



UNIVERSIDAD DE GRANADA

ANÁLISIS Y PRONÓSTICO DE INCIDENCIAS EN REDES ELÉCTRICAS A TRAVÉS DE REDES NEURONALES TEMPORALES SOBRE GRAFOS

MARÍA AGUADO MARTÍNEZ

Trabajo Fin de Máster

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

Tutores

Miguel Molina Solana
Juan Gómez Romero

E.T.S. INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, 5 de septiembre de 2024

Agradecimientos

Me gustaría dedicar este trabajo a toda la gente que ha confiado en mí este año, incluso cuando yo misma tenía dudas. No ha sido fácil, pero cada dificultad ha sido una ocasión para aprender, mejorar y demostrar que con esfuerzo y dedicación se pueden superar incluso los retos más complejos.

Por supuesto, tengo que agradecer a mis dos tutores, D. Miguel Molina y D. Juan Gómez, por abrirme las puertas de su equipo y brindarme la oportunidad de experimentar de manera directa el mundo de la investigación. Vuestra guía y apoyo han sido fundamentales para mi desarrollo tanto profesional como personal.

DECLARACIÓN DE ORIGINALIDAD DEL TFM

Yo, **María Aguado Martínez**, con DNI 71168499J, declaro que el presente Trabajo de Fin de Máster es original, no habiéndose utilizado fuentes sin ser citadas debidamente. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la Normativa de Evaluación y de Calificación de los estudiantes de la Universidad de Granada, de 20 de mayo de 2013, esto conllevará automáticamente la calificación numérica de cero [...] independientemente del resto de las calificaciones que el estudiante hubiera obtenido. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagien.

Para que así conste lo firmo el 5 de septiembre de 2024

Granada, a 5 de septiembre de 2024

D. Miguel Molina Solana y Juan Gómez Romero, profesores del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Análisis y pronóstico de caídas en cadena en redes eléctricas a través de Redes Neuronales sobre Grafos (GNNs)*, ha sido realizado bajo nuestra supervisión por María Aguado Martínez, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda. Y para que conste, expedimos y firmamos el presente informe en Granada a 5 de septiembre de 2024.

Granada, a 5 de septiembre de 2024

Análisis y pronóstico de incidencias en redes eléctricas a través de Redes Neuronales Temporales sobre Grafos

María Aguado Martínez

Palabras clave

Series temporales, Eficiencia Energética, RNN, GNN, Redes Neuronales Temporales sobre Grafos, Predicción de Incidencias, Clasificación de Incidencias, Representación de Grafos

Resumen

Las transformaciones climáticas significativas que hemos observado y anticipado para el siglo XXI, incluyendo el calentamiento global, consecuencia de más de un siglo de emisiones netas de gases de efecto invernadero, reflejan cambios profundos que se han manifestado durante los últimos 65 años.

Las emisiones de gases de efecto invernadero relacionadas con la energía son el mayor contribuyente al calentamiento climático en los últimos años. Por este motivo, el papel de tecnologías avanzadas se vuelve fundamental para abordar y mitigar las incidencias en las redes eléctricas, pues éstas son utilizadas en la mayoría de los sectores, y tienen el potencial de generar un impacto positivo en ellos.

En este trabajo se ha experimentado con un sistema de transmisión de una red eléctrica en simulaciones con distintos tipos de perturbaciones en la red, en particular en el caso de cortocircuito de generador, cortocircuito de nodo, error de nodo, cortocircuito de rama y error de rama. Se ha modelado cada simulación como una secuencia de grafos, y se ha buscado experimentar con tecnologías de redes neuronales específicas para grafos.

El objetivo ha sido solucionar dos problemas principales: el primero, la identificación de tipos de incidencias utilizando datos históricos de la red; el segundo, el desarrollo de un sistema predictivo que, para un tipo específico de incidencia, proyecte el estado futuro de la red, de cara a poder anticipar condiciones de inestabilidad en la red eléctrica para facilitar acciones preventivas eficaces.

Se han incluido en la experimentación arquitecturas de red neuronal de grafos basadas en redes recurrentes (AGCRN, DyGrEncoder, MPNN-LSTM, EvolveGCN y DCRNN), y arquitecturas basadas en mecanismos de atención (ASTGCN, MSTGCN, MTGNN y STConv), comparándolas por medio de métricas adecuadas en cada caso con una arquitectura LSTM simple.

En el caso de la regresión, los resultados más consistentes en ambos problemas se encuentran con una arquitectura LSTM, sin embargo, también destacan favorablemente

los modelos DCRNN y DyGrEncoder. Por otro lado, en el problema de clasificación, las arquitecturas MSTGCN y MTGNN proporcionan un rendimiento superior al de LSTM en la identificación de todas las clases, a excepción del error de nodo, posiblemente debido a una alta variabilidad en las simulaciones pertenecientes a dicha clase.

Forecasting and Classification of Incidents in Electrical Networks Using Temporal Neural Networks on Graphs

María Aguado Martínez

Keywords Time series, Energy Efficiency, RNN, GNN, Temporal Neural Networks on Graphs, Representation Learning, Incident Prediction, Incident Classification

Abstract

The significant climate transformations we have observed and anticipated for the 21st century, including global warming, as a consequence of over a century of net greenhouse gas emissions, reflect profound changes that have manifested over the past 65 years.

Energy-related greenhouse gas emissions are the largest contributor to climate warming in recent years. For this reason, the role of advanced technologies becomes crucial in addressing and mitigating incidents in electrical networks, as these networks are utilized in most sectors and have the potential to generate a positive impact on them.

In this study, experiments have been conducted with a power grid transmission system through various network disturbance simulations, specifically focusing on five types of issues: generator trip, node trip, node fault, branch trip, and branch fault. Each simulation has been modeled as a sequence of graphs, and the experiments have been conducted using neural network technologies specifically designed for dynamic graphs.

The objective has been to solve two main problems: first, the identification of incident types using historical network data; second, the development of a predictive system that, for a specific type of incident, predicts the future state of the network, aiming to anticipate conditions of instability in the electrical grid to facilitate effective preventive actions.

The experimentation included graph neural network architectures based on recurrent networks (AGCRN, DyGrEncoder, MPNN-LSTM, EvolveGCN, and DCRNN), and architectures based on attention mechanisms (ASTGCN, MSTGCN, MTGNN, and STConv), comparing each one of them with a simple LSTM architecture, through suitable metrics in each case.

In the case of regression, the most consistent results across both problems are found with an LSTM architecture; however, the DCRNN and DyGrEncoder models also stand out favorably. On the other hand, in the classification problem, the MSTGCN and MTGNN architectures provide superior performance compared to LSTM in identifying all classes, except for the node error, possibly due to high variability in the simulations belonging to that class.

ÍNDICE GENERAL

I. INTRODUCCIÓN Y ANTECEDENTES	11
1. INTRODUCCIÓN	12
1.1. Objetivos	13
1.2. Temporización	14
1.3. Organización de la memoria	14
2. ANTECEDENTES	17
2.1. Trabajos Previos	17
2.2. Aprendizaje Automático	20
2.2.1. Fundamentos del Aprendizaje Automático	20
2.2.2. Tipos de Aprendizaje Automático	21
2.2.3. Aprendizaje Supervisado	23
II. MARCO TEÓRICO	27
3. REDES NEURONALES DE GRAFOS	28
3.1. Introducción a los Grafos	28
3.1.1. Formulación matemática	29
3.1.2. Aprendizaje Automático en Grafos	30
3.2. Descripción de las GNNs	32
3.2.1. Motivación. Aproximaciones iniciales	32
3.2.2. Codificador y Decodificador	33
3.2.3. Introducción a las GNN	36
3.3. Formalización de GNNs	37
3.3.1. Descripción detallada	37
3.3.2. Función de agregación	38
3.3.3. Función de actualización	42
3.3.4. Características de aristas	44
4. REDES NEURONALES EN SERIES TEMPORALES	46
4.1. Modelado de Series Temporales	46
4.1.1. Series Temporales	46
4.1.2. Redes Neuronales Convolucionales	48
4.1.3. Redes Neuronales Recurrentes	50
4.1.4. Arquitecturas Avanzadas de RNN	53
5. REDES NEURONALES DE GRAFOS TEMPORALES	58
5.1. Grafos Temporales	58
5.1.1. Tipos de Grafos Dinámicos	58
5.2. Redes Neuronales de Grafos Temporales	60
5.2.1. Redes basadas en RNN	60
5.2.2. Redes basadas en CNN	63
5.2.3. Redes Profundas	63

III. METODOLOGÍA	65
6. PRESENTACIÓN DE LOS DATOS Y SOFTWARE DESARROLLADO	66
6.1. Origen de los datos	66
6.1.1. Plataforma de simulación	67
6.1.2. Generación de series temporales de grafos	67
6.2. Descripción del problema	74
6.3. Desarrollo del software	77
6.3.1. Estructura modular	77
6.3.2. Clase de Entrenamiento Centralizada	78
6.3.3. Cargador de Datos	78
7. ALGORITMOS BASADOS EN MECANISMOS RECURRENTES	80
7.1. AGCRN	80
7.1.1. Módulos GCN avanzados	81
7.1.2. Predicciones de múltiples pasos	82
7.2. DyGrEncoder	83
7.2.1. Redes GGNN	83
7.2.2. Descripción detallada de la arquitectura	84
7.3. MPNN-LSTM	85
7.3.1. MPNNs	85
7.3.2. Integración de LSTMs	87
7.4. EvolveGCN	87
7.4.1. Evolución de Pesos	87
7.5. DCRNN	89
7.5.1. Modelado de dependencias espaciales	89
7.5.2. Modelado de dependencias temporales	90
8. ALGORITMOS BASADOS EN MECANISMOS DE ATENCIÓN	92
8.1. ASTGCN	92
8.1.1. Mecanismo de atención espaciotemporal	92
8.1.2. Convolución espaciotemporal	95
8.1.3. Fusión de componentes	97
8.2. MSTGCN	98
8.3. MTGNN	98
8.3.1. Capa de Aprendizaje del Grafo	98
8.3.2. Módulo de Convolución del Grafo	100
8.3.3. Módulo de Convolución Temporal	101
8.4. STGCN	103
8.4.1. Capa de Convolución Espacial	103
8.4.2. Capa de Convolución Temporal	104
8.4.3. Bloques Convolucionales Espacio-Temporales (ST-Conv)	105
IV. EXPERIMENTACIÓN	106
9. REGRESIÓN. PREDICCIÓN DE ESTADOS FUTUROS	107
9.1. Cortocircuito de generador	107
9.2. Cortocircuito de nodo	111
9.3. Error de nodo	114

9.4.	Cortocircuito de rama	115
9.5.	Error de rama	117
9.6.	Conclusiones	119
10.	CLASIFICACIÓN DEL TIPO DE INCIDENCIA	121
10.1.	Resultados generales	122
10.2.	Resultados por algoritmo	124
11.	CONCLUSIONES Y TRABAJO FUTURO	128
V.	APÉNDICES	134
A.	REDES NEURONALES	135
A.1.	Fundamentos de las Redes Neuronales	135
A.1.1.	Ejemplo. Red Neuronal Simple.	136
A.1.2.	Redes con más capas	138
A.2.	Diseño de una red neuronal	139
A.2.1.	Función de activación	139
A.2.2.	Función de pérdida	141
A.2.3.	No linearidad	143
A.3.	Entrenamiento de Redes. Backpropagation	143
B.	REGRESIÓN. RESULTADOS POR SIMULACIONES.	149
B.1.	Cortocircuito de generador	149
B.2.	Cortocircuito de nodo	155
B.3.	Error de nodo	158
B.4.	Cortocircuito de rama	168
B.5.	Error de rama	171

Parte I

INTRODUCCIÓN Y ANTECEDENTES

Exposición general de los objetivos e hipótesis del estudio. Justificación de las técnicas de análisis utilizadas.

INTRODUCCIÓN

Las transformaciones climáticas significativas que hemos observado y anticipado para el siglo XXI, incluyendo el calentamiento global, reflejan cambios profundos que se han manifestado durante los últimos 65 años. El cambio climático es un desafío complejo a nivel intergubernamental que influye en disciplinas ecológicas, ambientales, socio-políticas y socio-económicas, y se caracteriza por los cambios a largo plazo en las tendencias de temperatura y precipitaciones, junto con otros efectos en la presión o la humedad en el entorno [1].

El cambio climático provocado por el hombre es consecuencia de más de un siglo de emisiones netas de gases de efecto invernadero (GHG, por sus siglas en inglés) derivadas del uso de energía, los estilos de vida y los patrones de consumo y producción. Alcanzar objetivos climáticos ambiciosos, como limitar el calentamiento global a menos de 2 °C o incluso 1.5 °C, por lo tanto, requiere cambios radicales en los métodos de producción y en los estilos de vida de la humanidad [36].

Todas las rutas proyectadas a nivel global que limitan el calentamiento a 2°C o menos para el año 2100 implican reducciones rápidas, profundas y en la mayoría de los casos inmediatas de las emisiones de gases de efecto invernadero en todos los sectores. Las reducciones de las emisiones en la industria, el transporte, los edificios y las zonas urbanas pueden lograrse mediante una combinación de mejoras en eficiencia energética y mediante la transición hacia tecnologías y fuentes de energía con bajas emisiones de gases de efecto invernadero [21].

Además del efecto de las mejoras en eficiencia energética en los distintos sectores, si nos centramos únicamente en el sector energético, es un hecho ineludible que las emisiones de GHG relacionadas con la energía son el mayor contribuyente al calentamiento climático en los últimos años. En este sector, las opciones de adaptación del sistema energético más viables pasan por fortalecer la resiliencia de la infraestructura y asegurar sistemas de energía fiables, permitiendo que los sistemas eléctricos soporten y se recuperen de interrupciones causadas por eventos climáticos extremos o fallos técnicos.

En este contexto, el papel de tecnologías avanzadas se vuelve fundamental para abordar y mitigar las incidencias en las redes eléctricas [24], pues éstas son utilizadas en la mayoría de los sectores, y tienen el potencial de generar un impacto positivo en ellos. Considerando además que los sistemas eléctricos son especialmente suscepti-

bles a las fluctuaciones causadas por cambios climáticos severos, es esencial disponer de herramientas capaces de predecir y gestionar estas incidencias de manera precisa y eficiente.

En este trabajo, se modelará un sistema de transmisión de una red eléctrica como un grafo, y se buscará experimentar con tecnologías de redes neuronales específicas para grafos.

Nos enfocaremos principalmente en dos aplicaciones dentro del contexto de la eficiencia energética en redes eléctricas. La primera se orienta a la identificación de tipos de incidencias utilizando datos históricos de la red, lo cual nos permitirá analizar los fallos más frecuentes y desarrollar soluciones sistemáticas. La segunda aplicación se centra en el desarrollo de un sistema predictivo que, para un tipo específico de incidencia, proyecte el estado futuro de la red. El objetivo principal de este sistema es anticipar condiciones de inestabilidad en la red eléctrica para facilitar acciones preventivas eficaces.

1.1 OBJETIVOS

Los objetivos definidos y abordados en este proyecto fin de máster son los siguientes:

Objetivo 1
Estudiar y analizar el impacto del uso de tecnologías avanzadas en la eficiencia y fiabilidad del suministro eléctrico.
Objetivo 2
Profundizar en la teoría de Redes Neuronales sobre Grafos (GNNs), en particular las TGNs, para entender su aplicación en redes eléctricas.
Objetivo 3
Investigar y seleccionar herramientas para analizar datos en grafos y representar la estructura de la red eléctrica. En particular, obtener una representación adecuada y sistemática para el procesamiento de las series temporales de grafos.
Objetivo 4
Utilizar técnicas avanzadas de aprendizaje automático sobre grafos para asociar el tipo de incidencia adecuada a secuencias de estados de la red.

Objetivo 5

Utilizar técnicas avanzadas de aprendizaje automático sobre grafos para predecir el estado de la red en instantes futuros, en diferentes tipos de perturbaciones.

Objetivo 6

Evaluar el rendimiento de los modelos predictivos avanzadas, y compararlos con los resultados obtenidos al usar tecnologías más sencillas.

1.2 TEMPORIZACIÓN

La Figura 1 muestra la organización del proyecto mediante un Diagrama de Gantt, con un total de 721 horas dedicadas, como parte de un contrato de investigación en la Universidad de Granada dentro del proyecto IA4TES, de 6 meses a jornada completa.

Es necesario destacar la simultaneidad de distintas tareas independientes, como por ejemplo la documentación del marco teórico y la implementación y ejecución de los distintos experimentos de predicción. Este trabajo se ha realizado en el contexto del Trabajo de Fin de Máster del Máster Universitario en Ciencia de Datos e Ingeniería de Computadores con un esfuerzo previsto de 12ECTS.

Desarrollo del proyecto

Todo el software desarrollado en el marco de este proyecto se encuentra alojado en un repositorio público en GitHub, accesible en https://github.com/maaguado/GNNs_PowerGraph. Con el objetivo de facilitar la implementación del software en distintas máquinas, se ha utilizado un contenedor Docker, lo que permite una configuración y despliegue eficientes y reproducibles en cualquier entorno compatible. Aunque la mayoría de los experimentos se realizaron en una máquina local, algunos experimentos clave se llevaron a cabo en instancias de Máquina Virtual proporcionadas por Google Cloud Platform, aprovechando así la escalabilidad y recursos computacionales que ofrece la nube.

1.3 ORGANIZACIÓN DE LA MEMORIA

Esta memoria está organizada de la siguiente manera:

En primer lugar (Capítulo 2) realizaremos una revisión del estado actual de la tecnología de predicción en redes eléctricas y de los principios básicos del Aprendizaje Supervisado.



Figura 1: Organización temporal de las distintas tareas del trabajo

Más adelante, nos adentraremos en el estudio de los fundamentos teóricos de los modelos con los que se realizará la experimentación. Introduciremos conceptos como los grafos y las Redes Neuronales de Grafos (GNNs) (Capítulo 3), las series temporales, las Redes Neuronales Temporales (RNNs) (Capítulo 4), y finalmente veremos en qué consiste la arquitectura básica de las Redes Neuronales Temporales sobre Grafos (Capítulo 5).

Después, introduciremos la metodología usada en el trabajo. Comenzaremos presentando los datos utilizados en los experimentos (Capítulo 6), y después procederemos a estudiar en detalle cada una de las arquitecturas que pondremos en práctica más tarde (Capítulos 7 y 8).

Por último, presentamos los experimentos llevados a cabo como parte del estudio. Por una parte, tendremos los experimentos realizados en el marco del problema de regresión (Capítulo 9), y por otra parte tendremos los experimentos de clasificación (Capítulo 10). Se analizarán tanto los resultados individuales de cada una de las arquitecturas, junto con los desafíos principales que cada una de ellas ha supuesto, como los resultados en conjunto, de cara a analizar qué arquitectura sería la más adecuada en cada caso.

El documento termina con una discusión de las conclusiones más relevantes, así como indicando varias posibles líneas de trabajo futuro (Capítulo 11).

Además, es necesario mencionar que, con el objetivo de reducir la extensión del trabajo principal, se han incluido dos apéndices, que contienen, por una parte, los fundamentos teóricos de las Redes Neuronales (Apéndice A), y por otra, un estudio detallado de los resultados obtenidos en cada tipo de incidencia (Apéndice B).

2

ANTECEDENTES

En este capítulo introduciremos los antecedentes del trabajo, comenzando por una revisión de estudios previos en el sector en problemas similares (en particular, detección de anomalías en redes eléctricas). Después, estudiaremos conceptos básicos del Aprendizaje Automático, desde sus fundamentos hasta alguna de sus aplicaciones prácticas.

2.1 TRABAJOS PREVIOS

Tal y como hemos visto anteriormente, el desarrollo de tecnologías avanzadas resulta crucial en la mejora de la eficiencia energética en múltiples sectores. Sin embargo, para lograr una transición energética efectiva, es fundamental no solo innovar, sino también asegurar que las infraestructuras sean lo suficientemente resilientes. En el sector energético, las opciones de adaptación más viables pasan por fortalecer la infraestructura energética y asegurar sistemas fiables, capaces de soportar y recuperarse de interrupciones causadas por eventos climáticos extremos o fallos en la red.

En este trabajo nos centraremos en el segundo tipo de problema, y para ello es necesario clarificar qué consideramos como fallo en la red. Podemos definir un fallo en una red eléctrica como un estado en el que un componente del sistema se comporta de manera incorrecta, generalmente manifestando un flujo de corriente anormal. El ejemplo más común de esto es el cortocircuito, en el que la corriente supera los niveles operativos normales.

Además, este tipo de error inicial puede desencadenar una serie de fallos adicionales, propagándose rápidamente a lo largo de la red, lo que denominaremos fallos en cascada. Los fallos en cascada son especialmente peligrosos para el funcionamiento de la red, debido a que pueden causar una interrupción masiva del servicio.

Con la evolución hacia redes inteligentes, se busca integrar tecnologías de la información y comunicaciones en todos los aspectos de la generación, transmisión, distribución y consumo de electricidad, con el fin de minimizar el impacto ambiental, mejorar la eficiencia, y resiliencia, y reducir costos [26].

La vulnerabilidad de las redes ha sido objeto de estudio exhaustivo en los últimos años. Sin embargo, la mayoría de los trabajos computacionales no tienen en cuenta las interacciones en las redes eléctricas ni los fallos en cascada. Además, las investigaciones sobre cascadas únicamente se han centrado en modelos probabilísticos de propagación de fallos [42], donde se asumía que el fallo en una línea o nodo conllevaba un fallo con cierta probabilidad en las líneas o nodos cercanos. No obstante, los eventos reales de cascadas y los estudios de simulación muestran que la propagación real difiere de lo predicho por dichos modelos [7]. Más tarde, en [5] se modelaron los fallos en las redes como fallos de nodos y se estudiaron numéricamente las propiedades de las cascadas en redes reales, redes espacialmente incrustadas y redes aleatorias.

En la investigación más reciente sobre la estabilidad en las redes eléctricas, se han desarrollado modelos y algoritmos significativos para el análisis y la prevención de fallos en cascada. A continuación veremos alguno de ellos.

Para empezar, en [33] se utilizan el modelo de flujos de energía en corriente continua (DC) y el algoritmo de evolución de fallos en cascada (CFE). El modelo DC simplifica el modelo de corriente alterna (AC) representando la red como un grafo no dirigido, donde los nodos son los buses y las aristas son las líneas de transmisión. Este modelo calcula los flujos de energía basándose en la conservación del flujo en cada nodo y la relación entre los flujos y los ángulos de fase determinados por la reactancia de las líneas.

El algoritmo de fallos en cascada (CFE, por sus siglas en inglés) asume que cada línea de transmisión en la red tiene una capacidad máxima de flujo y que los fallos se propagan en rondas. En cada ronda, se identifican las líneas que han fallado y se eliminan del grafo de la red, lo cual puede desconectar partes de esta. En cada componente desconectado, se ajusta la demanda y la oferta de energía para mantener el equilibrio, mediante la simulación de la reducción de carga o la disminución de generación. Después de ajustar los flujos de energía, se identifica el siguiente conjunto de fallos.

Este ciclo se repite hasta que no se producen más errores y la red se estabiliza. El algoritmo permite así simular la evolución en cascada y estudiar cómo una fallo inicial puede llevar a fallos adicionales en otras partes de la red.

Por otro lado, en [9] se desarrolla un método para predecir problemas en las redes eléctricas, utilizando conceptos originados en la física. La idea clave se basa en analizar situaciones de baja probabilidad que podrían causar problemas serios en la red, usando un concepto llamado *instantones*. Los instantones son configuraciones especiales que representan casos extremos donde la red podría fallar bajo ciertas condiciones poco comunes.

En física, este enfoque se usa para estudiar cómo se comportan sistemas muy complejos cuando se encuentran en condiciones extremas. Al aplicar estos conceptos a las redes eléctricas, se busca entender y predecir situaciones en las que la red podría fallar, aunque estas situaciones sean muy improbables.

La técnica se ha adaptado de disciplinas como la teoría de campos y la hidrodinámica estadística, que tratan con grandes conjuntos de posibles estados y buscan los más significativos entre ellos, aquellos que pueden indicar la presencia de un problema serio o un fallo inminente. En el contexto de las redes eléctricas, estos métodos ayudan a identificar configuraciones críticas que son difíciles de detectar con técnicas convencionales, como las simulaciones de Monte Carlo, que generalmente requieren mucho tiempo y recursos computacionales para explorar todas las posibles situaciones.

Este enfoque ha demostrado ser útil también en la corrección de errores en sistemas de comunicación, donde ayuda a encontrar errores raros que pueden afectar el desempeño de los sistemas de decodificación, que de otro modo funcionarían correctamente bajo condiciones normales. La idea es guiar las búsquedas hacia áreas problemáticas específicas dentro de un gran espacio de posibilidades, optimizando así los recursos y mejorando la eficiencia de las redes eléctricas al anticipar y mitigar potenciales fallos antes de que ocurran.

Otra vía de investigación es la que utiliza la teoría de redes complejas para una representación más precisa y realista de los sistemas de transmisión eléctrica. En [40] se abordan específicamente las interacciones entre fallos por sobrecarga y fallos ocultos en los dispositivos de protección, un área que ha sido menos estudiada en modelos anteriores, que se centraban principalmente en las características topológicas o en modelos estadísticos simplificados de carga y capacidad.

El modelo propuesto en dicho trabajo integra características eléctricas detalladas, lo cual es crucial para una simulación precisa de las dinámicas de red en situaciones de fallo. Se utiliza una combinación de factores de distribución del flujo de potencia (denominados PTDF, por sus siglas en inglés) y modelos de carga de nodo basados en las propiedades eléctricas y no simplemente en la conectividad o la topología de la red.

Los PTDF son utilizados para determinar cómo se distribuye la energía a través de las líneas de transmisión, basados en la capacidad máxima de inyección de los nodos generadores y las demandas de los nodos de carga. Esto no solo refleja cómo el flujo de potencia es manejado en la práctica por los sistemas de transmisión sino que también permite modelar de manera más precisa cómo los fallos se propagan a través de la red, especialmente cuando se combinan con los fallos ocultos en los dispositivos de protección, que históricamente han sido una causa significativa de fallos en cascada y apagones extendidos [32].

El modelo presentado también aborda la capacidad de los nodos de red para manejar fluctuaciones en la demanda de energía, considerando las interacciones entre las cargas de los nodos y los fallos ocultos. Esto se logra al ajustar la carga operativa de cada nodo según los límites de capacidad máxima establecidos, asegurando que las operaciones dentro de la red permanezcan dentro de parámetros seguros incluso bajo condiciones de estrés.

En los últimos años, ha aumentado el interés por aplicar técnicas de aprendizaje automático para predecir fallos en cascada en redes eléctricas. Por ejemplo, en [16]

se utiliza un modelo de fallos en cascada basado en flujo de carga DC (Corriente Continua) para generar un conjunto de datos, que luego se usa para entrenar un algoritmo de Máquinas de Vectores de Soporte (SVM, por sus siglas en inglés).

De manera similar, en [47] se utilizan redes neuronales convolucionales para clasificar los fenómenos que desencadenan fallos en cascada en el sistema IEEE 39-bus, que es un modelo estándar utilizado para pruebas de simulación en el estudio de redes eléctricas.

Además, en [38] se propone el uso de Redes Neuronales sobre Grafos para clasificar fallos en redes eléctricas, lo que supone un paso innovador en el campo.

En este estudio, avanzamos en esta línea de investigación mediante experimentación con arquitecturas más avanzadas de Redes Neuronales sobre Grafos, específicamente las Redes Temporales sobre Grafos. Estas arquitecturas avanzadas nos facilitan no solo clasificar secuencias temporales de grafos, sino también procesarlas como series temporales de grafos dinámicos, lo cual nos permite hacer predicciones futuras, ampliando significativamente nuestras capacidades de modelado en redes eléctricas, pues todas ellas pueden considerarse una secuencia de grafos.

2.2 APRENDIZAJE AUTOMÁTICO

Antes de entrar de lleno en la descripción de estas técnicas, daremos unas breves pinceladas sobre el Aprendizaje Automático en general.

2.2.1 *Fundamentos del Aprendizaje Automático*

El Aprendizaje Automático (*Machine Learning*) es un campo de la Inteligencia Artificial, cuyo objetivo es permitir que un sistema aprenda y mejore de forma autónoma, mediante datos previamente recopilados, y utilizar dicho conocimiento para realizar tareas sin ser explícitamente programado para ello.

Aunque este concepto es relativamente nuevo, podría decirse que toda la ciencia pasa por ajustar modelos a los datos, pues cuando Galileo o Newton diseñaban sus experimentos y realizaban observaciones, estaban infiriendo conocimiento de los datos, estableciendo teorías que les permitirían hacer predicciones más adelante [3].

Desde la década de los cincuenta, se empezaron a obtener grandes avances en cuanto a algoritmos y métodos de ajuste a los datos, comenzando desde la regresión lineal hasta algoritmos más complejos como *Support Vector Regression* (SVM), pero no fue hasta los años setenta que se empezaron a investigar lo que conocemos a día de hoy como redes neuronales [11].

Los avances continuaron, y a principios del siglo XXI comenzó lo que podríamos describir como la era de la Inteligencia Artificial, impulsada por tres tendencias sincrónas que generaron un efecto sinérgico.

La primera tendencia es el Big Data. Tanto empresas como organizaciones a nivel mundial comenzaron a recopilar y almacenar enormes volúmenes de datos, dicha cantidad de datos se volvió tan masiva que nuevas aproximaciones surgieron por necesidad práctica más que por curiosidad científica.

La segunda tendencia fue la reducción del costo de la computación y la memoria paralela. Esta tendencia se manifestó cuando Google presentó su tecnología MapReduce, Nvidia hizo avances significativos en el mercado de las GPU, y el costo de la RAM disminuyó significativamente, lo que abrió la posibilidad de trabajar con grandes cantidades de datos en la memoria.

Finalmente, la tercera tendencia fue el desarrollo de nuevos algoritmos de aprendizaje automático profundo, que heredaron y desarrollaron la idea original del perceptrón, solventando las limitaciones principales de este tipo de algoritmos, presentadas en detalle en [4].

Actualmente, el Aprendizaje Automático está presente en multitud de campos y disciplinas, como por ejemplo en el sector sanitario, impulsando el desarrollo de medicinas y tratamientos [37], o mejorando el diagnóstico en pacientes con enfermedades neurodegenerativas [29].

Existen más aplicaciones emergentes en áreas como la conducción autónoma, donde los algoritmos de aprendizaje automático son fundamentales para la toma de decisiones en tiempo real y la mejora continua de la seguridad en las carreteras [28].

Además, el aprendizaje automático está revolucionando la industria financiera al predecir riesgos, detectar fraudes y ofrecer recomendaciones personalizadas de inversión [12].

A continuación abordaremos los diferentes tipos de aprendizaje automático, cada uno representando una forma única de extraer conocimiento de los datos. Estos enfoques surgieron para adaptarse a distintas condiciones y necesidades, ofreciendo herramientas específicas para explorar y comprender la información contenida en conjuntos de datos diversos.

2.2.2 *Tipos de Aprendizaje Automático*

Comenzaremos diferenciando tres tipos de Aprendizaje Automático: aprendizaje supervisado (que será en el que nos centraremos en este trabajo), aprendizaje no supervisado, y aprendizaje por refuerzo. Para cada uno de ellos estudiaremos cómo obtienen conocimiento a partir de los datos, y veremos ejemplos de problemas que pueden resolver.

En primer lugar, comenzamos por el **Aprendizaje Supervisado**. En esta categoría se encuentran los modelos que buscan obtener un mapeo entre unas variables de entrada X y unas variables de salida Y [10]. Para poder extraer conocimiento, es necesario partir de datos previamente etiquetados, es decir, datos que presenten tanto las variables de entrada como las variables de salida, que denominaremos datos de

entrenamiento. Los usaremos para definir concretamente la relación entre X e Y , de manera que posteriormente el modelo sea capaz de predecir el valor de Y de manera precisa para datos que no se han observado previamente.

El Aprendizaje Supervisado se destaca como una herramienta poderosa en la resolución de una amplia gama de problemas en diversos campos, lo que lo convierte en la subclase más significativa dentro del Aprendizaje Automático. Para continuar con los ejemplos previos, este enfoque se utiliza para tareas como clasificar imágenes médicas o información genética en el diagnóstico de enfermedades, predecir el nivel de riesgo de un cliente en el sector bancario o detectar fraudes financieros.

Dado que esta subclase es el enfoque principal de este trabajo, más adelante introduciremos los dos problemas principales del Aprendizaje Supervisado: clasificación y regresión, junto con los algoritmos principales para cada uno de ellos. Además, estudiaremos qué herramientas podemos usar para determinar la bondad de los modelos, y finalmente exploraremos en detalle uno de los algoritmos más importantes de esta categoría: las redes neuronales.

La segunda categoría de Aprendizaje Automático sería el **Aprendizaje No Supervisado**. La característica principal de estos algoritmos es que únicamente reciben unas variables de entrada X , y no reciben ningún tipo de información sobre las variables de salida. Puede parecer misterioso imaginar qué conocimiento puede obtener un sistema cuando no tiene ningún tipo de información sobre su entorno o sobre el objetivo que tiene que conseguir. Sin embargo, es posible establecer un marco formal para el Aprendizaje no Supervisado basándonos en que el objetivo del sistema es obtener relaciones y estructuras presentes en los datos [13]. Este tipo de algoritmos se suelen usar en las fases iniciales de tratamiento en datos, para identificar relaciones que sean imposibles de detectar a simple vista, por ejemplo, para identificar subgrupos o *clusters* dentro de los datos, sin necesidad de tener una clase etiquetada, o para reducir la dimensionalidad de los datos minimizando la pérdida de información.

Por último, estudiaremos el **Aprendizaje Por Refuerzo**. Este tipo de aprendizaje se basa en la idea de que aprendemos interactuando con el entorno. Cuando un niño juega en el parque y explora su alrededor, no hay nadie que le enseñe de manera explícita, sino que tiene una interacción con su entorno, que produce un flujo de información de causa-efecto sobre lo que ocurre a su alrededor, que le permite aprender sobre las consecuencias de sus acciones y sobre lo que debería hacer para conseguir sus objetivos. A lo largo de nuestra vida, estas interacciones son una de las fuentes más importantes de información sobre el mundo y sobre nosotros, desde aprender a conducir hasta cómo llevar una conversación, estamos alerta a cómo responde el entorno a lo que hacemos, y buscamos influenciar positivamente dicha respuesta con nuestro comportamiento.

El Aprendizaje por Refuerzo busca imitar dicho proceso de aprendizaje en sistemas informáticos, mediante el aprendizaje de un mapeo de situaciones a acciones, equivalente a aprender qué hacer según qué situaciones, para maximizar una señal numérica obtenida del entorno, también denominada recompensa. El modelo no sabe qué debe hacer, sino que tiene que descubrir qué acciones dan una mayor recompensa me-

diente prueba y error. En las situaciones más complejas e interesantes, las acciones no solo afectan la recompensa inmediata, sino que afectan a la siguiente situación, y por ende, a todas las recompensas siguientes. El aprendizaje por prueba y error y la recompensa retardada son las dos características principales del aprendizaje por refuerzo [34].

Algunas de las aplicaciones principales de este tipo de Aprendizaje Automático son la optimización de estrategias de inversión en bolsa, el desarrollo de sistemas de recomendación personalizados y el control de sistemas energéticos en edificios para maximizar la eficiencia energética y reducir el consumo.

A continuación vamos a estudiar con más detalle en qué consiste el Aprendizaje Supervisado.

2.2.3 *Aprendizaje Supervisado*

Tal y como hemos mencionado anteriormente, vamos a estudiar en qué consisten los dos problemas principales del Aprendizaje Supervisado, y veremos de manera superficial cómo funcionan los algoritmos más importantes para cada uno.

Es necesario destacar que, aunque el algoritmo varíe en función del tipo de problema y datos que tengamos, hay una metodología común a todos los problemas de Aprendizaje Supervisado, que nos permite evaluar y comparar los distintos modelos, y que estudiaremos en detalle antes de entrar en el estudio de algoritmos concretos.

En general podemos distinguir, según el tipo de variable de salida Y , dos problemas distintos: clasificación y regresión. En la clasificación, la variable de salida pertenece a un conjunto finito y discreto de categorías o clases, mientras que en la regresión, la variable de salida es continua y se busca predecir un valor numérico.

No entraremos en mucho detalle en los algoritmos concretos, pues, aunque es importante comprender que existen multitud de distintos algoritmos y ajustes, en este trabajo no vamos a experimentar con ellos, por lo que nos mantendremos en un estudio superficial.

2.2.3.1 *Evaluación de modelos. Sesgo y varianza.*

Comenzaremos presentando las metodologías más comunes para evaluar los modelos en el Aprendizaje Supervisado. Aunque en este tipo de problemas la intuición nos llevaría a utilizar todos los datos etiquetados para entrenar el modelo y obtener predicciones precisas para nuevas observaciones, esta aproximación dificulta la comparación entre modelos antes de tener nuevos datos para evaluar. En la práctica, es fundamental conocer qué modelo implementaremos antes de realizar predicciones sobre datos desconocidos.

Podemos suponer que cada modelo tiene una tasa de error sistemático en las nuevas observaciones, con una distribución específica. Por lo tanto, es crucial estimar esta ta-

sa de error antes de obtener los nuevos datos, pues es lo que nos permitirá seleccionar el modelo más adecuado de antemano.

Para estimar con precisión la tasa de error, es común dividir el conjunto inicial en dos subconjuntos: datos de entrenamiento y datos de evaluación. Mientras entrenamos el modelo con los primeros, lo evaluamos con los segundos. Es crucial que esta división sea apropiada: los datos de entrenamiento deben ser lo suficientemente grandes y representativos para que el modelo adquiera conocimientos, y el conjunto de evaluación debe ser lo bastante extenso para una estimación precisa de la tasa de error. Además, resulta lógico deducir que si la división cambia, el modelo resultante será distinto también.

La división en entrenamiento y evaluación es uno de los factores que afectan al problema principal en el Aprendizaje Supervisado: el compromiso entre sesgo y varianza. Para entender en qué consiste dicho compromiso es necesario introducir ambos conceptos por separado.

En primer lugar, la varianza se refiere a la variabilidad de las predicciones si hubiéramos entrenado el modelo sobre un conjunto de entrenamiento distinto. En una situación ideal, el modelo obtenido generaliza suficientemente bien los datos, y no cambiaría demasiado las predicciones al variar el conjunto de entrenamiento, pero en la práctica esto no siempre se cumple. Normalmente los modelos más flexibles son los que presentan una varianza más alta, pues están ajustándose demasiado a los datos de entrenamiento (*overfitting*).

En segundo lugar, el sesgo se refiere al error o diferencia entre las predicciones y el valor real, es la capacidad del modelo de predecir de manera precisa y ajustarse a los datos. En este caso, los modelos más flexibles tendrán un sesgo más bajo, pues pueden adaptarse mejor a los datos de entrenamiento.

Por lo tanto, el compromiso entre sesgo y varianza se refiere a la relación entre la capacidad de un modelo para ajustarse bien a los datos de entrenamiento y su capacidad para generalizar a nuevos datos. Dicho compromiso viene dado por la relación entre ambas propiedades, y es que un modelo con un sesgo alto tiene unas predicciones más estables, que no dependen tanto del conjunto de entrenamiento, lo que es equivalente a tener una varianza baja.

De la misma manera, un modelo con una varianza alta se ajustará demasiado a los datos de entrenamiento, y tendrá un sesgo bajo, pues las predicciones se acercarán más a los valores reales.

El objetivo es hallar un equilibrio entre ambos aspectos para lograr un modelo capaz de generalizar adecuadamente los datos de entrenamiento y obtener un buen rendimiento en datos nuevos [22].

Podemos proceder a estudiar los dos problemas fundamentales del Aprendizaje Supervisado, y sus algoritmos más comunes.

2.2.3.2 Regresión

Tal y como ya se ha mencionado, el problema de regresión se da cuando la variable de salida es continua. Puede parecer todavía muy ambiguo, pues no hemos descrito en detalle cómo se puede obtener el mapeo entre las variables de entrada y salida, pero la realidad es que depende del algoritmo.

El algoritmo más sencillo es el de **Regresión Lineal**, y pasa por ajustar unos parámetros β_0, β_1 de manera que las variables Y vengan dadas por:

$$Y = \beta_0 + \beta_1 X \quad (2.2.1)$$

Para ajustar dichos parámetros se utiliza el método de los mínimos cuadrados. No entraremos en detalle en qué consiste dicho método pues está fuera del foco de este trabajo.

Otro algoritmo de regresión son los **Árboles de Decisión**. Estos realizan una segmentación del espacio predictor (variables de entrada X) en varias regiones. Para hacer una predicción concreta, se suele utilizar la media o la moda de las observaciones de entrenamiento en la región a la que pertenece. El nombre de árbol de decisión viene porque el conjunto de reglas de división utilizado para segmentar el espacio predictor se puede resumir en un árbol.

Para segmentar el espacio en regiones se utilizan *cajas*, de manera que se minimice el error obtenido, dado por

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \quad (2.2.2)$$

donde \hat{y}_{R_j} es la media de los valores de salida Y para la *caja* j .

2.2.3.3 Clasificación

El segundo problema más común de Aprendizaje Supervisado es la clasificación, donde el objetivo es asignar una clase o etiqueta a los valores de entrada. Dicha etiqueta puede codificarse numéricamente, por ejemplo, si quisieramos clasificar pacientes en sanos (0) o enfermos (1), por lo que resulta lógico cuestionar por qué no se podrían utilizar algoritmos de regresión.

Hay dos motivos principales por los cuales esto no resulta adecuado. El primero, es que en el caso de tener únicamente dos clases, sería posible obtener predicciones más allá del intervalo [0,1], lo que es imposible en el contexto del problema. El segundo, es que no extendería bien a problemas con más de dos clases. Por eso, en estos problemas se suelen usar algoritmos específicos de clasificación.

El primer algoritmo es la **Regresión Logística**, donde en vez de predecir directamente la variable Y , se predice la *probabilidad de que la observación pertenezca a la clase Y*. Este

algoritmo pasa por modificar el modelado de la Regresión Lineal para que produzca salidas entre 0 y 1, utilizando la función logística:

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \quad (2.2.3)$$

Para ajustar el modelo, en este caso, se utiliza un método denominado el método de máxima verosimilitud. Este método de ajuste consiste en encontrar los valores de los parámetros β_0, β_1 que hacen que los datos observados sean más probables de acuerdo con el modelo propuesto.

En otras palabras, se busca maximizar la verosimilitud, que es la probabilidad de obtener los datos de partida dados los parámetros del modelo. Esto se logra ajustando los parámetros de manera que la función de verosimilitud sea máxima, y una vez obtenidos estos valores, se consideran las estimaciones de los parámetros del modelo que mejor se ajustan a los datos observados.

Para continuar con la extensión de los algoritmos de regresión ya estudiados, vamos a introducir de nuevo los **Árboles de Decisión**, en este caso para problemas de clasificación.

La mayor diferencia entre los árboles de regresión y clasificación es que en vez de utilizar la media de las observaciones de una determinada región para realizar predicciones, utilizaremos la clase más común dentro de esa región. Por lo demás, la manera de construir el árbol es muy similar, salvo que el método escogido para calcular el error debe ser acorde al tipo de predicciones que se están realizando, es decir, que no podríamos usar un mecanismo como el error cuadrático medio, sino que tendríamos que optar por métricas como la tasa de acierto, por ejemplo.

Tal y como ya hemos mencionado, estos algoritmos no son el objeto principal de estudio, pero es importante conocer la naturaleza de los problemas y la variedad de algoritmos posibles para cada uno. Podemos dar por finalizada la revisión de los antecedentes de este trabajo. En la siguiente parte, estudiaremos en detalle los fundamentos teóricos en los que se basan las arquitecturas utilizadas en la experimentación, comenzando desde el estudio de los grafos hasta la presentación de las Redes Neuronales Temporales sobre Grafos.

Parte II

MARCO TEÓRICO

Fundamentos conceptuales y teóricos de las Redes Neuronales Temporales de Grafos (TGNs), exposición de sus componentes y su funcionamiento detallado.

3

REDES NEURONALES DE GRAFOS

Comenzaremos el estudio del marco teórico introduciendo los fundamentos matemáticos esenciales para comprender y trabajar con grafos, un componente central en el estudio avanzado de las redes neuronales sobre grafos (GNNs). Los grafos proporcionan una estructura rica y versátil para representar relaciones y dependencias complejas, lo cual es fundamental en el contexto de este trabajo, pues las redes eléctricas se pueden modelar como tales.

Es necesario mencionar que la terminología y fundamentos sobre las Redes Neuronales se ha presentado como parte del Apéndice del trabajo (Capítulo A), con el objetivo de reducir la extensión del mismo.

Comenzaremos con una introducción básica a los grafos, donde estableceremos la formulación matemática que permite describir estos sistemas de manera precisa. Posteriormente, avanzaremos hacia cómo estas estructuras se aplican dentro del campo del Aprendizaje Automático en grafos, destacando las particularidades y desafíos que esto conlleva. Esto nos preparará para una discusión más detallada sobre las Redes Neuronales de Grafos (GNNs), incluyendo las motivaciones para su uso, las aproximaciones iniciales y su arquitectura fundamental, que comprende componentes como codificadores y decodificadores.

Además, se detallará el mecanismo interno y funcionamiento de las GNNs, por medio de las funciones de agregación y actualización, proporcionando una base sólida para entender cómo estas redes pueden aprender y hacer inferencias efectivas a partir de datos estructurados en forma de grafos.

3.1 INTRODUCCIÓN A LOS GRAFOS

Los Grafos son estructuras de datos que presentan una variedad muy grande de aplicaciones, como por ejemplo en la web, redes sociales, redes de comunicación, o redes de interacción en química. De manera general, un grafo es una colección de objetos o nodos, con un conjunto de interacciones o aristas entre pares de nodos. Si tomamos como ejemplo una red social, usaríamos nodos para representar individuos y las aristas para representar que dos individuos son amigos [17].

El poder de esta estructura de datos reside, por una parte, en que se centran en las relaciones entre puntos, en vez de en las propiedades individuales, como en las estructuras de datos tradicionales, y por otra parte, en su generalidad. Se puede utilizar el mismo formalismo para representar redes sociales o interacciones entre proteínas, pues comparten ciertas propiedades que lo permiten.

A continuación, introduciremos la formulación matemática de los grafos, una herramienta fundamental que nos permitirá comprender su estructura, propiedades y comportamiento. Después, estudiaremos los problemas principales en el contexto del aprendizaje automático con datos estructurados en forma de grafos, lo que nos permitirá comprender mejor cómo las relaciones entre nodos pueden ser aprovechadas para resolver problemas complejos en distintos dominios.

3.1.1 Formulación matemática

Vamos a definir formalmente en qué consiste un grafo. Diremos que un grafo $G = (V, E)$ está definido por un conjunto de nodos V y un conjunto de ejes o aristas E entre dichos nodos. Denotaremos cada arista que parte del nodo $u \in V$ al nodo $v \in V$ como $(u, v) \in E$. Además, diremos que el grado de un nodo es el número de conexiones que tiene.

Solemos representar los grafos mediante una matriz de adyacencia $A^i \in \mathbb{R}^{|V|, |V|}$. Para ello, se numeran los nodos para que cada uno represente una fila y una columna determinadas en la matriz, y después se incluye la información de los enlaces como las entradas de la matriz.

El tipo de grafo más simple es en el que como mucho existe un enlace entre cada par de nodos, no hay auto-enlaces, y los enlaces no tienen dirección. Los grafos con esta última propiedad se llamarán *grafos no dirigidos*.

Formalmente:

$$A[u, v] = \begin{cases} 1 & \text{si } (u, v) \in E \\ 0 & \text{en otro caso} \end{cases} \quad (3.1.1)$$

Si el grafo es no dirigido, entonces la matriz A será una matriz simétrica.

Además, hasta ahora únicamente hemos considerado la presencia de una relación entre nodos, y no el peso de dicha relación. Diremos que un grafo es *ponderado* si los valores de la matriz de adyacencia son valores reales arbitrarios.

En muchos casos también tenemos información de atributos o características asociadas a un grafo (por ejemplo, una imagen de perfil asociada a un usuario en una red social). Normalmente estos son atributos a nivel de nodo, que se representan utilizando una matriz de valores reales $X \in \mathbb{R}^{|V|, m}$, donde m es el número de características.

Será necesario, por lo tanto, que el orden de los nodos sea consistente con el orden en la matriz de adyacencia.

En casos raros, también se pueden considerar grafos que tienen características de arista, que pueden ser tanto valores reales como valores discretos y en algunos casos incluso asociamos características o atributos con grafos en su totalidad.

3.1.2 Aprendizaje Automático en Grafos

En secciones anteriores hemos introducido la naturaleza de los problemas principales de Aprendizaje Automático. En el caso del Aprendizaje Automático sobre grafos, los fundamentos se mantienen, pero surgen problemas específicos que no encajan completamente en una de las tres categorías de Aprendizaje Automático. A continuación veremos cuáles son las tareas principales que se pueden llevar a cabo sobre grafos.

3.1.2.1 Clasificación de nodos

En los problemas de clasificación de nodos, el objetivo es asignar una etiqueta a cada uno de los nodos presentes en el grafo. Por ejemplo, en el caso de la red social que veíamos anteriormente, una tarea de clasificación de nodos interesante podría ser la identificación de cuentas automatizadas o *bots*.

Como en los problemas usuales de clasificación, el objetivo es asignar una clase y_u , a cada uno de los nodos $u \in V$, partiendo de un conjunto de nodos previamente etiquetados, que conformarán nuestro conjunto de entrenamiento $V_{\text{train}} \subset V$.

A simple vista, la clasificación de nodos puede parecer un problema idéntico a la clasificación supervisada usual, presentada en secciones anteriores. Sin embargo, existen diferencias importantes:

La diferencia más importante es la **distribución de nodos**. Cuando construimos modelos de Aprendizaje Supervisado, asumimos que cada punto de datos es estadísticamente independiente de todos los demás, pues, de lo contrario, podríamos necesitar modelar las dependencias entre ellos. Además, también asumimos que los puntos están distribuidos de manera idéntica, pues de no ser así, no tenemos forma de garantizar que nuestro modelo generalice bien a nuevas observaciones. No obstante, cuando hablamos de los nodos presentes en grafos, se rompen completamente las dos suposiciones, y en lugar de modelar un conjunto de datos independientes e idénticamente distribuidos, estamos modelando un conjunto interconectado de nodos.

De hecho, la característica común a alguna de las aproximaciones más exitosas en clasificación de nodos, es precisamente que se explotan las conexiones entre nodos. Por ejemplo, asumiendo que los nodos van a tender a compartir atributos con sus vecinos en el grafo (*homofilia*), o que los nodos con estructuras de vecindario parecidas tendrán etiquetas similares (*equivalencia estructural*), entre otras.

Por lo tanto, en problemas de clasificación de nodos, es fundamental explotar la información sobre las relaciones presente en el grafo, y no considerar las observaciones como datos independientes.

3.1.2.2 *Predicción de enlaces*

El siguiente problema que vamos a estudiar es la predicción de enlaces. Este problema se da cuando no tenemos la información completa sobre las relaciones entre nodos, sino que buscamos predecir las interacciones que faltan. La estructura formal de este tipo de problemas es que, dado un conjunto de nodos V y un conjunto de enlaces $E_{\text{train}} \subset E$, el objetivo es inferir el resto de enlaces.

Por ejemplo, considerando el escenario previo de la red social, podríamos buscar determinar qué dos usuarios deberían estar enlazados según la estructura del grafo, con la finalidad de ofrecerles una recomendación de nuevos amigos.

Es necesario mencionar que la complejidad de esta tarea depende del tipo de grafo con el que se está trabajando. Por una parte, podríamos tener un grafo simple, no dirigido, que modele las relaciones de amistad de una red social, y entonces se podrían usar heurísticas muy sencillas basadas en cuántos vecinos comparten dos nodos.

Por otra parte, existen grafos más complejos, como los de tipo multirelacional, un tipo de grafo en el que los nodos y las aristas pueden pertenecer a diferentes tipos o clases. Esto significa que las aristas pueden representar diferentes tipos de relaciones entre los nodos, y cada tipo de relación puede tener sus propias características y propiedades. En este tipo de grafos, la predicción de enlaces requiere algoritmos más complejos.

3.1.2.3 *Clasificación o regresión sobre el grafo*

En ocasiones, el foco no reside en los nodos o las aristas del grafo, sino en propiedades del grafo considerado en su totalidad. Por ejemplo, si consideramos el grafo asociado a la estructura de una molécula, podríamos construir un modelo de regresión que prediga su grado de toxicidad.

En este tipo de problemas, buscamos aprender a través de grafos, pero en vez de aprender sobre los elementos de un solo grafo, partimos de un conjunto de grafos, y el objetivo es hacer predicciones sobre cada uno de ellos.

De todas las tareas de Aprendizaje Supervisado sobre grafos, la regresión y clasificación son las más similares al Aprendizaje Supervisado estándar, pues en este caso, se asume que los grafos que conforman el conjunto de entrenamiento son independientes e idénticamente distribuidos.

3.1.2.4 Detección de comunidades

Hasta ahora únicamente hemos tratado con problemas relacionados con el Aprendizaje Supervisado, donde teníamos un conjunto de entrenamiento, en el que la información sobre nodos y enlaces estaba completa. Sin embargo, en ocasiones es necesario descubrir patrones o relaciones en el grafo, que puedan aportar conocimiento, sin llegar a predecir ninguna información nueva. Resulta bastante sencillo comparar este tipo de tareas con las del Aprendizaje No Supervisado estándar.

La detección de comunidades en grafos es una tarea que consiste en identificar grupos o conjuntos de nodos que están más densamente interconectados entre sí que con el resto del grafo. Estas comunidades pueden representar subgrupos de nodos que comparten características similares o que están involucrados en procesos o funciones similares dentro de la red. Por ejemplo, en una red social, las comunidades pueden representar grupos de usuarios con intereses comunes, áreas geográficas similares o interacciones frecuentes. La detección de comunidades es importante no solo para entender la estructura y la dinámica de la red, sino también para aplicaciones prácticas como la segmentación de clientes, la recomendación de contenido y la detección de anomalías.

3.2 DESCRIPCIÓN DE LAS GNNS

Después de estudiar los fundamentos matemáticos de los grafos, estamos en disposición de profundizar los conocimientos presentados en capítulos anteriores sobre las Redes Neuronales, y presentar una arquitectura de red específica para los grafos. Comenzaremos esta sección explicando por qué el uso de redes básicas no es adecuado en estructuras de grafos, y más adelante se estudiará en detalle la arquitectura y funcionamiento de las GNNs (*Graph Neural Networks*).

3.2.1 Motivación. Aproximaciones iniciales

En la mayoría de los problemas de aprendizaje automático con grafos se pueden utilizar las técnicas estándar que estudiamos en el Capítulo 2. Sin embargo, surge un desafío fundamental: es necesario obtener una representación adecuada del grafo para alimentar estos algoritmos.

Aunque la matriz de adyacencia es una representación intuitiva, su tamaño es cuadrático en el número de nodos, lo que puede ser impracticable para grafos grandes. Esto es especialmente relevante considerando que el número de conexiones suele ser mucho menor que el número de entradas de la matriz, lo que resulta en un uso inefficiente del espacio. Además, la matriz de adyacencia no logra capturar de manera completa la información topológica y estructural del grafo.

Por otra parte, la matriz de adyacencia presenta una limitación importante, y es la dependencia en el orden de los nodos, es decir, que no es invariante a permutaciones. Esta propiedad es crucial al trabajar con grafos, dado que, aunque el orden de los nodos en la matriz varíe, el grafo subyacente es idéntico, y la estructura que lo codifica debe ser también invariante.

