. Motivación y objetivos

Esencialmente, detectar los patrones de mayor afluencia de vehículos en circulación permite planificar las rutas tratando de evitar zonas saturadas, o posponiendo el paso por ellas a segmentos temporales donde no estén altamente tensionadas y permitan una circulación más libre y, por tanto, más rápida. Una buena planificación ofrece múltiples beneficios:

- Ahorro de combustible

- Menor contaminación por emisiones
- Reducción de la duración de los trayectos
  - Coste de personal en empresas
  - Más tiempo libre y mejor conciliación en particulares
- Menor desgaste en vehículos y vías de tránsito al reducir el número de paradas y reanudaciones de marcha en caso de alta congestión
- Reducción del número de accidentes
- Posibilidad de desarrollar sistemas de navegación autónomos

El objetivo principal de este trabajo será el desarrollo de un sistema de aprendizaje automático que permita obtener una instantánea del estado del tráfico en la ciudad de Madrid en un momento concreto del futuro, mediante el análisis del histórico de tráfico. De este modo, con tan solo unos parámetros relativos a aspectos temporales, el sistema realizará una primera estimación que posteriormente propagará espacialmente usando la estructura propia de la red de carreteras de la ciudad. Esta parte es crucial, dado que no disponemos de histórico para todos los puntos de la red, sino únicamente de los que están dentro de los planes de control del Ayuntamiento; en resumen, trataremos de inferir cómo afecta a una vía que las cercanas tengan mayor o menor cantidad de vehículos, según características de la misma, instante del día o si se trata de un festivo, por ejemplo.

1.2. Alcance

El proyecto se plantea con el objetivo principal de desarrollar un sistema de aprendizaje lo más preciso posible, que permita explotar las características inherentes de las redes neuronales aplicadas a grafos y explorar las ventajas que ofrecen a la hora de tratar de solventar este tipo de problemas.

Para consultar los resultados del modelo, el usuario dispondrá de dos operaciones:

- Proporcionar un punto en el espacio y el tiempo, consultando el estado que tendría según el modelo la vía de circulación más cercana en ese instante
- Mediante un punto en el tiempo, poder consultar el estado de todas las carreteras contempladas en el estudio

El sistema estará abierto al desarrollo de nuevas operaciones de cara al usuario, o a la inclusión de nuevas características de entrada, ya sea en la parte inicial de la red o como características físicas de la red de carreteras. Se pretende que la interacción con otros sistemas se realice a través de una interfaz API REST expuesta y permitir de este modo la automatización en la extracción de resultados.

1.3. Planificación

En este apartado se expondrá la estructura planificada para la realización de este trabajo, así como una estimación aproximada del tiempo necesario para completar cada una de las secciones.

1. Búsqueda de literatura. Consta a su vez de dos subfases:

   a) En una primera, se realizará un estudio somero sobre la literatura que afecte de forma directa o tangencial al tema a tratar, tratando de extraer las ideas principales y/o puntos débiles de cada una.

   b) De forma posterior, se profundizará en aquella documentación que trate de resolver un problema equivalente o similar al propuesto, fase de la que debe salir un estudio concreto del estado de la cuestión que permita tomar una decisión en cuanto a qué vía seguir, o aspectos básicos sobre el sistema a proponer para su construcción.

2. Prototipo básico. Se construirá la base de un nuevo modelo, de forma simplificada, tratando de confirmar su viabilidad tanto en resultados como en complejidad computacional. Para ello, es necesario estudiar en profundidad las siguientes herramientas:

   a) DGL [Wang et al.,]

   b) MXNet [Chen et al.,]

   c) Open Street Maps [OpenStreetMap Contributors,]

3. Prototipado iterativo. Se refinará progresivamente el sistema partiendo del prototipo básico, a través de mejoras y sucesivas refactorizaciones que permitan añadir más características al modelo, así como obtener unas predicciones más cercanas a la realidad producida en los casos reservados para validación.

4. Pruebas y ajuste de hiperparámetros.

5. Análisis final del sistema donde se evaluará hasta qué punto el sistema cumple con su cometido, y se extraerán los resultados obtenidos con la aproximación. Se tratará de comparar diferentes enfoques entre sí para cuantificar si aporta una mejora razonablemente importante.

6. Redacción de la memoria con toda la información necesaria para la comprensión del proyecto, facilitando a su vez futuras extensiones del mismo.

La tabla 1.1 muestra el listado de tareas de alto nivel planificadas, así como la estimación de trabajo medida en horas que se prevé ocupe cada una.

Código	Fase	Horas estimadas
1.a	Estudio preliminar	30
1.b	Estudio profundo	50
2.a	DGL	30
2.b	MXNet	15
2.c	OSM	10
3	Prototipado	80
4	Pruebas y ajustes	40
5	Análisis final	15
6	Redacción memoria	30
Total		**300**

Tabla 1.1: Desglose estimaciones por fases

1.4. Estructura de la memoria

Tras un primer capítulo de introducción al problema y las ventajas que potencialmente pueden derivarse de su utilización, en el capítulo 2 se estudia la literatura y aproximaciones ya implementadas para tratar la resolución del problema planteado. Posteriormente, en el capítulo 3 se llevará a cabo un análisis de la información disponible en las fuentes de datos de origen de las que se hará uso y en el capítulo 4 se definirá el modelo propuesto. Una vez descrito el sistema a alto nivel y de forma detallada en el capítulo 5, el capítulo 6 mostrará algunos de los resultados obtenidos, tanto durante el proceso de aprendizaje mostrando su evolución como en casos reales del histórico de tráfico. El capítulo 7 contiene un análisis del trabajo realizado, tratando de enlazar con la planificación planteada. Por último, se extraen las conclusiones finales obtenidas en el capítulo 8 y se establecen vías futuras para mejorar y complementar el trabajo desarrollado.

1.5. Código fuente

Todos los ficheros que integran el código fuente de la aplicación están publicados en la siguiente dirección `https://github.com/lmcorreas/gnn-traffic-predictor`. Se trata de un repositorio de acceso público en una plataforma elegida por su facilidad de uso y gran integración con la mayoría de sistemas, así como por su amplia comunidad, como es *github*.

1.6. Resumen

Finalizamos este capítulo de introducción donde se ha abordado la exposición del problema de predicción de tráfico que ha sido planteado, y cómo impactaría de forma positiva en aspectos como el medio ambiente, salud mental y reducción de costes y accidentes. Para lograrlo, se propone un trabajo cuyo objetivo es lograr una precisión máxima en la estimación de la ocupación que tendrá cada vía de circulación en un instante concreto, así como permitir su consulta de forma externa para la explotación de los resultados en otros sistemas.

El trabajo ha sido dividido en distintas fases de adquisición de conocimientos y desarrollo, permitiendo una estimación de cada una de las tareas de alto nivel a completar. El código fuente será de libre acceso y permitirá la ampliación del trabajo realizado.

El trabajo continúa en el capítulo siguiente con un estudio detallado del estado de la cuestión, evaluando las posibilidades tanto técnicas como conceptuales para llevarlo a cabo.

Capítulo 2

Estado de la cuestión

La predicción espacio-temporal es una tarea que opera en un entorno dinámico y, como tal, requiere mecanismos que logren capturar unos patrones cambiantes tanto en el tiempo como en el espacio. La evolución del tráfico dentro de una red de carreteras sería un caso que ejemplificaría esta idea.

Una predicción precisa puede estar llena de ventajas, al poder servir de ayuda a los ciudadanos a no caer en un camino saturado mediante el ofrecimiento de rutas alternativas, o permitir planificar un viaje con antelación evitando picos de tráfico y ahorrando, por tanto, su tiempo y su dinero. Por todos estos factores y sus propias necesidades, es un aspecto crucial en sistemas autónomos de transporte.

Pero la detección, extracción y sintetizado de las dependencias espacio-temporales conlleva una dificultad de alto nivel. Existen ciertos eventos recurrentes como pueden ser las horas puntas, que es posible hallar y reproducir con bastante fidelidad. Sin embargo, las condiciones climatológicas o los accidentes producidos son hechos prácticamente imposibles de prever cuando se trata de un instante relativamente lejano en el tiempo, y sus efectos sobre el tráfico son tanto o más importantes.

Históricamente, los sistemas se han dividido principalmente en dos grandes grupos:

- Basados en métodos estadísticos. Por ejemplo, en [Han & Moutarde, 2012] se estudia la forma de aplicar factorizaciones de matrices para preservar la localidad espacial en redes a gran escala. Con ello, se logran extraer patrones de comportamiento del tráfico que poder usar directamente o, incluso, aplicar sobre otros modelos. El aspecto más importante es que presenta una imagen global del estado de la red, sin perder el detalle y las relaciones entre puntos cercanos o conectados entre sí directamente.

- Modelos de aprendizaje automático. Están basados en la extracción de patrones a partir de datos mínimamente procesados. Las redes neuronales son un claro ejemplo de este tipo de técnicas y se estudiarán diferentes casos en las siguientes secciones.

Debido a su versatilidad y facilidad de adaptación a la resolución de diferentes problemáticas, será en este segundo grupo donde centraremos el estudio. Haciendo uso de redes neuronales podremos

acortar el tiempo de extracción de dependencias a partir del histórico de datos disponible, así como detectar relaciones que no resultan demasiado evidentes.

Otro aspecto importante es la posibilidad de incorporar características al sistema de forma casi transparente, en lugar de necesitar ajustes profundos del modelo como sí requerirían los métodos estadísticos.

2.1. Redes neuronales

Los modelos de aprendizaje profundo han probado ser eficaces capturando los patrones espacio-temporales asociados al flujo de tráfico, permitiendo realizar predicciones de una gran calidad. Desde que se comenzó a popularizar el aprendizaje automático, han sido realizados estudios con multitud de modelos diferentes basados en distintos tipos de redes neuronales:

- Redes difusas (Fuzzy Neural Network). En [Yin et al., 2002] se construye un modelo que, en primera instancia, clasifica los datos de entrada entre un número de agrupaciones según sus características usando lógica difusa. Tras ello, la red experta es entrenada para, mediante una red neuronal convencional, hallar las relaciones entre entrada y salida. Los autores mencionan que uno de los factores más importantes es su poder de adaptación a entornos cambiantes como las condiciones de tráfico en tiempo real.

- Redes recurrentes (Recurrent Neural Network). Tratando de resolver los problemas existentes en redes tradicionales al procesar información de diferentes tamaños y/o composiciones, [Lint et al., 2002] presenta un compacto modelo basado en capas recurrentes que logra adaptarse sin reingenierías y que además ofrece unos resultados interesantes, aunque con un juego de datos sintético y muy concreto, lo que lo hace difícilmente extrapolable a otras aplicaciones.

- Redes convolucionales (Convolutional Neural Network). [Ma et al., 2017] presenta un modelo basado en el aprendizaje del tráfico a través de imágenes. En él, se aplica una red convolucional que abstrae la extracción de características para, una vez generada una representación, tratar de inferir el estado del tráfico mediante numerosos mecanismos de aprendizaje con el fin de compararlos entre sí. Existen multitud de estudios sobre este enfoque [Ma et al., 2017, Zhang et al., 2016, Yu et al., 2017]. Sin embargo, esta vía no ofrece resultados demasiado interesantes, dado que su aplicación es más adecuada a problemas donde se represente o trate un espacio euclidiano y donde la información se pueda representar como una matriz de dos dimensiones. El caso del tráfico no cumple esas premisas, puesto que un par de nodos pueden presentar una distancia euclidiana muy baja, pero estar conectados físicamente a través de una distancia mucho más grande.

- Redes de creencia profunda (Deep Belief Networks). [Huang et al., 2014] usa un arquitectura con una primera fase donde la DBN realiza el aprendizaje de características de forma no

supervisada.

- Autocodificadores. En [Lv et al., 2015] se presenta un autocodificador apilado profundo para aprender características genéricas del flujo de tráfico.

- Redes generativas antagónicas (Generative Adversarial Network). La idea de usar este tipo de modelos, como en [Lin et al., 2019], es interesante al ofrecer buenos resultados en otros campos. Básicamente, consta de un generador que crea estados sintéticos del tráfico y de un discriminador que trata de diferenciarlos de los estados reales. Ambas partes aprenden de forma autónoma y es posible su aplicación con un juego de datos relativamente pequeño. El objetivo del sistema consiste en que el generador logre producir cada vez estados más cercanos a la realidad, y que el discriminador sea capaz de detectar los factores que diferencian a los estados sintéticos de los reales.

Todas estas redes pueden ser combinadas entre sí para la generación de modelos más complejos que pueden llegar a mejorar las predicciones, aunque usualmente a costa de un mayor esfuerzo computacional. Se suele buscar un equilibrio entre estos dos factores.

2.2. Redes neuronales basadas en grafos

Las redes basadas en estructuras representables como un grafo, las GNN (Graph Neural Network), calculan una representación vectorial para cada nodo en el grafo, agregando iterativamente las características de los nodos vecinos. De esta forma, logra capturar las relaciones espaciales entre los distintos nodos de la red en función de si tienen una conexión directa, o la distancia existente entre ellos en número de saltos o en peso total de los arcos es relativamente baja.

Como el resto de redes neuronales, mediante el uso de una función de pérdida diferenciable y parametrizando cómo se realiza la mencionada agregación (puesto que un nodo puede tener varios nodos vecinos), extrae características de alto nivel que permiten la inferencia del estado del grafo tanto en su conjunto como a nivel local. Dicho de otro modo, la GNN representará con su estado una instancia del problema, el estado de un algoritmo iterativo, o ambos.

Son aplicables a multitud de campos de interés, por ejemplo, ciencia social (redes sociales), ciencias naturales (redes de interacción entre proteínas), o grafos de conocimiento. Todos ellos se pueden resumir en varias aproximaciones: clasificación de grafos, inferencia de nodos/arcos dentro de una red existente, o la regresión de características en nodos/arcos.

En [Zhou et al., 2018] se realiza un interesante análisis de las opciones y aplicabilidad de este tipo de redes. Establece clasificaciones según los grafos sean dirigidos o no dirigidos, homogéneos o heterogéneos y estáticos o dinámicos. Además, la función de pérdida se puede diseñar sobre distintos niveles:

- Nodo. Regresión o clasificación de nodos, así como el particionado de los nodos de un grafo en diferentes grupos.
- Arco. Incluye tareas de clasificación o predicción de arcos no existentes.
- Grafo. Trataría problemas de clasificación o regresión de grafos.

Observando desde la perspectiva de la supervisión, permiten llevar a cabo tanto aprendizajes semi y supervisados, así como no supervisados para la extracción de características.

2.3. Aprendizaje del tráfico usando GNN

Existen multitud de problemas que pueden modelarse con el objetivo de aprender patrones sobre el estado de un grafo. Sin embargo, el caso del tráfico se diferencia del resto en que no existen nodos/arcos aislados y la estructura de la red permanece constante puesto que raramente cambia. La densidad del tráfico en cada carretera es a su vez cambiante en el tiempo, lo que incluye una dimensión adicional a los datos.

Otro hecho diferencial es que las carreteras tienen características físicas que participan en el comportamiento del flujo de vehículos, como pueden ser la longitud, el tipo de vía, el límite de velocidad o el número de carriles.

Ser capaz de representar este problema mediante el uso de una GNN, permite obtener gran parte de sus beneficios: escalan linealmente con el número de características y la cantidad de nodos y arcos en la red. Cabe destacar en este punto que los sensores de tráfico existentes en las carreteras contienen correlaciones espaciales complejas. Debido a esto, la cercanía entre ellos en el espacio euclidiano conlleva un comportamiento que puede ser completamente opuesto en función de otros parámetros como pueden ser el destino o el tipo de vía en la que se encuentran. Como conclusión, podemos extraer que la estructura espacial del tráfico no es tanto euclidiana como direccional, tal y como se mencionó en el tratamiento del problema mediante redes convolucionales.

En [Li et al., 2017], se expone un problema relativamente simplificado en cuanto a su estructura espacial, con entre 200 y 325 sensores de dos conocidos juegos de datos en este ámbito (*METR-LA* y *PEMS-BAY*), cuya conectividad espacial es prácticamente lineal. Trata de resolver el problema mediante una red bautizada como DCRNN (Diffusion Convolutional Recurrent Neural Network), que a grandes rasgos es una arquitectura codificador-decodificador que incluye varias capas recurrentes en ambas fases.

[Cappart et al., 2021] realiza un análisis interesante de empleo de modelos de optimización combinatoria donde se define cómo adaptar un problema basado en algoritmos clásicos al uso de GNN para eliminar dependencias del preprocesado de datos realizado por humanos pasando a flujos de procesamiento automático de información que toman los datos en crudo y son modificados hasta que llegan al elemento que aprende de ellos (aproximación end-to-end). Otro enfoque interesante que menciona

el artículo sería el aprendizaje mediante una arquitectura codificador-procesador-decodificador, donde la etapa intermedia consistiría en una GNN entrenada para aplicar un algorimo sobre varias entradas abstractas. El primer paso tomaría los datos en crudo y generaría un *embedding* que trataría la parte de procesamiento. Ésta generaría una salida de la misma dimensión, que sería decodifida. Con esta arquitectura se lograría aproximar la función A incluyendo un procesamiento intermedio [g(P(f(x))) $\sim$ A(x)], permitiendo una aproximación a soluciones dentro de un problema NP-duro generado en tiempos polinomiales, sin la contrapartida de tener que sintetizar una entrada multidimensional en una serie de valores finitos.

Para capturar las relaciones espaciales y temporales dinámicas de los datos de tráfico, se han usado GNN con múltiples capas para la parte espacial y capas recurrentes para la temporal. Sin embargo, no suelen ofrecer un buen resultado dado que la propia estructura puede presentar diferencias en el número de saltos entre cruces, algo que hace difícil extraer la información (arcos de longitudes diferentes, proporción de cruces o número de saltos entre un punto y otro es variable). En [Roy et al., 2021] los autores proponen una arquitectura bautizada como SST-GGN (Simplified Spatio-temporal Traffic forecasting GNN, que sea capaz de codificar la dependencia espacial agregando diferentes representaciones de vecindad en lugar de apilar capas para recoger las relaciones espaciales. Uno de los principales problemas sería que dentro de una red, no todos los vecinos de un nodo concreto tienen el mismo impacto. Por ello, alimentar capas completamente conectadas con las representaciones de vecindades de diferentes grados hace que sean más efectivas. Con el fin de mejorar las predicciones, usa dos modelos diferentes: uno del mismo día y otro con el histórico; y para la recogida de las relaciones temporales usa una suma ponderada de los estados recientes. Su análisis se enfoca en predicciones a 15, 30, 45 y 60 minutos, tratando de comparar resultados con los modelos diseñados hasta el momento. Otro enfoque similar se describe en [Wu et al., 2019], donde se propone un apilación de niveles espacio-temporales donde tras una capa lineal se llevan a cabo las convoluciones necesarias. Un aspecto a destacar es que existe una conexión desde cada nivel a la capa de salida, lo que permite calcular la importancia de cada una en la predicción final.

Las capas recurrentes propuestas en otros trabajos destacan por su baja eficiencia, la cadena de propagación es muy larga y, por tanto, los cálculos muy complejos. [Li et al., 2021] propone un mecanismo por el cual, mediante un marco llamado DGCRN (Dynamic Graph Convolutional Recurrent Network), se generan hiper-redes en cada paso del entrenamiento mediante la extracción de características de los atributos de los nodos. Una vez extraídas estas representaciones, se usan para generar grafos dinámicos. Tras ello, mediante la integración con el grafo base estático, se realiza el aprendizaje de los parámetros necesarios. DGCRN sigue una arquitectura secuencia-secuencia, incluyendo un codificador y un decodificador. El modelo consiste en dos partes: generador de grafos para modelar las características dinámicas de la topología de red dinámica, y el módulo recurrente convolucional.

En [Cui et al., 2018] se propone una convolución diseñada a tal efecto que, además de realizar buenas predicciones, permite identificar los segmentos de vía más influyentes en redes de tráfico reales.

