UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

Redes neuronales en grafos para predecir propiedades de distribuciones atómicas

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Matemático

PRESENTA:

Diego Peña Angeles

TUTOR

Dr. J. Guadalupe Pérez Ramírez

CD. MX. 2023

Resumen

En este trabajo se describe un modelo de redes neuronales basado en grafos para predecir propiedades físicas de distribuciones atómicas con el objetivo de analizar su precisión y confiabilidad. El modelo que se ocupa de base es el propuesto por Txie Chan y Joffrey Grossmann (2020) el cual emplea los grafos para obtener propiedades del entorno local de la distribución debido a la posición no estructurada de los átomos. A partir del modelo se implementa un programa (escrito en `Python` y `Pytorch`), posteriormente el modelo se entrena con una base de datos de materiales. Una vez entrenado el modelo se realizan predicciones de la propiedad de brecha de banda de energía en estructuras cristalinas (distribuciones atómicas periódicas) y finalmente se analiza la eficacia del modelo. El código se puede encontrar en un repositorio de github.

Índice general

Introducción

En la última década el aprendizaje automático, y especialmente los métodos de aprendizaje profundo, han mostrado resultados notables en una amplia gama de campos científicos, lo que ha dado lugar a numerosas revelaciones, desde el reconocimiento de imágenes [1] hasta averiguar si una molécula es adecuada para un antibiótico [2]. Unas de las pocas limitaciones de estos métodos es que los datos de entrada a los modelos se encuentran en el espacio euclidiano, sin embargo existen muchas tareas en las que los datos a tratar no se encuentra en dicho espacio. Para poder trabajar con datos más complejos se está explorando el aprendizaje profundo geométrico [3], un término que engloba las nuevas técnicas utilizadas para generalizar los modelos de aprendizaje profundo (estructurados) a dominios no euclidianos, como lo son los grafos.

Los grafos son un tipo de estructura de datos que modela un conjunto de objetos (nodos) y sus relaciones (aristas). Recientemente, las investigaciones sobre aprendizaje profundo en grafos (subcampo del aprendizaje automático geométrico [3]) han recibido cada vez más atención debido a su gran poder expresivo, es decir, que pueden utilizarse para denotar un gran número de sistemas en diversas áreas, incluyendo las ciencias sociales (redes sociales [4]), la física (sistemas físicos [5]), la biología (redes de interacción proteína-proteína ([6]), la medicina (conectividad cerebral [7]) y muchos otros ámbitos de investigación.

Particularmente las técnicas de aprendizaje profundo en grafos ofrecen una nueva oportunidad para reducir significativamente los costos computacionales y acelerar el ritmo de descubrimiento y diseño de materiales. Se han adoptado modelos de aprendizaje profundo en grafos para predecir un amplio conjunto de propiedades de los materiales, como la estructura cristalina [8], la estabilidad de las fases [9], las estructuras electrónicas (como las brechas de banda de energía) [10], energías de atomización [11], así como potenciales de interacción entre átomos para simulaciones de dinámica molecular [12]. Para facilitar el aprendizaje efectivo de moléculas y sistemas de estado sólido, se han desarrollado recientemente representaciones de estructuras atómicas de enlace, como la matriz de Coulomb [13], la bolsa de enlaces [14] y la representación basada en el tensor de muchos cuerpos [15].

Es de ahí que ha surgido el concepto de grafo como un recolector y comunicador de información-material que sintetiza a las distribuciones atómicas. Aprovechando las conexiones de enlace entre los átomos y las interacciones de muchas partículas, los grafos

son una representación natural tanto para las moléculas como para los cristales [16]. Desde este punto de vista, la química cuántica y la física del estado sólido se fusionan en el paradigma del aprendizaje profundo.

Este trabajo se centra en el desarrollo reciente de aprendizaje profundo basado en grafos y sus aplicaciones tanto para moléculas como para cristales. Se presenta la historia del desarrollo de representaciones basadas en grafos para moléculas y cristales. Se revisan los procesos esenciales de aprendizaje definidos por el llamado paso de mensajes neuronales. Se presenta el desafío actual y las perspectivas futuras de este campo emergente en la encrucijada de la ciencia de los materiales, la física, la química y la matemática.

En el primer capítulo de este trabajo se aborda la teoría matemática detrás de los grafos y se explora una familia de algoritmos de aprendizaje automático que opera sobre grafos. En el siguiente capítulo se describen los algoritmos capaces de representar a moléculas y cristales con el objetivo de generar los datos de entrenamiento para diferentes modelos de redes neuronales en grafos. En la siguiente parte se analiza un modelo de redes neuronales en grafos capaz de predecir la brecha de banda de energía de los cristales. Por último se presenta el código escrito en `Python` donde se implementa el modelo.

Capítulo 1

Redes neuronales en grafos

En este capítulo se aborda la teoría matemática que subyace a los grafos y se explora una familia de algoritmos de aprendizaje automático que opera sobre grafos.

1.1. Grafos

Los grafos ofrecen una base matemática en la que podemos basarnos para analizar, comprender y aprender de los sistemas del mundo real. Formalmente, un grafo es un objeto matemático definido por el par $G = (V, E)$, donde V representa un conjunto finito de nodos y E un conjunto finito de aristas entre estos nodos. Se denota una arista que va del nodo $i \in V = \{1, 2, ..., m\}$ al nodo $j \in V = \{1, 2, ..., m\}$ como $e_{ij} \in E$. Las aristas pueden ser dirigidas, lo que significa que dependiendo de cuál de los nodos sea el de origen y el de destino respectivamente podría indicar diferentes características ($e_{ij} \in E$ pero pudiera ser que $e_{ji} \notin E$), o la arista podría ser no dirigida donde la arista no depende de dónde empieza y termina ($e_{ij} \in E \Leftrightarrow e_{ji} \in E$). Además es posible que los grafos permitan más de una arista entre cada par de nodos. ($e_{ij_k} \in E$ con $k \in \mathbb{N}$ donde k nos indica el numero de aristas que hay entre los nodos i y j). A este tipo de grafos se les conoce como multigrafos por obvias razones. Una forma conveniente de representar los grafos es mediante una matriz de adyacencia A. La matriz de adyacencia es una matriz cuadrada del tamaño del número de nodos del grafo al cuadrado $\left(A \in \mathbb{R}^{|V|x|V|}\right)$. Los elementos no nulos de la matriz indican una conexión entre nodos. En el caso más simple, una conexión se muestra como uno y ninguna conexión como cero ($e_{ij} \in E \Rightarrow A_{ij} = 1$ pero si $e_{ij} \notin E \Rightarrow A_{ij} = 0$).

Dependiendo de los datos a representar mediante grafos es posible añadir información de características asociadas a los nodos y las aristas. Por ejemplo, en una molécula, las propiedades de cada átomo (nodo) y el tipo de enlace entre ellos (aristas). La información almacenada en los grafos suele adoptar la forma de vectores de características $\vec{x}_i$. Cada nodo tiene un vector de características $\vec{x}_i$ y su dimensión es $\mathbb{R}^d$ donde d es el número de características de cada nodo. Para representar los vectores de características de todos los nodos se presenta cada característica sobre una columna dando así la

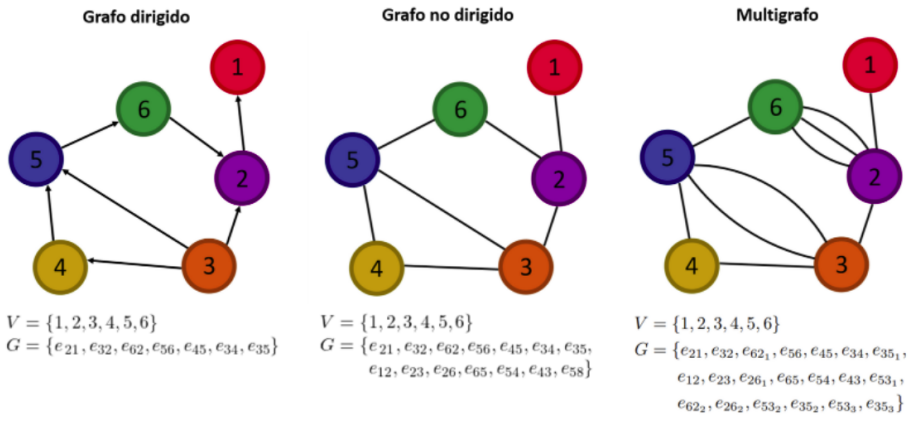

Figura 1.1: Tipos de grafos

matriz de características de nodos $X \in \mathbb{R}^{|V| \times d}$, así el elemento X_{ij} será la j-ésima característica del nodo i.

De igual forma cada arista tiene un vector de características $\vec{y}_{ij}$ y su dimensión es $\mathbb{R}^p$ donde p es el número de características de cada arista. Para representar los vectores de características de todas las aristas se presentan sobre la misma dimensión dando así el tensor de características de nodos $M \in \mathbb{R}^{|V| \times |V| \times p}$, así el elemento M_{ijl} será la l-ésima característica de la arista e_{ij}.

1.2. Paso de mensajes neuronales

La inteligencia artificial es la capacidad de un sistema informático de imitar funciones cognitivas humanas como el aprendizaje y la resolución de problemas [17]. Al campo dentro de la inteligencia artificial enfocado en estudiar y desarrollar los algoritmos y modelos estadísticos que dichos sistemas utilizan para realizar una tarea específica sin ser programados explícitamente se le conoce como aprendizaje automático [18]. Las técnicas de aprendizaje automático pueden clasificarse a grandes rasgos en tres categorías principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

El aprendizaje supervisado para problemas de clasficiación [18] es la tarea de aprender una función que modele la relación entre un conjunto de cantidades observables (características) y otro conjunto de variables relacionadas con ellas (etiquetas). El modelo infiere la función mediante el algoritmo de aprendizaje, el cual requiere datos de entrenamiento que consisten en muestras de características con sus respectivas etiquetas. La función permite predecir las etiquetas a partir de características en datos que no se utilizaron durante el entrenamiento. Una de las técnicas principales del aprendizaje supervisado es la de las redes neuronales artificiales [19]. Las redes neuronales artificiales son modelos matemáticos cuya base se encuentra en el modelo de regresión lineal y su invención se le atribuye a Rosenblatt [20] con su trabajo sobre el perceptrón multicapa. El término aprendizaje profundo se refiere a la práctica de utilizar redes neuronales con muchas capas.

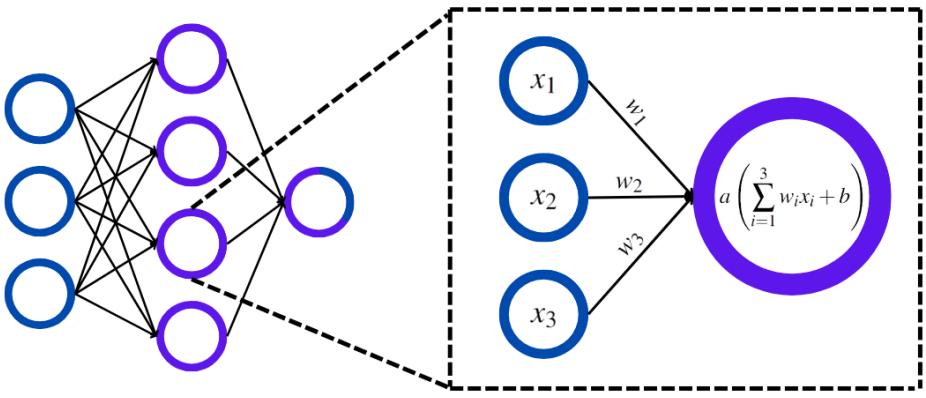

Figura 1.2: Red neuronal artificial.

La extensión de los modelos neuronales profundos a dominios sobre grafos se logra a partir del mismo paradigma que las convoluciones sobre imágenes (redes neuronales convolucionales [21]), para poder definir convoluciones en grafos se define un nuevo marco llamado paso de mensajes neuronales.

De manera general, un grafo representa un conjunto de objetos en el que algunos de ellos tienen una relación. Los objetos se representan como nodos y las relaciones como aristas entre esos nodos. La idea de las redes neuronales de grafos es explotar la estructura relacional de los datos y utilizarla para el desarrollo de un algoritmo de aprendizaje. Como cualquier otro algoritmo de aprendizaje profundo el objetivo de las redes neuronales de grafos es aprender un mapeo (función) de un grafo a una variable objetivo (etiqueta) aprovechando la relación (o interacción) entre los objetos del grafo. Como parte del algoritmo de aprendizaje, el modelo debe aprender cómo interactúan estos objetos. Las primeras redes neuronales de grafos fueron inventadas por Gori, Monfardini y Scarselli [22]. Para describir las diferentes arquitecturas se utiliza el marco de paso de mensajes neuronales [23].

Por simplicidad se describe el paso de mensajes neuronales en grafos no dirigidos (es trivial extender el formalismo a los multigrafos y grafos dirigidos). Dado un grafo $G = (V,E)$, cada nodo $u \in V$ inicia con un vector de características asociado $\vec{x}_u$ y durante el paso de mensajes por cada vector de características de nodo se aplican una serie de funciones que toman como entrada a los vectores de características de los nodos vecinos$\{\vec{x}_v | v$ es vecino de u$\}$ junto con los vectores de características de sus correspondientes aristas $\{\vec{y}_{uv} |$v es vecino de u$\}$ para devolver un nuevo vector de características del nodo u. Esta actualización de $\vec{x}_u$ se repite en cada paso de mensajes por lo que se definen los vectores $\mathbf{h}_u^{(t)}$ y $\mathbf{e}_{uv}^{(t)}$ del nodo y arista respectivamente para denotar los nuevos vectores de características en cada paso de mensajes t (donde $\mathbf{h}_u^{(0)} = \vec{x}_u$ y $\mathbf{e}_{uv}^{(0)} = \vec{y}_{uv}$), dichos vectores son llamados estados ocultos. El estado oculto del nodo se irá actualizando mediante el intercambio de información de la vecindad local de los nodos, este intercambio se formula como:

$$\mathbf{m}_u^{(t+1)} = \square_{v \in \mathcal{N}_{(u)}} \gamma\left(\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}, \mathbf{e}_{uv}^{(t)}\right), \tag{1.1}$$

$$\mathbf{h}_u^{(t+1)} = \phi\left(\mathbf{h}_u^{(t)}, \mathbf{m}_u^{(t+1)}\right), \tag{1.2}$$

donde γ y ϕ son funciones diferenciables y $\square$ es una función invariante a la permutación (su salida no cambia al reordenar los elementos de entrada) como lo es la suma o la media. $\mathcal{N}_{(u)}$ denota la vecindad local de u es decir, los nodos que tiene una arista con u. En cada paso de mensajes t, la función γ toma como entrada el estado oculto del vecino correspondiente $\mathbf{h}_v^{(t)}$, el del nodo central $\mathbf{h}_u^{(t)}$ y el de la arista $\mathbf{e}_{uv}^{(t)}$ que los une para generar un "mensaje"(un vector con información del vecino del nodo central), luego con la función $\square$ se recopilan los mensajes de todos los vecinos en un sólo vector, $\mathbf{m}_u^{(t+1)}$. Finalmente la función ϕ combina entonces el mensaje final $\mathbf{m}_u^{(t+1)}$ y el estado oculto anterior $\mathbf{h}_u^{(t)}$ del nodo u para generar el estado oculto actualizado $\mathbf{h}_u^{(t+1)}$. Los estados ocultos iniciales en $t = 0$ se establecen con las características de entrada para todos los nodos, es decir, $\mathbf{h}_u^0 = \vec{x}_u \ \forall u \in V$, así como con las de las aristas $\mathbf{e}_{uv}^0 = \vec{y}_{uv} \ \forall e \in E$. Después de ejecutar T pasos de mensajes, se puede utilizar la salida de la capa final (donde capa se denota a cada paso de mensajes t) para definir los estados finales para cada nodo, es decir,

$$\mathbf{z}_u = \mathbf{h}_u^{(T)}, \forall u \in V \tag{1.3}$$

Dado que los estados ocultos de las aristas no se están actualizando, se tiene que $\mathbf{e}_{uv}^{(t)} = \vec{y}_{uv}\ \forall t \in \{1,...,T\}$ sin embargo en algunos modelos las aristas se actualizan mediante una función diferenciable ψ que puede depender del estado oculto anterior de la arista y los de los nodos que la unen:

$$\mathbf{e}_{uv}^{(t+1)} = \psi\left(\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}, \mathbf{e}_{uv}^{(t)}\right) \tag{1.4}$$

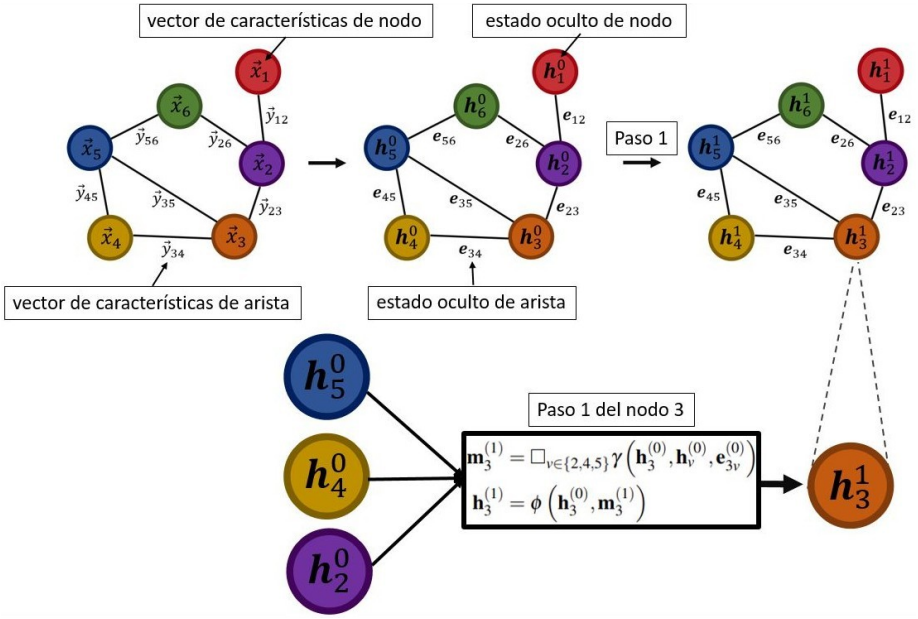

Figura 1.3: Visión general de cómo un solo nodo agrega mensajes de su vecindario local.

