UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Generación de grafos de cristales químicos usando redes neuronales recurrentes

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

Maestro en Ciencia e Ingeniería de la Computación

PRESENTA:

Pablo Camacho González

TUTOR:

Dr. Gibrán Fuentes Pineda
IIMAS, UNAM

CIUDAD UNIVERSITARIA, CDMX. MARZO 2022

Agradecimientos.

A mi madre que sin duda es una de las persona que mas me ha impulsado a seguir estudiando y se preocupa por mi futuro.
A mi padre, por formarme como una persona critica y responsable, por siempre impulsarme en hacer las cosas que me gustan.
A mis dos hermanos, por soportarme y compartir su vida conmigo.
Al Dr. Gibrán Fuentes Pineda por la guía, por su tiempo, los consejos y las criticas para desarrollar este trabajo, gracias.
Al Dr. Olmo Zavala por ser quien me impulso a estudiar la maestría, sin su constante impulso esto no existiría.
A todos los amigos y compañeros con los cuales en algún momento de mi vida compartí momentos que son parte de quien soy.
A la Universidad Nacional Autónoma de México, al Posgrado en Ciencia e Ingeniería de la Computación por ser mi centro de formación académica.
Al Consejo Nacional de Ciencia y Tecnología (Conacyt) por la beca otorgada para desarrollar la investigación que dio como resultado esta tesis.

¡BANG!

Índice general

Índice de figuras

Lista de símbolos

a	Un escalar
$\boldsymbol{A}$	Una matriz
$p_{modelo}(X)$	Distribución sobre un conjunto X
G	Conjunto de grafos
$\mathscr{G}$	Un grafo
V	Conjunto de nodos o vértices
$v = \{x, y\}$	Un nodo con posición en x, y
E	Conjunto de aristas
$e = \{u, v\}$	Una arista que une al nodo u con el nodo v
γ	Un camino sobre un grafo G
$A(\mathscr{G})$	Matriz de adyacencia del grafo G
$f(x)$	Función sobre x
$f : \mathbb{A} \to \mathbb{B}$	Función f con dominio $\mathbb{A}$ y rango $\mathbb{B}$
$\{1, 2, \ldots, n\}$	Conjunto con todos los enteros entre 1 y n
$A_{i,j}$	Valor dentro de una matriz con posición i, j
$O(n)$	Notación O grande, para medir la complejidad de un algoritmo
$\|V\|$	Número de elementos que contiene el conjunto V
x_i	Valor de entrada i-ésimo a la red neuronal
Θ	Matriz de parámetros de una red neuronal
$\mathbf{W}_{i,j}$	Valor i-ésimo dentro de la matriz de parámetros de la red neuronal

x_i^k	Valor de entrada de la neurona i-ésima en la k-ésima capa de una red neuronal
a_i^k	Salida de la neurona i-ésima en la k-ésima capa de una red neuronal
δ_i	Error de salida de la neurona i-ésima
$\Delta_{ij}^{(l)}$	Actualización de parámetros de la neurona i, j en la capa l
$D_{i,j}^{(l)}$	Acumulador de sumas de la neurona i, j en la capa l
$max(x, y)$	Valor máximo entre los valores x y y
$J(\Theta)$	Función de costos sobre Θ
$\frac{\partial y}{\partial x}$	Derivada parcial de y con respecto a x
$\log x$	Logaritmo natural de x
$\tanh x$	Tangente hiperbólica de x
$\sinh x$	Seno hiperbólico de x
$\cosh x$	Coseno hiperbólico de x
$d_e(u, v)$	Distancia euclidiana entre nodo u y el nodo v

Glosario de términos

GAN	Redes generativas antagónicas
RNN	Redes neuronales recurrentes
GraphGen	Modelo de redes neuronales recurrentes para la generación de grafos etiquetados
LSTM	Memorias de corto y largo plazo
GRU	Unidad recurrente regulada
CIF	Archivo de cristalografía
DFS	Recorrido en profundidad de un grafo
ReLU	Unidad lineal rectificada
GD	Descenso por gradiente
MMD	Discrepancia media máxima
NumA	Número de aristas generadas
NetworkX	Biblioteca de grafos en Python

Capítulo 1

Introducción

El uso de grafos en nuestra vida diaria parece ser casi nulo, sin embargo están ahí de forma intangible. Por ejemplo: las relaciones que tenemos con nuestra familia, la comunicación que tenemos con la gente que nos rodea, la ejecución de una tarea y los múltiples caminos que podemos tomar en un ruta de viaje pueden ser representados por un grafo. Los grafos también son capaces de representar conocimiento e incluso modelar compuestos químicos. En general, estas estructuras de datos son útiles para representar información relacional y estructural en múltiples dominios.

Un grafo está compuesto por dos componentes, nodos y aristas. Una persona, una ciudad o un elemento químico se pueden pueden representar con un nodo, mientras que la relación parental entre varias personas, las carreteras entre varias ciudades y los enlaces químicos entre elementos son representadas con aristas entre nodos. En un grafo un nodo es una entidad de información y una arista es la forma en que se relacionan las entidades entre ellas. Por lo tanto, un grafo de un proceso o de una situación se construye tomando las entidades como nodos y las relaciones como aristas. Sin embargo, los nodos y las aristas de un grafo construido de esta forma no pueden ser identificados. Para logra esto último existen los grafos etiquetados, los cuales mantienen el nombre de las entidades en los nodos y las aristas tienen el nombre de la relación que mantienen

los nodos; en el ejemplo de las relaciones parentales, los nodos tendrían el nombre de las personas y las aristas tendrían el parentesco que mantienen dos personas; mientras que en el ejemplo de las ciudades y las carreteras, los nodos tendrían el nombre de la ciudad y las aristas el nombre de las carreteras. En lugar de nombres las aristas pueden tener asociadas alguna característica; en el caso de las carreteras, la etiqueta de una arista puede contener el número de kilómetros de la carretera entre dos ciudades.

La cantidad de aristas y cómo están relacionadas con los nodos dan pie a diferentes tipos de grafos, entre ellos están los conexos y los disconexos. Los grafos conexos cumplen con la propiedad de que desde cualquier nodo se puede llegar a cualquier otro nodo siguiendo sus aristas, mientras que en los disconexos esto no se cumple para cualquier par de nodos. Si tomamos un grafo de las ciudades y las carreteras de México tendríamos un grafo conexo, ya que desde cualquier ciudad podemos llegar a cualquier otra ciudad mediante algunas de sus carreteras En contraste, si tenemos en un mismo grafo las de México y España, tendríamos un grafo disconexo, ya que no existe ninguna carretera entre cualquier ciudad de México y cualquier ciudad de España.

Tener una representación de los problemas en forma de grafo no solo nos sirve para ver de manera visual el problema o situación que queremos resolver, si no que también se pueden aplicar algoritmos que ayuden a obtener información dentro del grafo, por ejemplo la ruta más corta entre dos nodos, la ruta con menor peso acumulado en las aristas, encontrar un nodo específico, ordenar nodos, encontrar vecinos más cercanos y flujos máximos y mínimos.

Hasta ahora construir un grafo es tomar las entidades con sus relaciones y convertirlas en nodos y aristas respectivamente. Sin embargo, no siempre tenemos los datos para construir los grafos o se quieren obtener de manera sintética. Como ejemplo, generar el grafo de una interacción en alguna red social a partir de una palabra (nodo), el modelo 3d de algún objeto de la vida cotidiana o incluso generar grafos que tengan la capacidad de describir un nuevo componente químico. Pero generar este tipo de grafos

puede conllevar problemas, algunos de ellos son lidiar con la generación de un grafo de gran tamaño y que sean conexos o disconexos, ya que se pierda la interacción entre nodo-arista y por lo tanto en la forma del grafo. Otro problemas es la generación de las etiquetas, ya que está fuertemente ligada a la interacción entre nodo y arista. Uno de los problemas que es ajeno a los grafos es el tiempo de generación, dicho proceso puede llegar a ser de $O(n^4)$, donde n es el número de nodos en el grafos, lo que provoca que generar grafos de gran tamaño y complejidad sea un gran problema. Estos problemas son los que se tienen que resolver o son limitaciones de modelos que buscan generar grafos de uso específico o los que intentan generar cualquier tipo de grafo.

1.1. Planteamiento del problema

Dentro de la generación de grafos existen diferentes enfoques. El enfoque matemático tiene propiedades bien definidas dentro de la teoría de grafos; el modelo principal dentro de este enfoque es el de Erdos-Renyi [45], el cual se enfoca en generar gráficos aleatorios[1]. La generación de estos grafos se lleva a cabo eligiendo una distribución de probabilidad o definiendo un proceso aleatorio. Dada la forma en que se generan los grafos se tiene una limitada capacidad para modelar dependencias complejas entre los nodos y las aristas. Además, se tiene un enfoque muy restrictivo en términos de las estructuras que se pueden generar con dichos modelos y no tienen la capacidad de generar grafos etiquetados.

El segundo enfoque en la generación de grafos se trata de los modelos que están basados en redes neuronales. En este enfoque se encuentran los autorregresivos y los no autorregresivos, ambos dentro del aprendizaje no supervisado.

El objetivo de los modelos autorregresivos, como son las redes neuronales de grafos, es aprender a generar nuevas muestras similares al conjunto de datos de entrena-

[1] Grafo aleatorio es el término general para referirse a distribuciones de probabilidades sobre grafos

miento. Estos modelos surgen de la unión de un modelo lineal autorregresivo y una red completamente conectada. Por otro lado, en los no autorregresivos las salidas no dependen linealmente de los valores anteriores, esto lo logran aplicando algún algoritmo que rompa con la linealidad. Como ejemplo tenemos el trabajo de Grover [31], el cual es un codificador de red neuronal de grafos, mientras que para romper la linealidad usa un decodificador iterativo para la generación de grafos. Dichos modelos son poco escalables en cuanto al tamaño de un grafo, lo que provoca que para algunas aplicaciones no se pueden obtener grafos que cumplan con características buscadas. Sumado a esto tenemos que el tiempo de cómputo para la generación de un solo grafo tiene una cota de $O(n^2)$, además que al generar los componentes de forma independiente la calidad de los grafos que se generan se ve comprometida.

Ambos enfoques tienen la capacidad de generar grafos, que son capaces de resolver ciertos problemas. Sin embargo, cuando se trata de generar grafos que tiene mayor complejidad y tamaño los modelos se ven sobrepasados. Como son las estructuras moleculares o redes sociales, los modelos se ven limitados por la forma en que construyen los grafos o por la misma naturaleza del método.

Formalmente se busca un modelo generativo que aprenda una distribución $p_{modelo}(G)$ sobre un conjunto de grafos observados $G = \{\mathscr{G}_1, \mathscr{G}_2 \ldots, \mathscr{G}_n\}$. Donde los grafos pueden tener un número diferente de nodos y aristas, además de un conjunto variado de etiquetas en nodos y aristas. Por lo tanto, queremos un modelo que sea capaz de generar grafos similares de tamaño variado sobre la misma distribución $p_{modelo}(G)$. En conclusión, se busca aprender una distribución sobre un conjunto de grafos, sin importar sus características, este hecho es lo que hace interesante la generación de grafos.

Hasta el momento, los modelos de generación de grafos, tienen diferentes formas de representar los grafos para poder procesarlos. El uso de la matriz de adyacencias o en forma de cadena, son algunas de las formas que se suelen usar para trabajar con los grafos. Sin embargo, existen grafos que pierden características o que no se pueden re-

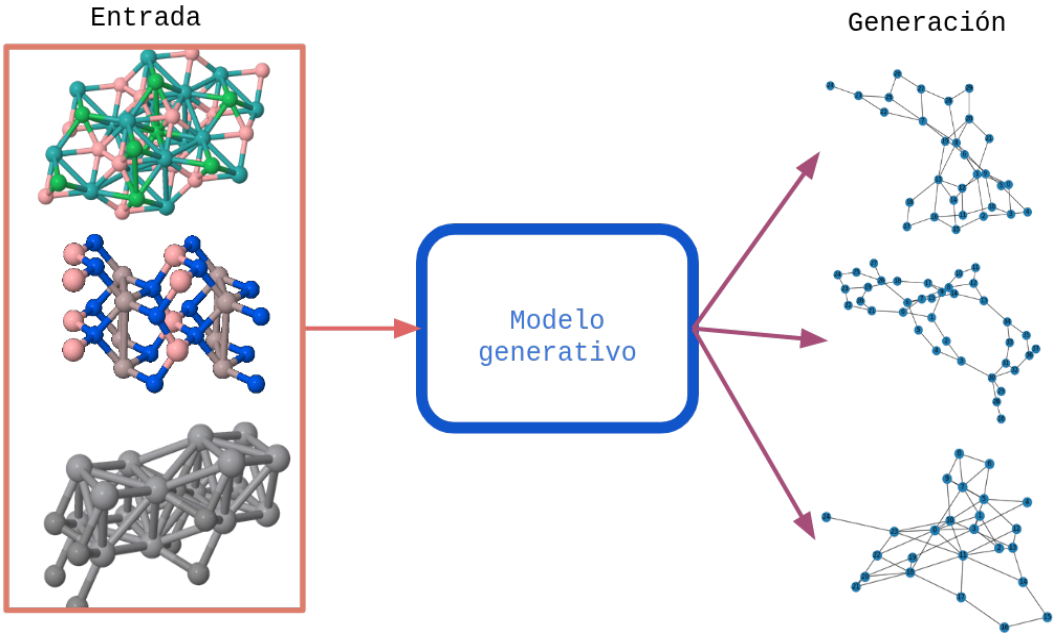

Figura 1.1: Ejemplos de cómo funciona un modelo generativo de grafos, la salida son grafos generados por el modelo generativo.

presentar en su totalidad, como es el caso de los grafos etiquetados o los grafos de tipo disconexo. Se sabe que los modelos generativos son capaces de generar nuevos objetos con las mismas características con los que se entreno. En el caso de la generación de grafos, si la representación usada por un modelo no permite los grafos disconexos o grafos etiquetados, tendremos que la generación estará limitada por dicha representación.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un método para generar grafos etiquetados de tipo conexo y disconexo, usando un modelo de aprendizaje no supervisado que hace uso de redes neuronales

recurrentes. El método desarrollado se verificará con grafos de cristales de perovskitas. Se usan estos grafos por que tienen una gran variedad de formas, tamaños y nodos con etiquetas lo cual permite probar con un modelo que genere grafos etiquetados.

1.2.2. Objetivos específicos

- Revisar y analizar el estado del arte en la tarea de generación de grafos etiquetados.
- Recopilar una base de datos de grafos etiquetados, en este caso grafos de cristales de perovskitas.
- Preprocesar los cristales de perovskitas, con el fin de obtener los grafos que codifiquen la información relevante.
- Hacer un análisis exploratorio de los grafos.
- Diseñar una nueva representación para los grafos que no se pueden generar por las restricciones impuestas por el modelo que se usa.
- Integrar la nueva representación dentro de la red neuronal recurrente para la generación de grafos de cristales de perovskitas.
- Evaluar la red neuronal recurrente con la representación propuesta en la generación de perovskitas.

1.3. Retos

Trabajar con grafos representa diversos retos, los cuales se detallan a continuación:

- **Obtener un conjunto de grafos apropiados.** Para poder entrenar los modelos generativos es necesario contar con una gama variada de grafos, es decir, que

contenga grafos con un número variado de nodos así como de etiquetas. Tener un conjunto que contenga gran diversidad de grafos nos va a permitir evaluar la capacidad de la representación propuesta para generación de grafos con diversidad de tamaño, forma y de etiquetas.

- **Conversión de los grafos.** A pesar de que la representación de un grafo es hasta cierto punto sencilla, hay conjuntos de datos que no proveen una representación de grafo como tal, por lo que es necesario obtenerla a partir de la representación original tratando de no perder información relevante del fenómeno a modelar.

- **Representación de un grafo**. Plantear una nueva representación para los grafos que no cumplan ciertas características, con el fin de que se puedan generar. Todo esto, buscando que las nueva representación sea lo más fiel posible a la original, además de medir la calidad de los grafos generados bajo la representación planteada

1.4. Aportes de la investigación

Los aportes de este proyecto de investigación son los siguientes:

- **Representación para cualquier tipo de grafo.** En este trabajo se propone una nueva representación para la generación de grafos conexos y disconexos, que permita que un modelo generativo tenga la capacidad de generar cualquiera de los dos tipos de grafos

- **Validación en datos de uso específico** Se aplica la modificación propuesta sobre un conjunto de grafos de cristales de perovskitas, con lo cual el modelo es validado con grafos que pertenecen a un problema específico.

