# ESCUELA POLITÉCNICA NACIONAL

## FACULTAD DE CIENCIAS

# EVALUACIÓN Y APLICACIÓN DE DIFERENTES ESTRUCTURAS DE REDES NEURONALES PARA GRAFOS

# EVALUACIÓN COMPARATIVA DE ARQUITECTURAS GNN

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO**

**DIANA CAROLINA TANDALLA VARGAS**

diana.tandalla@epn.edu.ec

**DIRECTOR: MIGUEL ALFONSO FLORES SÁNCHEZ**

miguel.flores@epn.edu.ec

**DMQ, JULIO 2024**

**CERTIFICACIONES**

Yo, DIANA CAROLINA TANDALLA VARGAS, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Diana Carolina Tandalla Vargas

Certifico que el presente trabajo de integración curricular fue desarrollado por Diana Carolina Tandalla Vargas, bajo mi supervisión.

Miguel Alfonso Flores Sánchez
**DIRECTOR**

## DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Diana Carolina Tandalla Vargas

Miguel Alfonso Flores Sánchez

**RESUMEN**

Este estudio se enfoca en el desarrollo y comparación de modelos híbridos de redes neuronales de grafos combinados con redes neuronales recurrentes para la predicción del tráfico urbano, evaluando su capacidad para capturar dependencias espaciales y temporales. La capacidad de predecir el tráfico es esencial para que los gobiernos y empresas privadas implementen medidas como la planificación de rutas, la expansión del transporte público y la mejora de los servicios de navegación.

Utilizando la base de datos METR-LA, se preprocesaron los datos para normalizar las características del tráfico y estructurar la red de carreteras urbanas como un grafo. En este grafo, los nodos representan los sensores de tráfico, que pueden estar ubicados en intersecciones entre dos calles, y las aristas representan las carreteras que conectan las ubicaciones de estos sensores. Las características del tráfico, como la velocidad, el flujo y la ocupación, se representaron como atributos de los nodos. En este caso, se utilizó la velocidad como la característica principal para predecir el tráfico.

Se implementaron varios modelos híbridos de GNN, incluyendo GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM, que consideran tanto las dimensiones espaciales como temporales. Estos modelos se entrenaron y validaron utilizando técnicas de validación cruzada para asegurar la precisión y evitar el sobreajuste. Los resultados se evaluaron mediante métricas estándar como el error cuadrático medio, el error absoluto medio y el error cuadrático medio de la raíz.

***Palabras clave:*** Predicción del Tráfico, Redes Neuronales de Grafos, Modelos Híbridos, Modelado de Grafos.

**ABSTRACT**

This study focuses on the development and comparison of hybrid Graph Neural Network models combined with Recurrent Neural Networks for urban traffic prediction, evaluating their ability to capture spatial and temporal dependencies. The ability to predict traffic is essential for governments and private companies to implement measures such as route planning, public transport expansion, and the improvement of navigation services.

Using the METR-LA dataset, data preprocessing was carried out to normalize traffic characteristics and structure the urban road network as a graph. In this graph, the nodes represent traffic sensors, which may be located at intersections of two streets, and the edges represent the roads connecting these sensor locations. Traffic characteristics such as speed, flow, and occupancy were represented as node attributes. In this case, speed was used as the main feature to predict traffic.

Several hybrid GNN models were implemented, including GCN-GRU, GCN-LSTM, GAT-GRU, and GAT-LSTM, considering both spatial and temporal dimensions. These models were trained and validated using cross-validation techniques to ensure accuracy and avoid overfitting. The results were evaluated using standard metrics such as Mean Squared Error, Mean Absolute Error and Root Mean Squared Error.

***Keywords:*** Traffic prediction, Graph Neural Networks, Hybrid Models, Graph modeling.

**DEDICATORIA**

*A mi madre, Marjorie Vargas, a mi padre, Edison Tandalla, a mi hermana y a mi novio y compañero, Adrián Enriquez, por su apoyo incondicional y sus consejos. Su presencia en mi vida es invaluable, y este logro es fruto de nuestro trabajo en equipo.*

**AGRADECIMIENTO**

A mi madre, Marjorie Vargas, quiero expresar mi profundo agradecimiento por estar siempre a mi lado, por creer en mi potencial y por cuidarme con amor incondicional. Cada palabra de aliento, cada muestra de perseverancia y cada gesto de cariño que me has brindado son tesoros que atesoro con gratitud. Esta obra es un homenaje a ti, mi fuente inagotable de fortaleza y amor en mi búsqueda de conocimiento. Tu influencia ha dejado una huella imborrable en mi vida, y mi éxito académico es testimonio de tu incansable dedicación.

A mi padre, Edison Tandalla, quiero reconocer su invaluable lección de perseverancia, dedicación y responsabilidad. Gracias por proporcionarme los recursos necesarios y por orientarme con tus sabios consejos. Tu amor y guía han sido pilares fundamentales en mi búsqueda de conocimiento.

A mi hermana, agradezco su enseñanza sobre la importancia de dedicar lo mejor de mí cuando me apasiona algo.

A mi mejor amigo, mi novio, mi compañero, Adrián Enriquez, quiero agradecerle por saber escucharme, por brindarme su apoyo y sus consejos. No hay persona más importante para mí que tú.

A mis amigos, les agradezco profundamente por su apoyo incondicional durante las noches de estudio. Su confianza en mí, a pesar de las dificultades, y su sincera amistad han sido pilares fundamentales en este proceso. Gracias por estar siempre presentes y por ofrecerme su ánimo y compañía en los momentos más difíciles.

Además, expreso mi más sincero agradecimiento a el Dr. Miguel Flores. Su fe en mi potencial y las invaluables oportunidades que me ha brindado han sido cruciales para mi desarrollo. Le estoy inmensamente agradecido no solo por las enseñanzas profesionales, sino también por los valores que me ha inculcado. Su orientación y apoyo han sido determinantes para mi crecimiento tanto personal como profesional.

# Índice general

# Índice de figuras

# Capítulo 1

## Descripción del componente desarrollado

### 1.1. Motivación

La predicción espacio-temporal, crucial en entornos dinámicos como las redes de tráfico, necesita capturar patrones que varían tanto en el tiempo como en el espacio. La predicción precisa del tráfico proporciona múltiples beneficios, como ayudar a los ciudadanos a evitar rutas congestionadas y planificar viajes eficientemente, ahorrando tiempo y dinero. Este aspecto es fundamental para los sistemas de transporte autónomos.

Sin embargo, la detección y síntesis de las dependencias espacio-temporales es una tarea compleja. Los modelos de GNN han emergido como una herramienta prometedora para abordar estos desafíos. Las GNN son capaces de capturar la complejidad de las redes de tráfico al modelar las interacciones entre los nodos (ubicaciones de los sensores de tráfico) y las aristas (conexiones entre sensores, es decir, las calles que conectan estos sensores), incorporando tanto las dependencias espaciales como temporales en sus predicciones.

Al realizar este estudio, se busca aportar una comprensión más profunda de cómo las GNN pueden mejorar la predicción del tráfico, facilitando la toma de decisiones en la planificación urbana y la operación de sistemas de transporte inteligente (ITS).

## 1.2. Antecedentes

Tradicionalmente, los sistemas de predicción se dividen en dos categorías principales:

1. Métodos Estadísticos
2. Modelos de Aprendizaje Automático

Las GNN calculan representaciones vectoriales para cada nodo en un grafo, agregando iterativamente las características de los nodos vecinos. Esto permite capturar las relaciones espaciales entre los nodos, ya sea por conexiones directas o por distancias cortas en términos de saltos o peso total de los arcos.

Las GNN tienen aplicaciones en diversos campos, como redes sociales en ciencias sociales, redes de interacción entre proteínas en ciencias naturales y grafos de conocimiento. Estas aplicaciones incluyen clasificación de grafos, inferencia de nodos o arcos dentro de una red existente y regresión de características en nodos o arcos.

Zhou et al. (2018) [37] ofrecen un análisis detallado de las opciones y aplicabilidad de las GNN. Clasifican los grafos como dirigidos o no dirigidos, homogéneos o heterogéneos, y estáticos o dinámicos.

La función de pérdida en las GNN puede diseñarse en distintos niveles:

- Nodo: para tareas de regresión o clasificación de nodos, así como particionamiento de nodos en diferentes grupos en función de sus características individuales y su relación con otros nodos.
- Arco o Arista: para clasificación o predicción de arcos inexistentes, es decir, predecir las propiedades o la presencia de aristas en el grafo.
- Grafo: para problemas de clasificación o regresión de grafos, es decir, para predecir la propiedad de un grafo completo.

Desde la perspectiva de la supervisión, las GNN permiten realizar aprendizajes supervisados, semi-supervisados y no supervisados para la extracción de características.

El desarrollo de investigación del aprendizaje profundo durante los últimos años ha permitido que aumente el número de investigadores que apliquen redes neuronales profundas para la predicción de tráfico de alta precisión [16]. Por ejemplo, Huang et al. [13] emplean una Red de Creencia Profunda (DBN) para aprender características efectivas para la predicción del flujo de tráfico de manera no supervisada. Jia et al. [14] propusieron un modelo híbrido de DBN y Perceptrón Multicapa (MLP) para la predicción de velocidad. Lv et al. [19] aplican Autoencoder Apilado (SAE) para extraer características de tráfico para la predicción del flujo de tráfico. Aunque todos los enfoques de aprendizaje profundo mencionados anteriormente han obtenido buenos resultados, se centran principalmente en modelar una sola secuencia temporal, lo que limita su capacidad para considerar las dependencias espaciales en las redes de tráfico [35].

El siguiente paso para extraer la dependencia espacial de los datos de tráfico, las investigaciones introducen Redes Neuronales Convolucionales (CNN) en las tareas de predicción de tráfico. Ma et al. [20] propusieron un método basado en imágenes que trata las redes de tráfico como imágenes y utiliza CNN para aprender las características espaciales. Yu et al. [33] han mostrado un buen resultado combinando CNN con la red de Memoria a Corto y Largo Plazo (LSTM) para TSP. Wang et al. [28] implementaron un mecanismo de corrección de errores en sus modelos de CNN para abordar los desafíos predictivos causados por eventos de tráfico imprevistos.

Las CNN tradicionales están limitadas a procesar únicamente estructuras espaciales en forma de rejilla, como las imágenes. Sin embargo, los datos frecuentemente se recopilan en espacios no euclidianos, como los grafos. Para abordar este problema, se propone el Aprendizaje Profundo Geométrico (GDL) [5].

Las Redes Convolucionales de Grafos (GCN) son uno de sus desarrollos que generalizan CNN a dominios de grafos [15, 6]. Para problemas relacionados con datos de tráfico, GCN es ampliamente adoptado para manejar diversas tareas tratando las redes de tráfico como grafos que pueden aprovechar completamente la información espacial en el tráfico [18, 33, 17].

Li et al. [18] propusieron un modelo híbrido basado en GCN que captura la dependencia espacial con caminatas aleatorias en la red de tráfico y

la dependencia temporal con LSTM. Yu et al. [33] propusieron Redes Convolucionales Espacio-Temporales de Grafos (STGCN) que emplean estructuras convolucionales tanto en el eje espacial como temporal. Actualmente, los enfoques basados en GCN están entre las técnicas más avanzadas en la investigación de predicción de tráfico [17].

Diversos modelos se han desarrollado para la predicción del tráfico, abordando el problema desde diferentes perspectivas. Una línea emergente en esta área de investigación es la utilización de imágenes satelitales para obtener mediciones, en lugar de depender únicamente de datos tabulares o sensores. Sin embargo, en este trabajo se centra en evaluar el rendimiento de algunos modelos híbridos en la predicción del tráfico sobre datos tabulares.

## 1.3. Objetivo general

Desarrollar y comparar modelos de redes neuronales de grafos para la predicción precisa del tráfico urbano, evaluando su capacidad para capturar dependencias espaciales y temporales, con el propósito de mejorar la comprensión de estos modelos en el contexto de la predicción de tráfico.

## 1.4. Objetivos específicos

1. Implementar diferentes arquitecturas de redes neuronales de grafos para la predicción de tráfico.
2. Comparar el rendimiento de los diferentes modelos de redes neuronales de grafos en términos de precisión y eficiencia computacional en conjuntos de datos históricos y en tiempo real.
3. Interpretar los resultados y proporcionar fortalezas y debilidades de cada uno de los modelos.

## 1.5. Alcance

La capacidad de predecir el tráfico urbano es esencial tanto para gobiernos como para empresas privadas, permitiéndoles implementar medidas que favorecen un crecimiento ordenado, como la planificación de rutas y la mejora de servicios de navegación. En este caso, predicciones precisas son fundamentales para los ITS, ya que optimizan el control vehicular y la eficiencia de las redes viales.

Uno de los principales desafíos en la predicción del tráfico es anticipar el flujo y la velocidad en las autopistas. Los sensores de vehículos ubicados en las carreteras urbanas pueden representarse como un grafo, utilizando la distancia euclidiana para medir las conexiones y características del tráfico como atributos de los nodos.

Los datos de tráfico presentan una variabilidad temporal y características físicas que afectan el flujo vehicular. Las GNN son adecuadas para este problema, ya que escalan linealmente con el número de características y nodos. Los sensores de tráfico contienen complejas correlaciones espaciales, haciendo que la estructura del tráfico sea más direccional que euclidiana.

Este trabajo compara diversos modelos de GNN para la predicción del tráfico, evaluando su precisión y eficiencia en la captura de dependencias espaciales y temporales, utilizando una base de datos de tráfico reconocida. La comparación proporciona una visión integral sobre la aplicabilidad y rendimiento de estos modelos en entornos urbanos complejos.

# Capítulo 2

## Marco teórico

Este capítulo tiene como objetivo proporcionar al lector los fundamentos clave de las redes neuronales y su aplicación en el aprendizaje del tráfico urbano utilizando Graph Neural Networks (GNN). Esto permitirá comprender las bases teóricas de la metodología utilizada en la sección 3. En primer lugar, en la sección 2.1, se abordarán los conceptos básicos de las redes neuronales, incluyendo diferentes tipos como redes neuronales de grafos (2.1.1), redes neuronales convolucionales (2.1.2), redes neuronales de atención de grafos (2.1.3), y redes neuronales recurrentes como LSTM y GRU (2.1.4). Después, en la sección 2.2, se dará una introducción al aprendizaje del tráfico utilizando GNNs, proporcionando una visión detallada sobre cómo estas redes pueden modelar y predecir patrones de tráfico. Finalmente, en la sección 2.3, se evaluarán las métricas para modelos de regresión en aprendizaje automático, las cuales son esenciales para medir el rendimiento y la precisión de los modelos desarrollados.

## 2.1. Red Neuronal

Una red neuronal es un modelo computacional inspirado en la estructura y funcionamiento del cerebro humano. Está compuesta por unidades básicas llamadas neuronas artificiales o nodos, organizadas en capas. Cada nodo en una capa está conectado a los nodos de la siguiente

capa mediante pesos. Las redes neuronales se utilizan para una variedad de tareas, incluyendo la clasificación, la regresión, el reconocimiento de patrones, y más [3].

El funcionamiento de una red neuronal comienza con la entrada de datos a la capa de entrada. Los datos se multiplican por los pesos y se pasan a través de una función de activación, que introduce no linealidad y permite a la red aprender relaciones complejas. Las funciones de activación comunes incluyen la sigmoide, ReLU y tangente hiperbólica. Esta operación se realiza en cada capa de la red, propagando la información desde la capa de entrada hasta la capa de salida [3].

Una vez que los datos han pasado por todas las capas, la salida de la red se compara con la salida esperada utilizando una función de pérdida o costo, como el error cuadrático medio o la entropía cruzada. Este cálculo del error es crucial para el proceso de aprendizaje.

Los gradientes del error se desarrollan utilizando el algoritmo de retropropagación y sirven para ajustar los pesos en la dirección que minimiza el error, mediante un algoritmo de optimización como el gradiente descendente [3].