La intuición básica que subyace al paso de mensajes es sencilla (figura 1.3): en cada paso, cada nodo agrega información de su vecindario local, y a medida que estos pasos progresan, las características del nodo contienen cada vez más información de los tramos más lejanos del grafo. De forma más precisa, después del primer paso $(t = 1)$, las características del nodo contienen información de su vecindad de primeros vecinos, es decir, de los nodos a los que puede llegar por un camino de 1 arista, después del segundo paso $(t = 2)$ cada nodo contiene información de su vecindad de segundos vecinos; y en general, después de t pasos cada nodo contiene información sobre su vecindad de t vecinos.

1.3. Arquitecturas de las redes

Hasta ahora, se ha discutido el marco de las redes neuronales en grafos de una manera relativamente abstracta como una serie de pasos de mensajes utilizando las funciones $\square$, γ y ψ (ecuación 2.1 y 2.2), sin embargo se plantea la cuestión de cómo podría definirse estas funciones para recoger la mejor información de los nodos vecinos. En los casos más sencillos, podría ser simplemente una suma de la información de la vecindad local (arquitectura básica de una red neuronal en grafos):

$$\mathbf{h}_u^{(t+1)} = g\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in \mathcal{N}_{(u)}} \mathbf{h}_v^{(t)} + \mathbf{b}^{(t)}\right) \tag{1.5}$$

donde $\mathbf{W}_{self}^{(t)}, \mathbf{W}_{neigh}^{(t)} \in \mathbb{R}^{d^{(t+1)} \times d^{(t)}}$ son matrices de parámetros entrenables y g denota una función de activación no lineal (por ejemplo, un tanh o ReLU). El término de sesgo $\mathbf{b}^{(t)} \in \mathbb{R}^{d^{(t)}}$ suele omitirse para simplificar la notación, pero incluirlo puede ser importante para conseguir un buen rendimiento. En esta ecuación se utilizan superíndices para denotar el paso en el que se encuentra cada estado. El paso de mensajes en el marco básico de la red neuronal en grafos es análogo al de un perceptrón multicapa ya que se basa en operaciones lineales seguidas de una función de activación no lineal. Primero se suman los mensajes entrantes de los vecinos y se multiplica por la matriz de parámetros entrenables $\mathbf{W}_{neigh}^{(t)}$; luego, se agrupa la información de los vecinos con el estado oculto anterior del nodo multiplicado por $\mathbf{W}_{self}^{(t)}$ utilizando una combinación lineal; y finalmente, se aplica una función de activación no lineal. Se define de forma equivalente el paso de mensajes básico mediante las funciones $\square$, γ y ψ:

$$\mathbf{m}_u^{(t+1)} = \sum_{v \in \mathcal{N}_{(u)}} \mathbf{h}_v \tag{1.6}$$

$$\mathbf{h}_u^{(t+1)} = g\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u + \mathbf{W}_{neigh}^{(t)}\mathbf{m}_u^{(t+1)} + \mathbf{b}^{(t)}\right) \tag{1.7}$$

donde

$$\square_{v \in \mathcal{N}_{(u)}} = \sum_{v \in \mathcal{N}_{(u)}} \tag{1.8}$$

$$\gamma\left(\mathbf{h}_v^{(t)}, \mathbf{h}_u^{(t)}, \mathbf{e}_{uv}^{(t)}\right) = \mathbf{h}_v^{(t)} \tag{1.9}$$

$$\phi\left(\mathbf{h}_u^{(t)}, \mathbf{m}_u^{(t+1)}\right) = g\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u + \mathbf{W}_{neigh}^{(t)}\mathbf{m}_u^{(t+1)} + \mathbf{b}^{(t)}\right) \tag{1.10}$$

El modelo básico de paso de mensajes de una red neuronal en grafos esbozado en la ecuación (2.7) puede alcanzar un gran rendimiento, y su capacidad teórica es bien

conocida. Sin embargo, al igual que un simple perceptrón multicapa o una red neuronal recurrente, el modelo puede mejorarse y generalizarse de muchas maneras. La operación más básica de agregación de información de vecindad local (ecuación (2.8)) simplemente toma la suma de los estados ocultos de los vecinos. Un problema de este enfoque es que puede llegar a ser inestable y muy sensible a los grados de los nodos, es decir a el número de vecinos que tenga el nodo. Para ejemplificar lo anterior dado un nodo u con 100 vecinos más que el nodo w entonces:

$$\| \sum_{v \in \mathcal{N}_{(u)}} \mathbf{h}_v^{(t)} \| \gg \| \sum_{v \in \mathcal{N}_{(w)}} \mathbf{h}_v^{(t)} \|$$

(para cualquier norma vectorial razonable $\|\cdot\|$). Esta drástica diferencia de magnitud puede conducir a inestabilidades numéricas y dificultades para la optimización. Una solución a este problema es simplemente normalizar la función $\square$ basada en los grados de los nodos implicados. El enfoque más sencillo es simplemente tomar una media en lugar de sumar:

$$\mathbf{h}_u^{(t+1)} = g\left(\mathbf{W} \sum_{v \in \mathcal{N}_{(u)} \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{\left|\mathcal{N}_{(u)}\right| \left|\mathcal{N}_{(v)}\right|}} \right), \tag{1.11}$$

donde $\mathbf{W}$ es una matriz de pesos entrenable y $\left|\mathcal{N}_{(u)}\right|$ denota el número de nodos vecinos del nodo u. En (2.11) se puede observar ahora que la información agregada está normalizada por el número de nodos vecinos que tiene cada nodo respectivo. Este tipo de normalización utilizada fue introducido por primera vez por Tipf y Welling en 2016 [24] y recibe el nombre de Red Convolucional sobre Grafos. Una observación sobre esta capa de convolución es que es isotrópica, es decir que la información de todos los nodos vecinos es igualmente importante, lo que podría reducir el rendimiento, ya que a veces la información debería ser más importante de un nodo vecino específico. Para hacer frente a esto, se puede utilizar una capa más elaborada, la GAT (*Graph Attention Layer*) [25]. La capa GAT, en cambio utiliza el mecanismo de atención que da lugar a una retroalimentación selectiva de los nodos vecinos.

$$\mathbf{h}_u^{(t+1)} = \sum_{v \in \mathcal{N}_{(u)}} \alpha_{uv} \mathbf{h}_v^{(t)}, \tag{1.12}$$

donde α_{uv} denota la atención en el vecino $v \in \mathcal{N}_{(u)}$ cuando se está agregando información en el nodo u. En el documento original de GAT, los pesos de atención se definen como

$$\alpha_{uv} = \frac{\exp\left(\sigma\left(\mathbf{a}^T \left[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v\right]\right)\right)}{\sum_{w \in \mathcal{N}_{(u)}} \exp\left(g\left(\mathbf{a}^T \left[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_w\right]\right)\right)} \tag{1.13}$$

Donde $\mathbf{a}$ es un vector de atención entrenable, $\oplus$ es un operador de concatenación y g es una función de activación no lineal. A partir de (2.13) se puede observar que cada

exponencial se divide con una suma de exponenciales, este método recibe el nombre de *softmax* y funciona como un factor de normalización en este caso. Añadir atención es una estrategia útil para aumentar la capacidad de representación de un modelo de red neuronal en grafos especialmente en los casos en los que se tiene conocimiento previo que indica que algunos vecinos pueden ser más informativos que otros.

A pesar de toda la variedad que existe de arquitecturas de redes neuronales en grafos, hay dificultad para crear redes profundas ya que después de una cierta cantidad de pasos de mensajes neuronales los estados ocultos de cada nodo se vuelven muy similares ya que obtienen información prácticamente de todos los nodos del grafo, esto se denomina *over-smoothing* [23]. No existe una capa específica que se ocupe de esto directamente, pero en los últimos años se han propuesto arquitecturas de las redes en las que en lugar de basarse únicamente en la salida de la capa de convolución final, se realiza una concatenación de los resultados de las capas anteriores. De esta manera se podría utilizar una mayor profundidad pero también manteniendo el comportamiento de los estados ocultos anteriores del nodo.

1.4. Capas de agrupación

Las capas de agrupación son un mapeo de los estados ocultos finales $\mathbf{z}_u$ a un sólo vector que permita recabar la información estructural del grafo tras haber finalizado las capas de paso de mensajes neuronales.

$$\mathscr{P}\left(\{\mathbf{z}_u \; \forall u \in V\}\right) \tag{1.14}$$

donde $\mathscr{P}$ es una función invariante a la permutación. Las capas de agrupación son esenciales, no sólo porque reducen significativamente la dimensión de los datos sino también para recoger las características clave del sistema representado.

La versión estándar de capa de agrupación en grafos se llama agrupación global donde las características de los nodos en cada grafo son agregados:

$$\mathscr{P}\left(\{\mathbf{z}_u \; \forall u \in V\}\right) = \frac{1}{|V|}\sum_{u \in V} \mathbf{z}_u \tag{1.15}$$

Sin embargo, la agrupación global no siempre es la más eficaz y tiene un pobre rendimiento con grafos de gran tamaño, ya que la disminución sustancial del tamaño de la muestra también hace que se diluyan las características importantes. Existen otros métodos de agrupación más eficaces, como la llamada capa de agrupación *set2set* propuesta por Vinyals, Bengio y Kudlur [26]. Se trata de una red neuronal recurrente, en concreto una red de memoria a corto plazo (LSTM) [27] que evoluciona un estado oculto global

$$\mathbf{q}^{(t+1)} = \mathbf{LSTM}\left(\mathbf{q}^{(t)}, \mathbf{r}^{(t)}\right) \tag{1.16}$$

$$\alpha_u^{t+1} = f\left(\mathbf{z}_u, \mathbf{q}^{(t+1)}\right) \tag{1.17}$$

$$\beta_u^{t+1} = \frac{\exp\left(\alpha_u^{t+1}\right)}{\sum_{w \in V} \alpha_w^{t+1}} \tag{1.18}$$

$$\mathbf{r}^{(t+1)} = \sum_{w \in V} \beta_u^{t+1} \mathbf{z}_w \tag{1.19}$$

Donde $\mathbf{q}^{(t+1)}$ representa un vector de atención en cada paso t. En la ecuación (2.17), el vector de atención se utiliza para calcular un valor escalar de atención sobre cada nodo utilizando una función de atención f (por ejemplo, un producto punto), y este escalar de atención se normaliza en la ecuación (2.18). Por último, en la ecuación (2.19) se calcula una suma ponderada de los estados ocultos de los nodos basándose en los pesos de la atención, y esta suma ponderada se utiliza para actualizar el vector de atención utilizando una actualización LSTM (Ecuación (2.16)). En general, los vectores $\mathbf{q}^{(t+1)}$ y $\mathbf{r}^{(t+1)}$ se inicializan con valores totalmente nulos, y después de iterar las ecuaciones (2.16)-(2.19) durante T pasos, se calcula un estado oculto para el grafo completo concatenando los vectores resultantes en cada paso $\mathbf{r}^{(t)}$.

$$\mathscr{P}\left(\{\mathbf{z}_u \; \forall u \in V\}\right) = \mathbf{r}_1 \oplus \mathbf{r}_2 \oplus ... \oplus \mathbf{r}_T \tag{1.20}$$

Este enfoque representa una arquitectura sofisticada para la capa de agrupación basada en la atención sobre un conjunto, y se ha convertido en un método de agrupación popular en muchas tareas de clasificación.

El poder expresivo de las redes neuronales en grafos permite predecir alguna variable objetivo a nivel de nodo, de enlace y de grafo. En este trabajo se ocupa únicamente la tarea de predecir a nivel de grafo, es por ello que se introducen las capas de agrupación.

1.5. Entrenamiento

Cualquier modelo de redes neuronales en grafos que predice alguna variable a nivel de grafo se puede ver como una función f de la siguiente forma:

$$f(G,W,b) = y \tag{1.21}$$

donde f toma como entradas al grafo G y a los parámetros W,b para devolver un valor y. El objetivo del aprendizaje supervisado es que a partir de una colección de muestras la red neuronal en grafos se entrene. Cada muestra está compuesta por un grafo G y su correspondiente etiqueta y, donde el grafo describe las características de un sistema y la etiqueta es una variable que depende de esas características. Ya que se tiene un conjunto de datos, se toma una de las muestras y entonces se pasa el grafo $G^{(i)}$ por la red neuronal para obtener una predicción $\hat{y}^{(i)}$.

El entrenamiento consiste en un algoritmo de optimización que dé como resultado el conjunto de parámetros W,b con los cuales la predicción $\hat{y}^{(i)}$ sea lo más parecida a la etiqueta $y^{(i)}$. Este algoritmo de optimización es un proceso iterativo en el que cada iteración se divide en tres etapas: propagación hacia delante, propagación hacia atrás y actualización de los parámetros.

En la propagación hacia delante se selecciona una muestra $G^{(i)},y^{(i)}$ y se transmite a $G^{(i)}$ desde la capa de entrada de la red neuronal hasta la capa de salida para obtener la predicción $\hat{y}^{(i)}$. Una vez hecho esto, lo siguiente es medir de alguna forma qué tan diferente es la predicción $\hat{y}^{(i)}$ de la etiqueta $y^{(i)}$. Por lo tanto lo que se tiene que hacer es elegir una función de pérdida, que es una métrica que mida la distancia que hay entre esas dos variables. Otra manera de interpretar a la función de pérdida es que ésta es una función que mide la probabilidad de que $\hat{y}^{(i)}$ sea igual a $y^{(i)}$. Una de las funciones de pérdida más utilizadas es la del cuadrado de la distancia entre la etiqueta y la predicción:

$$L_i = \left(\hat{y}^{(i)} - y^{(i)}\right)^2 \tag{1.22}$$

La propagación hacia delante y el cálculo de la función de pérdida es un proceso que se debe llevar a cabo sobre cada una de las muestras del conjunto de datos. Una vez hecho esto, el siguiente paso es calcular el promedio de las funciones de pérdida de todas las muestras:

$$J(W,b) = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{1.23}$$

$$= \frac{1}{N}\sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 \tag{1.24}$$

$$= \frac{1}{N}\sum_{i=1}^{N} \left(f(G^{(i)},W,b) - y^{(i)}\right)^2 \tag{1.25}$$

Al promedio de las funciones de pérdida se le conoce como función de costo J aunque también se le suele llamar simplemente como costo, y esa cantidad se puede interpretar como el error del modelo de la red neuronal en grafos. Por lo tanto, para conseguir las predicciones más precisas lo que se debe hacer es encontrar el conjunto de parámetros $\{W,b\}$ con el que el costo alcance su valor mínimo:

$$\min_{\{W,b\}} J(W,b) \tag{1.26}$$

Por lo general en un inicio a W y b se les otorgan valores aleatorios, así que es de esperar que inicialmente el costo J sea grande. Para reducir el costo del modelo se tienen que seleccionar otros parámetros con los que J esté más cerca de su valor mínimo, para lo cual se puede utilizar el gradiente ∇J ya que es el vector que contiene la información de cuanto crece la función en un punto específico, así que el negativo del gradiente apunta hacia donde hay un mayor descenso en la función. Si en el modelo hay M pesos y N sesgos entonces el gradiente ∇J estará dado por la siguiente expresión:

$$\nabla J(W,b) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_M}, \frac{\partial J}{\partial b_1}, \frac{\partial J}{\partial b_2}, ..., \frac{\partial J}{\partial b_N} \right) \tag{1.27}$$

Una vez que se han calculado las derivadas parciales de la función J respecto a todos los parámetros, se pasa a la última etapa: la actualización de parámetros. La actualización de los parámetros consiste en conseguir un nuevo conjunto de parámetros $\{W',b'\}$ a partir de los parámetros que inicialmente se tenían mediante la siguiente regla:

$$\{W',b'\} = \{W,b\} - \alpha \nabla J \tag{1.28}$$

donde α es un coeficiente llamado tasa de aprendizaje.