1.5. Metodología

El método que se plantea en esta tesis conlleva varios pasos que a continuación se describen a grandes rasgos. Como primer paso se obtienen los datos con los que se probará la representación. Se usará un conjunto de grafos de perovskitas, los cuales se obtienen a partir de los cristales de perovskitas representados en archivos *Crystallographic Information File* (CIF) del *Material Project* [36] y *Crys-tallography Open Database*. Posteriormente se dividen en grafos conexos y disconexos y se genera una nueva representación de los grafos disconexos agregando aristas con etiquetas especiales. Finalmente se entrena y evalúa el modelo de generación bajo diferentes condiciones.

1.6. Organización de la tesis

El resto de la tesis se encuentra organizada de la siguiente manera:

- Capítulo 2. Se examinan los métodos del estado del arte para la generación de grafos.
- Capítulo 3 Se describen los fundamentos teóricos que sustentan el proyecto, específicamente la teoría de grafos, el recorrido DFS y las redes neuronales recurrentes.
- Capítulo 4. Se presenta la representación propuesta para grafos conexos y disconexos.
- Capítulo 5. Se detalla el proceso de recopilación y preprocesamiento de los datos, se especifican los experimentos realizados y se discuten los resultados obtenidos.
- Capítulo 6. Se comentan las conclusiones y el trabajo futuro.

Capítulo 2

Estado del arte

En este capítulo se describen los modelos matemático y de aprendizaje automático más relevantes, con la capacidad de generar grafos.

2.1. Modelos matemáticos

El modelo de Erdős–Rényi [45], se trata de uno de los primeros modelos para la generación de grafos aleatorios. Dicho modelo genera un nuevo nodo que será enlazado con igual probabilidad al resto del grafo, por lo que posee independencia estadística con el resto de los nodos ya generados.

Dentro de los modelos matemáticos están los Modelos de gráficos aleatorios exponenciales (ERGMs), que son una familia de modelos estadísticos usados en redes de conocimientos, redes organizativas, redes sociales, etc. Estos modelos representan una distribución de probabilidad para cada posible grafo [24].

A pesar de que son modelos que se usan en la práctica, tienen varias limitaciones. Dado que generan nodos de manera independiente, pierden capacidad para modelar dependencias complejas y tienen un enfoque muy restrictivo en términos de las estructuras que pueden llegar a generar, además de no tener la capacidad de generar grafos

etiquetados.

2.2. Modelos de aprendizaje automático

Desde hace tiempo el uso de modelos de aprendizaje automático es de uso cada vez mas común. En el caso de los grafos son usados los modelos de generación no supervisados. Dentro de estos modelos están los que atacan problemas específicos, por ejemplo la generación de cristales o componentes químicos y los de uso más general donde se generan grafos con ciertas características, por ejemplo grafos etiquetados, conexos, etc.

2.2.1. Modelos de uso específico

Dentro de la generación de grafos, existen modelos que han sido desarrollados para atacar un problema en específico, como la generación de moléculas (por ejemplo, los modelos MolGAN [12] y NeVAE[40]) o cristales.

Dentro de la generación de moléculas están los que usan las redes generativas antagónicas como es el modelo MolGAN [12], el cual es usado en la generación de pequeñas moléculas. Este modelo hace uso de SMILE, la cual es una cadena en código ASCII que permite representar un componente químico sin ambigüedad. El modelo convierte el SMILE en un grafo y obtiene una matriz de adyacencia y una matriz de anotaciones, la segunda matriz contiene la información química. Como resultado, el modelo genera un grafo etiquetado, donde cada nodo tiene su elemento químico.

Otro tipo de redes usadas en la generación de moléculas son los autoenconder variacionales como es el modelo NeVAE [40] y el trabajo desarrollado por Wengong Jin[21]. Para el primer modelo, la representación de cada molécula es dado por un ordenamiento BFS usando una tripleta para cada nodo. Cada tripleta contiene el nombre de elemento del nodo, su posición x, y, z y el tipo de enlace. Mientras que para el se-

gundo trabajo, se hace uso de optimización con una función bayesiana, el grafo de la molécula es procesada para obtener un árbol del grafo original, además de no sólo usar un solo autoencoder, si no también un árbol decodificador para luego pasar por redes del tipo GRU.

El último tipo de redes usado en la generación de moléculas son las redes neuronales convolucionales. El modelo *Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation* [34] hace uso de estas redes junto con un gradiente de política, que ayuda a castigar o recompensar los grafos que se generan. El modelo hace uso de las SMILE al igual que el modelo MolGAN, sin embargo éste sólo convierte la SMILE en un grafo etiquetado.

Fuera de la generación de moléculas, las redes neuronales en grafos son usadas en el descubrimiento de medicamentos, como es el caso del trabajo desarrollado por Pietro Bongini[4], el cual usa las SMILE para obtener un grafo y ser usado por la redes en grafos para generar nuevos medicamentos.

Aunque los modelos descritos anteriormente utilizan aprendizaje automático para generara grafos, están enfocados en representaciones muy específicas al problema. Además, son poco escalables dada su naturaleza y el tiempo de generación es aproximadamente $O(n^2)$.

2.2.2. Modelos de uso general

Dentro de los modelos de uso general, existen diferentes enfoques, a continuación se mencionan los principales modelos, así como la representación de grafo que usa cada uno.

Las arquitecturas más usadas dentro de las de uso general son las redes neuronales en grafos, aquí tenemos los modelos GraphRNN [48], Graphite[15] y GRAN.

GraphRNN [48] usa como representación del grafo su matriz de adyacencia y aplica una función de permutación para ordenar los nodos. El modelo genera el primer nodo

y en cada paso de generación obtiene una arista y un nodo que sale del primer nodo, cuando genera todo los nodos y arista que salen del primer nodo, prosigue a generar los nodos y aristas del siguiente nodo, por lo tanto en el peor de los casos generar un grafo es de complejidad $O(n^2)$. El modelo fue probado en generación de proteínas, grafos de malla y grafos generados por el modelo de Erdős–Rényi. El modelo no tiene la capacidad de generar grafos etiquetados.

Graphite[15] se trata de un modelo semi-supervisado. La representación de los grafos que usa es la matriz de adyacencias, sin embargo, se hace una codificación usando el algoritmo de Weisfeiler-Lehman. Dicho algoritmo es una heurística que busca revisar si dos grafos son isomorfos. El modelo aplica una función de permutación sobre la matriz de adyacencia y sobre éste aplica el algoritmo que codifica si ambos grafos son isomorfos. El modelo fue probado en un grafo de citas bibliográficas obtenidas del dataset Cora y Citeseer, donde los artículos son los nodos y las citas son las aristas.

Uno de los modelos de gran relevancia y capacidad es GRAN [29]. El modelo usa la matriz de adyacencia como representación de los grafos, además de aplicar en la salida de la red una distribución de Bernoulli, la distribución de Bernoulli es la encargada de decidir si se genera o no una arista, lo que permite generar uniones entre aristas y nodos con mayor complejidad. En cada paso de generación se obtiene un nodo y todas sus posibles aristas, entonces desde el principio se tiene el tamaño del grafo que se va a generar. De tal forma que generar un grafo es de tiempo $O(n)$, donde n es el número de nodos que tendrá el grafo. Dicha cota, permite que el modelo sea capaz de generar grafos de gran tamaño, pero no tiene la capacidad de generar grafos etiquetados.

Ya fuera de las redes neuronales en grafos, están los que son autorregresivos que hacen uso de las capas LSTM y GRU. Como ejemplo de este tipo tenemos a DeepGMG [28]. Lo interesante de este modelo es la forma de representación de los grafos. Cada grafo lo describe como una tripleta que contiene; la probabilidad del nodo, la de las aristas y la probabilidad de conectarse con otros nodos. Entonces, en cada paso de

generación se produce un solo nodo y se conecta con el demás grafo existente uno a la vez. El modelo es poco escalable en cuanto al tamaño de los nodos y el tiempo de generación es de $O(n^2m)$, donde m es el numero de posibles arista que se pueden generar.

También tenemos modelos que hacen uso de codificadores variacionales, como es el caso de GraphVAE [42], que toma como entrada de grafo una tripleta que está conformada por: la matriz de adyacencia, un vector de características de las aristas y una matriz de características de los nodos. El modelo solo es capaz de generar grafo de tamaño pequeño, ya que el número de parámetros que se alojan en la memoria ocupan espacio $O(n^2)$, mientras que su complejidad computacional es de $O(n^4)$.

Por último se tiene al modelo GraphGen [14], que no hace uso de redes neuronales gráficas o autoenconder variacionales, si no un modelo recurrente que hace uso de capas LSTM. El modelo GraphGen tiene la capacidad de generar nodos y aristas etiquetadas, su tiempo de generación es de $O(m)$, donde m es el numero de posibles arista que se pueden generar. Siendo el más bajo de los modelos antes descritos.

El modelo aplica una función de permutación sobre la matriz de adyacencia para obtener un grafo isomorfo, después de aplicar la función se aplica un ordenamiento DFS, este ordenamiento dará la representación que usará la red. La representación se trata de una 5-tupla, de tal manera que el modelo ve a un grafo como una cadena de texto, como si se tratará de un problema de procesamiento de lenguaje natural. Uno de los principales problemas del modelo, es que no tiene la capacidad de generar grafos disconexos o que al aplicar la función de permutación no se obtenga un grafo isomorfo. Tomaremos la arquitectura de este modelo y el trabajo que se presenta aquí sera modificar la representación que usa el modelo para que acepte grafos conexos y disconexos, por lo tanto, en el capítulo de fundamentos se describirá el modelo más a detalle junto con sus limitaciones.

Capítulo 3

Fundamentos

Como parte de la tesis se busca desarrollar una representación para grafos que no cumplan características especificas, como son los grafos etiquetados y disconexos, además de usar un modelo de generación de grafos, sin embargo antes de presentar el modelo es importante mencionar los fundamentos en los que se basa. A continuación se mencionan los fundamentos usados de teoría de grafos, redes neuronales y redes recurrentes.

3.1. Grafos

Definición 3.1. *Un grafo se define como un par $\mathscr{G} = (V, E)$, donde V es un conjunto finito no vacío de vértices o nodos y E es un conjunto de aristas.*

Cuando $\mathscr{G}$ es un grafo *no dirigido* tenemos que las aristas son un conjunto de pares no ordenados de nodos, denotadas por $e = \{u, v\}$ que indica que la arista e une a los nodos u y v.

Si tenemos un nodo que se une a sí mismo con una arista, esta recibe el nombre de **lazo**; en la Figura 3.1 la arista e_4 es una arista lazo.

Definición 3.2. *Sea $\mathscr{G} = (V, E)$ un grafo. Se denomina subgrafo de $\mathscr{G}$ a un grafo cuyo*

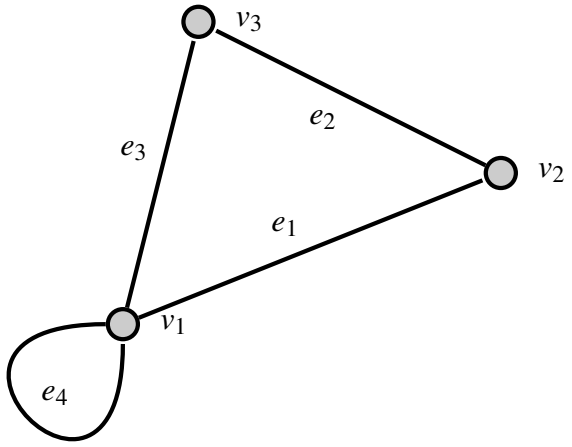

Figura 3.1: Ejemplo de un grafo con 3 nodos y 4 arista, donde G tiene como conjunto de nodos $V = \{v_1, v_2, v_3\}$ y el conjunto de aristas $E = \{e_1, e_2, e_3, e_4\}$.

conjunto de nodos es un subconjunto de los nodos de $\mathscr{G}$ y cuyo conjunto de aristas está formado por las aristas de $\mathscr{G}$ del subconjunto de nodos.

Definición 3.3. *Dados dos grafos $\mathscr{G}_1 = (V_1, E_1)$ y $\mathscr{G}_2 = (V_2, E_2)$, diremos que hay un homomorfismo entre ellos si existe un aplicación f*, definida como:

$$f = V_1 \to V_2,$$

tal que si $\{u, v\} \in E_1$, entonces $\{f(u), f(v)\} \in E_2$

Definición 3.4. *Diremos que $\mathscr{G}_1$ y $\mathscr{G}_2$ son isomorfos si f es una aplicación biyectiva que preserva las adyacencias, esto es, $\{u, v\} \in E_1 \Leftrightarrow \{f(u), f(v)\} \in E_2$. La biyección envía un nodo a otro nodo con el mismo grado.*

De la definición se deduce que dos grafos isomorfos necesariamente tienen que tener el mismo número de nodos, de aristas y de nodos con el mismo grado.

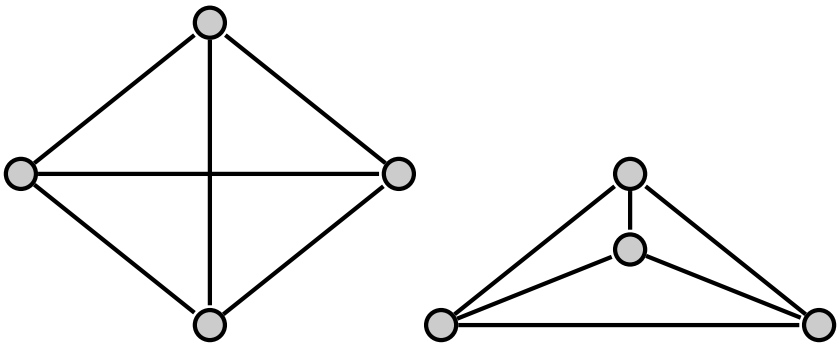

Figura 3.2: Ejemplo de dos grafos que son isomorfos.

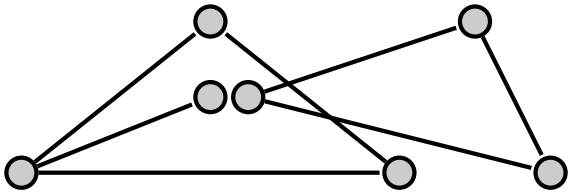

Figura 3.3: Grafo disconexo con dos componentes conexas.

Definición 3.5. *Un camino en $\mathscr{G}$ es una sucesión finita de nodos $\gamma = \{v_1, v_2, \cdots, v_n\}$ tales que dos elementos consecutivos de la misma son siempre adyacentes.*

Definición 3.6. *Un grafo se dice conexo si para cualquiera dos de sus nodos pueden unirse mediante un camino. Si un grafo no es conexo, se le llama disconexo. Se le llama componente conexa de un grafo a todo subgrafo conexo del mismo que tenga el mayor número posible de nodos.*

Definición 3.7. *Sea $\mathscr{G} = (V, E)$ un grafo con n nodos. $\mathscr{G}$ es un árbol si para cualquiera dos nodos hay un camino que los une y $\mathscr{G}$ tiene exactamente $n-1$ aristas. Por lo tanto, un árbol es un grafo conexo sin ciclos.*

Definición 3.8. *Un bosque se puede definir como una unión disjunta de árboles, es decir, es un grafo disconexo cuyos componentes son árboles.*

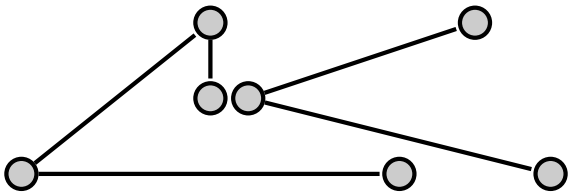

Figura 3.4: Ejemplo de un bosque que tiene dos arboles.