Este proceso de ajuste de pesos se repite iterativamente a través de muchas épocas, hasta que la red neuronal converge a una solución donde el error es mínimo. A través de este proceso, la red neuronal aprende a realizar tareas complejas, como la clasificación de imágenes [3].

En la figura 2.1 se puede ver el funcionamiento de una red neuronal

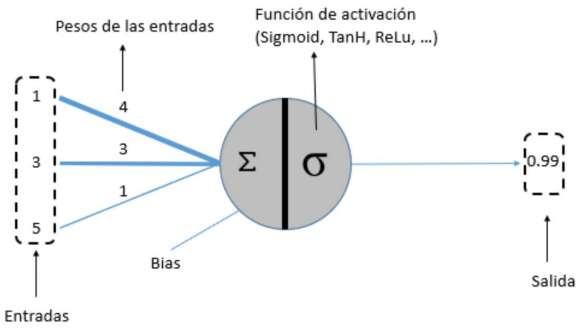



Figura 2.1: Ejemplo de una red neuronal [12]

Los hiperparámetros de una red neuronal son variables cruciales que

definen su arquitectura y comportamiento durante el entrenamiento y la inferencia. Estos parámetros, configurados antes del entrenamiento, incluyen la tasa de aprendizaje, el número de épocas, el tamaño del lote (batch size) y la configuración de las capas. La tasa de aprendizaje controla la velocidad de aprendizaje del modelo; una tasa muy alta puede causar inestabilidad, mientras que una tasa baja puede ralentizar el proceso. El número de épocas determina cuántas veces el algoritmo trabaja con el conjunto completo de datos; demasiadas épocas pueden llevar a sobreajuste y pocas pueden resultar en subajuste [11]. El tamaño del lote afecta la precisión de la estimación del gradiente durante el entrenamiento, con lotes más grandes proporcionando una estimación más estable pero a un costo computacional más alto. Finalmente, la configuración de las capas, que incluye el número de neuronas por capa y la profundidad de la red, es fundamental para captar la complejidad de los datos. La optimización de estos hiperparámetros es esencial para maximizar el rendimiento del modelo.

Técnicas como la validación cruzada, la búsqueda en cuadrícula y la búsqueda aleatoria son comúnmente empleadas para su ajuste. Este afinamiento de los hiperparámetros permite que la red neuronal se adapte mejor a la tarea específica, mejorando significativamente la precisión y eficiencia del modelo [11].

### 2.1.1. Red neuronal de grafos

Las Graph Neural Networks (GNN), o redes neuronales basadas en grafos [30], se desarrollan con el propósito de aprovechar la capacidad de los grafos para representar información en redes neuronales. A diferencia de las redes neuronales convencionales, que reciben como entrada datos codificados en vectores, matrices o tensores, las GNN están diseñadas específicamente para trabajar directamente con grafos. Esta característica distintiva permite a las GNN aprender de datos más complejos, ya que no solo consideran números individuales, sino también las relaciones entre estos [10].

El principio fundamental de una GNN se basa en la propagación de mensajes [9]. En una GNN, los nodos representan entidades y las aristas

representan relaciones entre estas entidades. La clave es que los nodos actualizan sus representaciones (o embeddings) iterativamente al combinar su información con la de sus vecinos, siguiendo las conexiones del gráfico. Esto se hace mediante una serie de pasos de agregación y actualización, permitiendo que la red capture y utilice tanto las características locales de los nodos como la estructura global del gráfico [10].

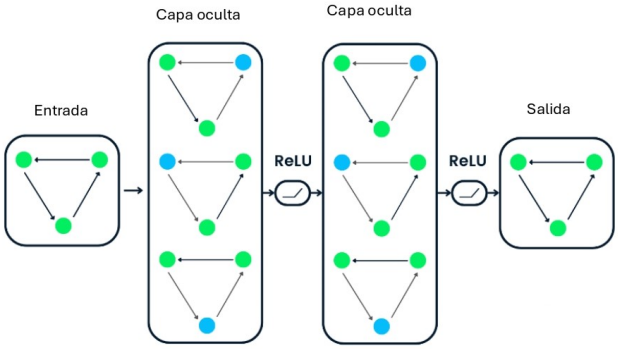



Figura 2.2: Ejemplo de estructura de un modelo GNN [2].

Una característica fundamental de las GNNs es su capacidad para realizar un aprendizaje de extremo a extremo. Esto significa que pueden aprender tanto las representaciones de los nodos como los parámetros de las capas de la red mediante la retropropagación del error. Esta capacidad de las GNNs para incorporar y aprovechar la estructura del grafo es lo que las hace poderosas y adecuadas para una amplia gama de tareas y aplicaciones que involucran datos en forma de grafos [10].

Las GNN pueden integrar la información de los vecinos de cada nodo. Cada nodo en el gráfico puede actualizar su representación basándose no solo en sus propias características, sino también en las características de sus nodos vecinos y las conexiones entre ellos. Esto se logra a través de procesos iterativos de agregación y actualización de información, que permiten que la representación de cada nodo capture tanto las características locales como el contexto global del gráfico. Este enfoque es especialmente útil en aplicaciones donde las interacciones entre entidades son críticas [10].

Además, las GNN son altamente flexibles y pueden aplicarse a grafos con diferentes estructuras y tamaños. No están limitadas a un número fijo de nodos o aristas y pueden manejar grafos dirigidos, no dirigidos,

ponderados, no ponderados, y más. Esta flexibilidad permite a las GNN generalizar a diferentes tipos de grafos y adaptarse a una variedad de aplicaciones y dominios, haciendo que sean una herramienta poderosa para el análisis de datos estructurados en forma de grafos [10].

### 2.1.2. Red neuronal convolucional

Las redes neuronales convolucionales (CNN) sobresalen en el manejo de datos visuales, de voz o señales de audio debido a su estructura especializada que incluye capas convolucionales, de agrupación y totalmente conectadas como se puede ver en la Figura 2.3 [36]. Inician con la capa convolucional que puede ser seguida por más capas del mismo tipo o de agrupación, culminando siempre en una capa totalmente conectada.

A lo largo de la red, la CNN incrementa su capacidad para detectar características, empezando por detalles simples como colores y bordes, avanzando hacia elementos más grandes, y finalmente reconociendo objetos completos en la información procesada [23].

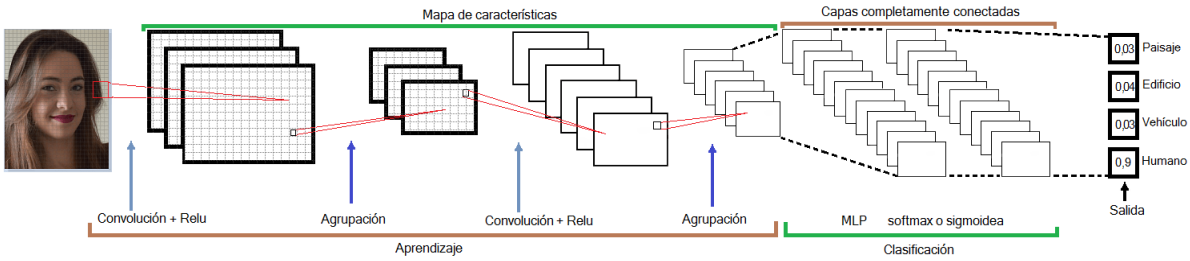



Figura 2.3: Ejemplo de estructura de un modelo CNN [1].

Para comprender el funcionamiento de una red neuronal convolucional, es fundamental conocer algunos conceptos clave: convolución, subsampling, filtro o kernel, padding, stride, pooling y fully connected. A continuación, se ofrece una breve descripción de estos conceptos:

1. **Convolución:** Es la operación principal en una CNN. Implica mover un filtro (o kernel) sobre la entrada para producir un mapa de características que resalta las características importantes de la imagen, como bordes o texturas. La convolución se realiza desplazando el filtro sobre la imagen de entrada en pasos definidos por el *stride*, calculando productos punto entre el filtro y pequeñas porciones de

la imagen, sumando los resultados para obtener un valor único en el mapa de características [32].

2. **Filtro o Kernel:** Es una pequeña matriz de pesos que se desliza sobre la entrada durante la convolución. Su objetivo es detectar patrones específicos en la entrada, como líneas o bordes. Cada posición del filtro sobre la imagen de entrada genera un valor en el mapa de características resultante [32].

3. **Padding:** Es el proceso de añadir píxeles adicionales alrededor de la entrada antes de aplicar la convolución. Esto ayuda a controlar el tamaño de la salida y puede ser útil para preservar la información en los bordes de la imagen. Existen diferentes tipos de padding, como el *zero-padding*, que añade ceros alrededor de la imagen [32].

4. **Stride:** Es el número de píxeles que el filtro se desplaza sobre la entrada en cada paso. Un stride mayor reduce el tamaño del mapa de características resultante, mientras que un stride menor proporciona una resolución más alta [32].

5. **Pooling:** Es una operación que reduce las dimensiones del mapa de características y resalta las características más importantes [32]. Hay dos tipos principales de pooling: Max Pooling selecciona el valor máximo de una región específica del mapa de características, preservando las características más prominentes, mientras que Average Pooling calcula el valor promedio de una región, proporcionando una representación suavizada de las características.

6. **Capa totalmente conectada:** Es una capa en la cual cada neurona está conectada a todas las neuronas de la capa anterior. Estas capas se utilizan al final de la CNN para realizar la clasificación o regresión basándose en las características extraídas por las capas convolucionales y de pooling [32].

Para mayor información, consulte el artículo completo *A review of convolutional neural networks in computer vision* [32].

**Red neuronal convolucional de grafos (GCN)**

Este tipo de red es una variante específica de las redes neuronales convolucionales, diseñada para manejar datos estructurados en forma de grafos. A diferencia de las CNNs tradicionales que operan con datos como imágenes, estas redes procesan directamente la información contenida en grafos[15]. Una GCN logra esto difundiendo y combinando información a través de las aristas del grafo en cada capa de la red.

Recordemos que un grafo está compuesto por nodos, aristas, características de los nodos y características de las aristas. Los componentes clave para un modelo GCN son:

- Matriz de adyacencia **A**: donde $A_{ij}$ representa la conexión o no entre los nodos $i$ y $j$.
- Matriz de características **X**: donde $X_i$ representa el vector de características del nodo $i$.

La operación de convolución en grafos se define en términos de los nodos y sus vecinos, y básicamente busca extraer y utilizar la información de los nodos conectados [15]. Los pasos a seguir son:

1. **Normalización de la matriz de adyacencia:** Este paso es crucial para estabilizar el entrenamiento y mantener la escala de las características.

   $$\hat{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

   donde $\hat{A}$ es la matriz de adyacencia normalizada y $D$ es una matriz diagonal donde $D_{ii} = \sum_j A_{ij}$.

2. **Agregación de vecinos:** Se agrega a cada nodo la información de sus vecinos utilizando la matriz de características:

   $$H^{(l+1)} = \sigma\left((\hat{A} + I)H^{(l)}W^{(l)}\right)$$

   donde $H^{(l)}$ es la matriz de características en la capa $l$ (con $H^{(0)} = X$), $W^{(l)}$ son los pesos de la capa $l$, $\sigma$ es una función de activación y $\hat{A}+I$ es la matriz de adyacencia con la identidad añadida para incluir las características del propio nodo.

La arquitectura típica de una GCN consiste en:

1. **Capa de entrada:** Está compuesta por representaciones iniciales de los nodos, que a menudo se denominan características de los nodos o atributos de los nodos. Por tanto, se tiene la matriz de características $X$ y matriz de adyacencia $A$.

2. **Capas convolucionales de grafos:** Aplican la operación de convolución a las características de los nodos.

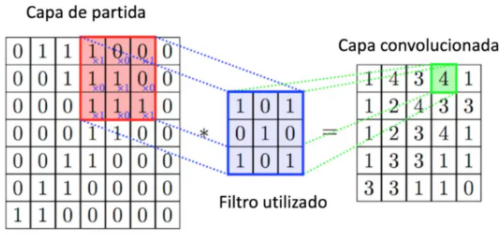



Figura 2.4: Ejemplo de convolución [7].

   Es decir, en cada capa convolucional de una GNN, los nodos intercambian mensajes con sus vecinos, combinando las representaciones de estos con la del nodo actual para actualizarla mediante una convolución basada en la estructura del grafo. Luego, se realiza una agregación de información donde se combinan las representaciones actualizadas de los nodos vecinos con la del nodo actual, obteniendo así una nueva representación del nodo.

   En resumen, una capa convolucional de grafo en una GCN realiza agregación, normalización, transformación lineal y aplicación de una función de activación para aprender nuevas representaciones de los nodos que capturan tanto sus características propias como la información estructural del grafo [10].

3. **Capas adicionales:** El proceso de la capa convolucional se repite en varias capas para capturar información de mayor alcance y mas abstracta dentro del grafo. Cada una de estas capas puede tener parámetros de aprendizaje que se ajustan durante el proceso de entrenamiento [10].

4. **Capas totalmente conectadas:** Combinan las características extraídas para tareas como la clasificación o la regresión.

5. **Capa de salida:** Produce las predicciones, como etiquetas de clasificación para nodos o aristas.

En las redes convolucionales de grafos, los hiperparámetros juegan un papel fundamental en la configuración de la arquitectura y el rendimiento del modelo. Entre los más importantes se encuentran la tasa de aprendizaje, que determina la velocidad con la que el modelo actualiza los pesos durante el entrenamiento; el tamaño del lote, que influye en la estabilidad de la convergencia y la eficiencia del entrenamiento; y el número de capas y características por capa, que configuran la profundidad y la capacidad del modelo para captar patrones complejos en los datos. Otro hiperparámetro crítico es el tipo de función de activación, como ReLU o sigmoid, que introduce no linealidades en el modelo y afecta su capacidad para aprender relaciones complejas [15].

### 2.1.3. Red neuronal de atención de grafos

GAT extiende GCN incorporando un mecanismo de atención explícito. Siguiendo una estrategia de auto-atención [26]. GAT reemplaza la operación de convolución anterior en la convolución de grafos con un mecanismo de atención. Para ilustrar mejor cómo se actualizan las características de los nodos de la capa $l$ a las de la capa $l+1$, describiremos paso a paso el proceso de atención:

1. Primero introducimos el componente constituyente de GAT, es decir, la capa de atención gráfica. La entrada a una capa GAT es un conjunto de características de nodos, digamos $h_l = \{h_1^l, h_2^l, \ldots, h_N^l\}$, $h_i^l \in \mathbb{R}^F$, donde $N$ es el número de nodos y $F$ es el número de características de cada nodo.

2. Para transformar las características de entrada, se utiliza una matriz de pesos compartida, $W \in \mathbb{R}^{F \times F}$ y de esta manera proyectar la entra a otro espacio de características de dimensión $F$.

3. El siguiente paso es definir un mecanismo de atención y de este modo calcular el coeficiente de atención de los nodos y sus vecinos:

$$e_{ij} = a(W\vec{h_i}, W\vec{h_j}), \quad a : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R},$$

donde $a(\cdot,\cdot)$ es el mecanismo de autoatención, $e_{ij}$ es el coeficiente de atención calculado. Como observación solo se toman encuenta los vecinos directos del nodo [26].

4. Finalmente, se utiliza una función softmax para normalizar los coeficientes de atención en una forma fácilmente comparable. Por último, se aplica una función de activación Leaky Rectified Linear Units (LeakyReLU) [29] y se obtiene el coeficiente de atención normalizado

$$\alpha_{ij} = softmax(LeakyReLU(e_ij)).$$

y con estos coeficientes se actualizan las características del modelo utilizando la regla de convolución:

$$h_i^{l+1} = \sigma\left(\sum_{j\in N(i)} \alpha_{ij} W^l h_j^l\right)$$

donde $N(i)$ corresponde al conjunto de nodos adyacentes que son inmediatos del nodo $i$.