Es aquí donde entran en juego las incrustaciones de nodos (*node embedding*, en inglés). El objetivo de estos métodos es codificar los nodos como vectores con una dimensión baja que resuman su posición en el grafo y la estructura de su vecindario local. En otras palabras, queremos proyectar los nodos en un espacio latente, donde las relaciones geométricas correspondan a relaciones (por ejemplo, aristas) en el grafo original o red. Estas representaciones se utilizan luego como entrada en algoritmos de aprendizaje automático estándar, lo que permite aplicar técnicas de ML a datos estructurados en forma de grafos de manera eficiente y efectiva.

A continuación vamos a introducir cuál es la metodología usual para obtener dicha representación.

3.2.2 Codificador y Decodificador

En esta aproximación se solventa el problema de encontrar una representación para el grafo mediante dos operaciones principales: codificación y decodificación.

La codificación (*encoder*, en inglés), mapea cada nodo $v \in V$ a un vector $z_v \in \mathbb{R}^d$. Formalmente:

$$\text{ENC} : V \rightarrow \mathbb{R}^d \quad (3.2.1)$$

En la mayoría de casos, se utilizan lo que denominaremos incrustaciones poco profundas, en las que dicha codificación es simplemente una búsqueda en el grafo original, así:

$$\text{ENC}(v) = Z[v] \quad (3.2.2)$$

donde $Z \in \mathbb{R}^{|V|,d}$ es una matriz que contiene las representaciones vectoriales y $Z[v]$ es la fila correspondiente al nodo v .

A continuación veremos de manera superficial cómo se puede obtener la matriz Z , pero antes es necesario destacar que las aproximaciones poco profundas se pueden generalizar, de manera que se utilicen las características del nodo o información estructural para generar la representación. Estas arquitecturas de codificación son precisamente las Redes Neuronales de Grafos, (*Graph Neural Network*, en inglés), el objeto principal de este trabajo.

El siguiente paso es la decodificación, que usará la representación generada para reconstruir la información sobre el vecindario de cada nodo en el grafo original. Aun-

que existen muchas alternativas para el decodificador, lo más común es utilizar la decodificación por pares, que se puede definir como sigue:

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+ \quad (3.2.3)$$

En la decodificación por pares se aplica la decodificación por pares de vectores (z_u, z_v) , y se busca reconstruir la relación entre los nodos u y v .

Normalmente, la decodificación se utiliza para obtener información particular sobre la relación entre dos nodos, como por ejemplo determinar si son vecinos o no, en cuyo caso la medida de similitud S será equivalente a la matriz de adyacencia A .

El objetivo general es optimizar tanto la codificación como la decodificación para minimizar una función de pérdida determinada al reconstruir el grafo, iterando a través del conjunto de entrenamiento. Formalmente, se busca:

$$\text{DEC}(\text{ENC}(u), \text{ENC}(v)) = \text{DEC}(z_u, z_v) \approx S[u, v] \quad (3.2.4)$$

donde $S[u, v]$ es una medida de similitud entre nodos en el grafo original.

Vamos a introducir brevemente algunas de las metodologías de codificación poco profunda más comunes.

3.2.2.1 Factorización de matrices

Esta aproximación utiliza la perspectiva de la factorización de matrices, buscando relacionar la decodificación del vecindario de un nodo con la reconstrucción de las entradas en la matriz de adyacencia. Bajo esta perspectiva, se busca una representación, que tendrá forma de matriz S , y que generalizará la matriz de adyacencia incluyendo información sobre la similitud entre nodos.

Existen distintas alternativas basadas en factorización de matrices. La primera de ellas es la que utiliza los **valores propios del Laplaciano**, en la que se establece la decodificación mediante la distancia L2 entre vectores:

$$\text{DEC}(z_u, z_v) = \|z_u - z_v\|_2^2. \quad (3.2.5)$$

Y la función de pérdida pondera los pares de nodos en función de su similitud en el grafo, de manera que se penaliza cuando dos nodos muy similares tienen representaciones muy lejanas. La idea es que se construya la matriz S de manera que verifique las propiedades de una matriz Laplaciana, para que se pueda afirmar que, si las representaciones z_u tienen dimensión d , entonces la solución óptima de la función de pérdida, viene dada por los d vectores propios del Laplaciano.

La segunda alternativa asume que la similitud entre dos nodos es proporcional al producto interno de sus representaciones.

$$\text{DEC}(z_u, z_v) = z_u^T z_v \quad (3.2.6)$$

La función de pérdida no es más que el error cuadrático medio entre la decodificación y la matriz S , que puede construirse de distintas formas, desde utilizar la matriz de adyacencia hasta alternativas más generales.

Una característica común a las dos alternativas introducidas es que sus funciones de pérdida pueden minimizarse utilizando algoritmos de factorización de matrices, como la descomposición en valores singulares (*SVD*, por sus siglas en inglés).

De hecho, al apilar las incrustaciones de nodos $z_u \in \mathbb{R}^d$ en una matriz $Z \in \mathbb{R}^{|V|,d}$, el objetivo de reconstrucción para estos enfoques puede escribirse como $L \approx \|ZZ^T - S\|_F^2$, lo que corresponde a una factorización de la matriz de similitud entre nodos S .

Intuitivamente, el objetivo de estos métodos es aprender incrustaciones para cada nodo de manera que el producto interno entre los vectores de incrustación aprendidos aproxime alguna medida determinista de similitud entre nodos.

3.2.2.2 Paseo aleatorio

Tal y como veníamos explicando, los métodos anteriores utilizan medidas deterministas para la similitud entre nodos. La siguiente alternativa adapta la aproximación del producto interno para utilizar medidas estocásticas de similitud entre nodos.

El fundamento de estas alternativas es codificar las representaciones de los nodos de manera que dos nodos tengan representaciones similares o cercanas si tienden a aparecer en los mismos paseos aleatorios sobre el grafo.

Para modelar un paseo aleatorio, se utiliza una distribución de probabilidad $p_{G,T}(v|u)$ que determina la probabilidad de visitar v en un paseo aleatorio de T pasos, que comience en u .

No vamos a entrar en detalle en la formulación matemática de estos métodos, pues no es el objetivo principal del estudio.

3.2.2.3 Limitaciones de las representaciones poco profundas

Hasta ahora hemos introducido la metodología para obtener representaciones poco profundas, en las que el codificador es un modelo que obtiene una única representación para cada nodo en el grafo, en forma de matriz Z . No obstante, esta metodología tiene varias limitaciones importantes:

1. La primera, es que el codificador no comparte ningún parámetro a la hora de codificar distintos nodos, es decir, se optimiza la representación de cada nodo de manera independiente. Esto es ineficiente desde el punto de vista estadístico, pues al compartir parámetros estamos introduciendo regularización, y desde el

punto de vista computacional, pues aumenta considerablemente el número de parámetros.

2. La segunda es que no se utilizan los atributos de los nodos. Para grafos con mucha información, esto resulta en codificaciones menos precisas de las que obtendríamos si se utilizara toda la información disponible.
3. Por último, los métodos de representaciones poco profundas son, por naturaleza, transductivos [18]. Esto significa que únicamente pueden generar representaciones para nodos que estuvieran presentes en el entrenamiento, por lo que no se pueden utilizar en problemas de inducción.

Para solucionar estas limitaciones, podemos utilizar codificadores que utilizan información estructural y atributos del grafo. Las más populares son las redes neuronales de grafos, o GNN (*Graph Neural Network*, en inglés), que estudiaremos a continuación.

3.2.3 Introducción a las GNN

Tras presentar los fundamentos y la motivación de las GNNs, podemos proceder a estudiar la característica principal de este tipo de redes: la utilización de un mecanismo de paso de mensajes entre nodos para obtener una representación más adecuada.

3.2.3.1 Paso de mensajes

Supongamos que tenemos un grafo $G = (V, E)$, con un conjunto de características de nodo $X \in \mathbb{R}^{d, |V|}$. El objetivo es encontrar una representación de nodo $z_u, \forall u \in V$.

En cada iteración del mecanismo de paso de mensajes, se actualiza la representación oculta $h_u^{(k)}$ correspondiente a cada nodo $u \in V$. Esta representación incluye información agregada sobre el vecindario de u , que denotaremos $\mathcal{N}(u)$.

Por lo tanto, se podría resumir cada iteración en dos pasos: un paso de agregación seguido de un paso de actualización. Matemáticamente, se puede expresar como sigue:

$$\begin{aligned} \mathbf{h}_u^{(k+1)} &= \text{UPDATE}^{(k)} \left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)} \left(\left\{ \mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u) \right\} \right) \right) \\ &= \text{UPDATE}^{(k)} \left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)} \right) \end{aligned} \quad (3.2.7)$$

donde las funciones UPDATE y AGGREGATE son funciones diferenciables arbitrarias, y $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ es el mensaje que se agrega desde el vecindario de u .

Por lo tanto, en cada iteración k de la red, la función AGGREGATE toma las representaciones correspondientes al vecindario de u , $\mathcal{N}(u)$, y genera el mensaje $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$.

Después, la función UPDATE combina dicho mensaje con la representación anterior del nodo, $\mathbf{h}_u^{(k)}$, para obtener la nueva representación.

Para la iteración inicial, es decir, $k = 0$, se consideran los vectores de características, y por lo tanto, $\mathbf{h}_u^{(0)} = \mathbf{x}_u, \forall u \in V$. En el caso en el que el grafo no disponga de dichas características, se podrían obtener mediante otras alternativas, como por ejemplo usar información estadística del nodo, o utilizar como característica un vector de identidad, con ceros y unos, que identifican de manera única cada nodo.

Es importante resaltar que a medida que progresá el proceso iterativo, la representación de cada nodo incorpora información de nodos más distantes. Por lo tanto, en la primera iteración, cada nodo contiene información acerca de aquellos que están a una distancia de un enlace, mientras que en la segunda iteración, abarca nodos situados a dos enlaces de distancia, y así sucesivamente.

Por último, mencionar que el paso de mensajes en el marco de GNN es análogo a un perceptrón multicapa estándar (MLP), ya que se basa en operaciones lineales seguidas de una única no linealidad por elemento. Primero sumamos los mensajes que provienen de los vecinos, luego, combinamos la información del vecindario, y finalmente, aplicamos una función no lineal a cada elemento elemento.

A continuación vamos a estudiar en detalle en qué consiste el mecanismo del paso de mensajes, concretando la naturaleza de las funciones UPDATE y AGGREGATE.

3.3 FORMALIZACIÓN DE GNNS

3.3.1 Descripción detallada

Comenzamos concretando aún más el proceso de obtención de la representación oculta de un nodo u en la iteración k , $\mathbf{h}_u^{(k)}$. Para ello, vamos a ampliar la expresión estudiada en la ecuación 3.2.7:

$$\mathbf{h}_u^{(k)} = \sigma \left(\mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right) \quad (3.3.1)$$

donde $\mathbf{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neigh}}^{(k)} \in \mathbb{R}^{d^k, d^{k-1}}$ son matrices con parámetros y la función σ representa la no-linealidad que vimos en capítulos anteriores.

Tenemos, por lo tanto, que las funciones de agregación y actualización vienen definidas por:

$$\text{UPDATE}^{(k)} \left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)} \right) = \sigma \left(\mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u + \mathbf{W}_{\text{neigh}}^{(k)} \mathbf{m}_{\mathcal{N}(u)}^{(k)} \right) \quad (3.3.2)$$

$$\text{AGGREGATE}^{(k)} \left(\left\{ \mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u) \right\} \right) = \mathbf{m}_{\mathcal{N}(u)}^{(k)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \quad (3.3.3)$$

Hasta ahora únicamente hemos visto expresiones a nivel de nodo. Sin embargo, resulta interesante mencionar que las mismas expresiones se pueden formular a nivel de grafo. Así, si consideramos $H^{(k)} \in \mathbb{R}^{|V|,d}$ como la matriz de representaciones de nodo en la iteración k , A como la matriz de adyacencia, obtenemos la siguiente formulación:

$$\mathbf{H}^{(t)} = \sigma \left(\mathbf{A} \mathbf{H}^{(k-1)} \mathbf{W}_{\text{neigh}}^{(k)} + \mathbf{H}^{(k-1)} \mathbf{W}_{\text{self}}^{(k)} \right) \quad (3.3.4)$$

Cabe destacar que se ha omitido el término del sesgo por simplicidad de notación, pero en la práctica es un término fundamental para la precisión de la codificación de nodos.

Además, la aproximación más usual para este mecanismo es en la que se añaden auto-enlaces en el grafo original, haciendo que el paso UPDATE no tenga que estar presente de manera explícita:

$$\mathbf{h}_u^{(k)} = \text{AGGREGATE} \left(\left\{ \mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u) \cup \{u\} \right\} \right) \quad (3.3.5)$$

Aunque el marco teórico presentado anteriormente demuestra un buen rendimiento, existen numerosas formas de mejorarlo y generalizarlo. En las siguientes secciones, se explorarán formas de ampliar la aproximación básica de las funciones UPDATE y AGGREGATE.

3.3.2 Función de agregación

Ya hemos visto, en la ecuación 3.3.3, que la función AGGREGATE es equivalente a calcular la suma de las representaciones de los vecinos en cada iteración. Su limitación principal radica en las diferencias en los grados de los nodos, pues, si tenemos dos nodos $u, v \in V$ donde el grado de u es mucho mayor que el de v , entonces tendremos que para casi cualquier norma se verifica que:

$$\left\| \sum_{w \in \mathcal{N}(u)} \mathbf{h}_w \right\| \gg \left\| \sum_{w' \in \mathcal{N}(v)} \mathbf{h}_{w'} \right\| \quad (3.3.6)$$

Esta gran diferencia en la magnitud puede dar lugar a inestabilidad y a dificultad para optimizar los parámetros. Para solventarlo, existen numerosas aproximaciones, que veremos a continuación.

3.3.2.1 Normalización del vecindario

La primera de ellas es la **normalización del vecindario**, mediante la cual, se normaliza la suma anterior, por ejemplo, calculando la media, o con otros factores de normalización, como la normalización simétrica [25].

$$\mathbf{m}_{\mathcal{N}(u)} = \frac{\sum_{v \in \mathcal{N}(u)} \mathbf{h}_v}{|\mathcal{N}(u)|} \quad \text{Media} \quad (3.3.7)$$

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}} \quad \text{Normalización Simétrica} \quad (3.3.8)$$

Esta es la alternativa utilizada, por ejemplo, en las Redes Neuronales Convolucionales sobre Grafos (GCN). La agregación en una GCN estándar, se realiza mediante una media ponderada de las características de los vecinos del nodo (normalización simétrica), y además, se incluyen auto-enlaces para omitir la función de actualización. Formalmente sería como sigue:

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{1}{c_{uv}} W^{(k)} h_v^{(k)}$$

donde c_{ij} es el factor de normalización, en este caso, el correspondiente a la normalización simétrica, y $W^{(k)}$ es una matriz de pesos de la capa k .

Este mecanismo de agregación asegura que la información de los vecinos se combine de manera equilibrada, manteniendo la estabilidad de las representaciones de los nodos a través de las capas de la red.

Sin embargo, aunque la normalización soluciona el problema de los grados y es fundamental para mejorar el rendimiento, también hay que tener en cuenta que supone una pérdida de información y que su uso dependerá del contexto del problema. Esto es debido a que tras normalizar, resulta más complicado usar las representaciones para distinguir nodos o incluso para obtener información estructural del grafo.

Normalmente, se utiliza la normalización cuando las características de los nodos son más importantes que la información estructural, o cuando hay una gran variabilidad en los grados de los nodos, que puede dar lugar a inestabilidad en la optimización.

3.3.2.2 Agregación como función sobre conjuntos

La segunda alternativa va un paso más allá, y pasa por considerar la función AGGREGATE como **una función sobre un conjunto no ordenado**: desde un conjunto de representaciones $\{\mathbf{h}_v, \forall v \in \mathcal{N}(u)\}$ obtenemos un único vector $\mathbf{m}_{\mathcal{N}(u)}$. El principal motivo detrás

de esta concepción es la posibilidad de usar teoría de redes neuronales invariantes a permutaciones.

Por ejemplo, en [45] se establece que cualquier función invariante a permutaciones que mapee un conjunto de representaciones a un único vector, puede aproximarse con una precisión arbitraria mediante la siguiente función:

$$\mathbf{m}_{\mathcal{N}(u)} = \text{MLP}_\theta \left(\sum_{v \in N(u)} \text{MLP}_\phi(\mathbf{h}_v) \right) \quad (3.3.9)$$

Donde MLP_θ representa el uso de un perceptrón multicapa con parámetros θ . Además, podría sustituirse la suma por otra función de reducción, como el máximo o el mínimo.

Estos enfoques a menudo suponen pequeños incrementos en el rendimiento, aunque también introducen un mayor riesgo de sobreajuste, dependiendo de la profundidad de las redes MLP utilizadas. Es común emplear MLP que tienen solo una capa oculta, ya que estos modelos son suficientes para satisfacer la teoría, pero no suponen un riesgo de sobreajuste.

Existen variaciones de la aproximación presentada en la expresión 3.3.9, como por ejemplo, *Janossy pooling*, que rompe con la suposición de que la función sea invariante a permutaciones. En su lugar, se utiliza una función sensible a permutaciones, y se calcula la media entre muchas permutaciones distintas. Formalmente:

$$\mathbf{m}_{\mathcal{N}(u)} = \text{MLP}_\theta \left(\frac{1}{|\Pi|} \sum_{\pi \in \Pi} \rho_\phi \left(\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, \dots, \mathbf{h}_{v_{|\mathcal{N}(u)|}} \right)_{\pi_i} \right) \quad (3.3.10)$$

donde Π es un conjunto de permutaciones, ρ_ϕ es una función sensible a permutaciones, que suele ser una red neuronal LSTM, pues es la arquitectura ideal para operar sobre secuencias.

La limitación principal de esta aproximación es que normalmente, resulta imposible que Π contenga todas las permutaciones posibles. En su lugar, se puede obtener una muestra aleatoria de permutaciones, o bien utilizar un orden fijo de nodos, como por ejemplo, según el grado, e ir rompiendo enlaces de manera aleatoria.

3.3.2.3 Mecanismos de atención

La última variación sobre la función básica de agregación se basa en la concepción de *atención*. La idea básica es asignar un peso de atención o importancia a cada vecino, que marcará la influencia del mismo a la hora de realizar la agregación.

El primer modelo basado en esta aproximación se denomina Red de Atención en Grafos (*Graph Attention Network*, GAT, en inglés), que utiliza los pesos de atención de

los elementos del vecindario de un nodo determinado. Es decir, para cada, $u \in \mathcal{N}(u)$, se tiene $\alpha_{u,v}$, y se usará para definir la función de agregación:

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v \quad (3.3.11)$$

Sin embargo, resulta natural preguntarse cómo se definen los pesos $\alpha_{u,v}$, y la realidad es que hay múltiples aproximaciones.

En primer lugar, se tienen que calcular lo que se llaman coeficientes de atención, que denotaremos como $c_{u,v}$, calculados como sigue:

$$c_{u,v} = (\mathbf{A}^t [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]) \quad (3.3.12)$$

donde A es una matriz de atención entrenable, W una matriz entrenable que se utiliza para obtener las representaciones intermedias, y el símbolo \oplus representa una concatenación.

A continuación, se introduce la no linearidad mediante el uso de una función de activación, que normalmente suele ser *Leaky Rectified Linear Unit* (ReLU), seguida de Softmax para que los pesos finales estén normalizados.

La aproximación original, presentada en [39], define los pesos como sigue:

$$\alpha_{u,v} = \frac{\exp(A^t [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v])}{\sum_{v' \in \mathcal{N}(u)} \exp(A^t [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}])} \quad (3.3.13)$$

En esta expresión se está haciendo uso únicamente de la función Softmax, pero como veníamos diciendo, lo más usual es incluir también una función ReLU, que transformaría la expresión anterior en

$$\alpha_{u,v} = \frac{\exp(A^t \text{LeakyReLU}(c_{u,v}))}{\sum_{v' \in \mathcal{N}(u)} \exp(A^t \text{LeakyReLU}(c_{u,v'}))} \quad (3.3.14)$$

Además, hay otras variantes en las que se calculan K pesos, $\alpha_{u,v,k}$, correspondientes a K capas de atención independientes, mediante una de las aproximaciones disponibles. Como resultado, se generan K mensajes agregados entre nodos, lo que dará lugar a K representaciones distintas. Esto hace que se requiera una transformación y agregación adicional. Por lo general, se utiliza el operador de concatenación:

$$\begin{aligned} \mathbf{a}_k &= \mathbf{W}_k \sum_{v \in \mathcal{N}(u)} \alpha_{u,v,k} \mathbf{h}_v \\ \mathbf{m}_{\mathcal{N}(u)} &= [\mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \dots \oplus \mathbf{a}_K] \end{aligned} \quad (3.3.15)$$

En el caso en el que estemos en la última capa, se podría utilizar otra aproximación, como el cálculo de la media.

El uso de mecanismos de atención es una estrategia útil para aumentar la capacidad representacional de un modelo GNN, especialmente en casos donde se conoce de antemano que hay nodos más informativos que otros.

Vamos a continuar con las aproximaciones generales para la función de actualización.

3.3.3 Función de actualización

Hasta este punto, únicamente hemos visto el enfoque básico de GNN, donde la operación de actualización implica una combinación lineal de la representación actual del nodo con el mensaje recibido por parte de sus vecinos. También se ha estudiado brevemente el enfoque de auto-enlaces, que simplemente implica agregar un autobucle al grafo antes de realizar la agregación del vecindario. A continuación veremos generalizaciones más diversas del operador UPDATE.

Antes de comenzar a estudiar en detalle dichas generalizaciones, vamos a introducir por qué es relevante generalizar la función básica estudiada anteriormente.

Un problema muy común en GNNs es lo que se conoce como suavizado excesivo (*over-smoothing*, en inglés), y su idea principal es que después de varias iteraciones del mecanismo de paso de mensajes, las representaciones de los nodos se vuelven muy similares entre sí. Además, el problema se vuelve más crítico cuando se utilizan auto-enlaces.

Vamos a proceder a introducir alguna de las soluciones al problema.

3.3.3.1 Concatenación y conexiones de salto

Resulta lógico deducir que el problema se agrava en los casos en los que la representación actualizada $\mathbf{h}_u^{(k+1)}$ depende demasiado de los mensajes recibidos del vecindario, pues llegará un momento en el que la información recibida del resto de nodos domine la representación del propio nodo.

Una solución natural al problema pasa por usar lo que llamaremos conexiones de salto y concatenación de vectores, cuyo objetivo es preservar la información del nodo de iteraciones anteriores. Estos métodos no son incompatibles con la función de actualización definida en secciones anteriores, que denotaremos como $\text{UPDATE}_{\text{base}}$, sino que se define como un paso más.

La variante más simple pasa por hacer una concatenación, que preservará la información del nodo, separándola de la información que viene del vecindario:

$$\text{UPDATE}_{\text{concat}} \left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)} \right) = \left[\text{UPDATE}_{\text{base}} \left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)} \right) \oplus \mathbf{h}_u \right] \quad (3.3.16)$$

Otra variante disponible es obtener una representación mediante interpolación lineal de la representación anterior y la salida de la función $\text{UPDATE}_{\text{base}}$:

$$\text{UPDATE}_{\text{interpolate}} \left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)} \right) = \alpha_1 \circ \text{UPDATE}_{\text{base}} \left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)} \right) + \alpha_2 \odot \mathbf{h}_u, \quad (3.3.17)$$

donde $\alpha_1, \alpha_2 \in [0, 1]^d$, son parámetros entrenables, que cumplen $\alpha_2 = 1 - \alpha_1$. Estos parámetros se pueden aprender de distintas formas, como por ejemplo, usando un perceptrón multicapa para generarlos, o aprender α_1 en cada capa de paso de mensajes.

En la práctica, estas técnicas tienden a ser más útiles para tareas de clasificación de nodos con GNNs relativamente profundas (de 2 a 5 capas), y destacan en grafos que presentan homofilia, es decir, donde la predicción para cada nodo está fuertemente relacionada con las características de su vecindario.

3.3.3.2 Actualización cerrada

La siguiente aproximación se basa en ver el mecanismo de paso de mensajes de una manera un poco diferente: se considera la función de agregación como una función que recibe observaciones de los vecinos y luego actualiza el estado oculto del nodo. Esta concepción es similar a la que se utiliza en las Redes Neuronales Recurrentes para actualizar los estados ocultos, y en ella se define la función de actualización como sigue:

$$\mathbf{h}_u^{(k)} = \text{GRU} \left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)} \right), \quad (3.3.18)$$

donde GRU denota la operación de actualización de una unidad recurrente. La ecuación 3.3.18 es simplemente un ejemplo, pero se podría usar cualquier función de actualización definida para Redes Neuronales Recurrentes.

Estas técnicas se utilizan para GNNs profundas (más de 10 capas), y son especialmente útiles en casos en los que la predicción conlleva un razonamiento complejo sobre la estructura del grafo.

3.3.3.3 Salida de la red

Hasta ahora hemos asumido que después de T iteraciones, la representación de cada nodo $h_u^{(T)}$ será la representación final del nodo, es decir, que $z_u = h_u^{(T)} \quad \forall u \in V$.

Sin embargo, existe una aproximación según la cual se establece la salida final z_u como una agregación de todas las representaciones intermedias, y se conoce como *conexiones de salto*.

$$z_u = f_{JK}(h_u^{(1)}, h_u^{(2)}, \dots, h_u^{(T)}) \quad (3.3.19)$$

donde el único requisito que tiene que cumplir f_{JK} es el de la diferenciabilidad, para poder realizar el entrenamiento de la red. En muchas ocasiones la función f_{JK} no es más que el resultado de concatenar todas las representaciones intermedias, pero se puede llegar a implementar de manera más compleja, con redes LSTM por ejemplo.

3.3.4 Características de aristas

En secciones previas, hemos discutido cómo los grafos pueden tener características a nivel de nodo y a nivel de aristas. Las primeras se han ido utilizando mientras explorábamos las funciones de agregación y actualización. Sin embargo, aún no hemos explorado cómo se pueden incorporar las características a nivel de arista en el proceso de paso de mensajes.

La realidad es que se pueden incorporar estas características en forma de atención, como si fuera una ponderación entre nodos, o bien concatenando la información con la representación de los vecinos en el paso de mensajes.

Pongamos como ejemplo, la función de agregación base AGGREGATE, que venía definida como:

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}^{(k)} \left(\left\{ \mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u) \right\} \right) \quad (3.3.20)$$

Si quisiéramos introducir las características de la arista (u, v) , definidos como $e_{(u,v)}$, entonces bastaría con considerar $\mathbf{m}_{\mathcal{N}(u)}$ como

$$\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}^{(k)} \left(\left\{ \mathbf{h}_v^{(k)} \oplus e_{(u,v)}, \forall v \in \mathcal{N}(u) \right\} \right) \quad (3.3.21)$$

Las características de las aristas desempeñan un papel crucial en la mejora de las representaciones y tal y como acabamos de ver, su incorporación es un problema sencillo de resolver, ya que se puede integrar fácilmente en el paso de mensajes y en las funciones de actualización.

Para concluir este capítulo, vamos a repasar brevemente lo que hemos estudiado. En primer lugar, exploramos los fundamentos de los grafos y las Redes Neuronales sobre Grafos, comenzando con una introducción a los grafos y su importancia, avanzando hacia la motivación y aproximaciones iniciales de las GNN. Después, desglosamos el funcionamiento del mecanismo de paso de mensajes, centrándonos en las funciones de agregación y actualización, y destacamos la relevancia de estas técnicas en la captura de la estructura y las características de los grafos.

Con esta comprensión sólida de las GNN y su funcionamiento interno, podemos avanzar al siguiente capítulo, donde discutiremos cómo estas técnicas pueden apli-

carse efectivamente a datos en forma de series temporales, abriendo nuevas posibilidades para el análisis y la predicción en dominios dinámicos, como son precisamente las redes eléctricas.

4

REDES NEURONALES EN SERIES TEMPORALES

Después de comprender en detalle el funcionamiento de las redes GNN, vamos a proceder a ir un paso más allá, y estudiaremos cómo podemos utilizar los mecanismos estudiados sobre grafos dinámicos, es decir, grafos que presentan una variabilidad a lo largo del tiempo.

Sin embargo, para poder comprender correctamente el funcionamiento de las nuevas herramientas, es necesario introducir ciertos conceptos fundamentales en el análisis y predicción de series temporales. Para ello, comenzaremos estudiando qué consideramos como series temporales, y alguna de sus propiedades, y después introduciremos brevemente la arquitectura de los dos tipos de red más usados en este tipo de datos: las Redes Neuronales Recurrentes (RNN, por sus siglas en inglés) y las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés).

4.1 MODELADO DE SERIES TEMPORALES

4.1.1 *Series Temporales*

Una serie temporal es una secuencia ordenada de observaciones registradas en intervalos de tiempo, normalmente constantes. En esta sección consideraremos una serie temporal como una función dependiente del tiempo, es decir, y_t , donde t es el tiempo.

La descomposición matemática usual determina que una serie temporal se puede dividir en cuatro componentes, definidas a continuación:

- La tendencia (T_t) refleja el progreso de la serie a largo plazo. No tiene por qué ser lineal, y puede ser a la baja, a la alza o invariable. Según el tipo de tendencia, se puede modelar por una recta o con alguna otra curva, como por ejemplo una variante de la exponencial.
- La ciclicidad (C_t) representa variaciones repetidas pero no periódicas, es decir, que no tienen una longitud en tiempo constante. Estas variaciones normalmente se deben a factores externos impredecibles, como por ejemplo una acción humana.

- La estacionalidad (S_t) incluye las variaciones periódicas de longitud constante. Las causas de estas variaciones también son periódicas, como por ejemplo la temperatura, el día de la semana, la hora del día o períodos de vacaciones.
- El ruido o residuo (R_t) abarca todas las variaciones debidas a influencias completamente aleatorias.

En la mayoría de problemas de Aprendizaje Supervisado, es necesario establecer la estructura subyacente de la serie temporal para poder determinar correctamente qué modelo usar y para ello usamos un proceso estadístico llamado descomposición. Al descomponer una serie temporal, podemos predecir de manera separada cada una de ellas y ajustarnos mejor a las propiedades particulares de cada componente, lo que nos permite hacer predicciones más precisas.

Existen distintos métodos para descomponer una serie temporal y todos ellos tienen un objetivo común: aislar cada componente de la serie temporal con la mayor precisión posible. La mayoría de los enfoques de descomposición consideran los componentes de tendencia y ciclo como un único componente determinado tendencia-ciclo (T_t).

El proceso de descomposición es empírico y consiste en retirar primero la componente tendencia-ciclo y más adelante separar la componente de estacionalidad. La componente de ruido se asume como aleatoria, y como tal no se puede prever, pero sí identificar.

La representación matemática general de cualquier método de descomposición es la siguiente:

$$y_t = f(T_t, S_t, R_t)$$

donde y_t es el valor de la serie temporal en el tiempo t , T_t es la componente tendencia-ciclo en el tiempo t , S_t es la componente de estacionalidad en el tiempo t , y R_t es la componente residual en el tiempo t .

Los dos enfoques principales de descomposición son la descomposición aditiva y la descomposición multiplicativa.

El modelo aditivo establece la serie temporal como una suma de las tres componentes, es decir:

$$y_t = T_t + S_t + R_t$$

Y el modelo multiplicativo considera el producto de las componentes:

$$y_t = T_t * S_t * R_t$$

También podemos obtener otros enfoques realizando una combinación entre el aditivo y el multiplicativo, sin embargo, la descomposición presentada puede no ser la correcta para todas las series temporales, debido a que pueden existir ciertos componentes o tendencias en la serie no consideradas por la descomposición usual.

Aunque existen multitud de algoritmos de predicción específicos para las series temporales, en nuestro caso particular nos interesan únicamente las herramientas que podamos combinar con los mecanismos estudiados en el capítulo anterior, es decir, arquitecturas de redes neuronales que produzcan un buen rendimiento en secuencias de datos. Como ya se ha mencionado anteriormente, estudiaremos dos de ellas: las Redes Neuronales Convolucionales (CNN), y las Redes Neuronales Recurrentes (RNN).

4.1.2 Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales, conocidas como CNN por sus siglas en inglés, son una clase especializada de redes neuronales diseñadas para el procesamiento de datos con una disposición en forma de cuadrícula, como imágenes, que pueden ser representadas como matrices de píxeles.

Su nombre viene dado por la operación matemática en la que se basa el funcionamiento de este tipo de red, y que veremos en detalle más adelante: la convolución. Así, una red convolucional es una red neuronal en la que se usa la convolución en vez de la multiplicación entre matrices para al menos una de sus capas [14].

Ahora procederemos a examinar el funcionamiento de estas arquitecturas, sin profundizar demasiado en los detalles matemáticos. Nuestro objetivo principal es desarrollar una comprensión básica de cómo operan estos mecanismos, con el fin de entender su relevancia en las estructuras de redes temporales de grafos.

4.1.2.1 Arquitectura CNN

Una red CNN normalmente se compone de tres capas: una capa convolucional, una capa de *pooling*, y una capa completamente conectada, como en las redes neuronales más básicas. Aunque esta sería la estructura más sencilla, es necesario notar que se pueden combinar de distintas maneras, o incluso incorporar otro tipo de capas y operaciones entre ellas. Esta arquitectura se puede estudiar en detalle en la Figura 2.

Comenzamos con la **capa convolucional**, que desencadena la operación central de convolución. Esta operación realiza la multiplicación entre dos matrices: la primera es un conjunto de parámetros entrenables conocido como el *kernel*, y la segunda es una sección de los datos presentes en cada capa.

La premisa fundamental es que el kernel tiene una dimensión mucho menor que los datos, lo que permite que se deslice a lo largo y ancho de la matriz, generando una representación comprimida de la porción correspondiente de los datos. Como resultado, se obtiene un mapa bidimensional de activación que contiene la salida del kernel para cada posición en la matriz de datos.

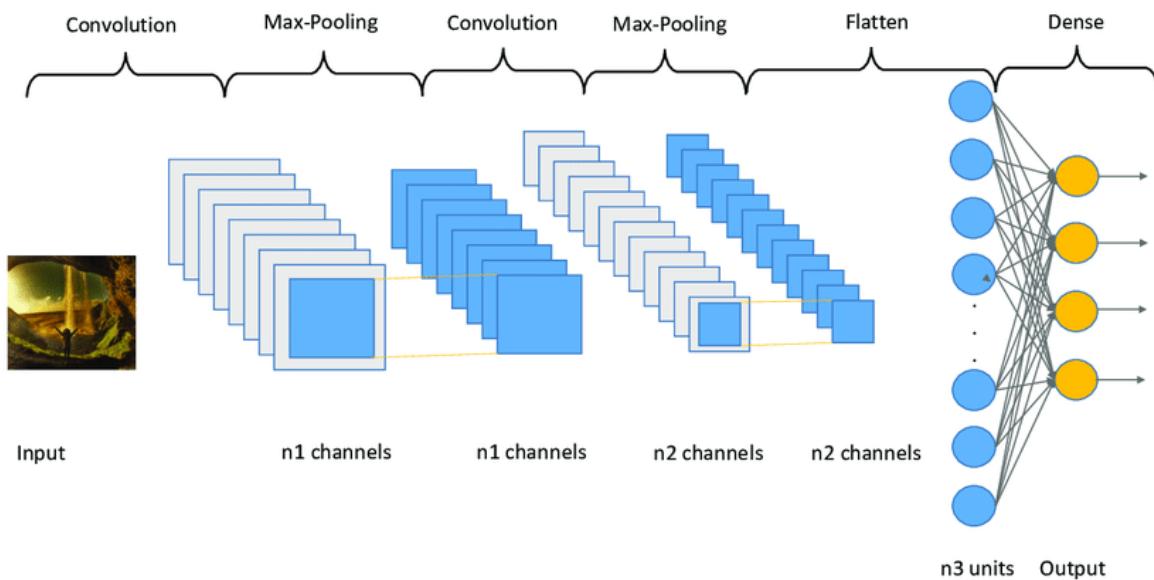


Figura 2: Arquitectura de una CNN

La operación de convolución se fundamenta en tres conceptos clave: interacción limitada, compartición de parámetros y representación equivariante. Vamos a estudiar en qué consiste cada uno de ellos.

En primer lugar, en las redes neuronales convolucionales, la interacción entre unidades está limitada. A diferencia de las redes tradicionales, donde cada unidad de entrada se conecta con todas las unidades de salida, en las CNNs esta interacción es restringida, lo que resulta especialmente útil en el procesamiento de imágenes, donde la cantidad de píxeles puede ser enorme. Esta limitación además supone una mejora en la eficiencia estadística del modelo.

Para la segunda, es necesario observar que si se comprueba la presencia de una característica en un punto A de la imagen, entonces tiene sentido comprobar su presencia en otros puntos. En las CNNs, esto se logra utilizando el mismo conjunto de pesos para la creación de los mapas de activación, a diferencia de las redes neuronales convencionales donde cada peso se usa una sola vez.

Debido a la compartición de parámetros, se puede afirmar que las representaciones obtenidas son equivariantes, lo que significa que si modificamos la entrada de una manera específica, la salida se verá afectada de manera similar.

El funcionamiento de la capa convolucional puede verse en resumen en la Figura 3.

Aunque no se haya tratado como una capa explícita, es común agregar una capa no lineal después de la capa convolucional, que aplica alguna de las operaciones no lineales mencionadas previamente.

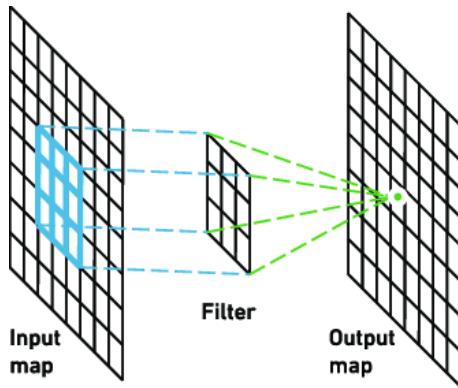


Figura 3: Capa convolucional de una CNN

Pasando al segundo tipo de capa en las CNNs: la capa de **pooling**, que se encarga de reemplazar la salida de la capa convolucional con un resumen de la misma, realizado mediante operaciones entre los valores de la matriz de activación. Estas operaciones suelen llevarse a cabo sobre un vecindario de tamaño específico, deslizándose de manera similar al kernel en la capa convolucional, y pueden incluir la norma L₂ o la media, entre otras.

Finalmente, se añade una capa completamente conectada, cuyo propósito es mapear la representación desde la entrada hasta la salida, ajustando las dimensiones según sea necesario.

Hasta ahora hemos estudiado la versión básica de las CNN, que se entiende de una manera más sencilla si se utiliza el ejemplo de las imágenes debido a su estructura matricial de píxeles. Sin embargo, en nuestro caso particular, nos interesa utilizar este tipo de red para series temporales, y la solución es relativamente intuitiva: se considera la serie temporal como una matriz unidimensional, y se sigue el mismo enfoque de procesamiento.

4.1.3 Redes Neuronales Recurrentes

A continuación nos adentraremos en el estudio de las redes neuronales recurrentes (*Recurrent Neural Network*), una estructura que ha demostrado ser altamente efectiva en la resolución de problemas relacionados con series temporales. Describiremos su arquitectura básica, detallando cada uno de sus componentes y su funcionamiento. Además, introduciremos las herramientas matemáticas subyacentes que permiten a esta estructura utilizar información relevante de una secuencia temporal durante un período de tiempo prolongado.

4.1.3.1 Arquitectura de RNN

Si recordamos la arquitectura más básica de una red neuronal (Figura 57), y tras lo que hemos estudiado, vemos que entre una observación procesada y la siguiente, no

hay ninguna relación. Por el contrario, si estudiamos la arquitectura de las redes recurrentes (Figura 4), vemos que se han añadido enlaces al propio nodo. Estos enlaces son los encargados de realizar un procesamiento en secuencia, y son lo que le dan el nombre a este tipo de redes.

Entrada Capa oculta Salida

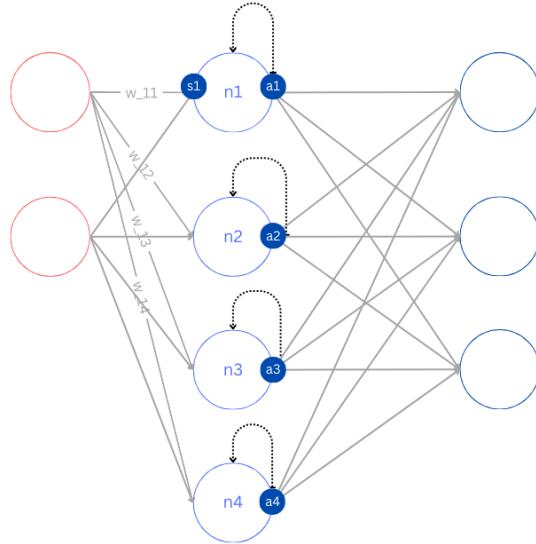


Figura 4: Arquitectura de una red recurrente

Además, aunque en la Figura 4 se ha mostrado una arquitectura posible de red, esta no es la única, y es que las entradas y salidas pueden variar de longitud, generando distintas arquitecturas, como por ejemplo uno-a-muchos, o muchos-a-uno. En la práctica, se utilizan diferentes tipos de RNN para diferentes casos de uso como, por ejemplo, la generación de música, la clasificación de sentimientos y la traducción automática.

Formalmente, la arquitectura de una capa RNN vista a alto nivel, es decir, considerando las capas, y no las neuronas particulares, es como sigue:

$$\begin{aligned} z_t &= Ua_{t-1} + Wr_t + b, \quad t = 0, \dots, N \\ a_t &= \sigma_t(z_t), \quad t = 0, \dots, N \end{aligned} \tag{4.1.1}$$

donde t es el índice de tiempo o punto en la secuencia, N es el horizonte de tiempo final, r_t es el vector de entrada, z_t es el estado oculto, que dará como salida el vector de activación a_t , después de procesarlo con una función de activación σ_t .

Como vemos, en cada paso t se utiliza el valor de activación de la capa en el paso anterior, por medio de una matriz de parámetros denotada U . Finalmente, se suele incorporar una última capa lineal, estática, para producir la salida definitiva de la red, que formalmente sería como sigue:

$$\begin{aligned} z_t &= Va_t + c, \quad t = 0, \dots, N \\ p_t &= \sigma_t(z_t), \quad t = 0, \dots, N \end{aligned} \tag{4.1.2}$$

donde de nuevo, z_t es el vector oculto de la capa, a_t es el vector de activación y σ_t es la función de activación. En este caso, la función de activación se adaptaría al tipo de salida buscada, por ejemplo, si quisieramos obtener un vector de probabilidades, usaríamos la función *softmax*.

4.1.3.2 Entrenamiento en RNN

Acabamos de ver un ejemplo de cómo se obtendrían h_t y a_t en una capa, y sabemos que este proceso forma parte del *forward propagation*.

Recordamos que en las redes neuronales tenemos un conjunto de entrenamiento dividido en lotes o *batches*, a través del cual se realizan iteraciones, o *epochs*. Una vez se ha terminado de procesar todas las entradas de un lote, se actualizan los pesos de cada capa.

En realidad, en las redes recurrentes no se procesan las observaciones de manera independiente, sino que se procesan en conjuntos secuenciales, por lo que en cada capa obtenemos un conjunto de estados ocultos, que sirven como entrada para la siguiente capa. Por este motivo, la actualización de los pesos de la red se tiene que hacer de una manera específica.

El algoritmo *Backpropagation Through Time* (BTT) es una extensión del algoritmo de retropropagación utilizado para entrenar redes neuronales recurrentes (RNN). Dado que las RNN poseen conexiones temporales que permiten el procesamiento secuencial de datos, BTT *desenrolla* la red en el tiempo, creando una copia de la red para cada paso de tiempo. Esto transforma la red RNN en una red que denominaremos red *feedforward* profunda, donde cada capa corresponde a un paso de tiempo.

Durante el proceso de entrenamiento, el algoritmo BTT calcula el error en cada paso temporal y lo retropropaga a través de estas capas desenrolladas, como si se estuviera realizando *backpropagation* al uso.

En la siguiente sección, exploraremos dos tipos avanzados de redes neuronales recurrentes, *Long Short-Term Memory* (LSTM) y *Gated Recurrent Units* (GRU), que son fundamentales en las herramientas de redes de grafos temporales que se analizarán más adelante.

Estas arquitecturas están diseñadas para abordar las limitaciones de las redes neuronales recurrentes tradicionales, que veremos a continuación. Sin embargo, la manera en la que lo logran es distinta: Las LSTM lo hacen mediante el uso de una estructura de memoria sofisticada, mientras que las GRU emplean una arquitectura más simplificada pero igualmente eficaz, como veremos a continuación.

Con este último comentario damos por finalizada la introducción a las redes RNN y CNN, y podemos proceder a estudiar en detalle los bloques LSTM y GRU.

4.1.4 Arquitecturas Avanzadas de RNN

Las Redes Neuronales Recurrentes se suelen utilizar para procesar secuencias de datos, como texto o series temporales, pero el algoritmo de retropropagación a través del tiempo las hace susceptibles de sufrir el **problema de explosión del gradiente**.

Este problema ocurre durante cuando el valor del gradiente se vuelve extremadamente grande, y puede llevar a que los pesos de la red neuronal se actualicen de manera muy brusca, lo que a su vez puede provocar que el modelo no converja o que tenga un desempeño pobre. El problema de explosión del gradiente supone el principal desafío en el entrenamiento de RNN.

Para abordar este problema, se han desarrollado arquitecturas RNN más avanzadas, que estudiaremos en detalle en esta sección.

4.1.4.1 Arquitectura LSTM

En secciones anteriores veíamos que una red neuronal está compuesta por capas que a su vez incorporan unidades o neuronas. Las redes LSTM introducen bloques de memoria que sustituyen a las neuronas.

Un bloque de memoria tiene componentes que lo convierten en una estructura más inteligente que las neuronas simples con una función de activación presentes en las redes neuronales usuales. Los bloques de memoria se comunican entre sí a través de puertas (*gates*) para controlar qué información se retiene y qué se olvida en cada paso temporal.

Las componentes de un bloque de memoria son:

- Un estado de celda que almacena información a largo plazo, que se actualiza y modifica a través de las puertas, y que será notado por c_t .
- Un estado de celda oculto, que es una representación comprimida del estado de celda, será notado por h_t .
- Una serie de puertas, que se abren o se cierran en función de las entradas, lo que permite controlar la cantidad de información que fluye hacia el estado de celda y hacia las salidas de la capa y de esta manera, la red puede aprender a recordar información importante y a ignorar la información innecesaria en una serie temporal. Existen tres tipos de puerta en un bloque:
 - **Puerta de olvido:** decide qué información almacenada en el bloque se pierde.
 - **Puerta de entrada:** decide qué información de entrada se utiliza para actualizar el estado.

- **Puerta de salida:** decide la salida, basándose en la entrada y en el estado del bloque.

En este tipo de red neuronal, los pesos están presentes en las puertas de cada bloque de memoria, y son lo que permite que se aprendan los patrones entre las entradas y las salidas de la red. En la Figura 5 se puede comprobar el detalle del bloque de memoria.

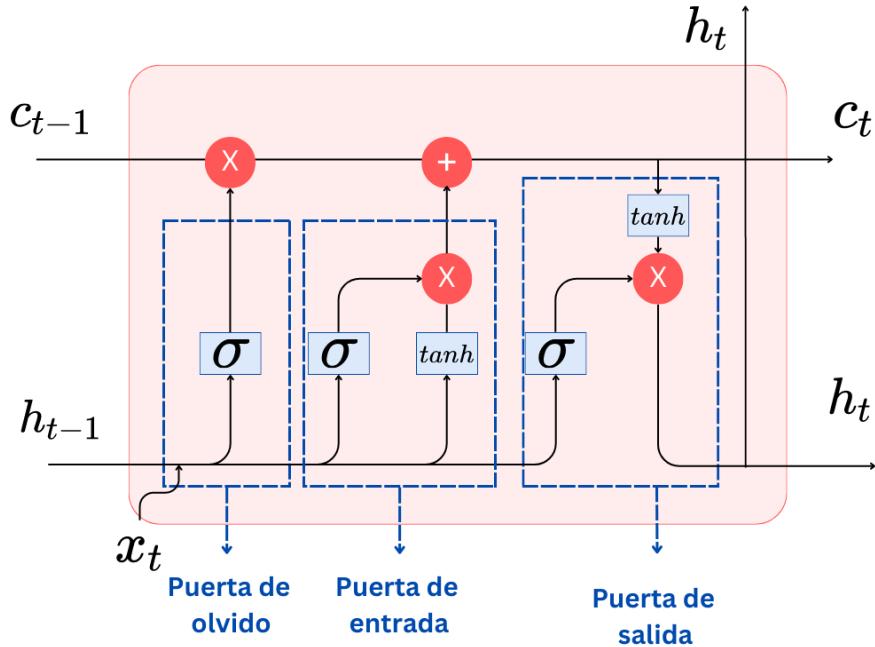


Figura 5: Estructura interna de un bloque de memoria

A continuación vamos a explicar dónde se encuentran los pesos y los sesgos en este tipo de arquitecturas.

Comenzamos por la **puerta de olvido**, donde se combina la entrada x_t y el estado oculto de celda anterior h_{t-1} , produciendo como salida un valor f_t , mediante la siguiente expresión:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (4.1.3)$$

donde W_f representan los pesos asociados con la entrada y con el estado oculto anterior, que se han concatenado, y b_f representa el sesgo. Cabe destacar el uso de la función sigmoide, que lo convierte en un valor entre 0 y 1. Más adelante, se multiplicará por el estado de celda anterior, y es donde se establecerá qué información del estado anterior es descartada.

Continuamos introduciendo en detalle la **puerta de entrada**. Se puede dividir en dos partes, la primera se encarga de valorar la importancia de la información que aporta la entrada, y la segunda se encarga de transformar la información.

Para la primera, se vuelve a realizar la misma operación, esta vez con pesos distintos:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (4.1.4)$$

Y para la segunda, simplemente hay que utilizar la función \tanh para que los valores se encuentren en el intervalo $[-1, 1]$:

$$n_t = \tanh(W_n[h_{t-1}, x_t] + b_n) \quad (4.1.5)$$

Una vez se ha obtenido la salida de la puerta de olvido y de la puerta de entrada, se puede proceder a realizar la actualización del estado de celda, que es como sigue:

$$c_t = f_t * c_{t-1} + i_t * n_t \quad (4.1.6)$$

Por último, se tiene que calcular la salida de la celda, por medio de la puerta de salida, de nuevo mediante la expresión:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (4.1.7)$$

Para calcular el siguiente estado oculto de la celda, se vuelve a utilizar \tanh para transformar el estado anterior, c_t :

$$h_t = o_t * \tanh(c_t) \quad (4.1.8)$$

Aunque nos hemos fijado en el detalle de una sola unidad o celda, en realidad una red neuronal LSTM está compuesta por varias capas, formadas a su vez por m unidades. A nivel práctico, en una capa únicamente hay un bloque de memoria, por lo que resulta natural preguntarse cómo puede darse una situación en la que haya m unidades. Para responder a esta cuestión, es necesario romper con la asociación entre neurona y bloque de memoria, y considerar el propio bloque como una capa, donde la dimensión del estado oculto será m . Esto quiere decir que, en principio, cuantas más unidades tenga una capa LSTM, más complejos (dimensiones más altas) serán los patrones que podrá recordar.

Las redes LSTM son altamente efectivas para gestionar dependencias a largo plazo mediante su compleja estructura de memoria. Sin embargo, existen otras arquitecturas que, aún siendo más simples, conserva la capacidad de manejar secuencias largas, y que además no tienen los mismos requisitos de memoria y computación. Una de ellas son las *Gated Recurrent Units*, GRU, como veremos a continuación.

4.1.4.2 Arquitectura GRU

Aunque las *Gated Recurrent Units* siguen un principio similar a las redes LSTM, pues los bloques funcionan de una manera parecida, hay una diferencia estructural entre ambas, y es que en las unidades GRU se incluyen únicamente dos puertas, en lugar de tres, lo que reduce la complejidad computacional. Se puede estudiar en detalle la estructura de una unidad en la Figura 6.

Vamos a seguir el mismo procedimiento que en el caso anterior, y comenzaremos por la **puerta de olvido**, que decide cuánto del estado oculto anterior h_{t-1} debe olvidarse antes de incorporar la nueva información x_t , produciendo como salida un valor r_t :

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \quad (4.1.9)$$

Luego procedemos con la **puerta de actualización**. En ella se combina la entrada x_t y el estado oculto anterior h_{t-1} para producir un valor z_t , mediante la siguiente expresión:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \quad (4.1.10)$$

donde W_z representan los pesos asociados con la entrada y con el estado oculto anterior, que se han concatenado, y b_z representa el sesgo. La función sigmoide se utiliza para obtener un valor entre 0 y 1, determinando cuánto del estado anterior se mantiene en el nuevo estado.

Después de calcular las puertas de olvido y actualización, procedemos a generar el contenido candidato del nuevo estado oculto \tilde{h}_t . Este candidato se calcula utilizando la entrada actual y el estado oculto anterior modificado por la puerta de olvido:

$$\tilde{h}_t = \tanh(W_h[r_t * h_{t-1}, x_t] + b_h) \quad (4.1.11)$$

Finalmente, el nuevo estado oculto h_t se obtiene combinando el estado oculto anterior y el contenido candidato, ponderados por la puerta de actualización:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad (4.1.12)$$

De esta manera, las GRU controlan de forma eficiente el flujo de información a través de sus puertas de reinicio y actualización, permitiendo que la red modele dependencias temporales sin la complejidad adicional de las LSTM.

Es necesario notar, además, que existen variantes de ambos tipos de RNN, y es que se puede modificar la arquitectura interna del bloque de memoria, para reducir aún más su complejidad, en lo que llamaríamos variantes *slim*. Sin embargo, no entraremos en ese tipo de mecanismos por no ser el objeto principal de este estudio.

Antes de continuar, es necesario recapitular lo que hemos estudiado hasta el momento. Hemos explorado dos áreas fundamentales: las redes neuronales sobre grafos, incluyendo sus variantes convolucionales y de atención, y las redes para series temporales, tales como las RNN (incluyendo LSTM y GRU) y las CNN.

En el próximo capítulo, fusionaremos estos conceptos para adentrarnos en el estudio detallado de las herramientas principales de este trabajo: las Redes Neuronales sobre Grafos Temporales.

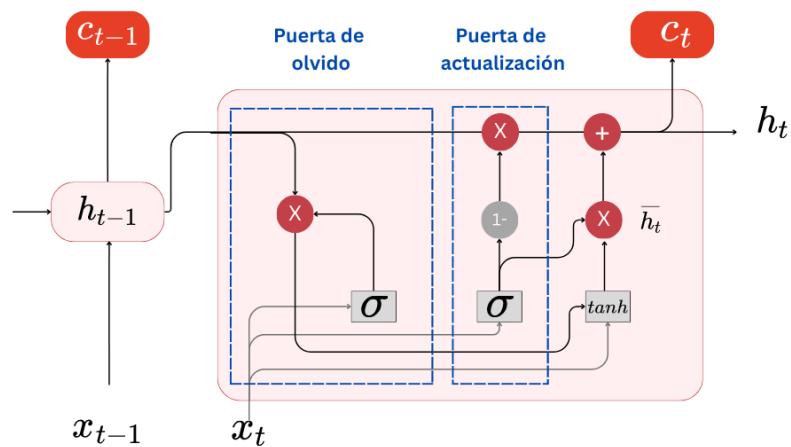


Figura 6: Estructura interna de una unidad GRU

5

REDES NEURONALES DE GRAFOS TEMPORALES

En este capítulo, nos adentraremos por fin en el estudio de los grafos temporales o grafos dinámicos, comenzando por comprender su definición y estructura fundamental. Exploraremos los diferentes tipos de grafos temporales y las problemáticas inherentes a ellos, tales como la evolución de los nodos y las aristas a lo largo del tiempo, y los desafíos en la representación y procesamiento de estos datos dinámicos.