Definición 3.9. *Sea* $\mathscr{G} = (V,E)$ un grafo, $V = \{x_1, \cdots, x_n\}$. Se denomina matriz de adyacencia del grafo $\mathscr{G}$ a la matriz de orden, donde $A(\mathscr{G}) = (a_{ij})_{i,j=1,\cdots,n}$,

$$a_{ij} := \begin{cases} 1 & si\ \{x_i, x_j\} \in E \\ 0 & en\ otro\ caso \end{cases}.$$

La matriz de adyacencia del grafo de la Figura 3.1 queda de la siguiente forma:

$$G = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

3.2. Recorrido en profundidad de un grafo (DFS)

Recorrer un grafo consiste en visitar cada un de los nodos a través de las aristas del grafo. De manera intuitiva lo que se hace es marcar un nodo cuando ha sido visitado y después visitar a sus nodos adyacentes, por lo tanto tenemos que inicialmente ningún nodo ha sido visitado [45]. DFS recorre un grafo en profundidad mientras que BFS lo hace en anchura. En la Figura 3.5 se puede ver la diferencia entre un recorrido DFS y BFS, la diferencia es más visible cuando se trata de un grafo de tipo árbol.

Para ejecutar de manera básica DFS lo primero es definir un algoritmo $DFS(\mathscr{G}, v)$, esto es, el recorrido en profundidad de $\mathscr{G}$ con origen en el nodo v. Dicho nodo se define por defecto o se puede elegir de manera aleatoria.

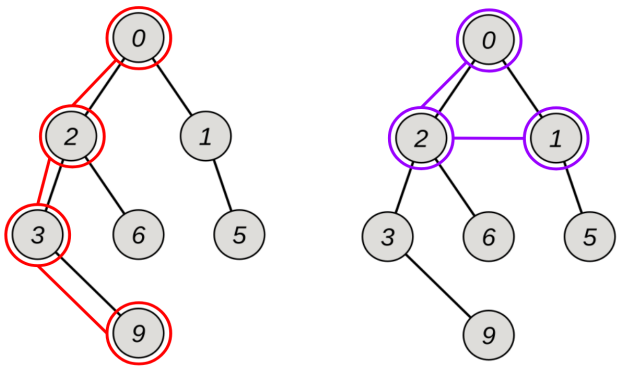

Figura 3.5: Ejemplo de un recorrido DFS y BFS en un árbol, el recorrido rojo es el DFS.

El algoritmo $DFS(\mathscr{G}, v)$ está definido de manera recursiva como sigue:

Algoritmo 1: Algoritmo DFS

Entrada: Un grafo y el nodo inicial

1. Se marca el nodo v.
2. Si los nodos adyacentes a v ya han sido visitados, se termina la ejecución del algoritmo, en caso contrario, se elige un nodo w adyacente a v que no esté marcado como visitado.
3. Se ejecuta $DFS(\mathscr{G}, w)$.

Salida: Lista con los nodos visitados

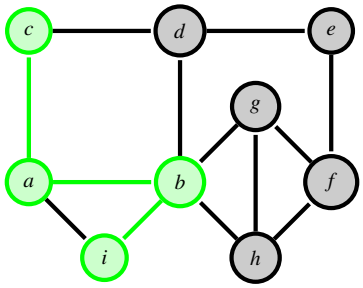

Figura 3.6: Ejemplo de una ejecución de DFS que inicia en el nodo c .

En el grafo de la Figura 3.6 se muestra una ejecución del algoritmo DFS, donde tenemos el recorrido $DFS(\mathscr{G}) = \{c, a, b, i\}$. Sin embargo, el recorrido no contiene todos los nodos del grafo, debido a que al llegar al nodo i se cumple la condición de que los nodos adyacentes a i ya han sido visitados y por lo tanto, se termina la ejecución. Para recorrer todo el grafo se hace lo que se conoce como *backtracking*, es decir, que se regresa al nodo inmediato anterior y se visita los nodos adyacentes que aún no han sido visitados. De esta forma, el algoritmo DFS ahora queda definido como sigue:

Algoritmo 2: Algoritmo DFS con *backtraking*

Entrada: Un grafo y el nodo inicial

- Mientras no se visiten todo los nodos del grafo:

  1. Se marca el nodo v.
  2. Si los nodos adyacentes a v ya han sido visitados, se regresa al nodo anterior inmediato y se ejecuta de nuevo el paso 2, en caso contrario, se elige un nodo w adyacente a v que no esté marcado como visitado.
  3. Se ejecuta $DFS(\mathscr{G}, w)$.

Salida: Lista con los nodos visitados

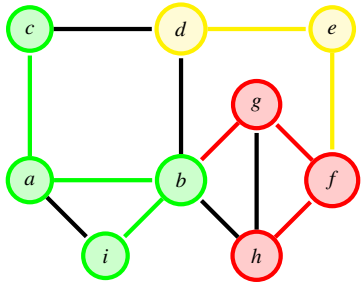

Figura 3.7: Ejemplo de una ejecución de DFS que inicia en el nodo c, se obtiene un árbol de recorrido

La ejecución de DFS nos proporciona un árbol de ejecución; a las aristas del grafo $\mathscr{G}$ que no pertenecen al árbol se les conoce como aristas de vuelta atrás (*backward*

edges) [3]. Este tipo de aristas también sirven para ordenar los nodos de un grafo, ya que permiten regresar dentro del recorrido cuando ya no se tienen nodos que se puedan visitar.

En el grafo de la Figura 3.7 se muestra el recorrido DFS con el *bactracking*, vemos que se obtiene un árbol y las aristas de vuelta atrás son las que están de color negro.

El recorrido que se obtiene con el algoritmo DFS no es único, dado que se pueden elegir distintos nodos tanto iniciales como adyacentes. Por lo tanto, se tiene que hay múltiples recorridos DFS para un solo grafo.

3.2.1. Ordenamiento con DFS

Una de las aplicaciones de DFS, así como de otros algoritmos de recorridos, es ordenar los nodos de un grafo dado un recorrido. En el caso de DFS sólo hay que hacer unas pequeñas adiciones para obtener los nodos ordenados. A continuación se menciona el proceso:

- Cuando se marca un nodo, se asigna una marca de tiempo.
- En el caso de no tener nodos adyacentes no visitados pero aún no haber visitado todos los nodos, ocurren 2 casos:
    - No hay aristas *backward*, por lo tanto se aplica *backtracking* y cuando se regresa a un nodo visitado el marcador de tiempo no aumenta.
    - Hay aristas *backward*, en este caso se visita el nodo de la arista *backward* y se ejecuta el paso 2 del algoritmo 2, si no hay nodo a visitar se recorre al siguiente nodo con la marca de tiempo mayor y se vuelve a ejecutar el paso 2 del algoritmo 2. Este proceso se repite haste tener nodo a visitar.

Para el grafo de la Figura 3.7 el orden de los nodos queda:

$$DFS(\mathscr{G}) = \{(c,1),(a,2),(b,3),(i,4),(g,5),(f,6),(h,7),(e,8),(d(9)\}$$

Lo que obtenemos es una lista de tuplas, cada una está formada por la etiqueta del nodo y su correspondiente marca de tiempo. De esta forma podemos ordenar los nodos de un grafo

El tiempo de ejecución de DFS está dado por el peor de lo casos que es visitar todos los nodos y todas las aristas, es decir, $O(|V|+|E|)$ [9].

3.3. Redes neuronales

Una neurona artificial[1] contiene un conjunto de entradas, un conjunto de pesos asociado a las entradas, una función integradora y una función de activación o salida. La Figura 3.8 muestra de forma esquemática el modelo de una neurona artificial.

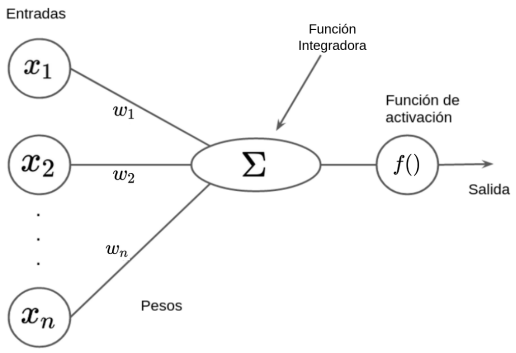

Figura 3.8: Modelo de una neurona en una red neuronal artificial.

Cada x_i es una entrada que tiene asociado un peso w_i. La función integradora está dada por la suma ponderada de cada entrada x_i con su correspondiente peso w_i más un sesgo b, es decir,

$$z = w_1x_1 + w_2x_2 + w_2x_2 ... + w_nx_n + b. \tag{3.1}$$

Para obtener la salida de la neurona se evalúa una función de activación $f()$ con la

[1] El contenido de esta sección fue extraída y adaptado de mi tesis de licenciatura [7]

suma ponderada de la ecuacion 3.1, por ejemplo una función sigmoide. Más adelante se describen a detalle las funciones de activación más comunes.

Una red neuronal artificial está conformada por múltiples neuronas que se agrupan de distintas maneras. La combinación del número de neuronas, el tipo de función de activación, el número de conexiones que puede tener una neurona con otras y el número de capas ocultas, da paso a un gran número de arquitecturas de redes neuronales.

La mayoría de las arquitecturas de redes neuronales cuenta con una capa de entrada y una capa de salida. La diferencia entre las arquitecturas se da en la topología, la cual consiste en la organización y la disposición de las neuronas y sus conexiones para formar una capa y conectividad de distintas capas. Por lo tanto, para diseñar la topología de una red neuronal se tiene que especificar: el número de capas, el número de neuronas en cada capa, el grado de conectividad de cada neurona y el tipo de conexiones entre cada capa. Dentro de las diferentes topologías que puede tener una red neuronal existen tres familias principales: las de conexión hacia adelante, las de conexiones laterales, y las de conexiones recurrentes (Figura 3.9).

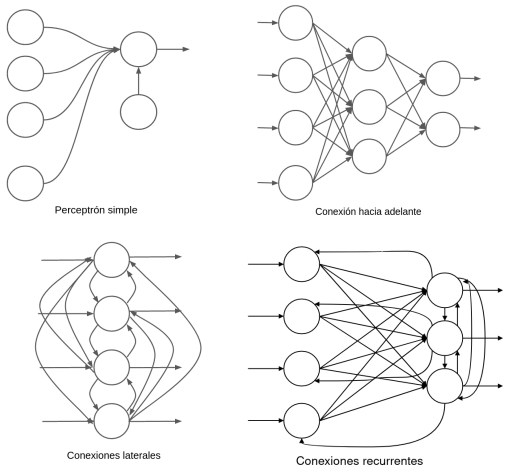

Figura 3.9: Diferentes familias de topologías en una red neuronal [43] .

Una de las arquitecturas más utilizadas es la perceptrón multicapa que tiene una to-

pología de conexión hacia adelante y agrupa conjuntos de neuronas en capas totalmente conectadas. A la capa que recibe los valores de los atributos de nuestro problema se le denomina capa de entrada, a la capa que contiene las salidas se le conoce como capa de salida y al resto de las capas se les llama capas ocultas. La Figura 3.10 muestra un esquema de una red neuronal artificial de tipo perceptrón multicapa, con una capa de entrada, una capa oculta y una capa de salida.

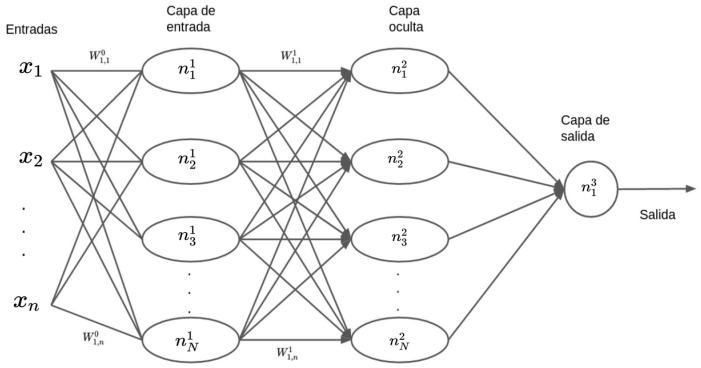

Figura 3.10: Ejemplo de una red neuronal artificial del tipo perceptrón multicapa.

Las entradas de la red neuronal (x_n) son las características o atributos de nuestro problema. Las neuronas se representan con n_i^k, donde i corresponde al número de la neurona y k al número de la capa. Los pesos entre las neuronas se representan como $w_{i,j}^k$, que indica que es un peso entre la neurona i de la capa k y la neurona j de la capa $k-1$. A su vez, el sesgo de la neurona i en la capa k se representa por b_i^k . La salida de una neurona i en la capa k está dada por:

$$a_i^k = f\left(\overbrace{\sum_{j=1}^{m}\left[w_{i,j}^{\{k\}}\cdot a_j^{\{k-1\}}\right]+b_i^{\{k\}}}^{z_i^{\{k\}}}\right), \mathrm{k} = 2, \ldots, \mathrm{L} \tag{3.2}$$

donde a_i^k representa la salida de una neurona, i es el número de la neurona, k es la capa en la red, $a_j^{\{k-1\}}$ es la salida de la neurona j de la capa $k-1$, $f()$ es la función de

activación, m es el número de salidas en la capa $k-1$ y L es el número total de capas de la red. Cuando $k=2$, $a_j^{\{k-1\}}$ se corresponde con las entradas x_j.

Una de las partes principales de una red neuronal artificial es la función de activación. Existen distintos tipos de funciones, cada una de ellas está enfocada a diferentes tipos de problemas; a continuación se mencionan algunas de estas funciones de activación.

- *Rectified Linear Unit (ReLU)*: Es una de las funciones de activación más populares La función ReLU es de la forma:

  $$f(x) = max(0,x)$$

  Una variante de la función ReLU es la función softplus que es de la forma:

  $$f(x) = log(exp(x)+1)$$

  A este tipo de funciones se les conoce como rectificadoras.

- Sigmoide: la función de activación sigmoide está definida por:

  $$f(x) = \frac{1}{(1+e^{-x})}$$

  La función hace que los errores más pequeños converjan a 0 y los errores grandes converjan a 1. En los inicios de la redes neuronales fue una de las funciones más usadas.

- Tanh: la función tanh (tangente hiperbólica) nos dará valores entre [-1,1] pero centrada en 0. Por lo tanto, los valores más grandes convergen a 1 y los más pequeños a -1. La función Tanh está dada por:

  $$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Softmax: es una generalización de la función logística. Usada en problemas de clasificación multietiqueta. La función softmax esta dada por:

$$SoftMax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

La elección de la función de activación en la capa de salida depende directamente del problema a resolver. Por ejemplo, si se tiene un problema donde el comportamiento está definido como una clasificación binaria se puede optar por la función sigmoide; si lo que se quiere es clasificar múltiples objetos comúnmente se usa la función Softmax [26]. En las capas ocultas es común usar funciones no saturadas, como ReLU o softplus, para disminuir el problema del desvanecimiento y explosión de los gradientes.

3.4. Redes neuronales recurrentes

Las redes *feed-forward* no tienen ciclos, están organizadas en capas que se conectan de manera unidireccional y producen una única salida para un cada dato de entrada. Sin embargo, las redes recurrentes permiten conexiones de manera arbitraria y ciclos, con esto se puede conseguir crear cierto grado de temporalidad, permitiendo que este tipo de redes tengan memoria [2]. Las redes recurrentes se consideran sistemas dinámicos o redes espacio-temporales, donde el cálculo de una entrada en un paso depende del paso anterior o en algunos casos del paso futuro.

Sin embargo las redes recurrentes presentan ciertas problemáticas en su entrenamiento, debido a que los gradientes retropropagados tienden a crecer de manera desmesurada o a desvanecerse con el tiempo ya que no solo dependen del error presente sino también de los pasos anteriores, lo cual dificulta aprender dependencias a largo plazo [20].

Las redes recurrentes se utilizan en una gran variedad de tareas, que van desde el tratamiento de secuencias, la continuación de una trayectoria, la predicción no lineal y

la modelación de sistemas dinámicos.

3.4.1. Memorias de corto y largo plazo (LSTM)

Una celda de memoria de corto y largo plazo (LSTM, por las siglas en inglés de *Long short-term memory*) es un tipo de red recurrente que tiene un estado C_t, donde t es una marca de tiempo, y varias compuertas cuyo objetivo es controlar el flujo de la información [20]. En una LSTM existen tres tipos de compuertas:

1. Compuerta de entrada. Esta compuerta controla cuánto de la información de la nueva entrada x_t y de la salida anterior h_{t-1} se agregar al estado anterior C_{t-1}.

$$i_t = \sigma(W_i \cdot [h_{t-1,}x_t] + b_i)$$
$$\tilde{C}_t = \tanh W_c \cdot [h_{t-1}, x_t] + b_C$$

2. Compuerta de olvido. Esta compuerta permite a la celda olvidar información del estado anterior C_{t-1}, dejando así espacio para posible información nueva.

