

Universidad Nacional Escuela de Informática

Curso: Estructuras Discretas para Informática

Ciclo I – 2025

GNN – Informe de Avance Fecha del Informe: 08 de mayo de 2025

Integrantes del Grupo:

Allan Daniel Acuña Villarreal

Juan Alexander Aguilar Artola

Isaac Andrés Espinoza Alvarado

Pablo Esteban Díaz Núñez

Grupo: 07 - 1pm

Fecha de Entrega: 16 de junio de 2025

Contenido

1.	Introducción.....	4
1.1.	Motivación y Justificación	4
1.2.	Objetivos y Alcances.....	4
1.3.	Estructura de Trabajo	5
2.	Conceptos de Grafos y sus Aplicaciones.....	6
2.1.	Definiciones.....	6
2.2.	Representaciones	7
2.3.	Métricas Comunes	7
2.4.	Aplicaciones	8
2.4.1.	Redes Sociales.....	8
2.4.2.	Grafos de Conocimiento	8
2.4.3.	Grafos de Citaciones.....	8
3.	Conceptos Básicos de ML usando Redes Neuronales (NNs).....	8
3.1.	Nociones de ML	8
3.1.1.	Definición.....	8
3.1.2.	Features y DataSets.....	9
3.1.3.	Entrenamiento	9
3.1.4.	Tipos de Aprendizaje: Supervisado y No supervisado	9
3.2.	Nociones de NN	10
3.2.1.	Componentes de la Red: Neuronas, Capas, Pesos, Activación	10
3.2.2.	Inferencia por propagación hacia adelante.....	10
3.2.3.	Entrenamiento	10
3.2.4.	Extra: Entrenar una red de reconocimiento de imágenes	12
3.3.	Aplicaciones	12
3.3.1.	Reconocimiento de Imágenes.....	12
3.3.2.	Procesamiento de Lenguajes.....	12
4.	Fundamentos y justificación de GNNs	13
4.1.	Definiciones Básicas	13
4.2.	Limitaciones de NN para Grafos.....	14
4.3.	Tipos y usos de GNNs:	14
4.3.1.	GCNs	14
4.3.2.	GATs.....	14
4.3.3.	GraphSAGE.....	14

5. Caso de Estudio GCN	15
5.1. Definición y Justificación.....	15
5.2. Características Distintivas.....	15
5.3. Modelo de Propagación	16
5.3.1. Features.....	16
5.3.2. Matriz de Adyacencia	16
5.3.3. Matriz de Pesos.....	16
5.3.4. Normalización y Función de Activación	17
5.4. Agregación de features y transformación de nodos.....	17
5.5. Aplicaciones	18
5.5.1. Caso Redes Sociales	18
6. Un Ejemplo Demostrativo de GCN.....	18
6.1. Descripción del Problema (el esperado es una clasificación de nodos) ..	19
6.2. Obtención del Dataset	19
6.3. Configuración de la GCN.....	20
6.4. Descripción del algoritmo e implementación en Python	21
6.5. Visualización y Análisis de Resultados	21
7. Conclusiones	24
8. Referencias.....	24
9. Anexos	25

1. Introducción

1.1. Motivación y Justificación

En la actualidad, una parte significativa de los datos que se generan y utilizan en diversas disciplinas presenta una naturaleza no estructurada o semiestructurada, siendo inadecuados para modelos tradicionales de aprendizaje automático basados en estructuras euclidianas. Las Redes Neuronales de Grafos (Graph Neural Networks, GNNs) surgen como una solución moderna para abordar problemas relacionados con datos relativos, permitiendo modelar directamente la estructura del grafo y las características de los nodos y aristas.

Esta investigación se motiva por el crecimiento acelerado del uso de grafos en áreas como redes sociales, biología computacional, química, procesamiento del lenguaje natural y sistemas de recomendación. Aplicaciones como la clasificación de nodos, predicción de enlaces, y detección de comunidades pueden resolverse de manera más efectiva mediante GNNs. En particular, las Graph Convolutional Networks (GCNs) destacan por su simplicidad y eficiencia, lo que las hace ideales para introducirse al campo de las redes neuronales sobre grafos.

El presente trabajo busca establecer un puente entre la teoría de grafos estudiada en el curso de Estructuras Discretas I y los avances contemporáneos en inteligencia artificial, demostrando cómo conceptos matemáticos fundamentales encuentran aplicación directa en tecnologías emergentes. Se pretende, además, fomentar una visión interdisciplinaria y crítica en el uso de herramientas de IA.

1.2. Objetivos y Alcances

Objetivo General

Comprender, aplicar y analizar los fundamentos de las Graph Neural Networks, con énfasis en la arquitectura GCN, a través de una implementación práctica en Python

y el uso de librerías especializadas, integrando conocimientos teóricos de grafos y aprendizaje automático.