Posteriormente, analizaremos las redes neuronales diseñadas específicamente para modelar grafos temporales, centrando el análisis en dos enfoques principales: las redes basadas en arquitecturas recurrentes (RNN), y las redes basadas en arquitecturas convolucionales (CNN). A lo largo del capítulo, se proporcionará una visión integral de estos enfoques, destacando sus capacidades y limitaciones en la modelización de grafos temporales.

5.1 GRAFOS TEMPORALES

En capítulos anteriores hemos introducido los fundamentos teóricos del aprendizaje profundo en grafos. Hasta el momento hemos considerado que los grafos en los que se basan estos métodos son estáticos, es decir, que no varían a lo largo del tiempo. Sin embargo, la mayoría de las interacciones reales modeladas por grafos, como por ejemplo las redes eléctricas, o las redes sociales, son dinámicas. Reducir estas interacciones a grafos estáticos produce resultados sub-óptimos, como es de esperar [43].

Por lo tanto, es necesario adaptar los mecanismos que tratan con grafos dinámicos para poder tener en cuenta la evolución del mismo a lo largo del tiempo, como veremos más adelante.

5.1.1 *Tipos de Grafos Dinámicos*

Dentro de los grafos dinámicos, que evolucionan a lo largo del tiempo, podemos considerar dos subtipos diferentes, en función del tipo de evolución que presenten [23]. Por una parte, podemos tener grafos cuya evolución es discreta, denominados

grafos temporales dinámicos discretos (DTDG, por sus siglas en inglés), y generados capturando el estado del grafo a intervalos de tiempo regulares.

Formalmente, un DTDG es una secuencia $[G(1), G(2), \dots, G(t)]$ de instantáneas del grafo donde cada $G(t) = (V(t), A(t), X(t))$ tiene vértices $V(t)$, una matriz de adyacencia $A(t)$ y una matriz de características $X(t)$.

Por otra parte, si la evolución es continua, diremos que el grafo es un **grafo temporal dinámico continuo** (CTDG, por sus siglas en inglés). Formalmente, es un par $(G(t_0), O)$ donde $G(t_0) = (V(t_0), A(t_0), X(t_0))$ es un grafo estático que representa un estado inicial en el tiempo t_0 y O es una secuencia de observaciones o eventos temporales.

Cada observación es una tupla de la forma $\{\text{tipo de evento, evento, marca temporal}\}$ donde el tipo de evento puede ser una adición de nodo o arista, eliminación de nodo o arista, actualización de características de nodo, etc. El evento representa el evento real que ocurrió, y la marca temporal es el momento en que ocurrió el evento.

En este trabajo nos centraremos en el primer tipo, puesto que los datos con los que trataremos corresponden a estados de una red eléctrica en intervalos de tiempo regulares.

Además, también podemos clasificar distintos grafos dinámicos según la ubicación de su evolución dentro de las propiedades del grafo. Por ejemplo, en una red social donde observamos variaciones tanto en las conexiones como en las etiquetas, estamos ante un **grafo dinámico con evolución topológica y de etiquetado**.

En contraste, podríamos tener una red en la que las etiquetas o pesos de los nodos son constantes, pero hay variaciones en las conexiones entre nodos. En este caso, nos referimos a un **grafo dinámico con evolución topológica**.

Por último, en el contexto relevante para este estudio, consideremos una red eléctrica. Aquí, las conexiones no se ven afectadas ya que corresponden a la infraestructura instalada. Sin embargo, sí se observan variaciones en el peso de dichas conexiones y en los nodos. Este tipo de grafo se clasifica como un **grafo dinámico con evolución de etiquetado**.

Podemos ver resumidas cada una de las tres alternativas en la Figura 7.

En este punto ya conocemos los fundamentos de los grafos dinámicos, y resulta natural preguntarse cómo podríamos utilizar las herramientas introducidas en el Capítulo 3 para procesarlos. Aunque en la práctica existen multitud de arquitecturas distintas, tal y como veremos más adelante, podemos estudiar de manera general su funcionamiento. A continuación introduciremos los conceptos básicos de las Redes de Grafos Temporales (TGNs, por sus siglas en inglés).

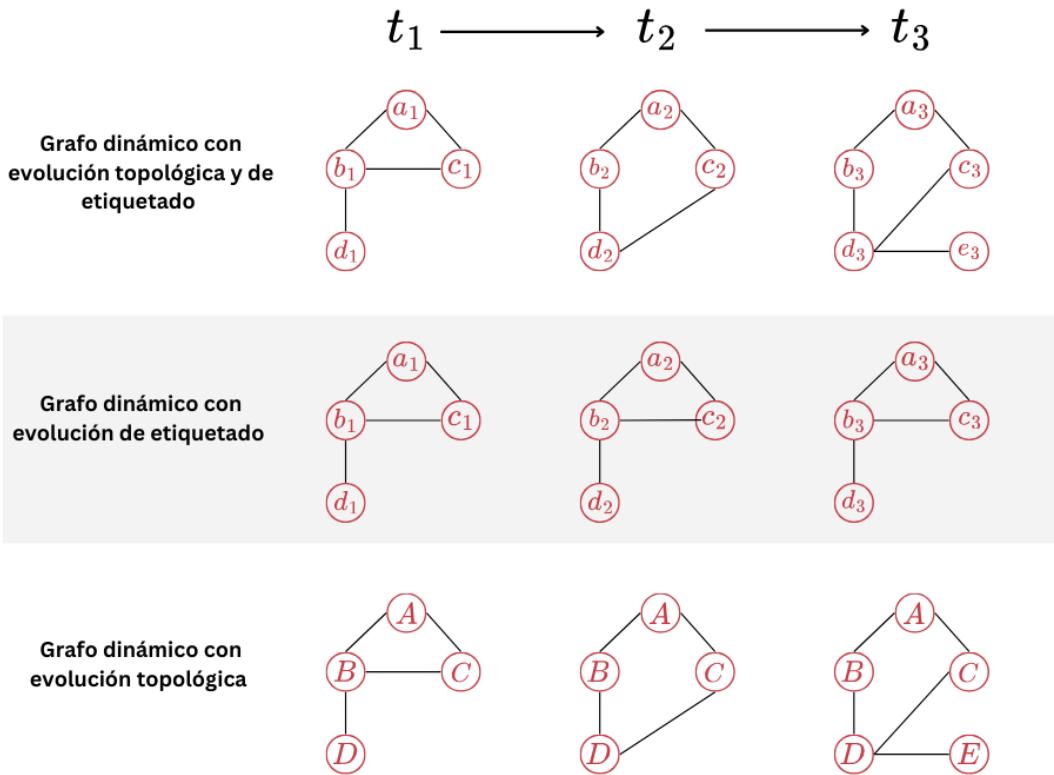


Figura 7: Tipos de Grafos Dinámicos en cuanto a localización de evolución

5.2 REDES NEURONALES DE GRAFOS TEMPORALES

Una forma natural de desarrollar modelos para grafos DTDG es combinar las redes neuronales de grafos, GNNs, con redes neuronales para datos secuenciales, RNNs, o CNNs. En esta sección vamos a estudiar ambas combinaciones, introduciendo una arquitectura básica para cada una de las alternativas.

5.2.1 Redes basadas en RNN

Suponemos que tenemos un grafo DTDG, es decir, $[G(1), G(2), \dots, G(\tau)]$ de instantáneas donde cada $G(t) = (V(t), A(t), X(t))$ tiene vértices $V(t)$, una matriz de adyacencia $A(t)$ y una matriz de características $X(t)$ para cada instante temporal $t \in \{1, \dots, \tau\}$.

Tal y como veíamos en el capítulo dedicado a las Redes Neuronales de Grafos (GNNs), nuestro objetivo es obtener una representación adecuada del grafo. Esta representación nos permitirá combinar las GNNs con otros tipos de redes neuronales para resolver diversos problemas, que van desde la predicción de enlaces en nodos específicos hasta la clasificación del grafo en su totalidad.

En este caso particular, buscamos obtener una representación del grafo en un instante de tiempo t , que no solo tenga en cuenta los datos presentes en ese instante, sino también la información de los instantes anteriores.

El proceso pasa por aplicar una GNN a cada uno de los grafos $G(t)$, obteniendo una secuencia de representaciones intermedias, en forma de secuencia de matrices $[Z(1), Z(2), \dots, Z(\tau)]$, donde cada fila corresponde a la codificación de un nodo.

Para incorporar el aspecto temporal del DTDG, podemos utilizar una red RNN sobre la secuencia de representaciones, por lo que la representación temporal del grafo en el instante t será el estado oculto del modelo RNN tras procesar la secuencia.

Este procedimiento se puede hacer a nivel de grafo o a nivel de nodo. En el segundo caso, para un nodo v_i , tendríamos una secuencia $[Z(1)_i, Z(2)_i, \dots, Z(\tau)_i]$, y bastaría con utilizar el estado oculto de la capa RNN como la representación temporal del nodo.

Vamos a integrar todo lo anterior para establecer el conjunto de ecuaciones que definen un modelo de este tipo, considerando para el ejemplo bloques de memoria LSTM.

En primer lugar, tenemos la capa GNN, definida sobre la secuencia de matrices de características de nodo y la matriz de adyacencia en el instante t , $X(t)$ y $A(t)$, respectivamente:

$$Z(t) = \text{GNN}(X(t), A(t)) \quad (5.2.1)$$

Después, tenemos todo el proceso correspondiente a un bloque LSTM, definido como sigue:

1. En primer lugar, tenemos la puerta de olvido

$$F(t) = \sigma(W_f[H(t-1), Z(t)] + b_f) \quad (5.2.2)$$

2. Despues, tenemos la puerta de entrada, definida en dos partes

$$I(t) = \sigma(W_i[H(t-1), Z(t)] + b_i) \quad (5.2.3)$$

$$N(t) = \tanh((W_n[H(t-1), Z(t)] + b_n)) \quad (5.2.4)$$

3. Una vez se ha obtenido la salida de la puerta de olvido y de la puerta de entrada, se puede proceder a realizar la actualización del estado de celda, que es como sigue

$$C(t) = F(t) \otimes C(t-1) + I(t) \otimes N(t) \quad (5.2.5)$$

4. A continuación se calcula el valor de la puerta de salida:

$$O(t) = \sigma(W_o[H(t-1), Z(t)] + b_o) \quad (5.2.6)$$

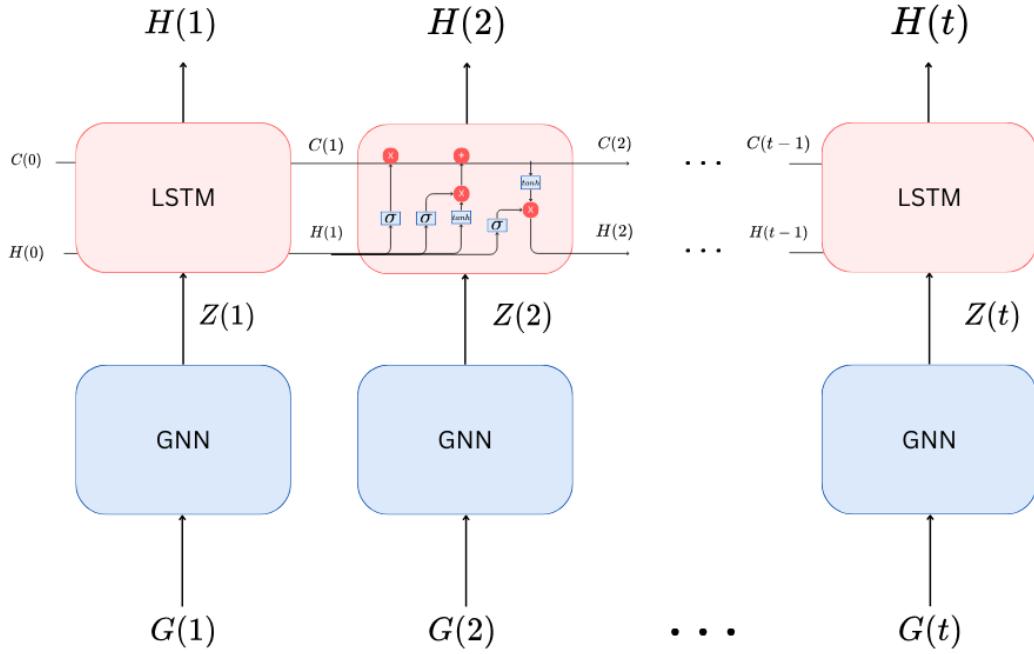


Figura 8: Arquitectura GNN-LSTM

5. Por último, calculamos el estado oculto de la celda, mediante

$$H(t) = O(t) * \tanh(C(t)) \quad (5.2.7)$$

Se puede estudiar en detalle la arquitectura de esta red de ejemplo en la Figura 8.

Es necesario insistir en que tanto las ecuaciones como la arquitectura mostrada no es más que una de las múltiples que podríamos tener. Para empezar, la arquitectura GNN puede variar según si es una red con mecanismos de atención (GAT), o una red convolucional (GCN), y además, la capa LSTM puede ser otro tipo de RNN, creando las diferentes variaciones: GCN-LSTM, GAT-LSTM, GCN-GRU, GAT-GRU, etc.

Tal y como veíamos en la sección anterior, existen grafos dinámicos cuya estructura se mantiene estable en el tiempo. En esos casos, al no tener diferencias en la dimensionalidad de las características del grafo, podríamos invertir el orden de las capas, situando en primer lugar las capas recurrentes y en segundo lugar las GNNs.

5.2.2 Redes basadas en CNN

De la misma manera que acabamos de ver, se pueden combinar capas de una red CNN junto con una red sobre grafos. El proceso, de nuevo, comienza aplicando una GNN a cada uno de los grafos $G(t)$, obteniendo una secuencia de representaciones intermedias $[Z(1), Z(2), \dots, Z(\tau)]$.

El siguiente paso es apilar las representaciones de cada nodo v_i , $[Z(1)_i, Z(2)_i, \dots, Z(\tau)_i]$ en una única matriz $H(0, i)$, definida como sigue:

$$H(t)_{(0,i)} = Z(t)_i + p(t) \quad t \in \{1, 2, \dots, \tau\}, i \in \{1, 2, \dots, |V|\} \quad (5.2.8)$$

donde donde $p(t)$ es el vector de codificación posicional para la posición t , que incluye información sobre la posición relativa de los nodos en la secuencia de entrada para que el modelo pueda comprender el orden y las relaciones espaciales entre ellos.

Por lo tanto, la fila t de $H_{(0,i)}$ contiene la representación $Z(t)_i$ de v_i obtenida aplicando el modelo GCN sobre el grafo $G(t)$, además de la codificación posicional. Y entonces el último paso sería aplicar una red convolucional (CNN), sobre la matriz $H(0, i)$.

5.2.3 Redes Profundas

Para finalizar este capítulo vamos a estudiar cómo convertir las redes que acabamos de introducir en redes profundas, pues la aproximación introducida incluye una única capa de cada tipo.

Consideramos el modelo GNN-LSTM anterior, vamos a renombrar las salidas de los módulos GNN y LSTM como $\{Z(1, t) : \forall t \in \{1, \dots, \tau\}\}$ y $\{H(1, t) : \forall t \in \{1, \dots, \tau\}\}$, respectivamente, para poder representar que nos encontramos en la primera capa de la red.

Podemos entonces, unir otro bloque GNN-LSTM, que recibe como entrada la secuencia $\{H(1, t) : \forall t \in \{1, \dots, \tau\}\}$ y produce, por una parte, la salida del bloque GNN: $\{Z(2, t) : \forall t \in \{1, \dots, \tau\}\}$ y la secuencia de estados ocultos de la capa LSTM, $\{H(2, t) : \forall t \in \{1, \dots, \tau\}\}$.

Por lo tanto, el modelo profundo correspondiente a una arquitectura GNN-LSTM viene dado formalmente por:

$$\begin{aligned} \mathbf{Z}(l, t) &= \text{GNN}(\mathbf{H}(l-1, t), \mathbf{A}(t)) \\ \mathbf{H}(l, t), \mathbf{C}(l, t) &= \text{LSTM}(\mathbf{Z}(l, t), \mathbf{H}(l, t-1), \mathbf{C}(l, t-1)) \end{aligned} \quad (5.2.9)$$

Podríamos considerar las dos ecuaciones anteriores como la definición de un **bloque GNN-LSTM**.

Esta sección concluye la exploración del marco teórico de este trabajo. Comenzamos introduciendo los principios del Aprendizaje Supervisado y luego nos sumergimos en los fundamentos de las redes neuronales sobre grafos. Posteriormente, examinamos los conceptos esenciales de las series temporales, así como las herramientas más comunes para su análisis. Finalmente, unimos todos estos conceptos en el estudio de las redes neuronales temporales sobre grafos, permitiéndonos trabajar con grafos dinámicos que evolucionan a lo largo del tiempo, como los grafos de redes eléctricas, que serán los protagonistas de las siguientes secciones.

Ahora, nos enfocaremos en examinar la metodología del trabajo. Presentaremos el conjunto de datos y más adelante nos sumergiremos en un estudio detallado de cada una de las arquitecturas que se han probado para abordar el problema, así como la presentación de los experimentos realizados.

Parte III

METODOLOGÍA

Presentación de los datos utilizados en el estudio. Descripción detallada de los problemas resueltos, junto con las técnicas y arquitecturas utilizadas.

6

PRESENTACIÓN DE LOS DATOS Y SOFTWARE DESARROLLADO

Como inicio de la presentación de la metodología de este trabajo, vamos a proceder a introducir los datos que vamos a utilizar a lo largo de las secciones siguientes. En primer lugar, exploraremos el origen del dataset y cómo fue recopilado, y después, realizaremos un breve análisis exploratorio, necesario para entender alguna de las decisiones de modelado que se han llevado a cabo. Más adelante, presentaremos los problemas que vamos a resolver en los capítulos siguientes, y finalmente terminaremos con una exposición del software desarrollado.

6.1 ORIGEN DE LOS DATOS

Los datos que utilizaremos a lo largo de este estudio provienen del artículo *A Multiscale Time-series Dataset with Benchmark for Machine Learning in Decarbonized Energy Grids* [46]. Este dataset sintetiza una red eléctrica conjunta de transmisión y distribución, capturando las interacciones de las componentes red (Figuras 10, 12 y 11). Además, incluye mediciones de potencia, voltaje y corriente a lo largo de varias escalas espacio-temporales.

Los datos en forma de series temporales que proporciona el artículo no son datos reales, sino que son el resultado de ejecutar distintas simulaciones sobre la red. Esto es debido a tres razones principales: la primera es que los datos operativos reales de una red eléctrica son confidenciales y no se pueden compartir públicamente. En segundo lugar, ciertos eventos de alto impacto son raramente observables en casos reales, por lo que serían insuficientes para el entrenamiento de algoritmos de ML. Por último, los datos reales no pueden capturar la dinámica de futuras redes teniendo en cuenta la influencia de la energía renovable en las mismas.

El proceso de obtención de los datos comienza recolectando la información real meteorológica y de carga, creando las primeras series temporales. A partir de ellas, se obtienen los perfiles de generación de energía solar y eólica, mediante unos modelos físicos de generación de energía renovable. Para obtener los datos de escala múltiple, en otras palabras, los grafos temporales, se llevan a cabo distintas simulaciones del flujo de energía bajo múltiples condiciones de carga y generación renovable, mante-

niendo fijo el estado inicial del grafo. Además, se realizan simulaciones dinámicas, en las que el estado del grafo se ve afectado por perturbaciones aleatorias, mediante una plataforma que es capaz de modelar tanto la transmisión como la distribución de la red. Éstas últimas son las que usaremos en el estudio.

A continuación vamos a introducir en detalle el proceso de generación de los datos, mediante el estudio de la plataforma de simulación, junto con el algoritmo de generación de las series temporales de grafos.

6.1.1 *Plataforma de simulación*

Diremos que la plataforma de simulación es conjunta, por su capacidad de modelado de la transmisión y la distribución (T+D). Para llevarlo a cabo, la plataforma se compone de un sistema de transmisión PSS/E con 23 buses y dos sistemas de distribución IEE, cada uno con 13 buses.

El sistema de transmisión PSS/E es un entorno de simulación ampliamente utilizado, comercializado por la empresa Siemens, para analizar el flujo de energía en redes eléctricas de gran escala. Los buses representan puntos de conexión en la red donde se intercambia energía eléctrica. Sin embargo, se han realizado algunos cambios al sistema original, que incluye 6 generadores térmicos y 7 buses de carga. En primer lugar, se han remplazado uno de los generadores térmicos por un modelo de turbina eólica para representar escenarios con alta presencia de energías renovables. Además, se han conectado dos buses de carga a los sistemas de distribución de la red mientras mantenemos el resto de los buses de carga conectados a una carga agrupada.

Por otra parte, los sistemas de distribución IEE, implementados por OpenDSS, representan redes eléctricas de menor escala, comúnmente utilizadas para la distribución de energía desde subestaciones hasta los consumidores finales, como hogares y negocios. En este contexto, los buses representan puntos de conexión donde se distribuye energía eléctrica a los consumidores. En cada red de distribución, se añadieron respectivamente modelos de paneles solares fotovoltaicos (PV) e inversores de potencia a los buses de carga, lo que representa la generación solar agregada en los tejados residenciales.

La Figura 9 muestra la arquitectura de la plataforma de simulación en detalle. A continuación, estudiaremos cómo se utiliza esta plataforma para generar las series temporales de grafos.

6.1.2 *Generación de series temporales de grafos*

Tal y como veíamos en capítulos anteriores, cuando se trata de redes eléctricas, existen multitud de perturbaciones, con distintos grados de consecuencias asociadas. Las más comunes incluyen fallos, como por ejemplo cortocircuitos entre conductores, desconexiones inesperadas de equipos y oscilaciones forzadas.

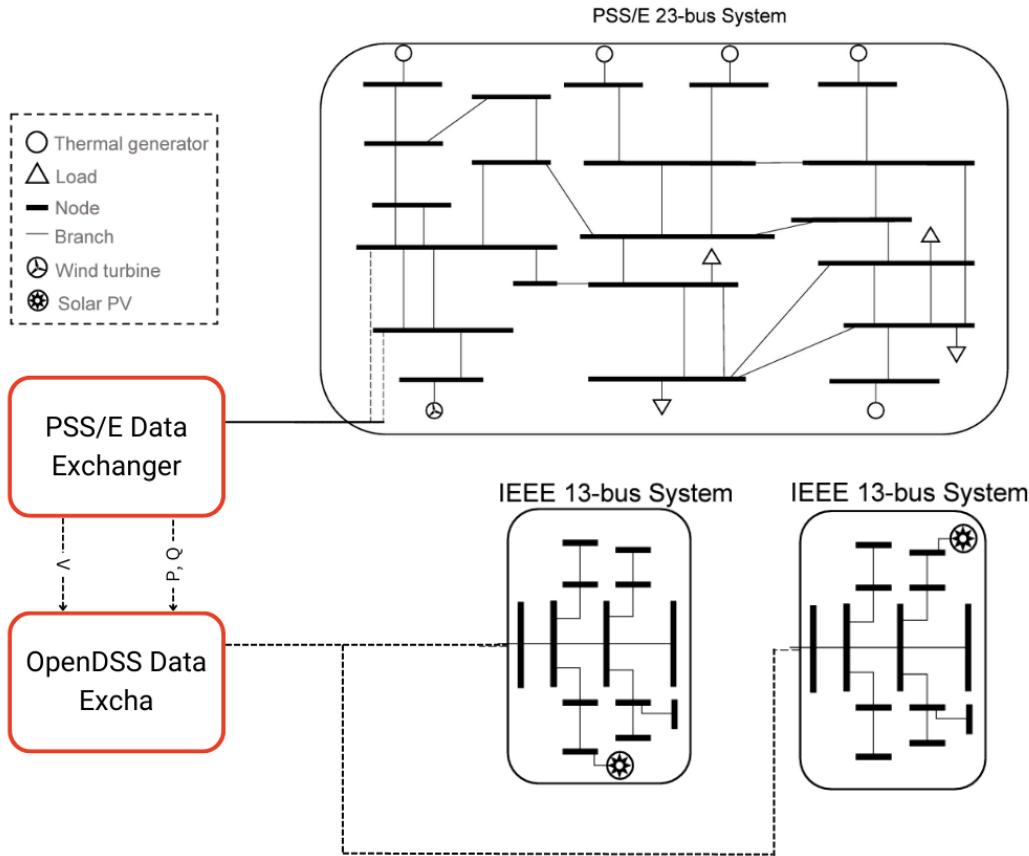


Figura 9: Arquitectura de la plataforma de simulación. Un sistema de transmisión PSS/E con 23 buses y dos sistemas de distribución IEE, cada uno con 13 buses.

Es necesario que se detecten y gestionen lo más rápidamente posible, ya que, de lo contrario, pueden causar fallos en cascada y daños incontrolables, provocando pérdidas de energía. En particular, las perturbaciones deben ser solucionadas dentro de lo que se denomina Tiempo Crítico de Despeje (CCT, por sus siglas en inglés), que suele ser del orden de 100 ms.

Antes de que ocurra una perturbación, se asume que el sistema eléctrico opera en torno a un punto de equilibrio estable. En este caso particular, el punto de equilibrio vendrá dado por el flujo de potencia en un estado estacionario, determinado por los perfiles de generación y demanda. Si no se cumple con el CCT, el sistema se volverá inestable.

Las componentes dinámicas de la red, que incluyen los generadores, turbinas, sistemas de paneles solares, y sus dispositivos de control, se establecen mediante variaciones aleatorias de valores de generación y demanda. Durante una perturbación, los estados de las componentes de la red se pueden representar mediante una serie de ecuaciones diferenciales, por lo que realizar una simulación es equivalente a resolver

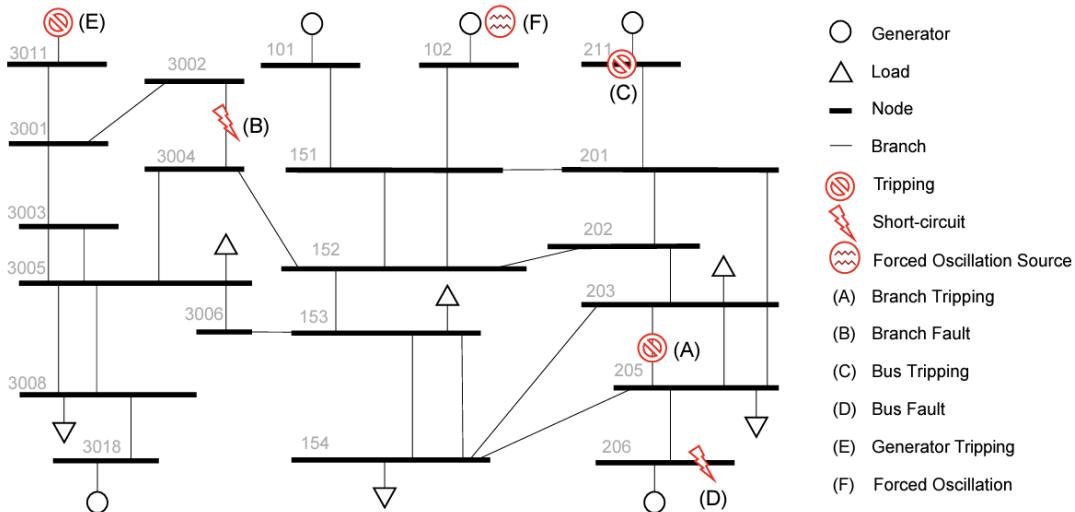


Figura 10: Perturbaciones estudiadas en la red

dichas ecuaciones, con el objetivo de obtener una serie temporal de voltajes, flujo de corrientes, entre otras variables.

El conjunto de perturbaciones que incluye el dataset está representado en la Figura 10. En particular, pueden clasificarse en cortocircuito o fallo de las distintas componentes (generador, bus o rama).

El algoritmo de obtención de los datos realiza pasos iterativos para asegurar que los datos obtenidos de distintos sistemas y simuladores están correlados entre sí. De esta manera, en cada punto de tiempo de la simulación, se intercambian los valores de voltaje y potencia entre el modelo de transmisión y los modelos de distribución. Además, también se incorporan los modelos dinámicos de paneles solares, implementados y ejecutados por separado. El detalle puede consultarse en el Algoritmo 1.

El resultado de ejecutar este código sobre los datos son 550 simulaciones, cada una de ellas clasificada como un tipo de perturbación distinto (error de rama, cortocircuito de rama, error de nodo, cortocircuito de nodo, cortocircuito de generador). Cada simulación está representada por medio de una serie temporal de grafos, con 960 observaciones, obtenidas a 240Hz, que preprocesaremos a intervalos iguales, obteniendo finalmente 800 observaciones. Cada serie incluye información de voltaje en los buses o nodos, y medidas de corriente y carga a través del sistema de transmisión, para cada punto temporal, conformando así la serie temporal de grafos.

La representación del grafo del sistema de transmisión en la primera simulación, en el primer instante temporal, se puede estudiar en la Figura 11. Es importante notar, que además de los voltajes de cada nodo, disponemos de información sobre la potencia activa y reactiva de cada enlace, que no incluimos en el gráfico por simplicidad.

Además, en la Figura 12 podemos ver un ejemplo de un fallo en un nodo, en particular en el nodo 205, junto con las consecuencias para los nodos vecinos.

Algorithm 1 Simulación de Eventos Transitorios Conjunta T+D

Seleccionar hora de los perfiles de viento, solar y carga
 Resolver el flujo de potencia en estado estacionario como punto de partida
 Crear una perturbación con tiempo de inicio aleatorio T_{on} , tiempo de despeje T_c y parámetros
 Crear objetos de modelo para todos los generadores PV e inicializar con el perfil solar
 Inicializar el proceso de simulación transitoria en PSS/E usando la solución en estado estacionario
 Inicializar el contador de tiempo $t = 0$
Mientras $t < T_{\max}$ **step**
 Si $t == T_{on}$ o $t == T_c$ **entonces** agregar/despejar perturbación
for d en la lista de circuitos de distribución **do**
 Leer archivos de casos del sistema de distribución en OpenDSS
 Seleccionar hora de los perfiles de carga y establecer cargas netas de bus
 Establecer voltaje de fuente al voltaje de nodo V_t^d en transmisión donde se encuentra el circuito de distribución
 Resolver modelos dinámicos de PV y obtener P/Q netos de los generadores PV
 Resolver el flujo de potencia y calcular P y Q totales de todas las 3 fases
 En transmisión, ajustar P_t^d y Q_t^d para el bus d que tiene el extremo del circuito de distribución
end for
 Avanzar la simulación transitoria del sistema de transmisión en OpenDSS
 Fin
 Recoger las mediciones de series temporales y almacenarlas en el archivo de salida

De forma similar, el impacto de la caída de una rama se ilustra claramente en la Figura 13. Es evidente cómo un fallo en una parte de la red afecta inmediatamente a las áreas adyacentes, destacando una variabilidad significativa en la forma de las series temporales entre distintas simulaciones. Exploraremos más a fondo esta característica en secciones posteriores.

Es importante notar que aunque el dataset se ha obtenido del artículo ya mencionado, ha sido necesario preprocesar los datos, y transformarlos en una estructura de datos apta para los algoritmos de predicción que veremos más adelante. El código correspondiente a la adaptación del dataset se encuentra en el repositorio público en GitHub, accesible en https://github.com/maaguado/GNNs_PowerGraph.

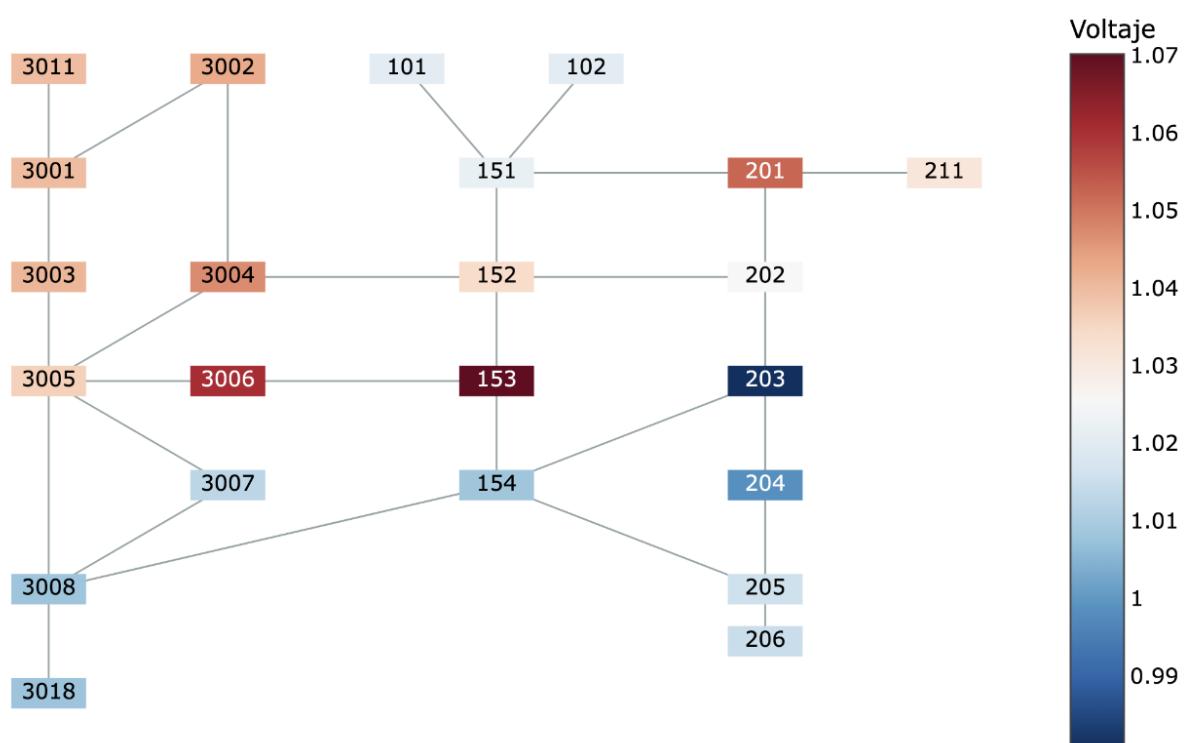


Figura 11: Primer instante temporal de la primera simulación

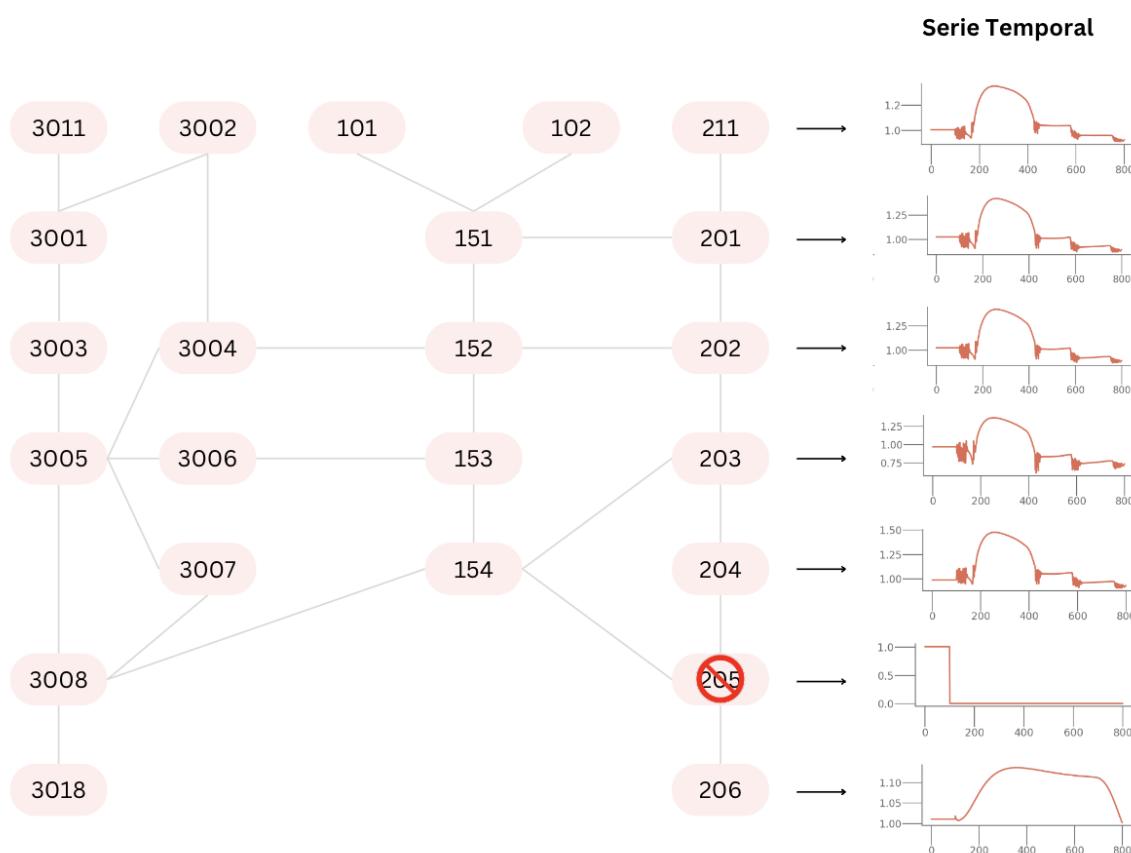


Figura 12: Evolución de las series temporales con un error en un nodo

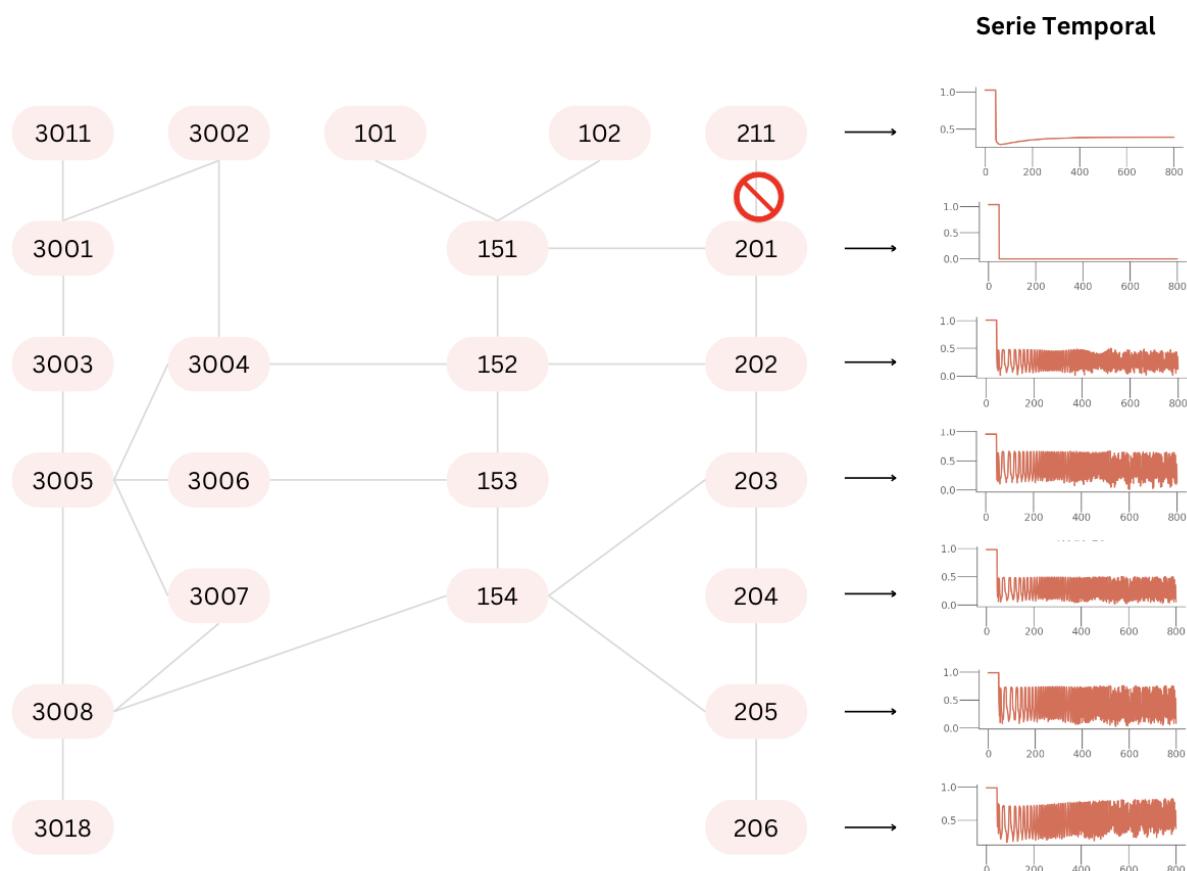


Figura 13: Evolución de las series temporales con un error en una rama

6.2 DESCRIPCIÓN DEL PROBLEMA

A continuación vamos a describir en detalle los dos problemas principales en los que se centrará este estudio.

En primer lugar, debido a la naturaleza temporal de los datos, resulta natural plantear un problema de predicción de series temporales, en el que el objetivo es predecir el estado de los 23 nodos del sistema de transmisión a lo largo de una ventana de t instantes de tiempo. La mayoría de técnicas de predicción con grafos temporales, se basan en una única serie temporal, de la que se obtienen las entradas y salidas mediante una ventana deslizante que recorre la serie completa. Sin embargo, en este caso, tenemos 550 simulaciones, cada una con una serie temporal, con una forma radicalmente distinta. La variabilidad en las series se puede observar en la Figura 14, donde hemos representado el voltaje del primer nodo de la red, en los cinco tipos de perturbaciones distintas, y dentro de cada tipo, hemos representado las quince primeras simulaciones.

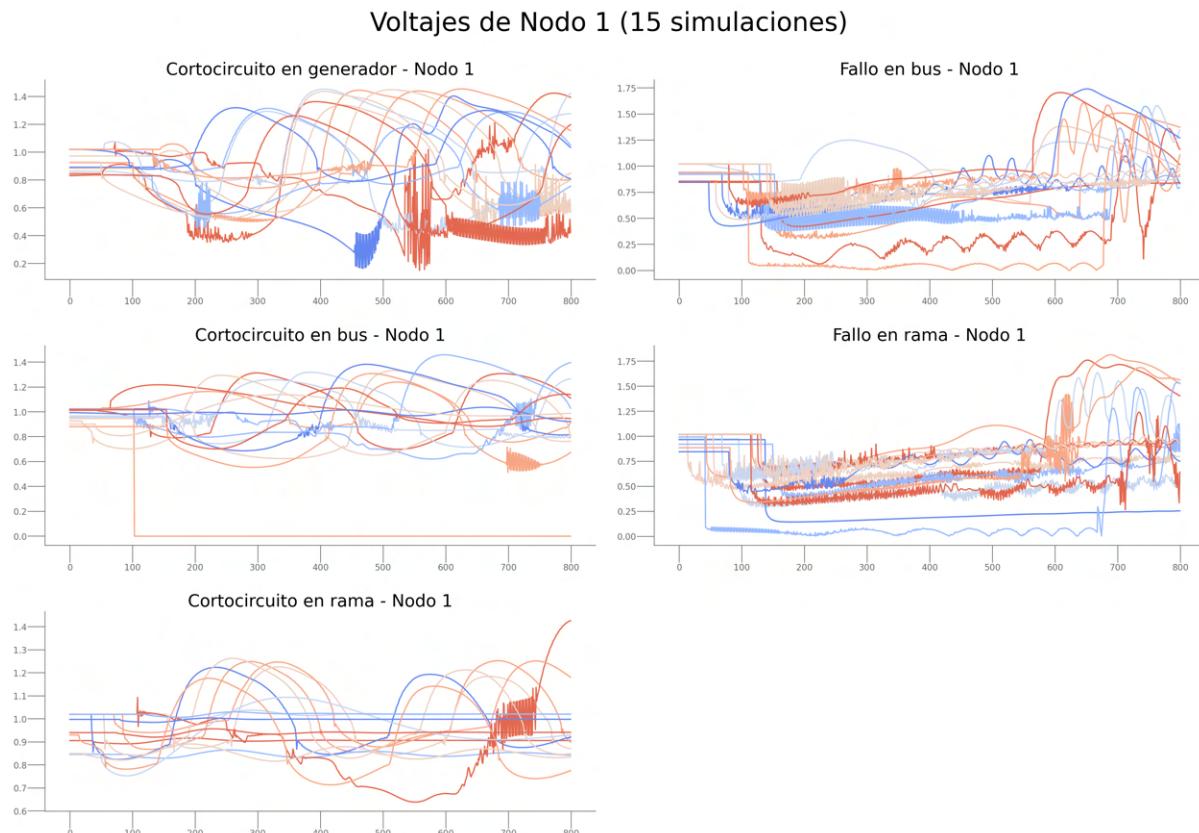


Figura 14: Nodo 1 a lo largo de quince simulaciones para cada tipo de perturbación

Además, para poder representar la variabilidad del voltaje de los nodos en las distintas simulaciones, vamos a estudiar las propiedades estadísticas de las series. En primer lugar, veremos la media y la desviación típica del primer nodo a lo largo del tiempo en las distintas simulaciones (Figura 15).

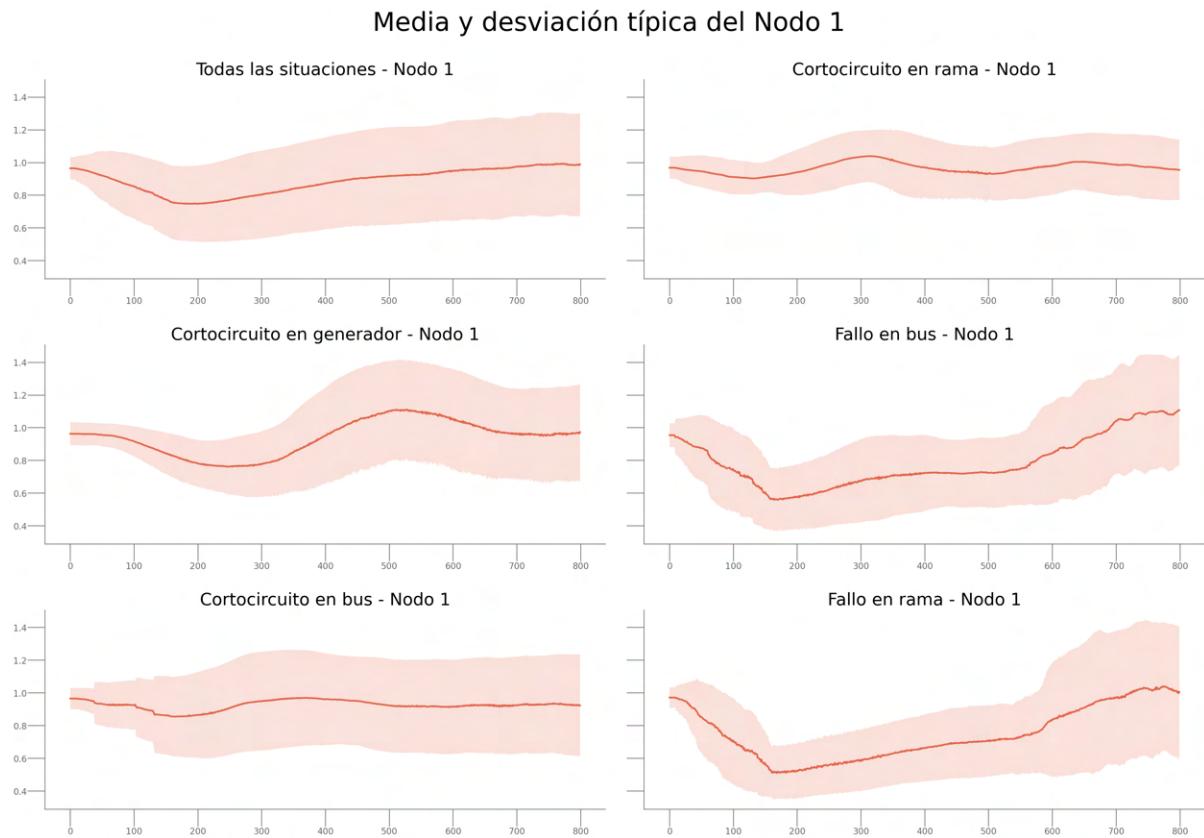


Figura 15: Media y desviación típica del Nodo 1 a lo largo del tiempo

Se puede observar como la desviación típica en las perturbaciones derivadas de cortocircuitos se mantiene más estable, especialmente en los cortocircuitos de rama. Por otra parte, en los fallos de bus o rama, ésta aumenta ligeramente al final de la serie, pues es cuando se dan los fallos. Sin embargo, hay mucha diferencia en la forma de las mismas, por lo que estas últimas perturbaciones serán más complicadas de modelar.

Por último vamos a estudiar la variabilidad intra-nodo para cada una de las simulaciones (Figura 16), considerando la media de toda la serie temporal para cada nodo. Podemos determinar que al analizar las simulaciones por tipo de perturbación, encontramos mayor consistencia en el voltaje entre los distintos nodos para las simulaciones correspondientes a los cortocircuitos de generador y de rama.

Con el análisis anterior podemos concluir que la diversidad presente en el dataset complica bastante el tratamiento y modelado de los datos, y nos deja con tres alternativas posibles:

- La primera opción consiste en que cada simulación esté asociada a una única serie temporal, lo que resultaría en 550 entradas en el conjunto. Aunque esta parece ser la decisión más intuitiva, presenta varios problemas. En primer lugar, la diversidad de las simulaciones impide que los modelos converjan, produciendo predicciones demasiado genéricas para ser útiles. Además, es crucial determi-

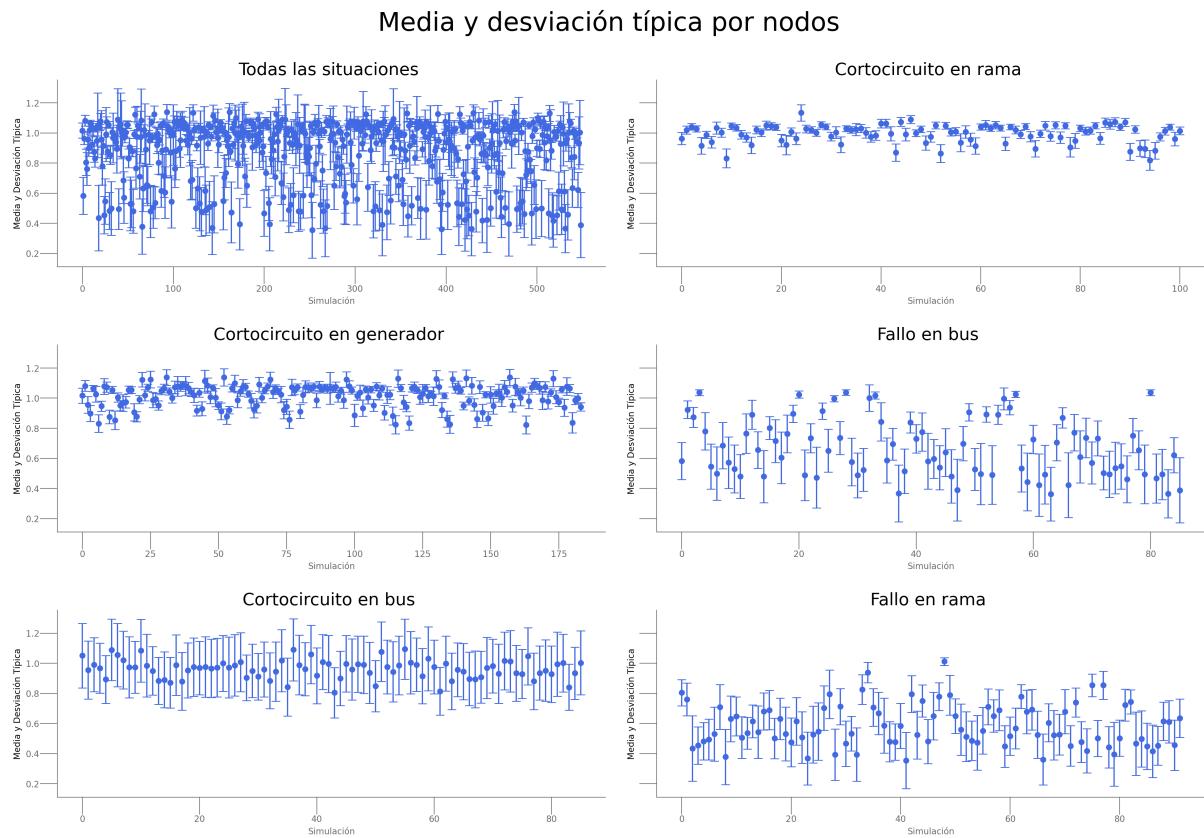


Figura 16: Media y desviación típica entre nodos

nar adecuadamente el número de puntos temporales de entrada y salida. Un número demasiado grande de puntos de entrada puede llevar a un sobreajuste del modelo, mientras que un número demasiado grande de puntos de salida puede reducir la calidad de las predicciones. Con esto en mente, todos los datos que no entren en la ventana temporal de entrada-salida se perderían, lo cual no es ideal. Además, para cada tipo de perturbación, el error se da en un instante diferente, lo que complica aún más la determinación de las ventanas de entrada y salida.

- La segunda opción soluciona el segundo problema planteado anteriormente, y es utilizar una ventana deslizante, a lo largo de todas las simulaciones, para evitar que se pierdan los datos. Sin embargo, esta opción mantiene la problemática de la diversidad de simulaciones, por lo que tampoco resulta adecuada.
- La última alternativa pasa por separar el dataset por tipos de perturbación, realizando los experimentos de manera separada en cada uno de ellos. Esto convierte el problema original de predicción de series temporales de grafos, en cinco problemas independientes, en los que se trata de establecer las consecuencias de cada uno de los tipos de perturbaciones en la red. Esta es la alternativa con la que se procederá en este trabajo.

Además, dado que cada una de las 550 simulaciones tiene su correspondiente etiqueta, es natural plantear un problema de clasificación de series temporales de grafos,

cuyo objetivo es identificar el origen de la perturbación. En este caso, el procesamiento de las redes es directo y no requiere tomar decisiones complejas como las mencionadas anteriormente. En la práctica, clasificar perturbaciones recurrentes puede ayudar a predecir y prevenir fallos futuros, pues si se identifica un patrón en las perturbaciones, se pueden tomar medidas preventivas en los componentes críticos antes de que se produzca un fallo.

6.3 DESARROLLO DEL SOFTWARE

Como parte de este trabajo se ha desarrollado un software modular orientado a la implementación y entrenamiento de modelos basados en arquitecturas de grafos temporales. Este software se ha diseñado con el objetivo de permitir la integración sencilla de distintas arquitecturas dentro de un marco común (Figura 17). Este software puede encontrarse en el repositorio público del proyecto: https://github.com/maaguado/GNNs_PowerGraph.