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

3. Compuerta de salida. Esta compuerta controla cuánto de la información del nuevo estado C_t se utiliza para obtener la salida h_t, que pasa al siguiente paso de tiempo $t+1$.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot \tanh C_t$$

Aquí, $[h_{t-1}, x_t]$ denota la concatenación de h_{t-1} con x_t.

Además de las compuertas, se calcula el nuevo estado C_t a través de la suma del estado anterior C_{t-1} regulado por la compuerta de olvido y la nueva información de la compuerta de entrada.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t.$$

Al igual que la salida h_t, el nuevo estado C_t pasa la siguiente paso de tiempo $t+1$.

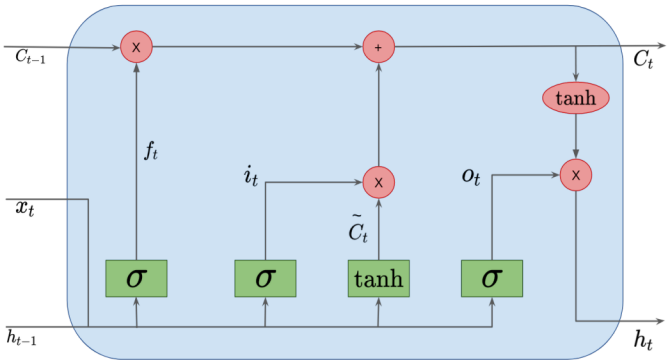

Figura 3.11: Bloque de memoria de una LSTM.

En la Figura 3.11 vemos el diagrama de una memoria LSTM. En dicha memoria vemos que tenemos que cada celda tiene tres entradas: la entrada del tiempo actual x_t, la salida anterior h_{t-1} y el estado anterior C_{t-1}. Además de las entradas, la celda cuenta con dos salidas : el nuevo estado C_t y la salida h_t.

3.4.2. Unidad recurrente regulada (GRU)

Las redes GRU son otro tipo de redes recurrentes, sin embargo a diferencia de las LSTM las GRU solo poseen dos compuertas.

- Compuerta de actualización. Esta compuerta indica cuánto del contenido de las celdas anteriores hay que mantener. Esta compuerta combina la de entrada y olvido de una LSTM

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

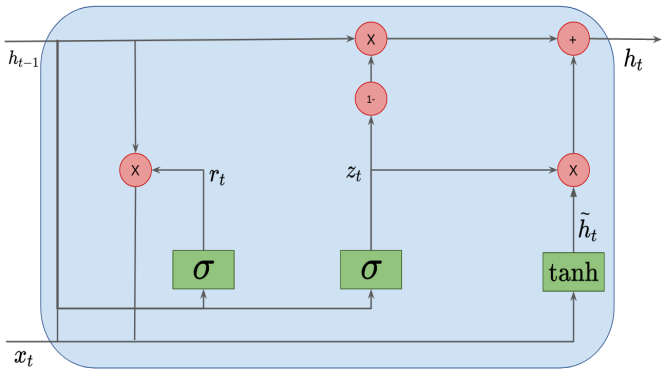

Figura 3.12: Bloque de memoria de una GRU.

- Compuerta de reinicio. Esta compuerta define cómo combinar la nueva entrada con la del estado anterior

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

En la Figura 3.12 vemos el diagrama de un bloque GRU, donde la salida h_t está definida como la combinación entre la salida previa y la salida propuesta.

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

Mientras que la salida h_t está definida por:

$$\tilde{h}_t = \tanh(W_{\tilde{h}}[h_{t-1} * r_t, x_t] + b_{\tilde{h}})$$

3.5. Entrenamiento de redes neuronales

El entrenamiento de las redes neuronales se formula como el problema de encontrar los valores de los pesos y los sesgos que minimicen la función de error. La solución a este problema no se puede obtener de manera analítica, por lo que se emplean algoritmos iterativos que aproximan la solución. Entre los algoritmos más populares se encuentran los que usan el gradiente para guiar la búsqueda, como el descenso por el

gradiente. Por otro lado, el cálculo de los gradientes en redes con múltiples capas puede ser muy costoso computacionalmente y se han desarrollado algoritmos eficientes para ello como la retropropagación hacia atrás.

3.5.1. Optimizadores

A continuación se describen dos algoritmos iterativos para encontrar los valores de los parámetros que minimicen la función de error.

Descenso por el gradiente (GD)

El descenso por el gradiente es un algoritmo iterativo usado para encontrar el mínimo de una función, siguiendo la dirección opuesta del gradiente [41]. Debido a que la primera derivada mide la rapidez con la que crece una función, al tener esta información podemos seguir la dirección contraria y con esto iremos descendiendo hasta un punto mínimo. La forma en que trabaja el algoritmo es la siguiente:

1. Localizarse en un punto inicial de la función.

2. Descender usando la información de las primeras derivadas parciales de la función respecto a los parámetros.

3. Ya que sabemos hacia dónde tenemos que descender, hay que decidir cuánto descenderemos. Para esto se utiliza un hiperparámetro que controla el tamaño del paso que se dará para descender; en el caso de la redes neuronales a este hiperparámetro se le conoce como tasa de aprendizaje. La regla de actualización de un parámetro θ en una iteración t está dada por

$$\theta[t] = \theta[t-1] - \alpha \frac{\partial E(\theta[t-1])}{\partial \theta[t-1]}$$

α es la tasa de aprendizaje y $E(\theta)$ es la función dependiente de θ que se quiere minimizar.

4. Para detener el proceso del descenso se puede hacer de dos maneras: la primera es cuando la diferencia entre la nueva posición y la posición de donde se viene está por debajo de cierto umbral. En la segunda el proceso se detiene cuando se ha hecho un número predefinido de iteraciones.

Para el entrenamiento de las redes neuronales, $E(\theta)$ corresponde a la función de error y θ son los pesos y sesgos.

Descenso por gradiente estocástico (SGD)

El algoritmo de descenso por el gradiente estocástico (SGD) es similar al GD, pero SGD aproxima el gradiente usando una muestra del conjunto de entrenamiento en lugar de usar el conjunto completo. En la presente tesis se hace uso del algoritmo Adam, el cual es una variante de SGD que tiene una tasa de aprendizaje adaptable y hace uso del primer y segundo momento de los gradientes [23].

El proceso de entrenamiento de una red neuronal usando descenso por el gradiente o alguna de sus variantes se lleva a cabo en dos etapas. La primera etapa consiste en propagar los datos a través de la red neuronal hasta obtener las salidas, a esta etapa se le conoce como propagación hacia adelante. La propagación hacia adelante no modifica los pesos ni sesgos de la red neuronal, sino que se encarga de propagar las entradas capa por capa hasta producir las salidas. Los pesos regularmente se inicializan con valores aleatorios pequeños y los sesgos con ceros. La propagación hacia adelante se realiza como se describe en la ecuación 3.2.

La segunda etapa, llamada propagación hacia atrás (*Backpropagation*), se encarga de calcular los gradientes de la función de error respecto a los pesos y sesgos y finalmente ajustarlos con la regla de actualización correspondiente. Presuponemos un conjunto de entrenamiento con n ejemplos $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$, una red neuronal con L capas cuyas neuronas tienen una función de activación sigmoide, una capa de salida con una neurona con función de activación lineal o identidad y el error cuadrático

medio (ECM) como $E(\theta)$:

$$ECM(\mathbf{W},\mathbf{b}) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

donde $y^{(i)}$ es la salida deseada y $\hat{y}^{(i)}$ es la salida de la propagación hacia adelante con la entrada $\mathbf{x}^{(i)}$ del ejemplo i. El algoritmo 3 describe el entrenamiento de esta red usando descenso por el gradiente y el algoritmo de propagación hacia atrás.

En el caso de redes neuronales recurrentes el entrenamiento se realiza con el algoritmo de retropropagación a través del tiempo, el cual se trata de una modificación del algoritmo 3. Esta modificación toma en cuenta el hecho de que los pesos y sesgos son compartidos por todos los pasos de tiempo, por lo que el gradiente no solo depende del

paso de tiempo actual sino también de los pasos de tiempo anteriores [39] [44].

Algoritmo 3: Algoritmo de propagación hacia atrás

Entrada: Conjunto de entrenamiento $(\mathbf{x}^{(1)},\mathbf{y}^{(1)}),\ldots,(\mathbf{x}^{(n)},\mathbf{y}^{(n)})$, valor de α y número de épocas C

1. Se inicializan los pesos $w_{i,j}^{k}[0]$ y sesgos $b_i[0]$ con valores aleatorios, para todo k,i,j.

2. Para $t=1,\ldots,C$, donde C es el número de iteraciones

   a) Asignamos $\Delta w_{i,j}^{\{k\}}=0$ y $\Delta b_i^{\{k\}}=0$ para cada k,i,j

   b) Para $e=1,\ldots,n$, con n el número total de ejemplos de entrenamiento.

      1) Hacemos propagación hacia adelante como en la ecuación 3.2 y guardamos cada $a_i^{\{k\}}$ para $k=2,3,...,L$, donde L es el número total de capas que tiene la red neuronal.

      2) Calculamos $\delta^{\{L\}}=a^{\{L\}}-y^{(e)}$ para la salida de la última capa de la red.

      3) Calculamos $\delta_j^{\{k-1\}}=\sum_i\left[(w_{i,j}^{\{k-1\}})^T\delta_i^k\left(a_j^{\{k-1\}}(1-a_j^{\{k-1\}})\right)\right]$ para $k=L,L-1,\ldots,2$, donde $\left(a_j^{\{k-1\}}(1-a_j^{\{k-1\}})\right)$ es la derivada de la función de activación sigmoide respecto a $z_j^{\{k-1\}}$.

      4) Actualizamos

$$\Delta w_{i,j}^{\{k\}}=\Delta w_{i,j}^{\{k\}}+a_j^{\{k\}}\delta_i^{\{k-1\}}$$
$$\Delta b_i^{\{k\}}=\Delta b_i^{\{k\}}+\delta_i^{\{k-1\}}.$$

   c) Se calcula el valor de la actualización de cada peso y sesgo de cada capa usando las $\Delta_{i,j}^{\{k\}}$ acumuladas para todos los ejemplos.

$$w_{i,j}^{\{k\}}[t]=w_{i,j}^{\{k\}}[t-1]-\alpha\frac{\Delta w_{i,j}^{\{k\}}}{n}$$
$$b_i^{\{k\}}[t]=b_i^{\{k\}}[t-1]-\alpha\frac{\Delta b_i^{\{k\}}}{n}$$

Capítulo 4

Método propuesto

En este capítulo se describe el método propuesto para generar grafos conexos y disconexos usando GraphGen [14] y una transformación en la representación de los grafos disconexos. GraphGen es un método de generación de grafos etiquetados que emplea una celda recurrente tipo LSTM [20].

El método busca aprender un modelo generativo de grafos que esté libre de presuposiciones, sea independiente del dominio y capture la compleja interacción entre las etiquetas semánticas y la estructura de los grafos, además de ser escalable y que tenga la capacidad de generar grafos etiquetados.

GraphGen toma un conjunto de grafos $G = \{\mathscr{G}_1, \cdots, \mathscr{G}_n\}$, donde a cada $\mathscr{G}$ se le aplica un función descrita más adelante. Por lo tanto, ahora tenemos un conjunto $S = \{F(\mathscr{G}_1), \cdots, F(\mathscr{G}_n)\}$ y lo que se busca es aprender $p_{modelo}(S) \approx p_{modelo}(G)$ usando un modelo auto-regresivo.

Antes de describir la arquitectura de GraphGen se describe la representación de los grafos utilizada.

4.1. Códigos mínimos DFS

Como se mencionó anteriormente, GraphGen aplica una función a los grafos. A continuación se describe dicha función o representación.

La representación de los grafos está dada por una 5-tupla, esta se obtiene a través de hacer un recorrido DFS con backtracking sobre el grafo. La 5-tupla es de la forma $(t_u, t_v, L_u, L_{(u,v)}, L_v)$, donde t_u es la marca de tiempo que tiene el nodo origen, t_v es la marca de tiempo que tiene el nodo destino, L_u es la etiqueta que tiene el nodo origen, $L_{(u,v)}$ es la etiqueta de la arista que une al nodo u con el nodo v y L_v es la etiqueta del nodo destino.

Para obtener esta 5-tupla, los grafos deben de cumplir dos características: ser grafos conexos y que al aplicarle una permutación sobre las filas de la matriz de adyacencia del grafo original el resultado sea un grafo isomorfo al original. Si el grafo no cumple con esto no se puede conseguir su representación. Como se mencionó en los objetivos, la tesis se enfoca en obtener una representación para los grafos que no tienen estas dos características, la cual se describe más adelante.

Para los grafo que sí cumplen estas características, se obtiene un ordenamiento de sus nodos usando DFS (véase sección 3.2.1). En particular, se usa una modificación de DFS que no solo guarda el nodo y su marca de tiempo, sino también la marca de tiempo del nodo destino, la etiqueta de la arista y la etiqueta del nodo destino.

Como se mencionó anteriormente un grafo puede tener varios ordenamientos DFS, por lo que GraphGen toma el mínimo, es decir, se obtienen todos los posibles ordenamientos (lexicográficamente) y se toma el que tenga menor tamaño.

La Figura 4.1 muestra un ejemplo de cómo se obtiene un código mínimo DFS. En el ejemplo también podemos ver las aristas *backward*, las cuales están representadas como líneas punteadas. Las 5-tuplas que se obtienen de los dos recorridos son las siguientes:

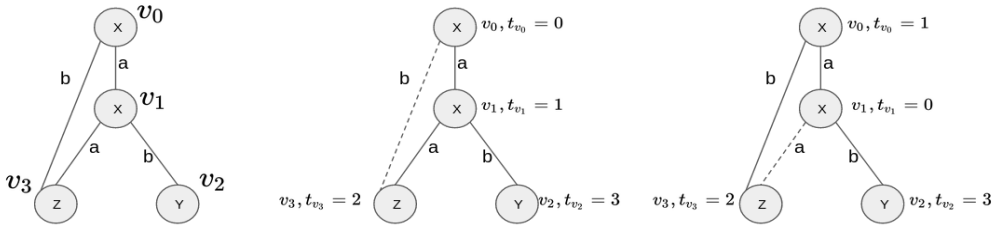

Figura 4.1: Ejemplo de un recorrido mínimo DFS.

$$\langle (0,1,X,a,X)(1,2,X,a,Z)(2,0,Z,b,X)(1,3,X,b,Y)\rangle$$
$$\langle (0,1,X,a,X)(1,2,X,b,Z)(2,0,Z,a,X)(0,3,X,b,Y)\rangle$$

Para los dos recorridos tenemos que las letras mayúsculas pertenecen a las etiquetas de los nodos y las letras minúsculas pertenecen a las etiquetas de las aristas. GraphGen recibe como entrada una 5-tupla por cada grafo de ejemplo y con esta representación el modelo ve a un grafo como una cadena , por lo que trata la generación como si fuera generación de cadenas de texto.

4.2. Arquitectura de GraphGen

El modelo GraphGen usa una LSTM personalizada que contiene un función de transición h_i, una función de *embedding* s_i y 5 funciones de salida, una para cada una de los 5 componente de la 5-tupla.

$$h_i = F_{trans}(h_{i-1}, f_{emb}(s_{i-1}))$$

$$t_u \sim_M \theta_{t_u} = f_{t_u}(h_i)$$

$$t_v \sim_M \theta_{t_v} = f_{t_v}(h_i)$$

$$L_u \sim_M \theta_{L_u} = f_{L_u}(h_i)$$

$$L_e \sim_M \theta_{L_e} = f_{L_e}(h_i)$$

$$L_v \sim_M \theta_{L_v} = f_{L_v}(h_i)$$

$$s_i = concat(t_u, t_v, L_u, L_e, L_v)$$

La función de transición es una LSTM apilada con 4 capas, mientras que la función de *embedding* es una función lineal simple y las 5 funciones de salida se tratan de una perceptrón multicapa con un dropout de 0.2. Se usa *AdamOptimizer* con un tamaño de lote de 32. En la Figura 4.2 se muestra un diagrama de la arquitectura de GraphGen.