El costo J también se puede representar como una hipersuperficie en la que cada punto corresponde a un conjunto específico de parámetros $\{W,b\}$.

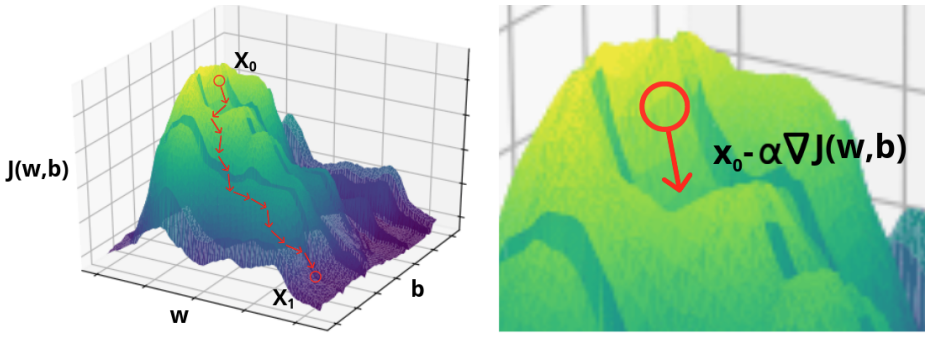

Figura 1.4: Gradiente descendente

En el lado izquierdo de la figura 1.4 está la ejemplificación del algoritmo en donde

se consideran unidimensionales a los vectores W y b. El algoritmo inicia eligiendo un punto de partida x_0 (usualmente aleatorio) y se avanza a una nueva posición a partir del negativo del gradiente $x_0 - \alpha \nabla J$. Este proceso se repite hasta que se alcanza el punto mínimo x_1 de la función.

Como el componente principal de este procedimiento es el vector gradiente, a este algoritmo de optimización se le conoce como descenso por el gradiente [28]. La desventaja del algoritmo es que aunque se avanza en dirección contraria al gradiente esta estrategia no garantiza que se llegue al punto mínimo global de la función, y en algunas ocasiones se llegará a mínimos locales. Es por ello que existen diversas variantes del descenso por el gradiente, como Momentum [29], Adagrad (Adaptative Gradient Algorithm) [30], RMS- Prop (Root Mean Square Propagation) [31], Adadelta [32] y Adam (Adaptive Moment Estimation) [33], entre otros.

Los parámetros en una red neuronal en grafos son los pesos y los sesgos que se ocupan en cada capa (tanto las capas de paso de mensajes neuronales como la de agregación). Sin embargo hay muchos otros parámetros (como la tasa de aprendizaje o el número de capas) que afectan en mayor o menor medida la duración del entrenamiento y la eficiencia del modelo final. A estos parámetros se les conoce como hiperparámetros, y entre los más importantes se encuentran el número de muestras que se utilizan en cada iteración del algoritmo de optimización y el número de iteraciones, también llamadas épocas.

En el planteamiento que se ha seguido hasta ahora se utiliza todo el conjunto de muestras, pero hay casos en los que un conjunto de datos puede llegar a tener miles o incluso millones de muestras, por lo que procesar toda esa información para realizar un solo paso del descenso por el gradiente podría tomar demasiado tiempo. Por esta razón es que se acostumbra que en cada época del entrenamiento se seleccione un subconjunto aleatorio del conjunto de datos, lo cual acelera notablemente los cálculos.

Es importante tener presente que todos los aspectos involucrados en la construcción y entrenamiento de una red neuronal afectan la eficiencia del modelo final. La selección de los hiperparámetros de la red e incluso la obtención del conjunto de datos no es una tarea trivial. Si no se tiene el suficiente cuidado se podrıa terminar con un modelo que en el entrenamiento haya conseguido un costo muy pequeño, pero que al aplicarlo en el ”mundo real”, con datos nuevos y desconocidos, arroje predicciones imprecisas. Por esta razón es que en la práctica se acostumbra a aplicar la red neuronal en un conjunto de datos que no se hayan utilizado para su entrenamiento y entonces se comparan las predicciones con los valores esperados. A este conjunto de datos se le conoce como conjunto de validación. El error que se obtenga en este conjunto de datos es el que va a indicar si los hiperparámetros seleccionados fueron acertados o no. Por último se aplica el modelo en otro conjunto de datos desconocidos y el error que se obtenga con esos datos es el que determina la eficiencia de todo el modelo.

Capítulo 2

Representación de distribuciones atómicas

Antes de la aparición de marcos de aprendizaje basados en grafos para estudiar la distribución atómica, la mayoría de los modelos de aprendizaje automático dependían totalmente de la eficacia de la representación de esta distribución. Como se muestra en la figura 2.1, el marco de los modelos consistía en transformar una distribución atómica representada mediante un vector o una matriz. Esta representación se utiliza entonces como entrada para un modelo de aprendizaje automático que se entrena para obtener una propiedad de la distribución. Es por ello que el cuello de botella clave que limita las aplicaciones de las técnicas tradicionales de aprendizaje profundo, como redes neuronales convolucionales (CNN) para predecir propiedades de las distribuciones atómicas [21] es la gran diversidad de estas distribuciones que no pueden describirse mediante vectores o matrices. Las CNN han mostrado logros prometedores en la predicción de propiedades para distribuciones atómicas pequeñas. Sin embargo, para las distribuciones atómicas grandes, los métodos basados en CNN resultan insuficientes. La existencia de estructuras de enlace e interacciones de muchas partículas en las distribuciones atómicas fomenta el uso de representaciones mediante grafos. Los grafos para distribuciones atómicas se introdujeron por primera vez en el entrenamiento de distribuciones atómicas pequeñas en el trabajo de D. Duvenaud et al. [34]. Se ha demostrado que las características efectivas extraídas de los grafos son más interpretables y las redes neuronales de grafos (GNN) bien entrenadas pueden lograr un rendimiento predictivo sobresaliente en una variedad de tareas.

En la siguiente parte del escrito se presentan distintos algoritmos para representar a las distribuciones atómicas. Hay tres invariancias de las distribuciones atómicas que se desean capturar por cualquier representación: invarianza bajo rotación, traslación y permutación (cambio en el orden particular de los átomos).

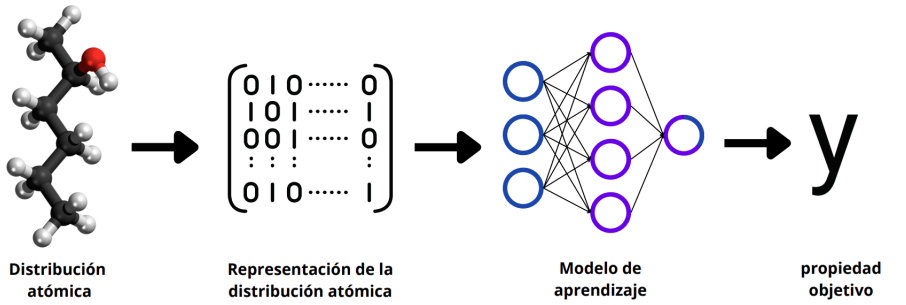

Figura 2.1: Diagrama para realizar predicciones de propiedades de distribuciones atómicas basadas en el aprendizaje automático.

2.1. Representación de moléculas

Una molécula está totalmente definida por los tipos de átomos y sus posiciones y, por tanto, todas las propiedades de interés pueden derivarse de esta información. El aprendizaje automático de moléculas gira en torno al desarrollo de representaciones moleculares para generar vectores o matrices de características y así utilizarlos en métodos como la regresión lineal, las máquinas de soporte vectorial y las redes neuronales.

La representación que se puede dar como entrada directa a la red neuronal es la llamada huella molecular (figura 2.2b) la cual es una representación vectorial de las moléculas que captura detalles precisos de las configuraciones atómicas en su interior. Dependiendo del problema en cuestión, la huella molecular puede ser más detallada. Usualmente se codifica información del tipo de grupos funcionales, propiedades moleculares y parejas de átomos. Una huella molecular popularmente utilizada es la huella de conectividad extendida [35] que se basa en las subestructuras de la molécula.

Un enfoque ampliamente utilizado de igual forma es la representación de cadena (SMILES *Simplified molecular-input line-entry system)* [36] en donde se utilizan caracteres alfanuméricos para codificar la estructura molecular (figura 2.2a). El mayor beneficio de esto, desde la perspectiva del aprendizaje automático, es que se pueden emplear todas las técnicas del procesamiento del lenguaje natural que se tienen hasta el día de hoy.

Una representación de las moléculas con un enfoque distinto es la matriz de Coulomb (figura 2.2c) [13] con entradas

$$C_{ij} = \begin{cases} 0.5Z_i^{2.4} & \text{para } i = j \\ \frac{Z_i Z_j}{\|\mathbf{r}_i - \mathbf{r}_j\|} & \text{para } i \neq j \end{cases} \tag{2.1}$$

Donde Z_i es la carga del átomo i y $\mathbf{r}_i$ es su posición. Así, la invarianza a la traslación y la rotación se consigue utilizando la distancia entre átomos en lugar de las posiciones absolutas. Dada la matriz de Coulomb y fijando la posición de un átomo es posible

resolver un sistema de ecuaciones para obtener los números de átomos originales y posiciones originales de los átomos (hasta la rotación y la traslación) y, por tanto, la representación no tiene pérdida de información. La ventaja de la representación matricial es que se pueden emplear las técnicas de redes neuronales convolucionales ya que operan en el espacio de las matrices. Sin embargo, la desventaja de la matriz de Coulomb es que no es invariante a las permutaciones de los índices de los átomos.

Para solucionar el problema Hansen et al. [14]) introduce una variante de la matriz

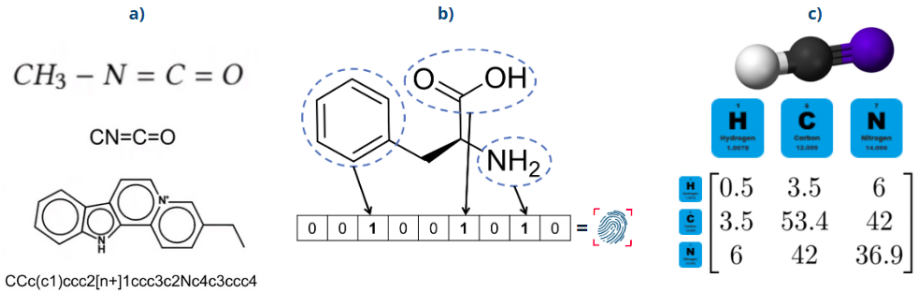

Figura 2.2: Representaciones moleculares. a) SMILES. b) Huella molecular. c) Matriz de Coulomb.

de Coulomb llamada bolsas de enlaces. En la representación se mantienen sólo las entradas de la matriz de Coulomb que corresponde a los enlaces entre un par de átomos y las entradas se organizan en diferentes bolsas en función de los tipos de átomos. De nuevo, las entradas se ordenan dentro de cada bolsa.

Para lograr mejorar la representación de bolsas de enlaces Huang y Lilienfeld [37] introducen la representación de enlaces y ángulos en la cual se desarrolla aún más al incluir también ángulos entre tripletes de átomos y torsiones entre cuatrillizos, lo que da una noción más detallada de la similitud.

2.2. Representación de cristales

Un cristal está formado por átomos dispuestos en un patrón que se repite periódicamente en tres dimensiones [38]. El patrón al que se refiere esta definición puede consistir en un grupo de átomos, una molécula o un grupo de moléculas y es conocido como celda unitaria. Se representa una celda unitaria con un conjunto de tipos de átomos con sus posiciones y un conjunto de tres vectores de celda $a,b,c \in \mathbb{R}^3$. Los vectores no necesitan ser ortogonales o de igual longitud. Los vectores forman una base de la celda unitaria, es decir que pueden llegar a todos los puntos mediante combinaciones lineales enteras de los tres vectores $t = ua + vb + wc$ como se muestra para dos dimensiones en la figura 2.3.

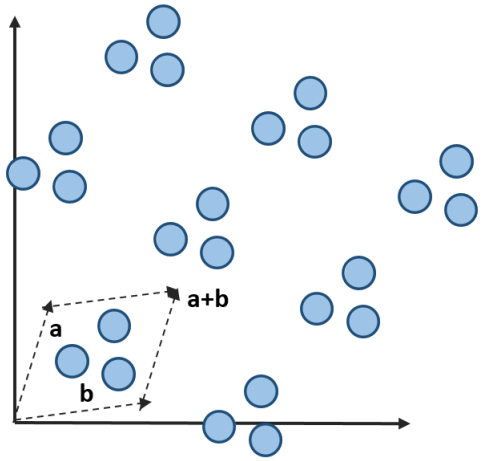

Figura 2.3: Ejemplo de una celda unitaria con vectores base **a** y **b** .

En el caso de los cristales, también es de interés diseñar representaciones que sean invariantes a la traslación, la rotación y la permutación. Además, como la elección de la celda unitaria no es única las representaciones también deben ser invariantes a la elección de la celda unitaria.

Faber et al. [39] han propuesto tres extensiones diferentes de la matriz de Coulomb que tienen en cuenta la periodicidad: la matriz de suma de Ewald, la matriz extendida de Coulomb y la matriz sinusoidal; en las cuales se consideran únicamente los átomos que se encuentran dentro de la celda unitaria. En la matriz sinusoidal las entradas se definen a partir de la siguiente expresión

$$C_{ij} = \begin{cases} 0.5Z_i^{2.4} & \text{para } i = j \\ Z_i Z_j \tilde{\Phi}(\mathbf{r}_i, \mathbf{r}_j) & \text{para } \quad i \neq j \end{cases} \tag{2.2}$$

Donde Z_i es la carga del átomo i, $\mathbf{r}_i$ es su posición y la función $\tilde{\Phi}(\mathbf{r}_i, \mathbf{r}_j)$ guarda la información de la periodicidad que existe dentro de la estructura como se puede ver en

la figura 2.4a.

Otra representación basada en las distancias interatómicas es la representación de la función de distribución radial parcial [40]. Esta representación considera la distribución de las distancias entre dos tipos de átomos.

Los cristales pueden clasificarse por su simetría y cada cristal pertenece a uno de los 230 grupos espaciales [41]. La simetría condiciona donde pueden estar los átomos, condición descrita por los llamados sitios de Wyckoff [41]. Esto fue utilizado recientemente por Jain y Bligaard [42] para construir un modelo en el que los tipos de átomos y los sitios Wyckoff se codifican en una matriz (figura 2.4b).

Los grafos son una alternativa para describir las estructuras cristalinas. Para ello se selecciona una esfera alrededor de cada átomo para analizar las interacciones entre este átomo y aquellos que quedan dentro de la esfera.