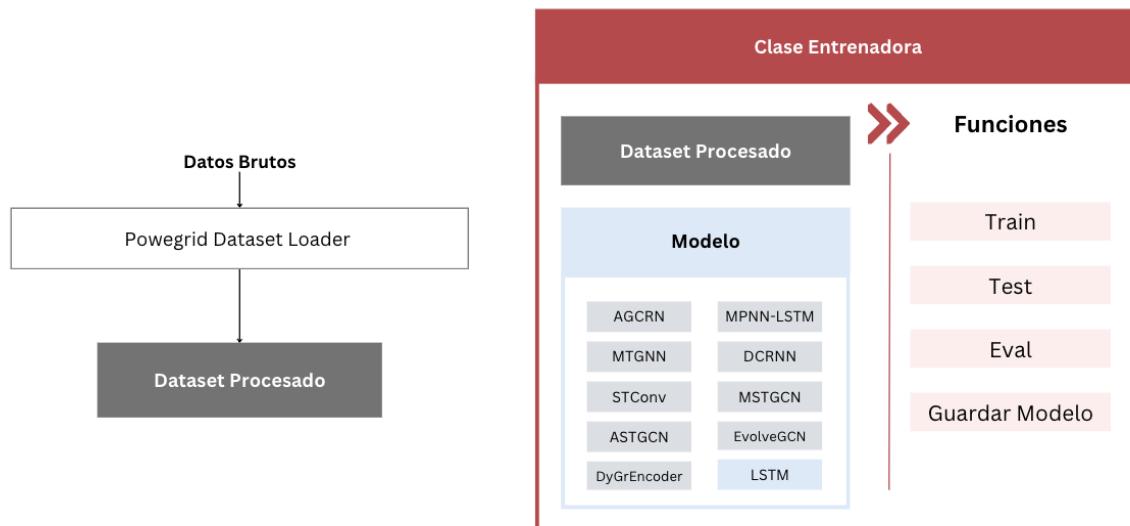


Figura 17: Estructura modular

6.3.1 Estructura modular

El desarrollo se ha llevado a cabo de manera modular, donde cada modelo de red neuronal se implementa como una clase separada dentro del módulo `models.py`. Los modelos se han construido siguiendo una estructura similar: en el constructor se definen las capas y parámetros específicos del modelo, mientras que en el método `forward`, se describe el flujo de datos a través de la red, admitiendo un problema tanto de regresión como de clasificación, con la adaptación necesaria en cada caso. Esta

uniformidad en la implementación de los modelos permite que todos ellos puedan ser gestionados de manera consistente por una clase entrenadora común.

6.3.2 Clase de Entrenamiento Centralizada

El corazón del sistema es una clase de entrenamiento ubicada en `trainer.py`, que se encarga de manejar todo el ciclo de vida de los modelos, desde la carga de datos, entrenamiento, validación, hasta la evaluación y exportación de resultados. Esta clase ha sido diseñada para ser flexible y extensible, de manera que, para cada modelo implementado, se crea una subclase de la misma, y que se encargará de las particularidades de cada una de las arquitecturas, sin prestar atención al bucle de entrenamiento y validación usual.

La clase entrenadora se encarga de las siguientes tareas:

- **División del conjunto de datos:** Mediante funciones auxiliares, se permite la partición del conjunto de datos en subconjuntos de entrenamiento, validación y prueba de manera adecuada al tipo de problema.
- **Entrenamiento del modelo:** Utiliza optimizadores como Adam y otros parámetros, como por ejemplo, *el learning rate*, para llevar a cabo las iteraciones de entrenamiento.
- **Evaluación del rendimiento:** Incluye múltiples métricas de evaluación, en función del problema que esté resolviendo, como por ejemplo *accuracy*, *precision*, *recall*, *F1-score*, entre otras, para obtener una visión completa del rendimiento del modelo.
- **Gestión de la ejecución:** Permite la carga y ejecución de distintos modelos sin cambios significativos en la lógica de entrenamiento, favoreciendo la reutilización del código.

Este enfoque centralizado garantiza la consistencia de los experimentos realizados, permitiendo que podamos comparar los resultados obtenidos en cada uno de ellos.

6.3.3 Cargador de Datos

El software también incluye un cargador de datos especializado, `PowerGridDatasetLoader`, diseñado para manejar conjuntos de datos relacionados con redes eléctricas. Esta clase se encarga de cargar y preprocessar los datos originales, obtenidos de [46], de cara a poder utilizarlos en tareas de regresión o clasificación. Entre las funcionalidades que ofrece, se incluyen:

- **Carga de datos de nodos y aristas:** Permite cargar datos sobre los voltajes de los nodos, atributos de aristas e índices de nodos, esenciales para la construcción del grafo temporal, pues contienen la información sobre la interacción entre elementos de la red.

- **Preprocesamiento para distintas tareas:** Adapta los datos según el tipo de problema a resolver, sea regresión o clasificación, asegurando que los datos estén en el formato adecuado para el entrenamiento de los modelos.
- **Transformaciones y normalización:** Facilita la transformación de identificadores de nodos y la normalización de las características, preparando los datos para su uso en modelos de aprendizaje automático.

Podemos dar por finalizada la introducción de los datos utilizados en este trabajo, y del software implementado. A continuación procederemos a presentar en qué consiste cada una de las arquitecturas, profundizando aún más en los conceptos introducidos en el marco teórico del trabajo.

ALGORITMOS BASADOS EN MECANISMOS RECURRENTES

En este capítulo, exploraremos en detalle las arquitecturas pioneras de las Redes Neuronales de Grafos (GNNs) temporales utilizadas en nuestros experimentos. Continuaremos con la segmentación propuesta en capítulos anteriores y comenzaremos analizando aquellos algoritmos que incorporan mecanismos de redes neuronales recurrentes. Posteriormente, en el capítulo siguiente, presentaremos las arquitecturas basadas en mecanismos de atención, completando así nuestra revisión de las principales arquitecturas en este campo.

Las arquitecturas presentadas son una selección de todas las arquitecturas disponibles para resolver este problema. Para poder llevarla a cabo, nos hemos basado en una revisión de los mismos [8], y seleccionamos los que tenían implementaciones en la librería Pytorch Geometric Temporal.

Es importante notar que, aunque las implementaciones originales forman de la librería, ha sido necesario adaptar la mayoría de los modelos para que pudieran admitir grafos dinámicos con evolución de etiquetado de los nodos y las aristas. Aún con esa adaptación, algunos de los modelos introducidos no admiten como entrada la secuencia de valores de las aristas.

7.1 AGCRN

La primera arquitectura que vamos a introducir es la de las Redes Neuronales Recurrentes Convolutivas sobre Grafos (AGCRN, por sus siglas en inglés) [6].

La arquitectura AGCRN representa un enfoque novedoso para la predicción de series temporales utilizando redes neuronales convolucionales sobre grafos basadas en RNN. Este modelo no admite como entrada información sobre la evolución de la matriz de adyacencia y valores de las aristas, sino que está diseñado para capturar correlaciones espaciales y temporales complejas de manera automática.

Consta principalmente de dos módulos adaptativos que mejoran la capacidad de las redes convolucionales en grafos (GCN) tradicionales: el *Node Adaptive Parameter Learning* (NAPL) y *Data Adaptive Graph Generation* (DAGG), que estudiaremos en detalle más adelante.

Además, la arquitectura AGCRN integra estos módulos con una red recurrente basada en unidades de GRU (Gated Recurrent Units) para modelar las dependencias temporales. Esta combinación permite al modelo aprender no solo patrones temporales a largo plazo, sino también adaptarse dinámicamente a las variaciones espaciales sin la necesidad de un grafo predefinido. Puede estudiarse en detalle en la Figura 18.

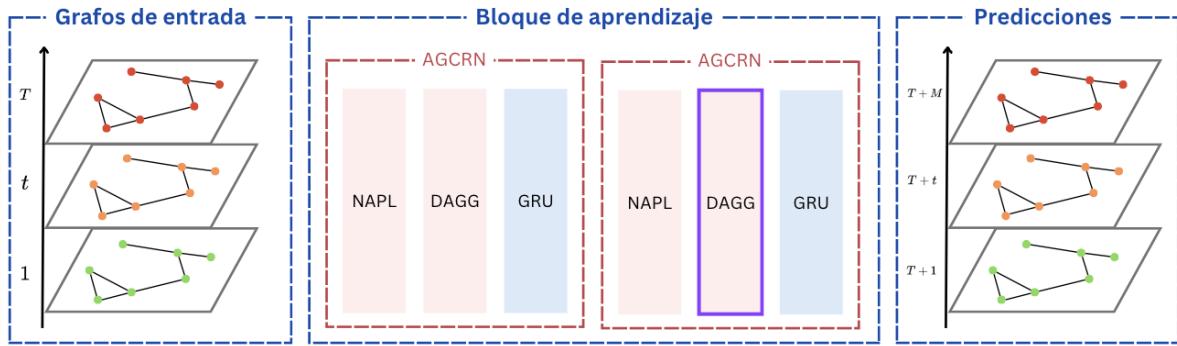


Figura 18: Arquitectura AGCRN

Este modelo está originalmente diseñado para realizar predicciones en series temporales multivariadas, en las que existe cierta correlación o interacción entre las distintas series. Sin embargo, en este estudio lo aplicaremos específicamente a la predicción de voltajes en los nodos de una red eléctrica, sin utilizar la información estructural de los grafos.

Formalmente, la ecuación de la arquitectura AGCRN es:

$$h_t = \text{GRU}(\text{NAPL-GCN}(X_t, A_t), h_{t-1}) \quad (7.1.1)$$

donde X_t son los datos de entrada en el tiempo t , A_t es la matriz de adyacencia generada por DAGG, y h_{t-1} es el estado oculto de la unidad GRU del paso de tiempo anterior.

Aunque el modelo está diseñado para realizar predicciones sobre series temporales multivariadas, donde hay cierta correlación o interacción entre series, en nuestro caso lo utilizaremos para la predicción de los voltajes de los nodos.

7.1.1 Módulos GCN avanzados

Vamos a estudiar de una manera más profunda en qué consisten los dos módulos que expanden la capacidad de las GCN. En primer lugar, vemos una descripción general:

1. **NAPL:** Este módulo se encarga de la personalización de los parámetros de la red a nivel de nodo, permitiendo que el modelo aprenda patrones específicos para cada nodo. Utiliza una factorización de parámetros que genera parámetros específicos del nodo a partir de un conjunto común de pesos y sesgos.

2. **DAGG:** Se encarga de la generación dinámica del grafo durante el entrenamiento, es decir, del aprendizaje de las dependencias entre los nodos. Este módulo genera una nueva matriz de adyacencia que refleja las correlaciones espaciales reales y cambiantes entre las series de tráfico.

El módulo NAPL, a diferencia de las GCN tradicionales, que usan parámetros globales, adapta los parámetros a nivel individual de cada nodo. Esto se logra mediante una factorización de los parámetros del modelo en dos componentes principales: una matriz de incrustación de nodos, E , y un conjunto compartido de pesos, W .

Cada nodo i tiene una incrustación única E_i que interactúa con el conjunto de pesos W para generar los pesos específicos del nodo:

$$\Theta_i = f(E_i, W) \quad (7.1.2)$$

donde Θ_i son los parámetros específicos del nodo i y f es una función que combina las incrustaciones y los pesos. Esta personalización permite que el modelo se adapte mejor a las características locales y temporales específicas de cada nodo.

El módulo DAGG, por otro lado, es responsable de adaptar dinámicamente la topología del grafo basada en los datos observados. En lugar de utilizar una matriz de adyacencia predefinida, se genera una matriz de adyacencia A en cada paso de tiempo utilizando las incrustaciones de nodos aprendidas, calculada como sigue:

$$A = \text{softmax}(\text{ReLU}(EE^T)) \quad (7.1.3)$$

Esta ecuación refleja cómo las relaciones entre nodos son inferidas y ajustadas continuamente, permitiendo que el modelo responda a cambios en las dinámicas de las series.

7.1.2 Predicciones de múltiples pasos

La capacidad de realizar predicciones de múltiples pasos es crucial para la planificación y gestión efectiva del tráfico. AGCRN plantea un apilamiento de varias capas con forma de codificador, con el objetivo de capturar los patrones espaciotemporales específicos de cada nodo. Se representa la entrada como $H \in \mathbb{R}^{N \times d_o}$.

Después, se puede obtener directamente la predicción de voltaje para los próximos τ pasos de todos los nodos aplicando una transformación lineal para proyectar la representación de $\mathbb{R}^{N \times d_o}$ a $\mathbb{R}^{N \times \tau}$. Esta transformación lineal puede ser, por ejemplo, una capa completamente conectada.

7.2 DYGRENCODER

La siguiente arquitectura que vamos a ver es DyGrEncoder [35], que emplea el aprendizaje no supervisado para representar grafos dinámicos que evolucionan en el tiempo, adaptándose así a cambios en la topología. Está compuesta por un codificador recurrente que proyecta la historia temporal de los grafos, denotada por H_{G_t} , en un espacio de dimensión k .

El proceso comienza con la aplicación de Redes Neuronales de Grafos con Puertas (GGNNs, por sus siglas en inglés) para capturar la topología del grafo en cada intervalo de tiempo t . Posteriormente, se utiliza el codificador para integrar tanto la historia H_{G_t} como la representación topológica recién obtenida. Esto permite que el modelo maneje eficientemente la evolución dinámica del grafo a lo largo del tiempo.

El último paso, es reconstruir el grafo en el instante t a partir de la representación obtenida, mediante un decodificador.

Comenzamos estudiando en qué consisten las redes neuronales de grafos con puertas.

7.2.1 Redes GGNN

Las redes GGNN son una extensión de las GNN, diseñadas para mejorar el aprendizaje del alcance entre distintos nodos del grafo. Se basan en la incorporación de bloques GRU para propagar un mensaje a través del vecindario de un nodo concreto. Veamos el modelo de propagación más en detalle a continuación.

Supongamos que A es la matriz de adyacencia, y sea $x_v(0)$ la representación inicial del nodo $v \in V$. Sea también $h_v(i)$ el estado oculto del nodo v en la iteración i .

Comenzamos inicializando el estado oculto de los nodos $h_v(0) = x_v \quad \forall v \in V$. En cada paso de la propagación se recopila información del vecindario de un nodo para obtener su representación. Después de múltiples iteraciones de este paso, la información se habrá transmitido a través de todos los nodos alcanzables para generar la representación de cada nodo.

$$x_v(i) = A_v[h_1(i-1), \dots, h_{|V|}(i-1)] + b \quad (7.2.1)$$

Una vez se ha obtenido la representación $x_v(i)$, se pasa a través de un bloque GRU al uso, tal y como vimos en capítulos anteriores.

7.2.2 Descripción detallada de la arquitectura

Tal y como ya hemos visto, la red GGNN construye una representación de la estructura topológica del grafo G_t , a través de M pasos de propagación de mensajes en G_t , con el mecanismo que acabamos de introducir.

Después, se calcula la media de los estados ocultos de los nodos. Nos referimos a esto como una representación estática del grafo ($\text{Emb}_{st}(G_t)$):

$$\text{Emb}_{st}(G_t) = \text{Avg}\{h_v(M) : \forall v \in V\} \quad (7.2.2)$$

A partir de la representación estática del grafo $\text{Emb}_{st}(G_t)$ y el histórico H_{G_t} , se utiliza un codificador LSTM, que proyectará el grafo G_t en una representación oculta. El codificador LSTM, denominado LSTM_{enc} , transmite la información del grafo dinámico G a lo largo de una ventana de observación de tamaño w y calcula la representación dinámica de G_t usando el conocimiento sobre la topología del grafo de los w pasos de tiempo pasados:

$$h_{\text{enc}}^t = \text{LSTM}_{enc}(\text{Emb}_{st}(G_t), h_{\text{enc}}^{t-1}) \quad (7.2.3)$$

Donde h_{enc}^t será la representación dinámica del grafo, notado por $\text{Emb}_{dy}(G_t)$.

Por último, pasamos al decodificador, LSTM_{dec} , cuyo principal objetivo es reconstruir la historia del grafo H_{G_t} en la ventana de tamaño w .

El decodificador actúa como un modelo autorregresivo y predice la topología del grafo en el instante t dado los grafos recientemente predichos en pasos de tiempo anteriores. Utiliza h_{enc}^T para inicializar su primer estado oculto.

Finalmente, incorporamos una capa lineal, debido a que el objetivo es poder realizar predicciones del estado del grafo en un tamaño de ventana determinado.

El detalle de la arquitectura en el caso del problema de predicción de enlaces se puede estudiar en la Figura 19.

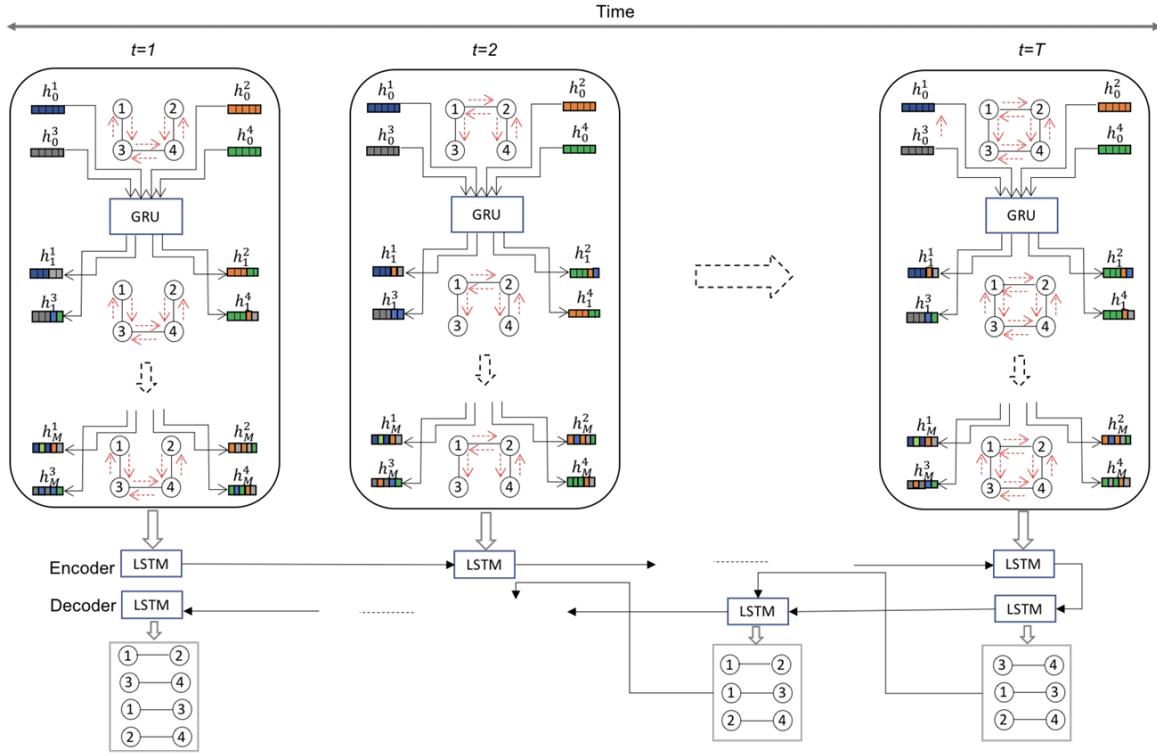


Figura 19: Arquitectura DyGrEncoder

7.3 MPNN-LSTM

El algoritmo *MPNN-LSTM* [30] representa una arquitectura avanzada que combina las Redes Neuronales de Paso de Mensajes (MPNNs) con LSTM. Esta combinación se destaca por su capacidad para capturar tanto la información espacial como temporal, lo que es crucial para la tarea para la que se creó, que es la predicción de la propagación de COVID-19 a partir de datos de movilidad y casos históricos.

La arquitectura en detalle se puede estudiar en la Figura 20.

7.3.1 MPNNs

Las MPNNs utilizan un mecanismo de paso de mensajes donde la representación de cada nodo se actualiza basándose en los mensajes recibidos de sus vecinos, permitiendo que el modelo capture relaciones y dependencias complejas dentro de la estructura del grafo. Cada iteración de paso de mensajes busca refinar la representación de los nodos integrando información de sus conexiones directas.

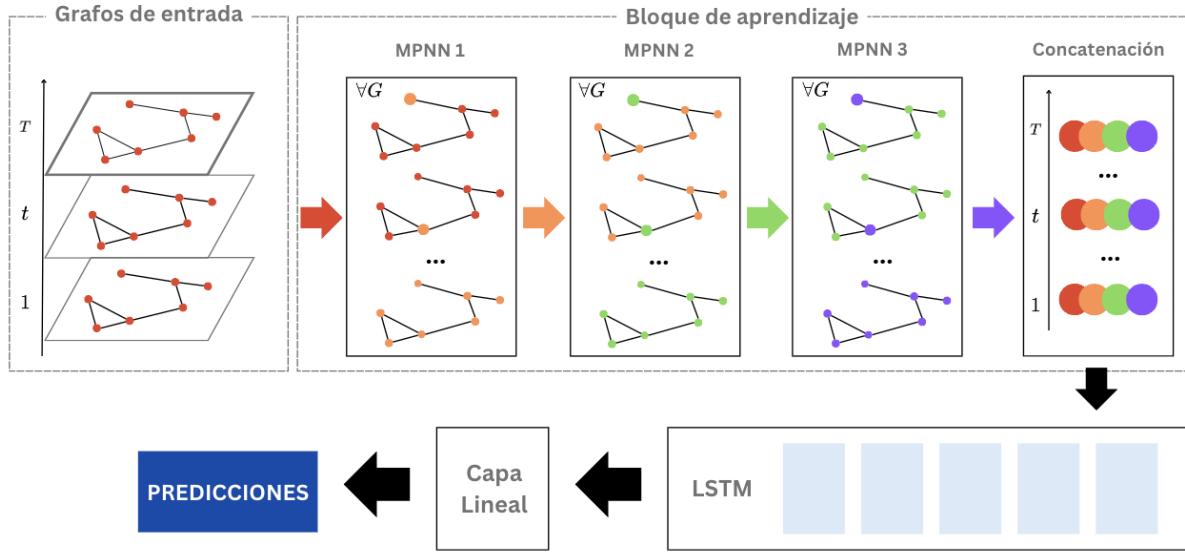


Figura 20: Arquitectura MPNN-LSTM

Formalmente, la agregación de los mensajes recibidos del vecindario se realiza mediante la siguiente expresión

$$H_i = f(\tilde{A}H_{i-1}W_i) \quad (7.3.1)$$

donde H_i es una matriz que contiene las representaciones de los nodos de la capa anterior, con $H_0 = X$, W_i es la matriz de parámetros entrenables de la capa i , y f es una función de activación no lineal como ReLU. Además, se normalizará la matriz de adyacencia A de tal manera que la suma de los pesos de las aristas entrantes de cada nodo sea igual a 1.

Es importante mencionar que, por simplicidad en la notación, se ha omitido el índice de tiempo, no obstante, el modelo mencionado se aplica a todos los grafos de la secuencia de entrada $G(1), \dots, G(T)$ por separado.

Dado un modelo con K capas de agregación de vecindario, las matrices \tilde{A} y H_0, \dots, H_K son específicas para un solo grafo, mientras que las matrices de pesos W_1, \dots, W_K se comparten entre todos los grafos.

A medida que el número de capas de agregación de vecindario aumenta, las características finales de los nodos capturan más y más información global. Sin embargo, también puede ser útil retener información local e intermedia, y para ello se concatenarán las matrices $H_0, H_1, H_2, \dots, H_K$ horizontalmente, es decir,

$$H = \text{CONCAT}(H_0, H_1, H_2, \dots, H_K)$$

Por lo tanto, las filas de la matriz H se podrán considerar como representaciones de los nodos que contienen información estructural a múltiples escalas.

7.3.2 Integración de LSTMs

Tal y como veíamos en capítulos anteriores, es usual incorporar una capa RNN después de obtener las representaciones de la secuencia de grafos. En este caso, se utilizará un bloque formado por dos capas LSTM.

Los estados ocultos del bloque LSTM capturarán la dinámica de propagación basada en la información codificada en las representaciones de los nodos. Las nuevas representaciones de los nodos corresponderán al estado oculto del último instante temporal de la segunda capa LSTM. Estas representaciones pasarán después por una capa de salida similar al MPNN, junto con las características iniciales para cada paso de tiempo.

7.4 EVOLVEGCN

El siguiente modelo que vamos a estudiar se denomina EvolveGCN [31], y busca modelar grafos dinámicos mediante la evolución de los parámetros de una red convolucional de grafos (GCN), por medio de un modelo recurrente.

La arquitectura incluye una red GCN usual con L capas para cada instante de tiempo t . Cada red GCN_l está conectada con el siguiente instante temporal mediante una estructura RNN, que puede ser LSTM o GRU, formando lo que denominamos una **unidad de convolución evolutiva** (EGCU, por sus siglas en inglés). El objetivo del bloque recurrente es actualizar la matriz de pesos $W_t^{(l)}$ en el tiempo t , tal y como veremos a continuación.

7.4.1 Evolución de Pesos

El mecanismo de evolución de pesos es la característica principal de EvolveGCN, pues se encarga de actualizar la matriz de pesos basándose en información actual e histórica. Por este motivo, resulta natural incorporar una arquitectura recurrente como las que se han estudiado en capítulos anteriores. Para ello, se pueden dar dos alternativas.

La primera opción, denominada opción H , es tratar la matriz $W_t^{(l)}$ como un estado oculto del sistema dinámico. Usamos un bloque GRU para actualizarlo con la entrada en el tiempo t al sistema, que serán las representaciones de los nodos $H_t^{(l)}$ obtenidas desde la red GCN_l .

De manera abstracta, podemos ver la actualización de pesos como:

$$\underbrace{W_t^{(l)}}_{\substack{\text{pesos GCN} \\ \text{estado oculto}}} = \text{GRU}\left(\underbrace{H_t^{(l)}}_{\substack{\text{representaciones de nodo} \\ \text{entrada}}}, \underbrace{W_{t-1}^{(l)}}_{\substack{\text{pesos GCN} \\ \text{estado oculto}}}\right),$$

El bloque GRU puede ser reemplazada por otras arquitecturas recurrentes, siempre que los roles de $W_t^{(l)}$, $H_t^{(l)}$, y $W_{t-1}^{(l)}$ estén claros.

La segunda opción, denominada opción O , es tratar $W_t^{(l)}$ como salida del sistema dinámico, y que por tanto, se convierte en entrada del siguiente instante temporal. Usamos un bloque LSTM para modelar esta relación de entrada-salida, pues este tipo de bloque recurrente mantiene información del sistema mediante un contexto de celda, que actúa como el estado oculto de una GRU.

En esta versión, las representaciones de los nodos no se usan en absoluto. De manera abstracta, podemos escribir el proceso como:

$$\underbrace{W_t^{(l)}}_{\substack{\text{pesos GCN} \\ \text{output}}} = \text{LSTM}\left(\underbrace{W_{t-1}^{(l)}}_{\substack{\text{pesos GCN} \\ \text{input}}}\right),$$

Podemos estudiar la diferencia entre ambas opciones en la Figura 21.

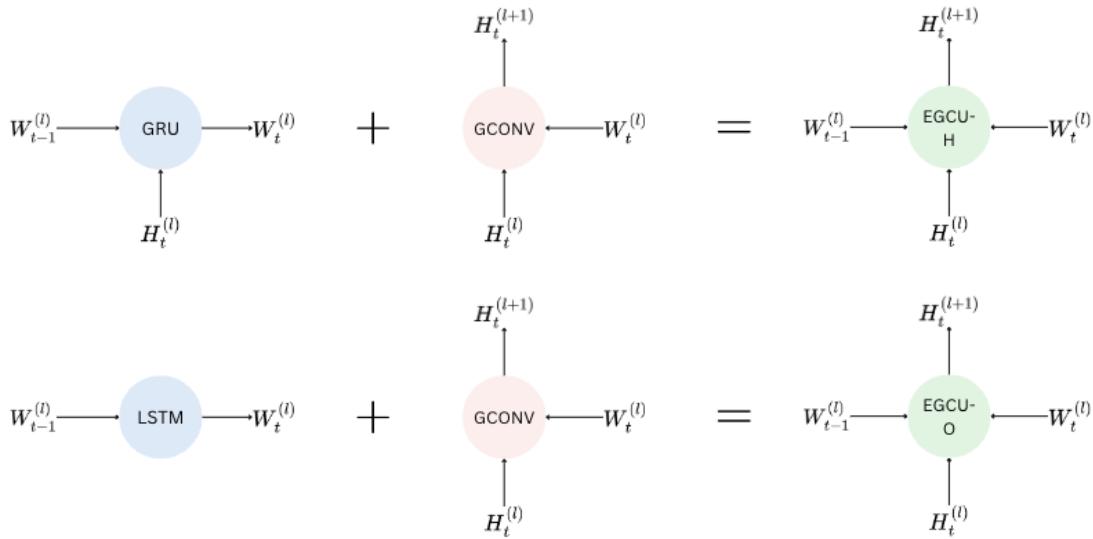


Figura 21: Bloques EGCU

Apilando las unidades de abajo hacia arriba, obtenemos una GCN con múltiples capas para cada paso de tiempo. Luego, desplegando horizontalmente, las unidades forman una red en la cual fluye la información $H^{(l)}$ y $W_t^{(l)}$. Llamamos a este modelo general red convolucional de grafos en evolución (EvolveGCN).

7.5 DCRNN

El siguiente algoritmo que vamos a estudiar se llama Red Neuronal Convolucional de Difusión (DCRNN, por sus siglas en inglés) [27], y su particularidad principal es la incorporación de un proceso de difusión en el grafo para modelar las dependencias espaciales. Además, también se modelan las dependencias temporales, mediante el uso de redes recurrentes.

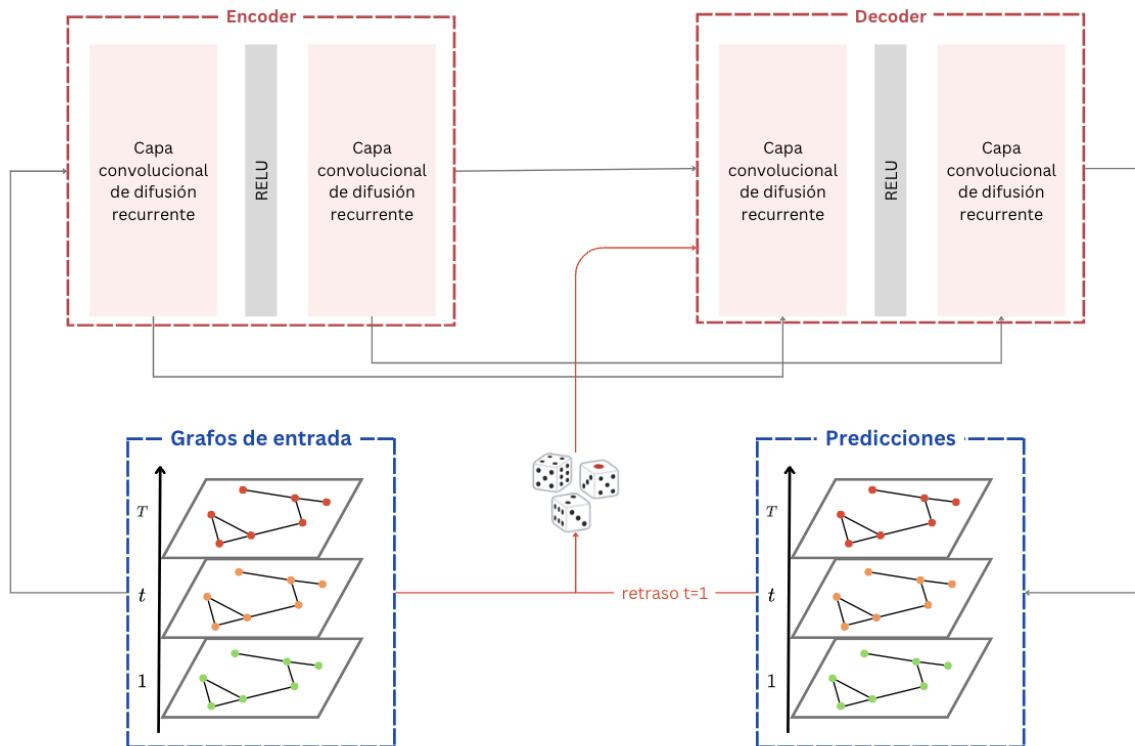


Figura 22: Arquitectura DCRNN con dos capas convolucionales recurrentes

La Figura 22 muestra la arquitectura detallada. A continuación estudiamos en detalle en qué consiste cada componente de DCRNN.

7.5.1 Modelado de dependencias espaciales

La mayoría de los sistemas representados por grafos tienen una dinámica estocástica, es decir, que su comportamiento está influenciado por variables aleatorias. En este contexto, los modelos de difusión son especialmente útiles pues capturan de manera natural las variaciones y la incertidumbre inherentes a estos sistemas.

En el caso de la arquitectura DCRNN, el proceso de difusión viene dado por un paseo aleatorio en el grafo, G con una probabilidad de reinicio $\alpha \in [0, 1]$, y una matriz de

transición de estado $D_O^{-1}W$, donde $D_O = \text{diag}(W\mathbf{1})$ es la matriz diagonal de grados de salida, y $\mathbf{1} \in \mathbb{R}^N$ denota el vector de unos.

Tras varias iteraciones, o pasos de tiempo, dicho proceso de Markov converge a una distribución estacionaria $P \in \mathbb{R}^{N \times N}$, donde cada fila $P_{i,:}$ representa la probabilidad de difusión desde el nodo v_i , indicando la proximidad con respecto al nodo v_i .

Además, se puede demostrar que la distribución estacionaria del proceso de difusión viene dada por una combinación ponderada de paseos aleatorios infinitos en el grafo, es decir, se puede representar como

$$P = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k D_O^{-1} W^k \quad (7.5.1)$$

Sin embargo, en la práctica, se utilizan K pasos del proceso de difusión y se asigna un peso entrenable a cada uno de ellos. También se incluye el proceso de difusión en dirección inversa para ofrecer al modelo más flexibilidad para capturar la influencia entre nodos con enlaces bidireccionales. Formalmente, la operación de convolución de difusión sobre una señal de grafo $X \in \mathbb{R}^{N \times P}$ y un filtro f_θ se define como:

$$X_{:,p} \star_G f_\theta = \sum_{k=0}^{K-1} \left(\theta_{k,1} D_O^{-1} W^k + \theta_{k,2} D_I^{-1} W^k \right) X_{:,p} \quad \text{para } p \in \{1, \dots, P\} \quad (7.5.2)$$

donde $\theta \in \mathbb{R}^{K \times 2}$ son los parámetros del filtro, y $D_O^{-1}W, D_I^{-1}W$ representan las matrices de transición del proceso de difusión original y el inverso, respectivamente.

Finalmente, con todo lo que acabamos de describir, podemos construir una **capa convolucional de difusión** que mapea entradas de P -dimensiones a salidas de Q -dimensiones, mediante la ecuación siguiente

$$H_{:,q} = \sum_{p=1}^P X_{:,p} \star_G f_{\Theta_{q,p},:} \quad \text{para } q \in \{1, \dots, Q\} \quad (7.5.3)$$

donde $X \in \mathbb{R}^{N \times P}$ es la entrada, $H \in \mathbb{R}^{N \times Q}$ es la salida, $\{f_{\Theta_{q,p},:}\}$ son los filtros y a es la función de activación (por ejemplo, ReLU, Sigmoide).

7.5.2 Modelado de dependencias temporales

Esta arquitectura incorpora bloques GRU para modelar la dependencia temporal. Sin embargo, se propone reemplazar las multiplicaciones entre matrices presentes en el bloque GRU con la operación de convolución de difusión, lo que lleva a definir la **Unidad Recurrente de Convolución Difusiva (DCGRU)**.

Las ecuaciones de la DCGRU son:

$$r(t) = \sigma(\Theta_r \star_G [X(t), H(t-1)] + b_r) \quad (7.5.4)$$

$$u(t) = \sigma(\Theta_u \star_G [X(t), H(t-1)] + b_u) \quad (7.5.5)$$

$$C(t) = \tanh(\Theta_C \star_G [X(t), (r(t) \odot H(t-1))] + b_c) \quad (7.5.6)$$

$$H(t) = u(t) \odot H(t-1) + (1 - u(t)) \odot C(t) \quad (7.5.7)$$

donde $X(t)$ y $H(t)$ denotan la entrada y la salida en el tiempo t , respectivamente; $r(t)$ y $u(t)$ son la puerta de reinicio y la puerta de actualización en el tiempo t ; \star_G denota la convolución de difusión definida anteriormente; y Θ_r , Θ_u , Θ_c son los parámetros para los filtros correspondientes.

Para la predicción de múltiples pasos hacia adelante, utilizamos una arquitectura de secuencia a secuencia con una configuración de codificador-decodificador. Tanto el codificador como el decodificador están compuestos por redes neuronales recurrentes que incorporan Unidades Recurrentes de Convolución Difusiva (DCGRU). Durante el proceso de entrenamiento, alimentamos el codificador con la serie temporal original. Los estados finales del codificador, se utilizan luego para inicializar el decodificador, obteniendo así las predicciones finales.

8

ALGORITMOS BASADOS EN MECANISMOS DE ATENCIÓN

Tras haber explorado las arquitecturas basadas en redes neuronales recurrentes en el capítulo anterior, vamos a proceder a introducir las arquitecturas de GNNs temporales que utilizan mecanismos de atención.

8.1 ASTGCN

Las Redes Neuronales Convolucionales Espacio-Temporales basadas en Atención (ASTGCN, por sus siglas en inglés) [15] se propusieron inicialmente para abordar el desafío de predecir el flujo de tráfico. Buscan capturar de manera efectiva las características dinámicas espaciotemporales de los grafos, mediante mecanismos de atención espaciotemporal con convoluciones para procesar de manera eficiente la información.

La arquitectura ASTGCN se compone de tres componentes independientes, cada uno diseñado para modelar diferentes propiedades temporales del flujo de tráfico: dependencias recientes, diarias y semanales. Aunque realmente nuestros datos no tienen esta periodicidad, hemos decidido incluirlo en la lista de algoritmos con los que vamos a experimentar de cara a comparar sus resultados. La arquitectura en detalle se muestra en la Figura 23.

Cada componente de ASTGCN, que llamaremos **bloque espaciotemporal**, incluye dos partes principales: el mecanismo de atención espaciotemporal, y la convolución espaciotemporal.

8.1.1 *Mecanismo de atención espaciotemporal*

La primera, el mecanismo de atención espaciotemporal, permite al modelo captar las correlaciones dinámicas espaciales y temporales en los datos. Se divide a su vez en dos tipos distintos de atención: atención espacial y temporal.

Comenzamos por la **atención espacial**, cuyo fundamento está en que los estados de diferentes nodos influyen entre sí, y esta influencia mutua es altamente dinámica.

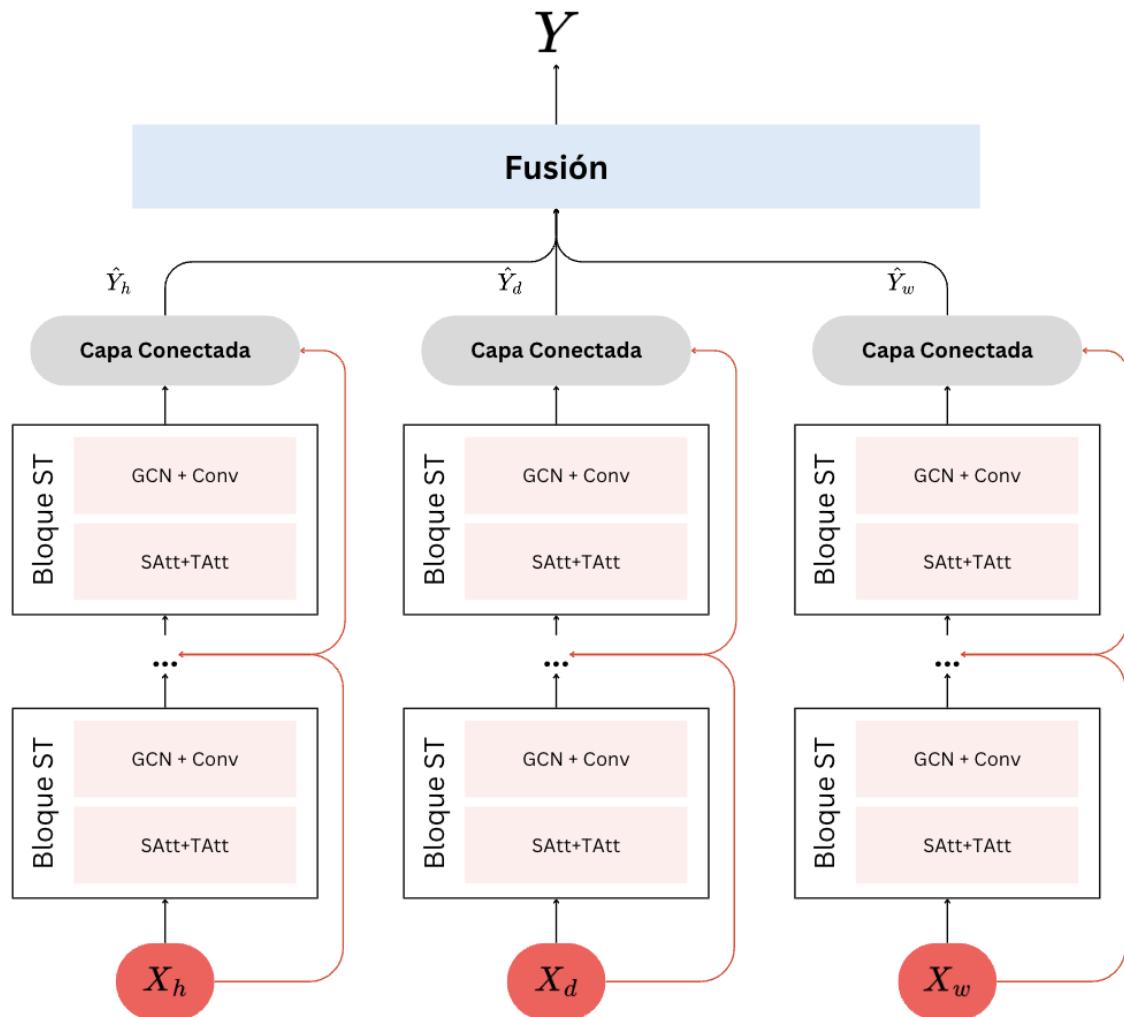


Figura 23: Arquitectura ASTGCN

Para ello, se utiliza un mecanismo de atención que captura dinámicamente las correlaciones entre nodos en la dimensión espacial.

Vamos a estudiar su funcionamiento tomando como ejemplo la atención espacial en la componente *reciente* X_h :

$$S = V_s \cdot \sigma \left((X_h^{(r-1)} W_1) W_2 (W_3 X_h^{(r-1)})^T + b_s \right) \quad (8.1.1)$$

$$S'_{i,j} = \frac{\exp(S_{i,j})}{\sum_{j=1}^N \exp(S_{i,j})} \quad (8.1.2)$$

donde:

- $X_h^{(r-1)} = (X_1, X_2, \dots, X_{T_{r-1}}) \in \mathbb{R}^{N \times C_{r-1} \times T_{r-1}}$ es la entrada del bloque espacio-temporal r .
- C_{r-1} es el número de canales de los datos de entrada en la capa r , cuyo valor inicial C_0 es equivalente al número de grafos de entrada.
- T_{r-1} es la longitud de la dimensión temporal en la capa r , cuyo valor inicial dependerá de si estamos en el componente reciente, diario o semanal.
- $V_s, b_s \in \mathbb{R}^{N \times N}, W_1 \in \mathbb{R}^{T_{r-1}}, W_2 \in \mathbb{R}^{C_{r-1} \times T_{r-1}}, W_3 \in \mathbb{R}^{C_{r-1}}$ son parámetros entrenables

La matriz de atención S se calcula dinámicamente de acuerdo con la entrada actual de esta capa, y cada elemento $S_{i,j}$ en S representa semánticamente la fuerza de correlación entre el nodo i y el nodo j .

Por último, se utilizará una función softmax para asegurar que los pesos de atención de un nodo sumen uno. Al realizar las convoluciones en el grafo, acompañaremos la matriz de adyacencia A con la matriz de atención espacial $S' \in \mathbb{R}^{N \times N}$ para ajustar dinámicamente los pesos de impacto entre nodos.

Seguidamente, vamos a pasar a la **atención temporal**, que busca modelar las correlaciones entre las condiciones de los nodos en diferentes instantes de tiempo, teniendo en cuenta que éstas también varían según las circunstancias. De una manera similar, tenemos:

$$E = V_e \cdot \sigma(((X_h^{(r-1)})^T U_1) U_2 (U_3 X_h^{(r-1)}) + b_e) \quad (8.1.3)$$

$$E'_{i,j} = \frac{\exp(E_{i,j})}{\sum_{j=1}^{T_{r-1}} \exp(E_{i,j})} \quad (8.1.4)$$

donde $V_e, b_e \in \mathbb{R}^{T_{r-1} \times T_{r-1}}, U_1 \in \mathbb{R}^N, U_2 \in \mathbb{R}^{C_{r-1} \times N}, U_3 \in \mathbb{R}^{C_{r-1}}$ son parámetros entrenables.

El valor de un elemento $E_{i,j}$ en E representa la fuerza de las dependencias entre el tiempo i y j . De nuevo, se normalizará E es por la función softmax para obtener la matriz de atención temporal normalizada, que se aplicará directamente a la entrada para obtener:

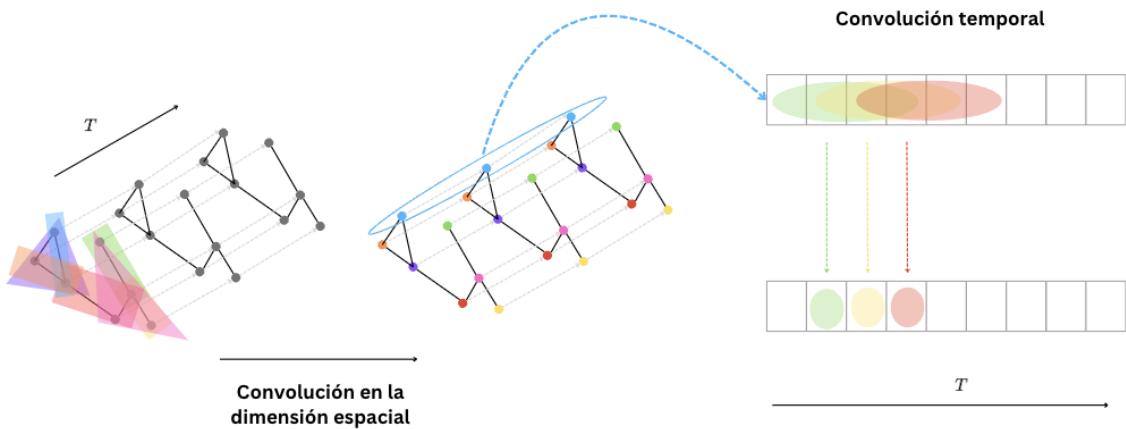


Figura 24: Convolución espaciotemporal

$$\hat{X}_h^{(r-1)} = (\hat{X}_1, \hat{X}_2, \dots, \hat{X}_{T_{r-1}}) = (X_1, X_2, \dots, X_{T_{r-1}}) E' \in \mathbb{R}^{N \times C_{r-1} \times T_{r-1}} \quad (8.1.5)$$

Que será utilizado para ajustar dinámicamente la entrada fusionando información relevante.

8.1.2 Convolución espaciotemporal

La segunda parte del bloque espaciotemporal es la que se encarga de realizar la convolución, y recibe como entrada los grafos ajustados por el mecanismo de atención que acabamos de estudiar. Como su propio nombre indica, dicha convolución se realiza tanto en la dimensión espacial como en la dimensión temporal (Figura 24).

Comenzamos estudiando la convolución a lo largo de la dimensión espacial, que, en cada instante de tiempo realizará convoluciones de grafos, tal y como hemos visto en secciones anteriores.

En el análisis de grafos espectral, podemos representar un grafo por su matriz Laplaciana correspondiente, definida como $L = D - A$, donde A es la matriz de adyacencia del grafo, y D es la matriz diagonal de grados, con cada elemento D_{ii} representa el grado del nodo i . Esta matriz es clave, pues las propiedades de la estructura del grafo se pueden obtener a partir de ella y sus autovalores. Además, se puede obtener una matriz Laplaciana normalizada, mediante $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, donde I_N es una matriz unitaria.

La descomposición en autovalores de la matriz Laplaciana es

$$L = U \Lambda U^T \quad (8.1.6)$$

donde $\Lambda = \text{diag}([\lambda_0, \dots, \lambda_{N-1}])$ es una matriz diagonal, y U es la base de Fourier, formada por los autovectores de la matriz Laplaciana normalizada. Esta base es fundamental para transformar las señales del grafo al dominio espectral, facilitando operaciones como la convolución de grafos.

En cada instante de tiempo, la señal en el grafo, denotada por x , se convierte mediante la transformada de Fourier del grafo: $\hat{x} = U^T x$.

Podemos definir formalmente la operación convolución de grafo por medio de un filtro g_θ , denotada por $g_\theta *_G x$, como sigue

$$g_\theta *_G x = g_\theta(L)x \quad (8.1.7)$$

$$= g_\theta(U\Lambda U^T)x \quad (8.1.8)$$

$$= Ug_\theta(\Lambda)U^T x \quad (8.1.9)$$

Según esta definición, una señal de grafo x es filtrada por un kernel g_θ mediante la multiplicación entre g_θ y la transformada de Fourier del grafo $U^T x$.

Este método aprovecha las correlaciones de las señales sobre la red en la dimensión espacial, permitiendo una filtración eficaz basada en la topología del grafo. Sin embargo, resulta computacionalmente costoso obtener la descomposición de la matriz Laplaciana normalizada L en dimensiones altas, por lo que se utilizan polinomios de Chebyshev para aproximar la convolución de forma eficiente.

Así, podemos restringir el filtro $g_\theta(\Lambda)$ a $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$, donde $\theta_k \in \mathbb{R}^K$ es el vector de coeficientes del polinomio, que determinará el radio máximo de la convolución.

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \quad (8.1.10)$$

Sin embargo, todavía es necesario descomponer la matriz Laplaciana ajustada. Para solucionarlo, se suele utilizar la forma escalada de la Laplaciana, definida por $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$, donde λ_{\max} es el máximo autovalor de L , aproximada por un polinomio de Chebyshev $T_k(\tilde{\Lambda})$:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \quad (8.1.11)$$

Por lo tanto, la operación de convolución se puede reescribir como sigue:

$$g_\theta *_G x = g_\theta(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \quad (8.1.12)$$

En nuestro caso particular, es necesario incluir la información obtenida en el mecanismo de atención espacial, por lo que redefinimos la operación:

$$g_\theta *_G x = \sum_{k=0}^{K-1} \theta_k (T_k(\tilde{L}) \odot S') x, \quad (8.1.13)$$

donde \odot representa el producto Hadamard.

Después de realizar la convolución de grafos en la dimensión espacial, se apila una capa de convolución estándar en la dimensión temporal. Esta capa actualiza la señal de cada nodo combinando la información de los instantes de tiempo adyacentes.

Como ejemplo, la operación en la r -ésima capa del componente reciente se puede describir con la siguiente fórmula:

$$X_h^{(r)} = \text{ReLU}(\Phi * (\text{ReLU}(g_\theta *_G X_h^{(r-1)}))) \quad (8.1.14)$$

donde $*$ denota una operación de convolución estándar, y Φ representa los parámetros de convolución temporal.

Un módulo de atención espaciotemporal junto con un módulo de convolución espaciotemporal forma un bloque espaciotemporal. Se apilan múltiples bloques espaciotemporales para extraer un rango más amplio de correlaciones dinámicas.

Finalmente, se añade una capa completamente conectada para asegurar que la salida de cada componente tenga la misma dimensión y forma que el objetivo de la predicción. La capa completamente conectada final utiliza ReLU como función de activación.

8.1.3 Fusión de componentes

Por último, será necesario integrar las salidas de las tres componentes. Cuando se fusionan las salidas de las diferentes componentes, los pesos de impacto de las tres componentes para cada nodo son diferentes y deben aprenderse de los datos históricos. Así, el resultado final de la predicción después de la fusión es:

$$\hat{Y} = W_h \odot \hat{Y}_h + W_d \odot \hat{Y}_d + W_w \odot \hat{Y}_w \quad (8.1.15)$$

donde \odot denota el producto de Hadamard y W_h , W_d , y W_w son parámetros de aprendizaje que reflejan los grados de influencia de las tres componentes temporales en las predicciones.

8.2 MSTGCN

La siguiente arquitectura es una versión simplificada del ASTGCN, conocida como Redes Neuronales Convolucionales Espacio-Temporales Multi-Componente basadas en Atención (MSTGCN, por sus siglas en inglés), que elimina los componentes de atención espaciotemporal.

Las configuraciones y arquitectura de MSTGCN son idénticas a las de ASTGCN, con la excepción de que no incorpora ningún mecanismo de atención espaciotemporal. Este cambio busca simplificar el modelo reduciendo la complejidad computacional y posiblemente acelerar el tiempo de entrenamiento, aunque puede afectar la capacidad del modelo para ajustarse correctamente a los datos.

8.3 MTGNN

Las Redes Neuronales Espacio-Temporales Multi-Componente sobre Grafos (MTGNN, por sus siglas en inglés) [41], fueron originalmente creadas para la predicción de series temporales multivariantes, donde normalmente se dan correlaciones entre las variables que permiten modelar la serie completa como una serie de grafos dinámicos.

Esta es una de las arquitecturas que no utilizan la información topológica (presencia de enlaces y las características de los mismos), sino que infieren dichas propiedades a lo largo del entrenamiento. Aunque sería especialmente interesante en casos donde no se dispone de esa información, en nuestro caso también es interesante, pues nos permite comprobar si un modelo adaptativo puede funcionar mejor, al no restringir los efectos de un problema en la red a los nodos o enlaces inmediatamente cercanos.

MTGNN está formada por tres componentes distintas. La primera, una Capa de Aprendizaje de Grafo, que infiere de manera adaptativa la matriz de adyacencia del grafo a partir de los datos. La segunda es un Módulo de Convolución de Grafo, que aplica convoluciones sobre el grafo para procesar y extraer características espaciales relevantes. Por último, tenemos un Módulo de Convolución Temporal, que permiten al modelo identificar patrones a lo largo del tiempo.

En la práctica, la arquitectura es como se ve en la Figura 25, donde después de la capa de aprendizaje, tenemos m módulos de convolución de grafo, m módulos de convolución temporal, y un módulo de salida. Además, se añaden conexiones de salto después de cada módulo de convolución temporal.

A continuación vemos cómo funciona cada una de ellas.