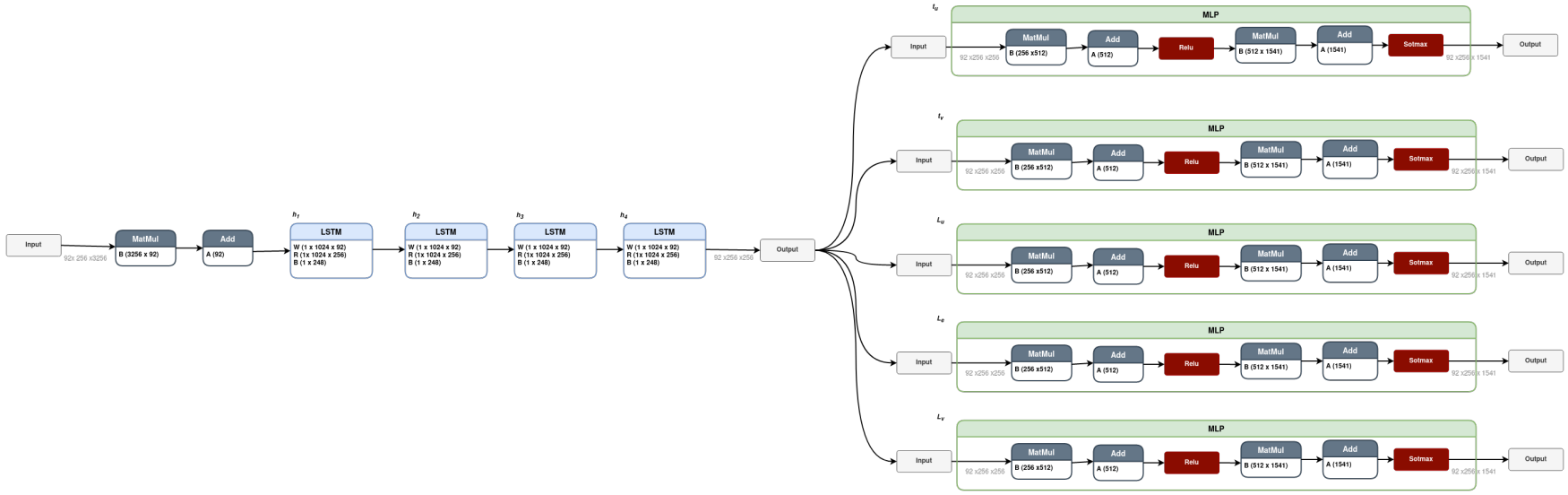

Figura 4.2: Diagrama de la arquitectura de GraphGen.

4.3. Métricas

Uno de los problemas que se tiene en los modelos de generación, es cómo evaluarlos. En los modelos de predicción podemos tomar el valor meta y compararlo con la salida de los modelos y con base en esto evaluamos la calidad de la predicción. Sin embargo, en un modelo de generación no se tienen valores meta. Para medir la generación de un modelo se usan métricas dependiendo de lo que se esté generando.

Para la generación de grafos, las métricas que se usan tienen diferentes fines, a continuación se describen las métricas a usar y su fin.

- ***Similitud de grafos.*** Con esta métrica se busca capturar la similitud de los grafos generados, de tal forma que considere tanto la estructura de los grafos como las etiquetas generadas.

  - ***Maximum Mean Discrepancy (MMD).*** Mide la distancia entre dos grafos, haciendo coincidir pares de subgrafos con diferentes radios y distancias [11]. Cuanto menor sea la distancia, mejor es el rendimiento. Esta métrica usa diferentes núcleos, estos calculan la distancia de diferentes formas, a continuación se mencionan los núcleos que se usan.

    - ***Sub-graph Pairwise Distance Kernel(NSPDK.)***
    - ***MMD Degree***
    - ***MMD Clustering***
    - ***MMD Orbits***
    - ***MMD Node label***
    - ***MMD Edge label***

    El núcleo NSPDK es la métrica que se puede considerar más importante, ya que captura la similitud global en lugar de las propiedades individuales locales.

- ***Métricas estructurales.*** Estas métricas están orientadas a la estructura de los grafos generados, es específico el promedio del número de nodos y de aristas.

- ***Redundancia.*** A pesar de obtener un valor bajo en las métricas de similitud, los grafos pueden ser iguales a los que se usaron para el entrenamiento y por lo tanto no ser muy útiles. Idealmente queremos grafos diversos y similares, pero no idénticos. Para eso se usan las siguientes métricas :

    - ***Novelty***. Mide el porcentaje de grafos generados que no son subgrafos de los grafos de entrenamiento y viceversa.
    - ***Uniqueness.*** Captura la diversidad de los grafos generados.

Estas métricas nos permiten evaluar distintos aspectos de los grafos que se generen por los modelos entrenados.

4.4. Modificación de la representación de los grafos

Uno de los inconvenientes que tiene GraphGen son los requerimientos que impone en los grafos para crear su representación. A continuación se describe el procedimiento propuesto para modificar los grafos que no cumplen estas características de tal manera que los grafos resultantes sí las cumplan.

4.4.1. Grafos disconexos

Es de interés que el modelo genere grafos que sean disconexos. Sin embargo, con la función de representación de GrapGen esto no es posible, ya que al aplicar un recorrido DFS este sólo nos entregaría el recorrido de uno de los componentes conexos y no de todo el grafo. Para resolver esto, no se modificará la representación si no que la modificación será sobre el grafo y las etiquetas de las aristas.

Sabemos que un grafo disconexo tiene al menos un componente conexo. Con esto en mente, vamos a asignar una etiqueta de pertenencia a las aristas de cada componente conexo, este proceso se realiza cuando se construye el grafo. La etiqueta que se le asigne a cada arista debe ser la misma para todas y no debe de estar relacionada con las etiquetas de los nodos.

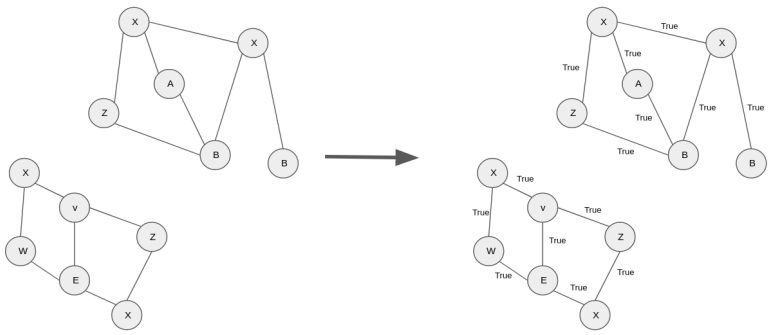

Figura 4.3: Ejemplo del etiquetado de un grafo disconexo.

En la Figura 4.3 vemos un ejemplo de un grafo disconexo con 2 componentes conexos, usamos de etiqueta el valor *True*. Cabe mencionar que esta etiqueta no tiene que ser booleana, ya que para el modelo es una simple etiqueta que no da ningún valor.

Ya que tenemos el grafo con las aristas etiquetadas, el siguiente paso es unir los grafos. Dado que podemos tener más de dos componentes conexos, el proceso de unir los grafos lo podemos realizar de manera iterativa. A continuación se describe este proceso:

- Mientras existan componentes conexos no conectados:

    1. Tomamos dos componentes conexos
    2. Calculamos los dos nodos más cercanos entre dos componentes vecinas.
    3. Tomamos los dos de menor distancia entre si y se unen con una arista y una etiqueta, la etiqueta debe ser la misma para todas las aristas generadas.

4. Se genera un nuevo grafo que ahora se compone de la unión de las dos componentes conexas y si es el caso de las componentes restantes.

Como tenemos grafos donde los nodos tiene coordenadas x y y, podemos calcular la distancia entre dos nodos u y v usando su distancia euclidiana:

$$d_e(u,v) = \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2}$$

donde (x_u, y_u) y (x_v, y_v) son las coordenadas del nodo u y el nodo v respectivamente.

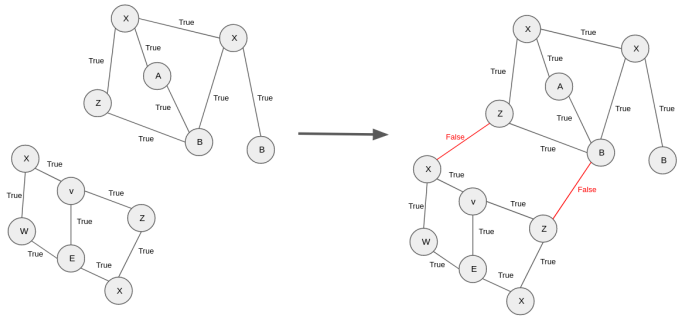

Figura 4.4: Ejemplo de la unión de un grafo disconexo con dos componentes conectados.

En la Figura 4.4 vemos el resultado de unir dos componentes conexos. A pesar de que este algoritmo funciona para cualquier grafo disconexo, hay algunas excepciones. Una de estas excepciones ocurre cuando uno de los componentes conexos sólo tiene un solo nodo. Para este caso, se revisa que el número de nodos de cada componente sea mayor de 2, sino se cumple, se toma ese nodo y se busca el más cercano del otro componente. Otra excepción ocurre cuando la mayoría de los componentes conexos solo tienen un nodo. Para este caso, sólo tomamos los dos nodos y no se calcula la distancia, sino que simplemente se conectan los dos nodos con una arista y su etiqueta.

Ahora ya tenemos un grafo conexo, sin embargo no se garantiza que al aplicar la función de permutación se obtenga un grafo isomorfo al original.

4.4.2. Grafos no isomorfos

Cuando se aplica la función de permutación al grafo se busca un grafo isomorfo al original. Por definición, un grafo isomorfo tiene el mismo número de nodos, de aristas y cada nodo tiene el mismo grado. Por lo que si el grafo no es isomorfo lo que obtenemos es un grafo disconexo y caemos en el problema para el que ya hemos dado la solución.

Como este problema recae en el problema resuelto anteriormente, podemos hacer dos experimentos:

1. Se aplica la función de permutación y si no es grafo isomorfo, entonces aplicamos el proceso del grafo disconexo.

2. No se aplica la función de permutación, por lo que el modelo será entrenado con los grafos originales, sólo se verifica que sea un grafo conexo.

Capítulo 5

Resultados experimentales

Hasta el momento sólo se ha descrito todo el proceso para cualquier tipo de grafo, sin embargo para hacer los experimentos es importante tener una tarea dónde validar el método propuesto. Para esto tomaremos la generación de grafos de cristales de perovskita. Los grafos de perovskita tienen una gran variedad de tamaños y formas, además de tener una cantidad considerable de etiquetas en los nodos y tener grafos disconexos.

5.1. Obtención y preprocesamiento de los datos

En la Figura 5.1 se muestran los pasos a seguir en la obtención, el preprocesamiento de los archivos que contienen los cristales y su visualización como grafos.

Los grafos de cristales se obtienen de dos sitios ***Material Project*** **(MP)** [36] y ***Crystallography Open Database*** **(COD)**, de los cuales obtendremos 1082 y 32419 cristales respectivamente.

Para aplicar la modificación propuesta, hay que preprocesar los cristales, los cuales están en archivos CIF y llevarlos hasta archivos xyz. En estos últimos se realiza la modificación de los grafos. Los archivos CIF son archivos de cristalografía que contienen cada nodo con su posición *xyz* y el elemento químico, además de información química

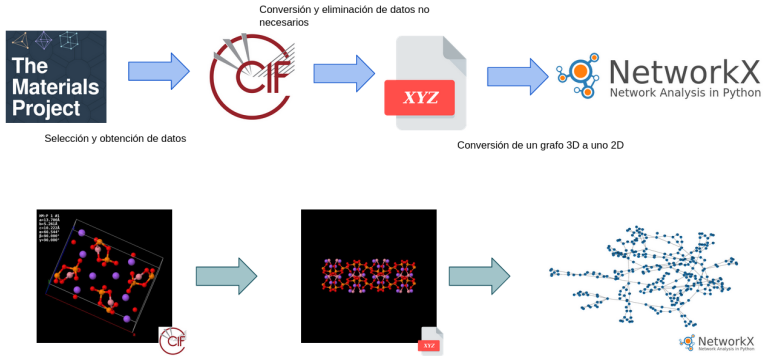

Figura 5.1: Diagrama de los pasos para la obtención de datos y la conversión de los grafos.

adicional que no es de interés para este trabajo.

Un archivo xyz, sólo contiene la posición *xy* y el elemento del nodo, al pasar a este nuevo formato se pierde una dimensión y todos los demás datos que no se necesitan. Para realizar este proceso usamos una biblioteca xyz2graph hecha específicamente para esto, lo único que necesita es darle un tamaño de celda para extraer el grafo. Encontrar un tamaño de celda que obtenga la mejor representación de un cristal es un problema que depende de cada grafo. Como el objetivo del trabajo es simplemente validar la generación de grafos, elegimos una celda fija de un tamaño 2×2 o celda doble para todos los cristales. Una celda unitaria es la porción mas simple de una estructura cristalina. El tamaño de celda doble permite apreciar la estructura y la periodicidad del cristal.

5.1.1. Características de los datos

Las características de los grafos con los que se trabaja son las siguientes:

- **Etiquetas en los nodos.** Los nodos de las perovskitas pueden tener 118 posibles etiquetas. Esto se debe a que existen 118 elementos químicos.
- **Tamaño.** Cada grafo tiene un mínimo de 20 nodos, por lo tanto puede tener

$n(n-1)/2$ máximo de aristas en el caso de ser un grafo completo.

- **Estructura.** Se refiere a si el grafo es conexo o disconexo y cuántos componentes conexos tiene cada grafo; el número total de grafos conexos es de 540 y de grafos disconexos 542.

5.1.2. Conjuntos de datos

Los conjuntos de datos se basan en los diferentes experimentos que se busca hacer con la representación de los grafos originales y con la propuesta en este trabajo. Se formaron 4 conjuntos: el primero contiene 540 grafos de perovskitas de MP que son conexos, el segundo contiene 542 grafos disconexos de MP, el tercero contiene tanto los grafos conexos como los disconexos de MP y el cuarto es un conjunto que contiene 32419 grafos, que es una combinación de grafos conexos y disconexos de MP y COD.

5.2. Experimentos

El número de experimentos que se realizarán depende de los conjuntos de datos y de la forma en que se modificaron los grafos. En todos los experimentos, se entrenan los modelos durante 10,000 épocas y para la evaluación se generan 2560 grafos; excepto el último experimento, que se entrena durante 4,000 épocas.

- ***Experimento con grafos conexos.*** El primer experimento sólo contiene grafos conexos. Se usa GraphGen con la configuración mencionada en la sección 4.2. Dentro de este experimento hay 2 variantes, las cuales se basan en cambiar el número de nodos previos que puede usar el modelo para regresar dentro del grafo generado.
    - ***Experimento con 2 nodos previos***
    - ***Experimento con 10 nodos previos***

- ***Experimento con grafos disconexos.*** Este toma el conjunto de grafos disconexos. En este experimento se aplica la modificación a los grafos para obtener grafos conexos y se entrenan dos modelos, uno donde se aplica la función de permutación y otro donde no se aplica la función de permutación.

- ***Experimento con todos los grafos.*** Este toma el conjunto de grafos conexos y los disconexos. Por lo tanto, a los grafos disconexos se les aplica la modificación del grafo para obtener grafos conexos y no se aplica la función de permutación.

- ***Experimento de gran tamaño.*** Este experimento está hecho con el cuarto conjunto. Esta hecho para ver el comportamiento del modelo con una cantidad considerable de grafos. Para fines del experimento no se contabilizó la cantidad de grafos que son conexos y los que son disconexos, para este experimento no se aplica la función de permutación.

Todos los experimentos se evalúan con las mismas métricas que se mencionan en la sección 4.3. Para hacer esto se toman de forma aleatoria los grafos y se calculan las métricas de cada uno, por lo tanto lo que se reporta es un promedio para cada métrica.

Al calcular los códigos mínimos DFS se obtiene el promedio, máximo y mínimo de aristas y nodos, con esto el modelo obtiene criterios para la detención de generación de nodos y aristas.

5.3. Resultados

Para los resultados de cada experimento, se muestran las métricas antes descritas junto con algunos ejemplos de los grafos de entrenamiento y de grafos generados por el modelo entrenado.

En la Figura 5.2 se muestran algunos ejemplos de entrenamiento para los grafos de cristales que son conexos, mientras que en la Figura 5.3 se muestran ejemplos de

entrenamiento para grafos disconexos.