Objetivos Específicos

- Investigar el origen, evolución y fundamentos teóricos de las GNNs, considerando su relación con las redes neuronales tradicionales.
- Estudiar las representaciones comunes de grafos y su importancia en el modelado de problemas reales.
- Analizar el funcionamiento interno de las GCNs, sus mecanismos de propagación de información y las funciones de agregación.
- Implementar un modelo funcional sobre un dataset real (Cora) y realizar experimentos controlados para evaluar su desempeño.
- Elaborar una visualización e interpretación de los resultados, tanto numéricos como gráficos, para facilitar su comprensión.
- Diseñar una segunda demo personalizada con un grafo simulado de red social que permita contrastar con el modelo base.

Alcances

Este proyecto se limita a una exploración introductoria pero profunda del tema de las GNNs, sin pretender alcanzar modelos de estado del arte. La implementación se basa en herramientas de código abierto (PyTorch Geometric) y datasets conocidos como Cora, sin incurrir en complejidad computacional alta. El enfoque es académico, analítico y práctico, con el propósito de fortalecer el entendimiento conceptual y técnico de los estudiantes.

1.3. Estructura de Trabajo

Este informe se encuentra dividido en los siguientes capítulos:

- **Capítulo 1:** Introducción, motivación, objetivos y justificación del trabajo.
- **Capítulo 2:** Revisión de los conceptos fundamentales de grafos y sus aplicaciones más comunes.

- **Capítulo 3:** Fundamentos del aprendizaje automático y redes neuronales artificiales.
- **Capítulo 4:** Justificación del uso de GNNs y descripción teórica de las GCNs.
- **Capítulo 5:** Implementación práctica de un modelo GCN utilizando el dataset Cora.
- **Capítulo 6:** Resultados obtenidos, análisis de métricas y visualizaciones generadas.
- **Capítulo 7:** Conclusiones, reflexiones y trabajos futuros.
- **Anexos:** Código fuente, enlaces a los repositorios y bibliografía consultada.

2. Conceptos de Grafos y sus Aplicaciones

2.1. Definiciones

Un grafo G es una estructura matemática fundamental definida como un par ordenado $G = (V, E)$, donde:

- **V** es el conjunto de vértices o nodos.
- **E** es el conjunto de aristas o enlaces, que representan relaciones entre pares de nodos.

Los grafos pueden clasificarse en dirigidos (digrafos) o no dirigidos, dependiendo de si las aristas tienen una dirección asociada. Asimismo, pueden ser ponderados si cada arista lleva un peso numérico, y simples o multigrafos según la existencia de aristas múltiples entre nodos.

Esta abstracción permite modelar problemas reales donde lo relevante no es solo la existencia de elementos individuales, sino la forma en que están relacionados entre sí.

2.2. Representaciones

En el contexto computacional, los grafos pueden representarse de distintas maneras, cada una con sus ventajas:

- **Matriz de Adyacencia:** Estructura matricial de tamaño $|V| \times |V|$. Su ventaja es la rapidez en la consulta de existencia de aristas, aunque su uso de memoria puede ser elevado en grafos dispersos.
- **Lista de Adyacencia:** Utiliza estructuras como listas o diccionarios para almacenar los vecinos de cada nodo. Es eficiente en memoria y permite recorrer los vecinos rápidamente.
- **Lista de Aristas:** Representa el grafo como una lista de pares de nodos. Es útil para manipulación directa o representación gráfica.

2.3. Métricas Comunes

El análisis estructural de un grafo se basa en ciertas métricas que permiten caracterizar su comportamiento y propiedades:

- **Grado de un nodo:** Número de aristas incidentes a un nodo. En grafos dirigidos se distingue entre grado de entrada y de salida.
- **Centralidad:** Mide la influencia o importancia relativa de un nodo. Ejemplos incluyen centralidad de grado, cercanía y betweenness.
- **Caminos mínimos:** Distancia más corta entre pares de nodos, calculada mediante algoritmos como Dijkstra o Floyd-Warshall.
- **Componentes conexas:** Subgrafos en los que todos los nodos están conectados entre sí.

2.4. Aplicaciones

2.4.1. Redes Sociales

Las plataformas sociales pueden modelarse como grafos donde los nodos son usuarios y las aristas representan relaciones de amistad o interacción. Las GNNs permiten realizar tareas como detección de comunidades, recomendación de amigos, análisis de influencia y clasificación de usuarios por afinidad.

2.4.2. Grafos de Conocimiento

Se utilizan en inteligencia artificial y semántica web para representar entidades (nodos) y relaciones (aristas). Permiten realizar inferencias automáticas y razonamiento lógico sobre bases de conocimiento estructuradas.

2.4.3. Grafos de Citaciones

Comunes en bibliometría, modelan la relación de citación entre documentos académicos. Permiten estudiar la relevancia de publicaciones, predecir tendencias de investigación y categorizar documentos según sus relaciones.

3. Conceptos Básicos de ML usando Redes Neuronales (NNs)

3.1. Nociones de ML

3.1.1. Definición

El aprendizaje automático (Machine Learning, ML) es una disciplina de la inteligencia artificial que permite a los sistemas informáticos identificar patrones y

realizar predicciones o decisiones sin ser programados de manera explícita para cada tarea. Esta definición coincide con lo planteado por Mitchell (1997), quien describe el aprendizaje automático como la capacidad de un sistema para mejorar su rendimiento en una tarea mediante la experiencia.