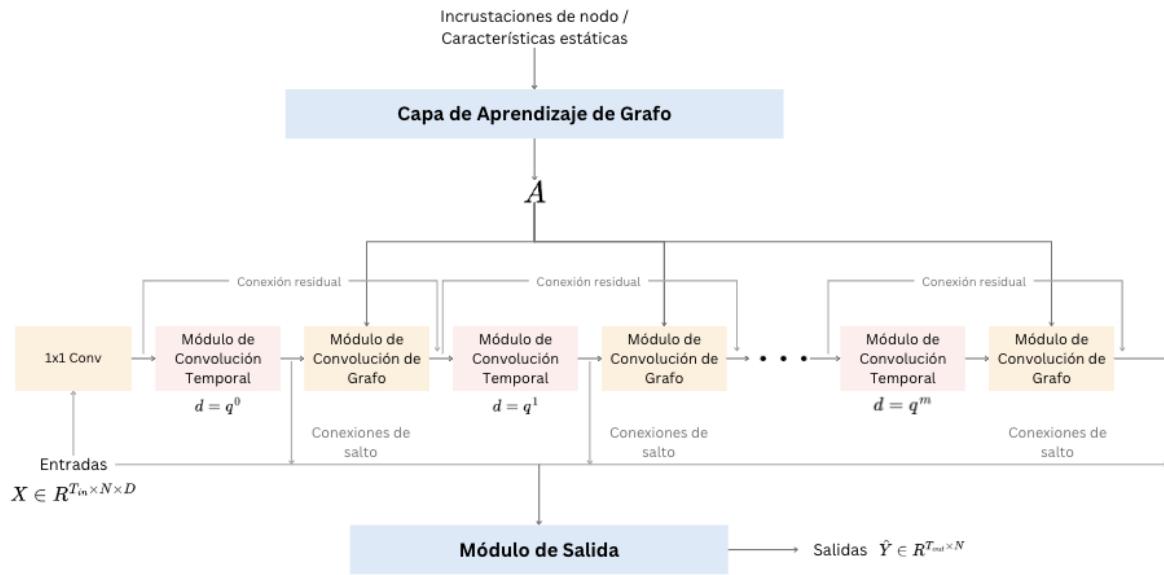


Figura 25: Arquitectura MTGNN

8.3.1 Capa de Aprendizaje del Grafo

Tradicionalmente, para construir un grafo, los estudios existentes miden la similitud entre pares de nodos utilizando métricas de distancia, como el producto punto y la distancia euclidiana. Sin embargo, este enfoque conlleva una alta complejidad temporal y espacial de $O(N^2)$, limitando la capacidad del modelo para manejar grafos más grandes.

Para superar esta limitación, se adopta un enfoque de muestreo que solo calcula las relaciones par a par entre un subconjunto de nodos, eliminando el cuello de botella de la computación. Este método reduce significativamente la complejidad sin sacrificar la capacidad de capturar relaciones esenciales.

Un aspecto innovador de la capa de aprendizaje del grafo es su capacidad para extraer relaciones unidireccionales, fundamental en escenarios como la predicción del estado de la red en el futuro, donde el cambio en la condición de un nodo puede influir en la condición de otro.

Formalmente, la matriz se obtiene mediante las siguientes operaciones:

$$M_1 = \tanh(\alpha E_1 \Theta_1) \quad (8.3.1)$$

$$M_2 = \tanh(\alpha E_2 \Theta_2) \quad (8.3.2)$$

$$A = \text{ReLU}(\tanh(\alpha(M_1 M_2^T - M_2 M_1^T))) \quad (8.3.3)$$

donde E_1, E_2 representan las incrustaciones de nodos, inicializados aleatoriamente, y actualizados durante el entrenamiento, Θ_1, Θ_2 son parámetros del modelo, y α es un hiperparámetro que controla la tasa de saturación de la función de activación.

La propiedad asimétrica de la matriz de adyacencia obtenida se logra a través de la resta y la función de activación ReLU, asegurando que si A_{vu} es positivo, su contraparte diagonal A_{uv} será cero.

Es importante notar que las entradas para la capa de aprendizaje del grafo no se limitan a las incrustaciones de los nodos. En casos como el de nuestro problema, donde disponemos de información sobre los atributos de cada nodo, también se puede configurar $E_1 = E_2 = Z$, donde Z es la matriz estática de características de nodos.

8.3.2 Módulo de Convolución del Grafo

Tal y como veníamos explicando, el módulo de convolución de grafo tiene como objetivo fusionar la información de un nodo con la de sus vecinos para manejar las dependencias espaciales. Este módulo consiste en dos capas de propagación *mix-hop* que procesan por separado la información, y cuyas salidas se sumarán para obtener la salida final del módulo.

La arquitectura del módulo se puede estudiar en la Figura 26.

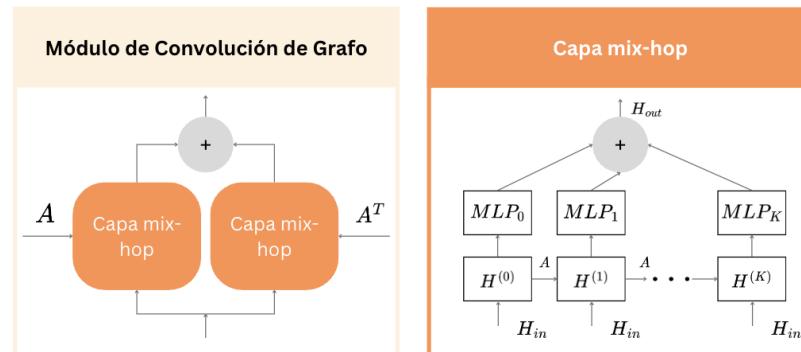


Figura 26: Módulo de Convolución del Grafo

Veamos en qué consiste una capa *mix-hop*.

Capa mix-hop

Esta capa se compone de dos pasos:

1. El paso de propagación de información
2. El paso de selección de información.

El paso de propagación de información se define matemáticamente como sigue:

$$H^{(k)} = \beta H_{in} + (1 - \beta) \tilde{A} H^{(k-1)}, \quad (8.3.4)$$

donde β es un hiperparámetro que controla la proporción de retención de los estados originales del nodo raíz.

El paso de selección de información se define como:

$$H_{\text{out}} = \sum_{i=0}^K H^{(k)} W^{(k)}, \quad (8.3.5)$$

donde K es la profundidad de la propagación, H_{in} representa los estados ocultos de entrada, y $W^{(k)}$ es la matriz de parámetros que actúa como un selector de características. Este paso filtra la información importante producida en cada salto.

La idea de mix-hop ha sido explorada en otros trabajos, en los que se aplica un mecanismo de atención para ponderar la información entre diferentes saltos. La aproximación en MTGNN mantiene un equilibrio entre la información local y del vecindario para evitar el problema de *over-smoothing* común en las GCNs, donde la información de saltos más altos puede no contribuir o contribuir negativamente al rendimiento general.

8.3.3 Módulo de Convolución Temporal

El módulo de convolución incorpora dos capas de incepción dilatada, un concepto nuevo hasta el momento.

Una capa de incepción dilatada es una técnica diseñada para el análisis de datos secuenciales, que combina dos enfoques:

- La operación de incepción, que utilizan filtros de varios tamaños para capturar patrones a diferentes escalas temporales.
- La operación de convolución dilatada, que consiguen ampliar el rango temporal sin aumentar el número de parámetros.

La primera de las capas que forman el módulo es seguida por una función de activación tangente hiperbólica y funciona como un filtro. La otra es seguida por una función de activación sigmoide y actúa como una compuerta para controlar la cantidad de información que el filtro puede transmitir al siguiente módulo. La arquitectura del módulo puede estudiarse en la Figura 27.

Vamos a comenzar el estudio de este módulo presentando en qué consiste la operación de convolución dilatada, otro concepto que no se había introducido anteriormente. Despues, procederemos a estudiar el funcionamiento de una capa de incepción dilatada.

Convolución dilatada

La operación de convolución dilatada es una técnica avanzada utilizada en el procesamiento de señales unidimensionales, que permite capturar dependencias a largo plazo en datos secuenciales. Se basan en una convolución estándar, pero introducen un parámetro adicional conocido como el *factor de dilatación*, que espacia los elementos

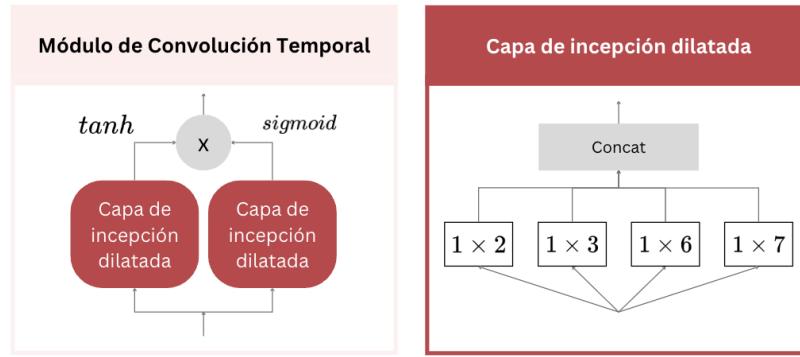


Figura 27: Módulo de Convolución Temporal

del kernel de convolución, permitiendo abarcar una región más amplia de la entrada sin aumentar el número de parámetros del modelo.

La operación de convolución dilatada se define matemáticamente como sigue:

$$y[t] = \sum_{k=0}^{K-1} x[t + k \cdot d] \cdot w[k] \quad (8.3.6)$$

donde $y[t]$ es el valor de salida en el tiempo t , $x[t]$ es la entrada, $w[k]$ son los pesos del filtro, K es el tamaño del kernel, y d es el factor de dilatación, que determina el intervalo entre los elementos del kernel aplicados a la entrada.

Capa de Incepción Dilatada

Tal y como hemos visto anteriormente, una capa de incepción dilatada consta de dos operaciones: la incepción y la convolución.

La elección del tamaño adecuado del filtro es un problema muy común en las redes convolucionales. El tamaño del filtro puede ser demasiado grande para representar patrones de señales a corto plazo de manera sutil, o demasiado pequeño para descubrir patrones de señales a largo plazo de manera suficiente. Para solucionarlo, se propone utilizar una incepción temporal, que consiste en el uso de cuatro tamaños distintos de filtro, en particular, 1×2 , 1×3 , 1×6 y 1×7 .

Por otro lado, tenemos la operación de convolución dilatada, que utiliza un filtro de convolución estándar en entradas submuestreadas con una cierta frecuencia. Por ejemplo, donde el factor de dilatación es 2, aplica la convolución estándar en entradas muestreadas cada dos pasos.

La capa de incepción dilatada surge al combinar la incepción y la dilatación. Dado una entrada de secuencia 1D $z \in \mathbb{R}^T$ y filtros que consisten en $f_{1 \times 2} \in \mathbb{R}^2$, $f_{1 \times 3} \in \mathbb{R}^3$, $f_{1 \times 6} \in \mathbb{R}^6$, y $f_{1 \times 7} \in \mathbb{R}^7$, la capa de incepción dilatada toma la forma,

$$z = \text{concat}(z * f_{1 \times 2}, z * f_{1 \times 3}, z * f_{1 \times 6}, z * f_{1 \times 7}), \quad (8.3.7)$$

donde \star es la operación de convolución dilatada, las salidas de los cuatro filtros son truncadas a la misma longitud de acuerdo con el filtro más grande y concatenadas a través de la dimensión del canal.

8.4 STGCN

El siguiente tipo de red que vamos a estudiar, denominado Red Convolutacional Espacio-Temporal sobre Grafos (STGCN, por sus siglas en inglés) [44], fue también creado en el contexto de la predicción del tráfico en una ciudad.

Se compone de varios Bloques Convolucionales Espacio-Temporales (bloques ST-Conv), donde cada uno está formado por dos capas de convolución temporal con puerta y una capa de convolución espacial entre ellas. Veamos en qué consiste cada una de ellas.

La arquitectura detallada se puede estudiar de manera visual en la Figura 28.

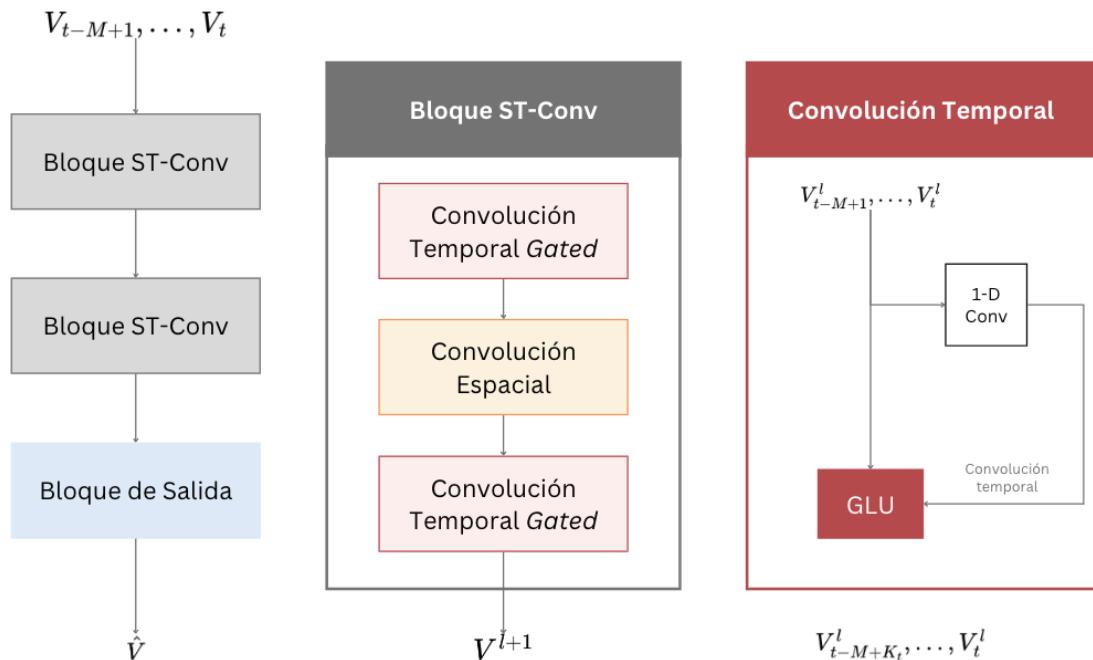


Figura 28: Arquitectura de STGCN y sus componentes

8.4.1 Capa de Convolución Espacial

La operación de convolución definida en la Sección 8.1, denotada por $g_\theta *_G x$:

$$g_\theta *_G x = g_\theta(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \quad (8.4.1)$$

puede extenderse a tensores multidimensionales. Para una señal con C_i canales $X \in \mathbb{R}^{n \times C_i}$, la convolución de grafo se puede generalizar mediante:

$$y_j = \sum_{i=1}^{C_i} g_\theta^{i,j}(L)x_i \in \mathbb{R}^n, \quad 1 \leq j \leq C_o \quad (8.4.2)$$

con los vectores $C_i \times C_o$ de coeficientes de Chebyshev de $g_\theta^{i,j} \in \mathbb{R}^K$ (donde C_i, C_o son el tamaño de entrada y salida de los mapas de características, respectivamente).

La convolución de grafo para variables 2-D se denota como $g_\theta *_G X$ con $g_\theta \in \mathbb{R}^{K \times C_i \times C_o}$.

Especificamente, la entrada para la predicción está compuesta por M grafos. Cada grafo v_t puede considerarse como una matriz $X_t \in \mathbb{R}^{n \times C_i}$ cuya columna i tiene dimensión C_i y representa el valor del nodo i del grafo, como (en nuestro caso, $C_i = 1$).

Para cada paso de tiempo $t \in M$ se realiza la operación de convolución de grafo con el mismo kernel g_θ sobre $X_t \in \mathbb{R}^{n \times C_i}$ en paralelo. Así, la convolución de grafo puede generalizarse aún más en variables 3-D, notada como $\Theta *_G X$ con $X \in \mathbb{R}^{M \times n \times C_i}$.

8.4.2 Capa de Convolución Temporal

En esta arquitectura se utilizan estructuras completamente convolucionales a lo largo del eje temporal para capturar comportamientos dinámicos temporales. La capa convolucional temporal contiene una convolución causal 1-D con un kernel o filtro de anchura K_t seguido por unidades lineales con compuerta (GLU) como no linealidad.

Las GLU, estructuras no estudiadas en este trabajo hasta el momento, son comúnmente utilizadas en las redes neuronales convolucionales y pese a que su nombre es similar a las GRU, no poseen un comportamiento recurrente. Operan multiplicando la salida de una transformación lineal por una compuerta sigmoide, lo que permite realizar una selección adaptativa de características a lo largo de cada capa de la red. Este mecanismo hace que las GLU sean útiles para controlar la cantidad de información no lineal que fluye a través de la red.

$$GLU(x) = (Wx + b) \odot \sigma(Vx + c) \quad (8.4.3)$$

Donde W, V son matrices de pesos, b, c son términos de sesgo, σ denota la función sigmoide y \odot representa el producto Hadamard.

Volviendo a la convolución temporal, para cada nodo en el grafo G , la convolución explora K_t vecinos de los elementos de entrada, lo que reduce la longitud de las

secuencias en $K_t - 1$ cada vez. Por tanto, la entrada de la convolución temporal para cada nodo puede considerarse como una secuencia de longitud M con C_i canales, la denotaremos $Y \in \mathbb{R}^{M \times C_i}$.

El kernel de convolución $g_\gamma \in \mathbb{R}^{K_t \times C_i \times 2C_o}$ está diseñado para mapear la entrada Y a un único elemento de salida $[P, Q] \in \mathbb{R}^{(M-K_t+1) \times (2C_o)}$ (P, Q se dividen por la mitad con el mismo tamaño de canales). Como resultado, la convolución temporal con compuerta se puede definir como,

$$g_\gamma *_T Y = P \odot \sigma(Q) \in \mathbb{R}^{(M-K_t+1) \times C_o}, \quad (8.4.4)$$

donde P, Q son las entradas de las compuertas GLU, \odot denota el producto Hadamard. La compuerta sigmoide $\sigma(Q)$ controla qué entrada P de los estados actuales es relevante para descubrir las variaciones dinámicas en las series temporales.

Las compuertas no lineales contribuyen también al uso del campo de entrada completo a través de capas temporales apiladas, conectadas entre sí a través de conexiones residuales. De manera similar, la convolución temporal también puede generalizarse a variables 3-D empleando el mismo kernel de convolución g_γ a cada nodo $Y_i \in \mathbb{R}^{M \times C_i}$ (por ejemplo, estaciones de sensores) en G de manera equitativa, denotado como $g_\gamma *_T Y''$ con $Y \in \mathbb{R}^{M \times n \times C_i}$.

8.4.3 Bloques Convolucionales Espacio-Temporales (ST-Conv)

Para fusionar características tanto del dominio espacial como del temporal, se construye el bloque convencional espaciotemporal (ST-Conv block). Este bloque puede ser apilado o extendido dependiendo de la escala y complejidad de los casos particulares. La capa espacial en el medio sirve para conectar dos capas temporales, permitiendo una rápida propagación del estado espacial desde la convolución de grafo a través de las convoluciones temporales. Además, esta disposición de las capas ayuda a la red a lograr la compresión de escala y la compactación de características mediante la reducción y ampliación de canales C a través de la capa convolucional de grafo. Además, se utiliza la normalización de capa dentro de cada bloque ST-Conv para prevenir el sobreajuste.

La entrada y salida de los bloques ST-Conv son todos tensores 3-D. Para la entrada $v_l \in \mathbb{R}^{M \times n \times C_l}$ del bloque l , la salida $v_{l+1} \in \mathbb{R}^{(M-2(K_l-1)) \times n \times C_{l+1}}$ se calcula mediante,

$$v_{l+1} = \Gamma_{l1} *_T \text{ReLU}(g_{\theta_l} *_G (\Gamma_{l0} *_T v_l)), \quad (8.4.5)$$

donde g_{θ_l} es el kernel espectral de la convolución de grafo.

Después de apilar dos bloques ST-Conv, añadimos una capa adicional de convolución temporal con una capa completamente conectada como capa de salida al final.

Parte IV

EXPERIMENTACIÓN

Exposición de los experimentos llevados a cabo en el contexto de predicción y clasificación. Comparación de resultados entre arquitecturas.

9

REGRESIÓN. PREDICCIÓN DE ESTADOS FUTUROS

Para estudiar los resultados de los experimentos realizados, vamos a analizar las medidas de rendimiento obtenidas sobre el conjunto de entrenamiento, evaluación y prueba, así como la diferencia entre las métricas por nodo en el conjunto de prueba para cada una de las arquitecturas, obteniendo una visión global del rendimiento de cada modelo. Más adelante, en el Apéndice B, se pueden observar en detalle alguna de las predicciones para las simulaciones del conjunto de test, así como una descripción detallada de los parámetros ajustados en cada arquitectura.

Es necesario destacar que, aunque originalmente se deseaba realizar el ajuste de los modelos mediante una búsqueda de fuerza bruta en el espacio de parámetros, algunos de los modelos no permitían llevarla a cabo debido a restricciones computacionales. Por ese motivo, en algunos casos se ha tenido que adaptar esta búsqueda, reduciendo el número de parámetros incluidos en la búsqueda, o incluso eliminando por completo este paso. Como ejemplo, el modelo STConv estuvo ejecutando la búsqueda en local durante una semana, de manera ininterrumpida, para una de las cinco simulaciones, y únicamente completó el 20 % de las iteraciones. Finalmente, se decidió realizar tres tipos de ajuste distintos: Búsqueda por Fuerza Bruta (BFB), Búsqueda Aleatoria (BA), y Búsqueda Manual (BM).

En la Figura 29 se puede estudiar el procedimiento experimental en su totalidad, desde el procesamiento de los datos brutos, en forma de ficheros independientes, hasta la exportación y comparación de los resultados.

Segmentaremos este capítulo en las cinco simulaciones que se presentaron en el Capítulo 6: cortocircuito de generador, cortocircuito de nodo, error de nodo, cortocircuito de rama y error de rama.

9.1 CORTOCIRCUITO DE GENERADOR

Los resultados generales de todos los experimentos se pueden estudiar en la Tabla 1. En ella podemos estudiar los valores de la función de pérdida, que en este caso es el **error cuadrático medio**, y de la **métrica R₂**, también denominada coeficiente de determinación.

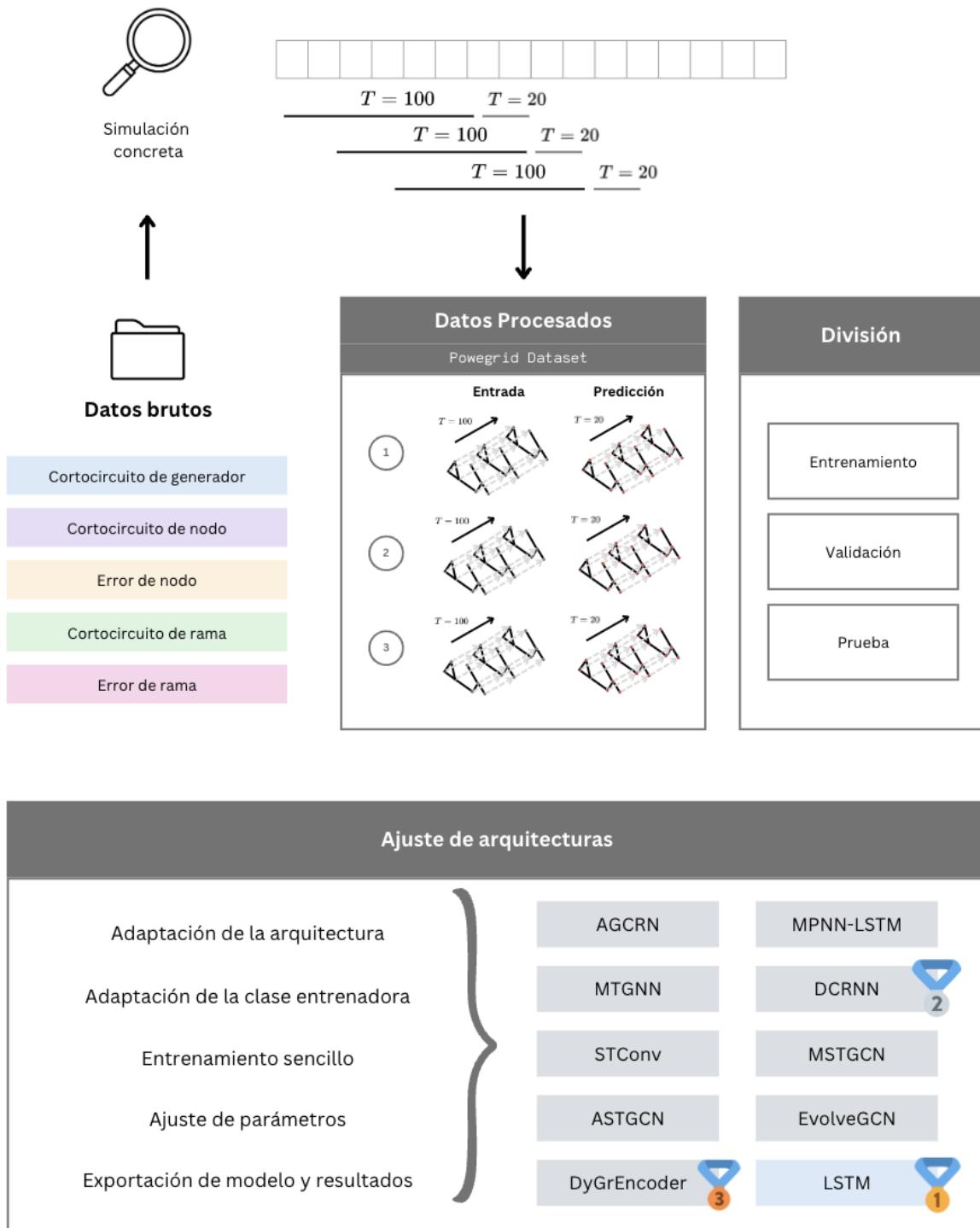


Figura 29: Flujo experimental completo

El coeficiente de determinación, r^2 , es una métrica utilizada para evaluar la precisión de un modelo de regresión. Se define como

$$r^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2},$$

donde y_i representa los valores observados, \hat{y}_i los valores predichos por el modelo, y \bar{y} la media de los valores observados.

El numerador, $\sum_{i=1}^n (y_i - \hat{y}_i)^2$, es la suma de los cuadrados de los errores de predicción (también conocido como suma de los cuadrados de los residuos), mientras que el denominador, $\sum_{i=1}^n (y_i - \bar{y})^2$, es la suma total de los cuadrados (también conocido como suma total de los cuadrados). Un valor de r^2 cercano a 1 indica que el modelo explica bien la variabilidad de los datos, mientras que un valor cercano a 0 indica que el modelo no explica bien la variabilidad.

Modelo	Train Loss	Test Loss	Eval Loss	Eval R2	Test R2	Tipo de Ajuste
Baseline						
LSTM	0.0087	0.0076	0.0098	0.3412	0.2042	Baseline
Recurrentes						
MPNN_LSTM	0.0593	0.0459	0.0528	-16.7440	-18.2476	BFB
DyGrEncoder	0.0068	0.0062	0.0089	-1.2861	-1.2048	BFB
EvolveGCN	0.0826	0.0639	0.0787	-20.7123	-21.7990	BFB
DCRNN	0.0094	0.0071	0.0097	-1.7065	-1.8767	BFB
AGCRN	0.0104	0.0087	0.0115	0.3193	-0.1950	BFB
Atención						
ASTGCN	0.0089	0.0071	0.0103	-1.7555	-1.7838	BA
MSTGCN	0.0091	0.0075	0.0103	-1.7902	-1.9371	BFB
MTGNN	0.0101	0.0074	0.0124	-2.2600	-1.6072	BA
STConv	0.0175	0.0448	0.0473	-13.1932	-25.3412	BM

Cuadro 1: Resultados de experimentos para cortocircuito del generador. Los mejores resultados en el conjunto de test están en negrita y los peores en cursiva.

Además, en la Figura 30 podemos analizar la función de pérdida en los conjuntos de entrenamiento, prueba y validación. Los resultados en general son positivos, debido a que la función pérdida en el conjunto de prueba es menor que en el entrenamiento, lo cual nos lleva a descartar la presencia de *overfitting*. Es necesario mencionar que a la hora de entrenar cada uno de los modelos, se estudió la evolución de la función de pérdida en el conjunto de entrenamiento y validación, de cara a poder detectar y corregir un problema de ese estilo.

Además, en las Figuras 30, 31, y 32 se han eliminado de la figura los algoritmos EvolveGCN, MPNN-LSTM y STConv, debido a que la presencia de dichos resultados impedía el análisis correcto de los demás algoritmos.

Tal y como se ha mencionado en ocasiones anteriores, para este tipo de problemas no resulta adecuado estudiar únicamente los valores de las métricas a nivel global, calculando el promedio de todos los nodos. Esto es debido a que el objetivo, además de ser obtener un valor global de la función de pérdida lo más bajo posible, es que

dicho valor sea estable a través de los nodos. De esta manera, podríamos determinar si el modelo capta correctamente las interacciones entre los nodos, es decir, en el caso en el que un nodo se vea afectado por una incidencia en el generador, que sepa determinar de manera fiable qué ocurre en todos los demás.

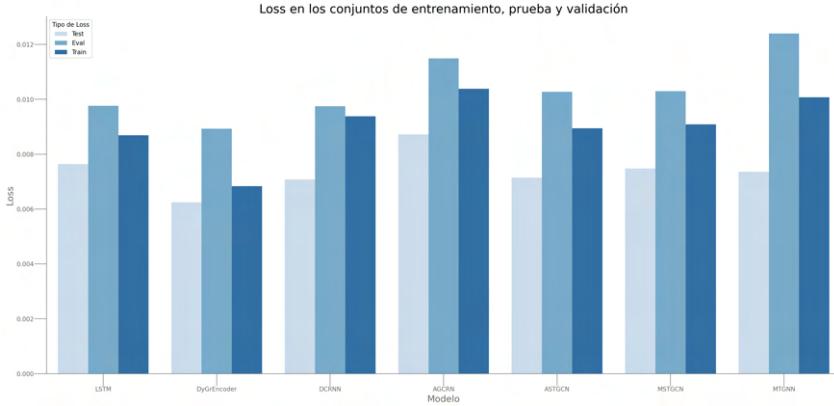


Figura 30: Resultados en el conjunto de test para los distintos modelos estudiados

Comprobar dicha estabilidad es una tarea más complicada, debido a la variación inherente entre los datos: primero entre los nodos, y después entre cada una de las simulaciones. Podemos obtener una idea sobre cómo varía la función de pérdida a lo largo de los nodos analizando la Figura 31, en la que se han representado los valores promedio de la función de pérdida por nodo y arquitectura. Podemos observar cómo el modelo DyGrEncoder tiene, en general, los valores más bajos y el menor contraste entre nodos. Asimismo, podemos determinar que el Nodo 211 es el que resulta más fácil de predecir correctamente para todos los modelos, mientras que el Nodo 154 es el más complicado.

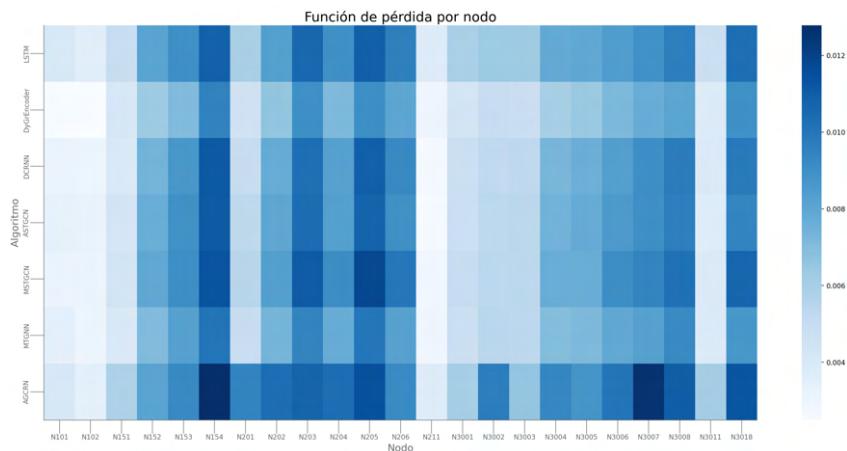


Figura 31: Función de pérdida por nodos y arquitecturas

En la Figura 31 podemos visualizar el contraste entre cada uno de los nodos, pero no obtenemos una comparación visual exacta de la variación intra-nodo. Para solucionarlo, podemos estudiar la Figura 32, donde se ha representado la varianza exacta

en cada arquitectura. Comprobamos que el modelo más adecuado para este tipo de problemas es el que utiliza una arquitectura DyGrEncoder.

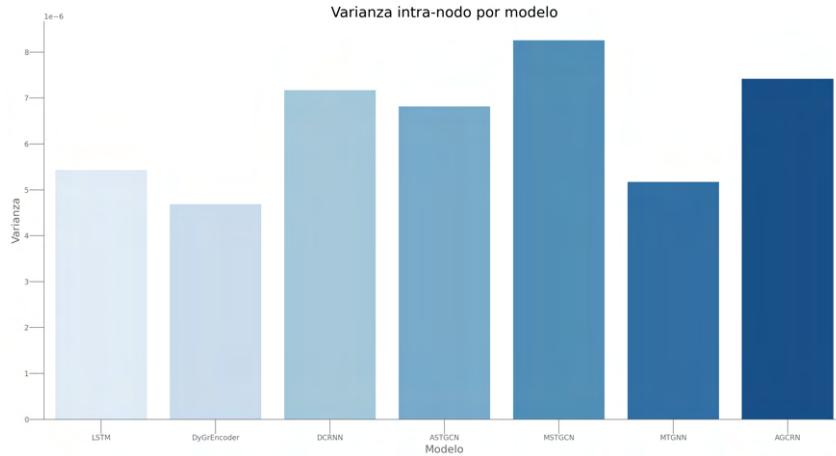


Figura 32: Variación intra-nodo de la función de pérdida por arquitecturas

9.2 CORTOCIRCUITO DE NODO

Continuamos con el segundo tipo de incidencia: el cortocircuito en el nodo. En la Tabla 2 se encuentran los resultados de todos los experimentos. Cabe destacar la diferencia en la métrica R₂ frente al tipo de incidencia anterior, pues en este caso obtenemos valores positivos en la mayoría de arquitecturas para los conjuntos de evaluación y prueba. Además, en este caso, el modelo con mejores resultados al considerar la función de pérdida en el conjunto de test es DCRNN. Por otro lado, si analizamos la métrica R₂, el mejor ajuste vendría dado por un modelo LSTM, lo cual nos lleva a concluir que el conocimiento de la estructura del grafo no implica necesariamente una mejora en el ajuste de la red.

Modelo	Train Loss	Test Loss	Eval Loss	Eval R ₂	Test R ₂	Tipo de Ajuste
Baseline						
LSTM	0.0055	0.0055	0.0054	0.9123	0.9231	Baseline
Recurrentes						
MPNN_LSTM	0.0058	0.0068	0.0063	0.8606	0.8459	BFB
DyGrEncoder	0.0044	0.0053	0.0060	0.8299	0.8274	BFB
EvolveGCN	<i>0.1027</i>	<i>0.0939</i>	<i>0.0926</i>	<i>-1.0828</i>	<i>-1.1974</i>	BFB
DCRNN	0.0041	0.0044	0.0041	0.9012	0.8896	BFB
AGCRN	0.0055	0.0071	0.0082	0.8295	0.8963	BFB
Atención						
MSTGCN	0.0060	0.0077	0.0068	0.8462	0.8216	BFB
MTGNN	0.0107	0.0069	0.0080	0.8041	0.8212	BA
ASTGCN	0.0253	0.0242	0.0216	0.2807	0.2744	BA
STConv	0.0310	0.0401	0.0360	0.1808	0.1469	BM

Cuadro 2: Resultados de experimentos para cortocircuito del nodo. Los mejores resultados en el conjunto de test están en negrita y los peores en cursiva.

Siguiendo un procedimiento similar al caso anterior, podemos observar en la Figura 33 la variación en el valor de la función de pérdida en cada uno de los conjuntos. En este caso, se han eliminado los algoritmos EvolveGCN, STConv y ASTGCN.

Es notable cómo el valor de la función de pérdida en el conjunto de prueba (barra del color claro) no sigue el mismo patrón que los otros conjuntos, sino que varía según el algoritmo considerado. En algunos modelos, como DyGrEncoder o AGCRN, el valor de la función de pérdida en el conjunto de entrenamiento es menor que en el conjunto de validación, pero mayor que en el conjunto de prueba. En otros casos, el valor en el conjunto de prueba es mayor que en los otros dos conjuntos.

No obstante, las diferencias observadas no son significativas. Además, considerando el estudio previo sobre la evolución de la función de pérdida en evaluación y entrenamiento, podemos descartar problemas de sobreajuste.

Si estudiamos el contraste en el Error Cuadrático Medio por nodos (Figura 34), vemos que en este caso el nodo más fácil de predecir vuelve a ser el Nodo 211, mientras que el que mayor error acumula es el Nodo 3007. Esto puede ser debido a la distribución de los errores considerados. En este caso vemos claramente cómo hay tres modelos con una variación más alta, y son MSTGCN, MTGNN y AGCRN. Al comprobar el valor exacto de la variación (Figura 35), podemos confirmar que éstos son los modelos menos estables entre los nodos. Además, también podemos concluir que una arquitectura DCRNN resulta adecuada en este caso.

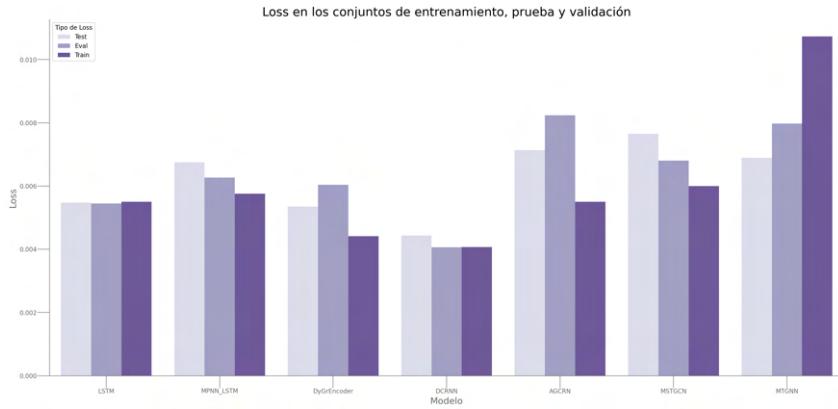


Figura 33: Resultados en el conjunto de test para los distintos modelos estudiados

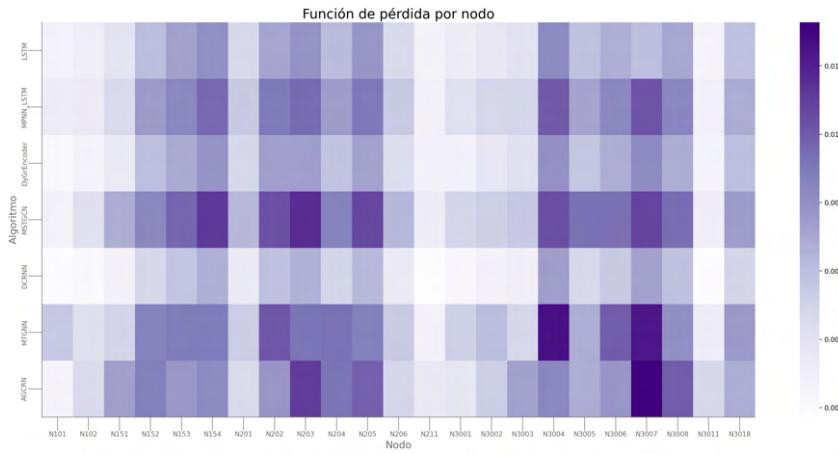


Figura 34: Función de pérdida por nodos y arquitecturas

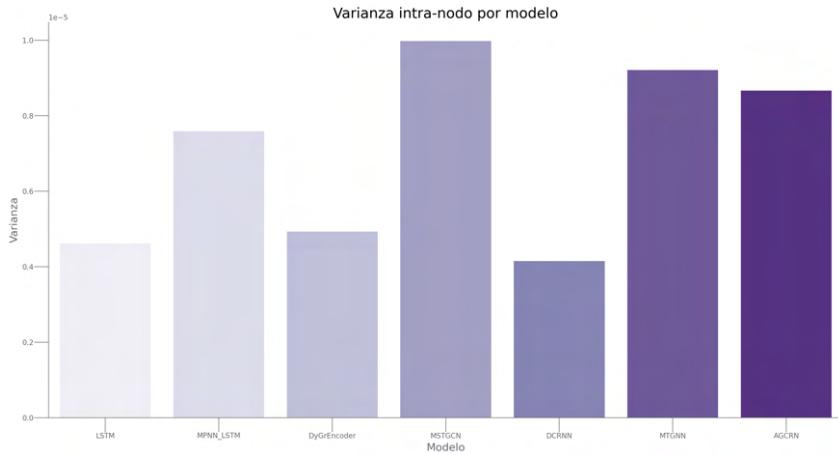


Figura 35: Variación intra-nodo de la función de pérdida por arquitecturas

9.3 ERROR DE NODO

Examinamos ahora los resultados del siguiente tipo de incidencia: el error en un nodo. En la Tabla 3 se encuentran los resultados generales de cada uno de los algoritmos. De nuevo, la mayoría de algoritmos tienen un rendimiento general peor que el modelo LSTM, excepto por uno, que es el modelo DyGrEncoder.

Modelo	Train Loss	Test Loss	Eval Loss	Eval R ₂	Test R ₂	Tipo de Ajuste
Baseline						
LSTM	0.0190	0.0173	0.0247	0.5413	0.6037	Baseline
Recurrentes						
MPNN_LSTM	0.0171	0.0167	0.0251	-1.9816	-0.6824	BFB
DyGrEncoder	0.0151	0.0164	0.0242	-1.7006	-0.6648	BFB
EvolveGCN	0.1487	0.1191	0.1560	-33.3613	-22.3205	BFB
DCRNN	0.0182	0.0167	0.0235	-2.3065	-0.8422	BFB
AGCRN	0.0207	0.0211	0.0265	0.5270	0.1874	BFB
Atención						
MSTGCN	0.0179	0.0184	0.0274	-2.1713	-0.9033	BFB
MTGNN	0.0224	0.0204	0.0236	-2.6749	-2.5251	BA
ASTGCN	0.0198	0.0173	0.0273	-2.1535	-0.6574	BA
STConv	0.0491	0.0618	0.0814	-13.8120	-11.3370	BM

Cuadro 3: Resultados de experimentos para error del nodo. Los mejores resultados en el conjunto de test están en negrita y los peores en cursiva.

En la Figura 36 se encuentra la comparación de la función de pérdida en cada uno de los tres subconjuntos considerados. En este caso, la mayoría de algoritmos presentan unos resultados parecidos: los resultados en el conjunto de prueba son mejores que en el conjunto de evaluación, lo cual de nuevo nos ayuda a descartar problemas de sobreajuste. Es necesario mencionar que en este tipo de incidencias la variabilidad entre simulaciones es muy alta, y puede haber coincidido que las simulaciones incluidas en el conjunto de evaluación sean muy distintas entre sí, provocando unos resultados peores en él.

Al estudiar la función de pérdida en cada uno de los nodos (Figura 37), y la variabilidad intra-nodo por algoritmos (Figura 38) vemos que los valores son relativamente coherente entre algoritmos, pues no se observan grandes variaciones entre ellos. Sin embargo, sí que hay uno que destaca, y es en el caso de AGCRN, en el que obtenemos por lo general peores resultados, con un contraste más alto entre nodos.

En este caso, no destaca especialmente ningún nodo como sistemáticamente más difícil de predecir, con un error mayor en cada uno de los algoritmos, sino que en general, los resultados son bastante consistentes.

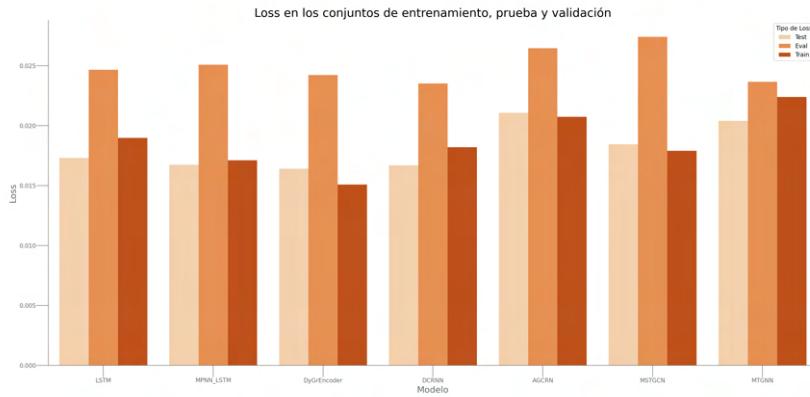


Figura 36: Resultados en el conjunto de test para los distintos modelos estudiados

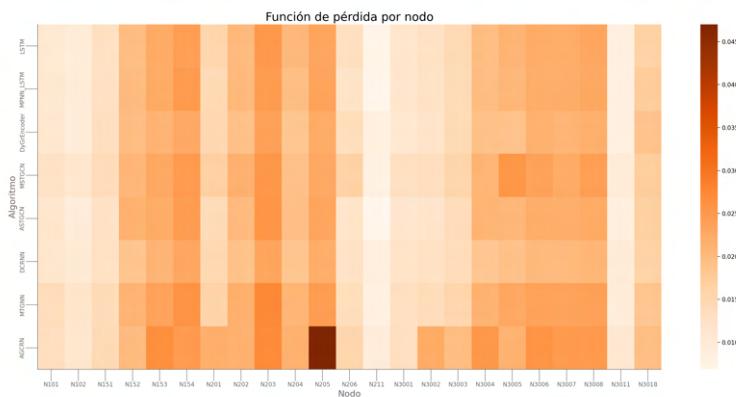


Figura 37: Función de pérdida por nodos y arquitecturas

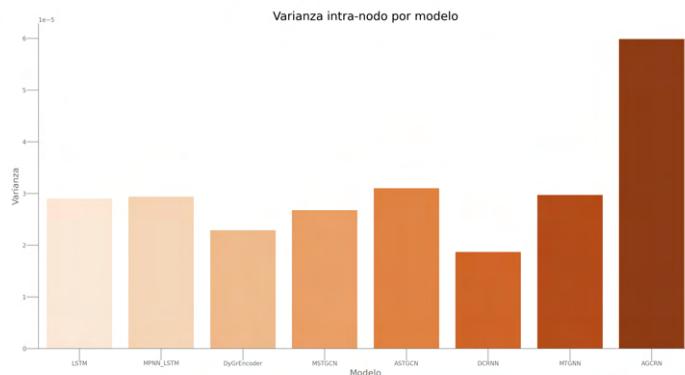


Figura 38: Variación intra-nodo de la función de pérdida por arquitecturas

9.4 CORTOCIRCUITO DE RAMA

Procedemos a analizar los resultados del siguiente tipo de incidencia: el cortocircuito de rama (Tabla 4). En este caso, vemos cómo los resultados son bastante consistentes

entre las arquitecturas, a excepción de EvolveGCN y STConv, como ya viene siendo lo normal. Sin embargo, es destacable mencionar los resultados negativos en la métrica R₂, que únicamente presenta resultados normales en el caso de LSTM.

Modelo	Train Loss	Test Loss	Eval Loss	Eval R ₂	Test R ₂	Tipo de Ajuste
Baseline						
LSTM	0.0033	0.0037	0.0037	0.4355	0.3989	Baseline
Recurrentes						
DyGrEncoder	0.0027	0.0027	0.0024	-0.4868	-0.5494	BFB
MPNN_LSTM	0.0039	0.0042	0.0036	<i>-1.1954</i>	<i>-1.4533</i>	BFB
EvolveGCN	<i>0.0383</i>	<i>0.0437</i>	<i>0.0377</i>	<i>-16.5207</i>	<i>-20.6751</i>	BFB
DCRNN	0.0038	0.0035	0.0036	<i>-1.3250</i>	<i>-1.3695</i>	BFB
AGCRN	0.0055	0.0067	0.0063	-0.1600	-0.3244	BFB
Atención						
MSTGCN	0.0036	0.0040	0.0036	<i>-1.1724</i>	<i>-1.4048</i>	BFB
MTGNN	0.0061	0.0038	0.0042	-0.0830	-0.5036	BA
ASTGCN	0.0034	0.0033	0.0030	<i>-0.8216</i>	<i>-0.9841</i>	BA
STConv	<i>0.0313</i>	<i>0.0365</i>	<i>0.0350</i>	<i>-16.6335</i>	<i>-18.0980</i>	BM

Cuadro 4: Resultados de experimentos para la evaluación de modelos. Los mejores resultados en el conjunto de test están en negrita y los peores en cursiva.

Si analizamos la diferencia de la función de pérdida en cada uno de los conjuntos estudiados (Figura 39), observamos que, en este caso, no existe una coherencia uniforme entre los distintos algoritmos. Esta falta de consistencia sugiere que las simulaciones incluidas en cada conjunto no presentan una disparidad tan marcada, a diferencia de lo observado en la incidencia anterior.

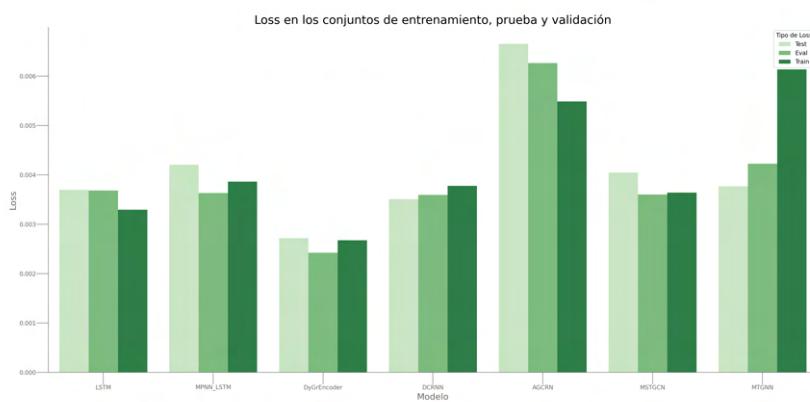


Figura 39: Resultados en el conjunto de test para los distintos modelos estudiados

Por otra parte, al analizar la variabilidad de la función de pérdida entre cada uno de los nodos (Figuras 40 y 41), confirmamos la bondad del ajuste del modelo DyGrEn-

coder, que no sólo presenta los mejores resultados a nivel global, sino que además es el que tiene unos resultados más consistentes entre los nodos.

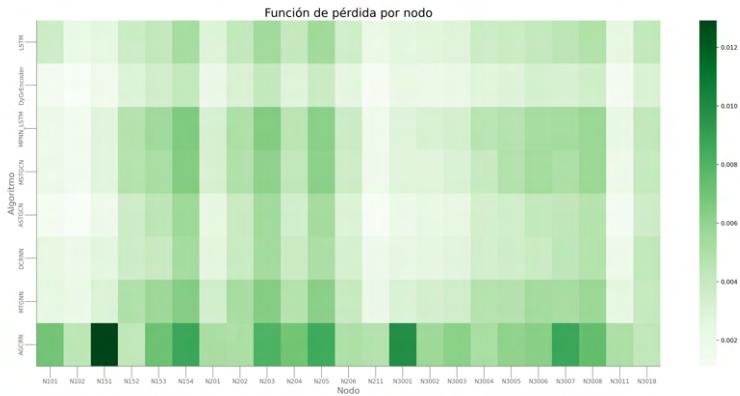


Figura 40: Función de pérdida por nodos y arquitecturas

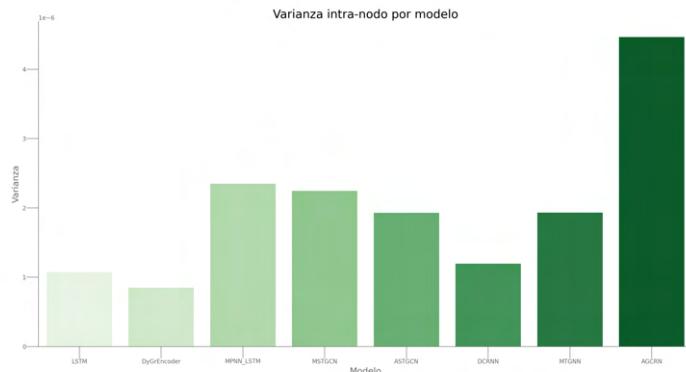


Figura 41: Variación intra-nodo de la función de pérdida por arquitecturas

9.5 ERROR DE RAMA

Damos paso al siguiente tipo de incidencia: el error de rama. Al analizar los resultados en la Tabla 5 vemos que en este caso, ninguno de los algoritmos estudiados consigue mejorar el rendimiento de una arquitectura LSTM, tanto a nivel de la función pérdida como a nivel de la métrica R₂.

Modelo	Train Loss	Test Loss	Eval Loss	Eval R ₂	Test R ₂	Tipo de Ajuste
Baseline						
LSTM	0.0190	0.0173	0.0247	0.5413	0.6037	Baseline
Recurrentes						
DyGrEncoder	0.0181	0.0206	0.0266	-2.7726	-1.6699	BFB
MPNN_LSTM	0.0202	0.0216	0.0282	-2.8993	-1.5651	BFB
EvolveGCN	0.1295	0.1116	0.1937	-19.8495	-13.3822	BFB
DCRNN	0.0196	0.0207	0.0259	-2.8389	-1.7440	BFB
AGCRN	0.0220	0.0277	0.0331	0.5360	0.5540	BFB
Atención						
MSTGCN	0.0223	0.0235	0.0305	-3.3846	-1.9988	BFB
MTGNN	0.0279	0.0176	0.0271	-3.6893	-0.0422	BA
ASTGCN	0.0232	0.0233	0.0314	-3.0665	-1.5871	BA
STConv	0.0772	0.0614	0.0788	-8.2722	-5.9585	BM

Cuadro 5: Resultados de experimentos para error de rama. Los mejores resultados en el conjunto de test están en negrita y los peores en cursiva.

Además, de nuevo, la función pérdida en el conjunto de prueba es menor que en el conjunto de validación, para todos los algoritmos considerados, tal y como se puede observar en la Figura 42. Sin embargo,

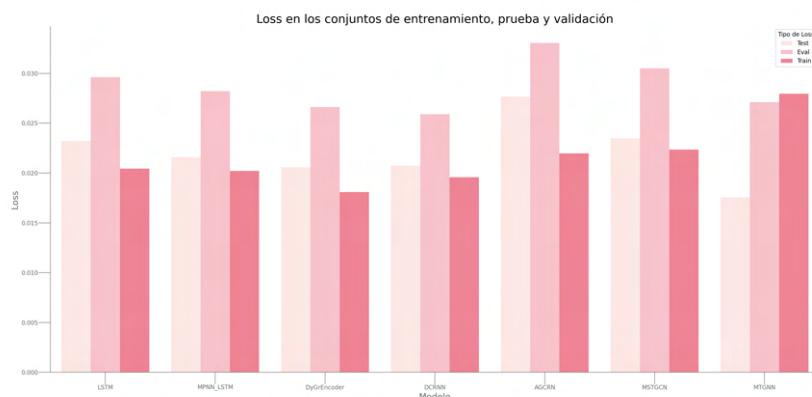


Figura 42: Resultados en el conjunto de test para los distintos modelos estudiados

Al estudiar la función pérdida por nodos, vemos que los resultados vuelven a ser bastante coherentes entre algoritmos, a excepción de la arquitectura AGCRN, que de nuevo presenta un contraste más elevado entre nodos (Figura 43).

Aunque en la Tabla 5 concluyéramos que ningún algoritmo mejora el rendimiento de LSTM, en la Figura 44 podemos observar cómo el modelo con menor variabilidad intra-nodo (Figura 44) es el que utiliza una arquitectura DCRNN.

Esta conclusión es interesante, porque permite valorar de una manera más completa el comportamiento de los modelos. Es fundamental que la evaluación de modelos sea coherente con el uso práctico y el problema considerado, y en este caso, tal y como se ha mencionado anteriormente, es importante que se reduzca la variabilidad en la función pérdida entre los nodos considerados. Si no se hubiera estudiado desde esta perspectiva, las conclusiones hubieran sido diferentes.