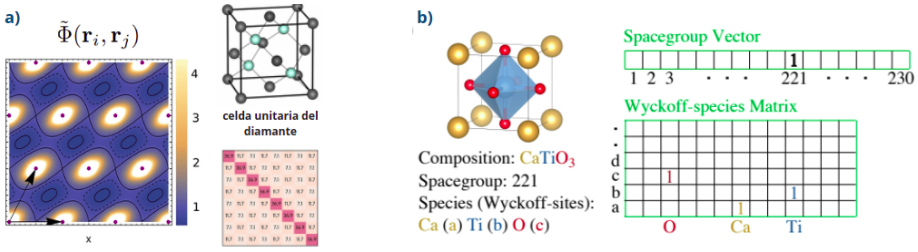

Figura 2.4: Representaciones de cristales. a) Matriz sinusoidal. b) Matriz de Wyckoff

2.3. Representación del entorno local

Las representaciones descritas anteriormente muestran a cada átomo enlazado con todos los demás en la distribución, sin embargo para distribuciones con muchos átomos esta premisa se vuelve costosa computacionalmente, es por ello que se desarrollan otras representaciones que se centran en el entorno local del átomo y no la interacción del átomo con toda la distribución. Un ejemplo excelente es la función de simetría centrada en el átomo [43], donde una serie de funciones predefinidas capturan información radial y angular sobre la vecindad de un átomo y una red neuronal se emplea para asignar una contribución energética a cada átomo. A partir de este modelo se derivan otros

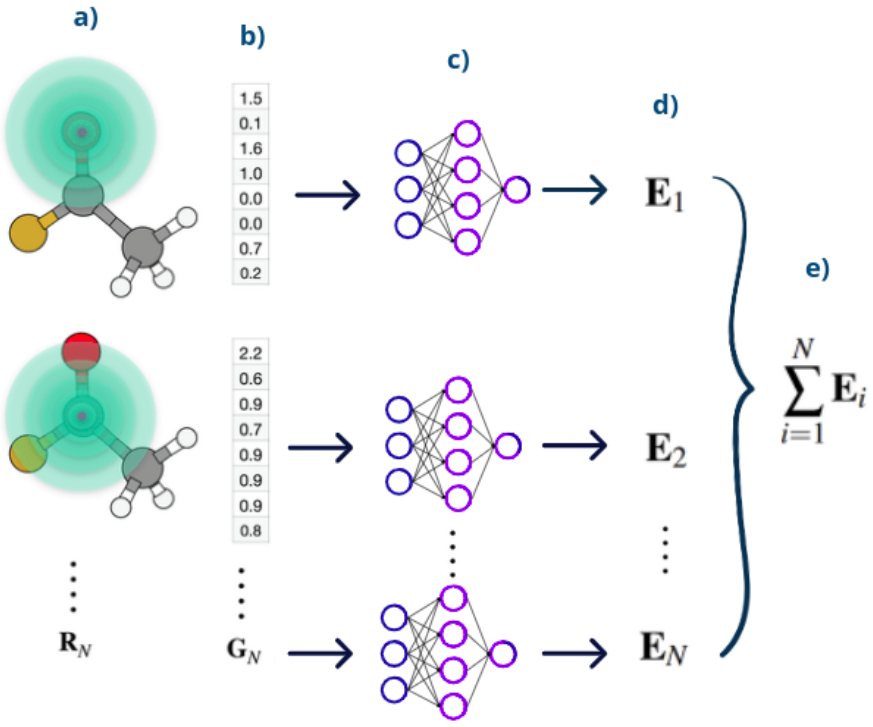

Figura 2.5: Representación del entorno local. a) Entorno local del átomo. b) Las funciones de simetría se codifican en vectores. c) Cada vector sirve de entrada para un modelo de red neuronal. d) Se predice la contribución atómica de la energía. e) Para obtener la energía total de la distribución se suman las contribuciones atómicas

usando el mismo principio pero variando las funciones que describen al entorno local, por ejemplo la representación *Smooth Overlap of Atomic Positions* [44], en donde se utilizan de igual forma funciones predefinidas las cuales son armónicos esféricos y funciones de base radial para describir la interacción entre el átomo y cada vecino.

La ventaja de las representaciones locales es que al introducir una distancia de cor-

te, también pueden utilizarse para estructuras cristalinas [44].

2.4. Representación por aprendizaje

Un posible inconveniente de las representaciones introducidas anteriormente es que deben ser diseñadas manualmente antes de entrenar la red neuronal. Si el diseño de la representación no es muy fiel a la distribución, por ejemplo cuando la representación es casi idéntica para dos estructuras muy diferentes, la potencia expresiva de la red neuronal y la precisión que se puede alcanzar son limitados a *priori*. Además, un número creciente de dimensiones de entrada puede volverse rápidamente caro desde el punto de vista computacional, tanto para diseñar las representaciones como para evaluar la red neuronal. Esto sucede cuando se modelan distribuciones de muchas partículas, en los que normalmente se asume la ortogonalidad entre distintos elementos (lo que aumenta el número de dimensiones de entrada). Es por ello que se da un cambio de paradigma en las representaciones de distribuciones atómicas en donde ya no es necesario diseñarlas manualmente sino que el mismo modelo toma las cargas nucleares y coordenadas cartesianas como entrada y aprende una representación adecuada de la distribución, a este nuevo tipo de arquitecturas se le conoce como arquitecturas de extremo a extremo.

Muchas arquitecturas de extremo a extremo se han inspirado en la red neuronal en grafos de Scarselli et al. [22] y más tarde se denominaron colectivamente como redes de paso de mensajes neuronales (*Message-passing neural network* MPNN). En este tipo de modelo, las moléculas se consideran grafos no dirigidos donde los átomos están representados por nodos y las interacciones entre ellos como aristas. Mediante el intercambio de información entre los nodos a lo largo de las aristas (paso de mensajes) pueden ser modeladas interacciones físicas complejas. Un ejemplo destacado es la red neuronal tensorial profunda (*Deep tensor neural network* DTNN) [45]. Este modelo ha sido un punto de partida para crear nuevas arquitecturas, como SchNet [12], Dimenet [11] o Physnet [46]. Debido a que no se utiliza ninguna regla fija para construir representaciones de distribuciones atómicas, las arquitecturas de extremo a extremo son capaces de adaptar automáticamente las representaciones del entorno a la variable objetivo.

2.5. Representación por grafos

En general, una molécula o un cristal puede representarse mediante un grafo G, donde los átomos individuales se representan como nodos y los enlaces se presentan como aristas, ya sean dirigidas o no dirigidas (figura 2.6). Los enlaces no son necesariamente enlaces químicos, sino que pueden ser cualquier interacción entre dos átomos y la elección de los enlaces depende de la tarea de aprendizaje. Tanto a los nodos como a las aristas en un grafo (para una molécula o un cristal) se le pueden atribuir características mediante un vector que representen propiedades a nivel atómico. En la siguiente parte del escrito se presentan distintos algoritmos para representar a las distribuciones atómicas.

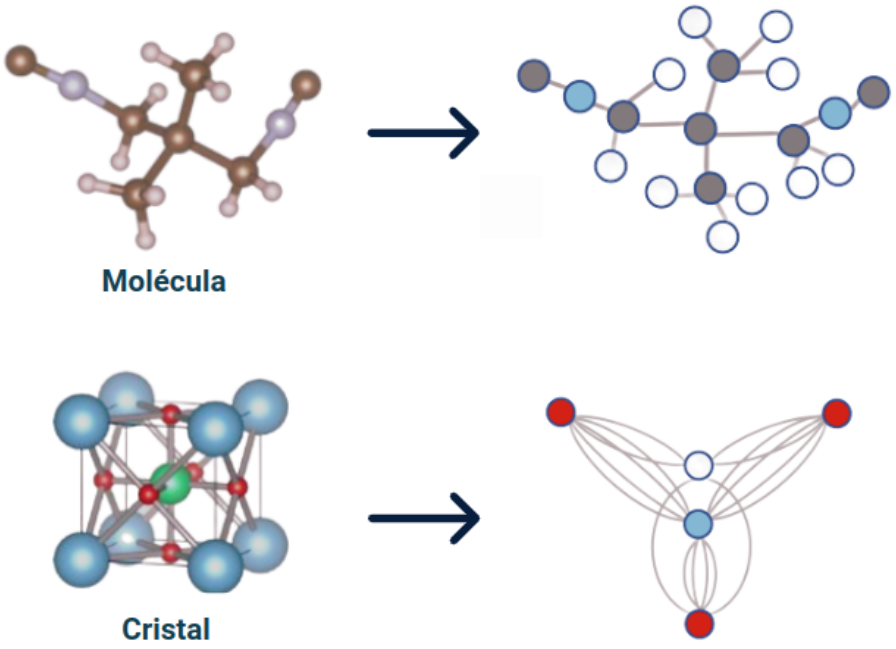

Figura 2.6: Una molécula o cristal puede representarse como un grafo, donde los nodos representan átomos en la molécula o celda cristalina, y las aristas representan cualquier información interatómica como la distancia. Los nodos y las aristas del grafo están entonces en vectores basados en múltiples niveles de información extraídos de las distribuciones atómicas. [16]

Capítulo 3

Red convolucional en grafos para predecir propiedades de distribuciones atómicas

Se han propuesto varias redes neuronales en grafos que formulan la tarea de predecir propiedades físicas de las distribuciones atómicas como el aprendizaje de un mapeo $f(G,W) \longrightarrow y$ donde W es un conjunto de parámetros, G es el grafo que representa a la distribución atómica y y es una propiedad objetivo. En este capítulo se describe uno de los primeros modelos que incorpora grafos a los cristales propuesto por T. Xie y J. C. Grossman en 2018 [10].

3.1. Base de datos

Para poder entrenar al modelo es necesario contar con una base de datos de estructuras cristalinas. Para ello se utiliza la base de datos de libre acceso llamada *Materials Project* [47]. La base de datos incluye un conjunto diverso de compuestos inorgánicos que van desde los metales simples hasta los minerales complejos. En total son 46,744 compuestos. Los cuales consisten en hasta siete elementos diferentes, de ellos el 90 % son compuestos binarios, ternarios y cuaternarios. El número de átomos en la celda primitiva oscila entre 1 y 200 y el 90 % de los compuestos tienen menos de 60 átomos. Teniendo en cuenta que la mayoría de las estructuras cristalinas proceden de la base de datos de *Inorganic Crystal Structure Database* (ICSD) [48], esta base de datos es una buena representación de los compuestos inorgánicos.

La información de cada estructura cristalina se guarda en archivos de tipo *CIF*, un *CIF* o *Crystallographic Information File* es el formato estándar para almacenar datos de las estructuras cristalinas. La información *CIF* tiene una estructura o formato específico que debe seguirse para que los programas que trabajan con estructuras cristalinas puedan leer el archivo. Los *CIF* de cada compuesto se descargan directamente

a través de la interfaz con la que cuenta *Pymatgen (Python Materials Genomics)* que es una robusta librería de *Python* de código abierto para el análisis de materiales.

```
# generated using pymatgen
data_NaCl
_symmetry_space_group_name_H-M   'P 1'
_cell_length_a   5.69169400
_cell_length_b   5.69169400
_cell_length_c   5.69169400
_cell_angle_alpha   90.00000000
_cell_angle_beta   90.00000000
_cell_angle_gamma   90.00000000
_symmetry_Int_Tables_number   1
_chemical_formula_structural   NaCl
_chemical_formula_sum   'Na4 Cl4'
_cell_volume   184.38459333
_cell_formula_units_Z   4
loop_
 _symmetry_equiv_pos_site_id
 _symmetry_equiv_pos_as_xyz
  1  'x, y, z'
loop_
 _atom_type_symbol
 _atom_type_oxidation_number
  Na+  1.0
  Cl-  -1.0
loop_
 _atom_site_type_symbol
 _atom_site_label
 _atom_site_symmetry_multiplicity
 _atom_site_fract_x
 _atom_site_fract_y
 _atom_site_fract_z
 _atom_site_occupancy
  Na+  Na0  1  0.00000000  0.00000000  0.00000000  1
  Na+  Na1  1  0.00000000  0.50000000  0.50000000  1
  Na+  Na2  1  0.50000000  0.00000000  0.50000000  1
  Na+  Na3  1  0.50000000  0.50000000  0.00000000  1
  Cl-  Cl4  1  0.50000000  0.00000000  0.00000000  1
  Cl-  Cl5  1  0.50000000  0.50000000  0.50000000  1
  Cl-  Cl6  1  0.00000000  0.00000000  0.50000000  1
  Cl-  Cl7  1  0.00000000  0.50000000  0.00000000  1
```

Figura 3.1: Archivo .cif del cloruro de sodio (NaCl)

Como se muestra en la figura 3.1 el archivo *cif* contiene la información de la estructura cristalina del cloruro de sodio (NaCl). El archivo de texto se divide en bloques donde cada bloque empieza con una línea de tipo *nombredelbloque_*. En la línea 2 empieza el bloque *data_NaCl* en donde se específica el nombre del grupo de simetría espacial del cristal, el tamaño de los vectores de la celda unitaria así como los ángulos entre ellos, la fórmula química, el volumen de la celda unitaria y las unidades. En el siguiente bloque llamado *loop_* viene la simetría, en el siguiente el tipo de símbolo del átomo y el numero de oxidación y en el último bloque contiene el símbolo del átomo, como se etiqueta, la multiplicidad de la simetría y sus posiciones en coordenadas cartesianas.

Dado que en este trabajo sólo se ocupan las coordenadas cartesianas y la simetría de la celda unitaria no se ahonda en las demás propiedades del archivo *cif*.

3.2. Construcción de grafos

Una vez obtenidos los datos se deben procesar al formato adecuado del modelo, es decir, se deben construir los grafos.

La codificación de las aristas con información de distancia es necesaria para capturar la información espacial. Fue propuesto por primera vez por el código DTNN en 2017 para capturar la información espacial de las moléculas [45]. Sin embargo, en DTNN, se utiliza la matriz de distancia completa, que corresponde a un grafo totalmente conectado para las moléculas . Esto no funciona bien para las distribuciones de materiales con celdas unitarias grandes porque tiene un escalamiento, y es menos claro cómo definir las distancias en una distribución periódica. En consecuencia, se utiliza el algoritmo del vecino más cercano para construir los grafos que representan la estructura tridimensional de los materiales, que tiene un escalamiento con respecto al tamaño de la distribución. En esta representación, las interacciones de corto alcance se aprenden directamente en cada operación de convolución del grafo con los vecinos más cercanos, mientras que las interacciones de largo alcance se aprenden propagando la información a través del grafo utilizando múltiples convoluciones del mismo. Como resultado, toda la distribución de cada material está representado por un grafo localmente conectado que corresponde a la estructura tridimensional en celda periódica.

Dado un conjunto de átomos en la celda unitaria en el espacio tridimensional que se repite periódicamente, se construye un grafo donde los nodos representan los átomos y las aristas representan la interacción cercana entre pares de átomos. Para determinar las aristas de un nodo se coloca una esfera de radio R_c alrededor de ese átomo. El átomo que está en el centro de la esfera se llama átomo central y al radio R_c se le conoce como radio de corte. Los átomos que se encuentran dentro de la esfera se definen como los átomos vecinos del átomo central por lo que se dibuja una arista dirigida del átomo central i al átomo vecino j, esto significa que las aristas son siempre bidireccionales. Además, como los nodos se repiten periódicamente, dos átomos pueden tener múltiples aristas dirigidas si se encuentran dentro de la distancia de corte en múltiples celdas repetidas. Si un átomo i tiene más de una arista hacia un átomo j, cada arista representa al átomo j en una celda diferente, lo que resulta en distancias relativas y aristas únicas. Desde el punto de vista del átomo, la representación de multigrafos anterior de una distribución atómica capta con precisión la estructura tridimensional que rodea a cada átomo, teniendo en cuenta las condiciones periódicas de los límites.

En la figura 3.2 está la ejemplificación de este método donde se limita a $\mathbb{R}^2$. Se cuenta con tres celdas unitarias adyacentes en donde cada celda tiene 3 átomos y del lado derecho se encuentra el grafo que se construye conforme cada paso del método. Para iniciar el procedimiento se elige la celda unitaria de en medio (por la periodicidad de la distribución el grafo es invariante a la elección de celda unitaria). En el primer paso 1) se toma como átomo central al átomo 0 y se dibuja la circunferencia con radio R_c, dado que los átomos 1 y 2 dentro de la misma celda caen dentro de la circunferencia se cuentan como átomos vecinos. De la misma forma el átomo 1 pero de la celda unitaria desplazada a la derecha cae dentro de la circunferencia, es por ello que se traza una

arista más en el grafo del átomo 0 al átomo 1. En el paso 2) el átomo central es el átomo 1 y los átomos vecinos son el 0 y 2 que están en la misma celda unitaria y el 0 que está una celda unitaria desplazada a la izquierda. Finalmente en el paso 3) el átomo central es el átomo 2 y los átomos vecinos son el 0 y 1.

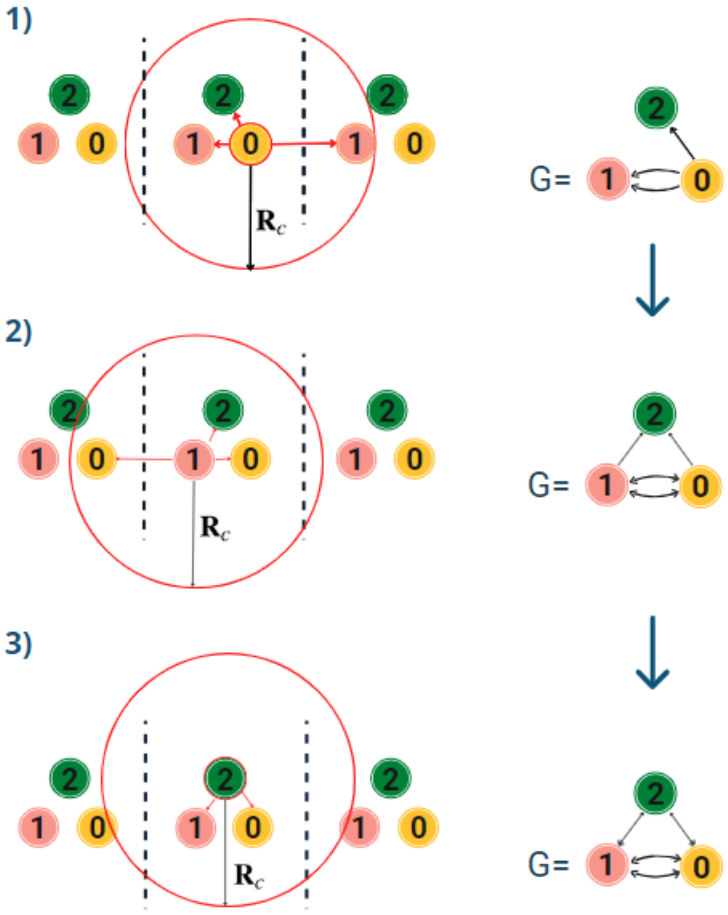

Figura 3.2: Construcción del grafo

Un aspecto importante a considerar es la elección del radio de corte R_c ya que este permanece fijo para todas las estructuras cristalinas, lo que limita al modelo. En el artículo se establece $R_c = 8\mathring{A}$ sin embargo se consideran 12 vecinos más cercanos para construir los grafos de todos los materiales, porque 12 es el número de vecinos bajo el empaquetamiento cerrado.