En la Figura 2.5 se puede ver el diagrama de un mecanismo de atención empleado al nodo $i$ con el nodo $j$.

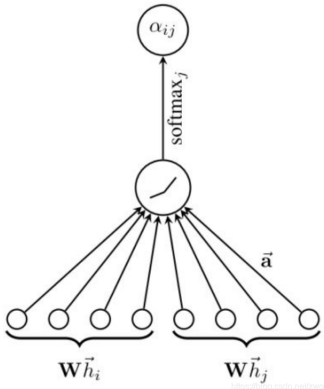



Figura 2.5: Mecanismo de atención $a(W\vec{h_i}, W\vec{h_j})$ empleado en el modelo GAT, parametrizado por un vector de pesos $\vec{a} \in \mathbb{R}^{2F}$, aplicando la función de activación LeankyReLU [27].

Agregando a este mecanismo de atención el modelo GAT propone el mecanismo de atención multicabeza el cual permite que el modelo apren-

da un coeficiente de atención a través de múltiples subespacios de representación. Para hacer el proceso de aprendizaje de auto-atención robusto, generalmente se adoptan estrategias de mecanismo de atención multi-cabeza [26, 29]. Específicamente, tomando como ejemplo el mecanismo de atención multi-cabeza adoptado en [27], $K$ mecanismos de atención independientes realizan la transformación mencionada a través de $K$ cabezas, es decir, $K$ procesos de atención independientes y sus características resultantes se concatenan juntas para desarrollar una representación de características de salida. Posteriormente, la salida final se obtiene promediando la concatenación de la representación de características [35]. Este proceso se define formalmente como:

$$\mathbf{h}_i^{l+1} = \begin{cases} \sigma\left(\|_{K=1}^{K}\left(\sum_{j\in N(i)} \alpha_{ij}^{K}\phi^{K}\mathbf{h}_j^{l}\right)\right), & \text{Concatenación} \\ \sigma\left(\frac{1}{K}\sum_{K=1}^{K}\sum_{j\in N(i)} \alpha_{ij}^{K}\phi^{K}\mathbf{h}_j^{l}\right), & \text{Promediado} \end{cases} \tag{2.1}$$

lo cual se puede observar en la Figura 2,6.

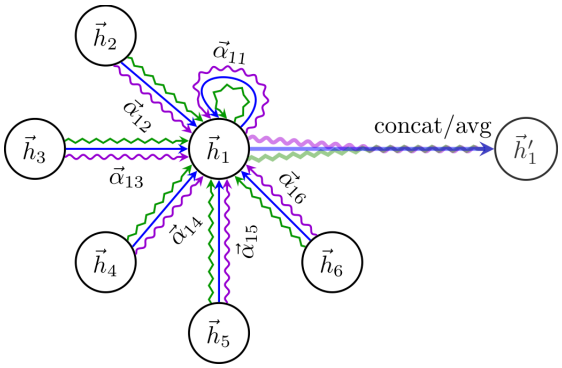



Figura 2.6: Ilustración de la atención multicabeza (con $K = 3$ cabezas) para el nodo 1 sobre sus vecinos. Diferentes estilos de flechas y colores denotan cálculos de atención independientes. Las características agregadas de cada cabeza se concatenan o promedian para obtener $\vec{h_1}$ [27].

Algunos de los hiperparámetros más importantes incluyen el tamaño de las características de cada nodo, el número de cabezas de atención en cada capa de atención, y la tasa de aprendizaje. El tamaño de las características determina la dimensión de los vectores de entrada que representan los nodos, afectando directamente la capacidad del modelo para procesar y diferenciar entre información compleja de los nodos. Las cabezas de atención múltiple permiten que el modelo atienda a diferentes subespacios de información simultáneamente, mejorando la capacidad

de la red para capturar diversas relaciones entre nodos de manera eficiente. La tasa de aprendizaje regula la velocidad a la que se actualizan los pesos de la red durante el entrenamiento, siendo esencial para alcanzar una convergencia óptima sin caer en mínimos locales o desestabilizar el aprendizaje. Además, la elección del número de capas y el método de regularización, como el dropout, son hiperparámetros que influyen significativamente en la generalización del modelo y en la prevención del sobreajuste [35].

### 2.1.4. Red neuronal recurrente: LSTM y GRU

Las redes neuronales recurrentes (RNN) son un tipo de red neuronal artificial creada para procesar datos secuenciales o series temporales [23].

A diferencia de las redes neuronales convencionales, las RNN poseen la capacidad de "recordar" información de entradas previas, permitiéndoles influenciar tanto las entradas como las salidas actuales. Mientras que las redes neuronales profundas convencionales procesan las entradas y salidas de manera aislada, las RNN toman en cuenta la interdependencia de los elementos anteriores en la secuencia. Sin embargo, las RNN unidireccionales tienen la limitación de no poder considerar eventos futuros para mejorar sus predicciones [23].

El entrenamiento de RNN es intensivo en términos de tiempo y memoria debido al procesamiento de cada paso temporal de la secuencia. Para manejar esto, la red se "desenrolla" en múltiples capas, equivalentes al número de pasos temporales, tratándola como una red feed-forward con los mismos pesos en cada capa, lo que acelera el entrenamiento [22].

Sin embargo, aumentar la longitud de la secuencia incrementa el número de capas, lo que puede causar el problema de desvanecimiento del gradiente. Para mitigar este efecto, se integran capas LSTM o GRU, que facilitan la retropropagación a través del tiempo, conectando eventos distantes en los datos de entrada y asegurando la efectividad de los pesos a lo largo de las capas [22].

**Red Neuronal Recurrente Long Short Term Memory**

Las LSTM son un tipo de RNN, pero se diferencian de otras RNN en su funcionamiento. Mientras que en las RNN tradicionales el módulo se repite con cada nueva entrada de información, las LSTM pueden retener información relevante durante períodos más largos. Esto se debe a su estructura en forma de cadena que permite la repetición del módulo, interactuando de una manera específica mediante cuatro capas de red neuronal [8].

El proceso de transferencia de datos en LSTM es similar al de las RNN estándar, pero la propagación de información se maneja de manera diferente. En una LSTM, se utiliza un mecanismo de células y puertas para decidir qué información procesar y cuál descartar. El estado celular funciona como una memoria, actuando como una vía para transferir información a lo largo del tiempo [8].

La Red Neuronal Recurrente LSTM introduce un conjunto de puertas y estados de celda que gestionan el flujo de información, resolviendo así el problema del desvanecimiento del gradiente en la predicción de series temporales a largo plazo. Los estados de celda son esenciales en los LSTM ya que almacenan y transfieren información a lo largo de todas las iteraciones temporales. Las unidades de compuerta incluyen tres tipos: la compuerta de entrada, la compuerta de olvido y la compuerta de salida. Estas compuertas deciden si agregar o eliminar información de un estado de celda [35].

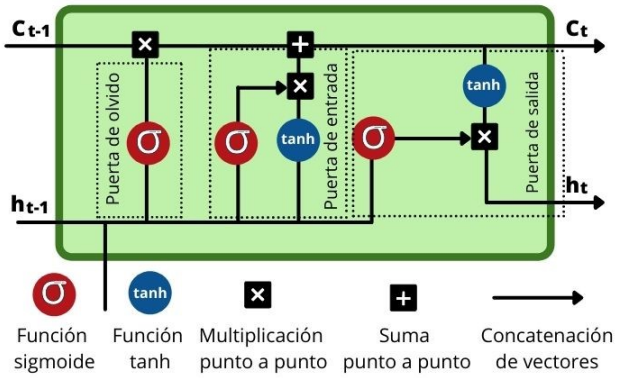



Figura 2.7: Estructura de un modelo LSTM. El diagrama muestra la estructura básica de un modelo LSTM [34].

El proceso de una LSTM, cuando $f_t = 1$, significa que la memoria a corto plazo se retiene completamente. Después de que los datos de ruido se ingresan, si pueden almacenarse en la celda depende de la compuerta de entrada, y la salida de la compuerta de entrada es $C_t$ como en la fórmula 2.2.

$$f_t = \sigma(W_f[h_{t-1}, n_t]) + b_f, \quad f_t \in [0, 1] \tag{2.2}$$

$$C_t = f_t \cdot C_{t-1} + \sigma(W_i[h_{t-1}, n_t] + b_i) \cdot tanh(W_C[h_{t-1}, n_t] + b_C) \tag{2.3}$$

En este caso $n_t$ representa el ruido de entrada de la capa actual, $h_{t-1}$ es el ruido de salida de la capa anterior y el estado oculto de la capa actual. La fórmula anterior representa el estado de la nueva celda después de descartar información inútil y retener algo de nueva información, donde $i_t = \sigma(W_i[h_{t-1}, n_t] + b_i)$ e $i_t$ representa la probabilidad de que se retenga la nueva información. El ruido de predicción de la salida depende de la compuerta de salida:

$$o_t = \sigma(W_o[h_{t-1}, n_t] + b_o) \tag{2.4}$$

$$Y_t = h_t = o_t \cdot tanh(C_t) \tag{2.5}$$

Aquí $o_t$ es la probabilidad de salida. Multiplicar $o_t$ y la función tangente hiperbólica $\tanh(C_t)$ tiene el propósito de controlar el filtrado del estado de la celda, y la salida $Y_t$ es el estado oculto de la siguiente capa. En la expresión anterior, $W_f$, $W_i$, $W_C$, $W_o$ son los vectores de peso de los parámetros de la función y $b_f$, $b_i$, $b_C$, $b_o$ son los vectores de sesgo.

Al configurar una LSTM, varios hiperparámetros son clave para su rendimiento. Uno de los más importantes es el número de unidades LSTM en cada capa, que determina la capacidad de la red para modelar la complejidad de la secuencia. Otro hiperparámetro crucial es la tasa de aprendizaje, que influye en la rapidez y efectividad con que la red converge durante el entrenamiento. Además, el tamaño del batch afecta directamente la estabilidad y la velocidad del proceso de aprendizaje, con tamaños de batch más grandes ofreciendo una estimación más robusta del gradiente pero aumentando el requerimiento computacional. Finalmente, el número de capas en la red puede ajustarse para aumentar la profundidad del modelo, permitiendo una representación más rica de las relaciones temporales, aunque a riesgo de sobreajuste. La optimización de estos hiperparámetros es esencial para lograr un equilibrio entre pre-

cisión y eficiencia en aplicaciones prácticas [8].

### Red de unidad recurrente cerrada

Las unidades de puertas recurrentes (GRU) son una variante de las RNN diseñadas para mejorar la retención de información a corto plazo, un problema común en las RNN tradicionales. Al igual que las LSTM, las GRU optimizan la retención de información mediante un mecanismo simplificado que utiliza estados ocultos en lugar del "estado de la celda" característico de las LSTM [23].

Las GRU emplean dos puertas: la puerta de restablecimiento y la puerta de actualización. La puerta de restablecimiento controla cuánto de la información pasada debe olvidarse, mientras que la puerta de actualización decide cuánta información nueva debe incorporarse al estado de la red. Esta configuración simplificada permite que las GRU sean más eficientes en términos de cálculo, especialmente en tareas donde no se requiere una regulación compleja de la información temporal [23].

Al no contar con celdas de memoria separadas, las GRU fusionan el almacenamiento y la transferencia de información en un único estado oculto, lo que las hace particularmente adecuadas para aplicaciones donde la velocidad de procesamiento y la menor complejidad arquitectónica son críticas. Aunque esta simplicidad puede resultar en una ligera disminución del control sobre la información comparado con las LSTM, en muchos casos, las GRU han demostrado ser igual de efectivas, especialmente en tareas de modelado de secuencias más cortas donde la gestión extensiva de la memoria a largo plazo no es tan crítica [23].

Segun [24] estructura de una red GRU está dada de la siguiente manera:

1. **Puerta de actualización-update:** primero se calcula la puerta de actualización $z_t$ para el paso de tiempo $t$:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

donde $W_z$ y $U_z$ son pesos asociados a la nueva entrada en este caso $x_t$ y a la información anterior $h_{t-1}$. La puerta de actualización en

las GRU permite transmitir al futuro la información relevante del pasado y combinarla con nueva información, evitando el problema del desvanecimiento del gradiente.

2. **Puerta restablecimiento:** esta puerta determina que cantidad de información debe olvidar. Su calculo esta dado por:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r),$$

donde $W_r$ y $U_r$ son pesos asociados a la nueva entrada en este caso $x_t$ y a la información anterior $h_{t-1}$.

Una vez calculado, el nuevo contenido de memoria que utilizará la puerta de reunión para almacenar información, se obtiene de la siguiente manera:

$$\widetilde{h_t} = tanh(W_n x_t + U_n(r_t \odot h_{t-1}) + b_n),$$

donde $W_r$ y $U_r$ son pesos asociados a la nueva entrada en este caso $x_t$ y a la información anterior $h_{t-1}$. Y por último la actualización del estado recurrente esta dado por

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h_t}.$$

Un ejemplo de esta implementación se puede ver en la Figura 2.8.

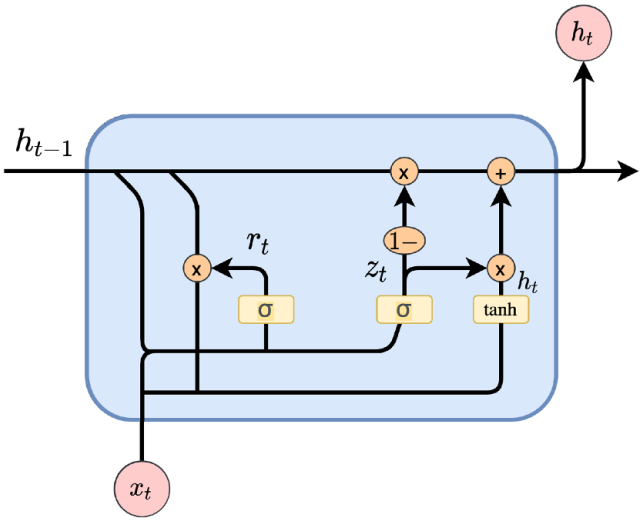



Figura 2.8: Estructura de unidad recurrente cerrada (GRU) [31].

Los hiperparámetros esenciales de una GRU incluyen el número de ca-

pas, el número de unidades por capa, la tasa de aprendizaje y el tamaño del lote (batch size). Más capas y unidades permiten al modelo capturar mejor la información en secuencias largas, aunque aumentan el riesgo de sobreajuste y los requerimientos computacionales. La tasa de aprendizaje afecta la rapidez con que el modelo se adapta durante el entrenamiento: tasas altas pueden causar inestabilidad y tasas bajas pueden ralentizar el proceso. El tamaño del lote influye en la estabilidad de la estimación del gradiente, afectando la velocidad de convergencia y la calidad del modelo final. También es crucial considerar la inicialización de los pesos y las funciones de activación, como la sigmoide para las puertas y la tangente hiperbólica para la transformación del estado oculto [23].

## 2.2. Aprendizaje del tráfico usando GNN

En el modelado de tráfico utilizando grafos, las redes son estructuralmente coherentes, sin nodos ni arcos aislados, y enfrentan variabilidad debido a la densidad fluctuante del tráfico y a las propiedades físicas de las vías, como su longitud y tipo. Las GNN son especialmente eficaces para abordar estos problemas, gracias a su habilidad para manejar un número creciente de características y la complejidad de la red, compuesta por numerosos nodos y arcos.

En este estudio, se han desarrollado modelos que combinan dos tipos de redes neuronales para analizar características espaciales y temporales. Para ello, se estudiaron tres artículos fundamentales que sirvieron de base para la implementación de los modelos. A continuación, se presenta una breve descripción de cada uno de ellos:

El artículo “T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction” propone una red convolucional de grafos temporal (T-GCN) para la predicción del tráfico [36]. La T-GCN combina la capacidad de las redes convolucionales de grafos (GCN) para capturar la estructura espacial de los datos con la capacidad de las redes recurrentes (GRU) para modelar dependencias temporales. Esta combinación permite una predicción más precisa del tráfico en redes viales complejas.