Para ello basa los cálculos en matrices de adyacencia y el concepto de 'velocidad de libre circulación', consiguiendo una nueva matriz que extrae características adicionales del grafo. El análisis de la capa de convolución propuesta permite interpretar la solución, y extraer de ella cuáles son las carreteras más influyentes.

Otro posible enfoque para dar solución al problema se muestra en [Zheng et al., 2019], donde los autores proponen la creación de una arquitectura codificador-decodificador donde un embedding espacio-temporal codifica los vértices en vectores que preserven la estructura de grafo de la red de circulación. Dado que esta es una representación estática que no puede almacenar las correlaciones dinámicas entre los sensores de tráfico de la red, resulta necesario el entrenamiento de otro embedding temporal para codificar cada instante en un vector y permitir el modelado de correlaciones no lineales entre diferentes instantes. Las predicciones finales las ofrecerá una fusión entre ambas partes.

2.4. Comparativa

Dado el alto número de posibles enfoques a la resolución del problema de predicción de tráfico, mostramos en la tabla 2.1 un pequeño resumen comparativo de las mismas, usando una codificación para indicar un nivel del 1 al 5 ($\downarrow\downarrow$ - muy bajo, $\downarrow$ - bajo, $\rightarrow$ - medio, $\uparrow$- alto, $\uparrow\uparrow$ - muy alto):

Método	**Adaptabilidad**	**Complejidad**	**Datos necesarios**	**Valoración general**
Fuzzy	$\downarrow$	$\rightarrow$	$\uparrow$	$\downarrow$
RNN	$\uparrow$	$\uparrow\uparrow$	$\rightarrow$	$\rightarrow$
CNN	$\downarrow$	$\rightarrow$	$\uparrow$	$\rightarrow$
DBN	$\downarrow$	$\downarrow$	$\downarrow$	$\rightarrow$
Autoencoder	$\rightarrow$	$\downarrow$	$\downarrow\downarrow$	$\rightarrow$
GAN	$\rightarrow$	$\rightarrow$	$\downarrow$	$\rightarrow$
DCRNN	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
SST-GGN	$\uparrow\uparrow$	$\rightarrow$	$\rightarrow$	$\uparrow$
DGCRN	$\uparrow$	$\uparrow$	$\uparrow$	$\rightarrow$
TGC-LSTM	$\rightarrow$	$\uparrow$	$\uparrow$	$\rightarrow$
GMAN	$\rightarrow$	$\rightarrow$	$\uparrow$	$\rightarrow$
Graph WaveNet	$\uparrow$	$\rightarrow$	$\downarrow$	$\uparrow$

Tabla 2.1: Tabla comparativa métodos de aprendizaje basados en redes neuronales

2.5. Resumen

En este capítulo se han mostrado los dos grandes grupos en los que se centran las metodologías que históricamente han sido usadas para resolver el problema planteado, con especial atención a diferentes estudios que presentan aproximaciones mediante estructuras basadas en redes neuronales, como las convolucionales o las recurrentes.

Tras esta introducción, entramos en el análisis del enfoque en el que se centrará el estudio, definiendo una red neuronal aplicada a grafos, así como las ventajas de usar este tipo de estructuras en problemas donde la información está estructurada o se puede modelar como un conjunto de nodos y arcos.

Finalizamos con una tabla comparativa que presenta una evaluación de los modelos presentados y su aplicabilidad al tratamiento del problema.

El siguiente paso será un análisis detallado de la información disponible y que será necesaria para lograr el aprendizaje por parte del sistema, así como cuál es la relación existente entre las diferentes fuentes de datos.

Capítulo 3

Análisis de información

El trabajo ha sido centrado en la ciudad de Madrid, puesto que la red de carreteras tiene un tamaño interesante y el portal abierto de datos del Ayuntamiento ofrece información suficiente como para llevar a cabo un estudio serio y completo con el que poder extraer conclusiones del funcionamiento del modelo propuesto. Las siguientes secciones mostrarán un análisis de los diferentes orígenes de datos empleados.

3.1. Datos espaciales

En primer lugar, debemos contar con una red lo suficientemente completa y detallada de las vías de circulación existentes en la ciudad. Para ello, haremos uso de OSMnx [Boeing, 2017], una librería que apoyándose en OpenStreetMap carga una red y la modela como un grafo. En esta estructura, los nodos representan los cruces entre diferentes vías y los arcos serían las propias carreteras.

La red cargada por OSMnx es completamente fiel a la realidad, por lo que el grafo es de un tamaño relativamente grande. Dado que cada cruce de vías genera un nodo diferente, construcciones como por ejemplo las glorietas pueden llegar a representarse por decenas de elementos. Con el fin de tratar de paliar el efecto que esto supondría a nivel de complejidad computacional, contamos con un mecanismo que OSMnx incluye para poder fusionar en uno solo todos los elementos cuya distancia sea inferior a un umbral determinado, reduciendo notablemente una complejidad que, para este problema en cuestión, aporta un valor testimonial.

Con el fin de mejorar la eficiencia del sistema ante repetidas ejecuciones, una vez procesada la información para adecuarla al problema es persistida a un fichero GraphML(A).

3.2. Ubicación puntos de medida

El Ayuntamiento de Madrid publica a través de su portal de datos abiertos un listado actualizado mensualmente que contiene la localización de los puntos de medición de densidad de tráfico

[Ayuntamiento de Madrid, c]. Esta información resulta elemental para nuestros intereses, puesto que es la única forma de relacionar los datos de histórico de tráfico con los espaciales.

Se trata de un fichero en formato texto separado por comas (CSV) donde se especifica, para cada punto de medición, la siguiente información:

- tipo_elem: Tiene dos posibles valores (M30 o URB).
- distrito: Identificador numérico del distrito donde se encuentra.
- id: Identificador numérico único del punto de medición.
- cod_cent: Identificador alfanumérico del punto.
- nombre: Nombre del punto de medida, siendo en la mayoría de los casos una expresión que denota la vía y sentido donde se encuentra.
- utm_x: Coordenada x del punto en UTM.
- utm_y: Coordenada y del punto en UTM.
- longitud: Longitud del punto de expresada en grados.
- latitud: Latitud del punto de expresada en grados.

Existe información no relevante para el objetivo de este trabajo, por lo que no serán tratados los campos nombre, distrito y tipo_elem.

3.3. Histórico de tráfico

El Ayuntamiento de Madrid pone a disposición del público en general en su portal de datos abiertos una serie de ficheros con el histórico del tráfico en todos los lugares donde ha instalado puntos de medición [Ayuntamiento de Madrid, b]. Publica un documento mensual en formato texto separado por comas (CSV) con información recogida con una periodicidad de 15 minutos. El documento consta de los siguientes campos para cada medición:

- id: Identificación única del punto de medida en los sistemas de control del tráfico del Ayuntamiento de Madrid.
- fecha: Fecha y hora oficiales de Madrid del instante donde se tomó la medición. Se usa formato yyyy-mm-dd hh:mi:ss.
- tipo_elem: Indica el tipo de vía según si pertenece a la circunvalación o es vía urbana. Tiene dos posibles valores (M30 o URB).

- intensidad: Intensidad del punto de medida en el periodo de 15 minutos (vehículos/hora). Un valor negativo implica la ausencia de datos.
- ocupacion: Tiempo de ocupación del punto de medida en el periodo de 15 minutos (%).
- carga: Carga de vehículos en el periodo de 15 minutos. Este valor es un porcentaje para representar grado de uso de la vía y es calculado por el sistema teniendo en cuenta intensidad, ocupación y capacidad de la vía.
- vmed: Velocidad media en km/h de los vehículos. Solo está indicada para puntos de medida interurbanos M30.
- error: Mediante un caracter representa si ha habido alguna muestra errónea en el periodo al cual hace referencia.
- periodo_integracion: Número de muestras recibidas y consideradas para el periodo de integración.

Lamentablemente no todos los campos son útiles, ya sea por estar indicado un valor válido solo para ciertos puntos de medida como en el caso de la velocidad media o por tener información irrelevante para nuestro propósito. Por tanto, en un primer preprocesado de la información procedemos a eliminar las tres últimas y a truncar el campo fecha para deshacernos de los segundos, puesto que es ':00' en todas las líneas recibidas, reduciendo en parte el espacio ocupado por el histórico. Aprovechamos a su vez a filtrar por los puntos de medición que se vayan a tener en consideración y los registros indicados como recogidos satisfactoriamente.

3.3.1. Arcos con mediciones

En el histórico publicado, no todos los arcos tienen un medidor instalado. Con la información tratada como conjunto de datos, disponemos de datos relativos a 1183 medidores sobre un total de 3021 arcos, aunque eso no signifique que cada muestra conste de tantas mediciones. Para ejemplificarlo gráficamente, en la imagen 3.1 se pueden observar los arcos en verde, que tienen alguna medición en el conjunto de datos y los que están en rojo, de los que no se dispone de dato alguno; deberá ser la GNN la que ayude a completar las estimaciones.

3.3.2. Incompletitud de muestras

Se observan numerosas inconsistencias en el juego de datos, pues prácticamente no existen instancias con valores de ocupación para todos los puntos de medición, estando por tanto incompletas y limitando de esta forma su utilidad. Usando un ejemplo concreto, el punto de medida con identificador 10265 (M-30, localizado en [40.4442,-3.6607]) solo se encuentra en 717 muestras en el mes

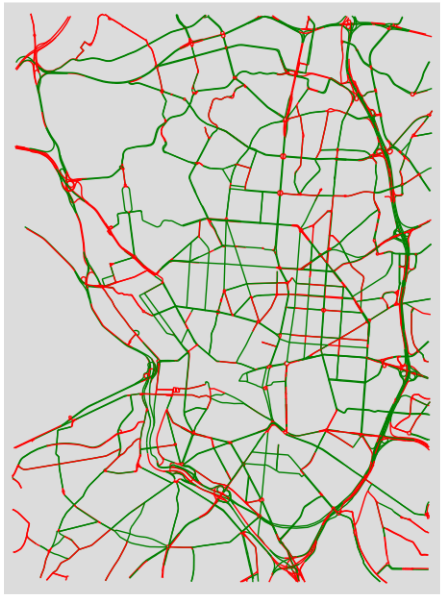

Figura 3.1: Arcos con mediciones asignadas

de febrero de 2022, siendo 2688 el total de instancias del mes. Por contra, el punto de medición con identificador 1001 (Jose Ortega y Gasset E-O - Pº Castellana-Serrano, localizado en [40.4305, -3.6883]) sí que aporta información en la totalidad de instancias.

La imagen 3.2 muestra el número de mediciones disponibles por instancia del conjunto de datos para los puntos de medida contemplados, un total de 1184, haciendo patente estas inconsistencias.

Se observan 28 descensos bruscos, más o menos agudos, correspondientes a los 28 días del mes, lo que hace evidentes intensas parcialidades de las muestras incluso dentro del mismo día. Dado que cada muestra está compuesta por un número concreto de valores, cada uno asociado a un punto de medición, trataremos cada valor ausente como un punto sin medida, por lo que no tendrá influencia sobre el cálculo. Esta opción parece la más sensata teniendo en cuenta el número de ausencias en mediciones, porque no resulta posible descartar todas las muestras incompletas al encontrarse en esta situación prácticamente la totalidad de las instancias del juego de datos. Sin embargo, este factor puede afectar de forma notable al cálculo, haciéndolo menos efectivo para ciertas partes de la red o segmentos temporales.

3.4. Tipología del día y festividades

Los patrones de tráfico no son uniformes a lo largo de todos los días, sino que existen variaciones bien diferenciadas según la naturaleza del mismo. Por ejemplo, los fines de semana presentan una

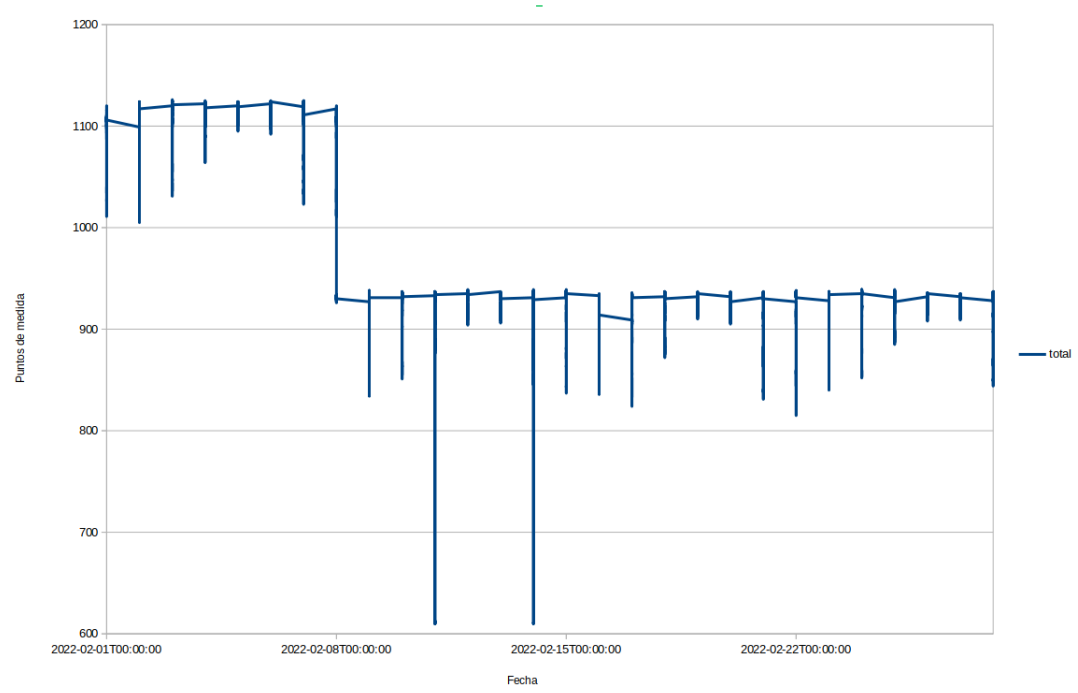

Figura 3.2: Mediciones por muestra febrero 2022

reducción notable del número de traslados hacia o desde los lugares de trabajo, debido a que la semana laboral de muchas personas comprende los días de lunes a viernes. Asimismo, se generan otras grandes afluencias cuyo horario y localización es diferente, como podría ser la saturación de las salidas de la ciudad en viernes por la tarde y sábado por la mañana, o de los accesos a ella el domingo por la tarde.

De igual modo, los días festivos generan un efecto similar, pero que también está relacionado con el día de la semana en el que se produce. Si hablamos de un festivo en lunes, los patrones de desplazamiento del domingo tienden a retrasarse un día. La misma circunstancia ocurre de forma inversa si el festivo se produce en viernes, aunque con ciertos matices derivados del adelanto del fin de la jornada laboral en viernes que no se observa de una forma tan intensa si es el jueves el último día laborable de la semana.

Debido a que puede ser información valiosa, además de marcar cada instancia del juego de datos con el día de la semana en la que se produjo, realizamos la lectura del fichero de festividades que aporta también el portal de datos abiertos del ayuntamiento [Ayuntamiento de Madrid, a]. En él, se observan los siguientes campos:

- Dia: Fecha a la que se refiere el registro.
- Dia_semana: Día de la semana expresado literalmente con la palabra en castellano.
- laborable / festivo / domingo festivo: Valor entre 'vacío', 'laborable', 'sábado', 'domingo' o 'festivo'.

- Tipo de Festivo: Especifica el ámbito al cual se aplica el festivo (nacional, autonómico o local).
- Festividad: Descripción breve de la festividad que se aplica ese día.

Los valores de los campos no son consistentes a lo largo de todo el documento, estando los más recientes más carentes de información. Por tanto, el tratamiento ha requerido de una normalización para extraer esa información que por los primeros registros es redundante pero facilita el proceso.

3.5. Resumen

Finaliza el capítulo dedicado a la descripción de las diferentes fuentes de datos y cómo su información está conectada entre sí por cercanía, como es el caso de la red y los puntos de medición; por fecha, como en las festividades y el histórico de tráfico, o por identificadores numéricos como ocurre con los puntos de medida y el histórico.

Esta información es rica en variedad, volumen y, aunque carece de consistencia entre las diferentes muestras en cuanto a número de mediciones por unidad, parece lo suficientemente buena como para lograr un aprendizaje correcto de los patrones buscados.

En el siguiente capítulo avanzaremos la estructura del modelo propuesto para lograr este fin.

Capítulo 4

Modelo propuesto

4.1. Modelado del problema

El objetivo de este apartado es tratar de dar una idea en profundidad de cómo se diseñará la red neuronal, sus entradas y salidas, así como el mecanismo para calcular cuándo una solución aportada es mejor que otra mediante una función de pérdida que permita propagar gradientes hacia atrás haciendo que aprenda de la información.

Dado que nuestro propósito es ser capaz de predecir el estado de congestión de la red en un momento determinado, deberemos poder inferir una gran cantidad de información a través de unos pocos valores, los que marcan el instante.

4.1.1. Entradas del sistema

Tal y como se definió en el capítulo anterior 3, se trata de la información disponible para alimentar al sistema, es decir, las características de cada uno de los elementos dentro del juego de datos.

Información espacial

Estos datos resultan de especial importancia, y constan de dos partes principales:

- Estructura de la red. Se trata de disponer de las conexiones entre los diferentes puntos de la red de carreteras, cuáles tienen cruces con otras, y el grado de vecindad de cada nodo, por ejemplo.

- Características de las vías. En este punto tendríamos los valores que marcan las capacidades físicas e información de las propias vías de circulación, o llevando la cuestión a terminología de grafos, los arcos que considera el sistema. Estaríamos hablando de información como la longitud, la velocidad máxima, el número de carriles con los que cuenta, etc..

En definitiva estamos considerando un grafo dirigido y aunque no consta de un peso único para cada uno de los arcos, sí que las características podrían en parte ser consideradas así. El sistema debe tener acceso a esta información, que asumiremos inmutable durante el transcurso del entrenamiento y posterior utilización del modelo entrenado.

Datos temporales

Consiste en los siguientes datos:

- Variables dummy para seleccionar el día de la semana al cual pertenece el elemento (lunes-sábado). Se trata de seis variables donde el número máximo de valores activos está en el rango [0, 1], dado que una instancia no puede pertenecer a dos días diferentes. La ausencia de activación en todas las características indicará el último valor posible (domingo).
- Selectores temporales. Conformado por cinco características diferentes relativas al año, mes, día, hora y minuto al cual pertenece la instancia en cuestión.
- Minutos desde y hasta las 0h, facilitando las tareas de continuidad de la información ante muestras consecutivas con incremento del valor de la hora.
- Flag festivo. Indica si la instancia pertenece a un día festivo, permitiendo desambiguar entre dos comportamientos diferentes como, por ejemplo, podría ser un lunes laborable con una afluencia de tráfico general importante, o uno festivo donde el nivel de tránsito será notablemente inferior o presentará patrones cuyo sentido o franjas horarias sean muy diferentes..
- Víspera o día tras festivo. El hecho de que la muestra sea relativa a un día antes o después de una festividad es una información importante y modifica notablemente los patrones de ese día.
- Variables dummy para los meses, facilitando también la continuidad ante cambios de año.

4.1.2. Salida de datos

La salida del sistema viene marcada, en gran medida, por la información disponible en el juego de datos del que haremos uso. Sería interesante que el sistema fuese capaz de inferir la velocidad media en cada uno de los arcos de la red, pero lamentablemente los valores de este tipo solo aparecen para los puntos de medición de la circunvalación M-30. Podríamos tratar de propagar espacialmente al resto de vías, pero la proporción de segmentos con valor es muy pequeña y poco variada como para que el resultado sea satisfactorio y/o interesante de cara al estudio.

Una vez descartada la velocidad, disponemos de tres valores (intensidad, ocupación y carga) tal y como especificamos en el capítulo anterior al realizar el análisis de la información. Con el fin de simplificar el problema en la medida de lo posible, se decide escoger uno de ellos que funcionará como salida del sistema. La intensidad indica el número de vehículos por hora, factor a tener muy

en cuenta para la propagación espacial, pero que nos impide realizar comparativas realistas entre diferentes salidas del sistema, puesto que no resulta sencillo estimar cuál sería la capacidad máxima de la vía en condiciones ideales.