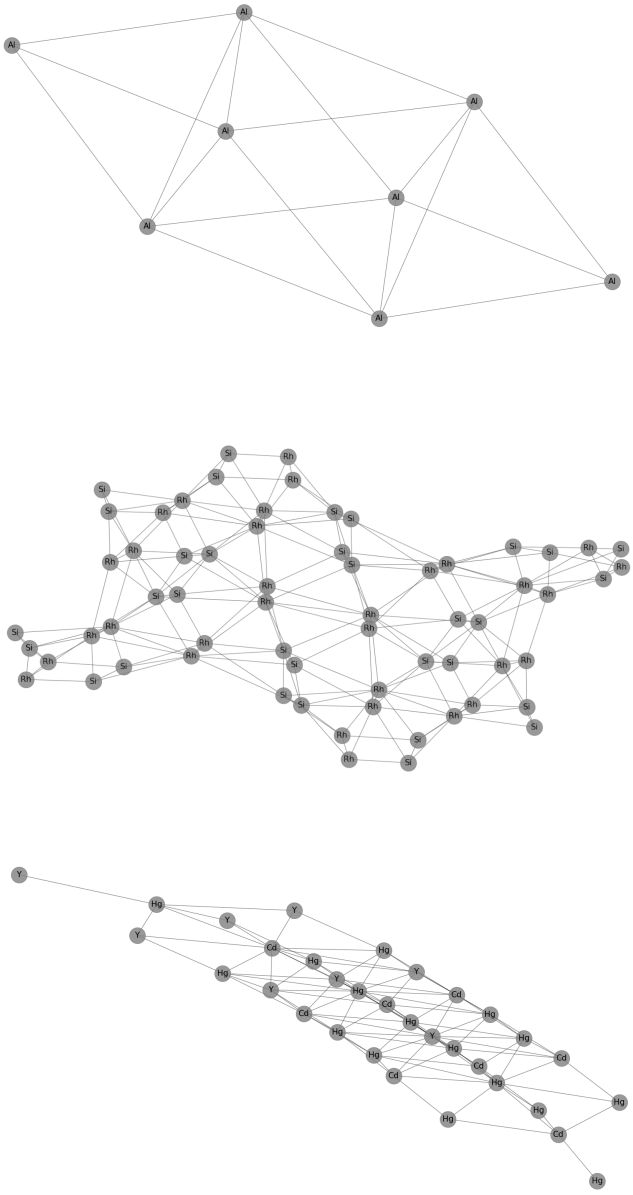

Figura 5.2: Ejemplo de tres grafos de entrenamiento para el experimento de grafos conexos.

Figura 5.3: Ejemplo de tres grafos de entrenamiento para el experimento de grafos disconexos.

5.4. Grafos conexos

El primer experimento, como ya se describió antes, sólo es entrenado con grafos conexos. Los grafos generados que se ilustran en la Figura 5.4, muestran que el modelo genera grafos que ya tienen cierta estructura y complejidad topológica. Sin embargo, la cantidad de nodos que se generan es menor en algunos casos que en los grafos que se le dieron de entrenamiento. Por el lado de las etiquetas se observa que mantienen cierto parecido con los grafos de entrenamiento, esto es que en un grafo no aparecen más de 3 etiquetas diferentes, hecho que también se cumple en los grafos de entrenamiento.

5.4.1. Grafos conexos con 2 nodos previos

Este experimento consta de los mismo datos que el anterior. Sin embargo, el número de nodos previos que se pueden usar en la generación es de 2, en el caso del primer experimento el número óptimo de nodos previo es calculado por el mismo DFS. En la Figura 5.5 se muestran los grafos generados por este experimento. El número de nodos que se generan aumenta, por lo tanto la complejidad y la estructura de los grafos es mayor.

5.4.2. Grafos conexos con 10 nodos previos

En este experimento, el número de nodos previos que se pueden usar es de 10. En el mismo caso que los dos experimentos anteriores se muestra un ejemplo de generación. Los grafos generados se ilustran en la Figura 5.6. Se puede observar que la mayoría de los grafos generados ya no tienen una estructura tan definida como en los experimentos anteriores. Sin embargo, esto no es una mala señal ya que dentro de los ejemplos de entrenamiento hay una cantidad reducida de grafos que no tienen una estructura tan definida. Que se generen este tipo de grafos nos dice que el modelo es capaz de generar cualquier tipo de grafos, aunque la cantidad de grafos de entrenamiento sea reducida.

A continuación se muestran las métricas de los tres experimentos hechos.

La Tabla 5.1 nos muestra el resultado de las métricas para los 3 experimentos. El último experimento es donde obtenemos 4 de las 5 métricas con un valor más bajo, esto nos habla de que un número grande de nodos previos impacta en la capacidad de generación del modelo. Una de las métricas que llama la atención es Uniqueness la cual para los tres experimentos es baja; recordemos que esta métrica nos dice la diversidad de grafos que se generan.

La Figura 5.7 es un ejemplo de por qué la métrica Uniqueness obtiene un valor bajo. Que sean similares hace descender la métrica de diversidad. Este hecho visto desde un punto computacional es malo, sin embargo al tratarse de grafos de cristales puede ser interesante ya que un cristal de forma similar pero con diferentes elementos químicos puede ser interesante.

5.5. Grafos disconexos

En estos experimentos es donde la modificación propuesta en este trabajo se pone a prueba. En el primer experimento no se aplica la función de permutación, es decir el modelo se entrena con los grafos originales. Mientras que en el segundo sí se aplica la función, por lo tanto el modelo se entrena con grafos isomorfos a los originales.

5.5.1. Grafos disconexos sin función de permutación

De igual manera que en los experimentos de grafos conexos, primero se muestran los grafos generados para cada experimento y después la tabla con las métricas.

En la Figura 5.8 se muestran dos grafos generados por este experimento. Las aristas de color rojo son las que tienen la etiqueta de no pertenecer al grafo. El grafo que se encuentra abajo es el resultado de quitar dichas aristas. Los grafos generados tienen cierta complejidad, incluso los componentes conexos mantienen la complejidad. Pero

	Grafos conexos	Conexos prev node 2	Conexos prev node 10
Novelty	0.955457	0.955975	**0.969369**
Uniqueness	**0.55262**	0.528838	0.504464
Promedio de nodos en generación	18.921094	**25.911328**	25.794531
Promedio de aristas en generación	41.725	64.262891	**65.349219**
MMD Degree	0.273265	0.161532	**0.153228**
MMD Clustering	0.143662	0.11319	**0.107206**
MMD Orbits	0.325998	0.128765	**0.125989**
MMD NSPDK	0.065829	**0.062741**	0.063029
MMD Node label	0.065802	0.065189	**0.065103**
MMD Edge label	0	0	0
MMD Joint Node label and degree	0.063223	**0.062875**	0.06307

Tabla 5.1: Resultado de las métricas para los tres experimentos hechos con grafos conexos.

no tienen una estructura bien definida como en los experimentos de grafos conexos, esto es algo que ya se esperaba dado que hay grafos disconexos más variados en cuanto a su estructura.

5.5.2. Grafos disconexos con la función de permutación

Para este experimento, sí se aplica la función de permutación, por lo que si no se obtiene un grafo isomorfo se vuelve a aplicar la modificación descrita en la sección 4 para obtener un grafo conexo.

Los grafos generados mostrados en la Figura 5.9 presentan complejidad y cierta estructura como es el caso del segundo grafo. Los componentes conectados mantienen cierta estructura, aunque no tienen gran complejidad. Otra cosa que se puede notar es que el número de aristas que no pertenecen son en cierto grado menor que en el experimento anterior. En la Tabla 5.2 se muestran las métricas de los dos experimentos con grafos disconexos. Observamos que en el experimento donde se aplica la función de permutación es donde se encuentran los valores más bajos para las métricas. Mientras que para Uniqueness se obtiene hasta ahora el valor más alto, sin embargo sigue siendo un valor bajo. Como en el experimento anterior esto se debe a que se generan grafos con la misma estructura pero diferentes etiquetas en los nodos.

5.6. Grafos conexos y disconexos

Este experimento consta de los dos conjuntos anteriores y no se aplica la función de permutación, por lo que el modelo se entrena con los grafos originales. Este experimento es igual de interesante que el de los grafos disconexos, ya que ahora se probará si el modelo y la representación propuesta tienen la capacidad de generar los dos tipo de grafos.

En la Figura 5.10 se muestran los grafos generados por este experimento. Los grafos

	Grafos disconexos sin permutación	Grafos disconexos con permutacion
Novelty	**0.997506**	0.993789
Uniqueness	0.547211	**0.628915**
Promedio de nodos en generación	27.207031	**27.903125**
Promedio de aristas en generación	35.215234	**36.168359**
MMD Degree	0.073814	**0.072817**
MMD Clustering	0.070666	**0.063918**
MMD Orbits	0.060429	**0.058009**
MMD NSPDK	0.065264	**0.065225**
MMD Node label	**0.069427**	0.071486
MMD Edge label	0.000492	**0.000356**
MMD Joint Node label and degree	0.063533	**0.06326**

Tabla 5.2: Resultado de las métricas para los dos experimentos hechos con grafos disconexos.

	Grafos conexos	Conexos prev node 2	Conexos prev node 10	Grafos disconexos sin permutación	Grafos disconexos con permutacion	Grafos conexos y disconexos
Novelty	0.955457	0.955975	0.969369	0.997506	**0.993789**	0.981818
Uniqueness	0.55262	0.528838	0.504464	0.547211	**0.628915**	0.621935
Promedio de nodos en generación	18.921094	**25.911328**	25.794531	27.207031	27.903125	24.296875
Promedio de aristas en generación	41.725	64.262891	**65.349219**	35.215234	36.168359	38.454687
MMD Degree	0.273265	0.161532	0.153228	0.073814	**0.072817**	0.123611
MMD Clustering	0.143662	0.11319	0.107206	0.070666	**0.063918**	0.137349
MMD Orbits	0.325998	0.128765	0.125989	0.060429	**0.058009**	0.13604
MMD NSPDK	0.065829	0.062741	0.063029	0.065264	0.065225	**0.063489**
MMD Node label	0.065802	0.065189	0.065103	0.069427	0.071486	**0.063449**
MMD Edge label	0	0	0	0.000492	0.000356	**0.000259**
MMD Joint Node label and degree	0.063223	0.062875	0.06307	0.063533	0.06326	**0.063038**

Tabla 5.3: Resultado de las métricas para todos los experimentos.

disconexos aunque no llegan a ser complejos también son similares a los grafos disconexos de entrenamiento. En el caso de los conexos si tienen una estructura definida y una buena complejidad.

Algo que sufren todos los experimentos es la métrica Uniqueness, hecho que para este experimento se mantiene, aunque este valor sube. El hecho de que se obtenga un mejor valor en dicha métrica se le puede atribuir a la diversidad de grafos que se le dieron al modelo para entrenar.

La Tabla 5.3 muestra las métricas de este ultimo experimento y de los anteriores, esto con el fin de poder comparar y discutir de mejor manera los resultados.

5.7. Experimento conjunto de gran tamaño

Este se trata del ultimo experimento, el modelo sera entrenado con cerca de 32 mil grafos. La razón de este experimento es medir la capacidad del modelo con un cantidad mayor de grafos y ver si con esto la métrica de Uniqueness aumenta y por ultimo ver si la complejidad de los grafos generados aumenta.

Como resultado obtenemos que la métrica de Uniqueness si aumenta, hecho que se le atribuye a que la cantidad de grafos ahora es mucho mayor que en los otros experi-

	Grafos conexos	Conexos prev node 2	Conexos prev node 10	Grafos disconexos sin permutación	Grafos disconexos con permutacion	Grafos conexos y disconexos	Conjunto de gran tamaño
Novelty	0.955457	0.955975	0.969369	**0.997506**	0.993789	0.981818	0.949153
Uniqueness	0.55262	0.528838	0.504464	0.547211	0.628915	0.621935	**0.989575**
Promedio de nodos en generación	18.921094	25.911328	25.794531	27.207031	27.903125	24.296875	**46.716797**
Promedio de aristas en generación	41.725	64.262891	65.349219	35.215234	36.168359	38.454687	**75.344531**
MMD Degree	0.273265	0.161532	0.153228	0.073814	0.072817	0.123611	**0.102881**
MMD Clustering	0.143662	0.11319	0.107206	0.070666	0.063918	**0.137349**	0.150493
MMD Orbits	0.325998	0.128765	0.125989	0.060429	0.058009	0.13604	**0.128302**
MMD NSPDK	0.065829	0.062741	0.063029	0.065264	0.065225	**0.063489**	0.066607
MMD Node label	0.065802	0.065189	0.065103	0.069427	0.071486	**0.063449**	0.067958
MMD Edge label	0	0	0	0.000492	0.000356	0.000259	**0.001613**
MMD Joint Node label and degree	0.063223	0.062875	0.06307	0.063533	0.06326	0.063038	**0.062691**

Tabla 5.4: Resultado de las métricas con el experimento del conjunto de gran tamaño.

mentos. Para las otras métricas, los valores son muy similares a los otros experimentos, este resultado ya era esperado. Para la complejidad de los grafos, como no se tiene una métrica que lo evalué se hace de manera visual. Ahora se tiene mayor complejidad en los grafos que se generan, esto se puede ver reflejado en que ahora la cantidad de aristas y nodos que se generan es mayor que en todos los experimentos anteriores. Todo lo anterior se puede atribuir a la mayor cantidad de grafos con los que se entreno el modelo.

En la tabla 5.3 y en la Figura 5.11 se muestran las métricas obtenidas y algunos de los grafos generados por el experimento.

5.8. Comparación de resultados

Es importante comparar los resultados obtenidos en la presente tesis con los que se evalúa el modelo GraphGen [14]. Para esto tomaremos 3 de los experimentos hechos en el articulo del modelo. A continuación se mencionan los experimentos, la cantidad de grafos con los que se entreno el modelo y por ultimo una tabla con los 3 experimentos y el resultado del último experimento presentado en esta sección 5.7

- Lung: Se trata de grafos de componentes químicos. Donde los nodos representan átomos, las aristas representan enlaces, las etiquetas de los nodos denotan tipo

de átomo y las etiquetas de las aristas el tipo de enlace. El modelo para este experimento es entrenado con 24 mil grafos.

- Enzymes: Se trata de un conjunto de grafos de enzimas terciarias que provienen de la base BRENDA. El tamaño del conjunto es de 575 proteínas

- Citeseer: Se trata de un solo grafo de una red de citas, donde cada nodo corresponde a una publicación y las aristas a la cita de un articulo en otro, las etiquetas de los nodos tienen el área de publicación. El grafo tiene 3312 nodos y 4460 aristas

	Lung	Enzymes	Citeseer	Conjunto de gran tamaño
Novelty	**1**	0.98	0.83	0.949153
Uniqueness	**1**	99	0.95	0.989575
Promedio de nodos en generación	35.20	32.44	35.99	**46.716797**
Promedio de aristas en generación	36.66	52.83	41.81	**75.344531**
MMD Degree	**0.009**	0.243	0.089	0.102881
MMD Clustering	**0**	0.198	0.083	0.150493
MMD Orbits	**0**	0.016	0.100	0.128302
MMD NSPDK	**0.035**	0.051	0.020	0.066607
MMD Node label	**0.001**	0.024	0.024	0.067958
MMD Edge label	**0**	-	-	0.001613
MMD Joint Node label and degree	**0.205**	0.249	0.013	0.062691

Tabla 5.5: Comparación del último experimento y los presentados en el articulo de GraphGen

Los resultados de Tabla 5.8 muestran que el modelo entrenado con el conjunto Lung es donde se obtienen los mejores resultados, este experimento es el más parecido con el

que se trabaja en la tesis, ya que tienen el mismo número de grafos de entrenamiento y los grafos tienen información parecida a la de los cristales de perovskitas. Sin embargo la comparación no es del toda justa, ya que los grafos del experimento Lung cumplen con los requisitos del modelo, mientras que el de nosotros tiene una mezcla de grafos que cumple y los que no cumplen, pero que se les aplicó la representación que se propone en la tesis.

A pesar de la diferencia de valores, los que se obtienen en la columna del experimento de la sección 5.7 no son muy dispares, incluso en el experimento de Citesser es mayor en alguna métricas. Además de que se generan grafos tanto conexos como disconexos, cosa que no ocurre en los tres experimentos presentados en el articulo de GraphGen.