3.1.2. Features y DataSets

Un dataset es una colección estructurada de ejemplos o instancias. Cada ejemplo se representa mediante un vector de características o **features**, que capturan información relevante para el problema a resolver. El diseño y selección de estas características influye directamente en el desempeño del modelo.

3.1.3. Entrenamiento

El proceso de entrenamiento consiste en ajustar los parámetros del modelo (pesos y sesgos) mediante algoritmos de optimización como el descenso del gradiente. El objetivo es minimizar una función de pérdida que cuantifica el error entre la salida esperada y la obtenida.

3.1.4. Tipos de Aprendizaje: Supervisado y No supervisado

- **Supervisado:** Utiliza ejemplos con etiquetas conocidas para entrenar el modelo.
- **No supervisado:** Busca patrones ocultos en datos no etiquetados, como agrupamientos.
- **Reforzado:** Basado en recompensas o castigos según acciones del agente.

3.2. Nociones de NN

3.2.1. Componentes de la Red: Neuronas, Capas, Pesos, Activación

Una red neuronal artificial se compone de:

- **Neuronas:** Unidades básicas de procesamiento.
- **Capas:** Agrupaciones de neuronas en secuencias (entrada, ocultas y salida).
- **Pesos y Bias:** Parámetros ajustables que modulan la información.
- **Funciones de Activación:** Introducen no linealidad (ReLU, Sigmoid, Tanh).

3.2.2. Inferencia por propagación hacia adelante

Durante la inferencia, los datos se procesan capa por capa multiplicando por los pesos y aplicando funciones de activación. El resultado es una predicción que se compara contra la etiqueta real para calcular el error.

3.2.3. Entrenamiento

3.2.3.1. Función de pérdida

La función de pérdida, también llamada función de costo, es un componente esencial en el entrenamiento de una red neuronal. Su propósito es cuantificar qué tan lejos están las predicciones del modelo respecto a los valores reales. En otras palabras, mide el error cometido en cada paso del aprendizaje y sirve como señal para que el modelo se ajuste.

Existen diferentes funciones de pérdida dependiendo del tipo de tarea:

- **Error cuadrático medio (MSE - Mean Squared Error):** Utilizada principalmente en problemas de regresión. Calcula la media de los cuadrados de las diferencias entre los valores predichos y los reales. Penaliza fuertemente errores grandes.
- **Entropía cruzada (Cross-Entropy Loss):** Usada en problemas de clasificación, especialmente cuando las salidas son probabilidades. Evalúa la distancia entre la distribución de probabilidad predicha y la real. Es especialmente útil cuando se busca que el modelo aprenda a distinguir entre múltiples clases.

Una buena elección de la función de pérdida es crucial para el correcto entrenamiento del modelo, ya que influye directamente en la dirección y magnitud de los ajustes que se realizarán sobre los parámetros internos.

3.2.3.2. Propagación hacia atrás

La propagación hacia atrás, o **backpropagation**, es el algoritmo central mediante el cual una red neuronal aprende. Consiste en calcular cómo cambia la función de pérdida con respecto a cada uno de los pesos de la red, usando el método del **gradiente descendente**.

Este proceso se realiza en dos fases:

1. **Forward pass (propagación hacia adelante):** Los datos de entrada pasan por las capas de la red hasta obtener una predicción. Luego se calcula la función de pérdida con esa predicción.
2. **Backward pass (propagación hacia atrás):** A partir del valor de la pérdida, se aplican las reglas del cálculo diferencial para obtener los gradientes de cada peso, utilizando la regla de la cadena. Estos gradientes indican la dirección en la que cada peso debe cambiar para reducir la pérdida.

Luego, mediante un optimizador como el **descenso del gradiente estocástico (SGD)** o **Adam**, los pesos son actualizados en esa dirección. El ciclo se repite durante múltiples épocas hasta que la red alcanza un rendimiento aceptable.

3.2.4. Extra: Entrenar una red de reconocimiento de imágenes

3.3. Aplicaciones

Las redes neuronales artificiales tienen un amplio rango de aplicaciones en el mundo real. A continuación, se destacan dos de las más influyentes:

3.3.1. Reconocimiento de Imágenes

Las **redes neuronales convolucionales (CNNs)** están diseñadas para procesar datos con una estructura de cuadrícula, como las imágenes. Usan filtros (kernels) que recorren la imagen para detectar patrones visuales como bordes, texturas y formas.

Aplicaciones prácticas incluyen:

- Diagnóstico médico por imágenes (detección de tumores en resonancias o rayos X).
- Sistemas de vigilancia y reconocimiento facial.
- Clasificación automática de objetos en imágenes.
- Vehículos autónomos que interpretan señales y obstáculos en tiempo real.

Gracias a las CNNs, hoy es posible alcanzar niveles de precisión superiores al 90% en tareas de visión por computadora.