En “Long-Term Recurrent Convolutional Network (LRCN)”, se introdu-

ce una red convolucional recurrente a largo plazo (LRCN), que integra GCN con LSTM para abordar problemas de predicción temporal en series de tiempo [25]. La LRCN es capaz de capturar patrones temporales a largo plazo y ha demostrado ser efectiva en la predicción del tráfico al aprovechar la estructura espacial de las redes de transporte y las dinámicas temporales.

El trabajo "Spatial-Temporal Graph Attention Networks: A Deep Learning Approach for Traffic Forecasting" presenta una red de atención de grafos espacio-temporales (GAT), que incorpora mecanismos de atención para mejorar la predicción del tráfico [35]. La GAT es capaz de asignar diferentes pesos a diferentes componentes de la red de tráfico, permitiendo así una modelación más precisa de las interacciones espacio-temporales. Se implementaron dos variantes: una combinando GAT con GRU y otra con LSTM, mostrando mejoras significativas en comparación con métodos tradicionales.

Estos artículos proporcionaron una base sólida para la implementación de los modelos utilizados en este estudio, permitiendo explorar diferentes combinaciones de técnicas de aprendizaje profundo para mejorar la precisión en la predicción del tráfico urbano. La metodología detallada de la implementación se explica con mayor profundidad en el siguiente capítulo.

## 2.3. Evaluación de Métricas para Modelos de Regresión en Aprendizaje Automático

El análisis cuantitativo del rendimiento de los modelos de regresión es crucial en el campo del aprendizaje automático, especialmente cuando se trata de predecir valores continuos con alta precisión. Diversas métricas de regresión son implementadas para proporcionar una evaluación detallada y cuantitativa de la exactitud de un modelo. Este segmento profundiza en dos métricas fundamentales: el error cuadrático medio (MSE) y el error absoluto medio (MAE), ambas vitales para entender la eficacia de los modelos en cuestión [4].

Para la comparación de modelos en este estudio, se han seleccionado

tres métricas clave: MSE, RMSE, y MAE. Cada una de estas métricas tiene características únicas:

- **MSE**: Esta métrica evalúa el promedio de los cuadrados de los errores, que son las diferencias cuadradas entre los valores reales y las predicciones. Un aspecto crucial del MSE es que penaliza de manera más severa los errores grandes, pues al elevar al cuadrado las diferencias, los errores grandes incrementan exponencialmente. Esto hace que el MSE sea particularmente útil en situaciones donde los grandes errores son inaceptables [4].
- **RMSE**: Al tomar la raíz cuadrada del MSE, el RMSE transforma los valores de error a la misma unidad de medida de la variable observada, lo que simplifica significativamente la interpretación de los errores. RMSE es más sensible a errores atípicos que el MAE y es considerado una de las métricas más confiables para modelar cuando se requiere sensibilidad a grandes errores [4].
- **MAE**: Esta métrica calcula el promedio de los errores absolutos entre los valores observados y los predichos, lo que proporciona una visión directa de la magnitud de los errores sin la influencia de la penalización cuadrática de los errores grandes como en el MSE. El MAE es útil cuando se necesitan comparaciones consistentes entre diversas evaluaciones de modelos, ya que no pondera excesivamente los errores como lo hace el MSE [4].

Las fórmulas para calcular estas métricas son las siguientes:

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

$$\textbf{Root Mean Squared Error (RMSE)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\textbf{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Aquí, $n$ representa el número de observaciones, $y_i$ es el valor observado y $\hat{y}_i$ el valor predicho. Estas métricas son cruciales para la evaluación precisa de los modelos de regresión, ya que ofrecen diferentes perspectivas sobre los errores de predicción y permiten una optimización efectiva del modelo en función de las necesidades específicas del problema.

# Capítulo 3

## Metodología

La metodología utilizada para la comparación de modelos de predicción de tráfico se basa en el uso de GNN, las cuales consideran tanto la dependencia espacial como temporal. Para este estudio, se empleó la base de datos METR-LA, reconocida en la predicción del tráfico y la investigación en aprendizaje automático. A continuación, se detallan los pasos seguidos para implementar y evaluar los modelos híbridos de GNN:

1. **Selección y Preprocesamiento de la Base de Datos:** Se utilizó la base de datos METR-LA, que contiene datos de velocidad de tráfico recopilados de detectores de bucle en la red de carreteras del condado de Los Ángeles. Las características del tráfico fueron normalizadas para facilitar el entrenamiento de los modelos.

2. **Representación del Grafo:** El conjunto de datos METR-LA se modeló como grafos temporales con características de nodos, conexiones (aristas) y valores objetivo organizados en secuencias temporales para facilitar la predicción del tráfico.

3. **Implementación de Modelos de Redes Neuronales de Grafos:** Se implementaron y compararon varios modelos de GNN, incluyendo GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM, que consideran tanto la dependencia espacial como temporal de los datos.

4. **Entrenamiento y Validación:** Los modelos se entrenaron utilizando

técnicas de validación cruzada para evaluar su rendimiento y evitar el sobreajuste. Se utilizaron métricas estándar como el MSE para evaluar la precisión de las predicciones.

5. **Análisis de Resultados:** Se compararon los resultados de los diferentes modelos para identificar el más preciso y eficiente. Se analizaron las predicciones en diferentes condiciones de tráfico y se evaluó la escalabilidad de los modelos en términos de número de nodos y características.

6. **Interpretación y Aplicaciones:** Finalmente, se interpretaron los resultados para proporcionar recomendaciones sobre el uso de GNNs en la gestión del tráfico y se discutieron posibles aplicaciones prácticas en sistemas de transporte inteligente.

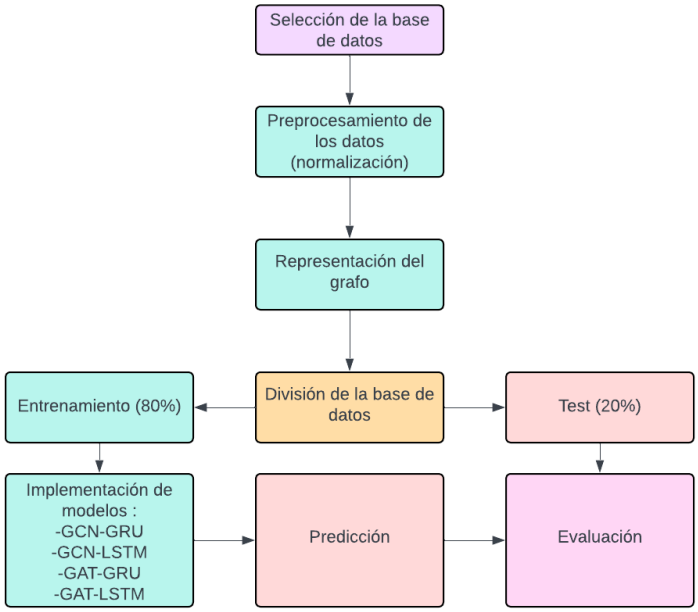



Figura 3.1: Diagrama de la metodología implementada.

## 3.1. Base de datos, preprocesamiento de los datos y representación de grafo

La capacidad de predecir patrones de tráfico eficientemente es fundamental para la gestión moderna de tráfico urbano. En este estudio, se

tomo la velocidad vehicular como variable principal para la predicción, aunque se reconoce que factores adicionales como condiciones climáticas y la densidad poblacional también influyen significativamente en los patrones de tráfico. Futuras investigaciones podrían incorporar estas variables para enriquecer los modelos y mejorar la precisión de las predicciones.

### 3.1.1. Selección de la base de datos

Se utilizó la base de datos METR-LA, un conjunto de datos de velocidad de tráfico recogido a través de detectores de bucle en la red de carreteras del condado de Los Ángeles. Este conjunto contiene mediciones detalladas de 207 sensores a lo largo de un período de cuatro meses, desde marzo hasta junio de 2012, con datos recopilados en intervalos de cinco minutos. Esto resulta en un total de 34,272 periodos de tiempo registrados, proporcionando una base de datos robusta para el análisis de tráfico [18].

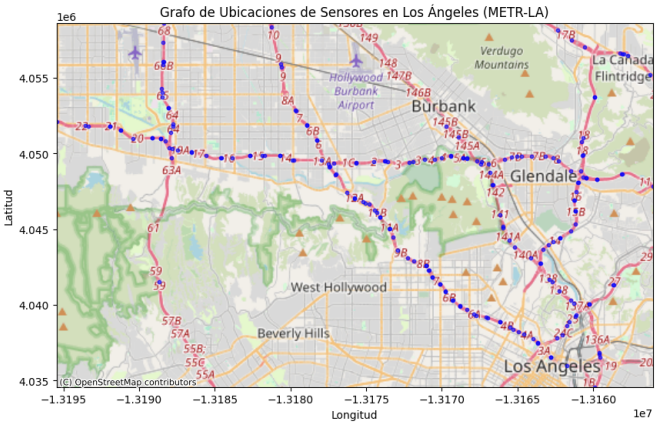



Figura 3.2: Ubicación de los sensores sobre el mapa de Los Ángeles.

Los datos fueron importados del paquete *torch_geometric_temporal* [21], lo que facilitó su representación y manejo como grafos temporales, con características de nodos, conexiones (aristas) y valores objetivo, organizados en secuencias temporales.

Además, se eligió la base de datos METR-LA porque es ampliamente reconocida y utilizada en la comunidad de investigación y académica, especialmente en el campo de la predicción del tráfico y el aprendizaje

automático. Además, se ha convertido en un estándar de referencia para comparar diferentes enfoques y modelos de predicción de tráfico. Esto permite validar y comparar los resultados de manera consistente con estudios previos.

### 3.1.2. Estructura y preprocesamiento de los datos

El conjunto de datos METR-LA se procesó para adaptarse a modelos de GNN, enfocándose en capturar la dinámica temporal y espacial del tráfico. El preprocesamiento incluyó la normalización de las velocidades vehiculares y la transformación de los datos en grafos temporales donde:

- **Adjacency Matrix (A)**: La matriz de adyacencia que representa las conexiones entre los sensores, estableciendo la estructura fundamental del grafo y reflejando la interconexión entre diferentes ubicaciones en la red de carreteras.
- **Features (X)**: Las características dinámicas de tráfico, representadas por la velocidad promedio registrada por cada sensor en los últimos 12 instantes de tiempo, que capturan la evolución temporal del tráfico en cada nodo.
- **Targets (y)**: Los valores objetivo del modelo, que en este caso son las velocidades de tráfico futuras que se buscan predecir, crucial para la planificación y gestión de tráfico.

### 3.1.3. División y uso de los datos

Para garantizar la robustez y la validez de los modelos desarrollados, el conjunto total de datos se dividió en 80 % para entrenamiento y 20 % para pruebas, correspondiendo a 27408 y 6852 observaciones respectivamente. Esta división permite que el modelo se entrene con una amplia variedad de datos y sea evaluado contra un conjunto independiente, asegurando que los resultados sean representativos y confiables. Las dimensiones de los datos de entrenamiento y prueba se establecieron como [27408, 12, 207, 1] para las características y [27408, 207, 1] para las etiquetas, con estructuras similares para el conjunto de pruebas.

Además, los datos fueron normalizados utilizando la técnica de Z-Score normalization. La media y la desviación estándar utilizadas para esta normalización fueron 54.41 y 19.49, respectivamente. Estas estadísticas fueron calculadas considerando todas las mediciones de todos los nodos y todas las observaciones temporales, garantizando una normalización consistente y precisa de los datos.

Para obtener detalles sobre la recopilación de la base de datos, consulte la sección de anexos A.1.2.

## 3.2. Implementación de modelos de Redes Neuronales de Grafos

En el desarrollo de este proyecto, se ha implementado 4 tipos diferentes de modelos híbridos que combinan arquitecturas de GNN con RNN para abordar el problema de la predicción del tráfico urbano. Estos modelos están diseñados para aprovechar tanto la información espacial, capturada mediante las GNN, como la información temporal, capturada mediante las RNN. A continuación, se presenta una tabla que resume los modelos implementados, destacando las componentes espaciales y temporales utilizadas, así como las referencias de los trabajos previos en los que se basan.

| Espacial | Temporal | Paper referencial |
|---|---|---|
| GCN | GRU | Modificación de: "T-GCN: A Temporal Graph ConvolutionalNetwork for Traffic Prediction" [36] |
| GCN | LSTM | Modificación de: "Long-Term Recurrent Convolutional Network (LRCN)" [25] |
| GAT | GRU | Modificación de: "Spatial-Temporal Graph Attention Networks: A Deep Learning Approach for Traffic Forecasting" [35] |
| GAT | LSTM | Modificación de: "Spatial-Temporal Graph Attention Networks: A Deep Learning Approach for Traffic Forecasting" [35] |

Cuadro 3.1: Modelos implementados

El diagrama base de los modelos implementados se puede ver en la Figura 3.3.

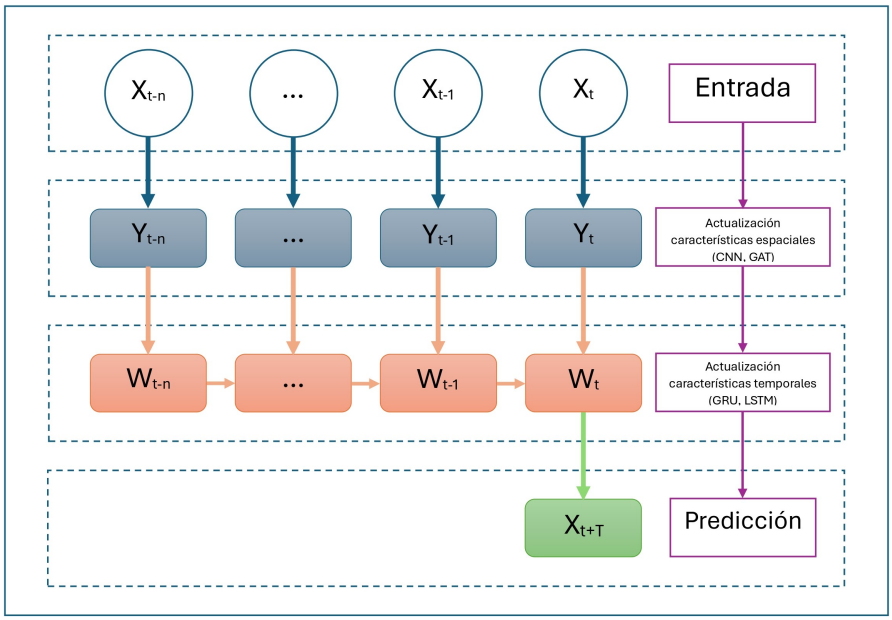



Figura 3.3: Diagrama general de los modelos implementados.

### 3.2.1. Herramientas y Bibliotecas Utilizadas

La plataforma de Google Colab y el lenguaje de programación Python fueron seleccionados por la flexibilidad y potencia de Python en el manejo de datos y modelado, así como por las capacidades interactivas de Jupyter para facilitar la visualización y análisis de los resultados.

Además, se utilizaron varias bibliotecas especializadas que apoyaron la manipulación de datos y la construcción de modelos de redes neuronales:

- **NumPy**: Para la manipulación numérica de datos, esencial para la preparación de los mismos.
- **Matplotlib y PyPlot**: Para la visualización de resultados y análisis del comportamiento de los modelos.
- **PyTorch y Torch Geometric**: Proporcionan estructuras y funciones necesarias para la construcción y entrenamiento de modelos de redes neuronales, especialmente para trabajar con grafos.

- **Sklearn**: Utilizada para calcular métricas de rendimiento como el MSE y el error absoluto medio.

Para obtener detalles sobre la importación de las librerías necesarias, consulte la sección de anexos A.1.1.