El empaquetamiento cerrado en los cristales se refiere a la disposición eficiente del espacio de las partículas constituyentes en una red cristalina. Para entender mejor este

empaquetamiento, se debe suponer que todas las partículas (átomos, moléculas e iones) tienen la misma forma sólida esférica. Por tanto, la celda unitaria de una red es una forma cúbica. Ahora bien, cuando se apilen esferas en la celda, siempre habrá algunos espacios vacíos. Para minimizar estos espacios vacíos, la disposición de estas esferas debe ser muy eficiente. Las esferas deben disponerse lo más juntas posible para eliminar los espacios vacíos.

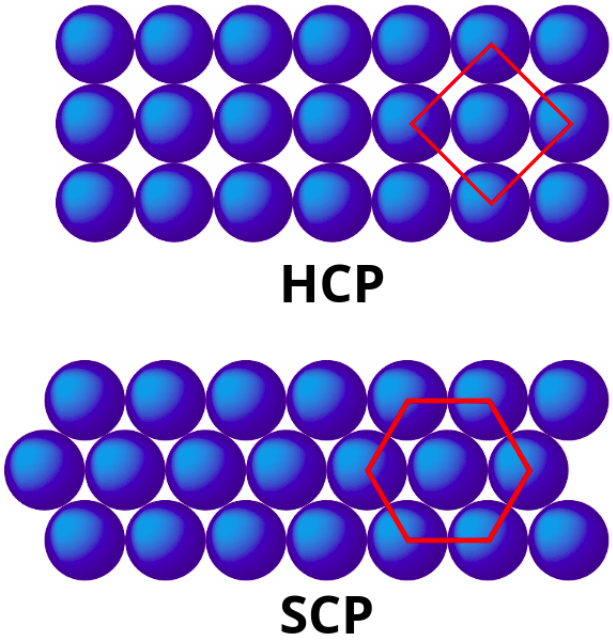

Figura 3.3: Empaquetamiento cerrado

La disposición más eficiente de las esferas en la celda cúbica permite 12 átomos alrededor del átomo central, por lo que es el número máximo de vecinos cercanos que puede tener un átomo dentro de una celda unitaria.

3.3. Propiedades de grafos

Una vez que se tiene la estructura del grafo, se añade información a los nodos y aristas a partir de vectores de características representando propiedades de átomos y enlaces. Esto debido a que la estructura del grafo por si sola no representa en su totalidad a la estructura cristalina. Un claro ejemplo es que el grafo no reconoce si el átomo se encuentra en una celda unitaria desplazada o no.

3.3.1. Propiedades de átomos

Las propiedades de átomos se codifican en vectores de características de nodos v_i utilizando una codificación llamada *one hot encoding* que consiste en convertir el rango de valores de una característica en un vector binario al cual se le asigna un 1 en la posición asociada a dicho valor específico y dejar las demás con 0. Para los valo-

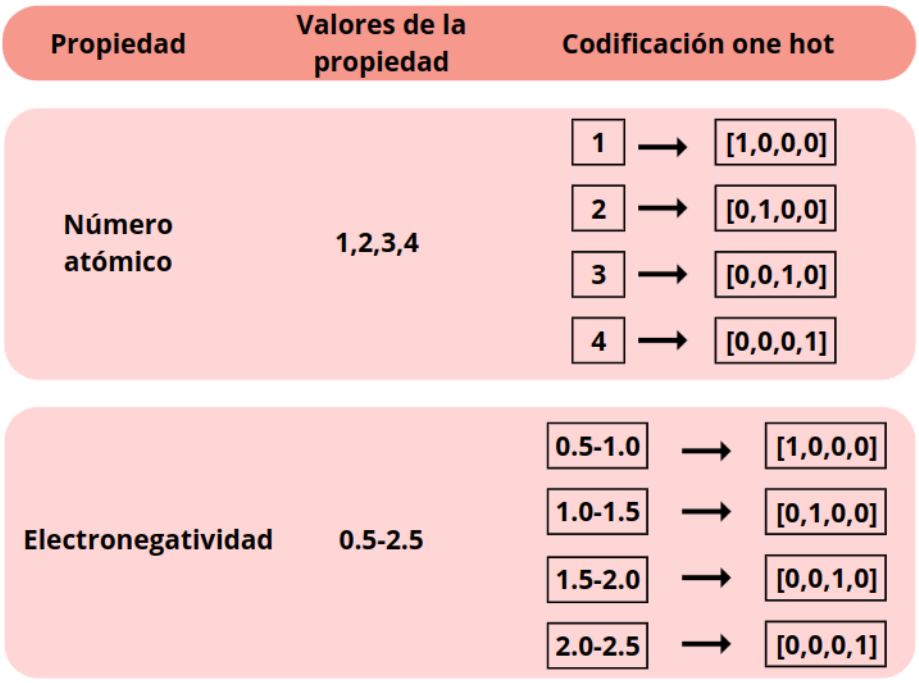

Figura 3.4: Codificación *one hot*

res discretos se añade una nueva columna al vector por cada valor distinto que pueda tomar la propiedad; para los valores continuos primero se particionan los valores que pueda tomar la propiedad en rangos uniformes y para construir el vector se añade una columna nueva por cada rango obtenido. La codificación no es estrictamente necesaria ya que es posible utilizar el valor numérico de las propiedades sin embargo se requiere normalizar los valores en dicho caso debido a que existen diferentes rangos de valores.

En la figura 3.4 se visualiza esta codificación a partir de 2 propiedades atómicas, el número atómico y la electronegatividad. En el ejemplo se reduce el rango de valores que permiten dichas propiedades para simplificar la codificación.

Son 9 propiedades que se añaden a los átomos, las cuales son grupo, periodo, electronegatividad, radio covalente, electrones de valencia, primera energía de ionización, afinidad de los electrones, bloque y volumen atómico. Dado que por cada propiedad tendremos un vector binario, para añadir las 9 propiedades en un solo vector lo que se hace es concatenar los 9 vectores.

3.3.2. Propiedades de enlaces

La propiedad de enlace se define como la distancia entre los dos átomos. Las distancias se convierten en un vector expandiéndolas en una serie de funciones de base gaussiana. Una función de base guassiana

$$\alpha e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.1}$$

es una función que es simétrica en torno a un punto y que suele decaer a cero a medida que se aleja del centro. El parámetro α es el valor del punto más alto de la campana, μ es la posición del centro de la campana y σ (la desviación estándar) controla el ancho de la campana.

La distancia d_{ij} entre los átomos i y j se transforman en un vector $\hat{\mathbf{d}}_{ij}$ de cierta longitud D, es decir $\hat{\mathbf{d}}_{ij} \in \mathbb{R}^D$ a partir de la siguiente expresión:

$$\hat{\mathbf{d}}_{ij} = \left[e^{-\frac{(d-(\mu_{\text{mín}}+k\Delta\mu))^2}{2\sigma^2}} \right]_{0 \leq k \leq \frac{\mu_{\text{máx}}}{\Delta\mu}} \tag{3.2}$$

El término de $\mu_{\text{mín}} + k\Delta\mu$ únicamente determina que la media de la distribución gaussiana μ va a variar en un intervalo por lo que va a adoptar varios valores según sean definidos los parámetros. Los primeros parámetros $\mu_{\text{mín}}$ y $\mu_{\text{máx}}$ son el valor mínimo y el valor máximo que μ puede tomar, es decir $\mu \in [\mu_{\text{mín}}, \mu_{\text{máx}}]$. El otro parámetro es $\Delta\mu$ que representa el tamaño del paso que μ puede dar y k es el índice que va a ir variando donde $0 \leq k \leq \frac{\mu_{\text{máx}}}{\Delta\mu}$. Finalmente σ es la desviación estándar.

Para visualizar la expansión gaussiana se puede observar en la figura 3.5 que si se considera a $d_{ij} = d = 1.5$ (la distancia entre el átomo i al átomo j es 1.5 Å) y se toman los siguientes valores para los parámetros: $\mu_{\text{mín}} = 0$, $\mu_{\text{máx}} = 2$, $\Delta\mu = 0.5$, $\sigma = 0.2$ entonces dado que $0 \leq k \leq \frac{\mu_{\text{máx}}}{\Delta\mu}$, sustituyendo da $0 \leq k \leq \frac{2}{0.5} \Rightarrow 0 \leq k \leq 4$. Para obtener los valores de μ basta con variar a k en la expresión $\mu_{\text{mín}} + k\Delta\mu$ (como μ depende de k se añade la notación μ_k):

$$\begin{aligned}
\mu_0 &= 0 + 0 \cdot 0.5 = 0 \\
\mu_1 &= 0 + 1 \cdot 0.5 = 0.5 \\
\mu_2 &= 0 + 2 \cdot 0.5 = 1 \\
\mu_3 &= 0 + 3 \cdot 0.5 = 1.5 \\
\mu_4 &= 0 + 4 \cdot 0.5 = 2
\end{aligned}$$

Dado que μ_k puede adoptar 5 valores distintos entonces se tienen 5 distribuciones gaussianas. Para generar la expansión de d se evalua sobre la función de distribución de

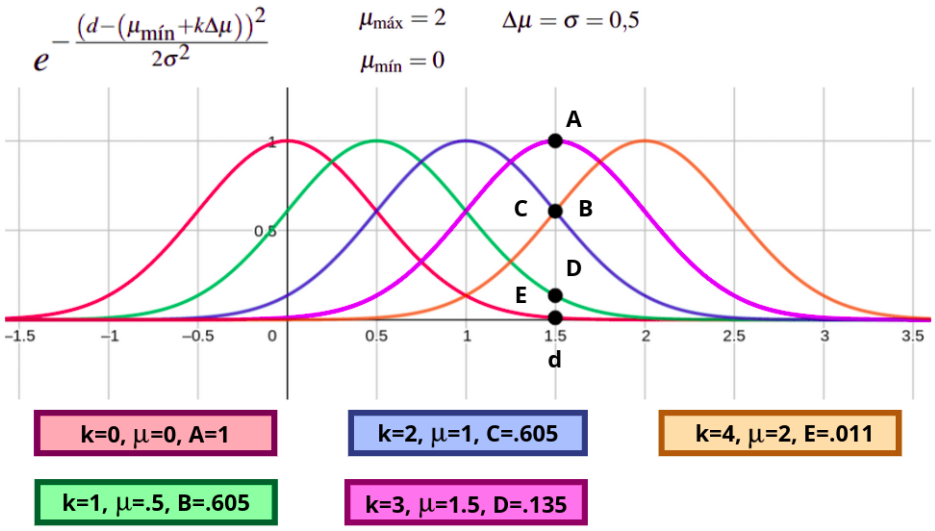

Figura 3.5: Expansión guassiana para d=1.5

cada una de ellas obteniendo así 5 valores. Por lo que el vector $\hat{\mathbf{d}}_{ij}$ queda de la siguiente manera $\hat{\mathbf{d}}_{ij} = [0.011, 0.135, 0.605, 1, 0.605]$

El parámetro $\Delta\mu$ define la granularidad de la expansión y determina la precisión. En el artículo, $\Delta\mu$ se fija en $0.2\mathring{A}$. Se elige de forma que todas las distancias que se producen en los conjuntos de datos estén cubiertas por las expansiones. La elección de menos centros corresponde a la reducción de la resolución del filtro, mientras que la restricción del rango de los centros corresponde al tamaño del filtro en una capa convolucional habitual.

La idea de utilizar gaussianas para representar las distancias se introdujo anteriormente [45]. Las distancias se expanden en una base gaussiana, dando lugar a un vector de características $\hat{\mathbf{d}}_{ij}$, que tiene en cuenta la diferente naturaleza de las interacciones en los distintos regímenes de distancia. Esto puede verse como una codificación suave de las distancias, lo que facilita que una red neuronal aprenda una función en la que los valores de la distancia no estén muy correlacionados.

3.4. Modelo

El modelo se divide en dos capas principales: la de convolución y la de agrupación. En esta parte del trabajo se describe detalladamente la arquitectura del modelo.

3.4.1. Capa de convolución

Antes de mandar los vectores de características al paso de mensajes neuronales se aplica una transformación lineal para reducir la dimensión de los vectores. Por lo que dado el estado oculto 0 (que es equivalente al vector de características inicial), se aplica la siguiente transformación:

$$\mathbf{v}_i^{(0)} W + b$$

Las capas de convolución actualizan iterativamente los estados ocultos del i-ésimo átomo $\mathbf{v}_i^{(t)}$ mediante el paso de mensajes neuronales con los átomos y enlaces circundantes. El marco de paso de mensajes neuronales que se utiliza en el modelo es el siguiente:

$$\mathbf{v}_i^{(t+1)} = g\left[\sum_{j \in \mathscr{N}_{(i)}} \sum_{k=1}^{M_j} \left(\mathbf{v}_j^{(t)} \oplus \mathbf{e}_{(i,j)_k}\right) \mathbf{W}_c^{(t)} + \mathbf{v}_i^{(t)} \mathbf{W}_s^{(t)} + \mathbf{b}^{(t)}\right] \tag{3.3}$$

donde $\mathbf{v}_i^{(t)}$ es el estado oculto del átomo i en el paso t, $\mathbf{e}_{(i,j)_k}$ es el vector de características del enlace entre el átomo i con el átomo j. $\mathscr{N}_{(i)}$ es la vecindad del átomo es decir el conjunto donde se encuentran los átomos vecinos y M_j es el número de enlaces que existen entre el átomo i y el átomo j. El término $\oplus$ denota la concatenación entre el estado oculto del átomo y del enlace. $\mathbf{W}_c^{(t)}$ y $\mathbf{W}_s^{(t)}$ son las matrices de los pesos dentro de la convolución, y $\mathbf{b}^{(t)}$ es el sesgo o *bias*. La función g es la función de activación *softplus*. Como se muestra en la ecuación el vector de características de los enlaces $\mathbf{e}_{(i,j)_k}$ no se actualiza durante el paso de mensajes neuronales, esto limita el modelo y se deja a trabajo futuro.

La función de activación *softplus* está definida como:

$$g(x) = ln\,(1 + e^x) \tag{3.4}$$

Esta función permite quitar la linealidad a los datos como se observa en la gráfica de la función (figura 3.6), en donde la función restringe la salida de los valores para que sea siempre positiva.

Para entender de forma mas completa este método de paso de mensajes neuronales se describe iteración tras iteración el funcionamiento de éste:

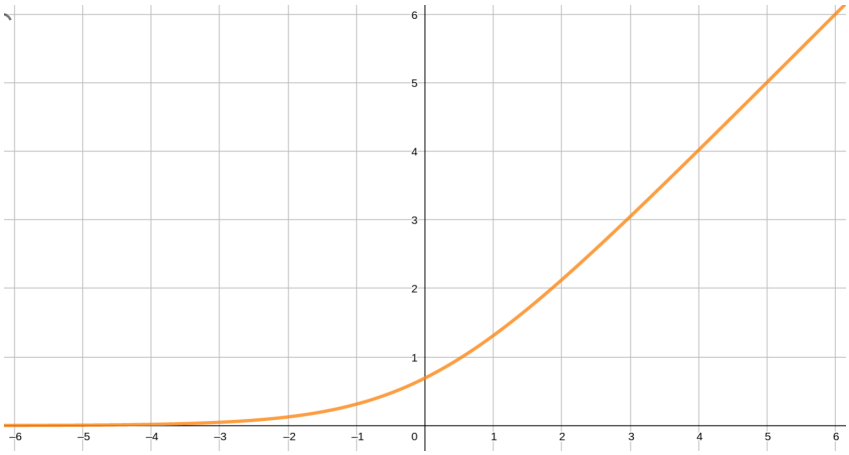

Figura 3.6: Gráfica de la función *softplus*

Iteración t=0: Se aplica la transformación lineal para reducir la dimensión de los vectores de características de los átomos.

Iteración t=1: Todos los estados ocultos de los átomos se actualizan con las contribuciones de los átomos vecinos y sus respectivos enlaces.