3.3.2. Procesamiento de Lenguajes

El lenguaje humano es complejo, ambiguo y contextual, lo cual lo convierte en un gran reto para las máquinas. Para abordarlo, se utilizan arquitecturas como:

- **Redes Recurrentes (RNNs):** Capturan dependencias temporales en secuencias de texto, útiles para tareas donde el orden importa, como predicción de texto y análisis de sentimientos.
- **Transformers:** Arquitectura más reciente y poderosa que se ha convertido en el estándar del NLP moderno. Permiten procesar secuencias en paralelo y capturar relaciones a largo plazo en un texto, lo cual ha llevado al desarrollo de modelos como BERT y GPT.

Aplicaciones destacadas incluyen:

- Traductores automáticos multilingües.
- Motores de búsqueda inteligentes.
- Sistemas de recomendación basados en texto.
- Asistentes virtuales como Alexa, Siri o ChatGPT.

4. Fundamentos y justificación de GNNs

4.1. Definiciones Básicas

Las Graph Neural Networks (GNNs) son arquitecturas de redes neuronales diseñadas específicamente para trabajar con datos estructurados en forma de grafo. Cada nodo del grafo tiene un vector de características, y la arquitectura de la red permite compartir información entre nodos a través de sus conexiones. Como explican Wu et al. (2020), esta capacidad de integrar estructura topológica y atributos de los nodos es una de las principales fortalezas de las GNN frente a otros modelos.

En esencia, una GNN aprende una representación (embedding) para cada nodo, que captura tanto sus atributos como la estructura de sus vecinos. Esto permite aplicar modelos de aprendizaje profundo a problemas donde la relación entre entidades es tan importante como las entidades mismas.

4.2. Limitaciones de NN para Grafos

Los modelos tradicionales, como las MLPs, están diseñados para entradas con estructura fija (vectores o matrices). En contraste, los grafos presentan características complejas:

- No hay orden definido en los nodos vecinos.
- El número de vecinos puede variar.
- La conectividad entre nodos influye directamente en el significado de los datos.

Estas características hacen inadecuados los modelos clásicos para tareas que dependen de la topología del grafo, y justifican el uso de GNNs.

4.3. Tipos y usos de GNNs:

4.3.1. GCNs

Aplican operaciones similares a convoluciones en el dominio del grafo. Son eficientes y fáciles de entrenar.

4.3.2. GATs

Introducen mecanismos de atención para ponderar la importancia de cada vecino.

4.3.3. GraphSAGE

Aprende funciones de agregación basadas en muestreo, adecuado para grafos grandes y dinámicos.

5. Caso de Estudio GCN

5.1. Definición y Justificación

Para ilustrar el funcionamiento de una red GCN, se optó por utilizar el conjunto de datos Cora, ampliamente empleado en la literatura como benchmark. Este dataset representa una red de citaciones entre artículos académicos en el campo de las ciencias de la computación. Cada nodo es un artículo, cada arista representa una citación entre artículos, y las características de cada nodo consisten en un vector binario que indica la presencia o ausencia de ciertas palabras clave.

Este dataset es ideal para validar una arquitectura GCN porque tiene una estructura de grafo definida, clases conocidas, y es suficientemente pequeño para entrenamiento en equipos personales, pero suficientemente complejo para ilustrar conceptos clave.

5.2. Características Distintivas

Las Graph Convolutional Networks (GCNs) se diferencian de otras redes neuronales por su capacidad para operar directamente sobre grafos, es decir, sobre estructuras que modelan relaciones entre entidades. A diferencia de las redes convolucionales tradicionales (CNNs), que se aplican sobre datos en grillas como imágenes, las GCNs procesan nodos interconectados en estructuras arbitrarias. Entre sus características más relevantes se incluyen:

- La capacidad de compartir información entre nodos a través de sus vecinos.
- El uso de funciones de agregación que son invariantes al orden de los vecinos.
- La integración de la topología del grafo dentro del proceso de aprendizaje.

Estas propiedades hacen a las GCNs especialmente útiles para tareas como clasificación de nodos, predicción de enlaces y generación de representaciones vectoriales de grafos completos.

5.3. Modelo de Propagación

El núcleo de las GCNs radica en su mecanismo de propagación de información, el cual permite a cada nodo actualizar su representación interna combinando su información propia con la de sus vecinos. Este proceso puede entenderse como una generalización de las convoluciones tradicionales a dominios no euclidianos.

5.3.1. Features

Cada nodo en un grafo tiene un vector de características asociado, que representa su información individual. Estas características pueden incluir atributos numéricos, indicadores binarios o vectores de palabras, dependiendo del contexto de la aplicación. El conjunto de características de todos los nodos se representa como una matriz X de tamaño $N \times F$, donde N es el número de nodos y F es la cantidad de características por nodo.

5.3.2. Matriz de Adyacencia

La estructura del grafo se codifica mediante una matriz de adyacencia A , de tamaño $N \times N$, donde $A[i][j] = 1$ si existe una arista entre los nodos i y j , y 0 en caso contrario. Para mejorar la propagación y estabilizar el aprendizaje, se suele trabajar con una versión normalizada de esta matriz, que incluye autoconexiones ($\tilde{A} = A + I$).