### 3.2.2. Hiperparámetros para Modelos Híbridos

Se implementaron modelos avanzados que combinan GCN o GAT con unidades recurrentes GRU y LSTM para modelar eficazmente el tráfico urbano. Estas combinaciones aprovechan la capacidad de las GCN y GAT para procesar la estructura de grafos, capturando las relaciones espaciales entre los nodos, que son cruciales para entender cómo los cambios en una parte del sistema de tráfico pueden influir en otras partes. Las unidades recurrentes GRU y LSTM se utilizan para modelar la dependencia temporal de los datos de tráfico, permitiendo al modelo prever cambios basados en tendencias y patrones históricos.

La implementación de cada uno de los modelos se encuentra en el capítulo de anexos (A). En la subsección A.1.3 se puede encontrar la implementación del modelo GCN-LSTM, en la subsección A.1.4 la del modelo GCN-GRU, en la subsección A.1.5 la del modelo GAT-LSTM, y en la subsección A.1.6 la del modelo GAT-GRU. Si se desea conocer el código completo el lector puede revisar en GitHub: `https://github.com/Diana1719/TIC_GNN`.

#### Configuración de Modelos GCN y GAT

#### Modelo GCN

- **Variable de Entrada Única:** La única variable de entrada utilizada es la velocidad de cada nodo.
- **Transformación inicial:** En lugar de trabajar con todas las características observables, el modelo utiliza una representación latente bidimensional para cada nodo. Esta representación comprime la información relevante en dos dimensiones, permitiendo capturar relaciones complejas y contextos dentro del grafo de manera eficiente.

**Modelo GAT**

- **Entrada:** Al igual que con la GCN, se enfoca en la velocidad como la única entrada para maximizar la eficiencia del modelo en el procesamiento de la característica más relevante.
- **Transformación inicial:** La atención es fundamental para asignar diferentes pesos a las influencias de los nodos vecinos, lo que mejora la capacidad del modelo para centrarse en las interacciones más relevantes. Esta red transforma cada nodo, produciendo una representación latente en dos dimensiones, preparando así los datos para el análisis temporal.

**GCN-LSTM (Graph Convolutional Network - Long Short-Term Memory)**

**Configuración del Modelo LSTM**

- **Entrada:** Cada nodo, después de ser procesado por la GCN, aporta dos características transformadas, es decir, la entrada tiene una dimension de $2 \times 207$ que el LSTM utiliza para entender cómo evoluciona el tráfico con el tiempo.
- **Transformación:** Proporciona al modelo suficiente flexibilidad para aprender complejidades en los datos sin ser tan grande como para causar un sobreajuste. De igual manera, esta red transforma los datos, generando una representación latente en cuatro dimensiones, preparándolos así para la capa totalmente conectada.
- **Dos Capas de LSTM:** Mejora la capacidad del modelo para procesar secuencias de datos más largas.
- **Salida:** El modelo está diseñado para garantizar que su optimización coincida perfectamente con el objetivo de predecir la velocidad del tráfico, un aspecto crucial para las necesidades de gestión del tráfico. Se emplea una capa totalmente conectada para este fin, con una entrada de dimensiones $4 \times 207$, y una salida que se ajusta según la cantidad de puntos temporales que se necesiten predecir.

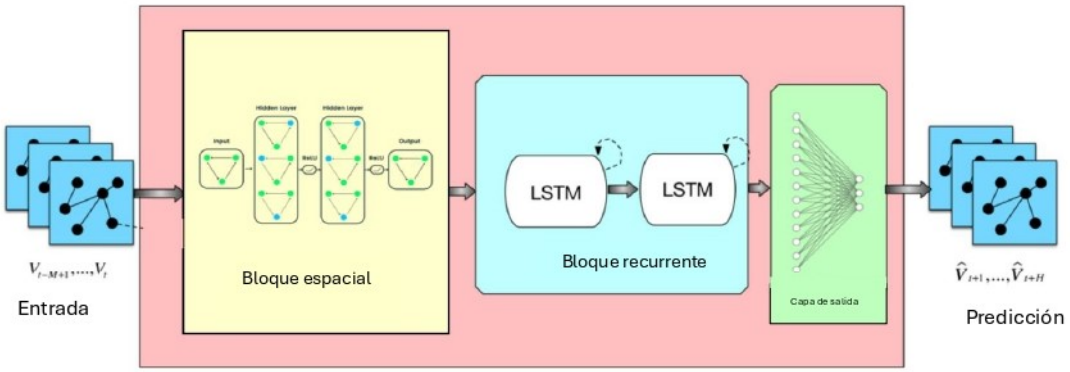



Figura 3.4: Arquitectura de redes de grafos espacio-temporales (GCN-LSTM). Modificado de: Arquitectura de redes de atención de grafos espacio-temporales (GAT-LSTM) [35].

### GCN-GRU (Graph Convolutional Network - Gated Recurrent Unit)

### Configuración del Modelo GRU

- **Entrada:** Cada nodo, después de ser procesado por la GCN, aporta dos características transformadas, es decir, la entrada tiene una dimension de $2 \times 207$ que el GRU utiliza para entender cómo evoluciona el tráfico con el tiempo.
- **Transformación:** Ofrece un equilibrio entre complejidad y rendimiento, proporcionando suficiente capacidad para detectar y memorizar patrones temporales importantes sin consumir recursos excesivos. Esta red transforma los datos, generando una representación latente en cuatro dimensiones, preparándolos así para la capa totalmente conectada.
- **Dos Capas de GRU:** Permite al modelo profundizar en el análisis temporal, detectando dependencias a largo plazo que pueden ser esenciales para predecir fluctuaciones y tendencias en el tráfico.
- **Salida:** El modelo se centra en predecir la velocidad del tráfico, que es la métrica final deseada. Para ello, se utiliza una capa totalmente conectada cuya entrada tiene dimensiones de $4 \times 207$ y cuya salida varía según el número de puntos en el tiempo a predecir.

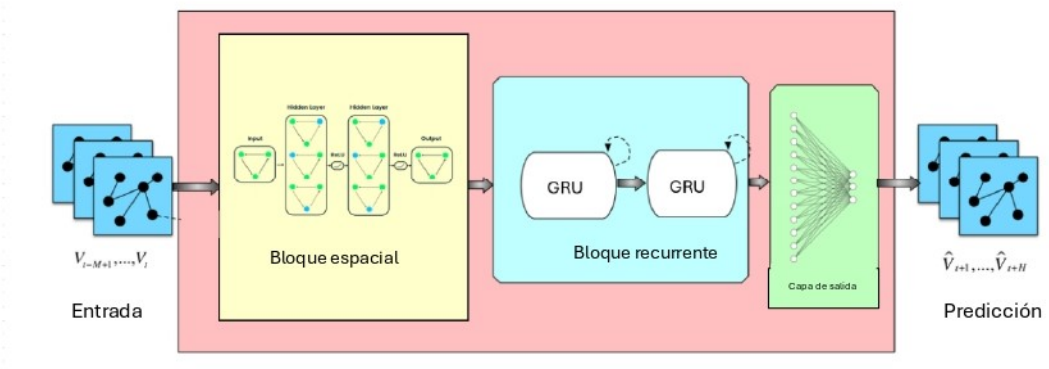



Figura 3.5: Arquitectura de redes de atención de grafos espacio-temporales (GCN-GRU). Modificado de: Arquitectura de redes de atención de grafos espacio-temporales (GAT-LSTM) [35].

**GAT-LSTM (Graph Attention Network - Long Short-Term Memory)**

**Configuración del Modelo LSTM para GAT-LSTM**

- **Entrada:** Cada nodo contribuye con dos características influenciadas por la atención, lo que ayuda al LSTM a considerar la importancia relativa de cada nodo en función de su impacto en el tráfico general.
- **Transformación:** Esta configuración permite al LSTM mantener y procesar información a lo largo de periodos más extensos, lo cual es crucial para captar dependencias temporales de largo alcance que pueden ser determinantes en la predicción de tendencias de tráfico y eventos cíclicos. Para ello, esta capa transforma los datos, obteniéndose una representación latente de cuatro dimensiones, la cual sera la entrada de la capa totalmente conectada.
- **Dos Capas de LSTM:** Profundiza en la capacidad de analizar secuencias largas, proporcionando una ventaja en la modelización de patrones temporales que pueden ser influenciados por eventos estacionales o anómalos.
- **Salida:** Alinea todos los recursos del modelo para optimizar la predicción de la velocidad, facilitando la interpretación directa de los

resultados y su aplicación en sistemas de gestión de tráfico. Similarmente, se emplea una capa totalmente conectada para este fin, con una entrada de dimensiones $4 \times 207$, y una salida que se ajusta según la cantidad de puntos temporales que se necesiten predecir.

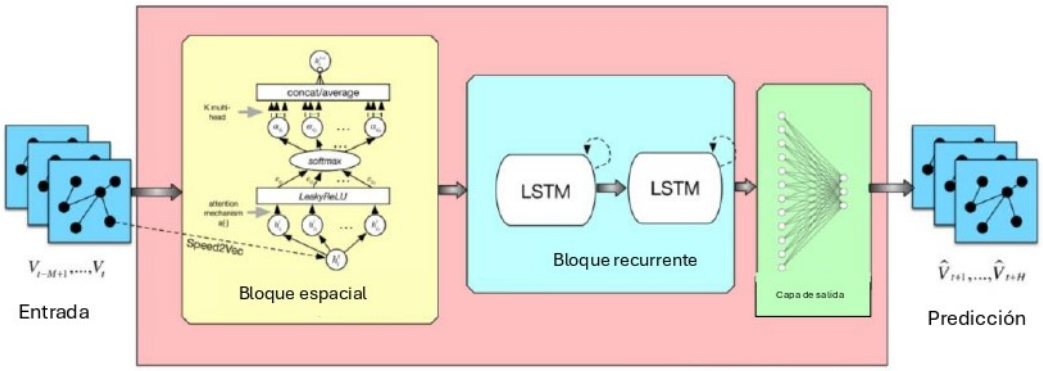



Figura 3.6: Arquitectura de redes de atención de grafos espacio-temporales (GAT-LSTM) . Modificado de: Arquitectura de redes de atención de grafos espacio-temporales (GAT-LSTM) [35].

**GAT-GRU (Graph Attention Network - Gated Recurrent Unit)**

**Configuración del Modelo GRU para GAT-GRU**

- **Entrada:** Esta configuración utiliza las características transformadas por el GAT, donde cada nodo tiene dos características representando la información ponderada por la atención, lo cual es crucial para reconocer qué nodos son más influyentes en el tráfico en tiempo real.
- **Transformación:** La dimensión oculta más grande permite al GRU procesar y memorizar patrones temporales complejos que surgen de las interacciones dinámicas entre nodos, especialmente en situaciones de tráfico congestionado o eventos no rutinarios. Esta red transforma los datos, generando una representación latente en cuatro dimensiones, preparándolos así para la capa totalmente conectada.
- **Dos Capas de GRU:** Al usar dos capas, el modelo puede extraer y refinar la información temporal a diferentes niveles de abstracción.

- **Salida:** Concentra el aprendizaje y la predicción en la velocidad, permitiendo que el modelo se especialice en prever esta variable esencial con mayor precisión. De igual manera, se emplea una capa totalmente conectada para este fin, con una entrada de dimensiones $4 \times 207$, y una salida que se ajusta según la cantidad de puntos temporales que se necesiten predecir.

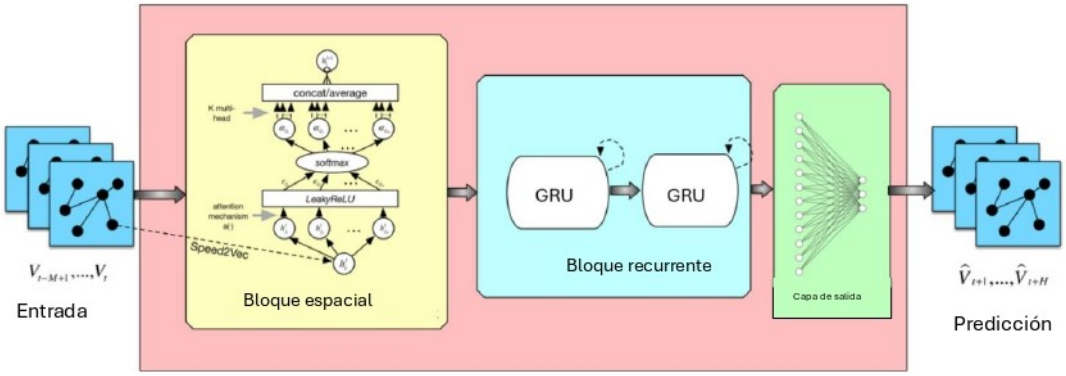



Figura 3.7: Arquitectura de redes de atención de grafos espacio-temporales (GAT-GRU). Modificado de: Arquitectura de redes de atención de grafos espacio-temporales (GAT-LSTM) [35].

### 3.2.3. Entrenamiento y Evaluación de Modelos

Los modelos se entrenaron en Google Colab, utilizando un entorno Jupyter Notebook con acceso a GPU. Se utilizó un tamaño de lote de 256 para optimizar la eficiencia de la memoria y equilibrar la velocidad de entrenamiento con la estabilidad de los gradientes.

El optimizador Adam fue seleccionado por su capacidad para manejar optimizaciones no convexas y su eficiencia computacional. Se estableció una tasa de aprendizaje de 0.0001 para facilitar una convergencia estable y minimizar las oscilaciones en los mínimos locales. Como métrica de evaluación, se utilizó el MSE debido a su capacidad para penalizar significativamente los errores grandes, esencial para la precisión en predicciones de velocidad del tráfico. El entrenamiento se limitó a 20 épocas para equilibrar el tiempo de entrenamiento y la capacidad del modelo para aprender patrones significativos sin incurrir en sobreajuste.

La elección de estos hiperparámetros se realizó de manera empírica

debido a las limitaciones computacionales, que impidieron un estudio exhaustivo para determinar los hiperparámetros óptimos para cada modelo. Sin embargo, con los hiperparámetros seleccionados, se logró una convergencia adecuada del modelo.

# Capítulo 4

## Resultados, conclusiones y recomendaciones

En la presente sección se exponen los resultados obtenidos para los modelos híbridos implementados: GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM. Los resultados de la predicción temporal en el conjunto de testeo, las medidas de evaluación de modelos de serie temporal y las métricas de bondad de ajuste, tales como MSE, RMSE y MAE, se presentan en la subsección 4.1 de Resultados. La discusión sobre el rendimiento de los modelos, su desempeño en la predicción de tráfico, el análisis de modelos específicos y la comparación de los modelos en diferentes horizontes temporales se abordan en la subsección 4.2 de Discusión de Resultados. Las conclusiones y recomendaciones respecto a las limitaciones de los modelos se discuten en la subsección 4.3 de Conclusiones y Recomendaciones.

## 4.1. Resultados

En la presente sección se exponen los resultados obtenidos para los modelos híbridos implementados: GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM. La sección 4.1.1 presenta la evolución del error durante el entrenamiento y la prueba, proporcionando una visión general de cómo cada modelo mejora a lo largo del tiempo. En la sección 4.1.2, se detallan las predicciones del siguiente instante para cada uno de los modelos, mostrando su capacidad para anticipar el próximo valor en la serie tem-

poral. La sección 4.1.3 ofrece una comparación de la serie temporal entre los valores reales y predichos, evaluando la precisión y la capacidad de los modelos para seguir las tendencias de los datos. La sección 4.1.4 evalúa el rendimiento de los modelos en diferentes horizontes temporales, analizando su efectividad en la predicción a corto, medio y largo plazo. Finalmente, la sección 4.1.5 se enfoca en la evaluación de modelos con diferentes hiperparámetros, explorando cómo ajustes específicos pueden influir en la precisión y robustez de las predicciones.

Es importante señalar que los resultados obtenidos no son directamente comparables con los de los artículos citados previamente. Debido a las limitaciones de costo computacional, no se utilizaron los mismos hiperparámetros. Por esta razón, los modelos implementados son una adaptación de los modelos propuestos en los artículos y no son exactamente iguales.