Iteración t=2: Todos los estados ocultos de los átomos $\mathbf{v}_i^{(2)}$ se actualizan con las contribuciones de los átomos vecinos y sus respectivos enlaces. Pero asimismo ya $\mathbf{v}_i^{(2)}$ contiene contribuciones por otros átomos. Así que, intuitivamente, $\mathbf{v}_i^{(2)}$ tiene ya información de sus primeros vecinos y con el paso de mensajes neuronales obtiene información de sus segundos vecinos, esto permite expresar las interacciones entre pares y de orden superior entre los átomos.

Una de las limitaciones de la ecuación 4.3 es que utiliza una matriz de pesos compartidos para todos los vecinos del átomo i, es decir que si se suponen 2 vecinos del átomo 0, sean los átomos 1 y 2 con sus estados ocultos correspondientes $\mathbf{v}_1^{(0)}$ y $\mathbf{v}_2^{(0)}$ (en el ejemplo no se considera la contribución de los enlaces) entonces la función de convolución se obtiene a partir de:

$$\mathbf{v}_1^{(1)} = g\left[\sum_{j\in\mathcal{N}_{(i)}}\sum_{k=1}^{M_j}\left(\mathbf{v}_j^{(0)}\oplus\mathbf{e}_{(i,j)_k}\right)\mathbf{W}_c^{(0)} + \mathbf{v}_0^{(0)}\mathbf{W}_s^{(0)} + \mathbf{b}^{(0)}\right] \tag{3.5}$$

$$= g\left[\left(\mathbf{v}_1^{(0)} + \mathbf{v}_2^{(0)}\right)\mathbf{W}_c^{(0)} + \mathbf{v}_0^{(0)}\mathbf{W}_s^{(0)} + \mathbf{b}^{(0)}\right] \tag{3.6}$$

$$= g\left[[v_{11}+v_{21}, v_{12}+v_{22}, v_{13}+v_{23}]\cdot[w_{c1}, w_{c2}, w_{c3}]^T + \mathbf{v}_0^{(0)}\mathbf{W}_s^{(0)} + \mathbf{b}^{(0)}\right] \tag{3.7}$$

$$= g\left[w_{c1}(v_{11}+v_{21}) + w_{c2}(v_{12}+v_{22}) + w_{c3}(v_{13}+v_{23}) + \mathbf{v}_0^{(0)}\mathbf{W}_s^{(0)} + \mathbf{b}^{(0)}\right] \tag{3.8}$$

Como se puede apreciar en la ecuación 4.8 la primer entrada de los estados ocultos de los vecinos 1 y 2 (v_{11} y v_{21} respectivamente) comparten el mismo peso w_{c1} lo que indica que se le está dando la misma importancia tanto al vecino 1 como al vecino 2 por lo tanto la matriz $\mathbf{W}_c^{(0)}$ no tiene en cuenta las diferencias de fuerza de interacción entre los vecinos.

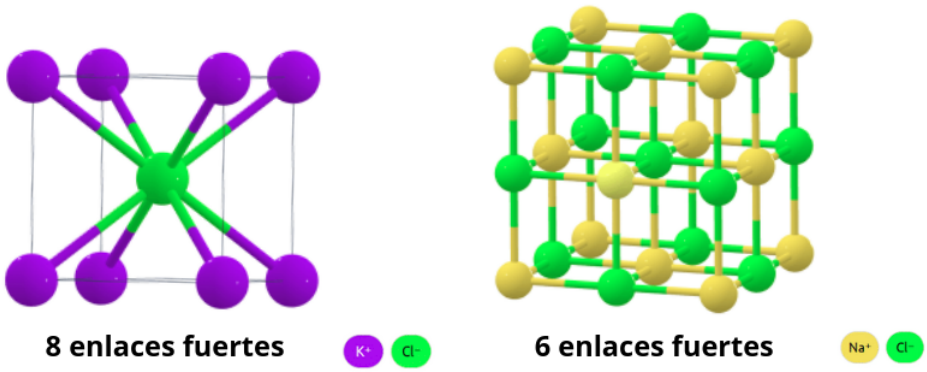

Figura 3.7: Celda unitaria del cloruro de potasio y sodio

Esto limita el modelo ya que los diferentes vecinos tienen diferente fuerza de interacción con el átomo central.

Para los materiales inorgánicos, se utilizan 12 vecinos más cercanos para construir los grafos de todos los materiales, porque 12 es el número de vecinos bajo el empaquetamiento cerrado como se había comentado anteriormente. Pero diferentes materiales tienen diferentes números de vecinos fuertes en su estructura. Por ejemplo, como se muestra en la figura 3.7 entre estos 12 vecinos más cercanos, el cloruro de potasio (KCl) tiene 8 vecinos con enlaces fuertes, y el cloruro de sodio (NaCl) tiene 6 vecinos con enlaces fuertes. Es muy difícil saber qué vecino es importante en un material inorgánico. Por ello, se diseña una arquitectura de atención para aprender la importancia de los vecinos. En concreto, se aprende un prefactor entre 0 y 1 utilizando la

información sobre la distancia entre átomos y los tipos de cada átomo.

$$\text{prefactor} = \sigma\left(\mathbf{v}_i^{(t)} \oplus \mathbf{v}_j^{(t)} \oplus \mathbf{e}_{(i,j)_k}\mathbf{W}_f^{(t)} + \mathbf{b}_f^{(t)}\right) \tag{3.9}$$

donde $\mathbf{v}_i^{(t)} \oplus \mathbf{v}_j^{(t)} \oplus \mathbf{e}_{(i,j)_k}$ denota la concatenación del estado oculto del átomo central $\mathbf{h}_u^{(t)}$ (en la iteración t), del átomo vecino $\mathbf{h}_v^{(t)}$ y del enlace. Además, σ denota la función sigmoide. La función sigmoide viene dada por la siguiente expresión:

$$S(x) = \frac{1}{1+e^{-x}} \tag{3.10}$$

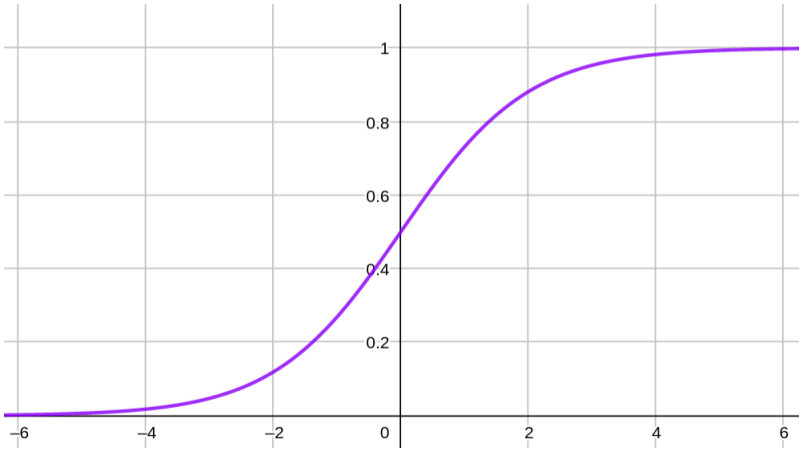

Figura 3.8: Gráfica de la función sigmoide

La ecuación 4.9 significa que, tras el entrenamiento, la red neuronal puede aprender automáticamente un prefactor que determina la importancia relativa de los 12 vecinos más cercanos. Para el NaCl (ver figura 3.6), los 6 vecinos más cercanos tendrán un prefactor cercano a 1, y los 6 restantes tendrán un prefactor cercano a 0. Para el KCl, los 8 vecinos más cercanos tendrán un prefactor cercano a 1, y los 4 restantes tendrán un prefactor cercano a 0. Este prefactor se utiliza entonces para controlar el flujo de información en el proceso de convolución del grafo.

Agregando este prefactor el paso de mensajes neuronales queda de la siguiente forma:

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + \sum_{j,k\in\mathcal{N}_{(i)}} \sigma\left(\mathbf{z}_{(i,j)_k}\mathbf{W}_f^{(t)} + \mathbf{b}_f^{(t)}\right) \odot g\left(\mathbf{z}_{(i,j)_k}\mathbf{W}_s^{(t)} + \mathbf{b}_s^{(t)}\right) \tag{3.11}$$

donde $\mathbf{z}_{(i,j)k} = \mathbf{v}_i^{(t)} \oplus \mathbf{v}_j^{(t)} \oplus \mathbf{e}_{(i,j)_k}$ denota la concatenación del estado oculto del nodo

central $\mathbf{v}_i^{(t)}$ (en el paso t), del nodo vecino $\mathbf{v}_j^{(t)}$ y de la arista que los une. Además, σ y g denotan las funciones sigmoide y *softplus*, respectivamente.

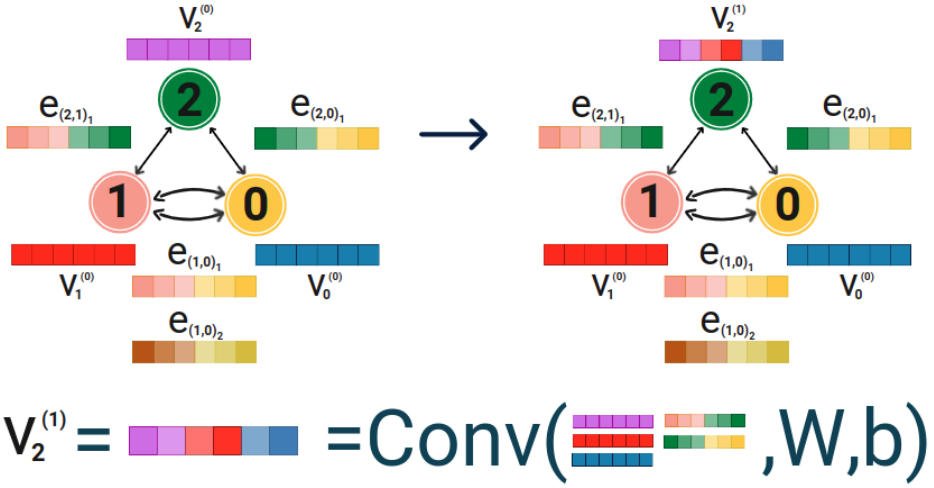

Figura 3.9: Actualización del estado oculto $\mathbf{v}_2^{(0)}$ del átomo 2 dentro del paso de mensajes neuronales definido por la Ecuación 4.12

De igual forma para entender mejor este nuevo paso de mensajes neuronales se ilustra la actualización de un átomo que la función definida como *conv* toma como entrada de tal forma que se actualiza a partir de:

$$\mathbf{v}_2^{(0)} + \sigma\left(\left(\mathbf{v}_2^{(0)} \oplus \mathbf{v}_0^{(0)} \oplus \mathbf{e}_{(2,0)_1} + \mathbf{v}_2^{(0)} \oplus \mathbf{v}_1^{(0)} \oplus \mathbf{e}_{(2,1)_1}\right)\mathbf{W}_f^{(0)} + \mathbf{b}_f^{(0)}\right) \odot g\left(\left(\mathbf{v}_2^{(0)} \oplus \mathbf{v}_0^{(0)} \oplus \mathbf{e}_{(2,0)_1} + \mathbf{v}_2^{(0)} \oplus \mathbf{v}_1^{(0)} \oplus \mathbf{e}_{(2,1)_1}\right)\mathbf{W}_s^{(0)} + \mathbf{b}_s^{(0)}\right) \quad (3.12)$$

3.4.2. Capa de agrupación

Después de T convoluciones, la red aprende automáticamente el estado oculto final $\mathbf{v}_i^T$ para cada átomo incluyendo iterativamente su entorno local. La capa de agrupación se utiliza entonces para producir un estado oculto global $\mathbf{v}_C$ para el cristal que puede representarse mediante una función de agrupación (figura 3.10). En el modelo se utiliza la agrupación global como función de agrupación

$$\mathbf{v}_C = \frac{1}{|V|}\sum_{i\in V}\mathbf{v}_i^T \tag{3.13}$$

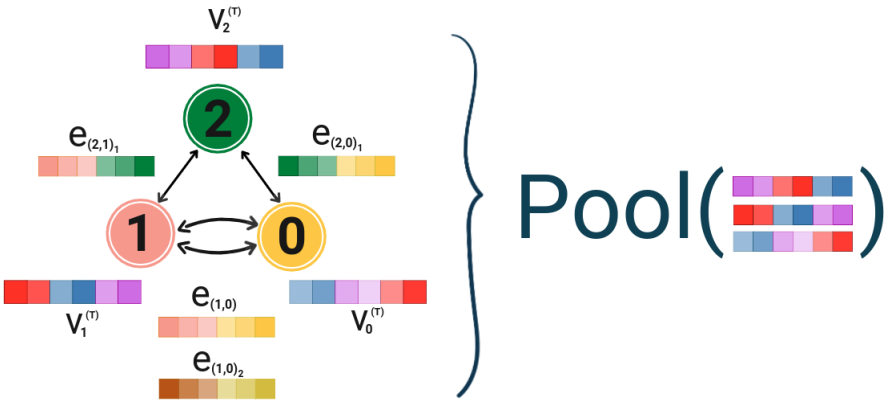

Figura 3.10: Función de agrupación

La función satisface la invarianza permutacional con respecto a los índices de los átomos y la invarianza de tamaño con respecto a la celda unitaria de elección.

Además de las capas convolucionales y de agrupación dos capas ocultas totalmente conectadas con profundidades L_1 y L_2 se añaden para capturar la compleja relación entre la estructura y las propiedades del cristal (figura 3.11).

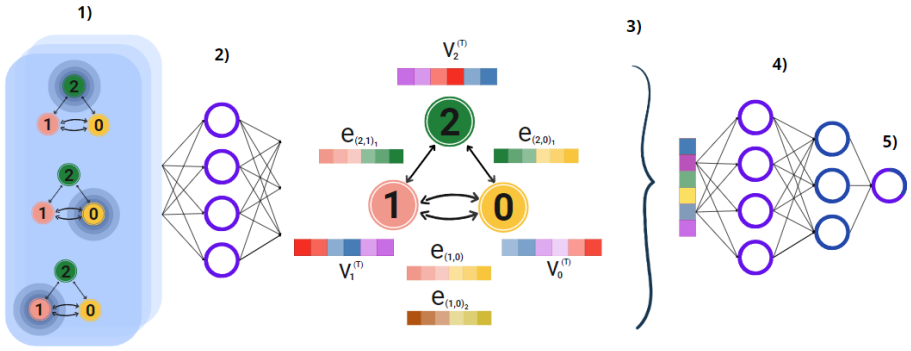

Figura 3.11: Arquitectura del modelo. 1) Capas de convolución donde se aplica el paso de mensajes neuronales. 2) Capa totalmente conectada L_1 aplicada a cada nodo, dando como resultado a un nuevo grafo en el que cada nodo representa el entorno local de cada átomo. 3) Capa de agrupación dando como resultado un vector que representa toda la estructura cristalina. 4) Capa totalmente conectada L_2. 5) Capa de salida para proporcionar la predicción.

3.5. Entrenamiento

El algoritmo del modelo se entrena de forma supervisada. Por cada cristal hay una propiedad objetivo correspondiente. En el artículo [10] se predicen 7 propiedades cristalinas: energía de formación, energía absoluta, brecha de banda de energía, energía de Fermi, módulo de masa, módulo de cizallamiento, relación de Poisson. Sin embargo en este trabajo se desea predecir únicamente la brecha de banda de energía debido a que la conductividad eléctrica de un semiconductor depende en gran medida de esta propiedad.

Se inicializan los pesos y sesgos de forma aleatoria, los datos de entrenamiento se pasan al modelo y se predice la brecha de banda de energía de cada cristal. Se calcula la diferencia entre la brecha de banda de energía predicha y el valor objetivo a partir de la función de costo. La función que se emplea en el modelo es la conocida como error medio absoluto definida de la siguiente manera:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{y}^{(i)} - y^{(i)}\right| \tag{3.14}$$

A partir de esta función se utiliza el algoritmo de optimización llamado Descenso por el Gradiente Estocástico para actualizar los parámetros de la red. El algoritmo de Descenso por el Gradiente Estocástico es una modificación simple del Descenso por el Gradiente. En el Descenso por el Gradiente Estocástico, se calcula el gradiente utilizando sólo una pequeña parte aleatoria de las observaciones en lugar de todas ellas. En algunos casos, este enfoque puede reducir el tiempo de cálculo.

Capítulo 4

Código

En resumen, dado un cristal (archivo *cif*), se construye su grafo a partir de las distancias y la celda unitaria (considerando su periodicidad), posteriormente se añade información a los nodos con propiedades atómicas y a las aristas con la expansión gaussiana de la distancia. El grafo se le pasa al modelo y mediante las convoluciones, la agrupación, las redes neuronales totalmente conectadas y el algoritmo del descenso por el gradiente estocástico puede utilizarse para entrenar los pesos y sesgos que luego se ocuparán para predecir la brecha de banda de energía de nuevos cristales similares.