Por otra parte, el valor de ocupación puede resultar engañoso en ciertas vías con semáforos o incluso con retenciones importantes, dado que indica tiempo de ocupación del punto de medida. Parece la medición menos útil de entre las disponibles.

En último lugar, existe la posibilidad de usar el valor de carga, un valor calculado que representa el valor de ocupación real de la vía medido en un porcentaje sobre el máximo posible. Parece ser el más interesante dado que potencialmente permite calcular cuál es la ocupación de cada punto de la red con respecto a los nodos y arcos cercanos.

4.2. Procesamiento de información

Dado que cada característica usada puede estar en una escala muy diferente, es decir, usando órdenes de magnitud distantes entre sí, se plantean modificaciones sobre los mismos de cara a mejorar el aprendizaje en la medida de lo posible.

4.2.1. Características de entrada

Las características de entrada al sistema, dada su naturaleza, están en diferentes escalas: no tiene el mismo rango de valores el año al que pertenece la muestra que el día del mes, por ejemplo. Por tanto, se procede a realizar una modificación de las mismas para dejarlas en una escala similar que facilite el procesado posterior. Debido al volumen de datos que permite manejar el sistema y que el rango de valores posibles original es conocido para todas las características contempladas, se procederá a dividir cada uno de los valores por el máximo previsto para ese campo:

- Día $/= 31$
- Mes $/= 12$
- Año $/= 2022$
- Hora $/= 23$
- Minuto $/= 59$
- Minutos desde 0:00 $/= 1440$
- Minutos hasta 0:00 $/= 1440$

El resto de características de entrada son variables dummy, por lo que su valor es 0 o 1 según estén activas o no. Debido a ello no necesitan modificación alguna.

4.2.2. Características grafo

El caso más importante dentro de las características físicas de las vías sería el de la localización física de los nodos de la red. En el caso de usar coordenadas UTM, las variaciones serán mínimas sobre el valor total del elemento, lo que conllevaría que cualquier cambio en los parámetros asociados al realizar la retropropagación de la red podría empeorar las predicciones al ser más sensible que el resto. Por ello, se decide usar los valores de longitud y latitud, que a su vez pasan por un proceso de escalado de tipo MinMax A para reducir posibles impactos negativos en el aprendizaje.

4.3. Modelo aprendizaje

El modelo contiene partes bien diferenciadas, que se encargan de procesamientos complementarios. En la imagen 4.1 se puede observar a alto nivel cómo es la estructura del modelo planteado, desde la información de entrada hasta que se genera la salida.

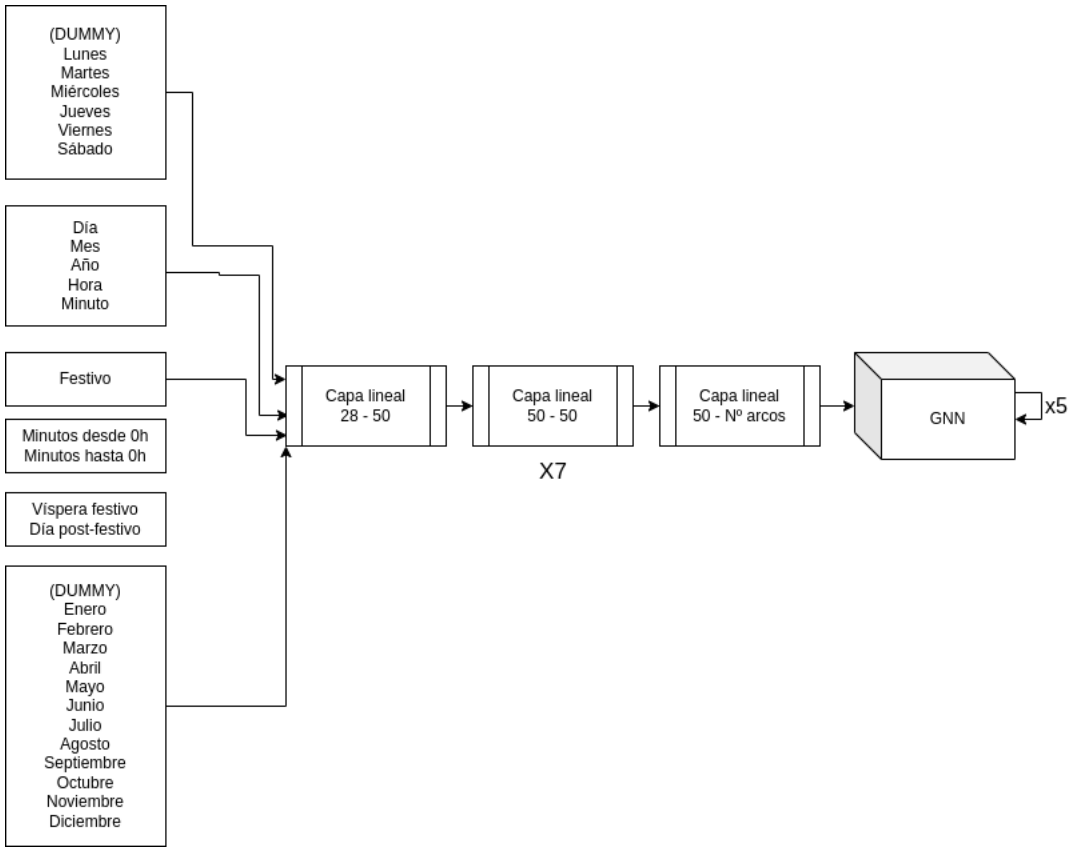

Figura 4.1: Estructura global de la red neuronal

4.3.1. Red reuronal de estimación de tráfico

Esta primera fase del modelo presenta tres partes de capas lineales del siguiente tamaño:

1. 28 x 50: Convierte los datos de entrada en características internas.

2. 50 x 50: Secuencia de 7 capas de neuronas con 50 características de entrada y de salida.

3. 50 x Nº total de arcos: Capa de salida de la primera red neuronal del sistema.

En este punto disponemos de una primera estimación de la densidad circulatoria con la que es posible hacer uso de la función de pérdida para ver cuánto se desvía de la realidad en cada arco con medidor instalado. Para el resto de arcos, únicamente es posible penalizar que el valor se sitúe fuera del rango permitido [0, 100], tratando de facilitar así su convergencia.

4.3.2. GNN para propagación espacial de datos

Para la segunda parte de la red neuronal, esto es, la que hace uso de la información espacial para la propagación de las características relativas al volumen de tráfico, se ha decidido hacer uso del framework Deep Graph Library (DGL - [Wang et al., 2019]). Este producto de código abierto nos otorga gran versatilidad, facilidad de uso y destaca por su construcción multi-backend, es decir, se adapta a diferentes productos de aprendizaje como pueden ser Pytorch y Tensorflow. En la siguiente imagen 4.2, extraída de la documentación de DGL, se puede observar la arquitectura simplificada:

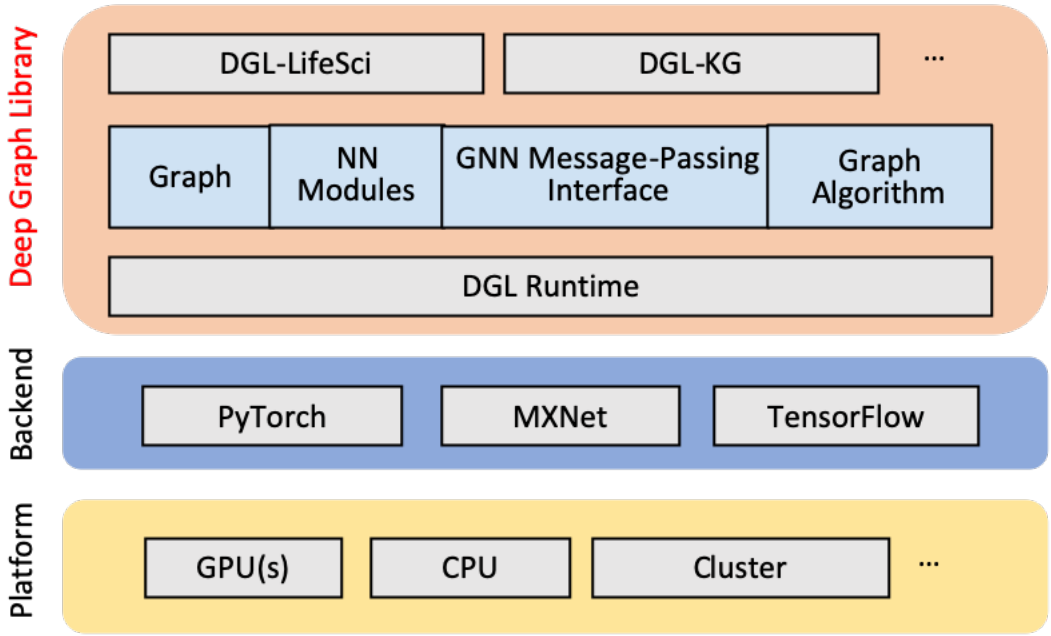

Figura 4.2: Arquitectura Deep Graph Library

Se observa cómo tiene la habilidad de hacer uso de tres backends diferentes, y éstos adaptarse a numerosas plataformas de cálculo que van desde la CPU hasta unidades de procesamiento gráfico permitiendo incluso distribuir los cálculos en clústeres de nodos.

Capa convolucional a medida

Tratando de conseguir una mayor adaptación al problema que nos ocupa, se ha desarrollado una nueva capa convolucional encargada de, mediante paso de mensajes a nodos vecinos con la información necesaria, propagar y calcular la densidad del flujo de tráfico en todos los puntos de la red sin medidor, tratando de extraer así los patrones de relación de nivel circulatorio entre nodos vecinos o, dicho de otro modo, cómo un nodo se nutre o se libera según el estado de todos los nodos con los que tiene conexión directa. Extendiendo este comportamiento hasta nodos a una distancia determinada, marcada por el número de capas convolucionales apiladas en la red de recurrencia, es posible crear buenas estimaciones aun teniendo datos disponibles para relativamente pocos puntos de la red.

La capa diseñada está compuesta por dos procesos de paso de mensajes entre nodos y arcos y viceversa para propagar información, así como de tres pequeñas redes neuronales que ayudan a aprender los patrones existentes. El diseño general, así como las características usadas por cada parte de la capa, es el de la imagen 4.3.

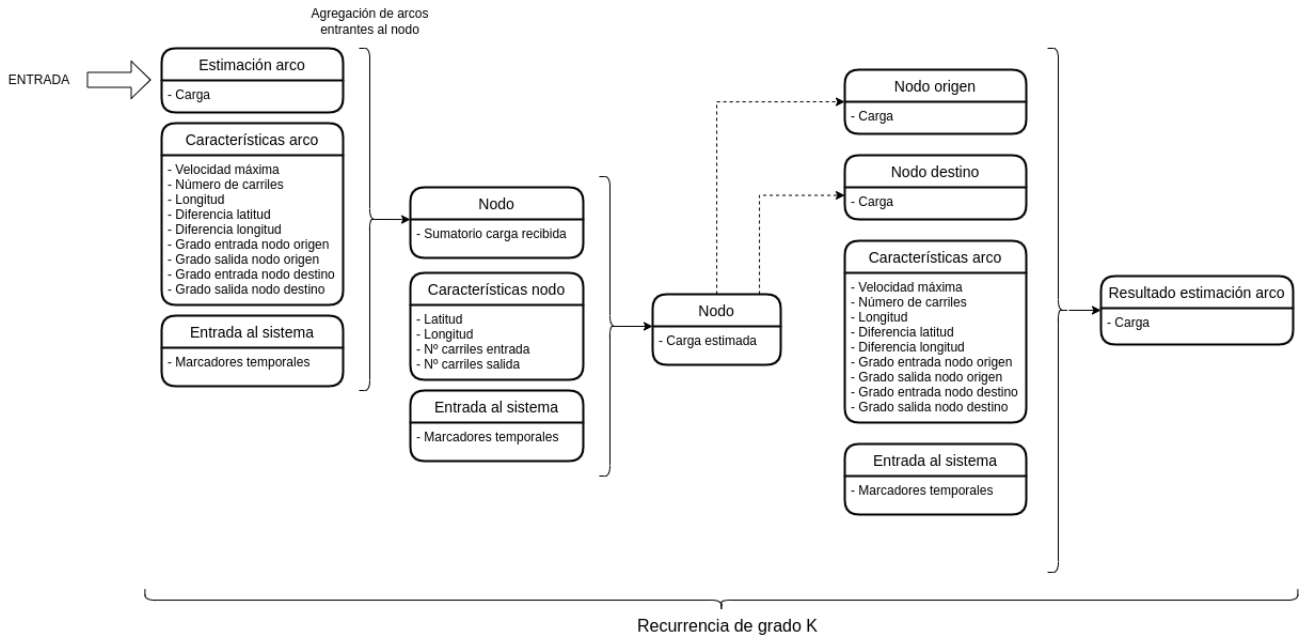

Figura 4.3: Estructura GNN desarrollada

Tal y como se puede observar, consta de tres fases principales:

1. En una primera, se toma para cada arco la estimación ofrecida por la red neuronal inicial, así como las características del propio arco y las de entrada al sistema inicial, las que marcan el instante temporal para el cual se está realizando la estimación. Como salida de esta parte se obtiene para cada arco, una estimación de la carga que aporta al nodo donde termina.

2. En segundo lugar, se realiza un sumatorio de los valores anteriores, donde cada nodo terminará con la suma de los arcos que finalizan en él. De esta forma, se tiene una estimación de la suma

de cargas estimada que le ofrecen sus arcos. Una vez se dispone de este valor calculado, se alimenta una nueva red neuronal junto con las características propias del nodo y la entrada inicial al sistema, lo que permite una estimación real de la carga del nodo.

3. Por último, se realiza un cálculo a nivel de arco. Se toman las estimaciones de carga de sus nodos origen y destino y, junto a sus propias características físicas y la entrada inicial al sistema, alimentan a una última red neuronal que ofrece como salida una estimación de carga para cada arco del sistema.

Redes neuronales internas

Las tres redes enumeradas en la sección anterior tienen una estructura de varias capas, siguiendo el concepto que se expone en la imagen 4.4.

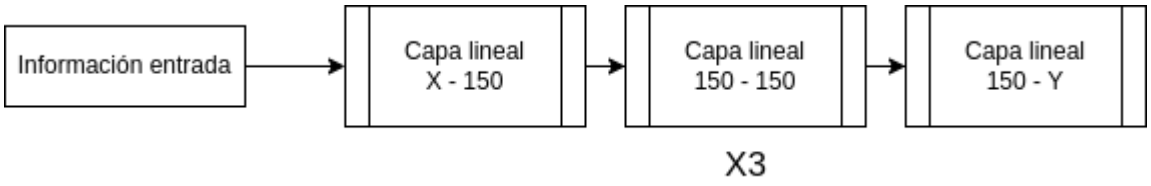

Figura 4.4: Estructura redes neuronales internas

Dado que las tres tienen una estructura de la información de entrada diferente, la forma de la primera capa no es fija y se calcula en base a los parámetros de configuración del problema. Su salida sí es común, y consiste en tantos valores como indique el parámetro de etiquetas de salida, existente en las definiciones del sistema.

4.3.3. Modelo de regresión

Una vez definidas las diferentes partes de la red neuronal, debemos ser capaces de aplicar un criterio claro sobre la consideración de una solución como buena y permitir el cálculo de gradientes para lograr el aprendizaje. Para ello, la aplicación de la función de pérdida puede no ser suficiente sobre los datos finales que ofrece el sistema como salida, por ser parciales y existir solo para los arcos con puntos de medida físicos.

Para paliar la ausencia de datos uniformemente distribuidos a lo largo de la red, y una vez establecida la arquitectura de la red neuronal, se opta por el cálculo de pérdida a dos niveles diferentes:

1. Tras la primera fase, que otorga tantos valores como arcos conforman la red. En este punto, disponemos de una estimación del nivel de congestión en cada arco, y ha mostrado empíricamente favorecer la convergencia a soluciones mejores en menor número de ciclos de aprendizaje. La aplicación de la función de pérdida en este punto aporta dos beneficios principales:

   a) Penalizar valores fuera de rango en los arcos carentes de medidor

   b) Penalizar desviaciones en arcos con datos reales para que la propagación espacial inicie con un valor óptimo

2. En salida de la red, donde se propone una nueva estimación ya habiendo propagado espacialmente mediante la GNN. Los puntos de medición reales son los mismos, pero la diferencia está en que se ha aplicado la recurrencia necesaria para detectar los patrones de flujo de tráfico o, en resumen, cómo afecta el estado de los nodos a los arcos con los que está conectado hasta una profundidad determinada.

3. De forma interna en la capa convolucional de la GNN, tal y como ha sido especificado en el apartado anterior.

4.3.4. Función de pérdida

Debido a que no disponemos de datos de medida en todos los arcos de la red, la función diferenciable para realizar el ajuste de la regresión no puede ser idéntica para todos los valores generados. Por ello, optamos por el desarrollo de una función *ad hoc* para este caso concreto, tratando de penalizar valores fuera del rango posible.

- En el caso de arcos que dispongan de puntos de medida y, por tanto, cuenten con valores reales, se aplicará una pérdida de tipo MSE A o error cuadrático medio. Ésta permite penalizar más aquellas muestras cuya distancia con la realidad sea mayor, debido a que se aplica el cuadrado a la diferencia entre ambas. Permite a la red converger a una solución buena si cuenta con datos consistentes y ésta es lo suficientemente parametrizable.

- Para arcos sin valores medidos no disponemos de información con la que contrastar, por lo que nos limitaremos a penalizar de forma cuadrática la distancia con respecto al límite más cercano si está fuera del rango aceptable para ella. Para facilitar la convergencia a una solución factible, incrementamos esta pérdida por un factor configurable de forma previa al entrenamiento para que la red tienda a ofrecer soluciones dentro del rango y, posteriormente, se centre en la calidad de las mismas usando todos los datos de medida disponibles.

El código de esta función se muestra en 4.1.

Penalizaciones internas

Durante algunas fases del desarrollo del trabajo, se ha observado que el hecho de concatenar redes neuronales sin control interno llevaba a valores intermedios no deseados, como por ejemplo estimaciones de carga negativas para los nodos. Debido a ello, se implementó un mecanismo por el cual es posible penalizar:

Algoritmo 4.1 Función de pérdida desarrollada

```
def forward(self, input: Tensor, target: Tensor, mask: Tensor, factor:
    float) -> Tensor:
  ret = 0.0

  if factor > 0:
    # Valores activos dentro de la máscara de existencia de datos reales
    masked = torch.mul(torch.mean((input[mask]-target[mask])**2), factor)

    numel = (~mask).sum().item()

    # Sin información real, se aplica control de rango
    below_zero = (input < 0)
    below = torch.mul(torch.div(torch.sum((input[((~mask) & below_zero)]
        ** 2)), numel), self.out_limits_factor * factor)

    over_one = (input > 1)
    over = torch.mul(torch.div(torch.sum(((input[((~mask) & over_max)] -
        1) ** 2)), numel), self.out_limits_factor * factor)

    ret = masked + below + over

  return ret
```

- Salida de la primera red neuronal de la capa convolucional, la que realiza estimación de sumatorio de carga. Este valor debería estar en el rango [0, ∞).

- Salida de la segunda red de la capa, encargada de realizar estimación de carga de los nodos. Este valor no debería salir del rango [0, 100].

Todos los valores fuera de esos rangos pueden penalizarse de forma cuadrática según su distancia al límite más cercano, pudiendo aplicar un factor multiplicador para incrementar su importancia en la función de pérdida. Se pueden observar los detalles en el algoritmo 4.2.