El lector puede encontrar en los anexos parte del código de la implementación de este proyecto. Sin embargo, todos los archivos necesarios para poder replicar el componente TIC se puede encontrar en GitHub: https://github.com/Diana1719/TIC_GNN.

### 4.1.1. Evolución del error

La figura 4.1 presenta la evolución de la función de pérdida durante el entrenamiento de cuatro modelos híbridos aplicados a la predicción del tráfico: GCN-GRU, GCN-LSTM, GAT-GRU, y GAT-LSTM. Cada subfigura muestra dos curvas, una para el error en el conjunto de entrenamiento y otra para el error en el conjunto de testeo, a lo largo de las iteraciones de entrenamiento.

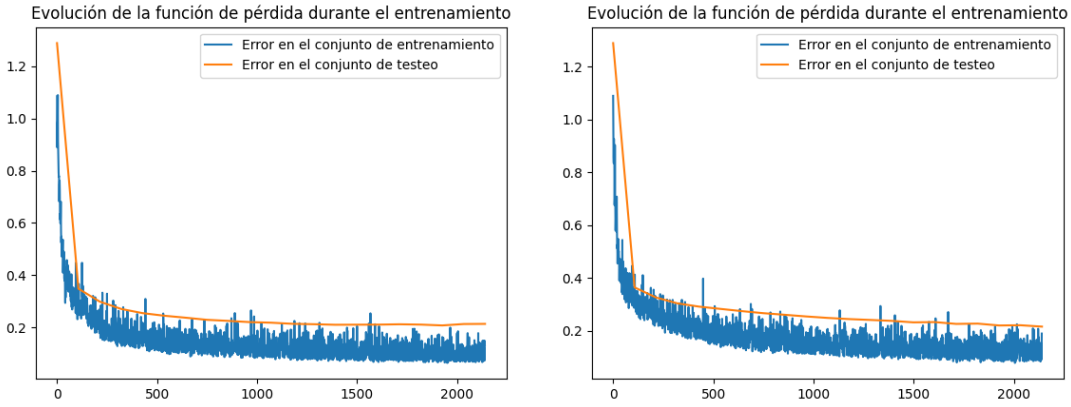
La subfigura 4.1a muestra el comportamiento del modelo GCN-LSTM. Similar al GCN-GRU, se observa una rápida disminución de la pérdida en las primeras iteraciones. La estabilización de la pérdida sugiere que el modelo también ha alcanzado un buen ajuste a los datos.

La subfigura 4.1b muestra la evolución de la pérdida para el modelo GCN-GRU. Al principio del entrenamiento, ambos errores, de entrenamiento y testeo, son altos, pero disminuyen rápidamente. Esto indica que el modelo está aprendiendo a partir de los datos. A medida que el

entrenamiento avanza, la pérdida se estabiliza, sugiriendo que el modelo ha alcanzado un punto de convergencia.

La subfigura 4.1c ilustra la evolución de la pérdida para el modelo GAT-LSTM. Similar a los otros modelos, muestra una rápida disminución inicial seguida de una estabilización. La curva de testeo es más suave comparada con la curva de entrenamiento, lo que sugiere que el modelo está generalizando bien.

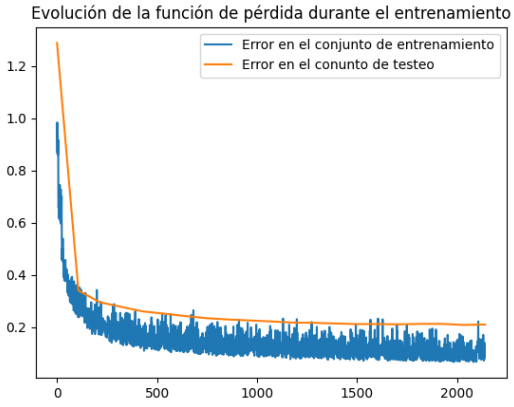
En la subfigura 4.1d , se presenta el modelo GAT-GRU. Este modelo también muestra una rápida disminución inicial en la pérdida, seguida por una estabilización. Los picos en la curva de entrenamiento pueden indicar cierta variabilidad en los datos o la optimización.



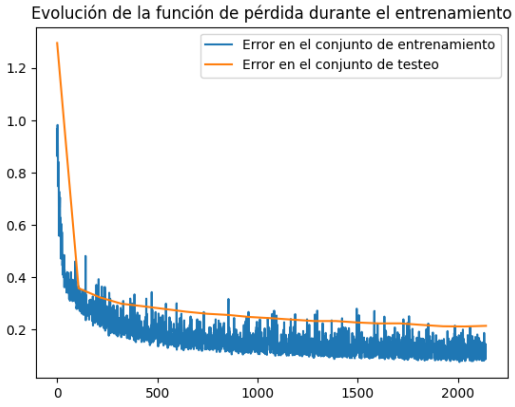

(a) GCN-LSTM




(b) GCN-GRU




(c) GAT-LSTM




(d) GAT-GRU

Figura 4.1: Evolución del error en los modelos híbridos en los conjuntos de validación y prueba.

### 4.1.2. Predicción del siguiente instante

A continuación, se muestran los resultados de la predicción del tráfico en el siguiente instante utilizando cuatro modelos híbridos (GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM).

Claro, aquí tienes la descripción en formato de párrafos:

La tabla 4.1 muestra las métricas de error de los cuatro modelos híbridos (GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU) en la predicción del tráfico. Las métricas utilizadas son el Error Cuadrático Medio (MSE), la Raíz del Error Cuadrático Medio (RMSE) y el Error Absoluto Medio (MAE). Estas métricas se calculan para evaluar la precisión de las predicciones de cada modelo.

| Modelos | 5 minutos | | |
|---|---|---|---|
| | MSE | RMSE | MAE |
| GCN-LSTM | 77.86 | 8.82 | 5.01 |
| GCN-GRU | **69.08** | **8.31** | **4.67** |
| GAT-LSTM | 77.32 | 8.79 | 5.13 |
| GAT-GRU | 79.34 | 8.91 | 4.70 |

Cuadro 4.1: Tabla de comparación de modelos en base a los resultados de error cuadrático medio (MSE), raiz del error cuadrático medio (RMSE) y error absoluto medio (MAE)

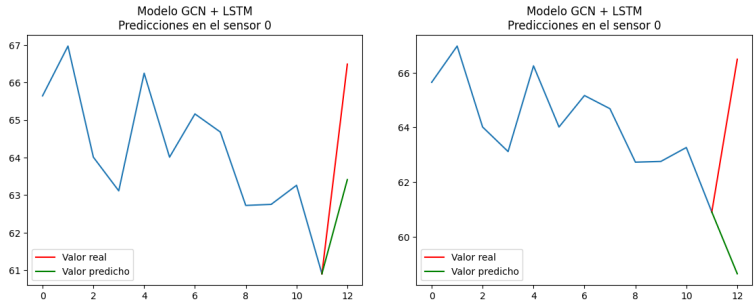
El modelo GCN-GRU es el más preciso, con un MSE de 69.08, un RMSE de 8.31 y un MAE de 4.67, lo que indica menores errores en las predicciones. El modelo GCN-LSTM, aunque menos preciso que el GCN-GRU, tiene un MSE de 77.86, un RMSE de 8.82 y un MAE de 5.01, ofreciendo predicciones razonablemente buenas. Por otro lado, el modelo GAT-GRU presenta el mayor MSE de 79.34, con un RMSE de 8.91 y un MAE de 4.70, sugiriendo mayores errores cuadráticos medios pero un error absoluto medio relativamente bajo.

A continuación, se analizarán también los resultados gráficamente. Los datos se han recopilado de múltiples sensores, pero como ejemplo, se presentarán los resultados de dos sensores específicos: el sensor 0 y el sensor 100. Esta comparación permitirá evaluar la consistencia y el desempeño de los modelos en diversos contextos de medición. La figura

4.2 muestra la predicción del siguiente instante de tráfico utilizando cada uno de los modelos para el sensor 0, mientras que La figura 4.3 muestra la predicción para el sensor 100. En cada subfigura, la línea azul representa los valores reales del tráfico, mientras que la línea roja representa los valores predichos por cada modelo.

La subfigura 4.2a sigue la tendencia general de los valores reales con una consistencia similar al GCN-GRU, aunque muestra desviaciones en cambios bruscos, lo que sugiere que el modelo puede mejorar en la predicción de eventos abruptos. De manera similar, la subfigura 4.2b captura bien las fluctuaciones y caídas en los datos, aunque también presenta algunas desviaciones en puntos de cambio abrupto, manteniendo en general una buena precisión.

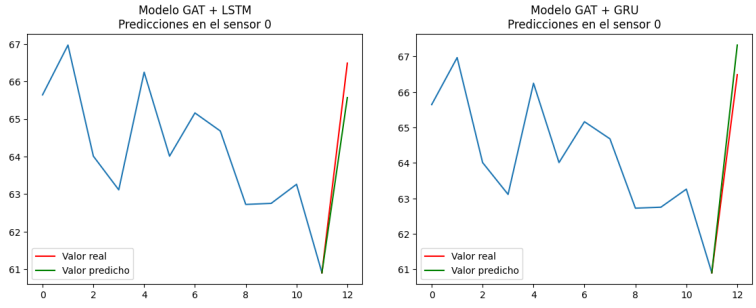
Las subfiguras 4.2c y 4.2d siguen la tendencia general de los valores reales, pero tienen dificultades para modelar eventos abruptos, similar a los otros modelos evaluados. En particular, la subfigura 4.2d muestra mayores dificultades en capturar estos cambios en comparación con los modelos basados en GCN, presentando desviaciones significativas en puntos críticos.




(a) GCN-LSTM




(b) GCN-GRU




(c) GAT-LSTM




(d) GAT-GRU

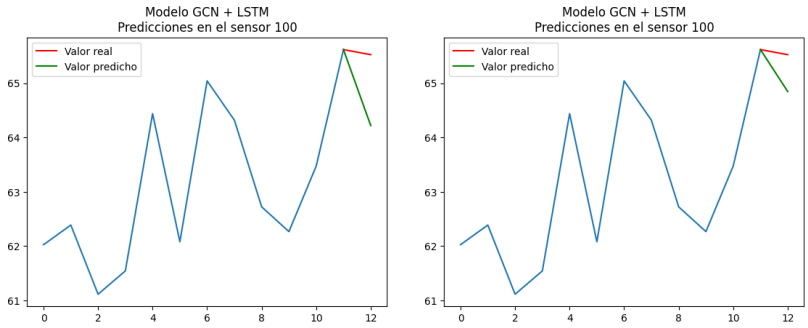
Figura 4.2: Predicción del siguiente instante utilizando cada uno de los modelos para el sensor 0.

El modelo GCN-LSTM (figura 4.3a) sigue la tendencia general de los valores reales, similar al comportamiento observado para el sensor 0. Aunque se observan desviaciones en los cambios bruscos, las predicciones son consistentes en la mayoría de los casos.

El modelo GCN-GRU (figura 4.3b) sigue de cerca la tendencia de los valores reales, aunque se observan algunas desviaciones en los puntos de mayor variabilidad. En general, el modelo captura bien la estructura del tráfico, demostrando un buen rendimiento.

El modelo GAT-LSTM (figura 4.3c) sigue la tendencia general de los valores reales, pero presenta desviaciones significativas en las caídas abruptas. Esto sugiere dificultades similares a las observadas en el sensor 0, indicando una limitación en la capacidad del modelo para predecir cambios rápidos.

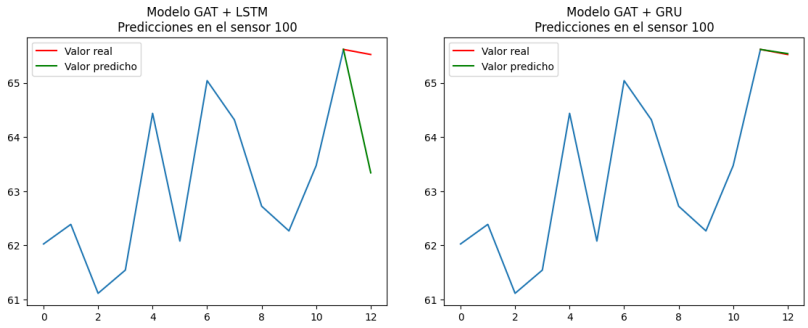
Finalmente, el modelo GAT-GRU (figura 4.3d) muestra que las predicciones siguen la tendencia de los valores reales, pero tiene dificultades para capturar los cambios abruptos, presentando desviaciones significativas en esos puntos, al igual que en el sensor 0.




(a) GCN-LSTM




(b) GCN-GRU




(c) GAT-LSTM




(d) GAT-GRU

Figura 4.3: Predicción del siguiente instante utilizando cada uno de los modelos para el sensor 100.

En general, todos los modelos híbridos (GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU) muestran una capacidad razonable para seguir la tendencia general de la serie temporal de tráfico tanto para el sensor 0 como para el sensor 100. Sin embargo, presentan desafíos similares en la captura de caídas abruptas y cambios rápidos. La comparación entre los sensores revela que las dificultades en la predicción de caídas pronunciadas son consistentes en ambos casos, lo que indica una posible área de mejora en los modelos GNN híbridos para la predicción del tráfico.

El modelo GCN-GRU demuestra una buena capacidad para capturar las fluctuaciones en los datos y seguir la tendencia general, aunque también presenta desviaciones en las caídas más pronunciadas; se destaca como el más preciso y consistente entre los cuatro modelos híbridos evaluados, mientras que el GAT-GRU presenta el mayor error cuadrático medio. Los modelos GCN-LSTM y GAT-LSTM muestran un rendimiento intermedio, con GCN-LSTM ligeramente mejor que GAT-LSTM.

### 4.1.3. Comparación serie temporal

A continuación, se presentan los resultados obtenidos en la predicción de la serie temporal de tráfico utilizando cuatro modelos híbridos (GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU). Los resultados se han obtenido de múltiples sensores, pero de igual manera que en el caso del análisis del siguiente instante, como ejemplo, se presentarán los resultados de dos sensores específicos: el sensor 0 y el sensor 100. La figura 4.4 muestra la comparación de la serie temporal de tráfico con los valores predichos sobre el sensor 0, mientras que la figura 4.5 muestra una comparación similar de la serie temporal de tráfico y los valores predichos, pero esta vez para el sensor 100. En cada subfigura, la línea azul representa los valores reales de la serie temporal, mientras que la línea roja representa los valores predichos por cada modelo.

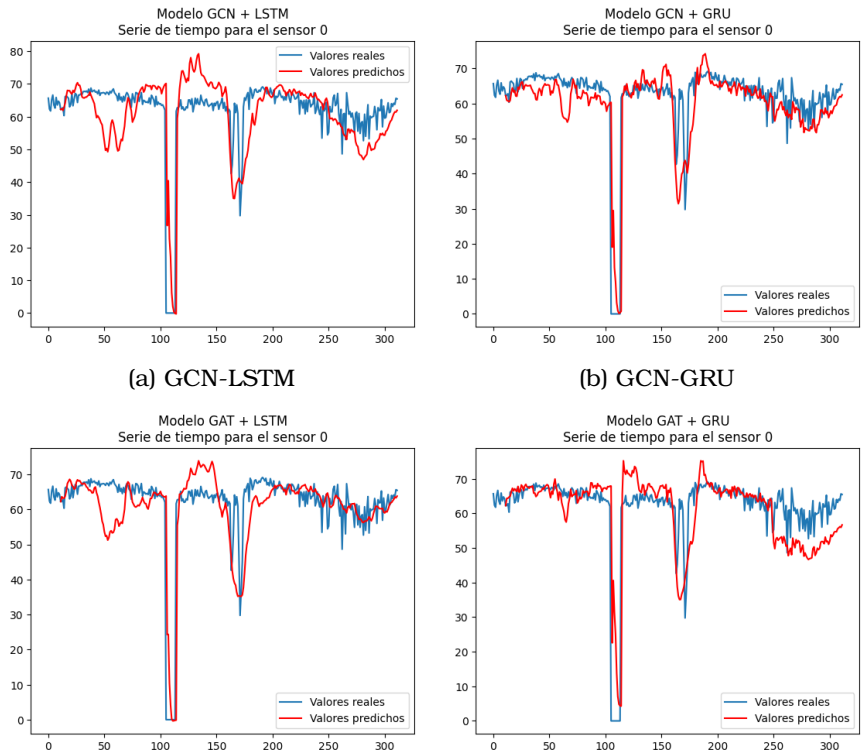
La subfigura 4.4a presenta el comportamiento del modelo GCN-LSTM. Similar al GCN-GRU, este modelo sigue de cerca la tendencia general de los valores reales, aunque se observan desviaciones en ciertos puntos, especialmente en las caídas bruscas.