5.3.3. Matriz de Pesos

Durante el entrenamiento, las GCNs aprenden matrices de pesos W que permiten transformar las características agregadas en nuevas representaciones útiles para la tarea. Cada capa convolucional multiplica la entrada agregada por una matriz de pesos, produciendo una salida transformada que luego pasa a través de una función de activación.

5.3.4. Normalización y Función de Activación

La normalización se aplica sobre la matriz de adyacencia para asegurar que cada nodo contribuya proporcionalmente al mensaje que envía. La normalización

$$\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2}$$

simétrica más común es $\hat{D}^{-1/2} \hat{A} D^{-1/2}$, donde D es la matriz diagonal de grados.

La función de activación, como ReLU (Rectified Linear Unit), se aplica después de cada propagación para introducir no linealidad, lo cual es fundamental para que la red pueda aprender representaciones complejas.

5.4. Agregación de features y transformación de nodos

En cada capa de una GCN, se realiza una operación de agregación que recopila información de los vecinos de un nodo y la combina con su propia información. Esta operación tiene como objetivo producir un vector de características actualizado que refleje tanto el contexto local del nodo como sus propias propiedades.

El proceso puede describirse matemáticamente como:

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

donde:

- $H^{(l)}$ es la matriz de características en la capa l
- \hat{A} es la matriz de adyacencia normalizada con autoconexiones
- $W^{(l)}$ son los pesos aprendidos en la capa l
- σ es la función de activación (como ReLU)

Este mecanismo permite que los nodos "escuchen" a sus vecinos y ajusten sus vectores internos para representar patrones estructurales y semánticos del grafo.

5.5. Aplicaciones

Las GCNs han demostrado su eficacia en tareas como:

- Clasificación de nodos en redes sociales o académicas
- Predicción de enlaces en sistemas de recomendación
- Análisis de estructuras moleculares en química y biología
- Detección de anomalías en sistemas de comunicación

5.5.1. Caso Redes Sociales

Uno de los ejemplos más ilustrativos de uso de GCNs es su aplicación a redes sociales. En este contexto, cada nodo representa a un usuario y las aristas reflejan relaciones como amistad, seguidores o interacciones. Las características del nodo pueden incluir edad, ubicación, intereses, actividad reciente, etc.

La GCN permite, por ejemplo, predecir el tipo de usuario (perfil) según su comportamiento y el de su círculo social. Esta capacidad de integrar información contextual y estructural es lo que diferencia a las GCNs de otros modelos de clasificación tradicionales, y abre la puerta a aplicaciones como:

- Recomendación de contactos
- Detección de bots o usuarios falsos
- Segmentación de mercado basada en comunidades

Este tipo de ejemplos motivó la creación de una demo visual alternativa dentro del presente trabajo, donde se genera un grafo simulado de red social para ilustrar el funcionamiento interno de una GCN con datos más comprensibles e interpretables por humanos, en contraste con la complejidad del dataset Cora.

6. Un Ejemplo Demostrativo de GCN

Este capítulo presenta una serie de ejemplos implementados con Graph Convolutional Networks (GCN) aplicadas sobre redes sociales ficticias. Se desarrollaron

cuatro versiones del modelo con mejoras progresivas, partiendo de una red básica hasta una red social ampliada con más nodos y atributos. A lo largo del capítulo se muestra cómo evoluciona el rendimiento del modelo, así como los códigos y gráficos más representativos.

6.1. Herramientas y Librerías Utilizadas

Para el desarrollo del modelo se utilizaron herramientas del ecosistema Python, destacando:

- **PyTorch**: biblioteca de machine learning para definir y entrenar modelos de redes neuronales.
- **PyTorch Geometric (PyG)**: extensión para trabajar con estructuras de grafos.
- **Pandas**: manejo de estructuras de datos tabulares (CSV).
- **Matplotlib**: generación de gráficos.
- **NetworkX**: construcción y visualización de grafos.

Estas herramientas permitieron construir, entrenar y visualizar los modelos de GCN.

6.2. Descripción del Problema (el esperado es una clasificación de nodos)

El problema a resolver es una clasificación de nodos dentro de una red social ficticia. Cada nodo representa una persona, y se busca predecir su tipo de interés (gamer, artista, deportista, o académico) según sus características y conexiones.

El modelo GCN aprovecha la información estructural del grafo (adyacencias) y los atributos individuales de cada nodo para clasificar de forma más precisa. Esta técnica se basa en el trabajo pionero de Kipf y Welling (2017), quienes demostraron que las convoluciones sobre grafos pueden propagar y agregar información útil para tareas de clasificación.