Algoritmo 4.2 Penalización valores internos fuera de rango

```
b = torch.clamp(nodes_sum[:, 0:defines.NUM_CHECKED_VALUES], 0)
nodes_sum_l = perdida(nodes_sum, b) * defines.INTER_MULT
l += nodes_sum_l

a = torch.clamp(nodes_pred, 0, 100)
nodes_pred_l = perdida(nodes_pred, a) * defines.INTER_MULT
l += nodes_pred_l
```

4.4. Bucle de entrenamiento

Con el fin de acelerar al máximo el entrenamiento de la red sin llegar a inestabilidades en la secuencia de soluciones obtenidas, se ha optado por un tratamiento automático y dirigido sobre la tasa de aprendizaje empleada. La tasa inicial está definida en el fichero de constantes, y se va ajustando progresivamente según el valor obtenido al evaluar el modelo sobre el conjunto de datos de validación. La variación está a su vez definida en una constante para permitir su fácil manipulación, y consiste en un valor entre 0 y 1 que representa en qué grado queda la tasa de aprendizaje según su valor previo. De esta forma, si aplicamos una variación de 0.5, cada vez que el error de validación aumente lo suficiente con respecto al *epoch* anterior, la tasa quedará en la mitad para el siguiente.

Adicionalmente, se ha optado por un tratamiento en dos fases: una primera en la que solo se entrena la primera parte del sistema, con el fin de obtener estimaciones razonablemente buenas para los arcos, dando paso a una segunda donde se entrena la red completa buscando el equilibrio final en las soluciones. Este equilibrio es configurable, dado que el sistema permite asignar una ponderación mayor o menor a cada salida del sistema mediante un factor multiplicador del valor de pérdida.

En el diagrama 4.5 se observa la estructura general del bucle de entrenamiento del sistema.

4.5. Optimización consumo memoria GPU

Cuando el sistema es alimentado por grandes volúmenes de datos o se apilan muchas capas convolucionales de forma recurrente, es posible que la información requiera una gran cantidad de memoria para gestionarlo. Este hecho genera un problema al usar unidades de procesamiento gráfico para el aprendizaje del modelo, puesto que salvo que sean de alta gama, cuentan con unos pocos Gigabytes de capacidad máxima. Dado que el juego de datos usado para el entrenamiento también emplea parte de la memoria, es posible encontrar problemas de incapacidad de reservar espacio para almacenar datos necesarios. Éstos no son tan frecuentes si el cálculo se realiza mediante CPU, dado que la RAM suele tener un mayor tamaño en la mayoría de dispositivos actuales. Sin embargo, la velocidad a la que se realizan operaciones de este tipo se ve ralentizado en gran medida usando CPU. Debido a ello, ha sido requerida la implementación de un mecanismo que, en lugar de disponer todo el juego de datos en la memoria donde se realizan los cálculos, lo deja en la RAM para que sea más rápido usarlos que desde disco, y va cargando datos a medida que los necesita en la memoria de la GPU. Este comportamiento se logra gracias al uso de la clase *DataLoader* de *PyTorch*. Además, permite realizar un remezclado aleatorio de las muestras para no consumirlas siempre en el mismo orden, por ejemplo, o la posibilidad de indicar el tamaño del minilote a procesar, acelerando a su vez los cálculos de esta manera.

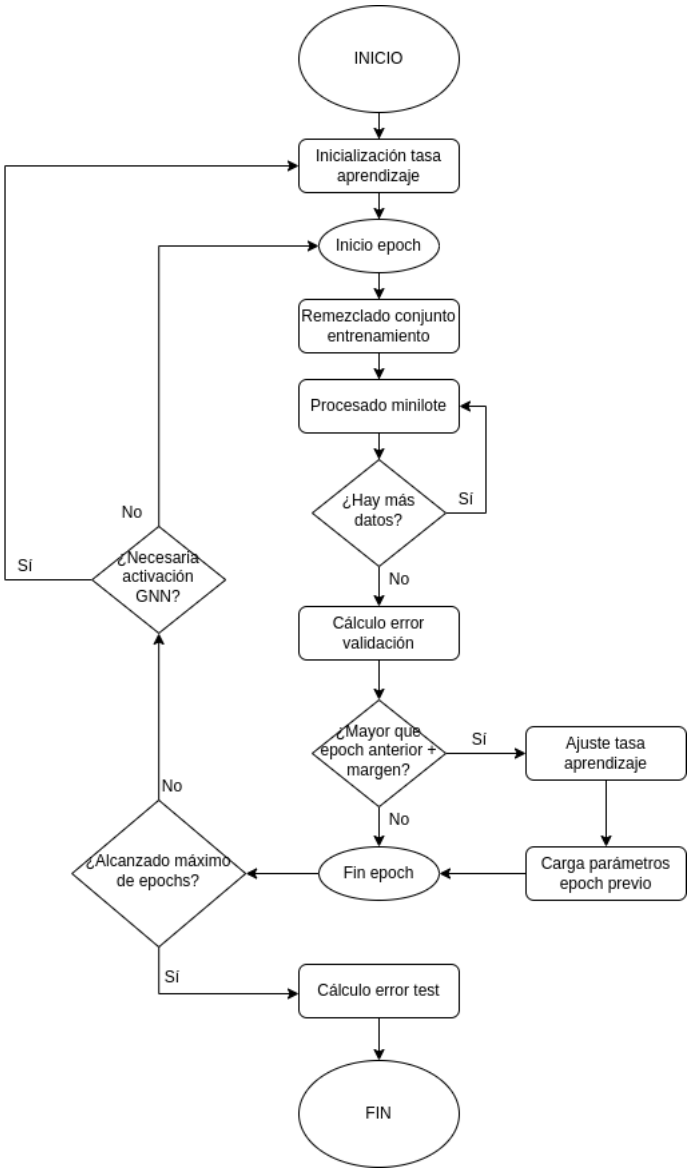

Figura 4.5: Flujo bucle entrenamiento

4.6. Resumen

Tras definir las dos principales entradas del sistema, espaciales y temporales, y especificar cuál será la salida del sistema que podremos contrastar con las etiquetas del conjunto de datos disponible, hemos visto cómo el modelo conjuga cuatro redes neuronales para lograr el aprendizaje en base a los datos. Una primera que ofrece una estimación para cada arco, y las otras tres apiladas recurrentemente para ofrecer la estimación final en base a propagaciones espaciales a lo largo de la red.

Dado que la información tratada no tiene etiquetas para todos los arcos de la red con la que poder evaluar su precisión, para los puntos sin medida se propone una penalización únicamente para datos que estén fuera del rango permitido, ofreciendo así un control de las soluciones y que no llegue

el punto en que el modelo sobreajuste los datos con tal de acertar en los puntos de medición.

Con el fin de facilitar el aprendizaje se ha diseñado un algoritmo a medida que toma decisiones de forma automática en base a los resultados ofrecidos por el conjunto de validación, adaptando la tasa de aprendizaje para evitar ciclos de cálculo sin acercamiento a una mejor solución.

El capítulo siguiente estará dedicado a cómo se ha estructurado el trabajo y cómo los diferentes componentes interactúan entre sí para aprender de los datos y ofrecer a un usuario externo la posibilidad el consumo de la información generada por el modelo.

Capítulo 5

Arquitectura y diseño

5.1. Definición de clases

Para tratar de ofrecer una mayor claridad en la exposición de la arquitectura del sistema, ésta ha sido dividida en varias partes diferenciadas.

- Núcleo: Encargada de los flujos de ejecución de la aplicación
- Red: Gestiona toda la información relacionada con el grafo sobre el que se construye la aplicación
- Datos: Procesamiento de información que servirá de juego de datos
- Modelo: Aprendizaje en base a los datos ingestados
    - Adaptador a framework: Interfaz para adaptar el sistema a diferentes frameworks de aprendizaje

5.1.1. Núcleo

Se trata de las clases cuyo cometido es llevar el flujo de la aplicación y ofrecer las funcionalidades tanto para el cálculo como para la consulta de resultados.

Server

Contiene el punto de inicio de la aplicación, delegando en la clase App el arranque de todos los componentes necesarios. Expone además una interfaz que permite consultas por agentes externos al sistema. La tabla de operaciones se encuentra en la sección 5.3.

App

Se encarga de orquestar todas las tareas necesarias para el estudio:

1. Carga de componentes de la aplicación
2. Lectura del mapa y puntos de medición
3. Descarga y procesamiento de datos
4. Inicialización del modelo
    a) Entrenamiento de la red en modo cálculo
    b) Extracción de resultados en modo consulta

5.1.2. Red

En este primer apartado contemplamos las clases que nos permitirán representar la red de carreteras y modificarla según necesidades con el fin de facilitar su uso y simplificar a su vez el problema.

Map

Se trata de la clase principal para la gestión del grafo que representa la red, su simplificación y tratamiento. Se encarga de realizar diferentes tareas de gran importancia:

1. Cargado de la información base del mapa basándose en la ubicación
2. Filtrado por tipo de vía, considerando las más relevantes de tal forma que sea posible simplificar el problema
3. Consolidación de intersecciones, considerando que multitud de nodos juntos pueden ser representados por un solo punto sin perder poder de representatividad, reduciendo considerablemente la complejidad del problema. Un caso de aplicación de esta consolidación podría ser el de las glorietas, donde confluyen varias vías, y donde cada una de ellas está representada por una cantidad importante de nodos y arcos.
4. Unificación de tramos consecutivos de vía. Tras realizar el filtrado de carreteras por su tipo y la consolidación de intersecciones, es posible hallar casos donde una misma vía está representada por varios arcos consecutivos sin que tenga conectividad con ningún nodo más. Por tanto, en este caso procederemos a fusionarlas en un solo arco y a eliminar los nodos intermedios.
5. Convertir grafo original a dirigido, duplicando arcos en caso de ser necesario para carreteras de doble sentido.

6. Eliminación de subgrafos aislados. Debido al tratamiento realizado y haber descartado vías según su naturaleza o tipología, es posible que hayan quedado partes desconectadas de la red principal, por lo que procederemos a eliminarlas del grafo, puesto que generarían problemas de propagación de valores mediante la GNN.

7. Cálculo de la correspondencia entre puntos de medida y arcos donde se encuentran instalados.

ControlPoint

Clase cuyo cometido es mantener la información necesaria para representar un punto de control del tráfico, un medidor instalado en alguna de las vías bajo estudio. Los valores que almacena son características físicas como las coordenadas donde se encuentra, así como códigos que nos permitirán identificar cuáles son sus mediciones de entre todas las muestras del juego de datos.

TrafficControlPoints

Se trata de una clase auxiliar encargada de realizar la lectura de los puntos de medición que afectan a la red tratada por el sistema, desde el fichero almacenado a tal efecto. Se apoya en la clase ControlPoint almacenando un listado de instancias de la misma.

TrafficControlPointsSanJose

Clase que hereda de *TrafficControlPoints* y que procesa el fichero con los puntos de medida del *benchmark PEMS-BAY.* Este documento dispone de una línea por cada punto de medición, donde se indica su identificador único y las coordenadas donde se ubica.

5.1.3. Datos

Conjunto de clases con la habilidad de descargar y procesar toda la información con la que el sistema será ingestado.

DataDownloader

Se trata de una clase que nos permitirá realizar la descarga y tratamiento básico de toda la información que constituirá el juego de datos que procesará el sistema.

Los ficheros con los datos de tráfico constan de un identificador incremental. Cada mes, con la publicación de un nuevo fichero, este identificador aumenta. Por tanto, el proceso tomaría un identificador inicial y otro final de cara a descargar todos los que se encuentren entre ellos, ambos incluidos. Una vez obtenidos, procederemos a almacenarlos en disco para que pueda ser reutilizados por otras ejecuciones.

Esta información se procesará de forma conveniente y, una vez finalizado el proceso, aportará otro nivel de caché más cercano al sistema ayudando a persistir las grandes matrices de datos listos para alimentar al entrenamiento.

DataDownloaderSanJose

Clase que, heredando de la anterior, acomete la lectura del fichero con la información de tráfico necesaria para la aplicación del modelo sobre el *benchmark PEMS-BAY*. Al contrario que ocurre con los datos de Madrid, cuya información está dividida en ficheros mensuales, en este caso consta de un único fichero CSV con la información completa, lo que simplifica en gran medida el procesamiento llevado a cabo por la clase.

TrafficDataset

Apoyándonos en las posibilidades que ofrece DGL para el tratamiento de juegos de datos, implementamos en esta clase la lógica necesaria para realizar todo el flujo de información desde la descarga hasta su persistimiento en disco una vez listo para alimentar a la red neuronal. En la figura 5.1 está descrito el proceso.

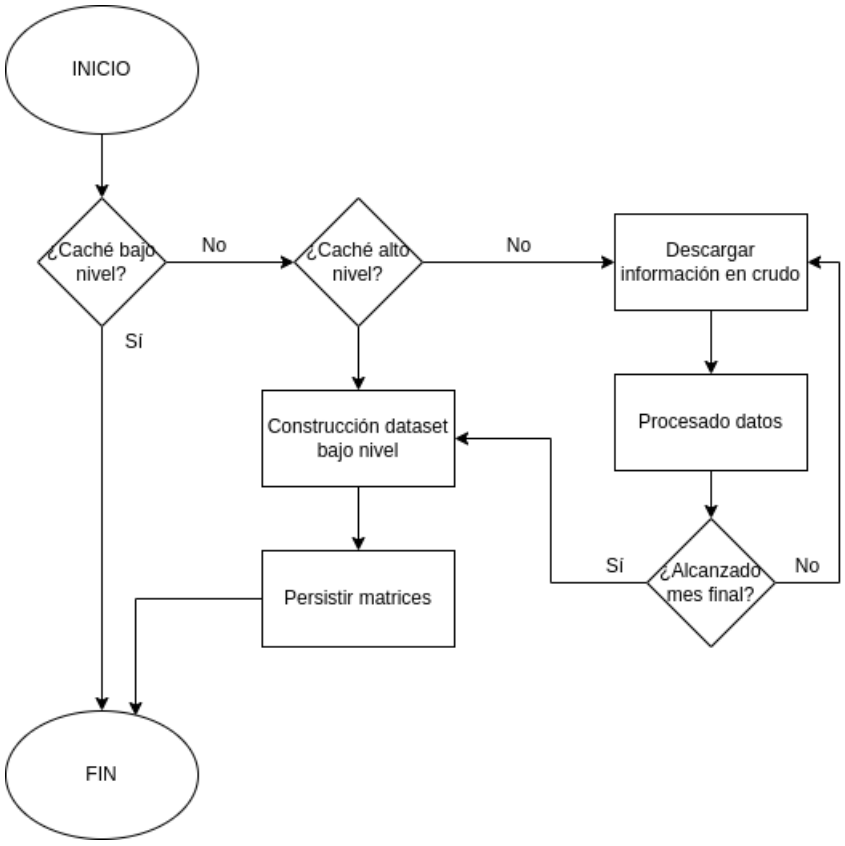

Figura 5.1: Flujo procesamiento juego de datos

TorchDataset

Es importante facilitar la ingesta de la información por parte de la red neuronal haciendo uso de útiles como el agrupado de instancias en mini-lotes o el remezclado de las mismas, por lo que se tomó la decisión de crear esta clase que hereda de *torch.utils.data.Dataset*. De esta forma, se simplifica de forma notable el proceso de gestión de la información. Además tiene la habilidad de realizar carga perezosa, o *lazy-loading*, para no requerir la carga de todo el juego de datos en la memoria de la GPU para su correcto funcionamiento, sino que el proceso se encarga automáticamente de ir ordenando esta tarea a medida que los datos son necesarios para el entrenamiento.

CalendarReader

Clase para la lectura del fichero de días festivos. Contiene la lógica necesaria para la unificación de sus datos, almacenando y permitiendo consultar la naturaleza del día dada una fecha concreta. El fichero facilitado contiene los datos para todo 2022, por lo que si se requiere de información para años posteriores se requeriría su descarga desde el sitio web de datos abiertos del Ayuntamiento de Madrid (`https://datos.madrid.es/egob/catalogo/300082-0-calendario_laboral.csv`).

5.1.4. Modelo

Es el turno de la parte encargada de llevar a cabo el aprendizaje a través de los datos.

TrafficPredictor

Módulo que hereda de *torch.nn.Module* y que contiene las cuatro pequeñas redes neuronales con las que se logra aprender de los datos. El detalle de la estructura del mismo se expone con mayor detalle en la sección 4.3 y subsección 4.3.2.

Como aspectos destacables de su funcionamiento, estaría la habilidad de permitir desactivaciones parciales (red lineal inicial *versus* red GNN), habilitando así cálculos en fases diferentes, centradas en alcanzar objetivos concretos, aunque el funcionamiento por defecto es que toda la red entrena en conjunto.

Permite controlar el número de apilamientos realizado usando la capa convolucional, generando una instancia de la clase *TrafficConvLayer* y la apila tantas veces que se indique, aunque dispone de un parámetro para controlar en tiempo de ejecución el número de veces que aplica la recurrencia e ir incrementándolo desde uno hasta el número máximo indicado. Con ello se logra acelerar el cálculo en fases iniciales, donde el aspecto importante es dirigir los parámetros hasta una zona razonable de su espectro, y que al menos respete el cálculo de la estimación inicial.

Por último, hace de adaptador al uso de procesamiento de información por lotes que propone DGL, cuyo mecanismo es el de generar un grafo con tantos subgrafos disjuntos como instancias tenga el lote a tratar y paralelizar así los cálculos al máximo.

TrafficConvLayer

Representa a una capa convolucional dentro de la GNN y realiza todos los cálculos necesarios para convertir una serie de valores de entrada, que marcan un instante en el tiempo, en varios conjuntos de salida.

Un detalle muy importante es que cuando el modelo apile capas convolucionales sea siempre con la misma instancia de esta clase, o los parámetros aprendidos por cada una pueden ser muy diferentes y sobreajustar demasiado los datos de entrenamiento con las características de la red empleada. De no ser así, no estaríamos hablando de un modelo recurrente y multiplicaríamos el número de parámetros totales, aunque con ello se reduciría la complejidad global y mejoraría algo el rendimiento.

TrafficLoss

Del mismo modo que los dos anteriores, hereda de *torch.nn.Module*. Acepta varios parámetros:

- Tensor a comprobar. Salida proporcionada por el modelo.
- Tensor objetivo. Datos reales que debería ofrecer como salida el modelo en un caso ideal, o *ground truth*.
- Máscara de datos reales. Dado que no todos los arcos presentan medición (por ausencia de medidor o por error), esta marca indicará qué valores hay que contrastar entre los dos tensores indicados previamente.
- Factor multiplicador de la pérdida provocada por el desvío de cada uno de los elementos. Permite personalizar el cálculo de soluciones, dando mayor importancia a ciertas partes de la red con el fin de que tengan un mayor peso en la pérdida global de la salida.

Adaptador a framework de aprendizaje

Debido a que DGL es un framework con posibilidad de adaptación a diferentes entornos de aprendizaje, decidimos facilitarlo en la medida de lo posible para el caso de futuras ampliaciones del trabajo.

BackendAdapter Se trata de un adaptador con las operaciones puramente dependientes del sistema de aprendizaje subyacente, eliminando acoplamientos innecesarios y la necesidad de tener operaciones de este tipo distribuidas por el código.

Las principales operaciones implementadas son:

- Creación/restructuración/concatenación de tensores
- Construcción capas lineales/secuenciales para generación de red neuronal

- Guardado y carga de parámetros del modelo construido
- Evaluación del modelo usando el juego de datos de validación o de pruebas
- Generación de cargador de datos

TorchAdapter Clase de la que hace uso el adaptador anterior para particularizar el cálculo al sistema Torch. En su construcción se selecciona en primer lugar el dispositivo de cálculo a emplear entre CPU y CUDA, así como la precisión necesaria para los cálculos (float/double).