5.9. Discusión

Esta discusión se toma a partir de dos cosas, las métricas de la Tabla 5.4 y el hecho de revisar de manera visual y por lo tanto empírica todos los grafos generados a partir de los experimentos.

La primera discusión se centra en la métrica Uniqueness, la cual nos dice la diversidad de los grafos que se generaron, dicha métrica es la que más bajos valores se obtienen en todos los resultados. Ver que dicho valor es tan bajo solo nos habla de que la variedad de grafos que genera el modelo es pobre. Sin embargo, al revisar de manera visual todos lo grafos generados no encontramos con casos como el de la Figura 5.7, en donde se generan grafo idénticos en su estructura pero con etiquetas en los nodos diferentes. Desde el punto de vista de grafos es algo que no se busca. Sin embargo, si como trabajo futuro se quiere usar el modelo para la generación de cristales de perovskitas, seria de interés ver que pasa cuando dos grafos son iguales en su estructura topológica pero con diferentes elementos.

Algo a notar es el experimento 5.6. La métrica Uniqueness mejora, hecho que se lo podemos atribuir a que el modelo ve mas variedad de grafos, pero no los suficientes para generar mayor variedad. Una posible solución seria entrenar a cada experimento con una cantidad mayor de grafos. En el experimento 5.7 se aumenta de manera considerable el numero de grafos para entrenar, la métrica de Uniqueness aumenta de manera considerable ya que el modelo tiene una variedad de grafos para entrenar, sin embargo las otras métricas se mantienen muy similares a los experimentos anteriores.

La segunda discusión es sobre los valores que se obtienen en la métrica MMD y todos los núcleos que se usan. Esta es una de las partes interesantes, dado que los mejores resultados se encuentran en los experimentos donde se usa la modificación propuesta en este trabajo. Como se mencionó en la sección de métricas, una de las mas importantes es la de MMD NSPDK en la cual el mejor resultado es donde el modelo se entrena con los grafos conexos, sin embargo los resultado son muy similares en los demás experimentos. Tener los resultado obtenidos en los experimentos donde los grafos cumplen los requisitos del modelo era esperado. Por eso tener valores similares incluso en algunos mejores en los experimentos donde se úso la modificación propuesta es algo a resaltar.

En el lado de la generación de etiquetas, se obtienen valores satisfactorios algo que se esperaba, sobre todo en la generación de etiquetas de los nodos. La generación de etiquetas en las aristas era de mayor interés, dado que esta generación es la que importa para poder ver la capacidad del modelo para generar grafos disconexos.

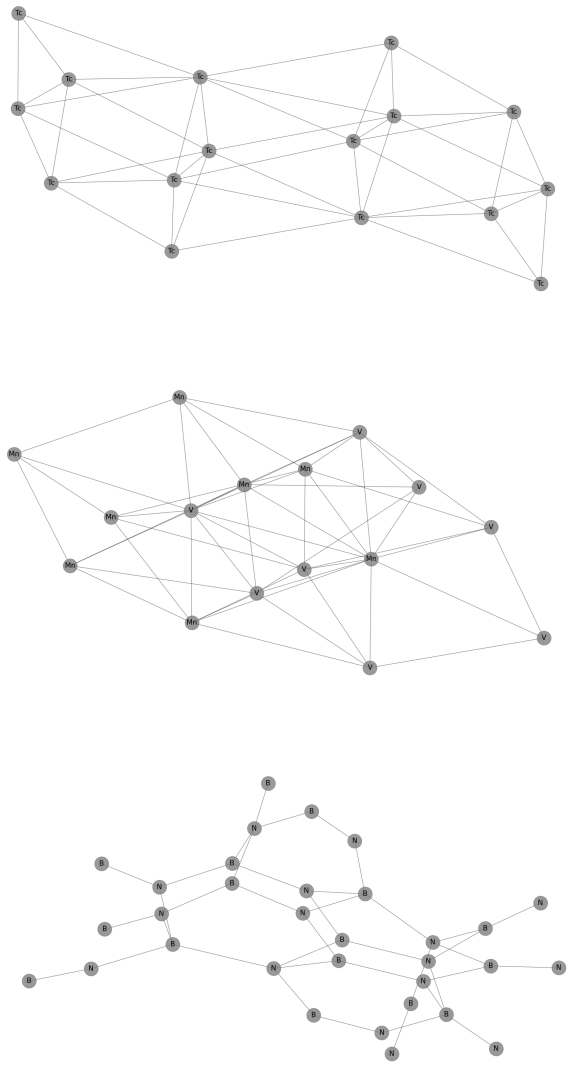

Figura 5.4: Ejemplos de grafos generados por el entrenamiento que solo contiene grafos conexos.

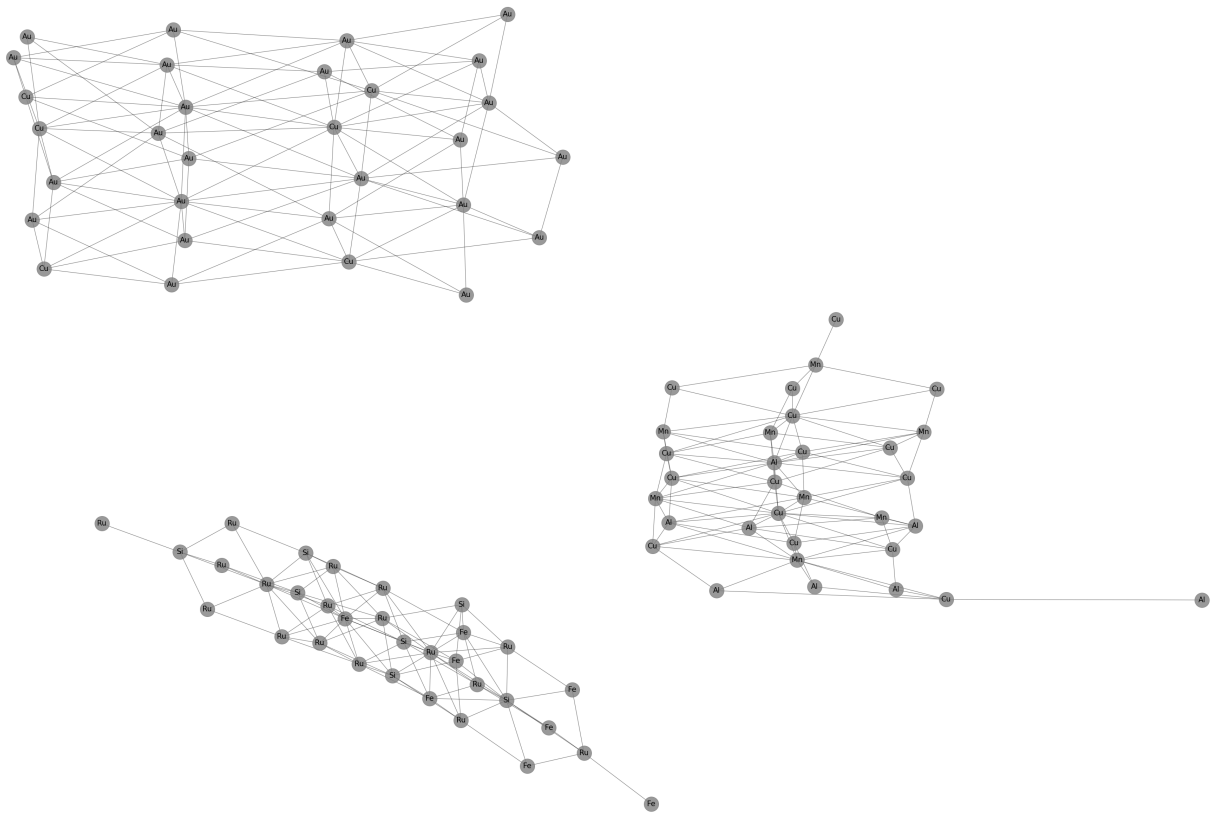

Figura 5.5: Ejemplos de grafos generados por el entrenamiento que solo contiene grafos conexos y restricción de 2 nodos previos en DFS.

Figura 5.6: Ejemplos de grafos generados por el entrenamiento que solo contiene grafos conexos y restricción de 10 nodos previos en DFS.

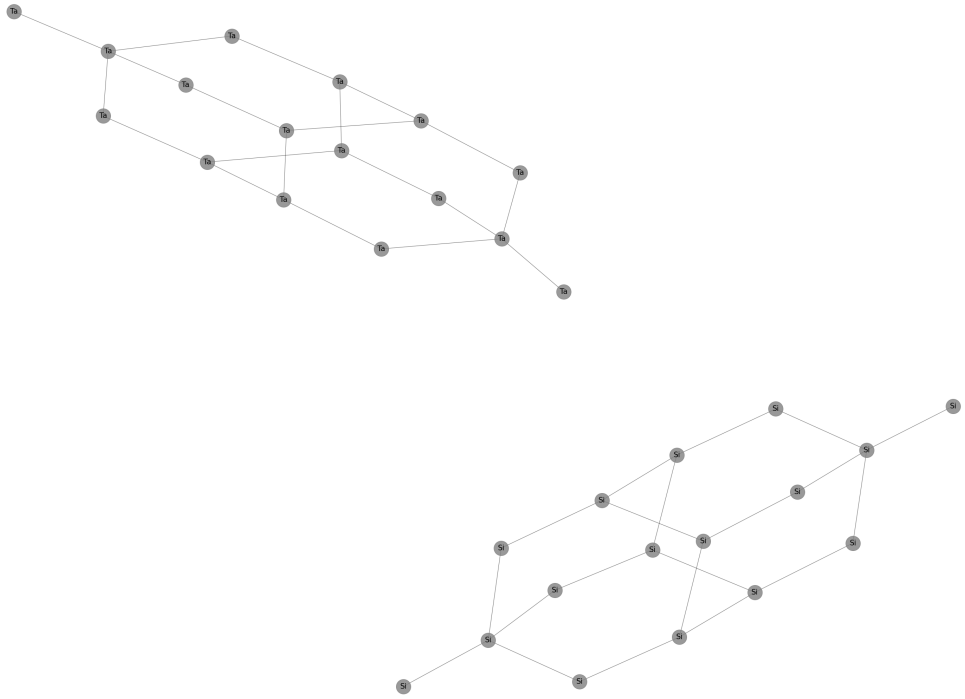

Figura 5.7: Ejemplos de dos grafos similares en donde solo cambian las etiquetas.

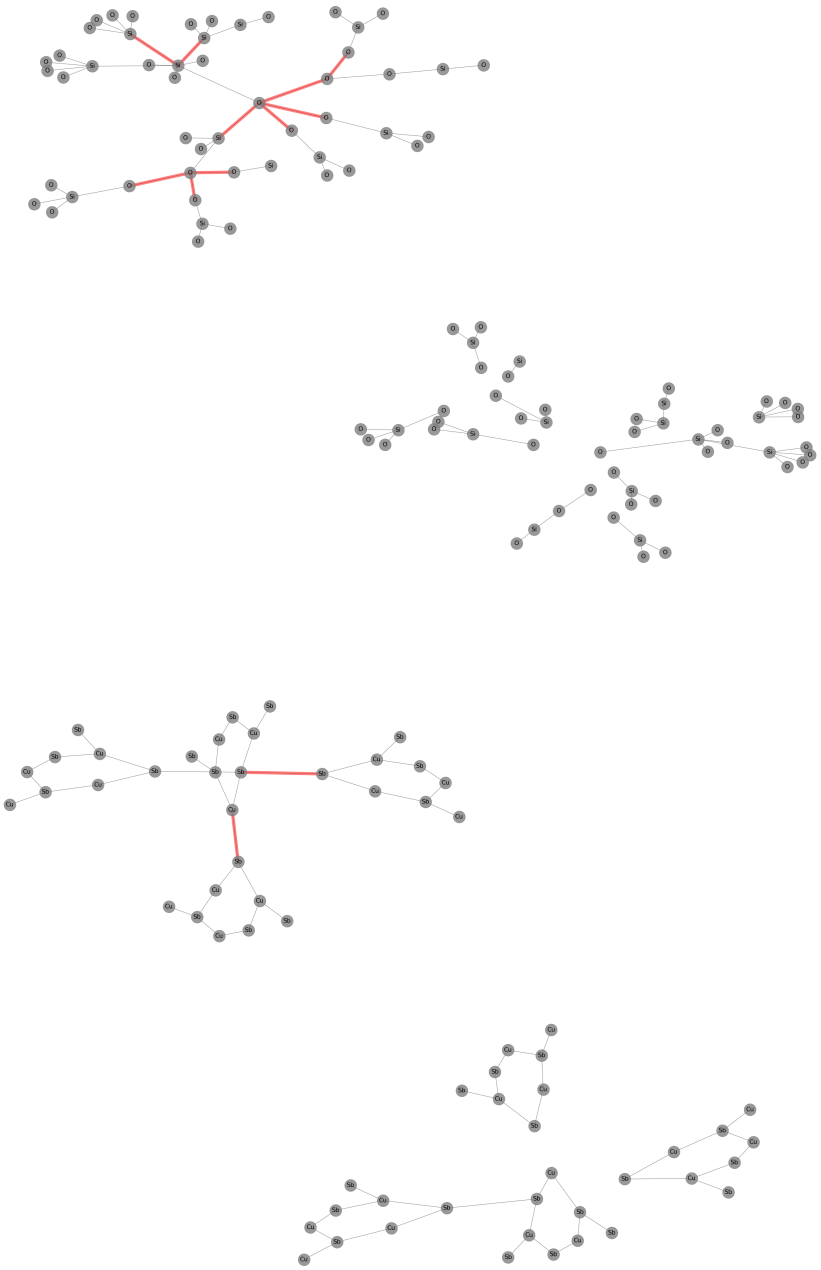

Figura 5.8: Ejemplos de dos grafos generados por el experimento de grafos disconexos sin permutación.

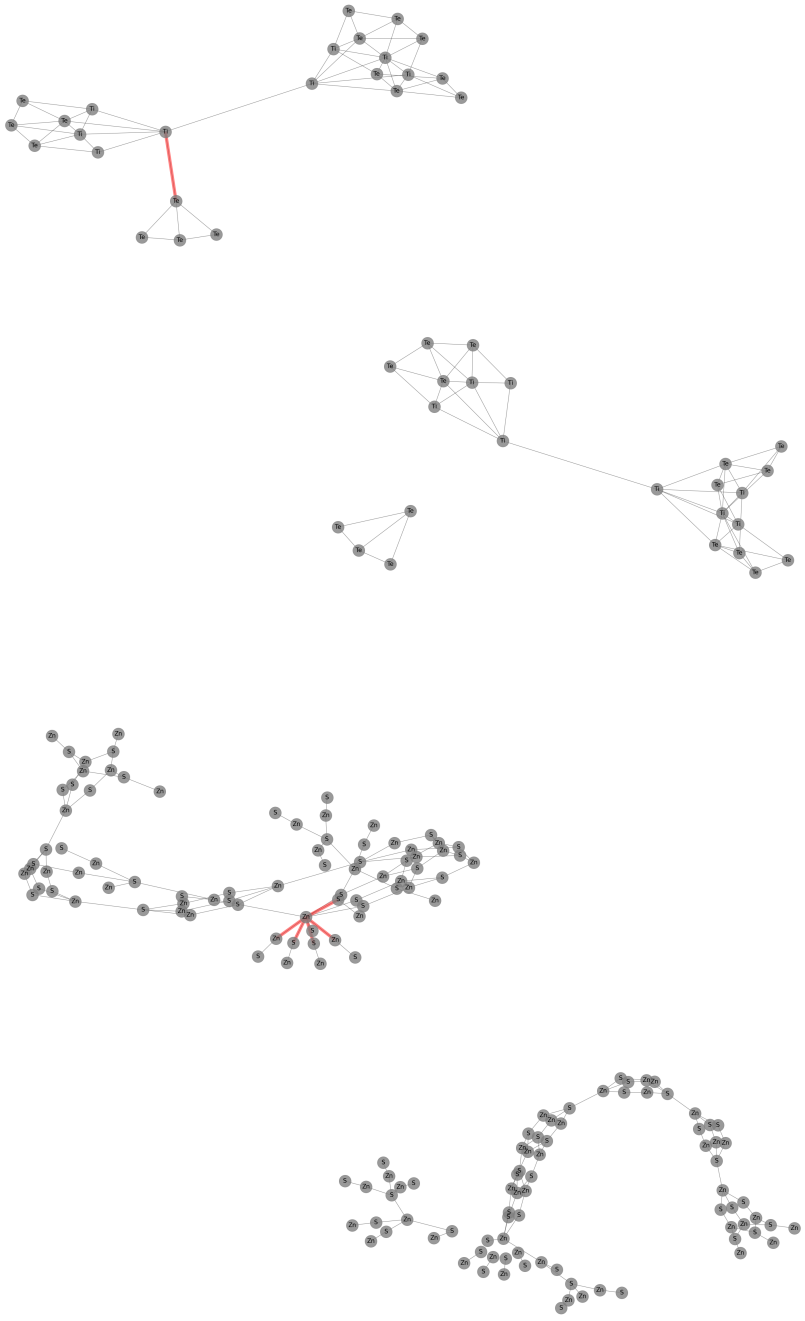

Figura 5.9: Ejemplos de dos grafos generados por el experimento de grafos disconexos con permutación.