6.3. Obtención del Dataset

Se construyó un dataset sintético en formato CSV llamado `dataset_red_social_ampliada.csv`, donde cada fila representa un nodo con los siguientes campos:

- Edad
- Gusto por los juegos
- Gusto por el arte
- Gusto por el deporte
- Gusto por lo académico
- Nombre del nodo (ID)
- Clase objetivo (etiqueta)

Este archivo fue usado en todas las versiones del modelo, desde la básica hasta la avanzada. El mismo fue procesado con pandas y networkx para construir la estructura del grafo y alimentar la red neuronal. Según Wu et al. (2020), la representación de grafos enriquecida con atributos es una de las características que distingue a las GNN de otros enfoques de aprendizaje estructurado.

6.4. Configuración de la GCN

La red GCN se define como una clase que hereda de `torch.nn.Module`. La arquitectura básica está compuesta por dos capas convolucionales sobre grafos:

```
class GCN(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, output_dim)

    def forward(self, data):
        x, edge_index = data.people_features, data.connections
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training, p=0.3)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

A medida que se implementaron versiones más avanzadas, se ajustaron hiperparámetros como:

- Número de épocas (epochs)
- Tamaño de capa oculta
- Uso de dropout
- Aumento de nodos y atributos

Estas mejoras permitieron estabilizar el aprendizaje y aumentar la precisión. Kipf y Welling (2017) ya habían señalado que incluso una arquitectura simple de dos capas podía ser suficiente para obtener buenos resultados en tareas semi-supervisadas sobre grafos.

6.5. Descripción del algoritmo e implementación en Python

El entrenamiento del modelo se realizó mediante el uso de la función de pérdida nll_loss y un optimizador Adam. Durante cada iteración se calculó la salida del modelo, la pérdida, y se realizó el paso de retropropagación.

```
for epoch in range(1, 201):
    model.train()
    optimizer.zero_grad()
    out = model(data)
    loss = F.nll_loss(out[data.train_mask], data.labels[data.train_mask])
    loss.backward()
    optimizer.step()