La subfigura 4.4b muestra los resultados del modelo GCN-GRU. Se

observa que los valores predichos (línea roja) siguen de cerca la tendencia de los valores reales (línea azul). Aunque hay momentos en los que las predicciones se desvían de los valores reales, el modelo logra capturar la mayoría de las fluctuaciones y caídas en los datos.

La subfigura 4.4c muestra las predicciones del modelo GAT-LSTM. Al igual que los otros modelos, sigue la tendencia de los valores reales, pero también presenta desviaciones significativas en las caídas abruptas, lo que sugiere una dificultad para modelar estos cambios rápidos.

En la subfigura 4.4d, correspondiente al modelo GAT-GRU, se puede ver que las predicciones también siguen la tendencia de los valores reales. Sin embargo, este modelo parece tener más dificultades en capturar las caídas pronunciadas, mostrando una mayor desviación en esos puntos críticos.




(a) GCN-LSTM (b) GCN-GRU

(c) GAT-LSTM (d) GAT-GRU

Figura 4.4: Comparación de la Serie Temporal con los Valores Predichos por los Modelos para el Sensor 0

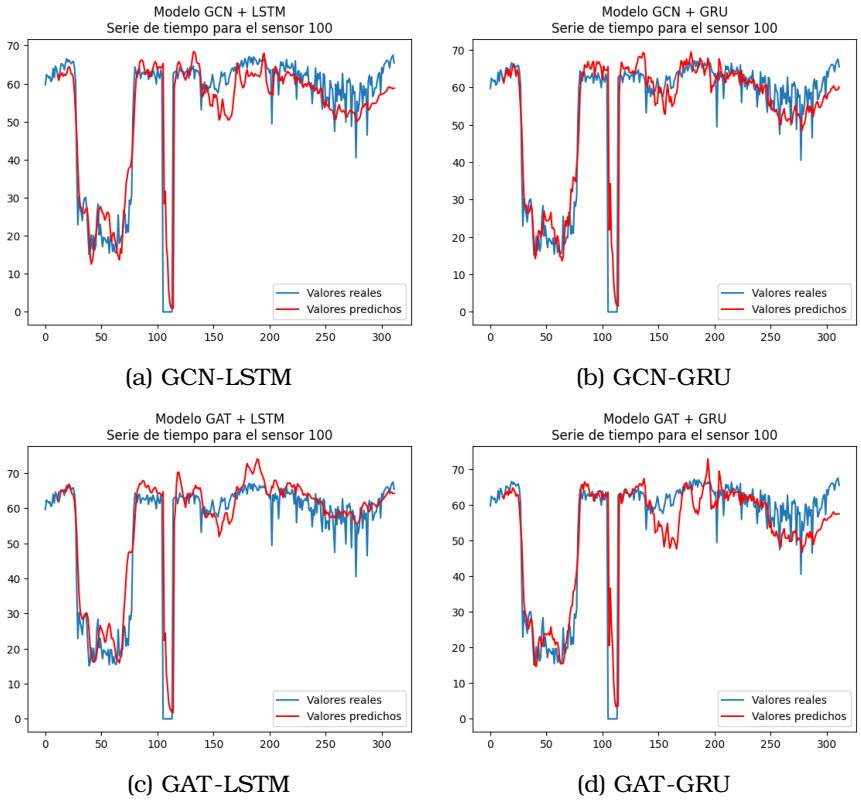
La subfigura 4.5a presenta el comportamiento del modelo GCN-LSTM para el sensor 100. Similar al comportamiento observado para el sensor

0, este modelo sigue la tendencia general de los valores reales, pero con algunas desviaciones notables en las caídas bruscas.

La subfigura 4.5b muestra los resultados del modelo GCN-GRU para el sensor 100. Los valores predichos siguen de cerca la tendencia de los valores reales, aunque se observan algunas desviaciones en los puntos de mayor variabilidad.

En la subfigura 4.5c muestra las predicciones del modelo GAT-LSTM para el sensor 100. Este modelo sigue la tendencia de los valores reales, pero al igual que en el sensor 0, presenta desviaciones significativas en las caídas abruptas, lo que sugiere dificultades para modelar cambios rápidos.

Finalmente, la subfigura 4.5d, correspondiente al modelo GAT-GRU para el sensor 100, se observa que las predicciones siguen la tendencia de los valores reales, pero con dificultades para capturar las caídas pronunciadas, mostrando desviaciones significativas en esos puntos.




(a) GCN-LSTM (b) GCN-GRU

(c) GAT-LSTM (d) GAT-GRU

Figura 4.5: Comparación de la Serie Temporal con los Valores Predichos por los Modelos para el Sensor 100

En general, todos los modelos híbridos (GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU) muestran una capacidad razonable para seguir la tendencia general de la serie temporal de tráfico tanto para el sensor 0 como para el sensor 100. Sin embargo, presentan desafíos similares en la captura de caídas abruptas y cambios rápidos. La comparación entre los sensores revela que las dificultades en la predicción de caídas pronunciadas son consistentes en ambos casos. En conclusión, el modelo GCN-GRU demuestra una buena capacidad para capturar las fluctuaciones en los datos y seguir la tendencia general, aunque también presenta desviaciones en las caídas más pronunciadas.

### 4.1.4. Evaluación de Modelos en Diferentes Horizontes Temporales

A continuación se presenta una tabla comparativa de los resultados de los modelos GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU en diferentes horizontes temporales para la predicción del tráfico. Los modelos se evaluaron para prever las condiciones del tráfico en los próximos 15, 30 y 60 minutos, correspondientes a periodos corto, medio y largo, con 3, 6 y 12 puntos de datos observados respectivamente. La tabla muestra los resultados en términos de error cuadrático medio (MSE) y error absoluto medio (MAE) para cada modelo y horizonte temporal.

| Modelos | 15 min | | | 30 min | | | 60 min | | |
|---|---|---|---|---|---|---|---|---|---|
| | MSE | RMSE | MAE | MSE | RMSE | MAE | MSE | RMSE | MAE |
| GCN-LSTM | 99.01 | 9.95 | 5.66 | 128.82 | 11.35 | 6.38 | 195.66 | 15.35 | 9.46 |
| GCN-GRU | **94.09** | **9.70** | **5.39** | 144.33 | 12.01 | 7.57 | 205.46 | 14.33 | 8.75 |
| GAT-LSTM | 111.30 | 10.55 | 5.96 | 154.51 | 12.43 | 7.34 | **154.25** | **12.42** | **7.34** |
| GAT-GRU | 107.54 | 10.37 | 5.87 | **117.94** | **10.86** | **6.05** | 193.76 | 13.92 | 7.81 |

Cuadro 4.2: Tabla de comparación de modelos en diferentes horizontes temporales en base a los resultados de error cuadrático medio (MSE), raiz del error cuadrático medio (RMSE) y error absoluto medio (MAE)

En el horizonte temporal de 15 minutos, el modelo GCN-GRU destaca por su rendimiento superior con un MSE de 94.09, un RMSE de 9.70 y un MAE de 5.39. Esto indica que este modelo es capaz de realizar predicciones precisas con bajos errores absolutos y cuadráticos. En comparación,

el GAT-GRU presenta el mayor MSE (107.54) y MAE (7.87), lo que sugiere que tiene más dificultades para predecir la velocidad con precisión en este corto plazo.

Para el horizonte temporal de 30 minutos, el GCN-GRU sigue mostrando un buen rendimiento con un MSE de 144.33, un RMSE de 12.01 y un MAE de 7.37. Sin embargo, el GAT-GRU tiene el mayor MSE (117.94) y un RMSE de 10.86, aunque su MAE (6.05) es el más bajo entre los modelos evaluados. Esto indica que, aunque presenta errores cuadráticos más altos, su capacidad para minimizar los errores absolutos en las predicciones es notablemente buena.

En el horizonte temporal de 60 minutos, se observa una disminución en la precisión de todos los modelos, como era de esperar debido a la mayor incertidumbre asociada con predicciones a más largo plazo. El GCN-GRU presenta un MSE de 205.46, un RMSE de 14.33 y un MAE de 8.75, mostrando un aumento significativo en los errores. En contraste, el GAT-LSTM muestra el mejor rendimiento en este horizonte con un MSE de 154.25, un RMSE de 12.42 y un MAE de 7.34, lo que sugiere que este modelo es más robusto para predicciones a largo plazo.

## 4.2. Discusión de Resultados

En esta sección, se presentan y analizan los resultados obtenidos de los modelos híbridos evaluados: GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU.

### 4.2.1. Rendimiento de los Modelos durante el Entrenamiento

En todos los modelos híbridos, se observó una rápida disminución de la pérdida durante las primeras iteraciones del entrenamiento, seguida de una estabilización. Esta tendencia sugiere que los modelos tienen una buena capacidad de aprendizaje y se ajustan bien a los datos de tráfico, alcanzando un punto de convergencia. Además, la pequeña diferencia entre las curvas de pérdida de entrenamiento y de testeo indica que los

modelos no están sobreajustándose, demostrando una buena capacidad de generalización.

Sin embargo, se identificaron picos en las curvas de pérdida de entrenamiento, especialmente en los modelos basados en GAT. Estos picos pueden indicar variabilidad en los datos de tráfico o la necesidad de ajustar los hiperparámetros o el algoritmo de optimización para mejorar la estabilidad del entrenamiento.

### 4.2.2. Desempeño en la Predicción de Tráfico

Todos los modelos híbridos demostraron una capacidad razonable para seguir la tendencia general de los valores reales del tráfico observados por el sensor 0 y por el sensor 100. No obstante, se observaron desviaciones notables en la predicción del siguiente instante en todos los modelos. Estas desviaciones indican que, aunque los modelos pueden capturar la tendencia general de los datos de tráfico, tienen dificultades para predecir con precisión cambios abruptos o patrones específicos en el siguiente instante.

### 4.2.3. Modelos Específicos

El modelo GCN-GRU, en particular, mostró una buena aproximación a los valores reales a lo largo del periodo observado, aunque aún presenta desviaciones en la predicción del siguiente instante. Esto sugiere que el modelo tiene una capacidad razonable para aprender los patrones de tráfico, pero puede beneficiarse de ajustes adicionales en los hiperparámetros o de técnicas de regularización para mejorar la precisión de las predicciones inmediatas.

En términos generales, todos los modelos híbridos (GCN-GRU, GCN-LSTM, GAT-GRU y GAT-LSTM) demostraron una capacidad razonable para seguir la tendencia general de la serie temporal de tráfico del sensor 0. Sin embargo, cada modelo presenta desviaciones en ciertos puntos, especialmente en las caídas bruscas de los datos. Esto indica que, aunque los modelos pueden capturar la estructura general de la serie temporal, tienen dificultades para predecir cambios abruptos con alta precisión.

El modelo GCN-GRU mostró una buena capacidad para capturar las fluctuaciones en los datos y seguir la tendencia general, aunque también presenta desviaciones en las caídas más pronunciadas.

### 4.2.4. Comparación de los Modelos en Diferentes Horizontes Temporales

En conclusión, el GCN-GRU es el modelo más preciso para predicciones a corto plazo (15 minutos), mientras que el GAT-LSTM se destaca en predicciones a largo plazo (60 minutos). Los modelos GCN-LSTM y GAT-GRU muestran un rendimiento intermedio, siendo más efectivos en horizontes temporales de 30 minutos. Estos resultados destacan la importancia de elegir el modelo adecuado según el horizonte temporal de interés y subrayan la capacidad de los modelos GAT-LSTM para manejar predicciones más desafiantes a largo plazo.

## 4.3. Conclusiones y recomendaciones

### 4.3.1. Conclusiones

- El trabajo desarrollado demuestra que los modelos de redes neuronales de grafos (GNN) aplicados a la predicción del tráfico urbano pueden capturar eficazmente las dependencias espaciales y temporales en los datos de tráfico. En particular, el modelo GCN-GRU mostró un rendimiento robusto y consistente en la mayoría de los horizontes temporales evaluados, destacándose especialmente en predicciones a corto y medio plazo.
- La comparación entre diferentes arquitecturas de GNN (GCN-LSTM, GCN-GRU, GAT-LSTM y GAT-GRU) revela que los modelos basados en GCN tienden a tener un mejor rendimiento en términos de precisión de predicción y capacidad de generalización. Los modelos basados en GAT mostraron un rendimiento inferior, especialmente en horizontes temporales más largos.
- Todos los modelos lograron una buena capacidad de generalización,

evidenciada por la estabilización de las pérdidas de entrenamiento y prueba. Sin embargo, las predicciones en puntos de cambio abrupto en la serie temporal de tráfico mostraron desviaciones significativas, indicando áreas para mejorar.

- Los objetivos específicos planteados en el proyecto fueron mayormente alcanzados. Se implementaron y compararon diferentes arquitecturas de GNN, y se evaluó su rendimiento en términos de precisión y eficiencia computacional. Aunque hubo desafíos en la predicción de cambios abruptos en el tráfico.

### 4.3.2. Recomendaciones

- Se recomienda explorar en detalle los conceptos fundamentales de las Redes Neuronales. Para ello, se sugiere revisar una lista de reproducción en YouTube donde se pueden encontrar explicaciones sobre conceptos importantes de deep learning, redes neuronales, redes neuronales convolucionales, redes de atención, entre otros. La lista está disponible en el siguiente enlace: https://bit.ly/3yjj8mP.
- Continuar con la optimización de hiperparámetros mediante estrategias avanzadas como la búsqueda en cuadrícula o aleatoria puede mejorar aún más la precisión de los modelos, especialmente en los puntos críticos de cambio abrupto.
- Incrementar la cantidad y diversidad de los datos de entrenamiento, incluyendo datos de tráfico de diferentes épocas del año y condiciones climáticas, puede ayudar a los modelos a aprender patrones más variados y mejorar la precisión de las predicciones a largo plazo.
- Explorar enfoques híbridos que combinen las fortalezas de los modelos GCN y LSTM podría ser beneficioso. Por ejemplo, una combinación de GCN-GRU y GCN-LSTM podría mejorar la capacidad predictiva en diferentes horizontes temporales.
- Implementar un sistema de evaluación continua para monitorear el rendimiento de los modelos en tiempo real y ajustar las estrategias de predicción según las condiciones cambiantes del tráfico puede

resultar en mejoras significativas en la precisión y robustez de las predicciones.

- Realizar investigaciones adicionales para abordar las desviaciones observadas en las predicciones durante los cambios abruptos en el tráfico. Esto puede incluir el desarrollo de nuevas arquitecturas de modelos o el uso de técnicas avanzadas de aprendizaje profundo.