5.1.5. Constantes definidas

El sistema ha sido desarrollado para favorecer la realización de diferentes pruebas modificando parámetros que marcan la estructura de la red o el comportamiento del entrenamiento; existe una definición aislada de los parámetros como constantes para cada uno de ellos. Los principales valores son los siguientes:

- DISTANCE_THRESHOLD: Es la distancia máxima que hará que nodos cercanos en la red se fusionen en el proceso de simplificado de la misma. Se mide en metros.
- RANDOM_SEED: Semilla usada en la generación de números aleatorios.
- NUM_INPUT_FEATURES: Número total de características de entrada al sistema.
- NUM_OUTPUT_VALUES: Número total de valores otorgados como salida para cada arco de la red.
- NUM_HIDDEN_FEATURES_LINEAR: Tamaño de cada capa oculta en la primera fase del modelo.
- NUM_HIDDEN_FEATURES_GNN: Número de características ocultas usada por la GNN.
- EPOCHS_WITHOUT_GNN: Permite especificar el número de ciclos de cálculo sin entrenar la GNN, mejorando solo las primeras estimaciones de los arcos.
- NUM_GNN_LAYERS: Tamaño de la recurrencia en la GNN.
- NUM_GNN_TO_TRAIN: Valor inicial de capas de la recurrencia a entrenar.
- VOID_SOME_POINTS_TRAINING: Modo de cálculo con anulación de datos de medidores.
- MODEL_PARAMS_PATH: Fichero con los parámetros del modelo.
- MODEL_VOIDING_PARAMS_PATH: Fichero con los parámetros del modelo en entrenamiento modo anulación de datos de medidores.

- BACKEND: Sistema de aprendizaje a usar para los cálculos (PYTORCH, MXNET, TENSORFLOW).
- ACTIVATE_GPU: Un valor True en este parámetro indica al sistema que los cálculos deben ser realizados mediante soporte de gráficos.
- BATCH_SIZE: Tamaño de los minilotes a procesar.
- SHUFFLE_TRAIN_DATA: Si está activo, indica que los datos de entrenamiento deben ser remezclados antes de su procesamiento.
- LEARNING_RATE: Tasa de aprendizaje inicial en caso de posponer el entrenamiento de la GNN.
- LEARNING_RATE_GNN: Tasa inicial para la fase de entrenamiento de GNN.
- DECAY_STEP: Pasos de decaimiento de la tasa de aprendizaje.
- GAMMA: Proporción en la que disminuye la tasa de aprendizaje.
- NUM_EPOCHS: Número de ciclos de cálculo máximos.
- OUT_LIMITS_FACTOR: Factor por el que se multiplicará el desvío fuera de límites de los valores otorgados como salida.
- TRAINING_SIZE: Valor entre 0 y 1 que indica la proporción de datos del total a usar para el entrenamiento del modelo
- VAL_SIZE: Valor entre 0 y 1 utilizado para fijar la proporción de datos usados para validación durante el entrenamiento.
- LINEAR_MULT: Multiplicador usado al calcular la pérdida de la primera fase del modelo, previa a la GNN.
- GNN_MULT: Multiplicador para la pérdida de la salida de la GNN.
- INTER_MULT: Indicando valor distinto de cero, permite penalizar valores intermedios de la GNN que se encuentren fuera de rango.
- RANDOMIZE_NUM_GNN: Modo de cálculo que permite, de forma aleatoria, generar un número entre 1 y el máximo de convoluciones apiladas para entrenar en cada epoch con todos los niveles de recurrencia posible.
- SECOND_MAP: Selecciona la ejecución del segundo problema tratado, el *Benchmark PEMS-BAY*.

5.2. Visión global del sistema

La figura 5.2 muestra las relaciones existentes entre las clases que conforman el sistema completo.

Figura 5.2: Arquitectura global del sistema

5.3. Interfaz de consulta

Un sistema de este tipo requiere poder hacer uso de las predicciones del modelo, y ha sido diseñada una interfaz que permite consultar el estado del tráfico basándose en un punto concreto del tiempo, y de forma opcional también del plano de carreteras.

La interfaz contiene las opciones que se muestran en la tabla 5.1

Operación	Ruta	Descripción	Parámetros	Observaciones	Requerido
GET	/traffic/point	Información de un punto	lat	Latitud (UTM)	X
			lon	Longitud (UTM)	X
			year	Año	
			month	Mes	
			day	Día	
			hour	Hora	
			minute	Minuto	
GET	/traffic/info	Información red completa	year	Año	
			month	Mes	
			day	Día	
			hour	Hora	
			minute	Minuto	

Tabla 5.1: Interfaz de consultas al modelo

Ambas operaciones retornan un objeto de tipo JSON A con la información requerida. Las analizaremos por separado mostrando los detalles de cada una:

5.3.1. /traffic/point

Esta operación permite solicitar información del arco más cercano a las coordenadas especificadas por latitud y longitud, en el instante indicado por el resto de parámetros (año, mes, día, hora y minuto).

Ejemplo de petición

Un ejemplo sería realizar una solicitud de tipo GET al endpoint junto a los parámetros especificados:

/traffic/point?lat=40.4305018691825&lon=-3.6883232754856&year=2022&month=5&day=15 &hour=10&minute=30. Con esta llamada, se extrae el valor de ocupación de vía para las 10:30 del día 15 de mayo de 2022.

Como respuesta, se obtendría el siguiente objeto:


```
{
        "request": {
        "lat": "40.4305018691825",
        "lon": "-3.6883232754856",
        "year": "2022",
        "month": "5",
        "day": "15",
        "hour": "10",
        "minute": "30",
        "using_now": false
    },
```

```
    "value": {
        "idx": 737,
        "name": "Calle de José Ortega y Gasset",
        "estimation": 19.22
    }
}
```


La estimación de ocupación de la vía sería de 19.22 %, tal y como expresa el campo *value.estimation*. Se incluyen los parámetros usados para la request y un campo llamado *using_now* que indica si ante la ausencia o invalidez de todos los anteriores se ha empleado el tiempo actual como sustituto.

5.3.2. /traffic/info

Se ha habilitado esta operación para que, si lo que se desea es una vista más general de las estimaciones, proceder a su consumo en algún sistema externo. Funciona de la misma forma que la anterior, con la salvedad de que no es necesario indicar ningún tipo de ubicación concreta, dado que retornará la información para toda la red contemplada en el sistema.

Ejemplo de petición

Como ejemplo, podríamos tomar una petición GET a la siguiente dirección: */traffic/info?year=2022 &month=5&day=15&hour=10&minute=30*

La respuesta sería la siguiente:


```
{
    "request": {
        "year": "2022",
        "month": "5",
        "day": "15",
        "hour": "10",
        "minute": "30",
        "using_now": false
    },
    "value": [
        {
            "idx": 0,
            "name": [
                "Calle Alcalá",
                "Calle de Velázquez",
                "Calle de Alcalá"
            ],
            "estimation": 16.38
        },
        {
            "idx": 1,
```

```
            "name": "Calle de Alcalá",
            "estimation": 14.51
        }
                ...
        {
            "idx": 3024,
            "name": "Calle Javier de Miguel",
            "estimation": 11.66
        }
    ]
}
```


Se observa cómo el campo value ahora es un listado de puntos, cada uno con su estimación concreta. Dado que se ha realizado una simplificación de la red aplicando uniones de arcos consecutivos sin conectividad con terceros nodos, es posible que un tramo tenga varios nombres, resultado de la unificación de los arcos originales, tal y como se observa en el primer caso del listado.

5.4. Resumen

Finaliza este capítulo habiendo definido en detalle el cometido de las diferentes clases que conforman el sistema, así como las relaciones existentes entre ellas, estableciendo agrupaciones según las tareas de las que se ocupan.

Además, ha sido especificado el detalle de la interfaz expuesta mediante una API REST para la consulta de los resultados ofrecidos por el modelo por sistemas consumidores externos. Ofrece posibilidad de consulta individual o de la red completa.

Una vez descrito el sistema en su conjunto, procederemos a mostrar los resultados obtenidos tras el entrenamiento del modelo y a la evaluación de su rendimiento siguiendo varios mecanismos de medición.

Capítulo 6

Resultados

En este capítulo repasaremos los cálculos realizados, así como los resultados obtenidos por la red para algunos casos englobados dentro de la partición de pruebas del juego de datos, de tal forma que estas instancias no hayan sido procesadas en ningún momento.

Se emplearán dos juegos de datos diferentes:

- Evaluaremos el modelo sobre el juego de datos procesado y definido durante este trabajo localizado en la ciudad de Madrid, tratando de medir su eficiencia a la hora de resolver tanto la estimación de tráfico para los puntos con medidor, como su capacidad para propagar espacialmente el nivel de saturación de las vías.
- Existe un juego de datos ampliamente usado en el campo de las predicciones del estado del tráfico, localizado en la ciudad de San José (California). Su nombre es *PEMS-BAY* y consiste en una serie de mediciones de velocidad media de los vehículos. Trataremos de predecir estos valores usando el modelo desarrollado para extraer una comparativa del rendimiento obtenido con las aproximaciones ya existentes.

6.1. Juego de datos de tráfico de Madrid

El conjunto de instancias de las que se ha hecho uso está compuesta por el histórico publicado para el rango temporal comprendido entre octubre de 2021 y marzo de 2022, ambos inclusive, lo que nos ofrece una representatividad lo suficientemente importante para el cometido del estudio sin alargar demasiado los tiempos de ejecución. El total de muestras disponibles asciende a 17468.

El juego de datos se ha dividido de forma aleatoria en tres subconjuntos:

1. Conjunto de entrenamiento (13975). Representa el 80 % del total de instancias, y se ha empleado para el aprendizaje de la red neuronal.
2. Conjunto de validación (2621). Contiene el 15 % del total de muestras, usándose para dirigir el cálculo y almacenar el mejor modelo.

3. Conjunto de pruebas (872). Instancias reservadas para su uso posterior, en la evaluación final de la calidad de la solución. Este subconjunto contiene las instancias usadas en próximas secciones para describir los resultados obtenidos.

Usualmente el reparto es 80-10-10, o 70/15/15, pero se ha estimado conveniente modificar las proporciones buscando obtener mejores resultados.

6.1.1. Especificación de pruebas

Para probar el rendimiento del sistema sobre este conjunto de datos de la forma más exhaustiva posible, se ha diseñado un procedimiento dividido en dos partes diferenciadas:

1. Análisis de salidas del sistema, tanto de la primera parte lineal de la red neuronal como tras las operaciones de propagación necesarias por parte de la GNN. Los nodos afectados por la evaluación será la totalidad de los puntos de medición de la red.

2. Análisis de salida mediante inhibición de puntos de medida para los cuales sí existen datos en el conjunto de muestras, proceso que permite analizar si la propagación espacial que ofrece el sistema está funcionando de forma adecuada.

Dado que el valor de salida ofrecido para cada arco está en el rango [0, 100], evaluaremos el desvío medio en los puntos de medida afectados por cada prueba, así como comparativas gráficas para disponer de una visión global de cuáles son las principales discrepancias entre los datos reales y los que ofrece el modelo.

Llegados a este punto, es importante recordar el marco de mediciones del conjunto de datos descrito en la sección 3.3.1, por el que de toda la red, 3021 arcos, tan solo existen mediciones para 1183 de ellos.

Volumen de conjuntos de prueba

En primer lugar, describiremos con mayor detalle los conjuntos reservados para pruebas del juego de datos original. Del total de 17468 muestras disponibles, tal y como se comentó al inicio del capítulo, 2621 seleccionadas aleatoriamente conforman el conjunto de validación, mientras que 872 están reservadas para las pruebas finales de evaluación del sistema. No todas las muestras tienen disponibles el mismo número de mediciones, debido a errores o a modificaciones en los sistemas de detección de tráfico. Las volumetrías de ambos conjuntos se muestran en la tabla 6.1.

6.1.2. Resultados del modelo

Este apartado estará dedicado a mostrar los resultados de las simulaciones realizadas con el modelo entrenado, contrastando con parte de las muestras reservadas para pruebas.

	Muestras	Mediciones	Mediciones/muestra
VAL	2621	1872416	714.39
TEST	872	611537	701.3

Tabla 6.1: Tamaño conjuntos validación y pruebas

Resultados numéricos

En esta sección trataremos de exponer una comparativa clara del rendimiento final del modelo, comparándolo con su versión básica, únicamente una red neuronal que estime ocupación de vías sin hacer uso de propagación espacial. De este modo, tendríamos tres valores diferentes a comparar entre sí:

- Red neuronal básica: Se trata de una red que aprende patrones temporales exclusivamente, y realiza estimaciones para todos los arcos de la red. Se toma como referencia a la hora de evaluar el impacto de la GNN.

- Red completa:

  - Salida intermedia: Ofrece la primera estimación. Sería equivalente a la red neuronal básica, pero ha sido entrenada en conjunto con la GNN.
  - Predicción GNN: Salida final del sistema haciendo uso de la proximidad de los elementos de la red.

		Sin GNN	Con GNN	
	Tipo error	Predicción	Salida intermedia	Predicción GNN
VAL	Por predicción	9.176162	9.191488	9.145563
	Por muestra	9.164991	9.17144	9.120586
TEST	Por predicción	9.259950	9.267505	9.213801
	Por muestra	9.256214	9.267568	9.20787

Tabla 6.2: Datos evaluación modelo

En la tabla 6.2 se puede observar cómo, aunque en una proporción no demasiado importante, los valores de desvío sobre los datos reales mejoran los ofrecidos por la salida intermedia tomada como base para el experimento. El ajuste de los hiperparámetros y el aumento del tiempo de aprendizaje del modelo podrían ayudar a que la diferencia fuese mayor.

Análisis error por medición En los boxplots mostrados en las figuras 6.1 y 6.2, es posible comprobar cuál es la distribución de los errores obtenidos para los conjuntos de validación y test, respectivamente.

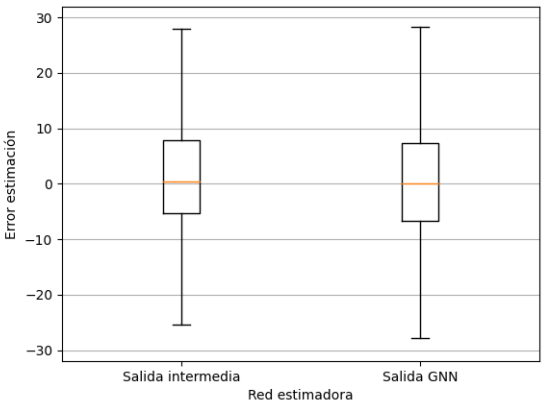

Figura 6.1: Boxplot error por medición datos validación

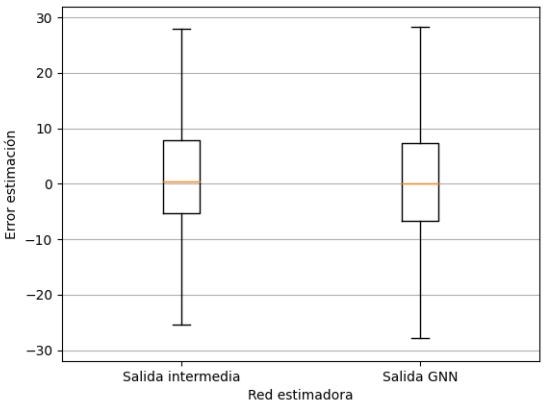

Figura 6.2: Boxplot error por medición datos pruebas

Ambos son prácticamente idénticos, por lo que consideramos que la solución tiene una generalización buena, y que el hecho de emplear el conjunto de validación en el entrenamiento no ha afectado al aprendizaje de la red.

Centraremos por tanto el análisis en la figura 6.2. Estableciendo la comparativa entre las dos salidas ofrecidas por la red, se observa que la mayor parte de las estimaciones estarían en el rango [-8, 8] de desvío sobre el valor real, lo que indica un resultado bastante bueno al ser prácticamente despreciable a efectos prácticos en este contexto. Es cierto que existen desvíos importantes en algunos arcos, por encima de 25 puntos de diferencia sobre el valor objetivo. Este desvío podría explicarse por el hecho de que el número de características contempladas por el sistema es reducido, tan solo puede aprender de patrones relativos a ciclos temporales como las semanas o las festividades. Factores

como las condiciones climatológicas o los accidentes producidos influyen en gran medida en el estado del tráfico, y no han sido contempladas como entrada al sistema.

Análisis error por muestra No todas las muestras cuentan con el mismo número de mediciones, tal y como fue expuesto en la sección 3.3.2, por lo que quizá la métrica más fiable sería la de error por predicción individual. Sin embargo, cabe la posibilidad de que las diferentes muestras completas de la red presenten diferencias en cuanto al error producido por el modelo según la cantidad de mediciones que contenga.

La figura 6.3 muestra una comparativa del error medio en valor absoluto de las predicciones contenidas en una muestra, según el número de mediciones existentes en ella. Esta sería la imagen que se corresponde con la salida intermedia del sistema, previa al cálculo con la GNN. Se observan patrones algo extraños, dado que los puntos no están dispuestos de forma regular y existen casos donde el error es alto para el número de mediciones. Estamos hablando de la especie de ramificaciones existentes en los segmentos entre 800 y 900 y, de igual forma entre los 1000 y 1150 aproximadamente. El error en estos casos no se distribuye de forma equitativa y la mayoría de los casos tienen el mismo volumen de error.

Existe una posible explicación para este fenómeno, y es que en estos casos los puntos de medición con valores válidos sean algunos de los que más error acumulan, los más inciertos. Esto se podría mitigar en cierta medida dando una importancia a cada predicción individual, valorando cómo de relevante es un dato de salida en función de características propias de la vía como el número de carriles, por ejemplo.

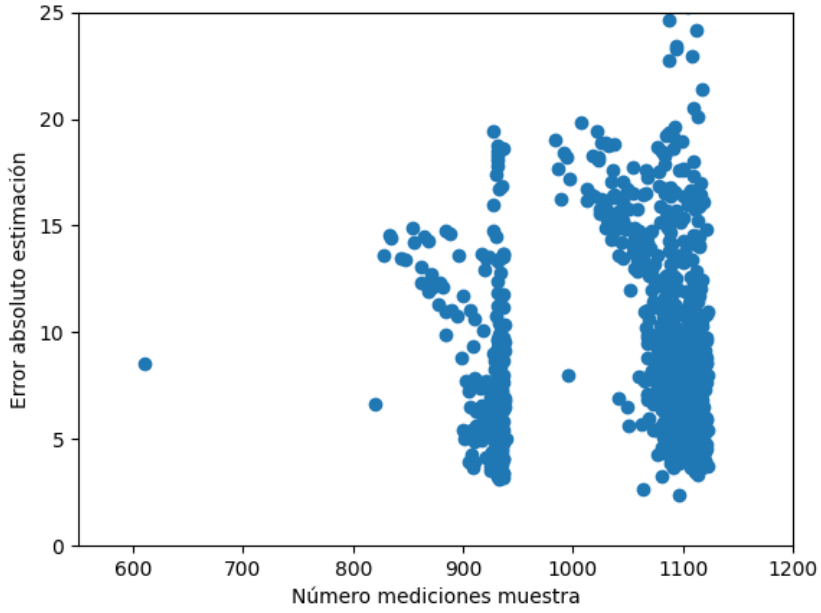

Figura 6.3: Error absoluto salida intermedia medio según número de mediciones por muestra

En la figura 6.4 se observa cómo la salida final del sistema reproduce los mismos patrones, aunque de forma ligeramente más compacta.

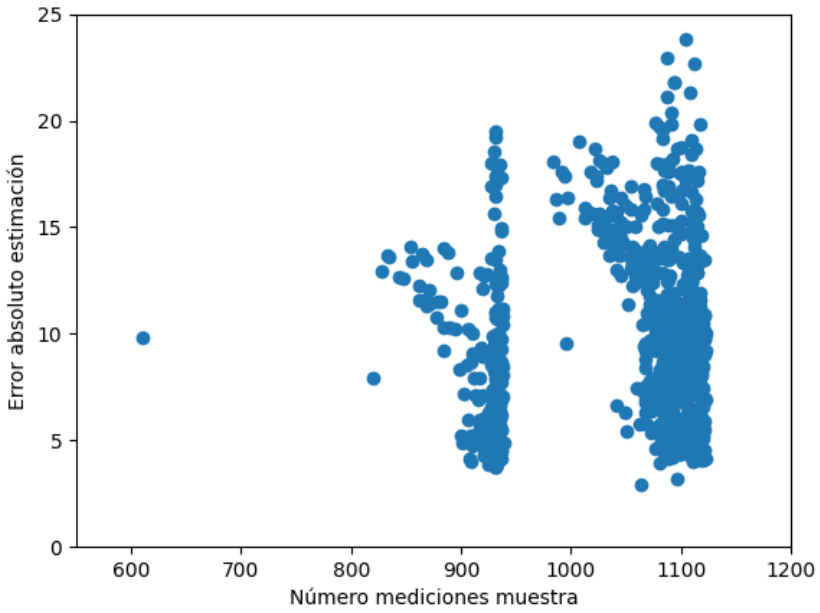

Figura 6.4: Error absoluto salida GNN medio según número de mediciones por muestra

Casos de prueba gráficos

Mostraremos los resultados obtenidos usando algunos ejemplos ilustrativos. Mencionar que en las imágenes relativas a la representación de datos originales, los arcos de la red pintados en negro representan vías sin medidores de tráfico, por lo que no se puede mostrar una imagen completa del estado. La situación de estos puntos es la que trataremos de inferir usando la GNN, así como mejorar las estimaciones realizadas por la primera parte de la red.