    model.eval()
    _, pred = out[data.test_mask].max(dim=1)
    correct = pred.eq(data.labels[data.test_mask]).sum().item()
    accuracy = correct / data.test_mask.sum().item()
    results.append((epoch, loss.item(), accuracy))
```

Además, en cada demo se almacenaron los resultados de precisión y pérdida en un archivo CSV para su posterior análisis:

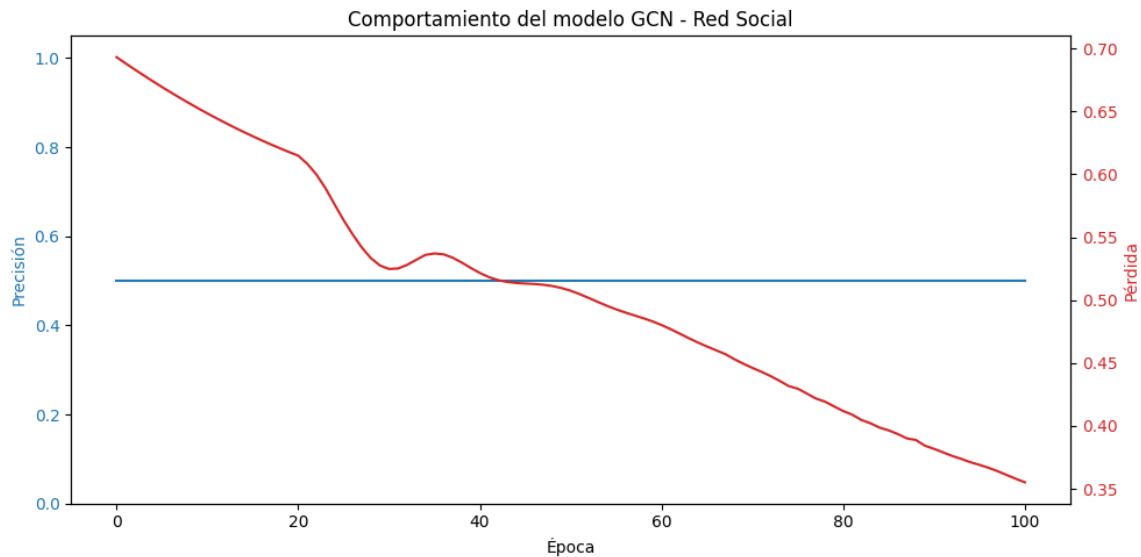
- resultados_entrenamiento_red_social.csv
- resultados_entrenamiento_red_social_mejorada.csv
- resultados_entrenamiento_red_social_mejorada_v2.csv
- resultados_entrenamiento_red_social_avanzada.csv

Como indican Wu et al. (2020), uno de los retos clave en la implementación de GNNs es encontrar un balance entre la complejidad del modelo y la estabilidad del entrenamiento, especialmente al aumentar el número de capas o nodos.

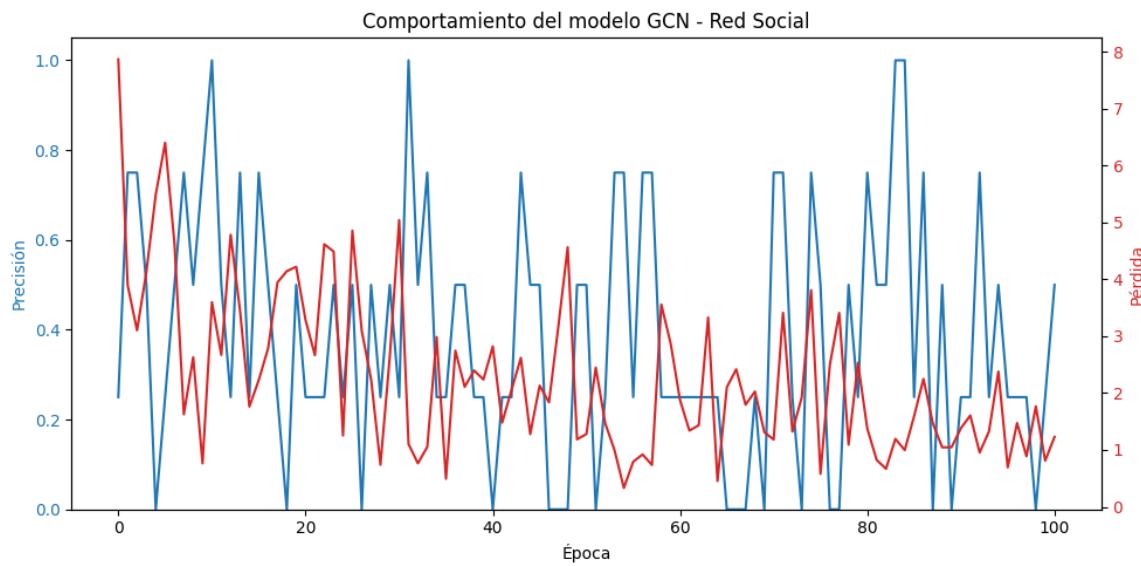
6.6. Visualización y Análisis de Resultados

Los resultados de entrenamiento se representaron en gráficos de precisión y pérdida a lo largo de las épocas, comparando entre las distintas versiones del modelo.

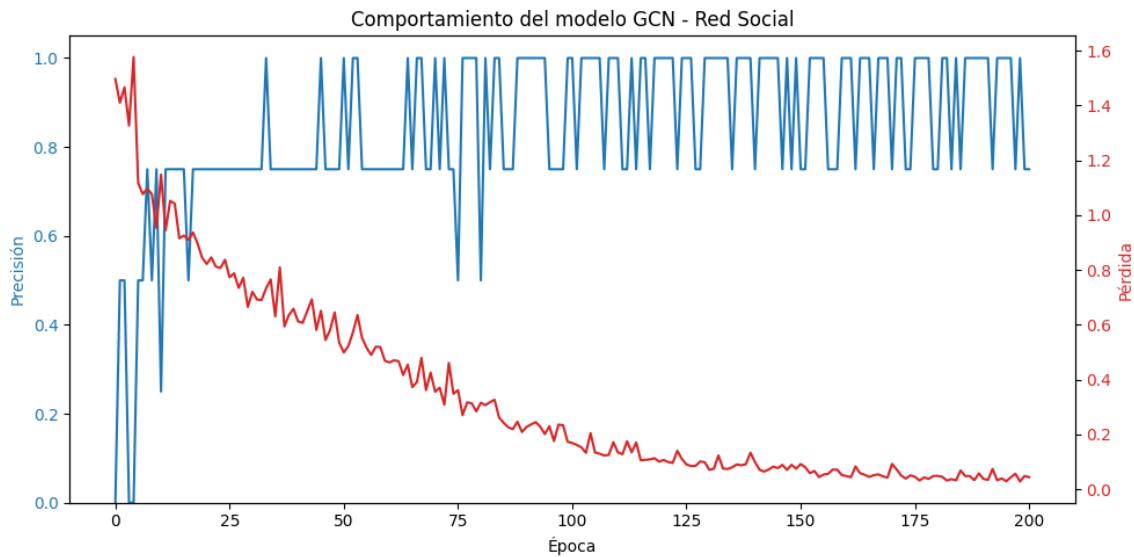
- 1) Básico



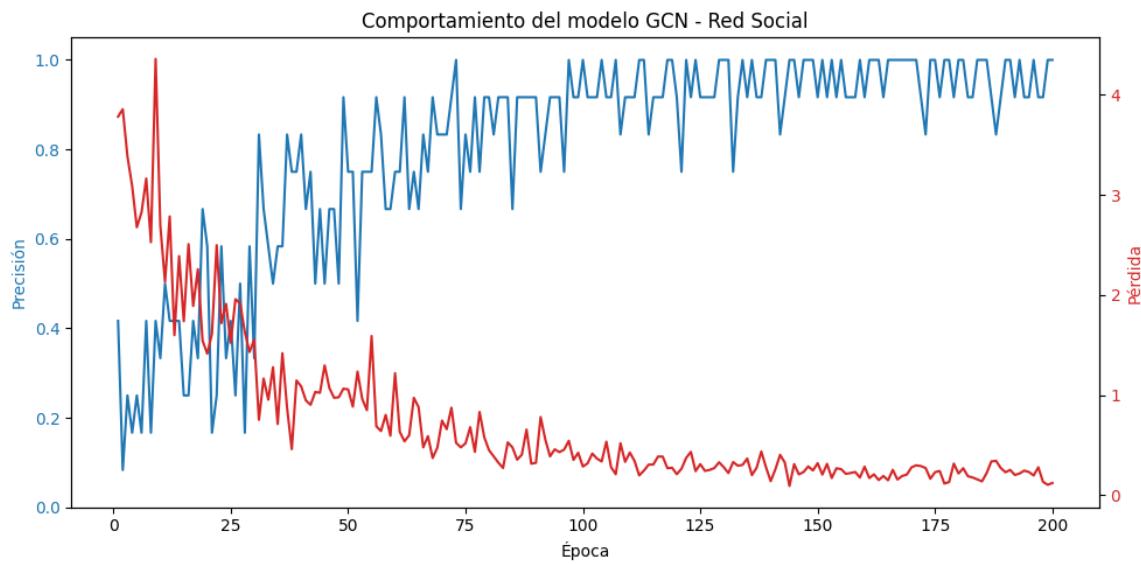
2) Mejorado



3) Mejorado v2



4) Avanzado



En las versiones más completas se observa una mejora significativa en la estabilidad y precisión, demostrando que el agregado de nodos, atributos y ajustes de arquitectura tuvo un impacto positivo en el rendimiento.

Como se afirma en Wu et al. (2020), el aumento de información contextual y la incorporación de capas adicionales tienden a mejorar la expresividad del modelo, siempre que se controle el sobreajuste y el sobreacoplamiento de los datos.

7. Conclusiones

Durante el desarrollo de este proyecto se logró comprender de manera básica cómo funcionan las redes neuronales aplicadas a grafos (GNN). Aunque partimos de conceptos sencillos, se obtuvo una visión más clara del enorme potencial que estas redes tienen en la actualidad. De hecho, tecnologías como ChatGPT y otras herramientas avanzadas de inteligencia artificial nacen de principios similares, y han evolucionado a soluciones de gran escala que impactan diversas industrias.

Además, se adquirieron conocimientos importantes sobre la diferencia entre las GNN y otras arquitecturas como las CNN. Las GNN destacan especialmente por su capacidad de aprender de estructuras no euclidianas, como las redes sociales, lo cual amplía enormemente su rango de aplicación.