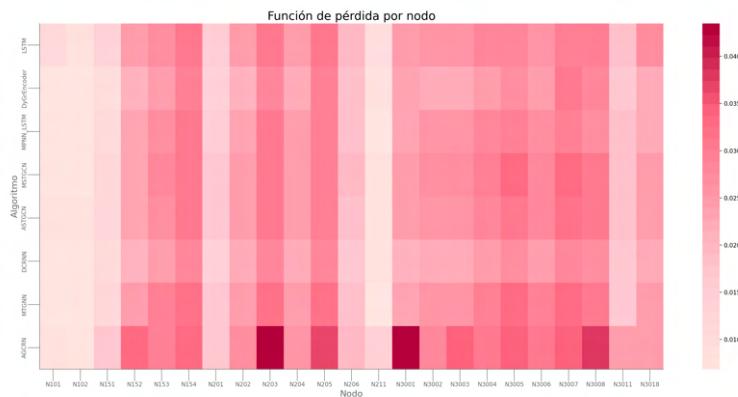


Figura 43: Función de pérdida por nodos y arquitecturas

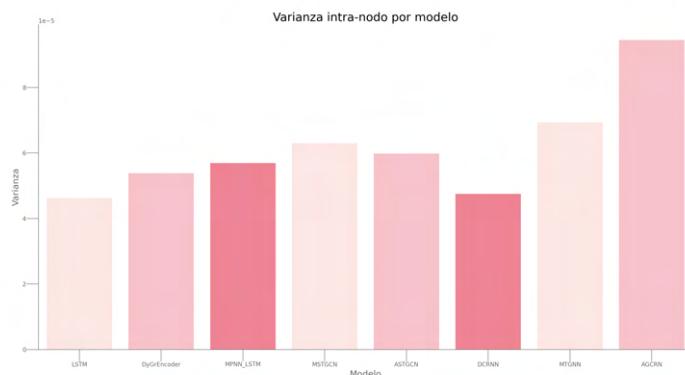


Figura 44: Variación intra-nodo de la función de pérdida por arquitecturas

9.6 CONCLUSIONES

A continuación vamos a repasar brevemente los resultados obtenidos en cada uno de los problemas.

Comenzamos por el **cortocircuito de generador**. En este problema, los mejores resultados en cuanto al valor de la función de pérdida y la variación intra-nodo se obtienen con una arquitectura DyGrEncoder. Sin embargo, se obtiene unos resultados negativos en la medida R² para todos los modelos, excepto LSTM, lo que nos puede llevar a cuestionar la bondad del ajuste.

En el caso del **cortocircuito de nodo**, se ha obtenido un ajuste mejor en cuanto a la función de pérdida y variación intra-nodo con un modelo DCRNN. Además, si analizamos la métrica R₂, en este caso obtenemos mejores resultados, con valores positivos cercanos a 1, y el mejor de ellos siendo un 0,92, con la arquitectura LSTM.

Respecto al caso de **error de nodo**, de nuevo, el mejor modelo en general resulta DyGrEncoder. Sin embargo, es destacable la diferencia en los resultados del conjunto de evaluación, posiblemente debido a la alta variabilidad en la estructura de las series temporales para las simulaciones de este conjunto. En este problema volvemos a encontrarnos con resultados negativos para la métrica R₂, excepto en el caso de LSTM.

En el caso del **cortocircuito de rama** los resultados son bastante consistentes entre las arquitecturas, destacando positivamente en cuanto a la función de pérdida y variación intra-nodo el modelo DyGrEncoder. De nuevo, se deben mencionar los resultados negativos en la métrica R₂, que únicamente presenta resultados normales en el caso de LSTM.

Por último, en el problema del **error de rama**, se ha obtenido como mejor modelo la arquitectura LSTM, tanto valorando la función de pérdida como la métrica R₂, seguido por un modelo DCRNN. Sin embargo, al analizar la variabilidad intra-nodo descubrimos que el modelo con menor variabilidad, y por tanto más consistente entre los nodos es DCRNN.

Podemos concluir, por lo tanto, que si nos fijamos exclusivamente en los resultados de la función de pérdida y la variabilidad intra-nodo, los mejores modelos son DCRNN, DyGrEncoder y LSTM. Si además incluimos la métrica R₂, deberíamos considerar únicamente LSTM, pues los resultados de las otras dos arquitecturas no son consistentes.

Por otra parte, si estudiamos cuál es el peor ajuste en cada uno de los modelos, obtenemos resultados consistentes, siendo el peor en todos el que utiliza una arquitectura EvolveGCN, seguido por un modelo STConv. Si bien es cierto que el modelo EvolveGCN no admite ajuste de parámetros, por lo que su rendimiento no puede ser mejorado, la arquitectura STConv sí que puede ajustarse, y por lo tanto se podrían mejorar los resultados obtenidos. Es necesario, sin embargo, tener en cuenta el coste computacional asociado a dicho ajuste, pues se trata del modelo que más tiempo ha tardado realizar el entrenamiento, y por lo tanto, sería necesario decidir si la supuesta mejora obtenida al ajustar los parámetros compensa el coste computacional.

10

CLASIFICACIÓN DEL TIPO DE INCIDENCIA

En este capítulo se presentarán los resultados obtenidos al poner a prueba los algoritmos introducidos previamente en este trabajo, aplicándolos a una tarea específica de predicción. En particular, el objetivo de estos experimentos es asociar correctamente el tipo de incidencia a cada una de las series temporales de grafos disponibles, utilizando para ello las arquitecturas de modelos desarrolladas.

Se busca asentar una base experimental sobre el problema de clasificación de incidencias, con el objetivo de probar el rendimiento de cada una de las arquitecturas presentadas en este tipo de problemas.

Para abordarlo, ha sido necesario realizar modificaciones en la arquitectura original de los modelos. En concreto, se ha calculado la media de las predicciones realizadas por cada nodo en el grafo, con el fin de generar una representación compacta de la predicción global. Posteriormente, esta representación ha sido alimentada a una capa lineal que actúa como un clasificador. Finalmente, se ha añadido una capa softmax en la salida del modelo, la cual tiene la función de establecer la probabilidad de pertenencia a cada una de las clases de incidencias posibles, permitiendo así la clasificación final.

En cuanto a la estructura de este capítulo, además de estudiar los resultados generales (Sección 10.1), vamos a estudiar los resultados individuales de cada algoritmo con más detalle (Sección 10.2).

Cabe mencionar que en este análisis no se incluirán los resultados obtenidos con el algoritmo STConv. La razón de esta omisión radica en la alta demanda computacional de dicho algoritmo, que excedió las capacidades disponibles para este trabajo. Aunque STConv fue probado inicialmente, los tiempos de entrenamiento y la necesidad de recursos computacionales imposibilitaron su uso efectivo en este conjunto de experimentos. Por lo tanto, los resultados presentados se centrarán en los otros algoritmos que sí pudieron ser evaluados de manera completa y eficiente.

De nuevo, en la Figura 45 se puede estudiar el procedimiento experimental en su totalidad. El procedimiento es muy similar al caso de regresión, debido a que el software de preprocesamiento y entrenamiento fue desarrollado de una manera adaptativa, para poder escoger qué tipo de problema se quiere resolver.

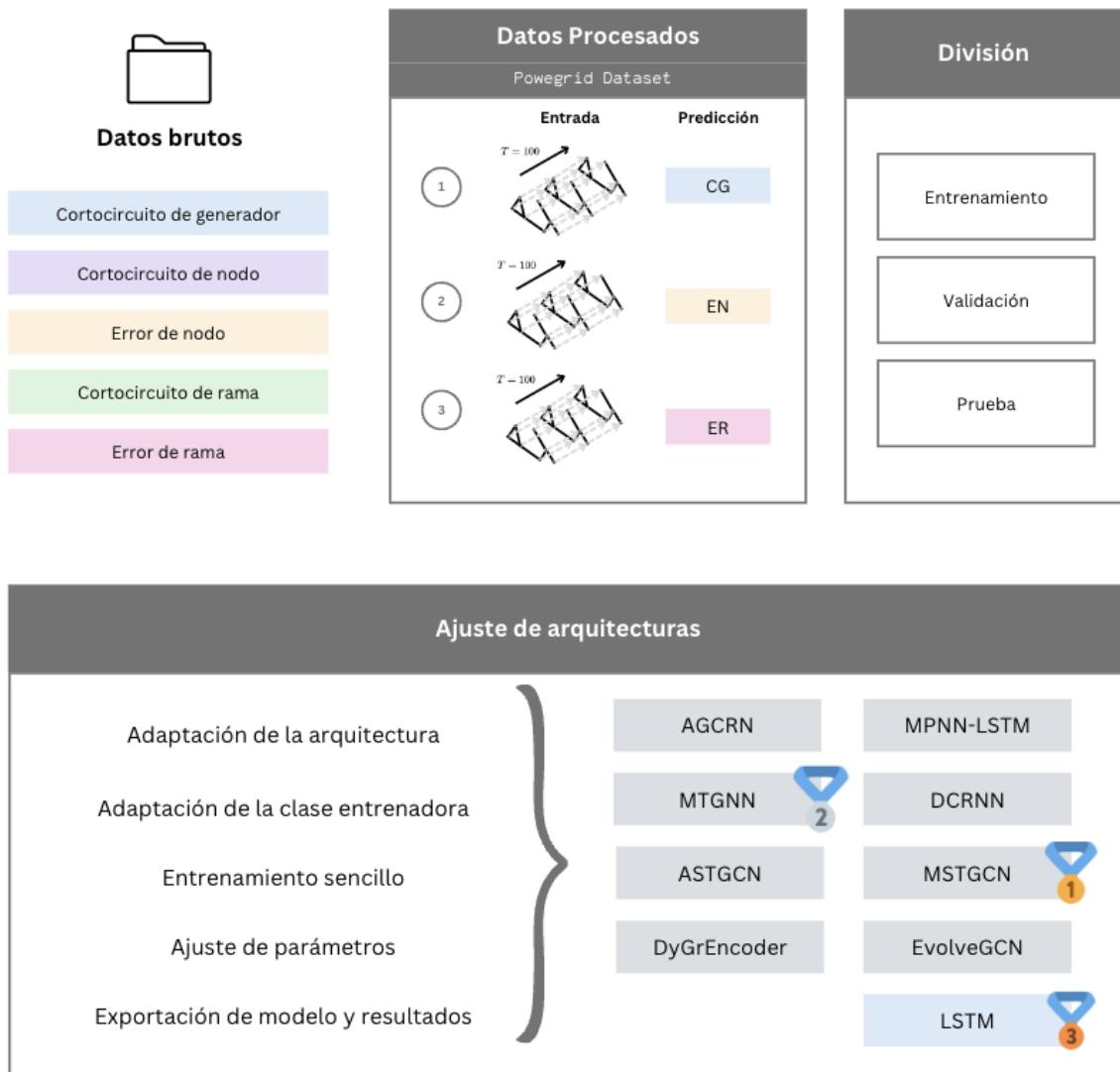


Figura 45: Flujo experimental completo

10.1 RESULTADOS GENERALES

Comenzamos estudiando los resultados generales, expuestos en la Tabla 6.

Modelo	Loss		Accuracy		Precision		Recall		F1 Score	
	Test	Eval								
Baseline										
LSTM	1.163	1.213	0.766	0.672	0.706	0.631	0.699	0.671	0.699	0.635
Recurrentes										
MPNN_LSTM	1.349	1.287	0.536	0.600	0.338	0.439	0.476	0.498	0.388	0.435
AGCRN	1.344	1.294	0.548	0.583	0.327	0.399	0.496	0.482	0.390	0.421
DyGrEncoder	1.278	1.244	0.631	0.627	0.586	0.523	0.576	0.498	0.575	0.505
DCRNN	1.366	1.297	0.548	0.600	0.327	0.439	0.496	0.498	0.390	0.419
Atención										
MSTGCN	1.116	1.170	0.810	0.747	0.731	0.700	0.721	0.720	0.717	0.707
MTGNN	1.065	1.164	0.843	0.732	0.673	0.585	0.747	0.680	0.699	0.622
ASTGCN	1.402	1.336	0.494	0.573	0.283	0.325	0.470	0.481	0.347	0.382

Cuadro 6: Resultados de las métricas de clasificación para diferentes modelos. Los mejores valores de cada métrica están en negrita y los peores en cursiva.

En el caso de clasificación la función de pérdida escogida para entrenar los modelos es la función de entropía cruzada (*cross entropy loss*). Esta función mide la diferencia entre dos distribuciones de probabilidad, y se puede estudiar en más detalle en el Apéndice A.

Un valor de entropía cruzada cercano a 0 indica que el modelo está realizando predicciones que están muy alineadas con las etiquetas verdaderas. La primera observación que podemos realizar al estudiar la tabla de resultados es que los valores de la función pérdida en el conjunto de evaluación y prueba no son positivos, puesto que todos están por encima de 1. Tal y como se ha mencionado en varias ocasiones, la forma de las series temporales es muy variada, y no se pueden extraer diferencias sistemáticas entre clases a simple vista. Sin embargo, resulta interesante comprobar si pueden existir diferencias subyacentes, que un modelo entrenado sí sea capaz de descubrir.

Para comparar el comportamiento de cada una de las arquitecturas, consideraremos además otro tipo de métricas: la precisión, exactitud, recall y F1.

La precisión mide la proporción de predicciones verdaderas sobre el total de observaciones predichas en cada clase. La exactitud, o *accuracy*, representa la proporción de predicciones correctas (tanto positivas como negativas) sobre el total de predicciones, proporcionando una visión general del rendimiento del modelo en todas las clases. El recall, también conocido como sensibilidad o *recall*, mide la proporción de verdaderos positivos sobre el total de positivos reales, evaluando la capacidad del modelo para detectar correctamente los casos positivos. Finalmente, la métrica F1 combina precisión y recall en una sola medida, calculando su media armónica. La F1 es espe-

cialmente útil cuando se busca un balance entre precisión y recall, siendo efectiva en escenarios con clases desbalanceadas.

Es destacable como el mejor modelo en cada una de las tres métricas es el modelo MSTGCN, mientras que uno de los peores es la versión compleja del mismo, ASTGCN. Esto nos lleva a comprender la importancia del ajuste de parámetros en cada una de las arquitecturas, puesto que no conocemos el rendimiento de la arquitectura ASTGCN en el caso de haber sido ajustado con una búsqueda por fuerza bruta.

En la Figura 46 podemos observar visualmente el valor de la función de entropía cruzada en cada uno de los conjuntos considerados. En este caso, el rendimiento en el conjunto de prueba es ligeramente inferior al resto de conjuntos, lo cual es un comportamiento normal en este tipo de modelos.

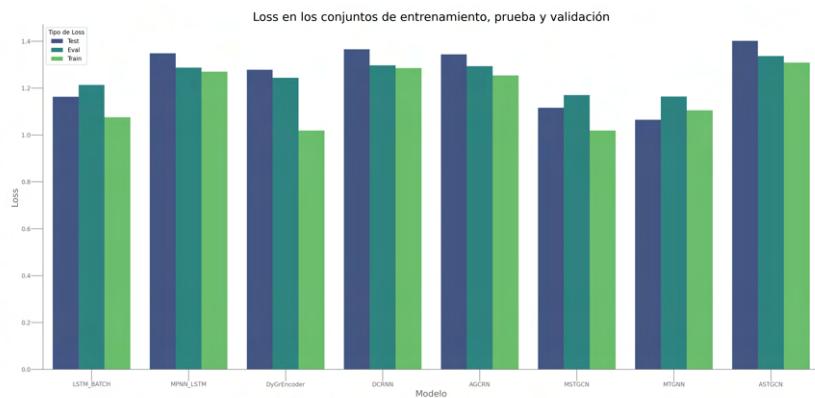


Figura 46: Función de pérdida en conjuntos de entrenamiento, validación y prueba

10.2 RESULTADOS POR ALGORITMO

En esta sección vamos a adentrarnos en los resultados detallados de cada uno de los experimentos. Para cada uno de ellos, estudiaremos la matriz de confusión obtenida al realizar las predicciones sobre el conjunto de prueba, y la precisión por clase.

Comenzamos estudiando cómo sería una clasificación mediante un bloque LSTM (Figura 47), pues el objetivo del trabajo es estudiar si la inclusión de mecanismos avanzados, con conocimiento de las interacciones de la red, supone una mejora frente a un modelo más sencillo. En este caso, vemos cómo el bloque LSTM realiza un buen trabajo separando las clases de cortocircuito de generador, de rama, y de nodo, y error de rama, pero que confunde la clase de error de nodo con las demás. Veamos qué ocurre con los modelos avanzados.

En la Figura 48 se muestran los resultados del algoritmo AGCRN. En ella, se observa que el algoritmo no logra identificar correctamente las incidencias que implican un fallo en un nodo. Esto es consistente con una observación previa: ambas clases presentan una gran variabilidad en las formas de los voltajes de cada simulación, lo que



Figura 47: LSTM

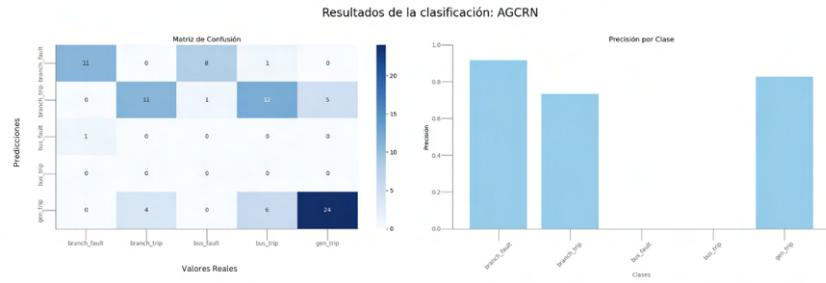


Figura 48: AGCRN

dificulta que los modelos puedan diferenciarlas adecuadamente. Además, esta misma conclusión puede obtenerse en el caso de los algoritmos MPNN-LSTM, DCRNN y ASTGCN (Figuras 50, 51 y 52, respectivamente), lo que nos lleva a confirmar que existe una dificultad añadida en estas clases, debido a la inconsistencia entre simulaciones. Sin embargo, vemos cómo los errores de predicción son entre ambas clases, es decir, que no interfieren demasiado con las predicciones de las otras clases, consiguiendo buenos resultados en ellas.

Por otra parte, si estudiamos la Figura 49, que muestra los resultados del DyGrEncoder, se observa un mejor desempeño. Aunque sigue mostrando una precisión baja en las dos clases mencionadas anteriormente, logra una alta precisión en las clases de cortocircuito de generador y cortocircuito de rama.

Resulta interesante estudiar los resultados de los modelos MSTGCN y MTGNN (Figuras 53 y 54), donde por primera vez obtenemos muy buenos resultados en la clase de cortocircuito de nodo. Sin embargo, en ambos casos se confunden las simulaciones de error de nodo con las simulaciones de todas las clases, en vez de confundirse únicamente con las de cortocircuito de nodo, como veíamos anteriormente.

En general los resultados obtenidos en la clasificación no son positivos, pues no hemos obtenido un modelo lo suficientemente preciso para todas las clases. Sin embargo, sí que se han obtenido resultados interesantes, que podrían sentar las bases para una investigación futura, planteando, por ejemplo, modelos binarios, que separen una clase de incidencia concreta frente al resto.

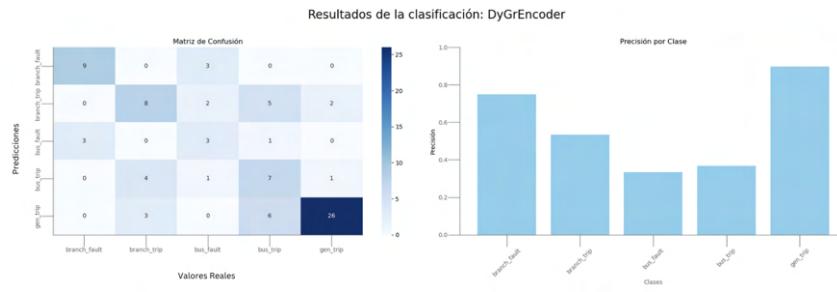


Figura 49: DyGrEncoder

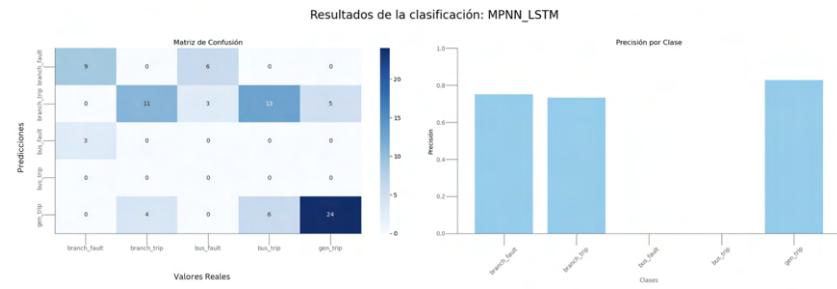


Figura 50: MPNN-LSTM

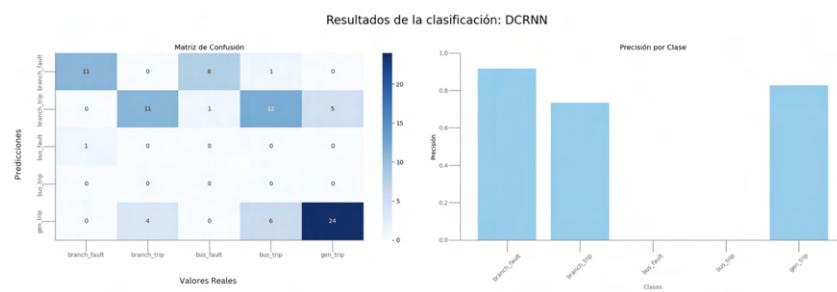


Figura 51: DCRNN



Figura 52: ASTGCN

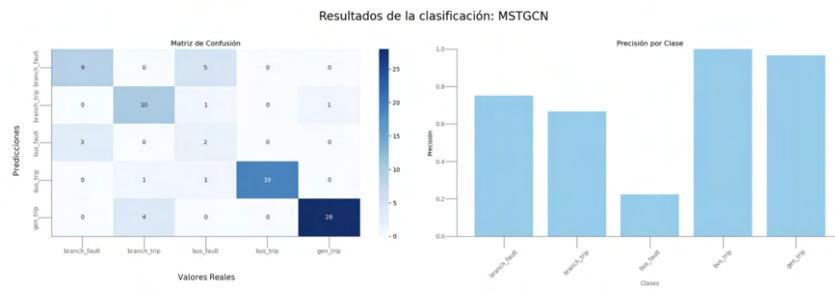


Figura 53: MSTGCN

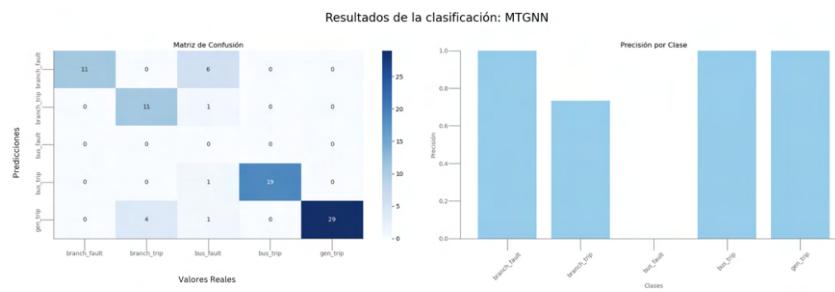


Figura 54: MTGNN

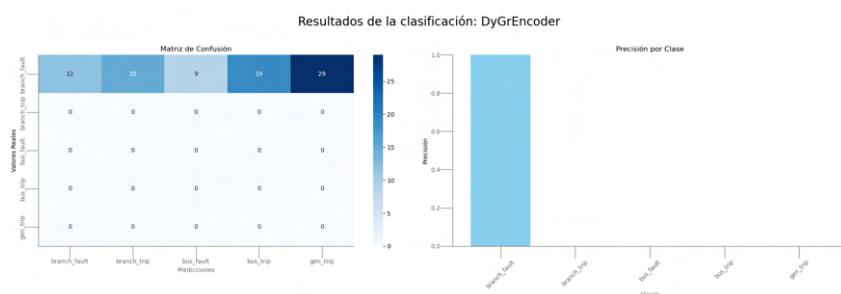


Figura 55: EvolveGCN

CONCLUSIONES Y TRABAJO FUTURO

Tras abordar el problema de predicción del voltaje de cada nodo de la red eléctrica en varios tipos de incidencia, podemos afirmar que los resultados más consistentes se obtienen con una arquitectura LSTM sencilla, es decir, que, aunque no se obtengan sistemáticamente los mejores resultados en cada tipo de incidencia, sí que se producen predicciones relativamente buenas en todas ellas con este tipo de arquitectura.

Sin embargo, es necesario mencionar que para cada incidencia, se ha obtenido al menos un modelo con un rendimiento superior al de LSTM, lo cual nos lleva a pensar en que los modelos avanzados, que tratan de manera explícita las interacciones entre los nodos de la red, pueden producir resultados mejores, especialmente en situaciones donde dichas interacciones resultan fundamentales, como por ejemplo un fallo en cascada.

En este caso es fundamental mencionar la complejidad de los datos: a lo largo del trabajo se ha mencionado en numerosas ocasiones la disparidad en el comportamiento de los nodos en diversas simulaciones, así como a través de los distintos tipos de incidencias. Este problema convierte el trabajo de predicción con modelos avanzados en una tarea más compleja, pues muchos de ellos fueron diseñados originalmente en el contexto de la predicción del tráfico en una ciudad, y las series temporales asociadas a ese problema presentan una variabilidad mucho menor. Además, el tamaño de la red es bastante reducido, y las interacciones entre los nodos son sencillas, lo que puede ser un factor determinante para obtener mejor rendimiento con un modelo sin conocimiento explícito de la arquitectura de la red.

Es por tanto natural pensar que una de las líneas de trabajo futuro consiste en estudiar el comportamiento de los algoritmos avanzados con una red más grande, en la que las interacciones entre los nodos sean más complejas. No obstante, una de las limitaciones principales de este campo es la disponibilidad de datos, siendo el dataset utilizado el único que se ajustaba a los requerimientos para poder realizar predicciones de este tipo, por lo que aumentar el número de colecciones de datos resulta fundamental para poder establecer conclusiones correctas sobre el potencial funcionamiento de las arquitecturas.

Por otro lado, también se ha estudiado el rendimiento de los algoritmos en una tarea de clasificación de incidencias, obteniendo en este caso una conclusión similar: existen arquitecturas que mejoran ligeramente los resultados de LSTM, pero implican una

complejidad de entrenamiento muy superior. De nuevo, sería interesante comprobar el rendimiento de dichas arquitecturas sobre un dataset más grande y con una mayor estabilidad en la representación de cada incidencia. Esto permitiría evaluar si el aumento en la complejidad del entrenamiento se justifica por una mejora significativa en la precisión y robustez del modelo, especialmente en escenarios más realistas y variados.

Es importante notar que, si bien las arquitecturas utilizadas estaban parcialmente desarrolladas, ha sido necesario adaptar cada una de ellas para poder incluir la información sobre las interacciones en la red. Por una parte, esto ha supuesto una dificultad añadida al trabajo, pues ha sido necesario entender perfectamente cada parte de las arquitecturas presentadas, siendo muy distintas entre sí. No obstante, por otra parte ha sido una oportunidad de generar un software modular, de manera que cada uno de los algoritmos pudiera encajar con la clase entrenadora principal. El software puede ser modificado de manera sencilla para poder incluir nuevos conjuntos de datos o nuevas arquitecturas.

Finalmente, mencionar que, si bien se han incluido los modelos principales implementados en la librería *Pytorch Geometric Temporal*, existen multitud de arquitecturas que utilizan los fundamentos introducidos en este trabajo. Se plantea como trabajo futuro, por lo tanto, incluir nuevas arquitecturas y comparar el rendimiento con las ya implementadas.

Este trabajo, junto con el software desarrollado, se ponen a disposición de los expertos en el campo, con la esperanza de que puedan ser utilizadas y adaptadas para abordar otros problemas relacionados con la predicción de series temporales en el ámbito de las incidencias en redes eléctricas. Se espera que este trabajo sirva como base para futuras investigaciones y desarrollos en esta área, contribuyendo así al avance del conocimiento y las soluciones en un campo tan relevante como la gestión de la energía.

BIBLIOGRAFÍA

- [1] Kashif Abbass, Muhammad Zeeshan Qasim, Huaming Song, Muntasir Murshed, Haider Mahmood, and Ijaz Younis. A review of the global climate change impacts, adaptation, and sustainable mitigation measures. *Environ. Sci. Pollut. Res. Int.*, 29(28):42539–42559, 2022.
- [2] Charu C Aggarwal. *Neural networks and deep learning: A textbook*. Springer International Publishing, Cham, 2023. ISBN 978-3-031-29641-3.
- [3] Ethem Alpaydin. *Machine Learning*. The MIT Press, August 2021. ISBN 9780262542524.
- [4] M Arbib. Review of 'perceptrons: An introduction to computational geometry' (minsky, m., and papert, s.; 1969). *IEEE Trans. Inf. Theory*, 15(6):738–739, 1969.
- [5] Andrea Asztalos, Sameet Sreenivasan, Boleslaw K Szymanski, and Gyorgy Korniss. Cascading failures in spatially-embedded random networks. *PLoS One*, 9(1):e84563, 2014.
- [6] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada, 2020.
- [7] Andrey Bernstein, Daniel Bienstock, David Hay, Meric Uzunoglu, and Gil Zussman. Power grid vulnerability to geographically correlated failures - analysis and control implications. 2012.
- [8] Hongjie Chen and Hoda Eldardiry. Graph time-series modeling in deep learning: A survey. *ACM Trans. Knowl. Discov. Data*, 18(5):1–35, 2024.
- [9] Michael Chertkov, Feng Pan, and Mikhail G Stepanov. Predicting failures in power grids: The case of static overloads. *IEEE Trans. Smart Grid*, 2(1):162–172, 2011.
- [10] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine Learning Techniques for Multimedia*, pages 21–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [11] Alexander L Fradkov. Early history of machine learning. *IFAC-PapersOnLine*, 53(2):1385–1390, 2020.
- [12] J Galindo and P Tamayo. Credit risk assessment using statistical and machine learning: Basic methodology and risk modeling applications. *Comput. Econ.*, 15(1/2):107–143, 2000.

- [13] Zoubin Ghahramani. Unsupervised learning. In *Advanced Lectures on Machine Learning*, pages 72–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
- [14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [15] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. *Proc. Conf. AAAI Artif. Intell.*, 33(01):922–929, 2019.
- [16] Sudha Gupta, Ruta Kambli, Sushama Wagh, and Faruk Kazi. Support-vector-machine-based proactive cascade prediction in smart grid using probabilistic framework. *IEEE Trans. Ind. Electron.*, 62(4):2478–2486, 2015.
- [17] William L Hamilton. *Graph Representation Learning*. Springer International Publishing, Cham, 2020. ISBN 978-3-031-00460-5.
- [18] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. 2017.
- [19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, 1989.
- [20] Peter J Huber. Robust estimation of a location parameter. *Ann. Math. Stat.*, 35(1): 73–101, 1964.
- [21] Intergovernmental Panel on Climate Change (IPCC). Emissions trends and drivers. In *Climate Change 2022 - Mitigation of Climate Change: Working Group III Contribution to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*, pages 215–294. Cambridge University Press, 2023.
- [22] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning: With applications in R*. Springer US, New York, NY, 2021.
- [23] Sevvandi Kandanaarachchi, Ziqi Xu, and Stefan Westerlund. Predicting the structure of dynamic graphs. 2024. doi: 10.48550/arXiv.2401.04280.
- [24] Jia-Ning Kang, Yi-Ming Wei, Lan-Cui Liu, Rong Han, Bi-Ying Yu, and Jin-Wei Wang. Energy systems for climate change mitigation: A systematic review. *Appl. Energy*, 263(114602):114602, 2020.
- [25] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2016.
- [26] Angel Esteban Labrador Rivas and Taufik Abrão. Faults in smart grid systems: Monitoring, detection and classification. *Electric Power Syst. Res.*, 189(106602): 106602, 2020.
- [27] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. 2017.

- [28] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Trans. Intell. Transp. Syst.*, 22(7):4316–4336, 2021.
- [29] Monika A Myszczynska, Poojitha N Ojamies, Alix M B Lacoste, Daniel Neil, Amir Saffari, Richard Mead, Guillaume M Hautbergue, Joanna D Holbrook, and Laura Ferraiuolo. Applications of machine learning to diagnosis and treatment of neurodegenerative diseases. *Nat. Rev. Neurol.*, 16(8):440–456, 2020.
- [30] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer graph neural networks for pandemic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(6), pages 4838–4845, 2021. doi: 10.1609/aaai.v35i6.16616.
- [31] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B Schardl, and Charles E Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5363–5370, 2020.
- [32] A G Phadke and J S Thorp. Expose hidden failures to prevent cascading outages [in power systems]. *IEEE Comput. Appl. Power*, 9(3):20–23, 1996.
- [33] Saleh Soltan, Dorian Mazauric, and Gil Zussman. Analysis of failures in power grids. *IEEE Trans. Control Netw. Syst.*, 4(2):288–300, 2017.
- [34] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.
- [35] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 301–307, New York, NY, USA, 2019. ACM.
- [36] UNFCCC. Conference of the Parties (COP). Adoption of the Paris Agreement. Proposal by the President, 2015. Accessed: 2024-7-15.
- [37] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, and Shanrong Zhao. Applications of machine learning in drug discovery and development. *Nat. Rev. Drug Discov.*, 18(6):463–477, 2019.
- [38] Anna Varbella, Blazhe Gjorgiev, and Giovanni Sansavini. Geometric deep learning for online prediction of cascading failures in power grids. *Reliab. Eng. Syst. Saf.*, 237(109341):109341, 2023.
- [39] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. 2018.

- [40] Fan Wenli, Liu Zhigang, Hu Ping, and Mei Shengwei. Cascading failure model in power grids using the complex network theory. *IET Gener. Transm. Distrib.*, 10(15):3940–3949, 2016.
- [41] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 753–763, New York, NY, USA, 2020. ACM.
- [42] Hongda Xiao and Edmund M Yeh. Cascading link failure in the power grid: A percolation-based analysis. In *2011 IEEE International Conference on Communications Workshops (ICC)*, Kyoto, Japan, 2011. doi: [10.1109/iccw.2011.5963573](https://doi.org/10.1109/iccw.2011.5963573).
- [43] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. 2020. doi: [10.48550/arXiv.2002.07962](https://arxiv.org/abs/2002.07962).
- [44] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-Temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, California, 2018. International Joint Conferences on Artificial Intelligence Organization.
- [45] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 2017.
- [46] Xiangtian Zheng, Nan Xu, Loc Trinh, Dongqi Wu, Tong Huang, S Sivarajani, Yan Liu, and Le Xie. A multi-scale time-series dataset with benchmark for machine learning in decarbonized energy grids. *Sci. Data*, 9(1):359, 2022.
- [47] Yuhong Zhu, Xiaoming Liu, Bo Chen, Donglei Sun, Dong Liu, and Yongzhi Zhou. Identification method of cascading failure in high-proportion renewable energy systems based on deep learning. *Energy Rep.*, 8:117–122, 2022.

Parte V

APÉNDICES

Exposición detallada de las predicciones obtenidas en cada una de las arquitecturas.

A

REDES NEURONALES

A continuación vamos a presentar las Redes Neuronales, un modelo inspirado en el funcionamiento del cerebro humano que ha demostrado ser especialmente eficaz en una amplia gama de tareas de Aprendizaje Supervisado, desde la clasificación de imágenes hasta la generación de texto, mediante el uso de unidades computacionales que buscan comportarse de una manera similar a las neuronas humanas.

Es necesario mencionar que para exponer los conceptos e ideas fundamentales de las Redes Neuronales, nos basaremos en el libro *Neural Networks and Deep Learning*, escrito por Charu C. Aggarwal [2].

En la siguiente sección introduciremos la intuición de las redes neuronales. Luego, analizaremos en detalle los componentes de las mismas mediante el uso de la red neuronal más simple que existe: el perceptrón. Posteriormente, exploraremos las decisiones de diseño que se deben tomar para modelar problemas con redes, y finalmente estudiaremos en detalle el algoritmo de entrenamiento *backpropagation*.

A.1 FUNDAMENTOS DE LAS REDES NEURONALES

Tal y como hemos mencionado, las Redes Neuronales buscan imitar el funcionamiento del sistema nervioso humano, mediante el uso de unidades de computación denominadas *neuronas*, que se conectan unas a otras con pesos determinados. Cada neurona almacena en su interior una variable que es el resultado de una función o cálculo, o que viene del exterior, en el caso de las neuronas de entrada.

En una red se obtienen las salidas mediante la propagación de los datos de entrada a través de las neuronas, usando los pesos específicos de los enlaces como parámetros intermedios. El aprendizaje comienza una vez se ha obtenido una salida, mediante la comparación de dicha salida con los datos reales, y el ajuste posterior de los pesos de la red, mediante un algoritmo denominado *backpropagation*.

Podríamos comparar el ajuste de los pesos como respuesta al error de predicción con una respuesta negativa en un organismo vivo, que hace que se ajusten las fuerzas sinápticas, los pesos entre las neuronas de nuestro sistema nervioso. Aunque pueda

parecer evidente, es necesario destacar que la representación computacional de las redes supone una simplificación de lo que ocurre en nuestro organismo.

Así, el objetivo de una red neuronal es aprender una función que relacione las variables de entrada con las variables de salida mediante la iteración en el conjunto de entrenamiento.

Vamos a considerar que estamos ante un problema de clasificación, es decir, tenemos un conjunto con d entradas y una única salida Y , que tiene dos valores posibles, y que denominaremos *valor observado*. Por lo tanto, cada observación del conjunto de entrenamiento tiene la forma (\bar{X}, Y) , donde

$$\begin{aligned}\bar{X} &= [x_1, \dots, x_d] \\ Y &= y \quad y \in \{-1, 1\}\end{aligned}$$

Nuestro objetivo, por lo tanto, es predecir el valor de Y para observaciones nuevas, diremos, por lo tanto, que Y es la variable objetivo. Esto es equivalente a decir que buscamos una función $f(\cdot)$, donde

$$y = f_{\bar{W}}(\bar{X}) \tag{A.1.1}$$

donde \bar{X} son las d variables de entrada y \bar{W} son los pesos o parámetros que se tienen que computar para poder construir la función.

Tal y como veíamos en los problemas de regresión o clasificación, el aprendizaje pasa por reducir la diferencia entre los valores reales y las predicciones, esto es, entre y y $f_{\bar{W}}(\bar{X})$, mediante el ajuste de los pesos.

Ahora que ya hemos introducido brevemente la intuición tras el aprendizaje con redes neuronales, vamos a estudiar como ejemplo la red neuronal más sencilla posible, denominada perceptrón.

A.1.1 *Ejemplo. Red Neuronal Simple.*

Esta red neuronal está formada por una única capa de entrada, con d neuronas, y una capa de salida con un único nodo. Todas las neuronas de la capa de entrada están conectadas a la neurona de salida mediante vértices, cuyos pesos vienen dados por $\bar{W} = [w_1, \dots, w_d]^T$.

Además, en el nodo de salida se produce la siguiente computación:

$$\bar{W} \cdot \bar{X}^T = \sum_{i=1}^d w_i x_i \tag{A.1.2}$$

Y la predicción final \hat{y} vendrá dada por el signo de este último valor, es lo que denominaremos *función de activación*. La estructura de esta red neuronal puede consultarse en la Figura 56.

$$\hat{y} = f(\bar{X}) = \text{sign} \left\{ \bar{W} \cdot \bar{X}^T \right\} = \text{sign} \left\{ \sum_{j=1}^d w_j x_j \right\} \quad (\text{A.1.3})$$

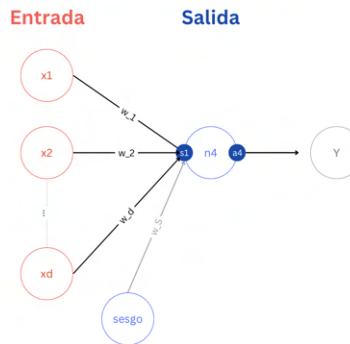


Figura 56: Estructura del perceptrón

Al inicio del entrenamiento, los pesos tienen valores aleatorios, y por lo tanto, las predicciones iniciales serán aleatorias también, y en consecuencia, pobres. Se suele definir una función de pérdida que cuantifique el coste cuando la predicción \hat{y} difiere del valor observado y , el ajuste de los pesos se realizará con el objetivo de minimizar el coste total a lo largo de todo el conjunto de entrenamiento.

Hay una limitación con esta aproximación, y es que en ocasiones hay una parte invariante de la predicción, denominada sesgo. Si consideramos el caso en el que las variables de entrada \bar{X} tienen media 0, pero las variables de salida Y no la tienen, entonces resulta evidente que la aproximación anterior no es correcta, pues debería darse la siguiente igualdad:

$$\underbrace{\bar{W} \cdot \sum_i \bar{X}_i^T}_{=0} \neq \underbrace{\sum_i y_i}_{\neq 0} \quad (\text{A.1.4})$$

En casos como esos, es necesario incluir el sesgo, que no es más que una neurona con valor fijo 1, pero que añade un peso ajustable al conectar con la capa de salida. La incorporación del sesgo hace que la forma final de las predicciones sea:

$$\underbrace{\bar{W} \cdot \sum_i \bar{X}_i^T}_{=0} \neq \underbrace{\sum_i y_i}_{\neq 0} \quad (\text{A.1.5})$$

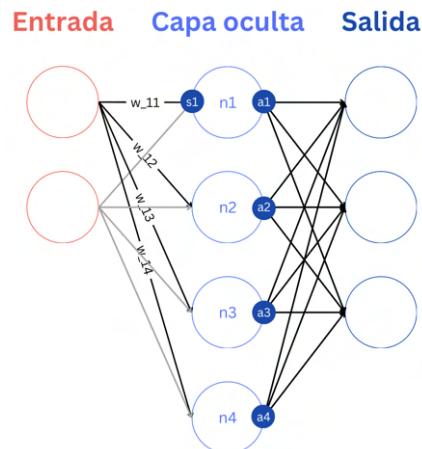


Figura 57: Arquitectura de una red más compleja

Resumiendo todo lo que hemos ido introduciendo: en una red neuronal, en particular el perceptrón, podemos diferenciar las siguientes componentes dentro de la arquitectura de una red neuronal:

- **Un estado de activación** para cada neurona a_k , que equivale a la salida de la misma.
- **Conexiones entre neuronas.** Cada conexión está definida por un peso w_j .
- **Una función de propagación**, que determina el efecto combinado de todas las neuronas conectadas a la neurona k , denotada por s_k . En el caso del perceptrón es la suma ponderada por los pesos.
- **Una función de activación** F_k , que dada la salida de la función de propagación y el estado de activación anterior a_k , calcule el nuevo estado de activación.
- Una entrada externa para cada unidad, denominada sesgo, y denotada por b_k .

A.1.2 Redes con más capas

Como ya hemos mencionado, la red anterior es la versión más simple dentro de las redes neuronales. La realidad es que con esta versión no estamos aprovechando el potencial que tiene este tipo de algoritmos, y en la práctica, se añaden capas ocultas entre la capa de entrada y de salida; podemos verlo de una manera visual y poco rigurosa en la Figura 57.

Las componentes en las redes con más capas son exactamente las mismas, con la única diferencia de que en este caso, en vez de tener un vector de pesos $\bar{W} = [w_1, \dots, w_d]$, tendremos una matriz de pesos.

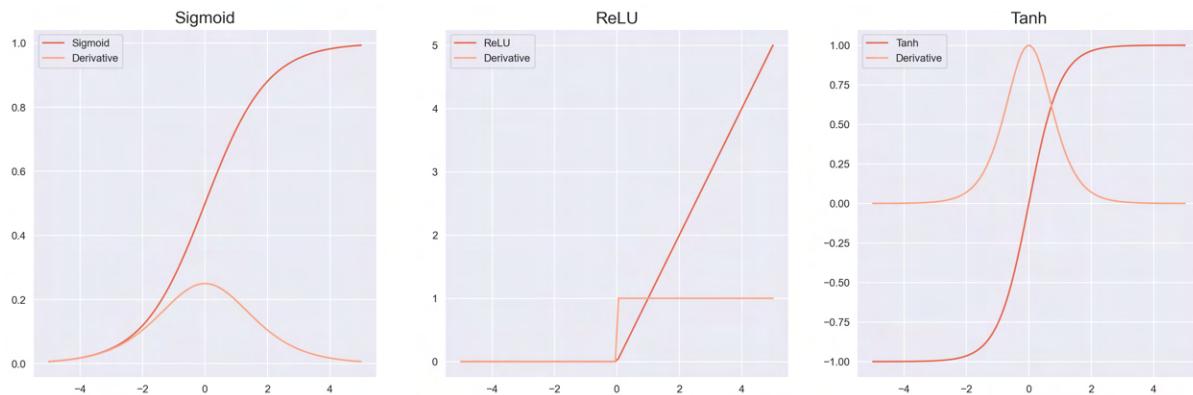


Figura 58: Funciones de activación más comunes

A.2 DISEÑO DE UNA RED NEURONAL

Ya hemos comentado alguna de las características y componentes principales de una red neuronal, a medida que introducíamos su funcionamiento. Sin embargo, resulta interesante conocer cuáles son exactamente, y qué alternativas existen para cada una de ellas, en función del problema al que nos estemos enfrentando.

A.2.1 Función de activación

Comenzaremos por la función de activación F_k , que para cada neurona n_k , toma como entrada el resultado de la Función de Propagación s_k , y el valor de la activación anterior a_k , y produce el nuevo valor de activación, aunque en la práctica únicamente depende del primero. Por lo tanto, tendrá la siguiente forma:

$$a_k = F_k(s_k) \quad (\text{A.2.1})$$

Normalmente se suelen utilizar como funciones de activación funciones crecientes que impongan un límite superior sobre el valor de la activación de la unidad, las más utilizadas se encuentran en la Figura 58. A continuación analizamos cada una de ellas, junto con su derivada, pues ambas son fundamentales para el algoritmo de entrenamiento de *backpropagation*.

La primera función que vamos a estudiar es la **función sigmoide**, cuya fórmula matemática es:

$$y(x) = \frac{1}{1 + e^{-x}}$$

$$y'(x) = y(x)(1 - y(x)) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

Esta función ha sido la más utilizada históricamente debido a su naturaleza diferenciable, que permitía que la red neuronal fuera entrenada mediante el algoritmo de *backpropagation*, que se basa precisamente en esta diferenciabilidad, tal y como veremos en la sección siguiente.

Sin embargo, su uso se ha reducido en la actualidad debido a ciertos problemas que presenta, como la saturación de gradientes, que se da cuando los valores de entrada de una función de activación producen gradientes muy pequeños o cercanos a cero. Esto puede ser un problema en el entrenamiento porque hace que los pesos de las capas anteriores no se actualicen adecuadamente, lo que lleva a una convergencia lenta o incluso a un estancamiento en el aprendizaje.

La segunda función de activación es la denominada **ReLU** o **rectificador**, que matemáticamente es:

$$y(x) = \max\{0, x\} \quad (\text{A.2.2})$$

Existe otra versión suavizada, $y(x) = \ln(1 + e^x)$, pero el objetivo es el mismo: solucionar el problema de la saturación de gradientes en las redes neuronales. A diferencia de la función anterior, la función ReLU es una función de activación lineal que sólo se activa cuando su entrada es positiva, es decir, devuelve la entrada si es mayor que cero y cero en caso contrario. Esto hace que el cálculo del gradiente sea más eficiente y acelera el proceso de entrenamiento de la red neuronal.

Por último veremos la función **tangente hiperbólica**, que como su propio nombre indica, es:

$$\begin{aligned} y(x) &= \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ y'(x) &= 1 - \tanh^2(x) \end{aligned}$$

La función \tanh tiene una forma similar a la sigmoide, pero su rango de valores se encuentra entre -1 y 1 en lugar de entre 0 y 1 . Esta función es útil cuando se desea una salida que oscile entre valores negativos y positivos, y es especialmente útil en redes neuronales que trabajan con datos que se centran en cero. Sin embargo, la función \tanh también puede sufrir de saturación de gradientes en los extremos, por lo que habrá que prestar atención a la hora de utilizarla.

Por último, aunque no está presente en la Figura 58, es necesario mencionar la función softmax, utilizada normalmente en la capa de salida para asignar m valores reales a m probabilidades de eventos discretos. Por ejemplo, en problemas de clasificación con m clases, para una observación concreta, en vez de obtener una salida que nos indique la clase, obtendremos la probabilidad de que pertenezca a dicha clase, mediante el uso de m neuronas en la capa de salida y esta función de activación.

La función softmax asigna valores reales a probabilidades que suman 1, específicamente, la función de activación para la i -ésima salida se define de la siguiente manera:

$$\Phi(v)_i = \frac{\exp(v_i)}{\sum_{j=1}^m \exp(v_j)} \quad \forall i \in \{1, \dots, m\} \quad (\text{A.2.3})$$

A.2.2 Función de pérdida

En secciones anteriores se ha mencionado brevemente el uso de una función de pérdida cuyo objetivo es medir la diferencia entre las predicciones y los valores observados, en forma de coste, para así poder actualizar correctamente los pesos de la red.

Además, las funciones de pérdida se pueden utilizar, una vez se ha entrenado la red, para obtener una medida de la calidad de las predicciones sobre el conjunto de evaluación, que nos permita comparar distintas arquitecturas o modelos entre sí.

La elección de la función de pérdida es crítica para definir las salidas de una manera que sea consistente con la aplicación en cuestión, es decir, que la función de pérdida escogida es dependiente de la estructura de las salidas de la red. Por lo tanto, resulta lógico pensar que tendremos que ajustar la función de pérdida al tipo de problema: clasificación o regresión.

Comenzamos por las funciones de pérdida más comunes en problemas de regresión, que son el error cuadrático medio, y el error absoluto medio.

El Error Cuadrático Medio (*Mean Squared Error*, en inglés) en un conjunto con N observaciones, viene dado por:

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (\text{A.2.4})$$

Esta función tiene múltiples propiedades que la hacen especialmente adecuada para actuar como función de pérdida. En primer lugar, la diferencia está elevada al cuadrado, y por tanto no importa si la predicción está por encima o por debajo del valor observado, únicamente se penalizan los valores con un gran error.

Además, es una función convexa, con un mínimo global claramente definido, lo que nos permite utilizar más fácilmente la optimización de los valores de los pesos por descenso de gradiente en el algoritmo de *backpropagation*.

El Error Absoluto Medio (*Mean Absolute Error*, en inglés), mide las diferencias absolutas entre las predicciones y los valores observados:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (\text{A.2.5})$$

Esta función se utiliza en ocasiones como alternativa al Error Cuadrático Medio, pues ésta última es muy sensible a valores atípicos, que suelen tener una distancia mayor a sus predicciones, y al elevarla al cuadrado la función de pérdida se ve claramente afectada. Por este motivo, en conjuntos con un número elevado de outliers, se suele usar el Error Absoluto Medio.

Sin embargo, también tiene sus limitaciones, si la distancia media entre las predicciones y los valores observados se aproxima a 0, el algoritmo del descenso del gradiente no funcionará, pues la función no es derivable en el 0.

Para solventarlo, se desarrolló una nueva función de pérdida, denominada Huber Loss [20], que presenta las ventajas del MSE y el MAE, solucionando sus mayores limitaciones:

$$\text{Huber Loss} = \begin{cases} \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 & \text{si } |y_i - \hat{y}_i| \leq \delta \\ \frac{1}{N} \sum_{i=1}^N \delta \left(|y_i - \hat{y}_i| - \frac{1}{2}\delta \right) & \text{si } |y_i - \hat{y}_i| > \delta \end{cases} \quad (\text{A.2.6})$$

Si la diferencia absoluta entre la predicción y el valor observado es menor o igual a un parámetro δ , entonces se aplicará MSE, en cualquier otro caso se aplicará MAE.

Vamos a proceder a estudiar las funciones de pérdida en problemas de clasificación, donde ya hemos visto que se obtiene una probabilidad de pertenencia a la clase i , p_i , y no la clase en sí.

La primera función de pérdida es la que se utiliza en problemas de clasificación binaria, es decir, problemas donde únicamente tenemos dos clases posibles, 0 o 1, y la denominamos *Binary Cross-Entropy* o *Log Loss*, en inglés. En este tipo de problemas, la salida será p_i , equivalente a la probabilidad de que la categoría sea 1, y entonces $1 - p_i$ es la probabilidad de que la categoría sea 0. Su fórmula matemática es la siguiente:

$$\text{CE Loss} = \frac{1}{n} \sum_{i=1}^N - (y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)) \quad (\text{A.2.7})$$

La segunda función de pérdida se utiliza en problemas de clasificación con m clases, y se llama *Categorical Cross-Entropy Loss*, y sigue una lógica similar a la anterior. De hecho, podríamos decir que la función *Binary Cross-Entropy* es un caso particular de *Categorical Cross-Entropy Loss* donde $m = 2$.

$$\text{CE Loss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \cdot \log(p_{ij}) \quad (\text{A.2.8})$$

donde p_{ij} es la probabilidad de que la observación i del conjunto de entrenamiento pertenezca a la clase j .

Hemos explorado las funciones de pérdida tanto en problemas de clasificación como de regresión. Ahora, nos adentraremos en cómo las redes neuronales abordan la modelización de la no linealidad para resolver una variedad de problemas complejos, más allá de los mecanismos usuales de Aprendizaje Supervisado.

A.2.3 *No linearidad*

A un nivel básico, una red neuronal es un grafo que se compone de funciones simples para obtener una función más compleja que modele la relación entre entradas y salidas. Sin embargo, no todas las funciones base son igualmente efectivas para lograr este objetivo, de hecho, las funciones utilizadas en las redes neuronales no se eligen arbitrariamente, sino que se diseñan cuidadosamente debido a ciertos tipos de propiedades.

Una propiedad bastante ilustrativa de las redes es que cualquier red en la que se utilice únicamente la identidad como función de activación es equivalente a una red con una sola capa, como la que hemos visto en secciones anteriores.

Por lo tanto, es crítico decidir correctamente qué funciones de activación se van a usar en cada una de las capas. Algunas observaciones que se deberían tener en cuenta son:

1. La composición de funciones lineales siempre es una función lineal.
2. La composición de funciones no lineales simples puede resultar en una función no lineal muy compleja.

Con estas observaciones podríamos deducir que las redes neuronales alcanzan su potencial cuando incluimos alguna función de activación no lineal en las capas intermedias.

El resultado de la aproximación universal de las redes neuronales postula que una combinación de funciones de *aplastamiento* que acotan el espacio, como la sigmoide, en una sola capa oculta seguida de una capa con función de activación lineal puede aproximar cualquier función medible con cualquier grado de precisión [19].

Es decir, una red de dos capas es suficiente siempre que el número de unidades ocultas sea lo suficientemente grande y se haya incluido algún tipo de no linealidad básica en la función de activación para poder modelar la arbitrariedad de cualquier función.

A.3 ENTRENAMIENTO DE REDES. BACKPROPAGATION

Ya hemos estudiado brevemente en qué consiste el entrenamiento en los problemas de Aprendizaje Supervisado: usamos una parte de los datos como conjunto de entrenamiento y el resto como conjunto de evaluación. En las Redes Neuronales esta lógica se mantiene, pero subdividiremos el conjunto inicial en tres conjuntos diferentes: entrenamiento, validación y prueba.

El **conjunto de entrenamiento** es el que se utiliza para ajustar los pesos de la red neuronal, el **conjunto de validación** se utiliza para ajustar los hiperparámetros y la arquitectura de la red, y el **conjunto de prueba** se utiliza para evaluar el rendimiento final del modelo.