La codificación de colores usada para las imágenes de plano de carreteras es la siguiente:

- Negro: Sin valor de medición
- Verde: Entre 0 % y 5 % de capacidad máxima
- Naranja: Entre 25 % y 50 %
- Rojo: Entre 50 % y 75 %
- Marrón: Entre 75 % y 100 %

Caso 1 El primer caso a mostrar es el correspondiente al 31 de enero de 2022, a las 20:30. La muestra está bastante completa, lo que hace que sea interesante para evaluar el rendimiento en su conjunto.

En una primera comparativa, ilustrada en la imagen 6.5, es posible observar cómo se acerca bastante al objetivo. Casi todas las zonas saturadas, destacadas en color rojo, se identifican sin problema; tanto el tramo este de la M-30 como calles aisladas en el norte y el oeste de la ciudad. Además, detecta con alta fidelidad la congestión del centro de la ciudad, con segmentos en verde, naranja y rojo.

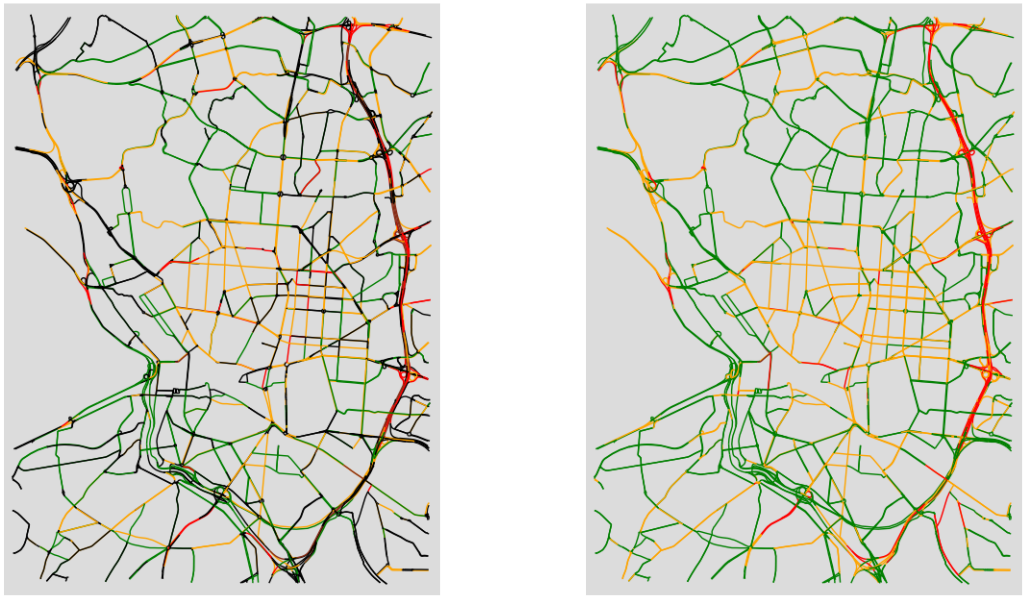

Figura 6.5: Caso 1. Original vs. Resultado. 31/01/2022 20:30

La representación anterior es meramente ilustrativa. En la imagen 6.6 se puede observar en una escala de colores similar la diferencia absoluta entre el original y la predicción realizada por el modelo. La escala de colores representa la diferencia en valor absoluto y se muestra junto al mapa.

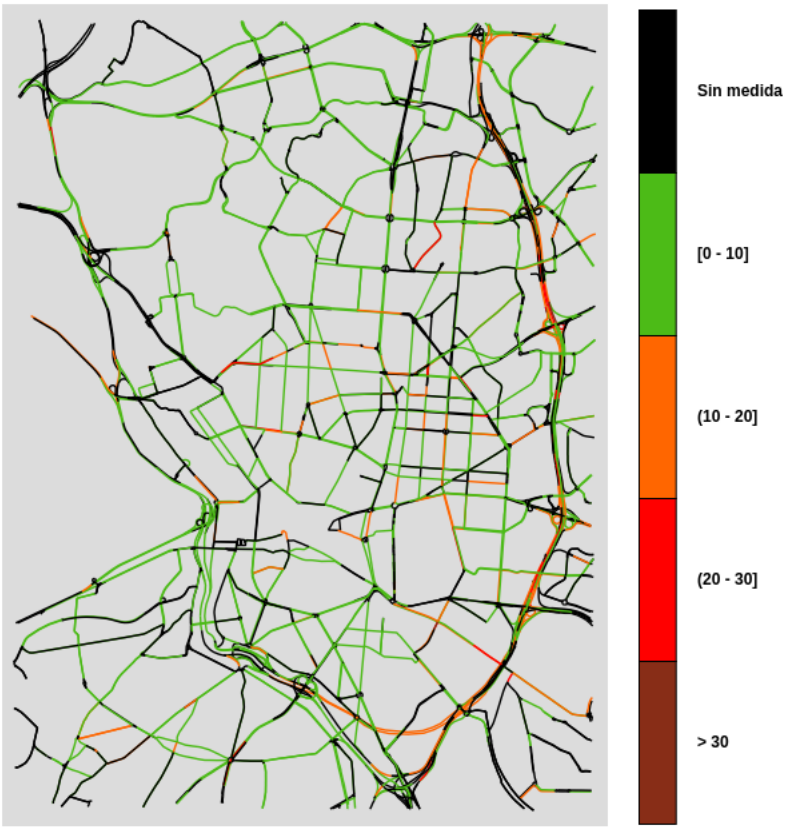

Figura 6.6: Caso 1. Diferencia absoluta entre original y predicción final

Aunque a priori parecía que la estimación apenas difería de los valores originales, esta vista de diferencias nos presenta una realidad distinta. Existen algunos errores altos que, aunque se producen en su mayoría en pequeños tramos viarios, deben ser tenidos en cuenta a la hora de analizar la precisión del sistema y establecer los mecanismos necesarios para su mitigación. Serían de especial interés los tramos de la circunvalación M-30 que presentan coloración naranja y roja, dado que se trata de una vía de alta capacidad que condicionaría mucho su aplicación a mecanismos de cálculo de rutas óptimas.

La tabla 6.3 muestra cómo se distribuyen los errores en función de la cuantía, y de si la estimación está bajo o sobre el valor objetivo. El modelo en este caso ha tendido principalmente a generar estimaciones por debajo del valor deseado, con más de dos tercios de las predicciones. También se observa cómo el número de predicciones cuyo error es menor a 10 puntos es superior al 80 % sobre el total de arcos con medidor.

Rango error	Infraestimado (<0)	Sobreestimado (>=0)	**Total arcos**
0-10	611	307	918
10-20	150	21	171
20-30	22	2	24
>30	7	1	8
Total	790	331	1121

Tabla 6.3: Caso 1. Distribución del error

Caso 2 El segundo caso a analizar será el de la muestra del 19 de marzo de 2022, a las 17:30. Es un caso interesante, dado que no dispone de mediciones en casi la totalidad de puntos de la zona este de la M-30, por lo que será de utilidad a la hora de mostrar cómo se propaga el valor de carga de las vías adyacentes a estos puntos y si la estimación dada es acorde.

En un primer vistazo a la imagen 6.7, el resultado es prácticamente idéntico al original, con la salvedad de algún pequeño segmento de vía en rojo no detectado. La GNN es capaz de complementar la información de los tramos faltantes, detectando la saturación de la vía de circunvalación, en sintonía con sus tramos cercanos.

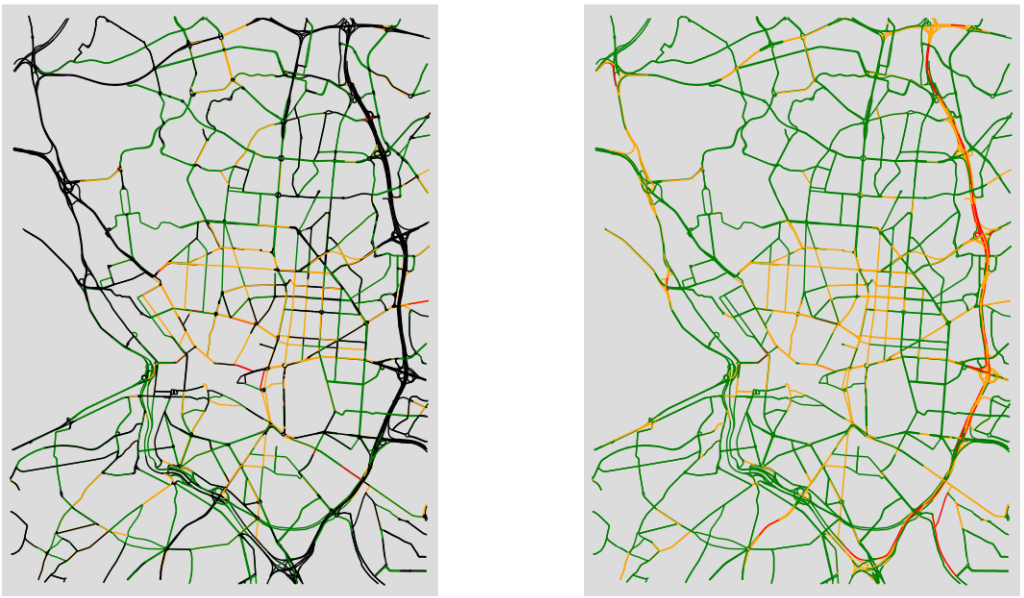

Figura 6.7: Caso 2. Original vs. Resultado. 19/03/2022 17:30

Si pasamos a mostrar el mapa de diferencia absoluta entre estimación y original de la imagen 6.8,

se aprecia un resultado mejor que en el primer caso. Existen muy pocos tramos donde la diferencia esté por encima de los 10 puntos.

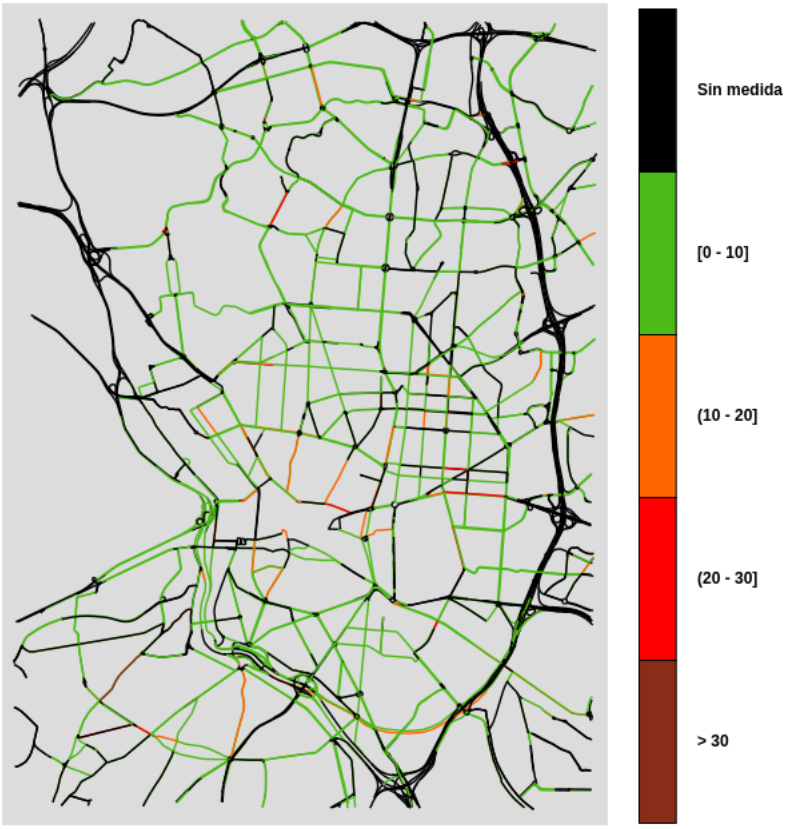

Figura 6.8: Caso 2. Diferencia absoluta entre original y predicción final

Llegados a este punto, mostramos en la tabla 6.4 cómo queda la distribución de los errores según rangos y estimaciones bajo o sobre el valor objetivo. En este segundo caso, tal y como habíamos remarcado en la imagen anterior, la mayoría de los arcos se encuentran por debajo de 10 puntos de diferencia, por lo que conforma una mejor estimación global. De igual modo, cabe resaltar que sigue la misma tendencia que el primer caso, y casi dos tercios de las estimaciones son inferiores al objetivo.

Rango error	Infraestimado (<0)	Sobreestimado (>=0)	**Total arcos**
0-10	546	315	861
10-20	52	10	62
20-30	11	1	12
>30	1	0	1
Total	610	326	936

Tabla 6.4: Caso 2. Distribución del error

Pruebas de propagación espacial

El procedimiento para las pruebas de esta sección ha consistido en entrenar el modelo tras haber anulado en el conjunto de datos de entrenamiento la información de un 10 % de entre el total de arcos de la red. De esta forma será posible evaluar si la GNN propaga bien la información entre arcos cercanos y si los resultados que arroja son interesantes a la hora de usarlos para realizar planificaciones. En la figura 6.9 se destacan en color naranja los arcos que han sido anulados y en verde los que mantienen su información para el entrenamiento.

En la tabla 6.5 figuran los resultados globales del entrenamiento del modelo tras la anulación de las mediciones de estos puntos, así como de forma específica para los arcos afectados, habilitando su comparativa.

		Global		Arcos afectados	
	Tipo error	Intermedia	GNN	Intermedia	GNN
VAL	Por predicción	9.191488	9.545563	12.309134	12.185358
	Por muestra	9.17144	9.520586	12.1996	12.054926
TEST	Por predicción	9.740004	9.91757	12.210588	12.064821
	Por muestra	9.727764	9.899653	12.122015	11.954261

Tabla 6.5: Datos evaluación modelo entrenamiento restringido

De forma lógica, el rendimiento global del sistema ha perdido algo de calidad, estamos restringiendo la información con la que alimentamos el entrenamiento y ejecutando validaciones sobre los datos sin restringir. Aun así, los resultados muestran que el sistema es estable y que la GNN ayuda a generar buenas estimaciones para los arcos cercanos a puntos de medición mediante la propagación de información.

Figura 6.9: Puntos de medición anulados en datos de entrenamiento

6.2. Juego de datos PEMS-BAY

El juego de datos denominado *PEMS-BAY* es un conjunto de información de tráfico que contiene la velocidad media en las algunas vías de circulación de zonas metropolitanas en California, concretamente de la ciudad de San José. Es ampliamente utilizado como *benchmark* para la comparación entre diferentes aproximaciones para la resolución del problema de predicción del estado del tráfico. Esta sección estará dedicada a tratar los resultados obtenidos aplicando el modelo desarrollado en este trabajo, comparando con los resultados existentes mediante el uso de diferentes mecanismos de aprendizaje, tal y como fue introducido en el capítulo 2.

6.2.1. Detalles del dataset

El juego de datos se compone de información relativa a 325 sensores distribuidos según se muestra en la imagen 6.10.

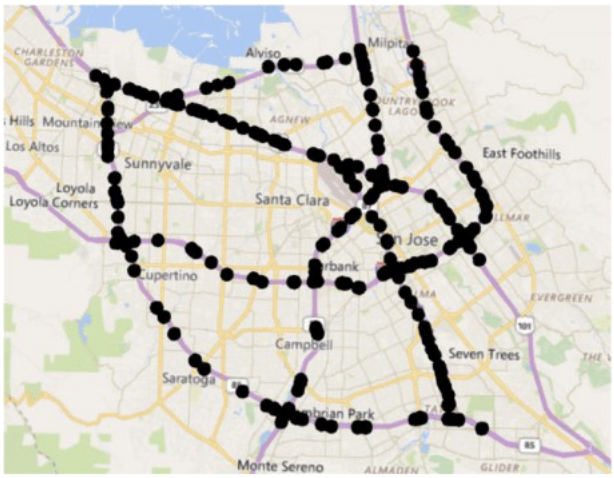

Figura 6.10: Distribución puntos de medida PEMS-BAY

Debido a que la estructura de modelo desarrollada en este trabajo trata de simplificar la red para evitar excesivos problemas de rendimiento, haremos uso de 250 del total de puntos de medida, reduciendo la complejidad al desechar la información de algunos de ellos para los cuales la distancia física con sus contiguos es mínima y se encuentran en el mismo tramo viario. En estos puntos consecutivos las mediciones son prácticamente idénticas y aumentan en gran medida el tamaño del grafo a tratar, por lo que se ha tomado la decisión de proceder a su descarte. En la imagen 6.11 se puede ver la estructura de la red de carreteras simplificada, así como los puntos de medición en color rojo.

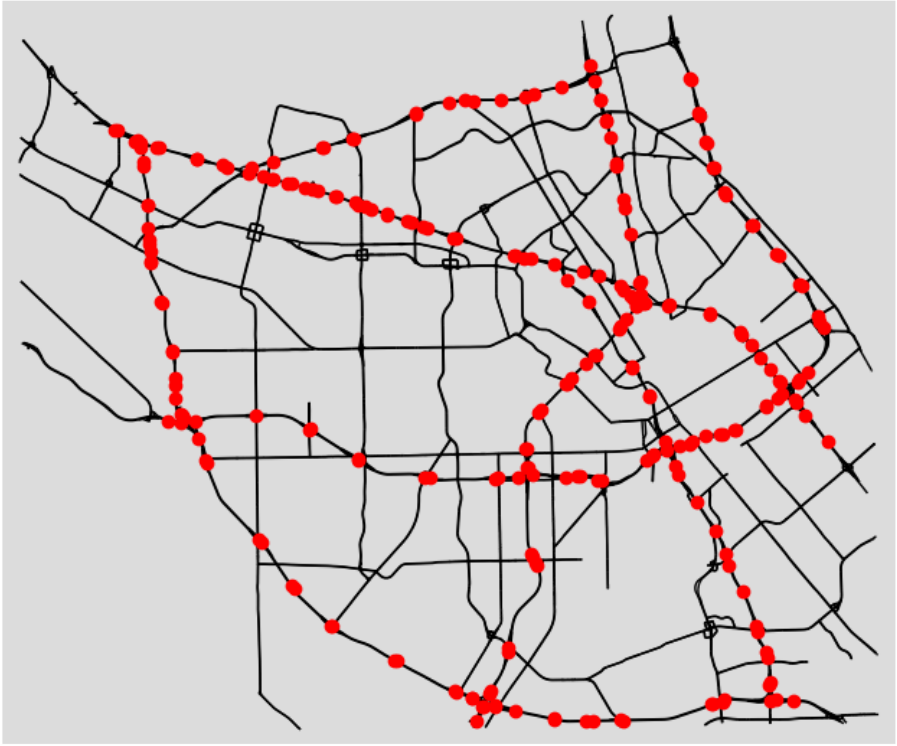

Figura 6.11: Red PEMS-BAY simplificada y puntos de medición

Este dataset contiene datos relativos al periodo comprendido entre el uno de enero y el 30 de junio del año 2017, con muestras tomadas cada 5 minutos para todos los puntos de medida de la red. A diferencia del conjunto de datos central de este trabajo, focalizado en la ciudad de Madrid, las muestras existentes son completas y todas son aplicables de forma íntegra sobre la red, sin parcialidades.