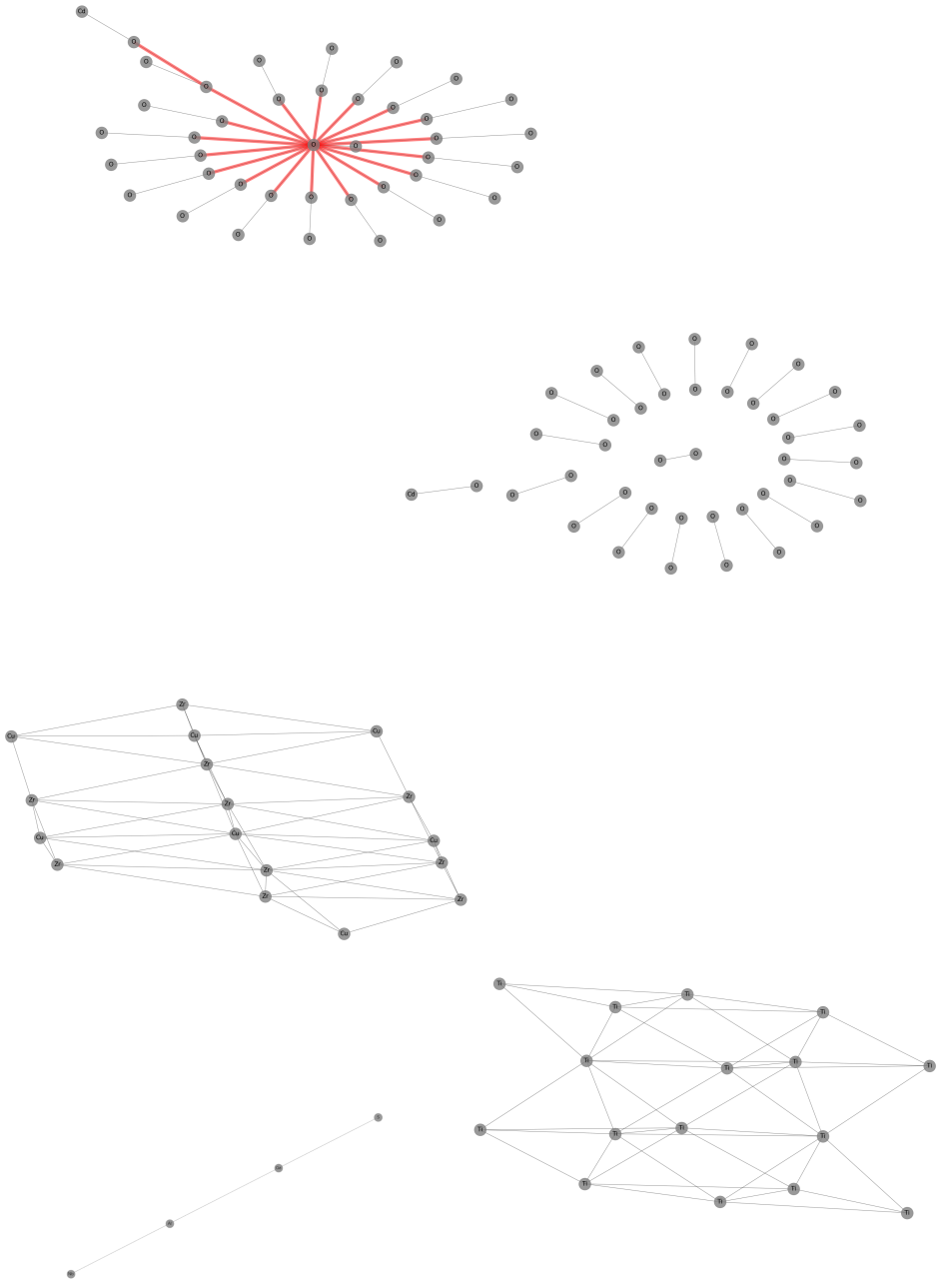

Figura 5.10: Ejemplos de grafos generados por el experimento que contiene los dos conjuntos de grafos.

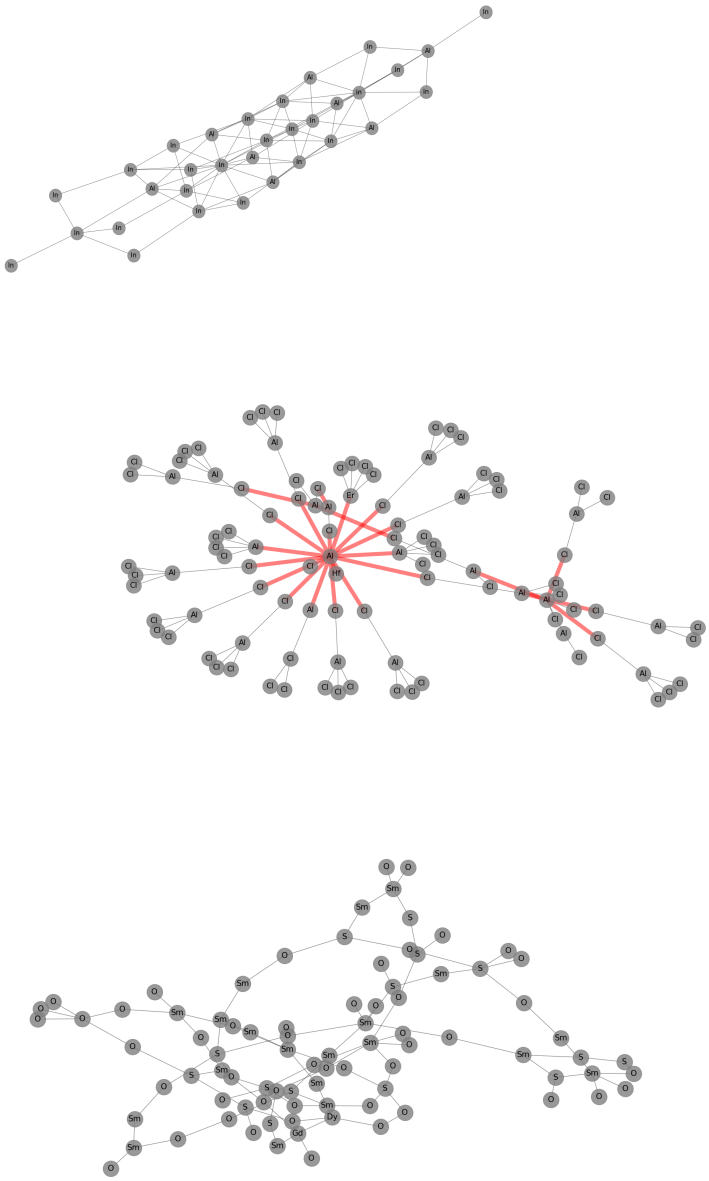

Figura 5.11: Ejemplos de grafos generados por el experimento con el conjunto de gran tamaño.

Capítulo 6

Conclusiones

En el caso de los grafos conexos se obtiene un resultado esperado, sin embargo que la métrica de diversidad sea baja nos habla de que falta una cantidad considerable de grafos para obtener mayor variedad, mientras que para las otras métricas se obtienen resultados aceptables. Para el caso de interés de los grafos disconexos, se mantiene la métrica de diversidad baja. Mientras que la métrica que más llama la atención es la MMD Edge label donde se cuantifica la calidad de las etiquetas generadas para las aristas, dicho valor es bastante bajo, por lo que deja ver que la propuesta para trabajar con grafos disconexos es satisfactoria, incluso cuando el entrenamiento mezcla los dos tipos de grafos. Este hecho refleja que la modificación propuesta trabaja de buena manera junto con el modelo original y que permite al modelo generar cualquier tipo de grafo. En general, la modificación propuesta sobre los grafos para que el modelo GraphGen tenga la capacidad de generar grafos conexos y disconexos es satisfactoria. Las métricas que se usan en este trabajo cumplen con dar certeza de que el modelo es capaz de generar grafos que son similares pero no iguales a los de entrenamiento. Dichas métricas deben ser tomadas como ayuda cuando se quieran generar grafos que cumplan con requisitos específicos de un tarea; ya que por sí solas, sólo nos ayudan a que cumplan características de grafos generales.

6.1. Trabajo futuro

- Probar los grafos desde un punto químico. Medir el desempeño de los grafos generados, pero ahora desde métricas especificas, es decir desde métricas que midan propiedades químicas.

- Grafos similares. Probar qué pasa con los grafos que se generan con la misma estructura pero con diferentes etiquetas en los nodos, pero ahora con métricas químicas. Esto con el fin de ver si el modelo está capturando de alguna manera alguna propiedad química que no se ve reflejada en las métricas usadas.

- Modificar el modelo. Modificar el modelo, pero ahora con el fin de obtener grafos que cumplan requisitos específicos de un grafo de una perovskita.

Bibliografía

[1] Andrew NG. Stochastic Gradient Descent - Universidad de Stanford.

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5:157–66, 1994.

[3] J. Bondy and U. Murty. *Graph Theory With Applications*. 1982.

[4] P. Bongini, M. Bianchini, and F. Scarselli. Molecular graph generation with Graph Neural Networks. *Neurocomputing*, 450:242–252, Aug. 2021. arXiv: 2012.07397.

[5] L. Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8), 1991.

[6] J. Brownlee. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, July 2017.

[7] P. Camacho González. *Sistema de Redes Neuronales para la prediccion de ozono en la CDMX y el area metropolitana*. Tesis Licenciatura, UNAM, Facultad de Ciencias, Ciudad de México, Sept. 2018.

[8] V. S. Chen, P. Varma, R. Krishna, M. Bernstein, C. Re, and L. Fei-Fei. Scene Graph Prediction With Limited Labels. page 11.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms*, volume 42. 2001. Publication Title: The Journal of the Operational Research Society.

[10] M. Cost and i. The discovery of the neuron, Aug. 2006.

[11] F. Costa and K. D. Grave. Fast Neighborhood Subgraph Pairwise Distance Kernel. page 8.

[12] N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecular graphs. *arXiv:1805.11973 [cs, stat]*, May 2018. arXiv: 1805.11973.

[13] S. Fan and B. Huang. Labeled Graph Generative Adversarial Networks. *arXiv:1906.03220 [cs, stat]*, Feb. 2021. arXiv: 1906.03220.

[14] N. Goyal, H. V. Jain, and S. Ranu. GraphGen: A Scalable Approach to Domain-agnostic Labeled Graph Generation. *Proceedings of The Web Conference 2020*, pages 1253–1263, Apr. 2020. arXiv: 2001.08184.

[15] A. Grover, A. Zweig, and S. Ermon. Graphite: Iterative Generative Modeling of Graphs. page 11.

[16] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik. Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. *arXiv:1705.10843 [cs, stat]*, Feb. 2018. arXiv: 1705.10843.

[17] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science*, 4(2):268–276, Feb. 2018.

[18] L. Han, R. C. Wilson, and E. R. Hancock. A Supergraph-based Generative Model. In *2010 20th International Conference on Pattern Recognition*, pages 1566–1569, 2010.

[19] J. Heaton. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618. *Genetic Programming and Evolvable Machines*, 19, 2017.

[20] S. Hochreiter and J. Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, 1997.

[21] W. Jin, R. Barzilay, and T. Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. *arXiv:1802.04364 [cs, stat]*, Mar. 2019. arXiv: 1802.04364.

[22] Kandel, E. R, Schwartz, J. H., Jessell, T. M., and Agud Aparicio. *Principios de neurociencia*. McGraw-Hill Interamericana de España., 4 ediciion edition, 2001.

[23] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: 1412.6980.

[24] J. Koskinen and G. Daraganova. Exponential random graph model fundamentals. *Exponential random graph models for social networks*, pages 49–76, 2013.

[25] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar Variational Autoencoder. *arXiv:1703.01925 [stat]*, Mar. 2017. arXiv: 1703.01925.

[26] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521:436–44, 2015.

[27] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker Graphs: An Approach to Modeling Networks. page 58.

[28] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning Deep Generative Models of Graphs. *arXiv:1803.03324 [cs, stat]*, Mar. 2018. arXiv: 1803.03324.

[29] R. Liao, Y. Li, Y. Song, S. Wang, C. Nash, W. L. Hamilton, D. Duvenaud, R. Urtasun, and R. S. Zemel. Efficient Graph Generation with Graph Recurrent Attention Networks. *arXiv:1910.00760 [cs, stat]*, July 2020. arXiv: 1910.00760.

[30] H. Ma, Y. Bian, Y. Rong, W. Huang, T. Xu, W. Xie, G. Ye, and J. Huang. Multi-View Graph Neural Networks for Molecular Property Prediction. *arXiv:2005.13607 [cs, q-bio, stat]*, June 2020. arXiv: 2005.13607.

[31] T. Ma, J. Chen, and C. Xiao. Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. page 12.

[32] D. Mandic and J. Chambers. Recurrent Neural Networks for Prediction: Learning Algorithms,Architectures and Stability. 2001. ISBN: 0471495174.

[33] M. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3, 01 1995.

[34] P. Mungofa, A. Schumann, and L. Waldo. Chemical crystal identification with deep learning machine vision. *BMC Research Notes*, 11(1):703, Dec. 2018.

[35] A. Nouira, N. Sokolovska, and J.-C. Crivello. CrystalGAN: Learning to Discover Crystallographic Structures with Generative Adversarial Networks. page 9.

[36] S. P. Ong, S. Cholia, A. Jain, M. Brafman, D. Gunter, G. Ceder, and K. A. Persson. The Materials Application Programming Interface (API): A simple, flexible and efficient API for materials data based on REpresentational State Transfer (REST) principles. *Computational Materials Science*, 97:209–215, Feb. 2015. Publisher: Elsevier BV.

[37] G. Pilania, P. V. Balachandran, C. Kim, and T. Lookman. Finding New Perovskite Halides via Machine Learning. *Frontiers in Materials*, 3, Apr. 2016.

[38] P. Raccuglia, K. C. Elbert, P. D. F. Adler, C. Falk, M. B. Wenny, A. Mollo, M. Zeller, S. A. Friedler, J. Schrier, and A. J. Norquist. Machine-learning-assisted materials discovery using failed experiments. *Nature*, 533(7601):73–76, May 2016. Number: 7601 Publisher: Nature Publishing Group.

[39] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

[40] B. Samanta, A. De, G. Jana, P. K. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez. NeVAE: A Deep Generative Model for Molecular Graphs. *arXiv:1802.05283 [physics, stat]*, Sept. 2019. arXiv: 1802.05283.

[41] Sebastian Ruder. An overview of gradient descent optimization algorithms, Jan. 2016.

[42] M. Simonovsky and N. Komodakis. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, volume 11139, pages 412–422. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science.

[43] F. v. Veen. The Neural Network Zoo, Sept. 2016.

[44] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model, Jan. 1988.

[45] D. West. *Introduction to Graph Theory (2nd Edition)*. 2000.

[46] T. Xie and J. C. Grossman. Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties. *Physical Review Letters*, 120(14):145301, Apr. 2018. arXiv: 1710.10324.

[47] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. *arXiv:1806.02473 [cs, stat]*, Feb. 2019. arXiv: 1806.02473.

[48] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. *arXiv:1802.08773 [cs]*, June 2018. arXiv: 1802.08773.

[49] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph Neural Networks: A Review of Methods and Applications. *arXiv:1812.08434 [cs, stat]*, July 2019. arXiv: 1812.08434.