En esta parte del trabajo se presenta el código donde se implementan todos los conceptos descritos en el capítulo anterior con el objetivo de predecir la brecha de banda de energía de cristales a partir de redes neuronales en grafos. El código está escrito en `Python 3.6` y fue probado en un ambiente con las siguientes paqueterías:

- `Pymatgen`
- `Pytorch`
- `Numpy`

Se usaron las paqueterías de `networkx` para visualizar la construcción de los grafos y `tensorboard` para visualizar el entrenamiento.

El código del modelo está distribuido entre tres programas principales llamado `data.py`, `model.py` y `main.py`. El primero de ellos contiene todas las clases que permiten procesar los datos del archivo tipo *cif* para así obtener el grafo aunado a la información a nivel atómica y a nivel de enlace. El segundo programa define las convoluciones en grafos y con ello la arquitectura global del modelo. El tercero es en donde se implementa el entrenamiento junto con los métodos para crear una interfaz adecuada al usuario para usar el código. A continuación se detalla la funcionalidad de cada uno de estos programas.

4.1. Código para el procesamiento de los datos

En el programa `data.py` existen 3 clases que contribuyen al procesamiento de los datos.

La primera clase se llama `GaussianDistance`, la cual permite expandir la matriz de las distancias entre cada par de átomos a partir de las funciones de base gaussiana.

$$\hat{\mathbf{d}}_{ij} = \left[e^{-\frac{(d-(\mu_{\text{mín}}+k\Delta\mu))^2}{2\sigma^2}} \right]_{0\leq k\leq \frac{\mu_{\text{máx}}}{\Delta\mu}} \tag{4.1}$$

Dentro de la clase primero se inicializan los parámetros que definen a la expansión gaussiana (Ecuación 5.1):

- `dmin`: $\mu_{\text{máx}}$
- `dmax`: $\mu_{\text{mín}}$
- `step`: $\Delta\mu$
- `var`: σ

```python
class GaussianDistance(object):
    def __init__(self, dmin, dmax, step, var=None):
        assert dmin < dmax
        assert dmax - dmin > step
        self.filter = np.arange(dmin, dmax+step, step)
        if var is None:
            var = step
        self.var = var

    def expand(self, distances):
        return np.exp(-(distances[..., np.newaxis] - self.filter)**2 /
                      self.var**2)
```

Figura 4.1: Expansión Gaussiana

La segunda clase es `AtomCustomJSONInitializer`, la cual toma como entrada el archivo tipo `.json` en donde vienen a manera de diccionario los vectores de características de los átomos, y precisamente la clase se encarga de leer el archivo y crear un diccionario de `python` para poder generar los vectores de características de los átomos.

La tercera clase es `CIFData`, la cual toma como entrada un directorio el cual debe contener la siguiente estructura:

- `root_dir`
  `id_prop.csv`
  `atom_init.json`
  `id0.cif`
  `id1.cif`

```
37  class AtomCustomJSONInitializer(AtomInitializer):
38      def __init__(self, elem_embedding_file):
39          with open(elem_embedding_file) as f:
40              elem_embedding = json.load(f)
41          elem_embedding = {int(key): value for key, value
42                            in elem_embedding.items()}
43          atom_types = set(elem_embedding.keys())
44          super(AtomCustomJSONInitializer, self).__init__(atom_types)
45          for key, value in elem_embedding.items():
46              self._embedding[key] = np.array(value, dtype=float)
```

Figura 4.2: Inicialización de los vectores de características de los átomos

En donde `root_dir` es la ruta del directorio en donde se encuentran los siguientes archivos:

- `id_prop.csv`: Archivo de texto en formato tipo *.csv* ("valores separados por comas"), el cual debe contener el identificador del cristal en la primera columna y la propiedad objetivo en la segunda columna, en este trabajo se utiliza la brecha de banda de energía.
- `atom_init.json`: Archivo de texto en formato tipo *.json*, el cual tiene un diccionario en donde la llave representa el valor del número atómico y su correspondiente valor es el vector de características del átomo.
- `id0.cif`: Archivo de texto en formato tipo *.cif*, el cual contiene toda la información del cristal relacionado con el identificador `id0`, los siguientes archivos son de este formato para los cristales a considerar para el modelo.

Además de la ruta del directorio, se deben inicializar otros parámetros para procesar los datos:

- `root_dir`: La ruta del directorio raíz del conjunto de datos.
- `max_num_nbr`: El número máximo de vecinos al construir el grafo del cristal.
- `radius`: El radio de corte para la búsqueda de vecinos.
- `random_seed`: Semilla aleatoria para barajar el conjunto de datos.
- `dmin`: La distancia mínima para construir la `GaussianDistance`.
- `step`: El tamaño del paso para construir la `GaussianDistance`.

```python
class CIFData(Dataset):
    def __init__(self, root_dir, max_num_nbr=12, radius=8, dmin=0, step=0.2,
                 random_seed=123):
        self.root_dir = root_dir
        self.max_num_nbr, self.radius = max_num_nbr, radius
        id_prop_file = os.path.join(self.root_dir, 'id_prop.csv')
        with open(id_prop_file) as f:
            reader = csv.reader(f)
            self.id_prop_data = [row for row in reader]
        random.seed(random_seed)
        random.shuffle(self.id_prop_data)
        atom_init_file = os.path.join(self.root_dir, 'atom_init.json')
        self.ari = AtomCustomJSONInitializer(atom_init_file)
        self.gdf = GaussianDistance(dmin=dmin, dmax=self.radius, step=step)

    def __len__(self):
        return len(self.id_prop_data)

    @functools.lru_cache(maxsize=None)  # Cache loaded structures
    def __getitem__(self, idx):
        cif_id, target = self.id_prop_data[idx]
        crystal = Structure.from_file(os.path.join(self.root_dir,
                                                   cif_id+'.cif'))
        atom_fea = np.vstack([self.ari.get_atom_fea(crystal[i].specie.number)
                              for i in range(len(crystal))])
        atom_fea = torch.Tensor(atom_fea)
        all_nbrs = crystal.get_all_neighbors(self.radius, include_index=True)
        all_nbrs = [sorted(nbrs, key=lambda x: x[1]) for nbrs in all_nbrs]
        nbr_fea_idx, nbr_fea = [], []
        for nbr in all_nbrs:
                nbr_fea_idx.append(list(map(lambda x: x[2], nbr)) +
                                   [0] * (self.max_num_nbr - len(nbr)))
                nbr_fea.append(list(map(lambda x: x[1], nbr)) +
                               [self.radius + 1.] * (self.max_num_nbr -
                                                     len(nbr)))
            else:
                nbr_fea_idx.append(list(map(lambda x: x[2],
                                            nbr[:self.max_num_nbr])))
                nbr_fea.append(list(map(lambda x: x[1],
                                        nbr[:self.max_num_nbr])))
        nbr_fea_idx, nbr_fea = np.array(nbr_fea_idx), np.array(nbr_fea)
        nbr_fea = self.gdf.expand(nbr_fea)
        atom_fea = torch.Tensor(atom_fea)
        nbr_fea = torch.Tensor(nbr_fea)
        nbr_fea_idx = torch.LongTensor(nbr_fea_idx)
        target = torch.Tensor([float(target)])
        return (atom_fea, nbr_fea, nbr_fea_idx), target, cif_id
```

Figura 4.3: Procesamiento de datos

4.2. Código del modelo

En el programa `model.py` existen 2 clases que contribuyen a la implementación del modelo.

La clase donde se definen las convoluciones en grafos es `ConvLayer`. Los parámetros que toma como entrada son:

- `orig_atom_fea_len`: Dimensión del estado oculto de los átomos.
- `nbr_fea_len`: Dimensión del vector de características de los enlaces.

La clase define la función de convolución:

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + \sum_{j,k\in\mathcal{N}_{(i)}} \sigma\left(\mathbf{z}_{(i,j)_k}\mathbf{W}_f^{(t)} + \mathbf{b}_f^{(t)}\right) \odot g\left(\mathbf{z}_{(i,j)_k}\mathbf{W}_s^{(t)} + \mathbf{b}_s^{(t)}\right) \tag{4.2}$$

Como se muestra en la figura 4.4 se define primero el tensor $\mathbf{z}_{(i,j)k} = \mathbf{v}_i^{(t)} \oplus \mathbf{v}_j^{(t)} \oplus \mathbf{e}_{(i,j)_k}$. Posteriormente se le aplica la transformación lineal a todo el tensor $\mathbf{z}_{(i,j)k}$ y luego se aplica una normalización por lotes que consiste en restarle la media y dividir por la varianza a cada conjunto de observaciones (lotes) que se generan debido al uso del descenso por el gradiente estocástico. Después se divide el tensor entre dos para así aplicar las funciones correspondientes. Al primero se le aplica la función sigmoide obteniendo así el prefactor de importancia y al segundo se le aplica la función *softplus* para obtener así la información de los vecinos del átomo (así como del mismo). Posteriormente se multiplican estas dos columnas, se suman para obtener un vector donde esté la información global y se aplica nuevamente una normalización por lotes. Finalmente se suma este vector al estado oculto anterior para así actualizarlo.

La clase principal se llama `CrystalGraphConvNet`, que es en donde se define toda la arquitectura del modelo. Los parámetros que toma como entrada son:

- `orig_atom_fea_len`: Dimensión del vector de características de los átomos.
- `nbr_fea_len`: Dimensión del vector de características de los enlaces.
- `atom_fea_len`: Dimensión del estado oculto de los átomos (Los que se pasaran a la convolución).
- `n_conv`: Número de capas convolucionales.
- `h_fea_len`: Dimensión del estado oculto de los átomos después de aplicar la función de agregación.
- `n_h`: Número de capas ocultas de tipo después de la agregación.

Los pasos que se ejecutan en dicha clase (figura 4.5) son los siguientes:

1. Se aplica una transformación lineal con el objetivo de reducir la dimensión del vector de características de los átomos.

2. Se aplican las convoluciones.
3. Se aplica la función de agrupación para obtener un vector global del cristal.
4. Se aplica una transformación lineal seguido de la función *softplus* al vector global del cristal.
5. Se aplican las redes neuronales totalmente conectadas.
6. Se aplica una transformación lineal para tener de salida un solo valor.

```python
import torch
import torch.nn as nn

class ConvLayer(nn.Module):
    def __init__(self, atom_fea_len, nbr_fea_len):

        super(ConvLayer, self).__init__()

        #inicializar los parámetros y las funciones a ocupar
        self.atom_fea_len = atom_fea_len
        self.nbr_fea_len = nbr_fea_len
        self.fc_full = nn.Linear(2*self.atom_fea_len+self.nbr_fea_len,
                                 2*self.atom_fea_len)
        self.sigmoid = nn.Sigmoid()
        self.softplus1 = nn.Softplus()
        self.bn1 = nn.BatchNorm1d(2*self.atom_fea_len)
        self.bn2 = nn.BatchNorm1d(self.atom_fea_len)
        self.softplus2 = nn.Softplus()

    def forward(self, atom_in_fea, nbr_fea, nbr_fea_idx):

        #Se construye el tensor nodo central, nodo vecino e información de enlace
        N, M = nbr_fea_idx.shape
        atom_nbr_fea = atom_in_fea[nbr_fea_idx, :]
        total_nbr_fea = torch.cat(
            [atom_in_fea.unsqueeze(1).expand(N, M, self.atom_fea_len),
             atom_nbr_fea, nbr_fea], dim=2)

        #Se aplica la transformación lineal
        total_gated_fea = self.fc_full(total_nbr_fea)
        total_gated_fea = self.bn1(total_gated_fea.view(
            -1, self.atom_fea_len*2)).view(N, M, self.atom_fea_len*2)

        #Se divide en dos tensores
        nbr_filter, nbr_core = total_gated_fea.chunk(2, dim=2)

        #Se deriva el prefactor de importancia de vecinos
        nbr_filter = self.sigmoid(nbr_filter)

        #Se aplica la función de activación
        nbr_core = self.softplus1(nbr_core)

        #Se multiplican ambos tensores y se suman
        nbr_sumed = torch.sum(nbr_filter * nbr_core, dim=1)
        nbr_sumed = self.bn2(nbr_sumed)

        #Se actualiza el estado oculto
        out = self.softplus2(atom_in_fea + nbr_sumed)
        return out
```

Figura 4.4: Convolución

```
class CrystalGraphConvNet(nn.Module):

    def __init__(self, orig_atom_fea_len, nbr_fea_len,
                 atom_fea_len=64, n_conv=3, h_fea_len=128, n_h=1):

        super(CrystalGraphConvNet, self).__init__()

        #Se inicializan las funciones a ocupar
        self.embedding = nn.Linear(orig_atom_fea_len, atom_fea_len)
        self.convs = nn.ModuleList([ConvLayer(atom_fea_len=atom_fea_len,
                                              nbr_fea_len=nbr_fea_len)
                                    for _ in range(n_conv)])
        self.conv_to_fc = nn.Linear(atom_fea_len, h_fea_len)
        self.conv_to_fc_softplus = nn.Softplus()
        if n_h > 1:
            self.fcs = nn.ModuleList([nn.Linear(h_fea_len, h_fea_len)
                                      for _ in range(n_h-1)])
            self.softpluses = nn.ModuleList([nn.Softplus()
                                             for _ in range(n_h-1)])
        self.fc_out = nn.Linear(h_fea_len, 1)

    def forward(self, atom_fea, nbr_fea, nbr_fea_idx, crystal_atom_idx):

        #Paso 1
        atom_fea = self.embedding(atom_fea)

        #Paso 2
        for conv_func in self.convs:
            atom_fea = conv_func(atom_fea, nbr_fea, nbr_fea_idx)

        #Paso 3
        crys_fea = self.pooling(atom_fea, crystal_atom_idx)

        #Paso 4
        crys_fea = self.conv_to_fc(self.conv_to_fc_softplus(crys_fea))
        crys_fea = self.conv_to_fc_softplus(crys_fea)

        #Paso 5
        if hasattr(self, 'fcs') and hasattr(self, 'softpluses'):
            for fc, softplus in zip(self.fcs, self.softpluses):
                crys_fea = softplus(fc(crys_fea))

        #Paso 6
        out = self.fc_out(crys_fea)
        return out

    def pooling(self, atom_fea, crystal_atom_idx):

        assert sum([len(idx_map) for idx_map in crystal_atom_idx]) ==\
            atom_fea.data.shape[0]
        summed_fea = [torch.mean(atom_fea[idx_map], dim=0, keepdim=True)
                      for idx_map in crystal_atom_idx]
        return torch.cat(summed_fea, dim=0)
```

Figura 4.5: Modelo

4.3. Código para el entrenamiento

El código para el entrenamiento se encuentra en el archivo `main.py`, ahí se define una función llamada `train` la cual toma como parámetros los siguientes:

- `train_loader`: La clase de los datos para entrenar que se genera a partir de `Pytorch`.
- `model`: El modelo una vez inicializado.
- `criterion`: La función de costo que se ocupa para el entrenamiento.
- `optimizer`: El algoritmo para optimizar los parámetros.
- `epoch`: El número de épocas que se calcularán.
- `normalizer`: La función para normalizar los valores a predecir.

La función empieza mandando los datos a la GPU en caso de contar la computadora con ella. Posteriormente normaliza la propiedad a predecir, es decir la brecha de banda de energía (es importante normalizar para una convergencia más rápida en la función de pérdida dentro del entrenamiento) y calcula la primera predicción del modelo. Ya con esos dos vectores y mediante la función de costo se calcula la perdida del modelo. Para medir la precisión del modelo se calcula el error medio absoluto entre los valores predichos y el valor real. Luego se saca el gradiente de la función y se realiza un paso del algoritmo de optimización, en este caso se utiliza el algoritmo de descenso por el gradiente estocástico. Finalmente se establece que se impriman los valores de pérdida, precisión y tiempo para cada época.

De igual forma se define la función de `validate` sólo que toma como entrada los datos de validación.

En la función `main` se realizan los siguientes pasos para juntar todos los archivos.

- Se le pide al usuario ingresar todos los datos necesarios para crear el modelo.
- Se leen y procesan los datos de los cristales con la función de CIFData que se importa del archivo `data.py`
- Con la función `get_train_val_test_loader` se dividen los datos para entrenamiento, validación y prueba.
- Se construye el modelo a partir de los datos ingresados por el usuario.
- Se entrena el modelo y finalmente predice la propiedad en los datos de prueba.