En la práctica, no ajustamos los pesos de la red tras cada observación procesada, como veíamos anteriormente, sino que dividiremos el conjunto de entrenamiento en lotes o *batches*, y los pesos de la red se ajustarán cada vez que se termine de procesar un lote. Además, no se recorren los datos de entrenamiento una única vez, sino que se realiza un número fijo de iteraciones o *epochs* que recorren todo el conjunto y en las que se ajustan los pesos de la red neuronal de acuerdo con el algoritmo del descenso del gradiente.

El número de *epochs* necesarios para entrenar una red neuronal depende de la complejidad del problema y de la cantidad de datos disponibles, siendo un parámetro más a decidir como parte del diseño de la red.

Como veíamos en secciones anteriores, el primer paso para entrenar la red es obtener una salida de la red, procesando las primeras observaciones del conjunto de entrenamiento, y a este proceso lo denominaremos *forward propagation*.

Forward propagation

Supongamos que el primer lote de entrenamiento está formado por $(x_1, y_1), \dots, (x_m, y_m)$ observaciones donde $x_i \in \mathbb{R}^d$, es decir, que cada variable de entrada es multidimensional, y tendrá tantas dimensiones como características. Vamos a modificar ligeramente la notación, para poder estudiar en detalle qué ocurre en las capas y las neuronas.

Formalmente, podríamos escribirlo como:

$$\mathbf{X} = \begin{bmatrix} & & & \\ | & | & \cdots & | \\ \mathbf{x}(1) & \mathbf{x}(2) & \dots & \mathbf{x}(m) \\ | & | & \cdots & | \end{bmatrix} \quad (\text{A.3.1})$$

donde $\mathbf{X} \in \mathbb{R}^{d,m}$.

También podríamos poner de esta forma el conjunto de salidas \mathbf{Y}

$$\mathbf{Y} = \begin{bmatrix} y(1) & y(2) & \dots & y(m) \end{bmatrix} \quad (\text{A.3.2})$$

La red neuronal está compuesta por $p \in \mathbb{N}$ capas donde cada capa contiene k_p neuronas. Vamos a centrarnos en una capa determinada, la capa $i \in \mathbb{N}$, que contiene k_i neuronas.

Podemos escribir el conjunto de valores de activación de las neuronas de la capa i como una matriz $A^i \in \mathbb{R}^{k_i, m}$. Cada columna representará el valor de la unidad j con una observación del conjunto de entrenamiento, y formalmente, se escribe:

$$\mathbf{A}^i = \left[\begin{array}{c|c|c|c|c} & & & \cdots & \\ \mathbf{a}^i(1) & \mathbf{a}^i(2) & \dots & \mathbf{a}^i(m) & \\ \hline & & \cdots & & \end{array} \right] \quad (\text{A.3.3})$$

Vamos a centrarnos en el primer elemento de la primera columna de A_i , es decir, el elemento $a_1^i(1)$. Este elemento representa la activación de la **primera unidad de la capa i** en la primera observación del conjunto de entrenamiento, esto es, utilizando (x_1, y_1) . Sabemos que la activación es el resultado de aplicar una función \mathcal{F} sobre el valor de la función de propagación $s_1^i(1)$.

Formalmente, tendríamos:

$$a_1^i(1) = \mathcal{F}_k(s_1^i(1)) \quad (\text{A.3.4})$$

donde el resultado de la función de propagación $s_1^i(1)$ viene dado por la multiplicación de los pesos que unen las neuronas de la capa $i - 1$ con la capa i

$$s_1^i(1) = \sum_{j \in \text{unidades en } i-1} w_{1j}^i(1)a_j^{i-1}(1) + b_1^i(1) \quad (\text{A.3.5})$$

Se puede observar que los subíndices siempre corresponden a la unidad y los superíndices representan la capa, así, $s_1^i(1)$ es la función de propagación de la primer unidad de la capa i , $w_{1j}^i(1)$ es el peso de la unidad j a la unidad 1 de la capa i , y $b_1^i(1)$ es el sesgo de la primera unidad de la capa i .

Esta forma de calcular $s_1^i(1)$ puede dar lugar a reescribir los pesos de la capa i de forma matricial, y entonces tendríamos $W^i \in \mathbb{R}^{k_i k_{i-1}}$, formalmente:

$$W^i(1) = \left[\begin{array}{ccc} w_{11}^i(1) & \cdots & w_{1k_{i-1}}^i(1) \\ w_{21}^i(1) & \cdots & w_{2k_{i-1}}^i(1) \\ \vdots & \ddots & \vdots \\ w_{k_i 1}^i(1) & \cdots & w_{k_i k_{i-1}}^i(1) \end{array} \right] \quad (\text{A.3.6})$$

Y entonces el vector $s^i(1) \in \mathbb{R}_i^k$ viene dado por:

$$\begin{aligned} s^i(1) &= W^i(1)a^{i-1}(1) + b^i(1) \\ &= W^i(1)\mathcal{F}(s^{i-1}(1)) + b^i(1) \end{aligned}$$

Si volvemos a la Figura 57, podemos ver que $a_1^i(1)$ es el resultado de aplicar \mathcal{F} sobre cada elemento del vector $s^i(1)$. Si repetimos esto de manera iterativa a través de las p capas, llegamos a producir la salida de la red neuronal para la primera observación, dada por $\hat{y}(1) = a^p(1) = \mathcal{F}(s^p(1))$.

Backpropagation

Una vez se ha obtenido la salida $\hat{y}(1), \dots, \hat{y}(m)$ realizando *forward propagation* para cada observación dentro del lote, es el momento de actualizar el valor de los pesos, utilizando el algoritmo del descenso del gradiente.

Para ello, utilizamos una función de pérdida, que se encarga de comparar la salida de la red con el valor real, notada por $\mathcal{L}(\hat{y}(i), y(i))$. Se pueden tomar diversas funciones como función de pérdida, por ejemplo, podemos tomar el error cuadrático medio:

$$\mathcal{L}(\hat{y}(i), y(i)) = \frac{1}{|y|} \sum_{i=1}^{|y|} (y(i) - \hat{y}(i))^2 \quad (\text{A.3.7})$$

En el entrenamiento de la red, el objetivo principal es minimizar una función de coste, que denotaremos por J , y que depende de los parámetros de la red ($W^1, \dots, W^p, b^1, \dots, b^p$). En el caso de que se actualicen los parámetros después de cada observación, y por tanto, no se utilicen lotes, la función coste será idéntica a la función de pérdida. En otro caso, tendremos:

$$J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}(i), y(i)) \quad (\text{A.3.8})$$

Se podría considerar una dependencia de J de los parámetros ($W^1, \dots, W^p, b^1, \dots, b^p$), pues acabamos de comprobar que se puede expresar cualquier salida $\hat{y}(i)$ como una función sobre los pesos y los sesgos de cada capa.

Para minimizar J utilizamos el descenso del gradiente, y por tanto será necesario calcular el gradiente de la función con respecto a los parámetros que queremos optimizar, así, será necesario calcular para cada capa i , las matrices:

$$\frac{\partial J}{\partial \mathbf{W}^i} = \begin{bmatrix} \frac{\partial J}{\partial w_{11}^i} & \cdots & \frac{\partial J}{\partial w_{1k_i-1}^i} \\ \frac{\partial J}{\partial w_{21}^i} & \cdots & \frac{\partial J}{\partial w_{2k_i-1}^i} \\ \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial w_{k_i 1}^i} & \cdots & \frac{\partial J}{\partial w_{k_i k_i-1}^i} \end{bmatrix}$$

$$\frac{\partial J}{\partial \mathbf{b}^i} = \begin{bmatrix} \frac{\partial J}{\partial b_1^i} \\ \frac{\partial J}{\partial b_2^i} \\ \vdots \\ \frac{\partial J}{\partial b_k^i} \end{bmatrix}$$

Por lo tanto, será necesario calcular:

$$\left(\frac{\partial J}{\partial \mathbf{W}^1}, \frac{\partial J}{\partial \mathbf{b}^1}, \dots, \frac{\partial J}{\partial \mathbf{W}^p}, \frac{\partial J}{\partial \mathbf{b}^p} \right) \quad (\text{A.3.9})$$

Para ello, se procede de atrás hacia delante, utilizando la regla de la cadena. Vamos a considerar la última capa p , con la función de activación sigmoide, recordamos que se verificaba lo siguiente:

$$\begin{aligned} z^p &= \mathbf{W}^p a^{p-1} + b^p \\ a^p &= \sigma(z^p) \\ J(a^p, y) &= -y \log(a^p) - (1-y) \log(1-a^p) \end{aligned} \quad (\text{A.3.10})$$

Y entonces, en esta capa será necesario calcular:

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{W}^p} &= \frac{dJ}{da^p} \frac{da^p}{dz^p} \frac{\partial z^p}{\partial \mathbf{W}^p} \\ \frac{\partial J}{\partial b^p} &= \frac{dJ}{da^p} \frac{da^p}{dz^p} \frac{\partial z^p}{\partial b^p} \end{aligned} \quad (\text{A.3.11})$$

Una vez se hayan calculado las expresiones anteriores, se puede proceder a calcular los gradientes correspondientes a la capa anterior, esto es, $p-1$:

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{W}^2} &= \frac{dJ}{dz^3} \frac{dz^3}{d\mathbf{a}^2} \frac{d\mathbf{a}^2}{d\mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{W}^2} \\ \frac{\partial J}{\partial \mathbf{b}^2} &= \frac{d\mathcal{L}}{dz^3} \frac{dz^3}{d\mathbf{a}^2} \frac{d\mathbf{a}^2}{d\mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{b}^2} \end{aligned} \quad (\text{A.3.12})$$

Dejando aparte la aparente complejidad, es importante observar que para calcular estos gradientes, se utilizan los valores obtenidos en el cálculo de los gradientes de la capa p . Por este motivo se procede de atrás hacia delante.

Una vez se han obtenido todos los gradientes, se puede aplicar el algoritmo del descenso del gradiente para actualizar los parámetros, resultando en $\forall i = 1, \dots, p$:

$$\begin{aligned} W^i &= W^i + \gamma \frac{\partial J}{\partial \mathbf{W}^i} \\ b^i &= b^i + \gamma \frac{\partial J}{\partial \mathbf{b}^i} \end{aligned}$$

La actualización de los pesos se repite cada vez que se termine de procesar un lote, e iterando a través de todo el conjunto de entrenamiento tantas veces como *epochs* se hayan fijado. Este proceso permite que la red pueda aprender a partir de los datos de entrada y producir una salida que se ajuste lo mejor posible a las etiquetas de entrenamiento correspondientes.

Es importante notar que aunque se ha estudiado en detalle el proceso de ajuste de pesos según el descenso del gradiente (también denominado SGD), existen otros algoritmos que mejoran la convergencia hacia un mínimo. Estos son el Adaptive Moment Estimation (*Adam*) y el Root Mean Square Propagation (*RMSProp*).

Adam es un optimizador adaptativo que ajusta el *learning rate* en función del historial de gradientes, lo que lo hace más eficiente que el SGD. También tiene un parámetro de momento que ayuda a suavizar el proceso de actualización de los pesos.

RMSProp es similar a *Adam* en el sentido de que es un optimizador adaptativo, pero utiliza una técnica diferente para ajustar el learning rate. *RMSProp* calcula un promedio móvil exponencial de los gradientes cuadrados y utiliza este valor para normalizar el gradiente.

En general cada optimizador tiene sus ventajas y desventajas, y la elección depende del problema y los datos específicos que se estén utilizando.

En este capítulo se ha presentado la arquitectura básica de una red neuronal. Además, se ha explicado cómo se utiliza el algoritmo de backpropagation para entrenar la red neuronal y ajustar los pesos de las conexiones. A continuación profundizaremos en los fundamentos teóricos de los grafos, y las arquitecturas de red disponibles para ellos, las Redes Neuronales de Grafos (GNNs, por sus siglas en inglés).

B

REGRESIÓN. RESULTADOS POR SIMULACIONES.

En este capítulo vamos a estudiar uno a uno los resultados de cada uno de los algoritmos introducidos anteriormente. Para cada uno de ellos, compararemos su rendimiento con el de una arquitectura recurrente usual formada por un bloque LSTM.

Para analizar los resultados al detalle, vamos a observar las predicciones realizadas en dos situaciones distintas del conjunto de test, para ocho nodos. La elección de los ocho nodos se ha llevado a cabo teniendo en cuenta el vecindario de los nodos involucrados en la incidencia: por ejemplo, si se dió un cortocircuito en el generador que estaba conectado al nodo 101, se han incluido en la visualización los nodos 101, 102, 151, y 201, que forman el vecindario del mismo, junto con otros cuatro nodos elegidos al azar.

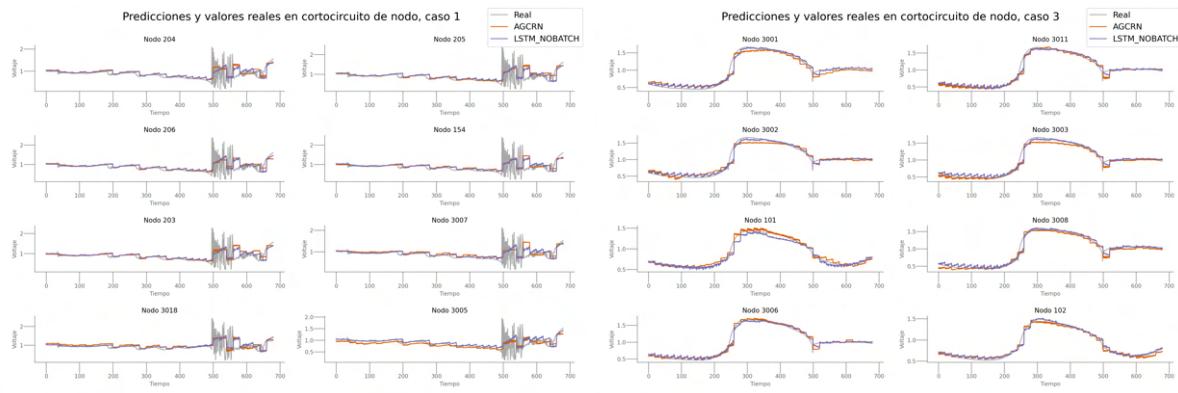
Sería positivo tener en cuenta que para cada tipo de incidencia hay una variación considerable entre simulaciones, tal y como veíamos en el Capítulo 6. Por lo tanto, las conclusiones obtenidas en este capítulo con las situaciones concretas que se van a estudiar no tienen por qué ser generalizables al resto, sino que suponen más bien una exemplificación del funcionamiento de cada modelo.

B.1 CORTOCIRCUITO DE GENERADOR

AGCRN

Para entrenar la red AGCRN se han probado combinaciones para tres parámetros distintos. El primero, `hidden_size`, marca la dimensión de salida del bloque AGCRN, equivalente a la entrada de la capa lineal completamente conectada, que dará la salida final. El segundo, `embedding`, denotará la dimensión de las representaciones de los nodos. Por último, `k`, representa el tamaño del filtro.

Al analizar los resultados de la arquitectura AGCRN (Figura 59) llegamos a la conclusión de que, al menos para este tipo de problemas, donde no hay grandes variaciones entre los estados de los nodos, se obtiene mejor rendimiento utilizando una arquitectura LSTM sencilla.



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

Figura 59: AGCRN

DyGrEncoder

A continuación estudiamos la Figura 60, donde se ilustran los resultados de la red DyGrEncoder con mejores resultados en test.

De nuevo, se ha realizado un ajuste de parámetros, probando distintas combinaciones de:

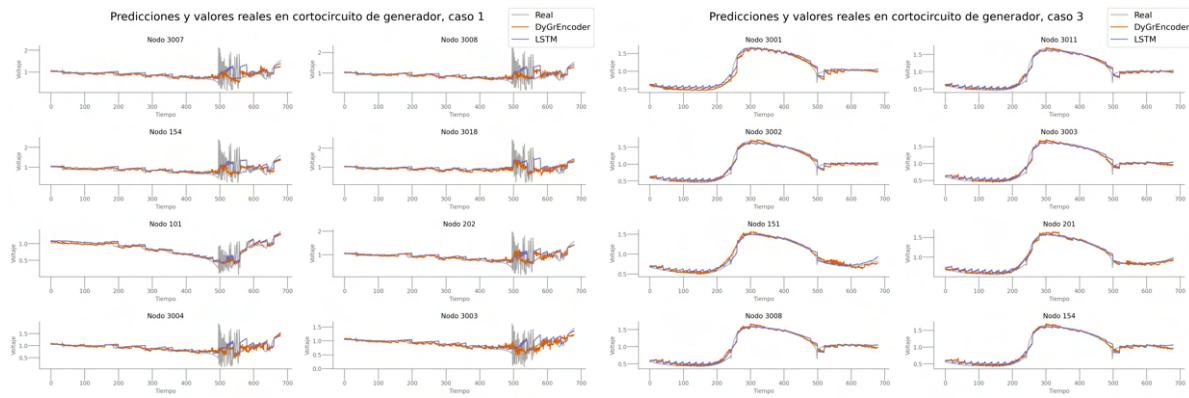
- `conv_num_layers`: indica el número de capas de convoluciones grafo-gateadas que se aplicarán.
- `conv_aggr`: especifica el esquema de agregación a utilizar, pudiendo ser la suma, la media o el máximo, lo cual define cómo se combinan las características de los nodos vecinos en el grafo.
- `lstm_num_layers`: representa el número de neuronas en cada capa del LSTM.

Además, debido a la capacidad limitada de procesamiento, se tuvo que establecer un valor por defecto para los parámetros `conv_out_channels`, que marca la dimensionabilidad del espacio de salida después de la convolución, y `lstm_out_channels`, que de manera similar, determina el número de canales de salida del LSTM. El valor por defecto escogido es equivalente al número de características de entrada.

La Figura 60 muestra que esta arquitectura alcanza un rendimiento comparable al de LSTM en ambos casos estudiados. Considerando la limitación en el ajuste completo de los parámetros, es razonable esperar una mejora en el rendimiento con mayor capacidad computacional.

MPNN-LSTM

Continuamos con la arquitectura MPNN-LSTM, una de las más sencillas en cuanto al ajuste de parámetros. En este caso, únicamente hemos ajustado dos de ellos: `hidden`, que representa la dimensión de la salida del bloque MPNN-LSTM, y `dropout`, que



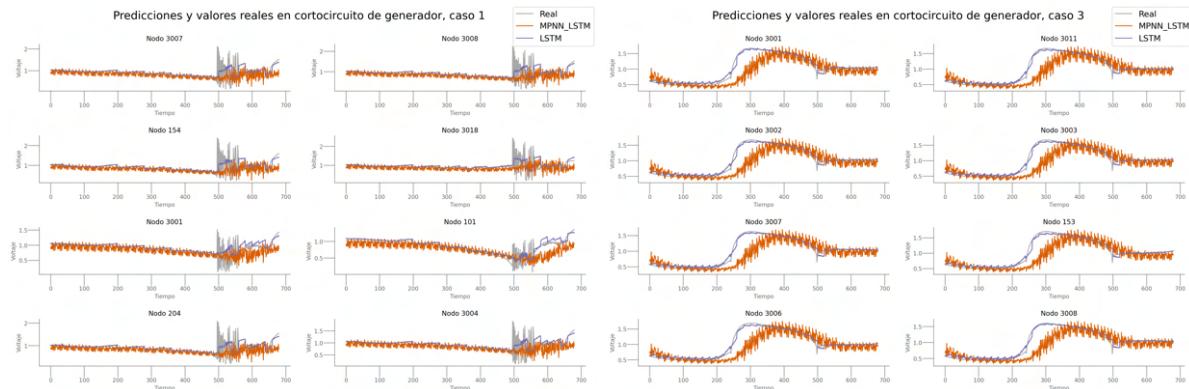
(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

Figura 60: DyGrEncoder

es la proporción de neuronas que se desactivan durante el entrenamiento, de manera aleatoria, para evitar el sobreajuste.

En la Figura 61 podemos observar cómo la arquitectura MPNN-LSTM no resulta especialmente adecuada para este tipo de datos. Presenta unas variaciones muy grandes que hacen que las predicciones tengan una forma sinusoidal y no se ajusten correctamente a los datos. Además, en el caso de la situación 3, en la que se da una subida relativamente brusca en el voltaje de los nodos, el modelo no es capaz de predecirla correctamente, lo cual es una representación muy clara de su pobre rendimiento.



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

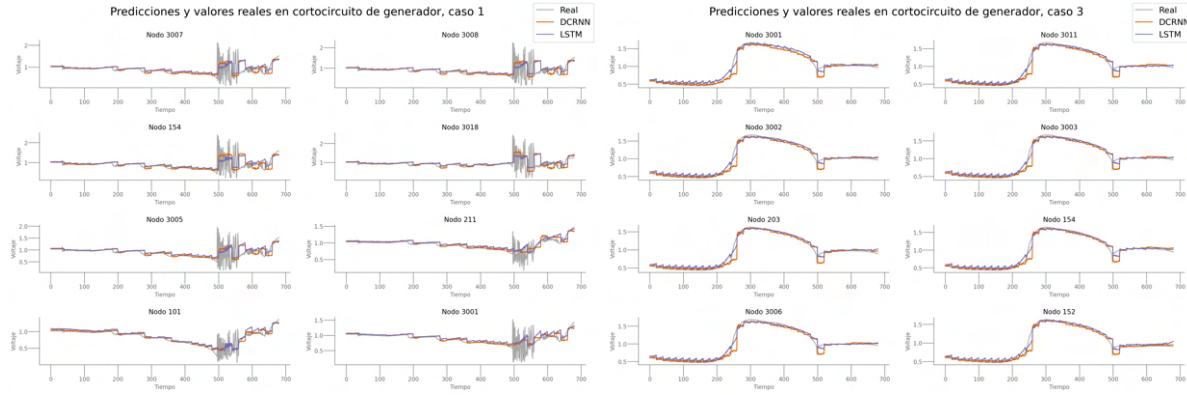
Figura 61: MPNN-LSTM

DCRNN

La siguiente arquitectura, DCRNN, es también una de las más sencillas, pues únicamente tiene como parámetro ajustable la dimensión de la salida del bloque DCRNN.

Al analizar la Figura 62, podemos comprobar cómo DCRNN tiene un rendimiento casi idéntico al de LSTM. Es destacable la bajada y subida repentina en los voltajes en

la situación de la derecha, pues ninguno de los dos modelos consiguen replicarla de manera correcta. Por una parte, LSTM no llega a modelar la bajada completa, y mantiene unas predicciones por encima de los valores reales; y por otra parte, DCRNN sí que predice correctamente la magnitud de la bajada, pero no es capaz de adaptarse a la subida.



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

Figura 62: DCRNN

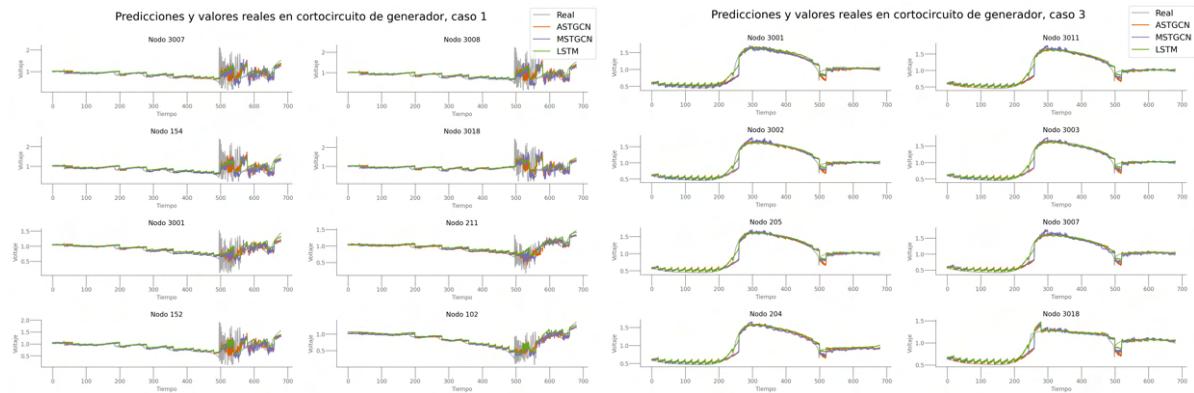
ASTGCN y MSTGCN

Debido a la similitud de las dos arquitecturas, vamos a estudiar los resultados de ambas de manera conjunta. En este caso, los parámetros que se ajustan son los siguientes:

- `nb_block`: Número de bloques ASTGCN o MSTGCN en el modelo.
- `in_channels`: Número de características de entrada.
- `K`: Orden de los polinomios de Chebyshev. El grado es `K-1`.
- `nb_chev_filters`: Número de filtros de Chebyshev.
- `nb_time_filters`: Número de filtros temporales.
- `time_strides`: Pasos temporales durante la convolución temporal.

En el caso de MSTGCN sí que se ha llevado a cabo una búsqueda por fuerza bruta con todas las combinaciones posibles, pero en el caso de ASTGCN, resultaba computacionalmente prohibitivo proceder de ese modo, por lo que se procedió mediante una búsqueda aleatoria, fijando 50 como número máximo de iteraciones.

En la Figura 63 podemos ver el rendimiento de ambas arquitecturas, de nuevo comparándolas con una LSTM básica. Resulta interesante comprobar cómo éstas arquitecturas producen predicciones con una variación más alta en la simulación de la izquierda. Sin embargo, no modelan correctamente la bajada drástica que se da en alguno de los nodos después del periodo de alta perturbación, pues las predicciones se mantienen en ese nivel de variabilidad alto, mientras que los valores reales bajan y se estabilizan. Resultaría especialmente interesante obtener modelos que sí que puedan predecir la estabilización de los voltajes en los nodos.



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

Figura 63: MTGNN y ASTGCN

MTGNN

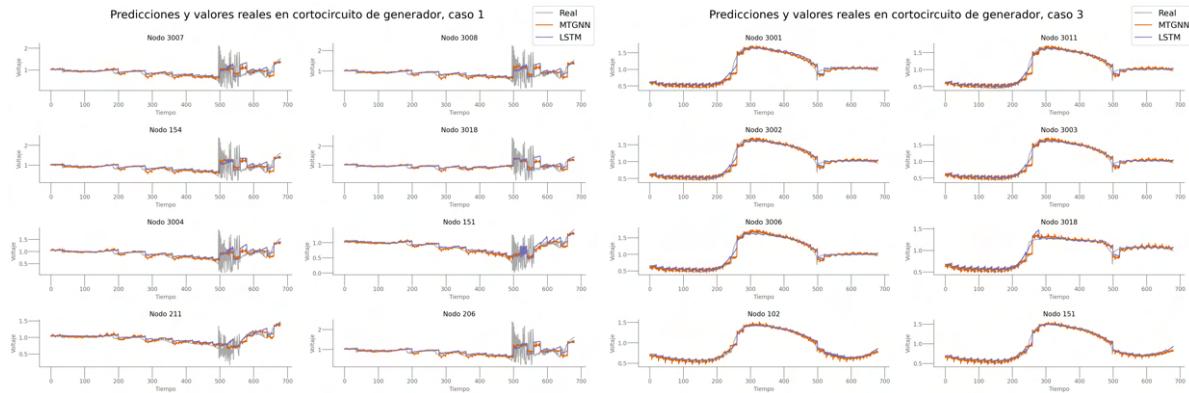
Podemos proceder entonces con la siguiente arquitectura, que ha sido una de las que, por las limitaciones del procesamiento local, se ha entrenado en una instancia de máquina virtual en Google Cloud, siguiendo el mismo procedimiento que en el entrenamiento de ASTGCN, mediante el cual se realiza una búsqueda aleatoria en el espacio de parámetros con un número máximo de iteraciones prefijado.

De nuevo, se ha realizado un ajuste de parámetros, probando distintas combinaciones de:

- `gcn_depth`: Profundidad de la convolución de grafo.
- `out_dim`: Dimensión de salida.
- `skip_channels`: Canales de salto.
- `residual_channels`: Canales residuales.
- `end_channels`: Canales finales.
- `conv_channels`: Canales de convolución.

El resto de parámetros del modelo, como por ejemplo `subgraph_size`, que determina el tamaño del subgrafo, o `propalpha`, que mide el ratio de retención de los estados originales de los nodos raíz en la propagación mix-hop, se establecieron por defecto.

Al analizar la Figura 64, vemos que MTGNN tiene un rendimiento muy similar a LSTM de manera general, pero que las predicciones obtenidas con MTGNN tienen más ruido y más variación. Podríamos pensar que esto sería algo positivo en el periodo de alta perturbación en la simulación de la izquierda, sin embargo, en la Figura 64 podemos observar que no es el caso, y que las predicciones no se ajustan de manera correcta a los datos.



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

Figura 64: MTGNN

STConv

Seguidamente, vamos a introducir cómo ha sido el entrenamiento del modelo con arquitectura STConv. En este caso, los parámetros que se tenían que ajustar son:

- `hidden_channels`: Número de unidades de salida del bloque convolucional.
- `out_channels`: Dimensión de salida.
- `kernel_size`: Tamaño del filtro considerado.
- `normalization`: Normalización para el grafo Laplaciano.

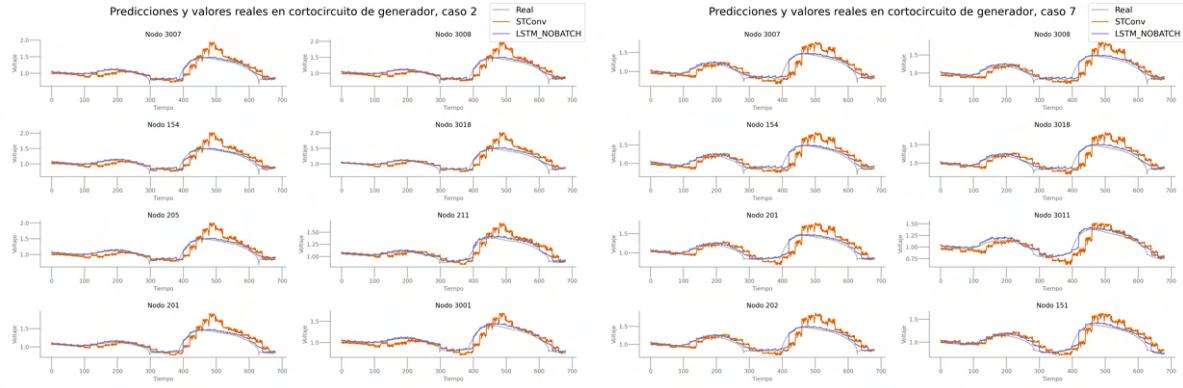
El problema con el entrenamiento de este modelo es que es computacionalmente costoso, y por este motivo, en primer lugar se intentó ejecutar una búsqueda aleatoria en Google Cloud. Sin embargo, el coste era demasiado elevado y los resultados obtenidos no fueron positivos, por lo que se decidió guardar el mejor modelo que se hubiera obtenido, sin terminar todas las iteraciones iniciales.

Esto quiere decir que por una parte, muy probablemente se pueda mejorar el rendimiento y la precisión del modelo, pero con un coste muy elevado a nivel computacional. Teniendo en cuenta que existen otros modelos con los que hemos obtenido buen rendimiento, no queda claro si sería una decisión correcta llevarlo a cabo, y por este motivo, en nuestro caso, decidimos mantener un modelo aparentemente peor.

Al observar la Figura 65, podemos ver cómo las predicciones del modelo no consiguen ajustarse del todo a los datos reales, sino que *reaccionan* a los datos de entrada.

EvolveGCN

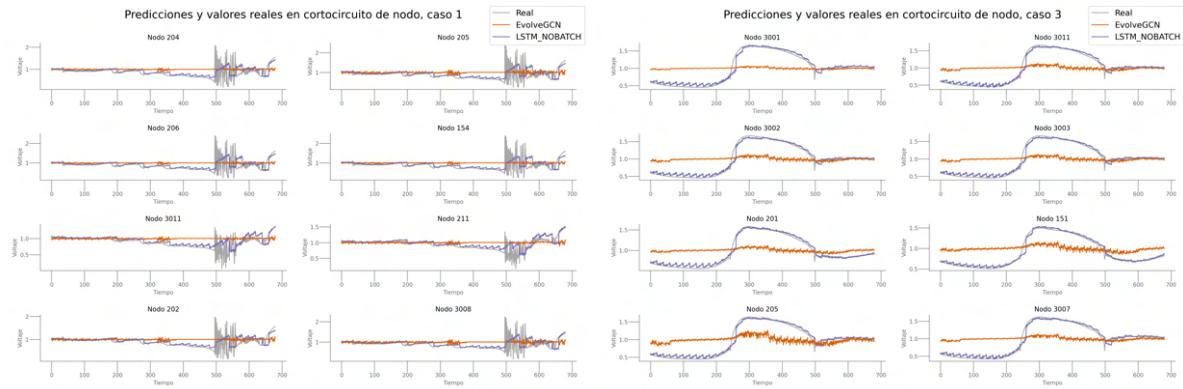
El siguiente algoritmo, EvolveGCN, no tiene ningún parámetro entrenable, es decir que no ha sido necesario realizar la búsqueda por fuerza bruta. Sin embargo, esta sencillez resulta negativa, pues no es posible mejorar el rendimiento del modelo, (Figura 66).



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

Figura 65: STConv



(a) Cortocircuito en el generador del bus 3018

(b) Cortocircuito en el generador del bus 3011

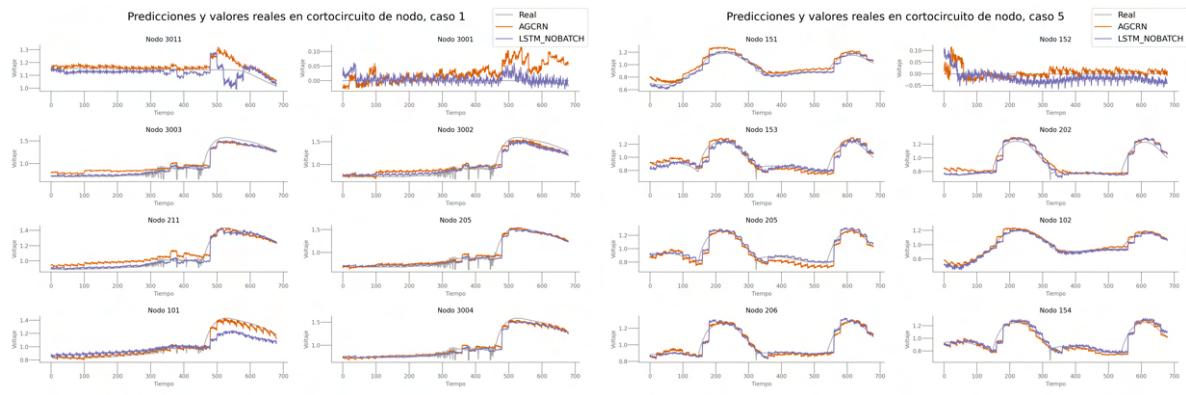
Figura 66: EvolveGCN

B.2 CORTOCIRCUITO DE NODO

Para revisar los resultados obtenidos vamos a seguir el mismo procedimiento, estudiando uno a uno los algoritmos comparando su rendimiento con el obtenido con un modelo LSTM básico.

En primer lugar, en la Figura 67, vemos que el algoritmo AGCRN no termina de ajustarse correctamente a los datos. Destaca especialmente en la imagen de la izquierda, en el nodo 3001, donde tras el cortocircuito el voltaje se mantiene en 0.0. Aunque ninguno de los dos modelos es capaz de captarlo, sí que se observa un cambio en la forma de las predicciones en el caso de LSTM, mientras que en AGCRN las predicciones se desvían.

En la Figura 68 podemos analizar la calidad de las predicciones de DyGrEncoder. Si recordamos los resultados obtenidos en la sección anterior, éste era el algoritmo cuyas predicciones en el conjunto de test se ajustaban más a los datos. Sin embargo, en este caso, pese a que se soluciona el problema mencionado anteriormente, en el que el algoritmo no captaba el cortocircuito completo en un nodo, el modelo DyGrEn-

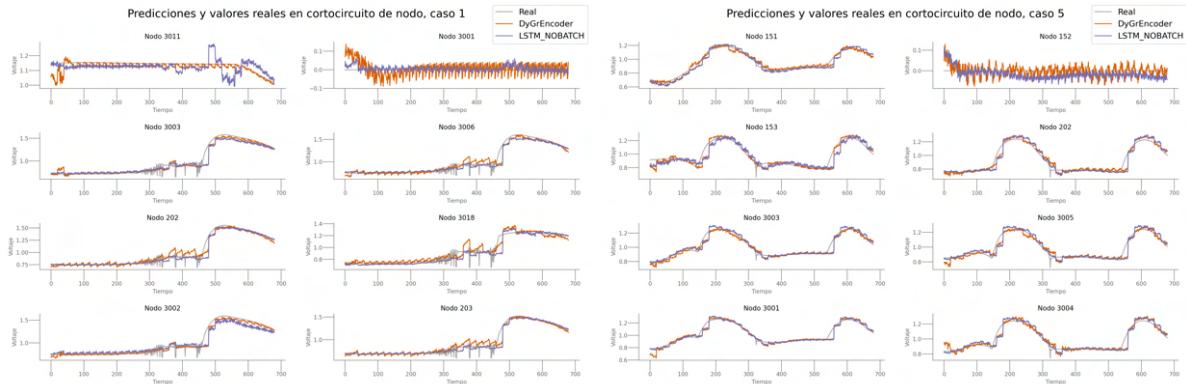


(a) Cortocircuito en el bus 3001

(b) Cortocircuito en el bus 152

Figura 67: AGCRN

coder añade una perturbación muy grande a los datos. Esto dificulta la identificación del nodo origen del problema, y por lo tanto complica la tarea de prevención de incidencias.



(a) Cortocircuito en el bus 3001

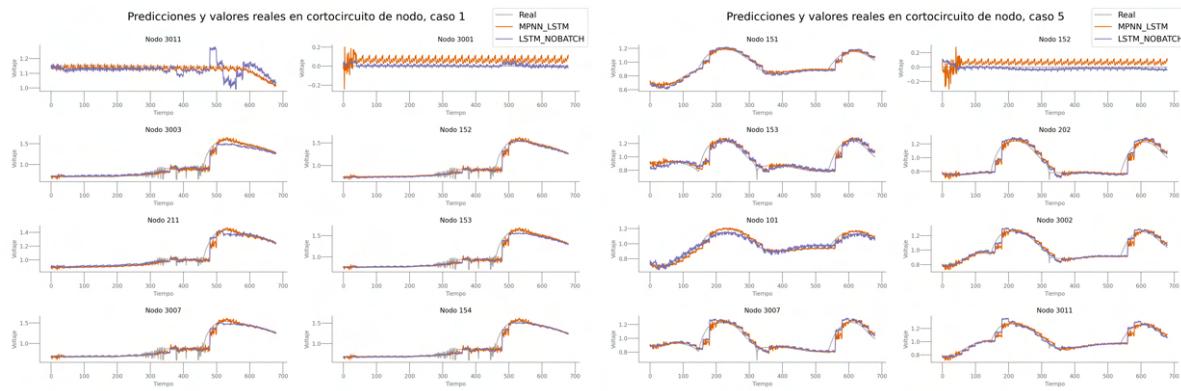
(b) Cortocircuito en el bus 152

Figura 68: DyGrEncoder

Continuamos con la arquitectura MPNN-LSTM, cuyos resultados pueden estudiarse en la Figura 69. En este caso es destacable la diferencia en el rendimiento frente al tipo de incidencia anterior, pues podemos determinar que las predicciones obtenidas se ajustan a los datos de manera muy superior a la forma en la que lo hacían en el caso de cortocircuito de generador.

En el caso de un modelo DCRNN, cuyos resultados se pueden estudiar en la Figura 70. En ella, podemos concluir que no hay gran diferencia entre el rendimiento del modelo LSTM y DCRNN, en ninguna de las dos situaciones analizadas.

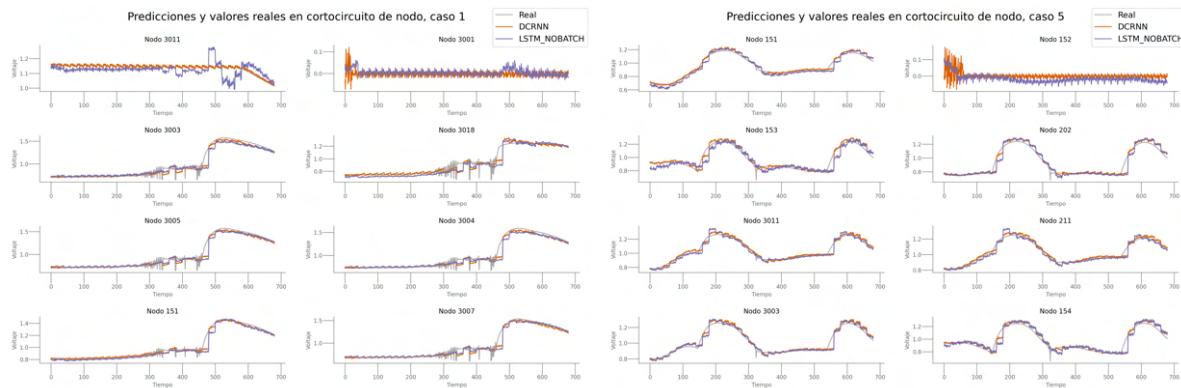
Las siguientes arquitecturas, ASTGCN y MSTGCN se estudiarán en conjunto. En la Figura 71 podemos comprobar cómo la arquitectura más compleja, ASTGCN, tiene un rendimiento inferior. Un motivo para justificarlo podría ser que el ajuste de los parámetros se realizó de una manera distinta debido a la falta de recursos computacionales, tal y como se vió anteriormente.



(a) Cortocircuito en el bus 3001

(b) Cortocircuito en el bus 152

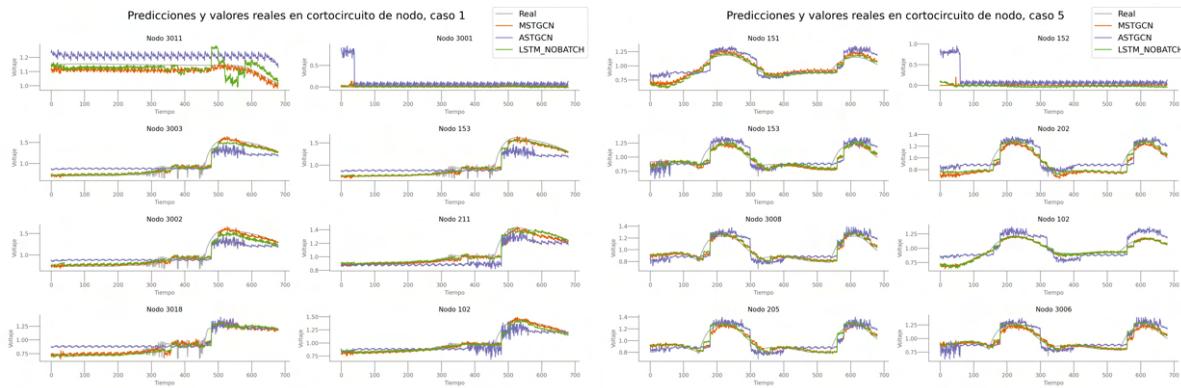
Figura 69: MPNN



(a) Cortocircuito en el bus 3001

(b) Cortocircuito en el bus 152

Figura 70: DCRNN



(a) Cortocircuito en el bus 3001

(b) Cortocircuito en el bus 152

Figura 71: MSTGCN

En la Figura 72, por otro lado, podemos analizar la forma que tienen las predicciones del modelo MTGNN. En este tipo de incidencias, las predicciones obtenidas por este

modelo tienen una perturbación mayor, y no se producen predicciones estables, lo que, de nuevo, complica la identificación de incidencias.

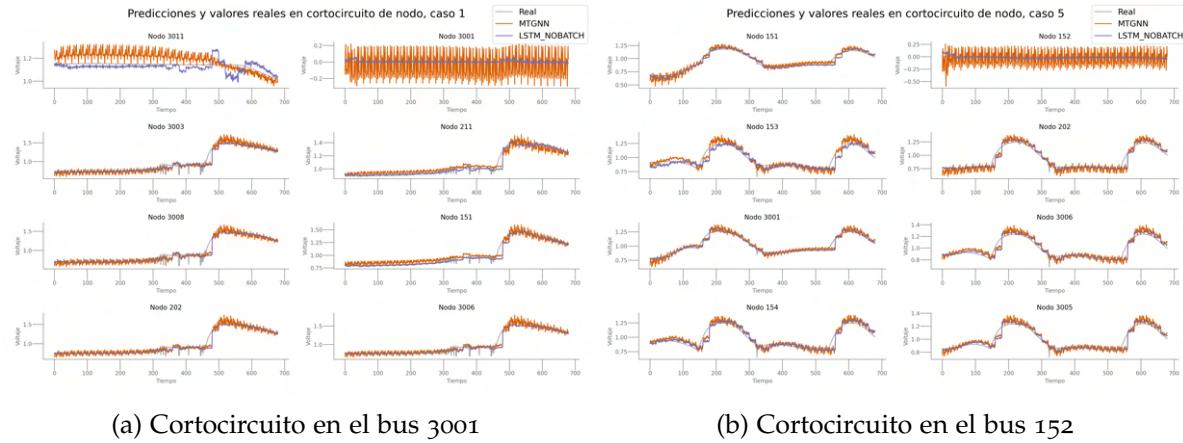


Figura 72: MTGNN

Podemos llegar a conclusiones similares en el caso de STConv (Figura 73), excepto que en este caso no solo estamos lidiando con alta perturbación, sino que en general el modelo no capta los patrones de los datos.

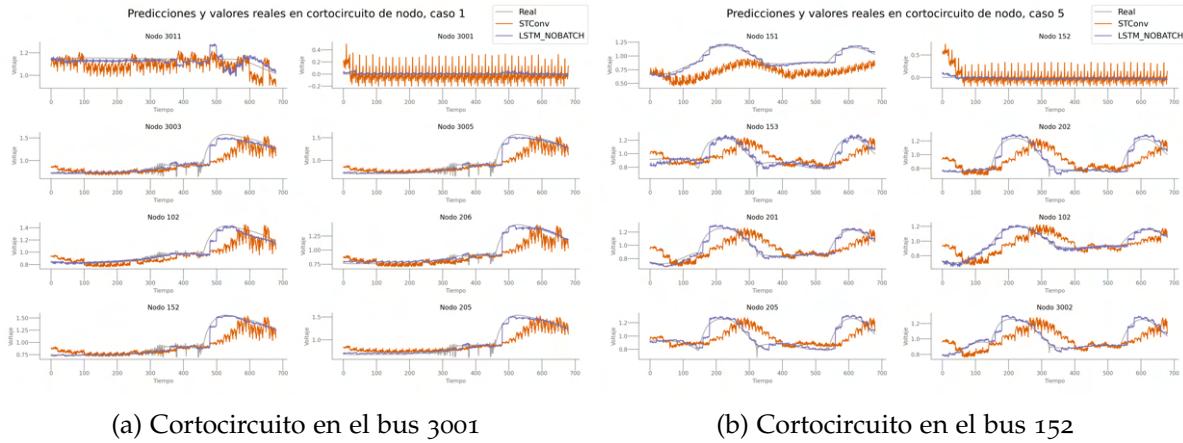
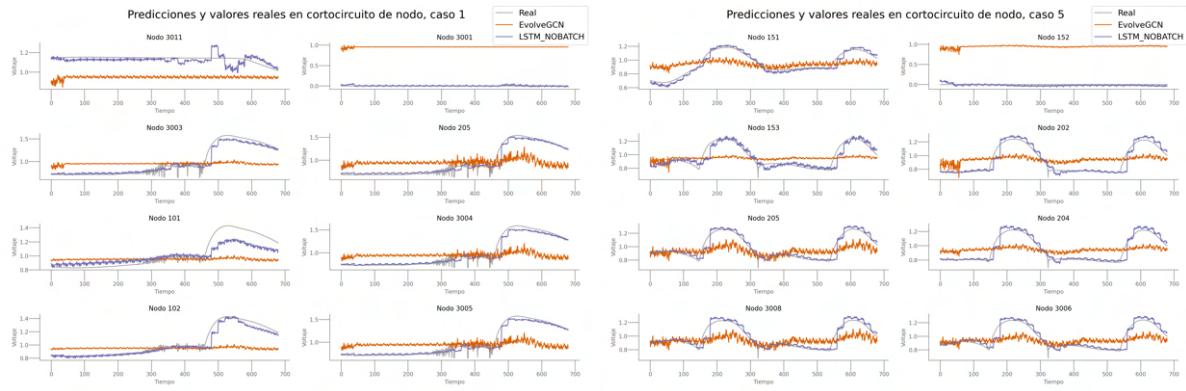


Figura 73: STConv

Por último, podemos estudiar en la Figura 74 los resultados de una arquitectura EvolveGCN. En este caso se repite la conclusión del tipo de incidencia anterior, y es que es el modelo que peor rendimiento produce.

B.3 ERROR DE NODO

Procederemos a analizar los resultados relacionados con el tipo de incidencia denominado "error de nodo". En este caso, no restringiremos el estudio a solo dos simulaciones. Esto se debe a que, dentro de este tipo de incidencia en particular, se observan



(a) Cortocircuito en el bus 3001

(b) Cortocircuito en el bus 152

Figura 74: EvolveGCN

patrones muy variados, lo que podría llevar a diferentes resultados según el algoritmo empleado. Por tanto, resulta interesante analizar estas diferencias.

En primer lugar, en la Figura 75, podemos estudiar los resultados de la arquitectura AGCRN. Se pueden realizar varias observaciones:

- La primera, es que en el caso de la primera Figura, en la que se da una subida repentina en todos los nodos, ninguno de los dos algoritmos es capaz de modelarla completamente.
- En el caso de la segunda simulación, ninguno de los dos algoritmos resultaría adecuado, pues las predicciones son muy lejanas a los valores reales.
- Por último, en la tercera simulación observamos patrones sinusoidales que ya habíamos visto en otro tipo de incidencias. En este caso, una arquitectura LSTM funcionaría mejor.

Continuamos con la arquitectura DyGrEncoder, cuyos resultados pueden observarse en la Figura 76. Podemos llegar a unas conclusiones muy similares a las que acabamos de realizar en el caso de AGCRN, excepto que en este caso, en la última simulación, el modelo DyGrEncoder realiza predicciones más acertadas, especialmente en el caso del nodo afectado.

Continuamos con la arquitectura MPNN-LSTM (Figura 77). Nuevamente, las conclusiones son similares, pero es importante destacar que, en la segunda simulación, donde los valores reales son muy estables y anteriormente no se había logrado un buen rendimiento, ahora obtenemos predicciones relativamente cercanas a estos valores. Aunque estas predicciones no son ideales debido a la gran perturbación asociada, lo que podría dificultar la implementación de algoritmos para detectar subidas o bajas repentinamente, representan una mejora en comparación con los algoritmos previos.

Al estudiar los resultados de la arquitectura DCRNN (Figura 78), podemos concluir que en cada una de las tres simulaciones estudiadas, la arquitectura LSTM resulta más acertada. En la primera simulación ambos modelos producen resultados similares, sin embargo, en la segunda y en la tercera, vemos cómo para el nodo afectado por la

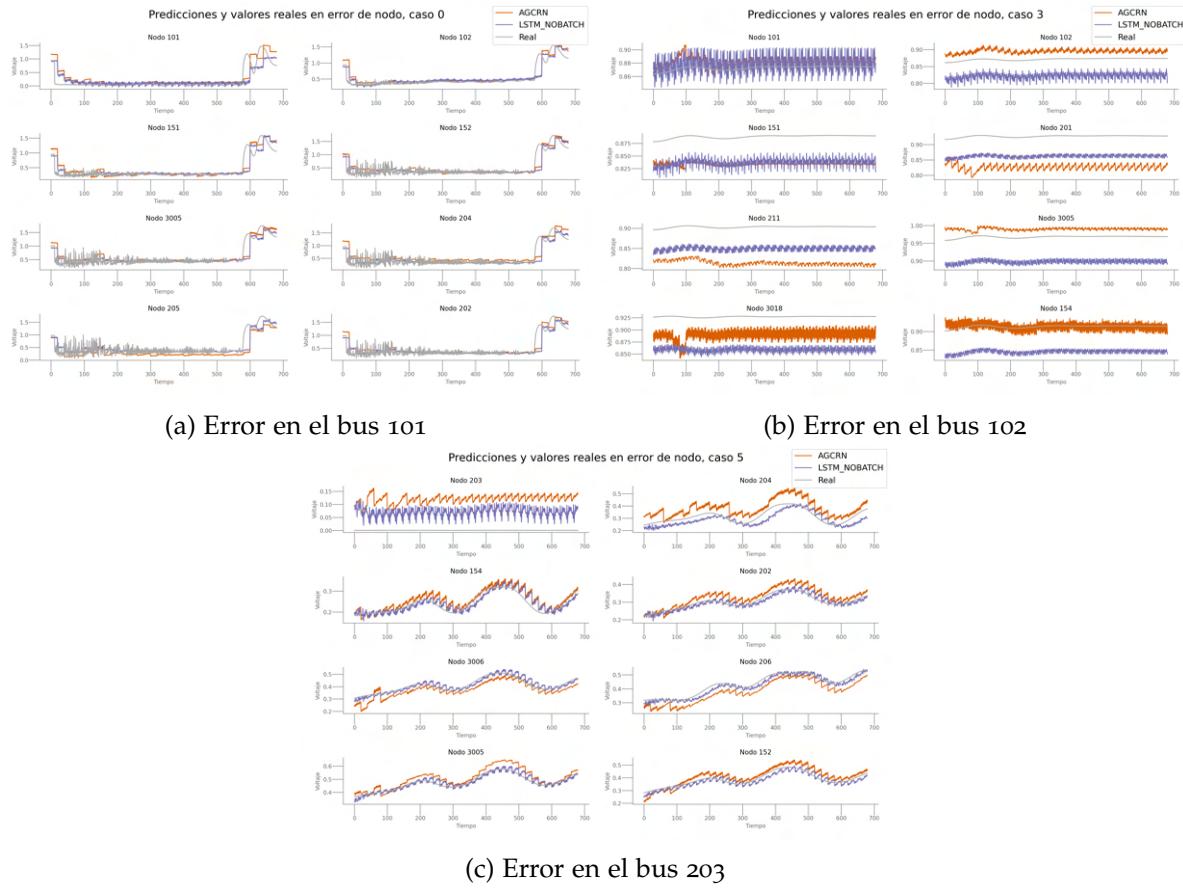


Figura 75: AGCRN

incidencia, se producen predicciones distintas, y en el caso de DCRNN, más alejadas de los valores reales.

En la Figura 79 podemos estudiar los resultados de las arquitecturas MSTGCN y ASTGCN. Si estudiamos en detalle las predicciones de cada arquitectura en la segunda simulación, vemos que el modelo ASTGCN produce los mejores resultados, puesto que capta correctamente los valores de voltaje, y además no contiene tanta perturbación, como era el caso de MPNN-LSTM, visto anteriormente.

Continuamos con la arquitectura MTGNN (Figura 80). En este caso, podemos concluir que esta arquitectura no es adecuada para este tipo de incidencias. Aunque en la primera simulación su rendimiento es similar al de LSTM, en la segunda y tercera simulaciones las predicciones presentan niveles de ruido y perturbación muy altos, lo que impide que se ajusten correctamente a los datos.

En el caso de un modelo con una arquitectura STConv (Figura 81), vemos cómo, pese a ser el modelo más costoso a nivel computacional, es de los que peor rendimiento han tenido hasta el momento, pues no consiguen modelar correctamente los datos en ninguna de las tres situaciones analizadas.