Es necesario tomar en consideración algunas cuestiones sobre la naturaleza de esta prueba y su aplicabilidad:

1. Se hace uso de cierta información que otras aproximaciones no emplean, como el grafo completo y las características de las vías de circulación que incluye la red. Asimismo, desechamos uno de los ficheros que componen el dataset y el cual contiene la matriz de distancias entre los puntos de medición por el mismo motivo.

2. Dado que el modelo planteado difiere en ciertos planteamientos básicos de la mayoría de los ya estudiados, los resultados no serían estrictamente comparables entre sí.

3. Los modelos usados en las comparativas no realizan estrictamente el mismo tipo de predicción, sino que la mayoría de ellos ofrecen estimaciones basadas en un histórico y en información reciente para predecir estados del futuro cercano.

4. Las pruebas estarán simplificadas por la unificación de vías y la compactación de la red.

5. Las instancias no se han categorizado exactamente igual en los subconjuntos de entrenamiento y pruebas por la dificultad que acarreaba su acoplamiento al modelo desarrollado.

6.2.2. Resultados del modelo

En este apartado procederemos a extraer los resultados ofrecidos por el modelo sobre el problema. Resulta imprescindible destacar que al medir ambos problemas diferentes magnitudes, porcentaje de ocupación en el caso de Madrid y velocidad media en el que nos ocupa, las cifras y representaciones gráficas de ambos no son comparables entre sí. Al final de esta sección trataremos de establecer las comparaciones aceptables, siempre tratando con resultados obtenidos para el mismo problema y tomando todas las precauciones posibles en el análisis.

Resultados numéricos

El número total de mediciones existentes es de 52116, algo superior al caso anteriormente tratado para Madrid. De igual modo, realizaremos la división del total en tres subconjuntos diferentes para entrenamiento, validación y pruebas, aplicando los mismos porcentajes (80/15/5) para cada subconjunto respectivamente.

De igual modo que en la sección anterior, evaluaremos dos datos por separado:

- Salida intermedia: Ofrece una primera estimación. Se trata de una red neuronal básica, con la salvedad de que ha sido entrenada en conjunto con la GNN.

- Predicción GNN: Salida final del sistema haciendo uso de la propagación espacial entre los elementos de la red.

En la tabla 6.6 se pueden observar los valores de error medio obtenidos para los subconjuntos de validación y pruebas. Los resultados difieren poco entre sí, por lo que parece que el modelo no está sobreajustado. Un dato relevante es cómo el error en la predicción realizada por la GNN es un 7 % inferior a la de la salida intermedia, lo que indica que contribuye a conseguir mejores resultados sobre la red neuronal inicial.

	Tipo error	Salida intermedia	Predicción GNN
VAL	Por predicción	5.874123	5.563825
	Por muestra	5.874124	5.563827
TEST	Por predicción	5.897216	5.601253
	Por muestra	5.897216	5.601253

Tabla 6.6: Datos evaluación sobre PEMS-BAY

Evaluación gráfica de las diferencias obtenidas

Los gráficos 6.12 y 6.13 muestran la distribución de los errores para los subconjuntos de validación y pruebas. Ambos son prácticamente idénticos, variando tan solo en pequeñas fracciones de unidad, reforzando así la idea de que el error es muy similar en ambos y marcando claramente la mejora en la salida de la GNN con respecto a la red neuronal inicial.

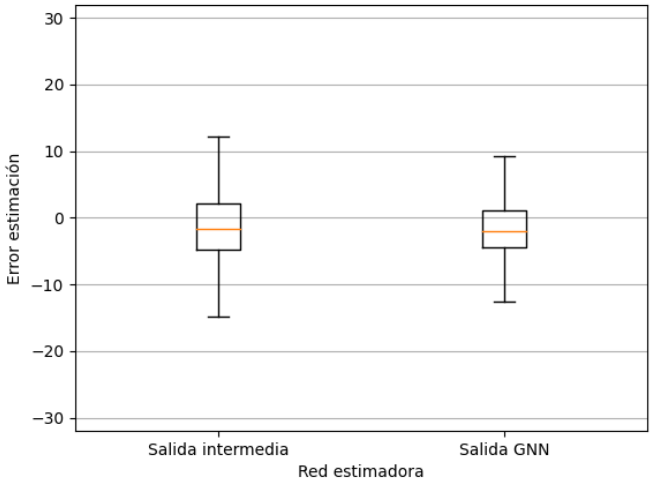

Figura 6.12: Boxplot error por medición de datos conjunto de validación (PEMS-BAY)

Comparativa con otras aproximaciones

Existen multitud de modelos con los que se ha tratado de resolver este problema, mejorando en gran medida los resultados obtenidos a lo largo del tiempo. Sin embargo, la aproximación desarrollada en este trabajo, tal y como se ha destacado antes, no es equiparable al 100 % con ellos al presentar diferencias en cuanto a los datos de entrada y el volumen de información tratada. Por ello, haremos un análisis somero de algunas de las opciones disponibles y trataremos de extrapolar los resultados usando toda la precaución en las comparativas.

La tabla 6.7 muestra los resultados ofrecidos sobre el dataset por tres de los modelos analizados

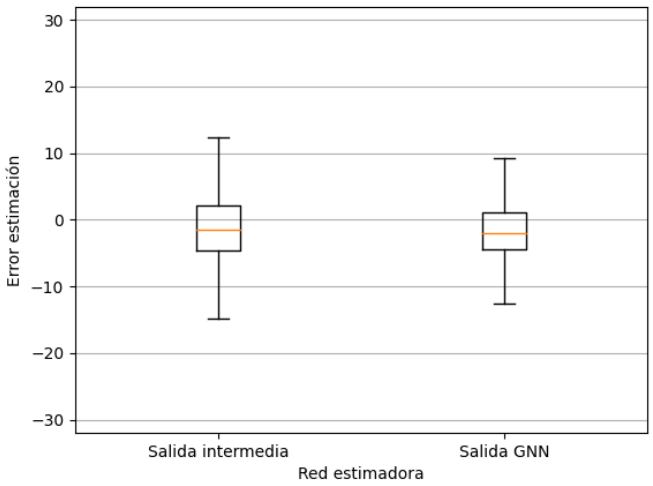

Figura 6.13: Boxplot error por medición de datos conjunto de pruebas (PEMS-BAY)

en el capítulo 2. Conceptualmente, se trata de tres modelos muy diferentes, al basar el aprendizaje de los patrones en distintos principios y arquitecturas.

Modelo	**Descripción**	**Año**	**Error**
DCRNN	Diffusion Convolutional Recurrent Neural Network [Li et al., 2017]	2017	4.74
Graph Wave-Net	Graph WaveNet for Deep Spatial-Temporal Graph Modelling [Wu et al., 2019]	2019	4.52
GMAN	Graph Multi-Attention Network [Zheng et al., 2019]	2019	4.32
GNN-TP	GNN Traffic Predictor	2022	5.6

Tabla 6.7: Comparativa entre aproximaciones sobre PEMS-BAY

Existen estudios posteriores, como [Yoo et al., 2021], que en lugar de ofrecer una arquitectura diferente se centra en aplicar funciones de pérdida evolucionadas, o en cómo conjugar los residuos de información que puedan quedar ocultos o descartados en los modelos anteriores.

Una vez expuestos los valores de error obtenidos con varios modelos de referencia para el mismo dataset, se concluye que el modelo desarrollado no mejora el rendimiento global de ninguno de ellos, aunque como ya fue mencionado, la comparativa no puede ser aplicada de forma directa dado el mecanismo de simplificación y compactación de vías llevado a cabo en éste. En cuanto a la distribución de los errores no se dispone de información suficiente en los artículos mencionados que permitan un estudio comparativo. Se propondrá al final de este documento como vía de trabajo futura

el desarrollo de una prueba en igualdad de condiciones que habilite la evaluación de la aplicabilidad efectiva sobre los *benchmarks* más conocidos.

6.3. Resumen

Finaliza el capítulo reservado a exponer las pruebas realizadas sobre el sistema, y la evaluación tanto numérica como gráfica de los resultados obtenidos. Se ha mostrado cómo el modelo, una vez entrenado, es capaz de dar una buena estimación para todos los arcos de la red, aunque no tuvieran mediciones válidas en el original. De igual modo, se ha realizado un pequeño estudio de cómo ha afectado la no completitud de las muestras al error, dejando patentes patrones de desvío en función del número de mediciones por muestra.

En cuanto al uso del benchmark PEMS-BAY para la evaluación del modelo, se ha llevado a cabo una prueba simplificada y expuesto los resultados, que aunque lejos de igualar los obtenidos mediante otros mecanismos, podrían mostrar vías interesantes de mejora en combinación con otras estrategias.

En el próximo capítulo se describirá la evolución del trabajo realizado, cuáles han sido sus fases y detalles propios del desarrollo ejecutado.

Capítulo 7

Desarrollo

7.1. Resumen tiempo empleado

Una vez finalizado el desarrollo del trabajo, trataremos de medir el desvío en las estimaciones definidas en la sección de planificación (1.3).

Además de las tareas a alto nivel definidas en dicha sección, han surgido algunas nuevas fruto de las dificultades encontradas durante el desarrollo:

- AD1: Ha sido necesario implementar mecanismos que simplifiquen la red de carreteras con la que se ha trabajado. Si bien es cierto que MXNet ya dispone de algunas funcionalidades, se ha requerido un proceso adicional de unificación de características para los casos en los que se unían tramos de vías con alguna característica no común.

- AD2: Implementación de componentes a medida, como la capa convolucional o la función de pérdida empleada. El desarrollo de estos componentes ha resultado imprescindible para lograr resultados satisfactorios.

- AD3: Adaptaciones para el entrenamiento del modelo usando el *benchmark PEMS-BAY* y extracción de resultados.

En la tabla 7.1 se ve cómo la fase 1, de estudio de documentación sobre el problema, ha llevado menos tiempo del estimado. Sin embargo, la parte de desarrollo y pruebas ha empleado esta diferencia, y junto con las tareas adicionales ha sumado unas 45 horas más de lo presupuestado.

Estudiando globalmente la dedicación al proyecto, el desvío ha sido de 25 horas (inferior al 10 % sobre el total), por lo que las estimaciones fueron bastante ajustadas a la realidad. Sin embargo, se tomó la decisión de añadir la comparativa con el *benchmark* para una mayor completitud del trabajo.

Código	Fase	Horas empleadas	Horas estimadas
1.a	Estudio preliminar	20	30
1.b	Estudio profundo	40	50
2.a	DGL	30	30
2.b	MXNet	20	15
2.c	OSM	10	10
3	Prototipado	85	80
4	Pruebas y ajustes	50	40
5	Análisis final	15	15
6	Redacción memoria	35	30
AD1	Reducción problema	10	-
AD2	Componentes a medida	10	-
AD3	*Adaptación a PEMS-BAY*	20	-
Total		**345**	**300**

Tabla 7.1: Desglose trabajo efectivo por fases

7.2. Evolución prototipado

En esta sección trataremos de enumerar las fases que han llevado a obtener el producto final mediante el iterativo incremento de funcionalidad.

1. Gestión de mapa de carreteras: Incluiría la descarga de toda la información de las vías de circulación, así como su procesamiento y estandarizado de características de nodos y arcos.

2. Obtención y procesado de la información que constituye el juego de datos.

3. Relación entre puntos de control, mapa y valores del *dataset*.

4. Diseño y desarrollo de red neuronal para estimación de carga por arco.

5. Función de coste y bucle de entrenamiento.

6. Diseño y desarrollo de capa convolucional de GNN.

7. Construcción modelo GNN mediante apilamiento de capas convolucionales.

8. Optimización consumo memoria en GPU.

9. Pintado de resultados sobre mapa.

10. Generación API para obtención de información.
11. Exportación de fichero con el juego de datos ya procesado para su uso futuro en otros trabajos.
12. Procesamiento datos del conjunto *PEMS-BAY*.

7.3. Estructura de directorios

Buscando una mejor interpretación de los ficheros temporales creados, así como la organización de los ficheros necesarios, haremos una enumeración de las partes más importantes del mismo.

- /cache: Directorio creado automáticamente por osmnx a la hora de descargar el mapa. Previene de tráfico de red extra cuando se solicita varias veces la misma zona del plano.
- /data: Es usado para almacenar todos los datos de la aplicación, desde ficheros en crudo descargados desde el origen, hasta los que contienen la información procesada.
  - /calendar: Contiene el fichero calendario.csv, que es el usado para la extracción de los días festivos.
  - /dataset: Directorio donde se cachean seis ficheros correspondientes a características y etiquetas de los conjuntos de entrenamiento, validación y pruebas. Se trata de ficheros binarios generados y leídos por el paquete numpy, para una mayor agilidad entre pruebas.
  - /processed: En él se almacenan los ficheros CSV mensuales una vez eliminadas las columnas no deseadas y la información de puntos de medida no relevantes para la resolución del problema.
  - /raw: Lugar donde se descargan temporalmente los ficheros originales de histórico de tráfico a la espera de ser procesados. Una vez se finaliza la transformación de cada uno, se eliminan progresivamente para evitar el uso indiscriminado de espacio en disco.
  - /ready: En él se vuelca el fichero unificado de datos procesados de histórico de tráfico, contribución de este estudio a usos futuros.
  - madrid_simple.gml: Persistencia del plano de vías de circulación una vez ejecutados los procesos de simplificación del problema, habiendo unificado arcos y nodos.
  - madrid_full.gml: Fichero GML donde se vuelca el plano con los nodos y arcos originales
  - pmed_ubicacion_03-2022.csv: Fichero original descargado del sitio web de datos abiertos del Ayuntamiento de Madrid con la localización de los puntos de medida de densidad circulatoria.
- /data_pems_bay: Equivalente al anterior, pero usado para el *benchmark PEMS-BAY*.

- /model: Directorio donde se sitúa el fichero con los parámetros del modelo de la iteración con la que menor ha sido la pérdida sobre el conjunto de validación. Se recupera en ciertos momentos durante el cálculo, así como teniendo el trabajo en modo consulta.

- /output: Se reserva para la exportación de representaciones gráficas de soluciones sobre el plano estudiado.

- /output_pems_bay: Equivalente al anterior, usado para el *benchmark PEMS-BAY*.

- /src: En él se encuentran todos los ficheros de código de la herramienta.

7.4. Tecnologías y productos empleados

El trabajo ha sido realizado tratando de usar tecnologías de amplio uso por la comunidad, intentando facilitar su comprensión e hipotética extensión por parte de desarrolladores interesados en ello. Estas herramientas facilitan a su vez la creación de prototipos y reducen el tiempo de desarrollo necesario para conseguir resultados.

Python es un lenguaje de alto nivel de programación interpretado y multiparadigma empleado para el desarrollo de cualquier aplicación. Centra su filosofía en la legibilidad de su código, siendo una restricción la correcta indentación del código. Dispone de multitud de paquetes que facilitan y acortan los tiempos de desarrollo en gran medida. Los más populares serían *numpy* para cálculos numéricos, o *pandas* para manejo de volúmenes importantes de información en modo tabla, incluyendo lectura y escritura de ficheros en formatos diversos.

O*penStreetMap [OpenStreetMap Contributors,]* se define como un proyecto colaborativo para crear mapas editables y libres. Los usuarios registrados pueden subir sus trazas desde el GPS ayudando a crear una base de datos con información vectorial, que es la usada como salida del sistema, en lugar de mapas como tal. Esta característica le otorga una gran potencia.

OSMnx [Boeing, 2017] es una herramienta para *python* cuya finalidad es el análisis de redes de vías de circulación de cualquier tipo. Permite la recuperación, modelado, análisis y visualización de las redes y cualquier información espacial usando como origen de datos *OpenStreetMap*.

Flask [Flask,] es un ligero framework para el desarrollo de aplicaciones web en *python*. Permite especificar las operaciones mediante *annotations* de *python*, y definir la lógica de funcionamiento como si de una función se tratara.

Encontramos en DGL [Wang et al.,] un framework para el desarrollo de modelos de aprendizaje aplicable a problemas cuya estructura esté basada en grafos. Entre sus principales características destacarían la agnosticidad sobre *framework* de aprendizaje, eficiencia, escalabilidad y aplicabilidad a ecosistemas diversos.

7.5. Resumen

Este penúltimo capítulo se ha centrado principalmente en cómo se ha desarrollado el trabajo en cuanto a tareas y tiempo empleado en cada una, destacando que las estimaciones para la parte de desarrollo fueron demasiado optimistas debido, principalmente, a la necesidad de desarrollo de componentes a medida para la GNN.

También ha sido especificada la estructura de ficheros del proyecto, como referencia para futuros trabajos que pudieran retomar el desarrollo.

El próximo capítulo estará dedicado a las conclusiones extraídas de la realización del estudio, así como a posibles líneas de trabajo futuro.

Capítulo 8

Conclusiones y trabajos futuros

8.1. Conclusiones

En este trabajo ha sido desarrollado un sistema predictivo del estado del tráfico aplicado a la ciudad de Madrid. Se ha diseñado de tal forma que, con unos pequeños ajustes, sería posible su adaptación a cualquier región con la única premisa de disponer de un histórico de mediciones de ocupación de las vías de circulación.

De igual modo, resultaría sencillo incluir nuevas fuentes de datos al conjunto de información para el aprendizaje; bastaría únicamente con cruzar la información usando como eje el punto en el tiempo en el que se produjo el estado o evento determinado. El sistema de caché del juego de datos lo tomará automáticamente y hara uso de él.

El modelo es capaz de aprender eficientemente los patrones de ocupación de vías, así como de propagar espacialmente estos valores a las partes de la red cuya medición no existe. Tomando como entrada únicamente un punto en el tiempo, ofrece una primera estimación para cada uno de los arcos que conforman la red, y aplicando la estructura de grafo que subyace a la red de carreteras, calcula una nueva estimación del estado en función del valor ofrecido para cada vía aplicando los conceptos de proximidad y conectividad. Al estar ambas estimaciones relacionadas en la cadena de operaciones de la red neuronal, la segunda estimación se contrasta también sobre los puntos con medición, para aplicar factores correctivos en caso de que ésta se desvíe del valor objetivo.

La principal limitación del predictor implementado sería su complejidad. Al tratarse de un sistema recursivo en su segunda parte, la encargada de aplicar la proximidad espacial, las cadenas de operaciones que dan como resultado cada valor pueden ser muy largas si se apilan muchas capas convolucionales. En este trabajo se ha usado un máximo de 5, pero en sistemas de cómputo más potentes podrían ser muchas más buscando obtener estimaciones más precisas. Sin embargo, el modelo planteado ya ofrece resultados buenos, cuyas estimaciones no varían más de 10 % de media para cada medidor contemplado.

La valoración del trabajo es positiva, habiendo logrado diseñar un sistema complejo donde una red

detecta los principales patrones estacionarios en cuanto a la saturación de las partes de la red, y otras tres redes aplican un aprendizaje intenso sobre las relaciones de saturación según la conectividad y características físicas de las carreteras. Al permitir su sencilla extensión, no tiene un límite definido puesto que habilita el enriquecimiento del modelo con información de diferentes fuentes y naturalezas, que también tienen su afectación sobre el problema tratado.

8.2. Trabajos futuros

Dado que se trata de una primera aproximación a la predicción de densidad de tráfico mediante el uso de redes neuronales aplicadas a grafos, especificando únicamente unos valores para identificar un instante en el tiempo, se abren una gran cantidad de posibilidades que permitan mejorar y/o complementar el modelo propuesto. En este apartado se expondrán las principales vías para, partiendo del trabajo realizado, incrementar su precisión y prestaciones tratando de lograr una mayor utilidad.

8.2.1. Nuevas características de entrada

En este apartado destacaremos ejemplos de información con el que enriquecer el modelo.

En primer lugar, existiría la opción de contemplar datos meteorológicos como entrada al sistema. Esto nos permitiría mitigar el efecto negativo que puedan tener los *outliers* en el cálculo. Como ejemplo, tendríamos que un día de lluvia en la ciudad de Madrid puede suponer una densidad circulatoria mucho mayor, efecto del aumento de distancia de seguridad entre los vehículos o la reducción de visibilidad que obliga a aminorar la velocidad. En el modelo presentado, este factor sería desconocido, por lo que tomaría la información como "inconsistente" no siguiendo los patrones hallados.