4.4. Resultados

En todo el proceso se ocuparon 46 741 cristales de la base de datos, 40 000 se destinaron para el entrenamiento, 4 000 para los datos de validación y 2 741 para datos de prueba (aproximadamente una separación de datos de 85 % entrenamiento, 8 % validación y 7 % prueba). En la figura 4.6 se observa la gráfica del error obtenidas durante el entrenamiento. Dado que la propiedad objetivo es la brecha de banda de energía y esta propiedad tiene unidades de energía en electronvoltios (eV), el error se mide en electronvoltios (eV). Se observa que el error en los datos de entrenamiento disminuye rápidamente durante las primeras 100 épocas y luego se estabiliza. Por otro lado, el error en los datos de validación sigue un patrón similar, pero se mantiene ligeramente por encima del error de entrenamiento. Esto sugiere que el modelo está tendiendo a sobreajustarse a los datos de entrenamiento. Para poder solucionar este problema se requieren mas datos de entrenamiento.

El error promedio absoluto (MAE) del modelo en datos de prueba (es decir en datos que aun no ha visto el modelo) es de 0.39 eV. Para tener un valor de referencia el MAE obtenido con un método de modelización de mecánica cuántica computacional llamado teoría funcional de la densidad (DFT) es de 0.6 eV [49] con respecto a los datos experimentales, sin embargo es importante mencionar que la eficacia del modelo esta determinada por los datos de entrenamiento que se obtienen a partir del método de DFT, por lo que no se puede comparar directamente el modelo con dicho método.

Cuando el MAE de los datos de prueba es significativamente menor a la desviación estándar del conjunto total de datos indica que las predicciones del modelo se ajustan estrechamente a los valores reales en los datos de prueba con lo cual el modelo es capaz de generalizar bien más allá de los datos de entrenamiento mitigando el ruido presente de los datos. Se observa que el MAE del modelo es de 0.39 eV, mientras que la desviación estándar de los datos es de 1.66 eV. Estos resultados reflejan una precisión y confiabilidad significativas en el modelo, lo cual respalda el logro del objetivo principal de este trabajo.

En la tabla de la figura 4.7 se muestran 6 cristales de diferente complejidad (figura 4.8) dentro del conjunto de datos de prueba, el promedio de error de las 6 muestras es de 0.33 eV (lo que es esperable considerando que en el conjunto total es de 0.39 eV) sin embargo para los 3 cristales cuya brecha de banda de energía calculada es cercana a 0 eV (metales) el error del modelo es mucho mayor que de los 3 cristales restantes (no metales), de manera general se obtiene un error promedio en todos los metales de 0.49 eV mientras que en los no metales se obtiene 0.23 eV, por lo que el modelo predice mucho mejor en los no metales.

Por otro lado, el tiempo de entrenamiento fue alrededor de 4 horas. En el último apartado se mencionan algunas sugerencias con las que se podría reducir el tiempo de entrenamiento así como métodos para mejorar el modelo.

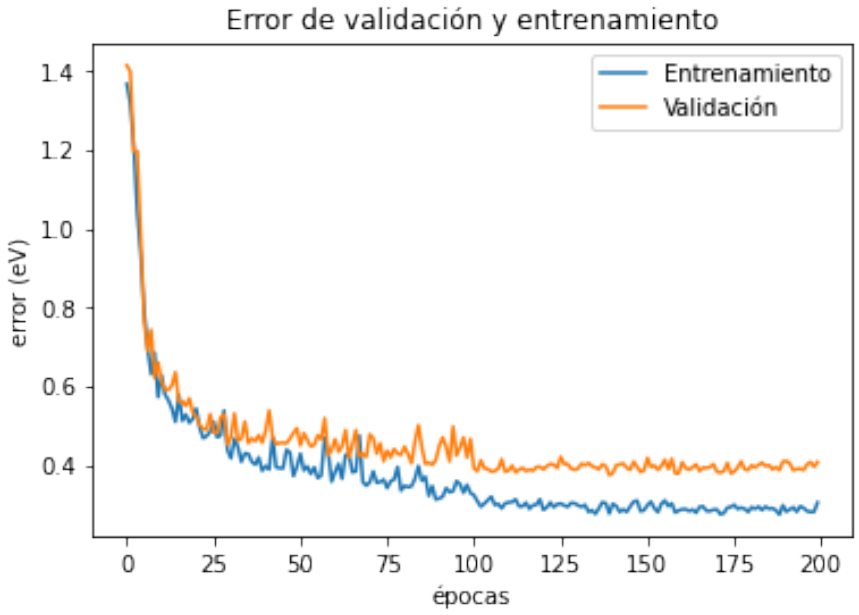

Figura 4.6: Gráfica del error con datos de entrenamiento y validación

Fórmula química del cristal	Brecha de banda de energía calculada con DFT	Brecha de banda de energía predicha por el modelo	Diferencia entre el valor real y el predicho	Numero de átomos en la celda unitaria	Metal
TiAl	0.0eV	0.53eV	0.53eV	2	Sí
Ca_2FeN_2	0.0eV	0.72eV	0.72eV	20	sI
Li_5LaO_4	3.4eV	3.67eV	0.27eV	80	No
$Zn_4As_2O_9$	1.45eV	1.65eV	0.2eV	120	No
$Sr_{14}Al_8Ge_3$	0.06eV	0.37eV	0.31eV	150	Si
$Li_2Mn_4P_4O_{15}$	3.04eV	3.046eV	0.006eV	200	No

Figura 4.7: Resultados de una muestra de cristales

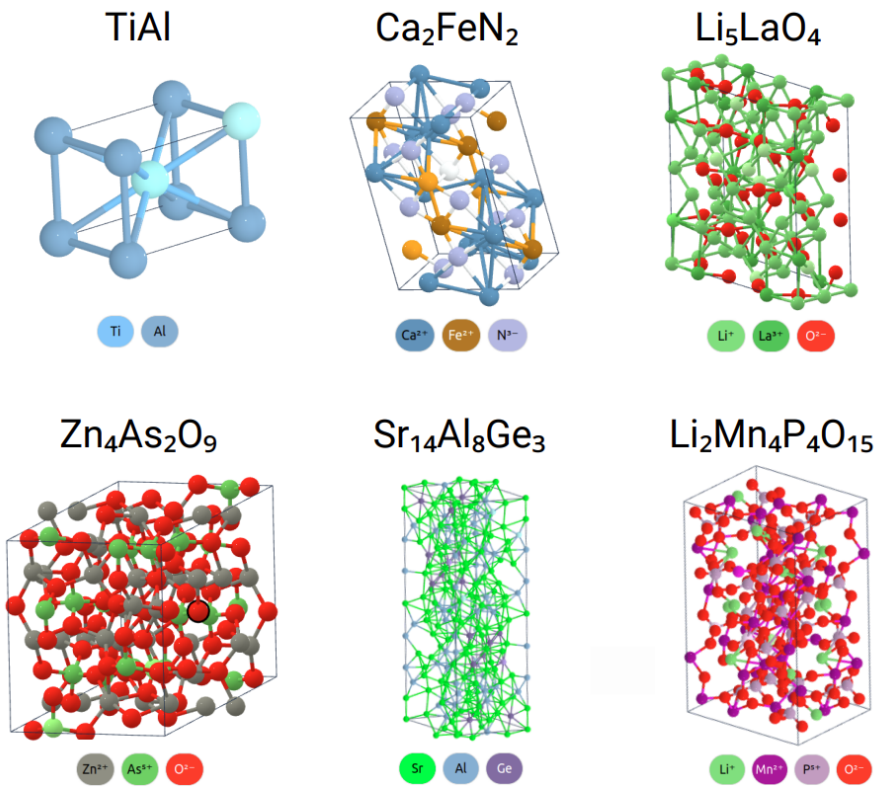

Figura 4.8: Celdas unitarias de los cristales

Conclusiones y trabajo futuro

Los marcos de aprendizaje profundo basados en grafos son una herramienta poderosa en el descubrimiento de materiales funcionales, identificando correlaciones entre la estructura y la propiedad del material y con ello haciendo predicciones eficientes de propiedades. Al representar a las distribuciones atómicas de los materiales mediante grafos y diseñando adecuadamente las estrategias de paso de mensajes, las arquitecturas de aprendizaje profundo adquieren un nuevo nivel de flexibilidad, eficiencia y fiabilidad hacia el aprendizaje mejorado del principio físico de las distribuciones de los materiales. En esta tesis se presentó uno de los primeros modelos reportados de redes neuronales en grafos para la predicción de propiedades de estructuras cristalinas [10], el cual demostró buena precisión en la predicción de la propiedad de la brecha de banda de energía de las estructuras cristalinas.

El total de cristales ocupados durante todo el procedimiento fue de 46 741, desafortunadamente para los materiales inorgánicos de estado sólido no se dispone de una gran cantidad de datos (al menos por ahora) lo que limita al modelo bajando la precisión de predicción. Otro reto está relacionado con los costes del entrenamiento del modelo ya que un solo paso del marco de mensajes neuronales en el grafo requiere $O(n^2d^2)$ multiplicaciones con valores de tipo 32-flotante (valores decimales con hasta 32 dígitos), donde n es el número de nodos y d es la dimensión de los vectores de los nodos. Esto es costoso desde el punto de vista computacional para grandes estructuras cristalinas. Para superar este problema de escalabilidad, el desarrollo de marcos de aprendizaje basados en grafos se ha enfocado en la construcción de novedosas arquitecturas de red que permitan la incorporación de principios físicos como las simetrías y las topologías.

Derivado de ello, se han realizado trabajos en donde la arquitectura del modelo se ha modificado con el objetivo de aumentar la precisión y reducir los tiempos de entrenamiento. Entre las modificaciones que se han sugerido se incluye una mejor representación química optimizando los enlaces interatómicos en los grafos del cristal teselando el espacio cristalino con los poliedros de Voronoi y con ello definiendo un nuevo algoritmo en la construcción del grafo [50].

Además de realizar modificaciones en la arquitectura de la red, una estrategia adicional para mejorar la precisión consistiría en categorizar inicialmente los materiales en metales y no metales. Esto podría permitir una mejor rendimiento del modelo en los conjuntos de datos y de igual forma se podrían explorar otras propiedades relevantes

que ayuden a mejorar la categorización de los datos.

Así mismo se podría mejorar los tiempos de entrenamiento utilizando librerıas como CuPy, Numba o CUDA (nvidia, 2021) con las que se puedan llevar a cabo cálculos de manera paralela, lo cual tiene el potencial de acelerar drásticamente los cálculos cuando se ejecute en múltiples GPUs ya que el modelo solo se puede ejecutar en una sola GPU.

Bibliografía

1. He, K., Zhang, X., Ren, S. y Sun, J. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition,* 770-778 (2016).
2. Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z. *et al.* A deep learning approach to antibiotic discovery. *Cell,* 688-702 (2020).
3. Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A. y Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine,* 18-42 (2017).
4. Wu, Y., Lian, D., Xu, Y., Wu, L. y Chen, E. Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelligence,* 1054-1061 (2020).
5. Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R. y Battaglia, P. Graph networks as learnable physics engines for inference and control. *International Conference on Machine Learning,* 4470-4479 (2018).
6. Fout, A., Byrd, J., Shariat, B. y Ben-Hur, A. Protein interface prediction using graph convolutional networks. *Advances in neural information processing systems* (2017).
7. Bessadok, A., Mahjoub, M. A. y Rekik, I. Graph neural networks in network neuroscience. *arXiv preprint arXiv:2106.03535* (2021).
8. Cheng, G., Gong, X.-G. y Yin, W.-J. Crystal structure prediction by combining graph network and optimization algorithm. *Nature,* 1-8 (2022).
9. Pandey, S., Qu, J., Stevanović, V., John, P. S. y Gorai, P. Predicting energy and stability of known and hypothetical crystals using graph neural network. *Patterns,* 100361 (2021).
10. Xie, T. y Grossman, J. C. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters,* 145301 (2018).
11. Klicpera, J., Groß, J. y Günnemann, S. Directional message passing for molecular graphs. *International Conference on Learning Representations* (2020).

12. Schütt, K., Kindermans, P.-J., Sauceda Felix, H. E., Chmiela, S., Tkatchenko, A. y Müller, K.-R. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems* (2017).

13. Rupp, M., Tkatchenko, A., Müller, K.-R. y Von Lilienfeld, O. A. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical review letters,* 058301 (2012).

14. Hansen, K., Biegler, F., Ramakrishnan, R., Pronobis, W., Von Lilienfeld, O. A., Muller, K.-R. y Tkatchenko, A. Machine learning predictions of molecular properties: Accurate many-body potentials and nonlocality in chemical space. *The journal of physical chemistry letters,* 2326-2331 (2015).

15. Huo, H. y Rupp, M. Unified representation of molecules and crystals for machine learning. *Machine Learning: Science and Technology,* 045017 (2017).

16. Gong, W. y Yan, Q. Graph-based deep learning frameworks for molecules and solid-state materials. *Computational Materials Science,* 110332 (2021).

17. Rani, M., Nayak, R. y Vyas, O. An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. *Knowledge-based systems,* 33-48 (2015).

18. Mahesh, B. Machine learning algorithms-a review. *International Journal of Science and Research,* 381-386 (2020).

19. Abraham, A. *Artificial neural networks* cap. 129 (John Wiley & Sons, Ltd Chichester, UK, 2005).

20. Malsburg, C. v. d. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms* 245-248 (Springer, 1986).

21. Cao, Z., Dan, Y., Xiong, Z., Niu, C., Li, X., Qian, S. y Hu, J. Convolutional neural networks for crystal material property prediction using hybrid orbital-field matrix and magpie descriptors. *Crystals,* 191 (2019).

22. Gori, M., Monfardini, G. y Scarselli, F. A new model for learning in graph domains. *Proceedings of the IEEE international conference on neural networks,* 729-734 (2005).

23. Hamilton, L. W. en *Synthesis Lectures on Artificial Intelligence and Machine Learning* 51-70 (Morgan Claypool publishers, 2020).

24. Kipf, T. N. y Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

25. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. y Bengio, Y. Graph attention networks. *stat,* 20 (2017).

26. Vinyals, O., Bengio, S. y Kudlur, M. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391* (2015).

27. Hochreiter, S. y Schmidhuber, J. Long short-term memory. *Neural computation,* 1735-1780 (1997).

28. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
29. Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics,* 1-17 (1964).
30. Duchi, J., Hazan, E. y Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* (2011).
31. Tieleman T., H. G. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning,* 26-31 (2012).
32. Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).
33. Kingma, D. P. y Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
34. Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. y Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems* (2015).
35. Rogers, D. y Hahn, M. Extended-connectivity fingerprints. *Journal of chemical information and modeling,* 742-754 (2010).
36. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences,* 31-36 (1988).
37. Huang, B. y Von Lilienfeld, O. A. Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity. *The Journal of Chemical Physics,* 161102 (2016).
38. De Graef, M. y McHenry, M. E. *Structure of materials: an introduction to crystallography, diffraction and symmetry* (Cambridge University Press, 2012).
39. Faber, F. A., Hutchison, L., Huang, B., Gilmer, J., Schoenholz, S. S., Dahl, G. E., Vinyals, O., Kearnes, S., Riley, P. F. y Von Lilienfeld, O. A. Prediction errors of molecular machine learning models lower than hybrid DFT error. *Journal of chemical theory and computation,* 5255-5264 (2017).
40. Schütt, K. T., Glawe, H., Brockherde, F., Sanna, A., Müller, K.-R. y Gross, E. K. How to represent crystal structures for machine learning: Towards fast prediction of electronic properties. *Physical Review B,* 205118 (2014).
41. Sands, D. E. *Introduction to crystallography* (Courier Corporation, 1993).
42. Jain, A. y Bligaard, T. Atomic-position independent descriptor for machine learning of material properties. *Physical Review B,* 214112 (2018).
43. Behler, J. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *The Journal of chemical physics,* 074106 (2011).
44. Bartók, A. P., Kondor, R. y Csányi, G. On representing chemical environments. *Physical Review B,* 184115 (2013).

45. Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R. y Tkatchenko, A. Quantum-chemical insights from deep tensor neural networks. *Nature,* 1-8 (2017).

46. Unke, O. T. y Meuwly, M. PhysNet: A neural network for predicting energies, forces, dipole moments, and partial charges. *Journal of chemical theory and computation,* 3678-3693 (2019).

47. Jain, A, Ong, S., Hautier, G, Chen, W, Richards, W., Dacek, S, Cholia, S, Gunter, D, Skinner, D, Ceder, G *et al.* The Materials Project: A Materials Genome Approach to Accelerating Materials Innovation. *APL Materials* (2013).

48. Hellenbrandt, M. The inorganic crystal structure database (ICSD)—present and future. *Crystallography Reviews,* 17-22 (2004).

49. Jain, A., Hautier, G., Moore, C. J., Ong, S. P., Fischer, C. C., Mueller, T., Persson, K. A. y Ceder, G. A high-throughput infrastructure for density functional theory calculations. *Computational Materials Science,* 2295-2310 (2011).

50. Park, C. W. y Wolverton, C. Developing an improved crystal graph convolutional neural network framework for accelerated materials discovery. *Physical Review Materials,* 063801 (2020).