# Capítulo A

## Título anexo

## A.1. Código implementado en python

### A.1.1. Configuración del Entorno de Desarrollo y Herramientas Utilizadas

El código siguiente muestra un ejemplo de cómo se cargaron y prepararon los datos utilizando estas herramientas:

```
from traffic_data import METRLADatasetLoader
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
from torch_geometric.nn import GATv2Conv
from torch_geometric.data import Data, Batch
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

Listing A.1: Librerias utilizadas

### A.1.2. Código Fuente para la Carga y Preparación de Datos

```
# Cargamos los datos
loader = METRLADatasetLoader()
adj, weig, x, y = loader.get_dataset(num_timesteps_in=12,
                                     num_timesteps_out=12)
# Omitimos la variable del tiempo
x = [i[:, 0, :] for i in x]
adj = torch.tensor(adj)

print(len(x), len(y))  # Instancias para entrenamiento
print(x[0].shape)  # Cada instancia tiene 207 nodos en 12 momentos
print(y[0].shape)  # El ground truth es el grafo en el siguiente
    momento

# Convertimos la lista a un tensor aumentando una dimension  m  s
X = torch.tensor(np.array(x)).permute(0, 2, 1).unsqueeze(-1)
Y = torch.tensor(np.array(y))

# Particionamos en train y test y cargamos los datos en DataLoader
train_p = 0.8  # Porcentaje para entrenamiento
batch_size = 256

train_size = int(train_p * len(X))
test_size = len(X) - train_size

X_train, X_test = torch.split(X, [train_size, test_size])
Y_train, Y_test = torch.split(Y, [train_size, test_size])

train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
                          shuffle=True, drop_last=True)
test_loader = DataLoader(test_dataset, batch_size=int(test_size),
                         shuffle=False, drop_last=True)
```

Listing A.2: Código fuente para la carga y preparación de los datos

### A.1.3. GCN-LSTM (Graph Convolutional Network - Long Short-Term Memory)

```
class GCN_LSTM_Model(nn.Module):
```

```
    def __init__(self, node_feature_size, hidden_dim_gat,
        hidden_dim_lstm, lstm_layers, output_size, num_nodes):
        super(GCN_LSTM_Model, self).__init__()
        self.gcn_conv = GCNConv(node_feature_size, hidden_dim_gat)
        self.lstm = nn.LSTM(hidden_dim_gat * num_nodes, hidden_dim_lstm
            * num_nodes, lstm_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim_lstm * num_nodes, output_size *
            num_nodes)
        self.num_nodes = num_nodes

    def forward(self, graph_sequence, edge_index):
        batch_size, seq_len, num_nodes, node_feature_size =
            graph_sequence.size()

        # Preparar las secuencias de grafos para GCNConv
        graph_sequence = graph_sequence.view(batch_size * seq_len,
            num_nodes, node_feature_size)
        x = graph_sequence.view(-1, node_feature_size)  # (batch_size *
            seq_len * num_nodes) x node_feature_size

        # Ajustar edge_index para batch processing
        edge_index_batch = edge_index.repeat(1, batch_size * seq_len)
        offset = torch.arange(0, batch_size * seq_len * num_nodes, step
            =num_nodes, dtype=torch.long).repeat_interleave(edge_index.
            size(1))
        offset = offset.to(device)
        edge_index_batch = edge_index_batch + offset

        # Procesar todos los grafos en la secuencia de una vez con
            GATConv
        gcn_output = self.gcn_conv(x, edge_index_batch)
        gcn_output = gcn_output.view(batch_size, seq_len, num_nodes,
            -1) # -1 = hidden_dim para las features de cada nodo

        # Pasar las salidas de GCNConv a LSTM
        gcn_output = gcn_output.view(batch_size, seq_len, -1)  #
            batch_size x seq_len x (num_nodes * hidden_dim)
        lstm_out, _ = self.lstm(gcn_output)

        # Predecir el siguiente estado del grafo
        lstm_out = lstm_out[:, -1, :]  # batch_size x hidden_dim
        out = self.fc(lstm_out)
```

```
        out = out.view(batch_size, num_nodes, -1)  # batch_size x num_nodes x output_size

        return out
```

### A.1.4. GCN-GRU (Graph Convolutional Network - Gated Recurrent Unit)

```
class GCN_GRU_Model(nn.Module):
    def __init__(self, node_feature_size, hidden_dim_gcn, hidden_dim_gru, gru_layers, output_size, num_nodes):
        super(GCN_GRU_Model, self).__init__()
        self.gcn_conv = GCNConv(node_feature_size, hidden_dim_gcn)
        self.gru = nn.GRU(hidden_dim_gcn * num_nodes, hidden_dim_gru * num_nodes, gru_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim_gru * num_nodes, output_size * num_nodes)
        self.num_nodes = num_nodes

    def forward(self, graph_sequence, edge_index):
        batch_size, seq_len, num_nodes, node_feature_size = graph_sequence.size()

        # Preparar las secuencias de grafos para GCNConv
        graph_sequence = graph_sequence.view(batch_size * seq_len, num_nodes, node_feature_size)
        x = graph_sequence.view(-1, node_feature_size)  # (batch_size * seq_len * num_nodes) x node_feature_size

        # Ajustar edge_index para batch processing
        edge_index_batch = edge_index.repeat(1, batch_size * seq_len)
        offset = torch.arange(0, batch_size * seq_len * num_nodes, step=num_nodes, dtype=torch.long).repeat_interleave(edge_index.size(1))
        offset = offset.to(device)
        edge_index_batch = edge_index_batch + offset

        # Procesar todos los grafos en la secuencia de una vez con GCNConv
        gcn_output = self.gcn_conv(x, edge_index_batch)
```

```
        gcn_output = gcn_output.view(batch_size, seq_len, num_nodes,
            -1) # -1 = hidden_dim para las features de cada nodo

        # Pasar las salidas de GCNConv a GRU
        gcn_output = gcn_output.view(batch_size, seq_len, -1)  #
            batch_size x seq_len x (num_nodes * hidden_dim)
        gru_out, _ = self.gru(gcn_output)

        # Predecir el siguiente estado del grafo
        gru_out = gru_out[:, -1, :]  # batch_size x hidden_dim
        out = self.fc(gru_out)
        out = out.view(batch_size, num_nodes, -1)  # batch_size x
            num_nodes x output_size

        return out
```

Listing A.3: GCN-GRU Model in PyTorch

### A.1.5. GAT-LSTM (Graph Attention Network - Long Short-Term Memory)

```
class GATv2_LSTM_Model(nn.Module):
    def __init__(self, node_feature_size, hidden_dim_gat,
        hidden_dim_lstm, lstm_layers, output_size, num_nodes):
        super(GATv2_LSTM_Model, self).__init__()
        self.gat_conv = GATv2Conv(node_feature_size, hidden_dim_gat,
            heads=1)
        self.lstm = nn.LSTM(hidden_dim_gat * num_nodes, hidden_dim_lstm
            * num_nodes, lstm_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim_lstm * num_nodes, output_size *
            num_nodes)
        self.num_nodes = num_nodes

    def forward(self, graph_sequence, edge_index):
        batch_size, seq_len, num_nodes, node_feature_size =
            graph_sequence.size()

        # Preparar las secuencias de grafos para GATConv
        graph_sequence = graph_sequence.view(batch_size * seq_len,
            num_nodes, node_feature_size)
        x = graph_sequence.view(-1, node_feature_size)  # (batch_size *
            seq_len * num_nodes) x node_feature_size
```

```
15
16          # Ajustar edge_index para batch processing
17          edge_index_batch = edge_index.repeat(1, batch_size * seq_len)
18          offset = torch.arange(0, batch_size * seq_len * num_nodes, step
                =num_nodes, dtype=torch.long).repeat_interleave(edge_index.
                size(1))
19          offset = offset.to(device)
20          edge_index_batch = edge_index_batch + offset
21
22          # Procesar todos los grafos en la secuencia de una vez con
                GATConv
23          gat_output = self.gat_conv(x, edge_index_batch)
24          gat_output = gat_output.view(batch_size, seq_len, num_nodes,
                -1) # -1 = hidden_dim para las features de cada nodo
25
26          # Pasar las salidas de GATConv a LSTM
27          gat_output = gat_output.view(batch_size, seq_len, -1)  #
                batch_size x seq_len x (num_nodes * hidden_dim)
28          lstm_out, _ = self.lstm(gat_output)
29
30          # Predecir el siguiente estado del grafo
31          lstm_out = lstm_out[:, -1, :]  # batch_size x hidden_dim
32          out = self.fc(lstm_out)
33          out = out.view(batch_size, num_nodes, -1)  # batch_size x
                num_nodes x output_size
34
35          return out
```

### A.1.6. GAT-GRU (Graph Attention Network - Gated Recurrent Unit)

```
1  class GATv2_GRU_Model(nn.Module):
2      def __init__(self, node_feature_size, hidden_dim_gat,
           hidden_dim_gru, gru_layers, output_size, num_nodes):
3          super(GATv2_GRU_Model, self).__init__()
4          self.gat_conv = GATv2Conv(node_feature_size, hidden_dim_gat,
               heads=1)
5          self.gru = nn.GRU(hidden_dim_gat * num_nodes, hidden_dim_gru *
               num_nodes, gru_layers, batch_first=True)
6          self.fc = nn.Linear(hidden_dim_gru * num_nodes, output_size *
               num_nodes)
```

```
        self.num_nodes = num_nodes

    def forward(self, graph_sequence, edge_index):
        batch_size, seq_len, num_nodes, node_feature_size =
            graph_sequence.size()

        # Preparar las secuencias de grafos para GATConv
        graph_sequence = graph_sequence.view(batch_size * seq_len,
            num_nodes, node_feature_size)
        x = graph_sequence.view(-1, node_feature_size)  # (batch_size *
             seq_len * num_nodes) x node_feature_size

        # Ajustar edge_index para batch processing
        edge_index_batch = edge_index.repeat(1, batch_size * seq_len)
        offset = torch.arange(0, batch_size * seq_len * num_nodes, step
            =num_nodes, dtype=torch.long).repeat_interleave(edge_index.
            size(1))
        offset = offset.to(device)
        edge_index_batch = edge_index_batch + offset

        # Procesar todos los grafos en la secuencia de una vez con
            GATConv
        gat_output = self.gat_conv(x, edge_index_batch)
        gat_output = gat_output.view(batch_size, seq_len, num_nodes,
            -1) # -1 = hidden_dim para las features de cada nodo

        # Pasar las salidas de GATConv a GRU
        gat_output = gat_output.view(batch_size, seq_len, -1)  #
            batch_size x seq_len x (num_nodes * hidden_dim)
        gru_out, _ = self.gru(gat_output)

        # Predecir el siguiente estado del grafo
        gru_out = gru_out[:, -1, :]  # batch_size x hidden_dim
        out = self.fc(gru_out)
        out = out.view(batch_size, num_nodes, -1)  # batch_size x
            num_nodes x output_size

        return out
```

# Referencias bibliográficas


[1] Analytics Steps. Common architectures of convolutional neural networks. 2022. Último acceso el 22 de junio de 2024.

[2] Abid Ali Awan. A comprehensive introduction to graph neural networks (gnns), 2023. Accessed: 2024-07-17.

[3] Biobook. ¿cómo funcionan las redes neuronales? 2024. Accessed: 2024-05-25.

[4] Alexei Botchkarev. A new typology design of performance metrics to measure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 14:045–076, 2019.

[5] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Aprendizaje profundo geométrico: Más allá de los datos euclidianos. *IEEE Signal Processing Magazine*, 34(4):18–42, Jul 2017.

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Redes espectrales y redes localmente conectadas en grafos. *arXiv preprint arXiv:1312.6203*, 2013. [En línea]. Disponible: https://arxiv.org/abs/1312.6203.

[7] Diego Calvo. Red neuronal convolucional cnn, 2017. Accessed: 2024-07-17.

[8] DataScience.eu. Comprensión de las redes de lstm. 2021. Accessed: 2024-05-25.

[9] Xiaolong Fan, Maoguo Gong, Yue Wu, A. K. Qin, and Yu Xie. Propagation enhanced neural message passing for graph representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(2):1952–1964, 2023.

[10] Cristian Vilanova González. Redes neuronales sobre grafos (gnn): una prueba de concepto para sistemas recomendadores. Technical report, Escuela Superior de Ingeniería y Tecnología, Universidad de La Laguna, 2023. La Laguna, 14 de julio de 2023, Accessed: 2024-07-17.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[12] I. Gómez. Introducción a las redes neuronales. 2020. Accessed: 2024-05-25.

[13] Wei Huang, Hong Hao Song, Guang, and Kun Xie. Arquitectura profunda para la predicción del flujo de tráfico: Redes de creencias profundas con aprendizaje multitarea. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2191–2201, Oct 2014.

[14] Yanan Jia, Jian Wu, and Yan Du. Predicción de la velocidad del tráfico utilizando método de aprendizaje profundo. In *Proceedings of the IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1217–1222, Nov 2016.

[15] Thomas N. Kipf and Max Welling. Clasificación semi-supervisada con redes convolucionales de grafos. *arXiv preprint arXiv:1609.02907*, 2016. [En línea]. Disponible: https://arxiv.org/abs/1609.02907.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Aprendizaje profundo. *Nature*, 521(7553):436–444, May 2015.

[17] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: ¿pueden las gcns ser tan profundas como las cnns? *arXiv preprint arXiv:1904.03751*, 2019. [En línea]. Disponible: https://arxiv.org/abs/1904.03751.

[18] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Red neuronal recurrente de convolución por difusión: Previsión del tráfico impulsada por datos. *arXiv preprint arXiv:1707.01926*, 2017. [En línea]. Disponible: https://arxiv.org/abs/1707.01926.

[19] Yisheng Lv, Yanjie Duan, Wenbo Kang, Zan Li, and Fei-Yue Wang. Predicción del flujo de tráfico con grandes datos: un enfoque de aprendizaje profundo. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, Apr 2015.

[20] Xiaolei Ma, Zhaoguo Dai, Zhenyu He, Jian Ma, Yi Wang, and Yinhai Wang. Aprendiendo el tráfico como imágenes: una red neuronal convolucional profunda para la predicción de velocidad en redes de transporte de gran escala. *Sensors*, 17(4):818, Apr 2017.

[21] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, page 4564–4573, 2021.

[22] Luis Serrano. Redes neuronales recurrentes. 2023. Accedido: 24-06-2024.

[23] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, March 2020.

[24] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[25] Muhammad Uzzaman, Chandan Debnath, Md Ashraf Uddin, Manowarul Islam, Md. Alamin Talukder, and Shamima Parvez. Lrcn based human activity recognition from video data. *SSRN Electronic Journal*, 01 2022.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5998–6008, 2017.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lió, and Yoshua Bengio. Graph attention networks. 2018.

[28] Jian Wang, Qin Gu, Jian Wu, Guang Liu, and Zheng Xiong. Predicción de la velocidad del tráfico y exploración de la fuente de congestión: Un método de aprendizaje profundo. In *Proceedings of the IEEE 16th International Conference on Data Mining (ICDM)*, pages 499–508, Dec 2016.

[29] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Evaluación empírica de activaciones rectificadas en redes convolucionales. 2015. [En línea]. Disponible: https://arxiv.org/abs/1505.00853.

[30] Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. A comprehensive survey of graph neural networks for knowledge graphs. *IEEE Access*, 10:75729–75741, 2022.

[31] Firat Yilmaz and Engin Yildiztepe. Statistical evaluation of deep learning models for stock return forecasting. *Computational Economics*, 63, 10 2022.

[32] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, 2024.

[33] Hongwei Yu, Zhenghua Wu, Shaowu Wang, Yi Wang, and Xiaolei Ma. Redes convolucionales recurrentes espacio-temporales para la predicción del tráfico en redes de transporte. *Sensors*, 17(7):1501, Jun 2017.

[34] Qingtian Zeng, Yu Liang, Geng Chen, Hua Duan, and Chunguo Li. Noise prediction of chemical industry park based on multi-station

prophet and multivariate lstm fitting model. *EURASIP Journal on Advances in Signal Processing*, 2021, 10 2021.

[35] Chenhan Zhang, James J. Q. Yu, and Yi Liu. Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. *IEEE Access*, 7:166246–166256, 2019.

[36] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, September 2020.

[37] T. Zhou. Deep learning models for route planning in road networks. 2018.