Otro aspecto importante, y relacionado en parte con el anterior, sería la detección de accidentes que perjudican gravemente la circulación. Madrid es una ciudad con una saturación de tráfico muy elevada en horas concretas, donde los trabajadores se desplazan entre su vivienda y el lugar de trabajo. Por ello, sería interesante contar con una funcionalidad que permita contemplar lugares y ventanas donde se han producido accidentes o averías de vehículos que hayan afectado a la velocidad de circulación. De esta forma, el modelo podría aprender sobre probabilidad de accidentes en vías y ofrecerla como salida, permitiendo ponderar el riesgo de quedar atrapado en un atasco y, quizá, escoger rutas más seguras en función del tiempo que conlleven o su distancia.

8.2.2. Alteraciones red de carreteras

Otro aspecto que resulta muy interesante sería el de, una vez entrenado el modelo y extraídos los patrones de flujo de tráfico a lo largo y ancho de la red, realizar estudios aplicando modificaciones sobre ella. Dado que la red logra extraer patrones de circulación teniendo en cuenta datos como el

número de carriles, la conectividad con otras o la velocidad máxima permitida en un tramo concreto, resultaría interesante estudiar la reacción a pequeñas variaciones de la red de carreteras, y cómo afectaría a los tramos localizados en la misma zona. Quizá el establecimiento de un tramo nuevo entre dos nodos muy concurridos sea interesante de cara a la fluidez de la circulación, o la inclusión de un nuevo carril sobre una vía existente genere el mismo efecto.

8.2.3. Mejoras en procesamiento de puntos de detección

Dado que el Ayuntamiento puede llegar a modificar la ubicación de los puntos de medida, sería interesante manejar una asignación 1:N entre arcos y puntos de medida, de tal forma que sea posible contemplar instancias relativas a diferentes periodos de tiempo de una forma más efectiva. Como ejemplo, en febrero de 2022 un gran número de puntos de medida de la zona este de la M30 han sido modificados y su identificador ha sido sustituido por otro. Un caso muy concreto es el medidor 10265, del que no se hallan datos más allá del 8 de febrero de 2022, limitando la utilidad de las muestras relativas a este periodo temporal.

Otro factor relevante sería usar alguna otra fuente de datos que permita ubicar exactamente los puntos de medida. La aproximación aplicada en este trabajo, empleando la cercanía con un arco de la red de carreteras es válida en un alto número de casos, pero es posible que se haya producido alguna asignación errónea que esté provocando pérdidas de efectividad del sistema.

8.2.4. Importancia de puntos de medida

Durante la realización del trabajo, se ha observado que penalizar todas las vías con un error del mismo orden de magnitud puede llevar a mejores soluciones generales. Sin embargo, no parece tener la misma relevancia fallar en la predicción de un tramo de la M-30 que en una calle del centro de la ciudad. Por ello, se propone como vía futura de trabajo la aplicación de una ponderación del error generado, permitiendo penalizar en mayor medida el error producido en vías con multitud de carriles, o cuya conectividad dentro de la red sea grande, puesto que en este caso una desviación en la estimación propagaría el error a los nodos y arcos cercanos. Esta vía podría llevar incluso a mejores soluciones globales.

8.2.5. Cálculo de rutas óptimas

La finalidad del modelo es ofrecer estimaciones los más precisas posibles, por lo que una posible ampliación al proyecto sería el hecho de calcular un peso asociado a cada arco de cómo penalizaría el nivel de saturación a su tiempo de desplazamiento. Una vez conseguido esto, sería posible plantear algoritmos de caminos mínimos para ofrecer la ruta más rápida entre dos puntos de la red, basándose en la salida del modelo de aprendizaje en lugar de en la longitud de los arcos, por ejemplo.

8.2.6. Aplicación sobre *benchmarks* del sector

Se propone como una vía de trabajo futuro el cálculo de predicciones usando conocidos *benchmarks* de estimación de volumen de tráfico como *METR-LA*, al igual que ha sido aplicado *PEMS-BAY*, de tal forma que sea posible evaluar su comportamiento ante un problema que ha sido abordado con numerosos métodos diferentes, permitiendo su comparativa. Para ello, sería necesario realizar ajustes al modelo que permitieran el cálculo sin compactación de tramos viarios, al menos en los que existan varios puntos de medida. Esta vía estaría altamente relacionada con lo descrito en la sección 8.2.3.

8.3. Contribuciones

Durante el desarrollo del proyecto, se han llevado a cabo tareas que pueden ayudar a una agilización de futuros trabajos, tanto a nivel de código de componentes relacionados con la GNN como información ya procesada que acelere y facilite su utilización.

8.3.1. Componentes GNN desarrollados

Debido a la ausencia de componentes estándar dentro de DGL (*Deep Graph Library*) que pudieran ayudar al cometido del proyecto llevado a cabo, tal y como se hizo mención en secciones anteriores, se optó por desarrollar un módulo específico que se adaptase plenamente al modelo planteado. Dado que tiene que modelarse como una estructura de grafo, consta de tres partes principales:

1. Función principal de comportamiento del módulo. Su nombre en terminología de *Torch* es *forward*.

2. Cálculo de valores a propagar de arcos a nodos y agregación a nivel de nodo.

3. Propagación de nodos a arcos como generación de salida de la capa.

El código de todos los componentes desarrollados se encuentra accesible en `https://github.com/lmcorreas/gnn-traffic-predictor`, tal y como fue descrito en la sección 1.5.

Función *forward*

Esta función constituye el punto de entrada a la capa convolucional. Recibe un grafo y una matriz de valores (N valores por M muestras) correspondiente al minilote a procesar, y su misión está en ofrecer una salida con la información procesada, que pueda ser posteriormente comparada con los valores reales y retropropagar así para lograr que el modelo realice la fase de aprendizaje. En el algoritmo 8.1 se puede ver el código detallado y comentado, donde es necesario tener en cuenta que las variables 'edata' y 'ndata' corresponden a los datos a nivel de arcos y nodos, respectivamente.

Algoritmo 8.1 Función forward de capa convolucional GNN

```
def forward(self, graph, h):
  with graph.local_scope():
    # Guardamos por separado el estado temporal y la información de
        entrada de la primera fase expandida
    graph.edata['h_e'], graph.edata['input'], graph.ndata['input'] = h

    # Lanza la actualización de la información en los nodos. Usa función
        sum para la agregación.
    graph.update_all(self.upd_all, fn.sum('v', 'h'))

    s = graph.ndata['h']

    # Concatenación de todos los datos de entrada
    score = self.wrapper.cat([
      graph.ndata['h'],
      graph.ndata['input'],
      graph.ndata['lat'][:,None],
      graph.ndata['lon'][:,None],
      graph.ndata['inputLanes'][:,None],
      graph.ndata['outputLanes'][:,None]
    ], 1)

    # La información pasa por una serie de capas lineales previamente a
        propagar a los arcos
    for l in self.linear_reduction:
      score = l(score)

    graph.ndata['h'] = score

    # Lanza la actualización de la propagación a los arcos
    graph.apply_edges(self.apply_edges)

    # Retorna tanto la salida final a nivel de arco, como las intermedias
        de nodos y arcos
    return graph.edata['h_e'], score, s
```

Función actualizadora de nodos

La función mostrada en 8.2 especifica cómo se lleva a cabo la actualización de los valores en los arcos. Tras tomar la entrada recibida de la función *forward*, realiza la concatenación con el resto de datos que pueden resultar de interés, como la velocidad, el número de carriles o la ubicación, así como el grado de los nodos extremos de cada arco. Todo ello se hace pasar por una red neuronal de varias capas lineales con cuyo ajuste de parámetros se logrará el aprendizaje, es decir, ajustar qué valores de la información de entrada son más importantes de cara a ofrecer una salida lo más cercana posible al *ground truth*.

Es importante ser consciente de que el paso por la red neuronal se realiza una vez por cada uno de los arcos de la red, de forma paralelizada gracias a DGL. Esta puntuación por arco será la que la función *forward* agregue con una función (en este caso *sum*), para generar una serie de valores únicos para cada nodo.

Algoritmo 8.2 Función upd_all de capa convolucional GNN

```
def upd_all(self, edges):
  # Recuperamos el valor almacenado en forward para usarlo junto al resto de caracte
  h_e = edges.data['h_e']

  # Concatenamos toda la información a contemplar
  score = self.wrapper.cat([
    h_e,
    edges.data['speed_kph'][:,None],
    edges.data['lanes'][:,None],
        edges.data['length'][:,None],
        (edges.src['lat'] - edges.dst['lat'])[:,None],
        (edges.src['lon'] - edges.dst['lon'])[:,None],
        edges.data['origin_in_degree'][:,None],
        edges.data['origin_out_degree'][:,None],
        edges.data['dest_in_degree'][:,None],
        edges.data['dest_out_degree'][:,None],
        edges.data['input']
  ], 1)

  # Procesamiento en serie de capas lineales
  for l in self.linear_nodes:
    score = l(score)

  # El retorno es una serie de valores para cada arco (parametrizable),
  # que será asignada a la variable indicada para proceder a la agregación
  return {'v': score}
```

Función propagadora a arcos

En el último paso de la capa convolucional, mostrado en la función 8.3, se programa cómo se realiza la actualización del valor de cada uno de los arcos basándose en la información recién calculada de los nodos que conecta. La estructura es muy similar a la anterior función, puesto que toma los valores calculados para los nodos extremo del arco y concatena una serie de datos de interés como la velocidad del arco, el número de carriles, la ubicación, etc.; todo ello se procesa con una nueva red neuronal que genera un valor actualizado para cada arco, que será la salida final de la capa convolucional.

De igual modo que en la anterior, se produce un paralelizado de los cálculos, puesto que la red neuronal debe ocuparse de transformar la información relativa a todos los arcos de la red.

Algoritmo 8.3 Función apply_edges de capa convolucional GNN

```
def apply_edges(self, edges):
  h_u = edges.src['h']
  h_v = edges.dst['h']

  # Pasamos al modelo como entrada:
  # 1- Características del nodo origen
  # 2- Características del nodo destino
  # 3- Características físicas del arco
  score = self.wrapper.cat([
    h_u,
    h_v,
    edges.data['speed_kph'][:,None],
    edges.data['lanes'][:,None],
    edges.data['length'][:,None],
    (edges.src['lat'] - edges.dst['lat'])[:,None],
    (edges.src['lon'] - edges.dst['lon'])[:,None],
    edges.data['origin_in_degree'][:,None],
    edges.data['origin_out_degree'][:,None],
    edges.data['dest_in_degree'][:,None],
    edges.data['dest_out_degree'][:,None],
    edges.data['input']
  ], 1)

  for l in self.linear_edges:
    score = l(score)

  # El retorno es una serie de valores para cada arco (parametrizable),
  # que será asignada a la variable indicada dentro de 'edata'
  return {'h_e': score}
```

8.3.2. Histórico de tráfico

Usando el código desarrollado dentro del trabajo, es posible exportar los datos procesados en un único fichero que contiene un elemento por cada punto de medición e instante de medida, es decir, tantas filas como mediciones unitarias se han tomado del origen de datos para el rango temporal especificado. Se trata de un archivo separado por comas (CSV) llamado processed.csv y ubicado en el directorio data/ready del proyecto, y cuenta con la siguiente estructura:

- id: Identificador del punto de medida.
- lunes
- martes
- miercoles
- jueves
- viernes
- sabado
- dia
- mes
- anyo
- hora
- minuto
- festivo: Flag de si se trata de un día festivo.
- carga: Porcentaje de uso sobre el total de la capacidad de la vía, corresponde al *ground truth*.

8.4. Resumen

El estudio realizado ofrece una gran cantidad de vías de trabajo para mejorar su precisión o incluir una funcionalidad mayor. Contar con una mayor cantidad de información de entrada permitiría estimaciones más cercanas a la realidad, dotando al producto de una potencia mayor. No obstante, se trata de un producto usable en su estado actual, dado que las pruebas realizadas han sido satisfactorias en cuanto a rendimiento de cálculo.

Una vez finalizado el trabajo, destacar el hecho de haber llegado al objetivo planteado en un tiempo cercano al estimado, cubriendo el alcance requerido. No es menos cierto que el principal

problema de la estructura generada aparece cuando se tratan de apilar muchas capas, debido a que la complejidad de los cálculos hace necesarios unos recursos de cálculo y memoria muy altos, que se incrementan en gran medida con cada convolución incluida en el modelo.

Se trata de un sistema altamente parametrizable y para el que ejecutar pruebas de concepto con diferentes configuraciones es muy sencillo y rápido, además de reproducible por la asignación de semillas de números aleatorios. Permite además la generación de un fichero con los datos procesados, para su uso posterior en este u otro sistema que se desee desarrollar.

Bibliografía

[Ayuntamiento de Madrid, a] Ayuntamiento de Madrid. *Calendario laboral*. https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=9f710c96da3f9510VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

[Ayuntamiento de Madrid, b] Ayuntamiento de Madrid. *Tráfico. histórico de datos del tráfico desde 2013*. https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=33cb30c367e78410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

[Ayuntamiento de Madrid, c] Ayuntamiento de Madrid. *Tráfico. ubicación de los puntos de medida del tráfico*. https://datos.madrid.es/portal/site/egob/menuitem.c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnextoid=ee941ce6ba6d3410VgnVCM1000000b205a0aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD

[Boeing, 2017] Boeing, G. (2017). Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65, 126–139. https://doi.org/https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[Cappart et al., 2021] Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., & Velickovic, P. (2021). Combinatorial optimization and reasoning with graph neural networks. *CoRR*, abs/2102.09544. https://arxiv.org/abs/2102.09544

[Chen et al.,] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., & Zhang, Z. *A truly open source deep learning framework suited for flexible research prototyping and production.* https://mxnet.apache.org/

[Cui et al., 2018] Cui, Z., Henrickson, K., Ke, R., & Wang, Y. (2018). High-order graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *CoRR*, abs/1802.07007. http://arxiv.org/abs/1802.07007

[Drew, 1968] Drew, D. R. (1968). *Traffic Flow Theory and Control*.

[Flask,] Flask. *The python micro framework for building web applications*. https://palletsprojects.com/p/flask/

[Han & Moutarde, 2012] Han, Y. & Moutarde, F. (2012). Statistical traffic state analysis in large-scale transportation networks using locality-preserving non-negative matrix factorization. *CoRR*, abs/1212.5264. http://arxiv.org/abs/1212.5264

[Huang et al., 2014] Huang, W., Song, G., Hong, H., & Xie, K. (2014). Deep architecture for traffic flow prediction: Deep belief networks with multitask learning. *IEEE Transactions on Intelligent Transportation Systems*, 15(5), 2191–2201. https://doi.org/10.1109/TITS.2014.2311123

[Li et al., 2021] Li, F., Feng, J., Yan, H., Jin, G., Jin, D., & Li, Y. (2021). Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. *CoRR*, abs/2104.14917. https://arxiv.org/abs/2104.14917

[Li et al., 2017] Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2017). Graph convolutional recurrent neural network: Data-driven traffic forecasting. *CoRR*, abs/1707.01926. http://arxiv.org/abs/1707.01926

[Lin et al., 2019] Lin, Y., Dai, X., Li, L., & Wang, F.-Y. (2019). Pattern sensitive prediction of traffic flow based on generative adversarial framework. *IEEE Transactions on Intelligent Transportation Systems*, 20(6), 2395–2400. https://doi.org/10.1109/TITS.2018.2857224

[Lint et al., 2002] Lint, J., Hoogendoorn, S., & Zuvlen, H. (2002). Freeway travel time prediction with state-space neural networks: Modeling state-space dynamics with recurrent neural networks. *Transportation Research Record*, 1811. https://doi.org/10.3141/1811-04

[Lv et al., 2015] Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F.-Y. (2015). Traffic flow prediction with big data: A deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2), 865–873. https://doi.org/10.1109/TITS.2014.2345663

[Ma et al., 2017] Ma, X., Dai, Z., He, Z., & Wang, Y. (2017). Learning traffic as images: A deep convolution neural network for large-scale transportation network speed prediction. *CoRR*, abs/1701.04245. http://arxiv.org/abs/1701.04245

[OpenStreetMap Contributors,] OpenStreetMap Contributors. *Mapa del mundo de uso libre bajo licencia abierta*. https://www.openstreetmap.org/

[PapersWithCode,] PapersWithCode. *Paperswithcode: Traffic prediction on pems-bay*. https://paperswithcode.com/sota/traffic-prediction-on-pems-bay

[Roy et al., 2021] Roy, A., Roy, K. K., Ali, A. A., Amin, M. A., & Rahman, A. K. M. M. (2021). SST-GNN: simplified spatio-temporal traffic forecasting model using graph neural network. *CoRR*, abs/2104.00055. https://arxiv.org/abs/2104.00055

[Wang et al.,] Wang, M., Zheng, D., Gan, G., Li, M., Ye, Z., Ma, C., Zhou, J., Song, X., Xiao, T., He, T., Zhang, J., ming Ye, W., Karypis, G., & Zhang, Z. *Deep graph library. easy deep learning on graphs.* https://www.dgl.ai/

[Wang et al., 2019] Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., & Zhang, Z. (2019). Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*.

[Wu et al., 2019] Wu, Z., Pan, S., Long, G., Jiang, J., & Zhang, C. (2019). *Graph wavenet for deep spatial-temporal graph modeling*. https://doi.org/10.48550/ARXIV.1906.00121

[Yin et al., 2002] Yin, H., Wong, S., Xu, J., & Wong, C. (2002). Urban traffic flow prediction using a fuzzy-neural approach. *Transportation Research Part C: Emerging Technologies*, 10(2), 85–98. https://doi.org/https://doi.org/10.1016/S0968-090X(01)00004-3

[Yoo et al., 2021] Yoo, B., Lee, J., Ju, J., Chung, S., Kim, S., & Choi, J. (2021). Conditional temporal neural processes with covariance loss. *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 12051–12061. https://proceedings.mlr.press/v139/yoo21b.html

[Yu et al., 2017] Yu, H., Wu, Z., Wang, S., Wang, Y., & Ma, X. (2017). Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks. *CoRR*, abs/1705.02699. http://arxiv.org/abs/1705.02699

[Zhang et al., 2016] Zhang, J., Zheng, Y., & Qi, D. (2016). Deep spatio-temporal residual networks for citywide crowd flows prediction. *CoRR*, abs/1610.00081. http://arxiv.org/abs/1610.00081

[Zheng et al., 2019] Zheng, C., Fan, X., Wang, C., & Qi, J. (2019). *Gman: A graph multi-attention network for traffic prediction*. https://doi.org/10.48550/ARXIV.1911.08415

[Zhou et al., 2018] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., & Sun, M. (2018). Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434. http://arxiv.org/abs/1812.08434

[Zhou, 2018] Zhou, T. (2018). *Deep Learning Models for Route Planning in Road Networks*. https://doi.org/10.13140/RG.2.2.30527.56486

Anexo A

Glosario de términos

- GNN : Graph Neural Network. Red neuronal diseñada para su aplicación sobre datos estructurados con forma de grafo.
- GraphML: Formato de fichero basado en XML diseñado para representación de información estructurada como grafo.
- UTM: Sistema de coordenadas universal transversal de Mercator (Universal Transverse Mercator). Está basado en la proyección cartográfica transvesa de Mercator, y permite que las magnitudes se expresen en metros.
- MSE : Mean Squared Error o error cuadrático medio. Mide la media del cuadrado de los errores generados o diferencia entre las estimaciones y los datos reales.
- Escalado de tipo MinMax : Transforma una característica de tal forma que todos los valores quedan en el rango [0, 1]. Se logra mediante la fórmula de escalado siguiente: X_scaled = (X - X_min) / (X_max - X_min). De este modo, el valor mínimo del juego de datos quedará como 0 y el máximo como 1, con los valores intermedios a una distancia proporcional a ellos según su cuantía original.
- JSON: Acrónimo de JavaScript Object Notation, es un formato de texto sencilo para el intercambio de datos.