A nivel práctico, fue enriquecedor observar cómo pequeños cambios en la arquitectura o en la cantidad de datos pueden mejorar considerablemente el rendimiento del modelo. La implementación progresiva de mejoras permitió analizar el comportamiento de la red, entender los conceptos detrás de capas convolucionales sobre grafos, y aplicar metodologías actuales para el entrenamiento, evaluación y visualización de resultados.

En resumen, esta experiencia no solo fortaleció la comprensión de GCN y su funcionamiento, sino que también incentivó el interés por explorar con mayor profundidad el campo de la inteligencia artificial aplicada a grafos.

8. Referencias

Kipf, T. N., & Welling, M. (2017). *Semi-supervised classification with graph convolutional networks*. International Conference on Learning Representations (ICLR). <https://arxiv.org/abs/1609.02907>

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). *A comprehensive survey on graph neural networks*. IEEE Transactions on Neural Networks and Learning Systems. <https://arxiv.org/abs/1901.00596>

PyTorch Geometric. (s. f.). *PyTorch Geometric Documentation*. <https://pytorch-geometric.readthedocs.io>

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). *NetworkX: Software for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks*. Proceedings of the 7th Python in Science Conference (SciPy2008). <https://networkx.org/>

OpenAI. (2024). *ChatGPT (versión GPT-4o) [Modelo de lenguaje]*.
<https://chat.openai.com>

Nota: Se utilizó ChatGPT de forma parcial como apoyo para la generación de fragmentos de código y redacción técnica.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1), 4–24.
<https://doi.org/10.1109/TNNLS.2020.2978386>

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

9. Anexos

A continuación se proporciona el enlace al repositorio oficial del proyecto en GitHub, el cual contiene todos los archivos fuente, scripts de entrenamiento, gráficas, dataset utilizado y documentación técnica necesaria para su revisión:

🔗 **Repositorio del proyecto:**
<https://github.com/IsaacEspAI1/proyecto-red-gcn-grupo-7>

El repositorio está organizado en carpetas que incluyen:

- demos/: Scripts de entrenamiento de las distintas versiones de la red GCN.
- csv/: Resultados numéricos de entrenamiento (precisión y pérdida por época).
- graficos/: Imágenes PNG de los gráficos generados.
- dataset_red_social_ampliada.csv: Dataset utilizado en todos los modelos.
- README.md: Descripción general del proyecto y cómo ejecutarlo.

Este enlace permite a cualquier persona (incluyendo el profesor revisor) acceder libremente a todo el contenido del proyecto para su análisis o ejecución, sin necesidad de credenciales adicionales.