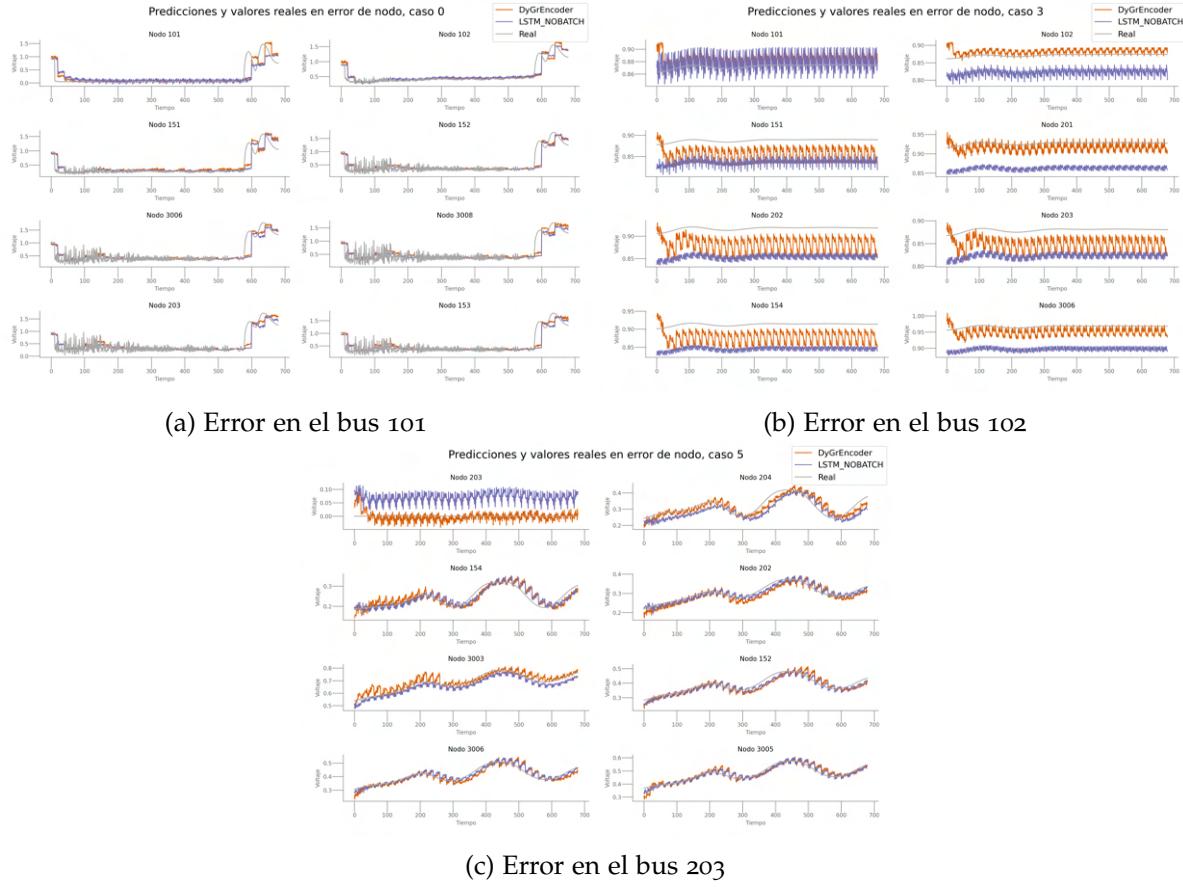


Figura 76: DyGrEncoder

Por último, vemos los resultados de EvolveGCN en la Figura 82. Llegamos a la misma conclusión a la que habíamos llegado con esta arquitectura en otro tipo de simulaciones, y es que

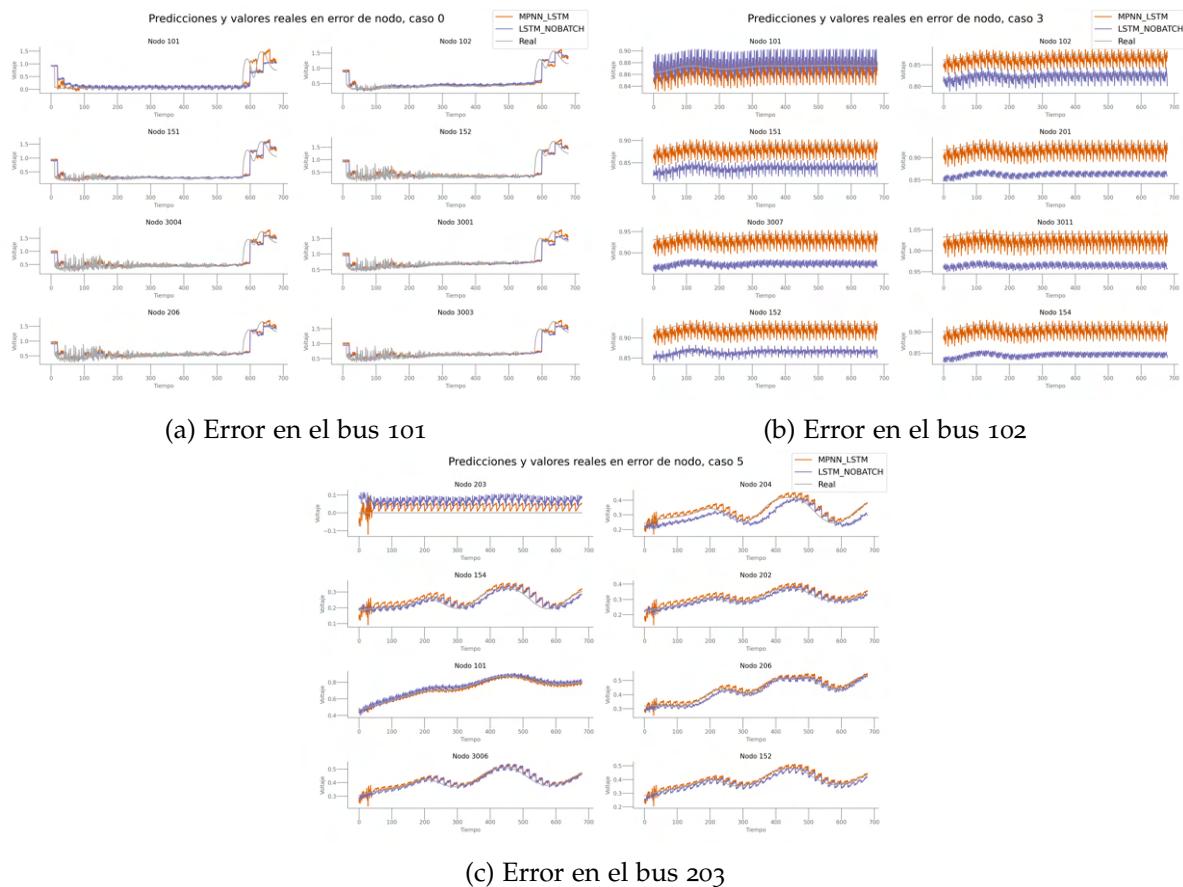
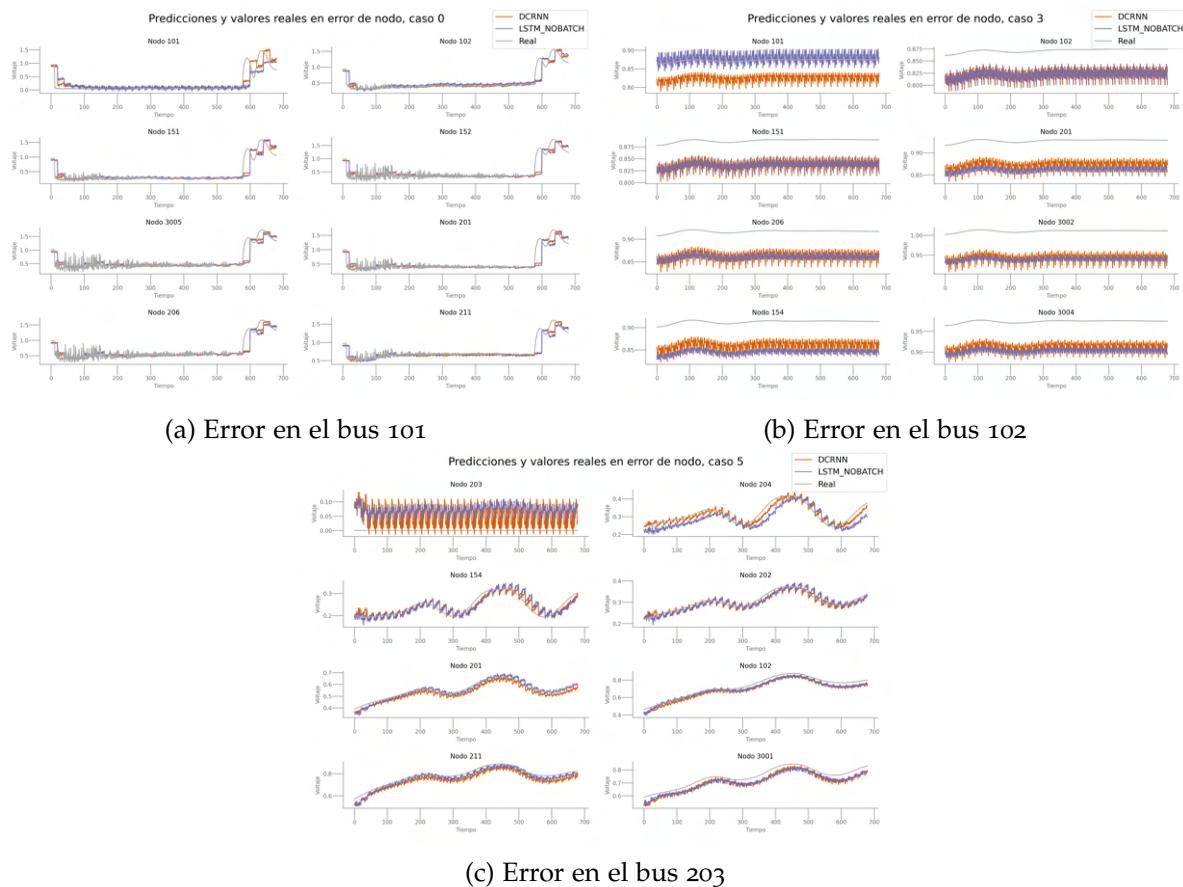


Figura 77: MPNN-LSTM



(c) Error en el bus 203

Figura 78: DCRNN

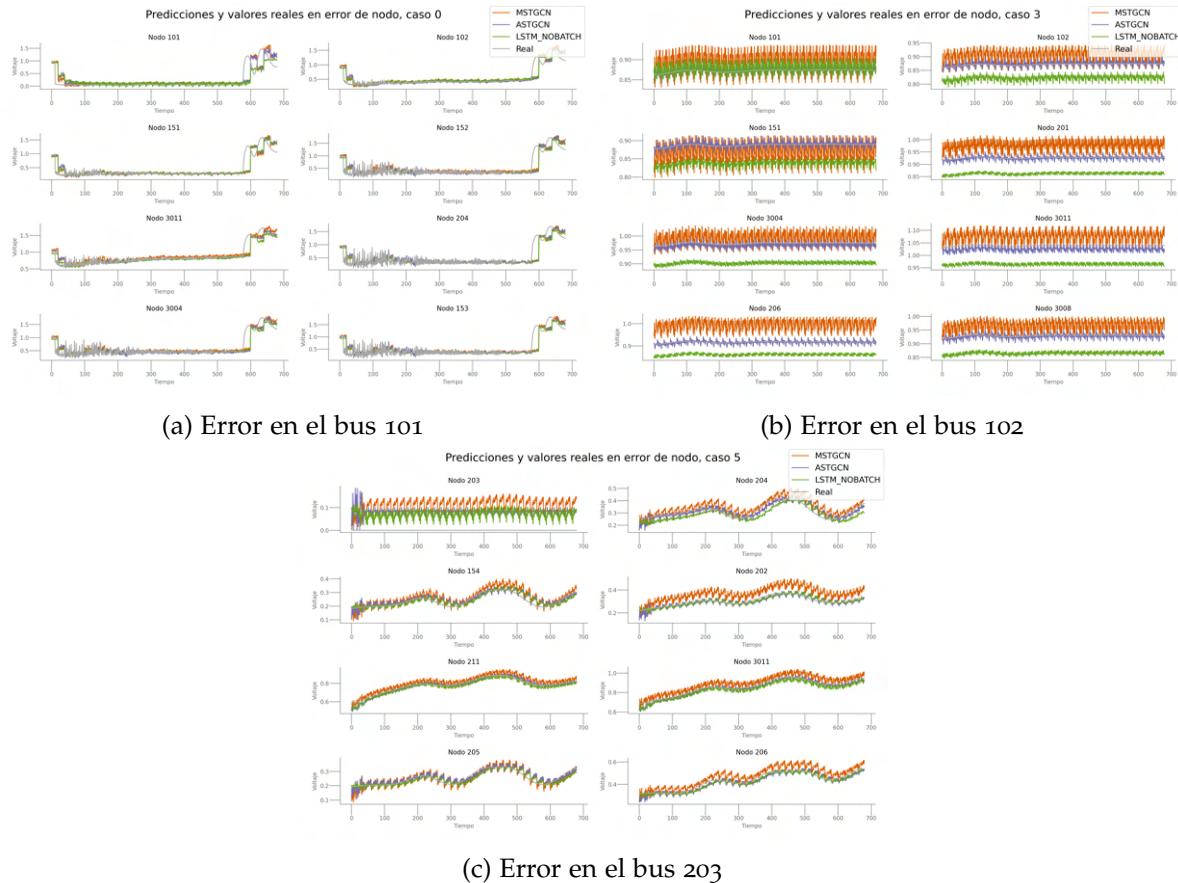


Figura 79: MSTGCN y ASTGCN

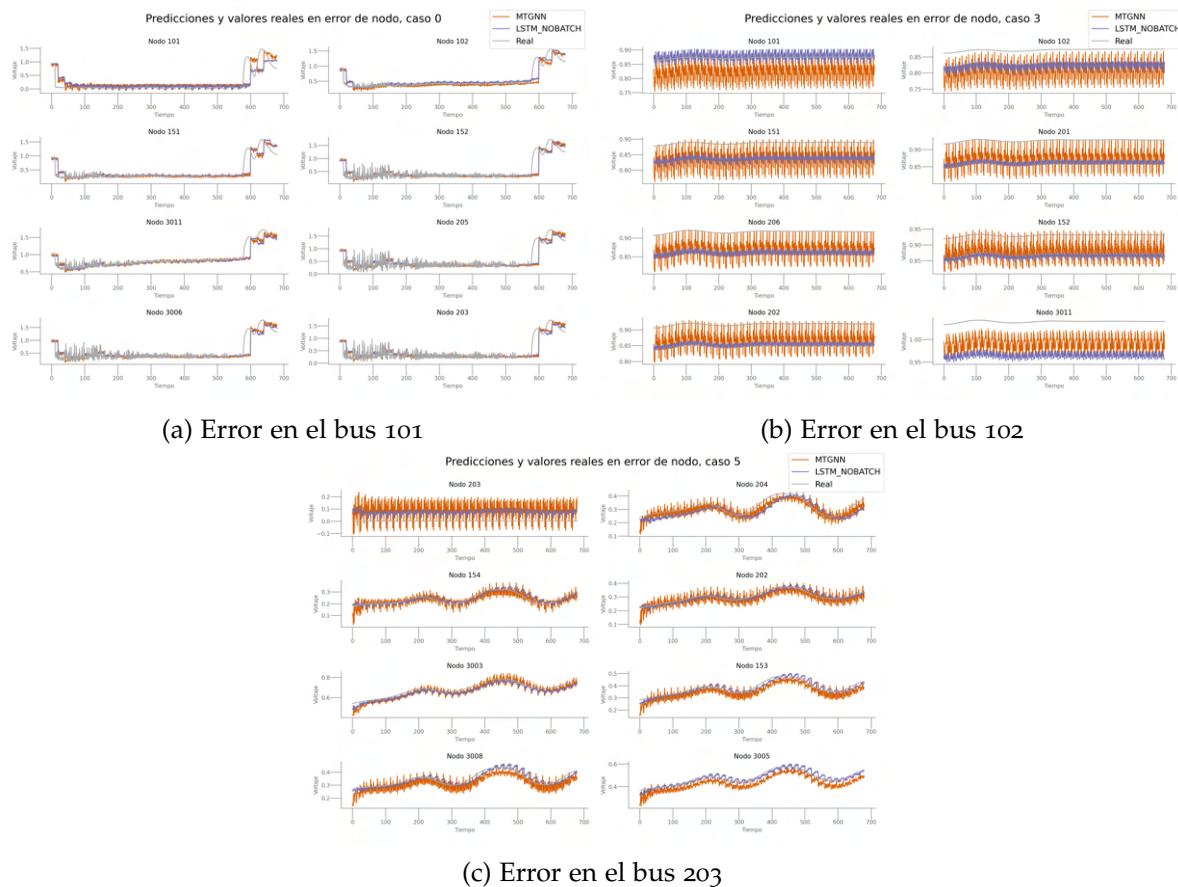


Figura 8o: MTGNN

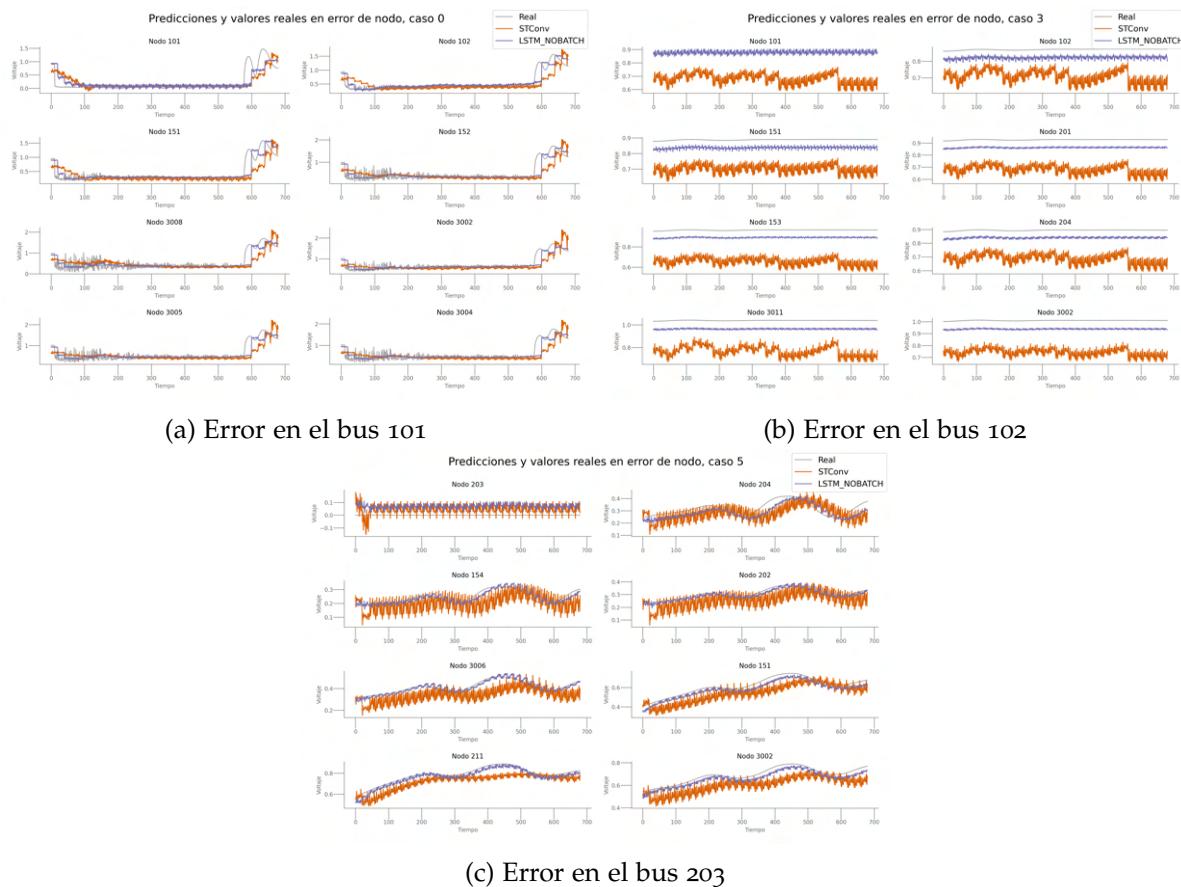


Figura 81: STConv

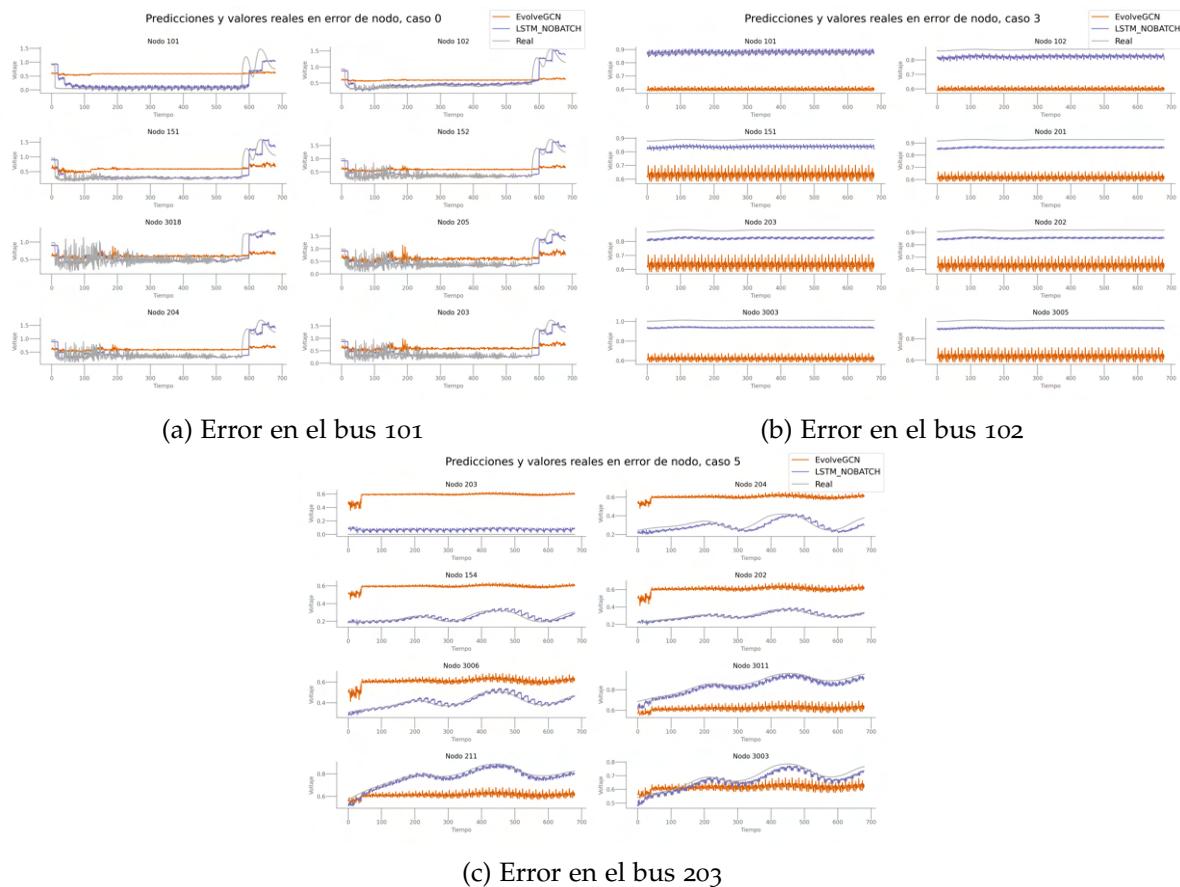


Figura 82: EvolveGCN

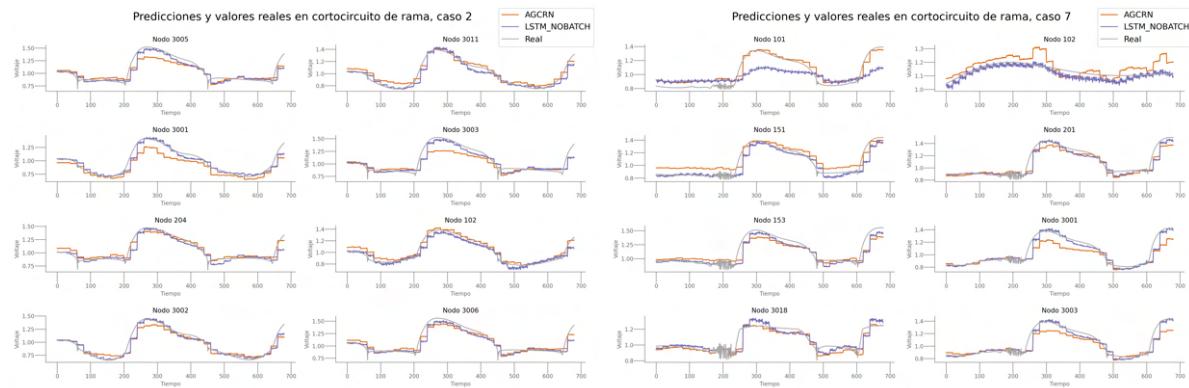
B.4 CORTOCIRCUITO DE RAMA

Finalmente, llegamos a las dos últimas incidencias, que tienen que ver con un fallo en la rama que une dos nodos. En este caso, los ocho nodos escogidos para estudiar las predicciones son los dos nodos que se encuentran conectados en los extremos de la rama, y sus vecindarios correspondientes, y en el caso de que haya que completar, se escogerán nodos aleatorios de la red.

Dado que en este tipo de incidencias la variación de los voltajes en el conjunto de prueba no es tan significativa como en incidencias anteriores, retomaremos el estudio con solo dos situaciones.

En la Figura 83 podemos analizar los resultados de la arquitectura AGCRN. Esta arquitectura destaca especialmente por su superioridad frente a la LSTM en la segunda simulación, donde logra captar correctamente la magnitud de la subida y bajada del voltaje en los dos nodos afectados. Este resultado es significativo, ya que demuestra cómo el conocimiento de los enlaces subyacentes en la red puede mejorar la precisión de las predicciones, que es el objetivo principal de todos los algoritmos estudiados en este trabajo.

Sin embargo, todavía no vemos unas predicciones continuas, que mantengan la forma de los datos reales. Se puede observar a simple vista como las predicciones van reaccionando a los valores reales, pues al fin y al cabo se están prediciendo los siguientes 20 instantes únicamente.



(a) Cortocircuito en la rama: bus 3001 y 3003

(b) Cortocircuito en la rama: bus 102 y 151

Figura 83: AGCRN

Podemos llegar a unas conclusiones parecidas en el caso de DyGrEncoder y MPNN-LSTM (Figuras 84 y 85, respectivamente), excepto que en este caso las predicciones son más *continuas*, es decir, que no hay tanta perturbación como en casos anteriores.

En la Figura 86, donde estudiamos los resultados de DCRNN, vemos cómo se vuelve a dar la misma situación, y en este caso, el ruido asociado a las predicciones de DCRNN es menor, por lo que queda al descubierto la diferencia en cuanto al rendimiento entre ambos modelos.

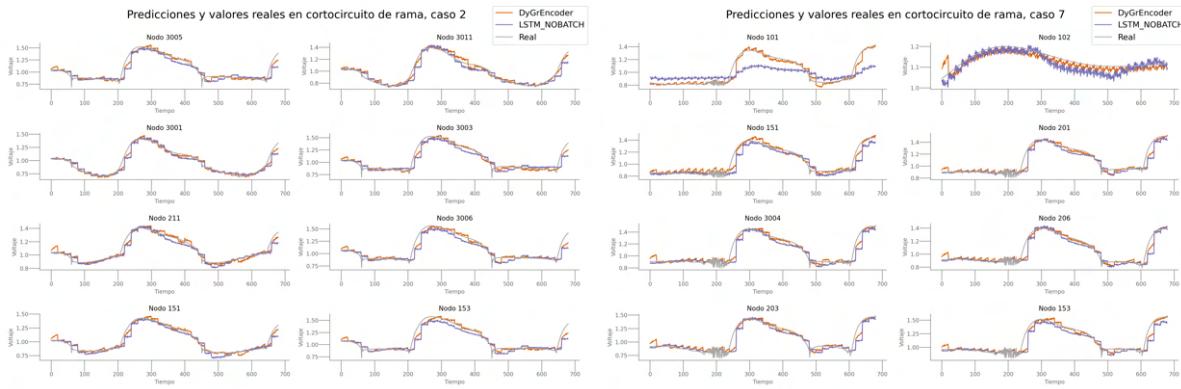


Figura 84: DyGrEncoder

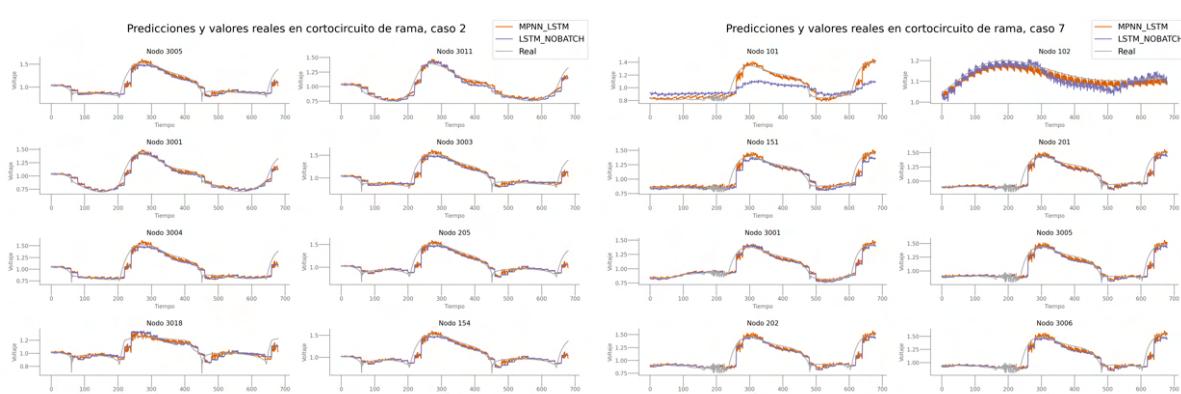


Figura 85: MPNN-LSTM

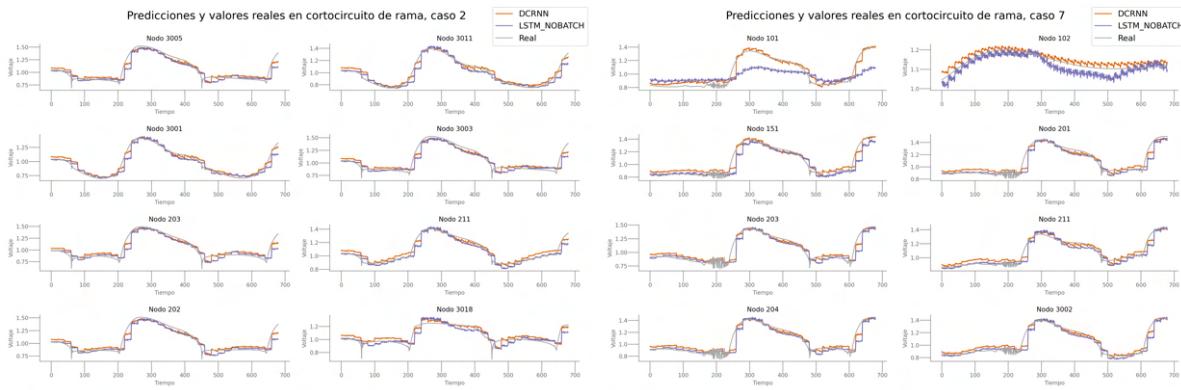


Figura 86: DCRNN

Podemos mantener unas conclusiones muy similares en el caso de ASTGCN y MSTGCN (Figura 87). Sin embargo, se puede realizar además una observación nueva, y es que al completar las figuras con nodos aleatorios, hemos visto cómo en la primera simu-

lación, en el nodo 101, la arquitectura LSTM no capta correctamente la forma de los voltajes. Volvemos a tener un ejemplo, esta vez en una simulación distinta, de una situación en la que tener información sobre las conexiones entre los nodos nos proporciona una mayor precisión en las predicciones, tal y como veíamos en el caso de AGCRN.

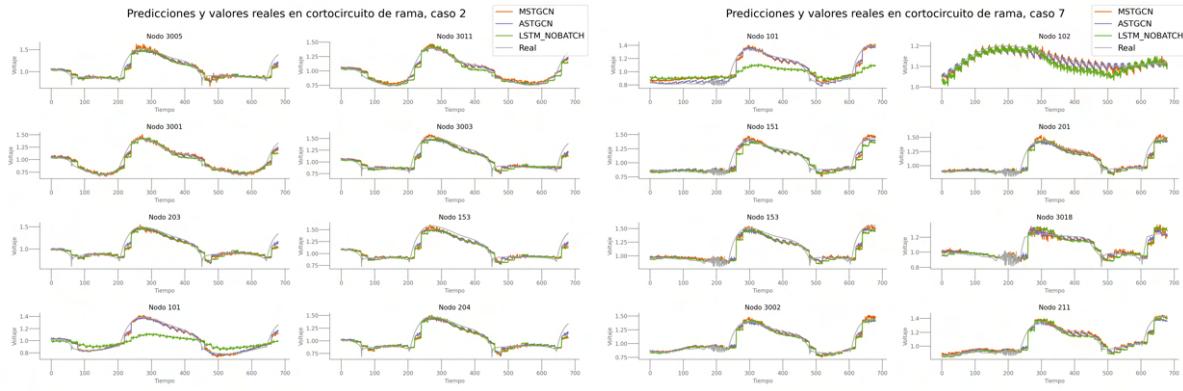


Figura 87: MSTGCN y ASTGCN

Al estudiar los resultados de la arquitectura MTGNN (Figura 88), vemos que, en los picos de voltaje, se obtienen predicciones con un alto nivel de ruido, especialmente en el caso del nodo 152 en la segunda simulación estudiada.

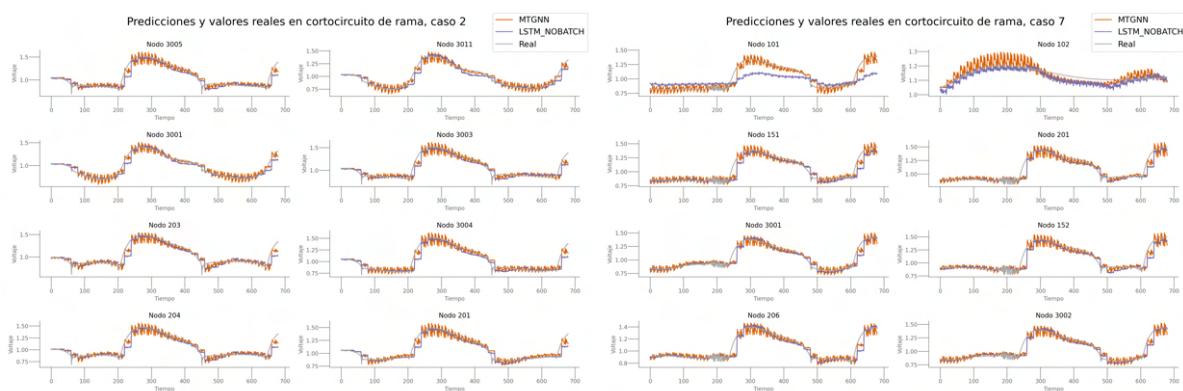
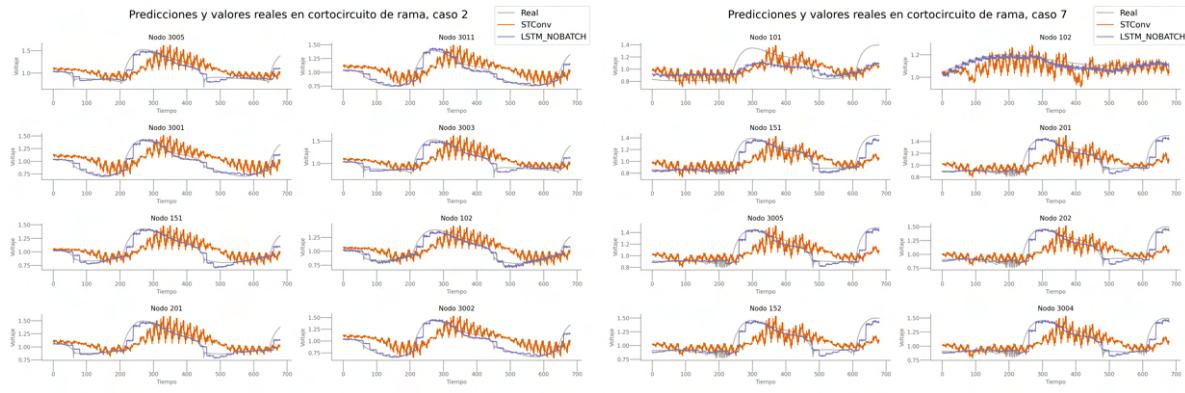


Figura 88: MTGNN

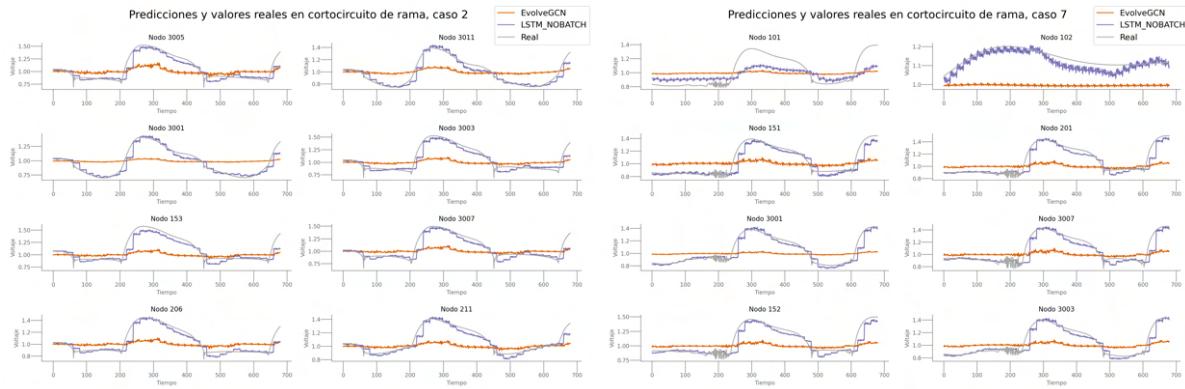
Por último, en las Figuras 89 y 90 se evidencia nuevamente que ninguna de las dos arquitecturas es adecuada para este tipo de datos. Esta falta de adecuación podría atribuirse a varios factores. En primer lugar, el tamaño reducido de la red puede limitar la capacidad de estas arquitecturas para potenciar la información sobre las interacciones entre nodos. Además, la inconsistencia en los datos podría estar afectando la capacidad de estas arquitecturas para generalizar patrones temporales y espaciales de manera efectiva.



(a) Cortocircuito en la rama: bus 3001 y 3003

(b) Cortocircuito en la rama: bus 102 y 151

Figura 89: STConv



(a) Cortocircuito en la rama: bus 3001 y 3003

(b) Cortocircuito en la rama: bus 102 y 151

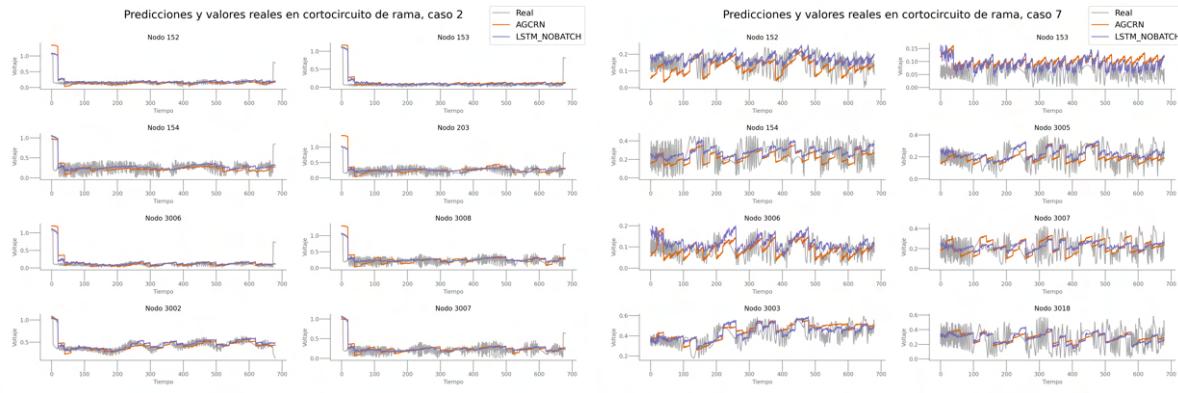
Figura 90: EvolveGCN

B.5 ERROR DE RAMA

Finalizaremos este capítulo estudiando el último tipo de incidencia, el error de rama. Es necesario destacar, al igual que se hizo en el caso del error del nodo, que la forma de las series temporales en la red presenta grandes variaciones entre distintas simulaciones, por lo que, aunque se vayan a analizar los resultados de dos simulaciones del conjunto de prueba, podríamos obtener conclusiones distintas al estudiar las demás.

Comenzamos por la arquitectura AGCRN, cuyos resultados se pueden estudiar en la Figura 91. Podemos ver cómo contrasta el rendimiento de ambos modelos en las dos simulaciones. Por una parte, en la primera, donde tenemos cierta estabilidad, ambos funcionan relativamente bien, aunque no consiguen modelar la bajada repentina. Por otra parte, en la segunda simulación concluimos que ninguna de las dos arquitecturas logra modelar la alta perturbación con precisión.

En el caso de una arquitectura DyGrEncoder (Figura 92), podemos obtener la misma conclusión con respecto a la primera simulación. Para la segunda, sin embargo, podemos observar cómo el modelo predice una subida repentina en dos nodos, que

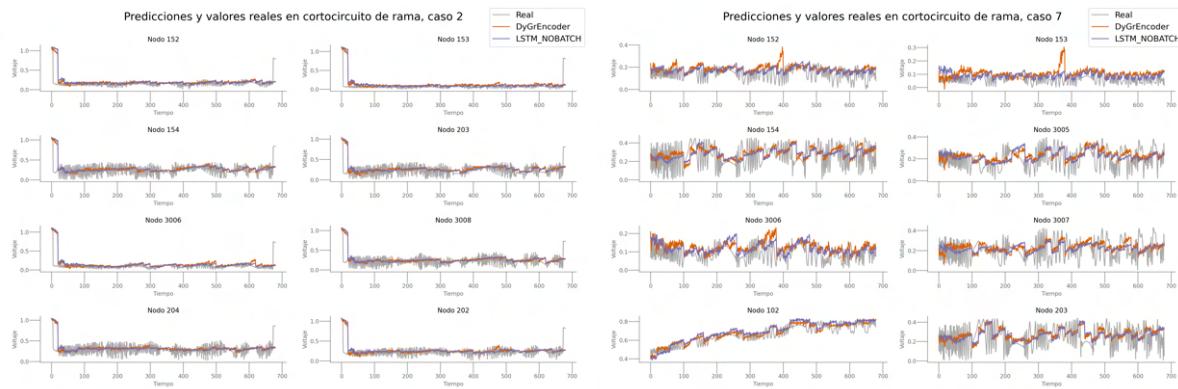


(a) Error en la rama: bus 3005 y 3008

(b) Error en la rama: bus 3005 y 3008

Figura 91: AGCRN

discrepa con los datos reales. Esto podría causar problemas en el caso de querer implantar un modelo de detección de incidencias, que ya hemos visto que es una de las aplicaciones posibles.



(a) Error en la rama: bus 3005 y 3008

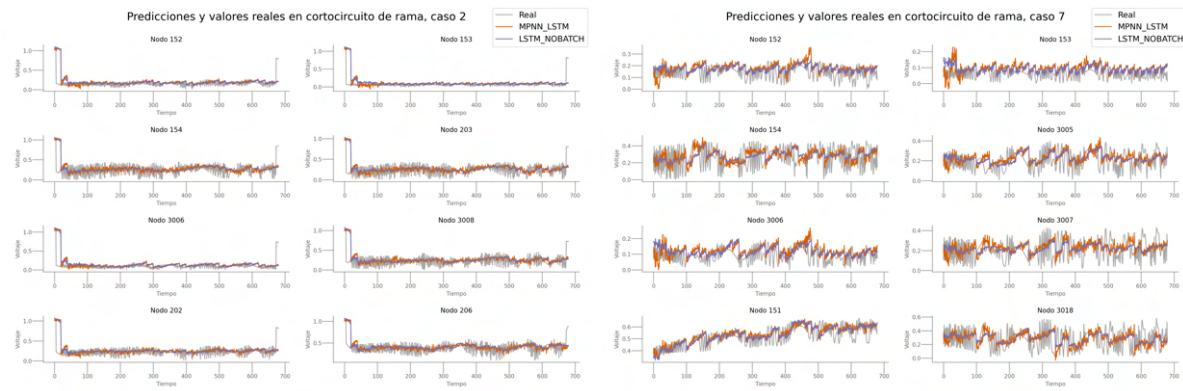
(b) Error en la rama: bus 3005 y 3008

Figura 92: DyGrEncoder

Podríamos llegar a unas conclusiones similares en el caso de MPNN-LSTM y DCRNN (Figuras 93 y 94, respectivamente), pero además se puede observar cómo mejoran las predicciones de la segunda simulación, pues aunque todavía no se captan la mayoría de matrices de los datos reales, se obtienen predicciones más precisas.

Continuamos con las arquitecturas ASTGCN y MSTGCN (Figura 95). Podemos observar cómo MSTGCN funciona mejor que ASTGCN en el caso de la segunda simulación, puesto que ASTGCN produce predicciones que están demasiado alejadas de los valores reales.

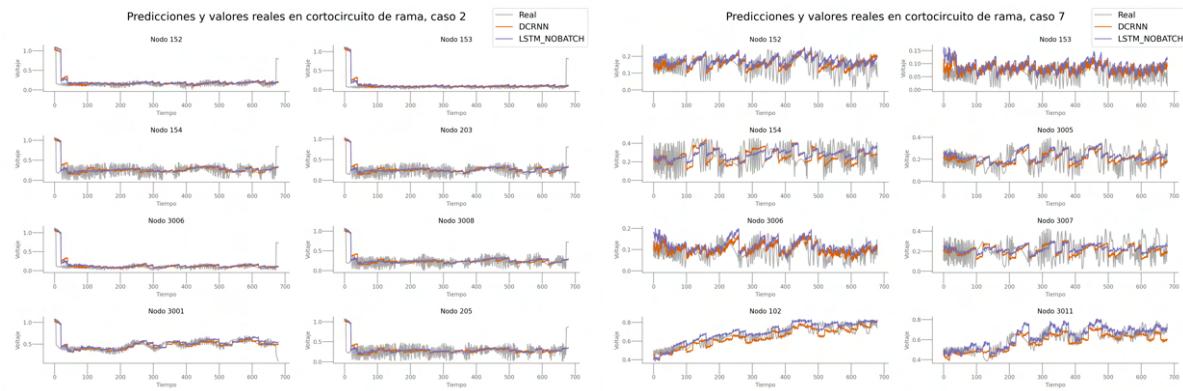
Al estudiar los resultados del modelo MTGNN (Figura 96), vemos claramente como no es una arquitectura adecuada para estos datos. En la segunda simulación, aunque en ciertos nodos produce un rendimiento parecido a LSTM, en otros obtenemos unas predicciones con una perturbación demasiado alta, que imposibilita comprender la



(a) Error en la rama: bus 3005 y 3008

(b) Error en la rama: bus 3005 y 3008

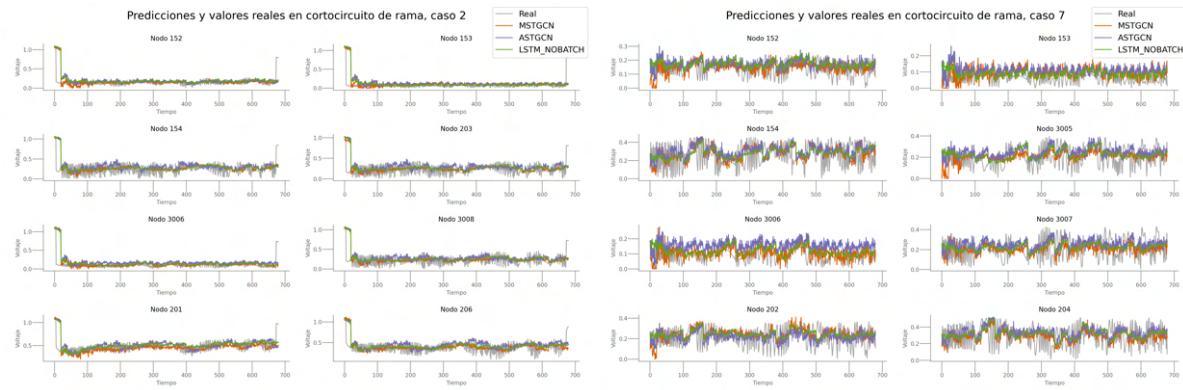
Figura 93: MPNN-LSTM



(a) Error en la rama: bus 3005 y 3008

(b) Error en la rama: bus 3005 y 3008

Figura 94: DCRNN

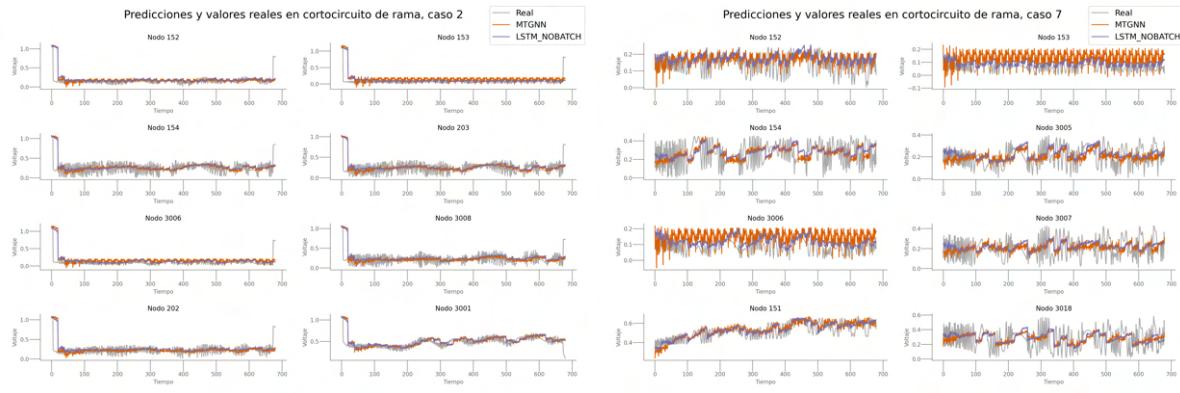


(a) Error en la rama: bus 3005 y 3008

(b) Error en la rama: bus 3005 y 3008

Figura 95: MSTGCN y ASTGCN

estructura de los datos. Esta simulación ejemplifica las situaciones en las que añadir información estructural sobre la red puede afectar negativamente a la precisión de las predicciones.



(a) Error en la rama: bus 3005 y 3008

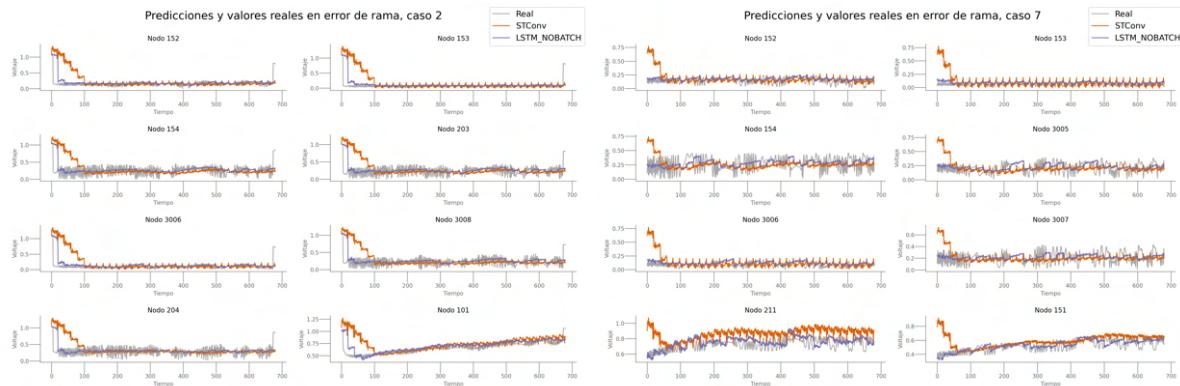
(b) Error en la rama: bus 3005 y 3008

Figura 96: MTGNN

Por último, en las Figuras 98 y 97 podemos estudiar el comportamiento de EvolveGCN y STConv, respectivamente. No resulta difícil concluir algo similar a los casos anteriores, y es que estos modelos no resultan adecuado para estos datos.

En el caso de EvolveGCN, los resultados actuales representan el límite de ajuste que se ha podido alcanzar con los parámetros y la configuración utilizada. Por otro lado, en el caso de STConv, el enfoque es diferente. Aquí, se presenta un desafío adicional relacionado con el gran coste computacional que conlleva el ajuste correcto de sus parámetros.

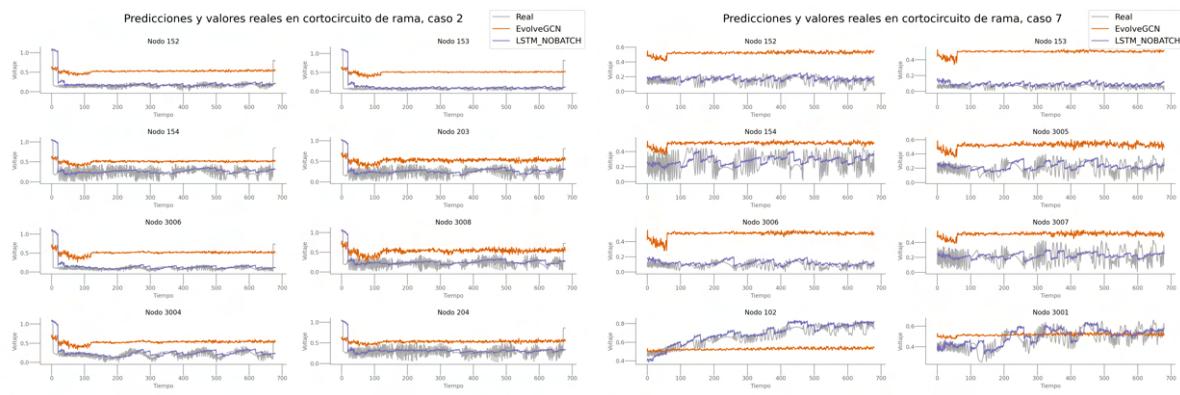
A diferencia de EvolveGCN, STConv podría beneficiarse de un ajuste más exhaustivo de sus hiperparámetros. Sin embargo, este proceso exige un delicado balance entre el tiempo y los recursos computacionales que se deben invertir y las mejoras marginales que se podrían lograr en el rendimiento del modelo. En este sentido, es crucial evaluar si las posibles ganancias en precisión y desempeño justifican el costo asociado con las pruebas adicionales y el ajuste del modelo.



(a) Error en la rama: bus 3005 y 3008

(b) Error en la rama: bus 3005 y 3008

Figura 97: STConv



(a) Error en la rama: bus 3005 y 3008

(b) Error en la rama: bus 3005 y 3008

Figura 98: EvolveGCN