# UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

## FACULTAD DE INGENIERÍA

### SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

---

# APROXIMACIÓN DE PROPIEDADES FISICOQUÍMICAS BASADO EN UN ENFOQUE DE *MESSAGE PASSING NEURAL NETWORK*

POR:

**AARÓN LEONARDO VELÁZQUEZ RUIZ**

**TESIS PRESENTADA COMO REQUISITO PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA EN COMPUTACIÓN**

CHIHUAHUA, CHIH., MÉXICO 7 DE ABRIL DEL 2022

Esta pagina se dejó intencionalmente en blanco.

# Dedicatoria

Para Maritrini, Mariana, Aarón, Irene y Almendra.

# Agradecimientos

Agradezco a Facultad de Ingeniería de la Universidad, los profesores de la Maestría en Ingeniería en Computación, en especial a todos aquellos con los que tome clase. A mis sinodales por tomarse el tiempo de leer este trabajo y ayudarme con él. Agradezco a Maritrini por brindarme su apoyo emocional e intelectual cada que lo necesité, sin ella esto jamás habría sido posible. Por último y no por ello menos importante (todo lo contrario) agradezco a la doctora Graciela no sólo por los conocimientos compartidos, sino también en especial por su amistad y paciencia.

# Resumen

Esta investigación se condujo con la finalidad de analizar propiedades fisicoquímicas de sistemas moleculares utilizando un paradigma de la inteligencia artíficial conocido como *Machine Learning*. Existen varios métodos para encontrar observables escalares en sistemas moleculares, sin embargo en los últimos años los investigadores de varias subramas de la física han optado por utilizar métodos en su totalidad computacionales ya sea por lo maleable de estos modelos para aceptar diferentes tipos de datos y por los excelentes resultados encontrados.

El objetivo de este trabajo de tesis fue diseñar y experimentar con bloques neuronales basados en estado del arte. Se propuso utilizar diferentes sistemas de *self-attention* como principal inovación al momento de complemetar un algoritmo de naturaleza *Message Passing*. Se demostró que la arquitectura propuesta aprende observables fisicoquímicos utilizando la naturaleza de las estucturas moleculares.

# Índice

# 1. Introducción

## 1.1. Mecánica Cuántica y sistemas moleculares

La mecánica cuántica es una rama de la física que nace en las primeras décadas del siglo pasado [22]. La entonces naciente área del conocimiento se dedicó al estudio de los cuerpos elementales de la materia. Dentro de sus fundamentos y postulados teóricos se encuentra la ecuación de Schrödinger [32], esta ecuación nace para encontrar la evolución dinámica de la partícula o dicho de otra manera; resolver el problema electrónico de un sistema.

Al encontrar, contraintuitivos pero exitosos resultados para la dinámica de partículas o sistemas elementales, era natural que los investigadores se interesaran en resolver problemas más complejos. Sin embargo, entre más complejos los sistemas a resolver, fue más necesario el recurrir a métodos de aproximación. Algunos ejemplos de estos métodos son: Born-Oppenheimer, Hartree-Fock y *Density Functional Theory* [7],[28],[38].

En la actualidad, el cálculo de observables en sistemas moleculares, cuenta con una amplia gama de aplicaciones en campos de química, ciencia de materiales, biología molecular, diseño de fármacos, entre otros [26],[27]. Sin embargo, el costo computacional que estas tareas requieren, ha limitado la exploración teórica y experimental.

## 1.2. *Machine Learning* y Máquinas de predicción molecular

Aproximadamente cien años después de la revolución científica que se logró de la mano de las ciencias físicas, se empezó a vivir un nuevo boom de aciertos científicos y tecnológicos; esta vez, de la mano de un campo interdisciplinario conocido como aprendizaje autómatico o *Machine Learning* [24]. Esta disciplina consiste en construir algoritmos para optimizar un criterio de rendimiento utilizando datos como experiencia pasada [1]. De manera general, el aprendizaje máquina se ha definido como el campo

de estudio que dan a las computadoras la habilidad de aprender sin explícitamente ser programadas para dicha tarea [30].

En general, el estado del arte, dentro del aprendizaje automático ha guiado a los investigadores interdisciplinarios al escoger técnicas y algoritmos en la resolución de problemas. Las redes convolucionales se investigan profundamente en la primera parte de la década pasada [20],[37], [14], [41]. El éxito de este tipo de modelos computacionales, se vió reflejado en varias subramas de las ciencias físicas, desde la fisicoquímica, estado sólido e incluso bases teóricas de la física cuántica [4], [34].

Otro acercamiento de aprendizaje automático conocido como *Message Passing* o *Graph Neural Network* usualmente se encuentra referenciado en la literatura de las maquínas de predicción de propiedades moleculares [40], [31], [12]. En este acercamiento se aprovecha la facilidad de algunos sistemas de ser representados como estructuras grafo y sus propiedades geométricas para alimentar redes neuronales.

En 2017, una investigación dentro del área del procesamiento de lenguaje natural (*NLP*) presenta un mecanismo de atención llamado *self-attention* que se vuelve parteaguas en la disciplinas del aprendizaje máquina [44]. Dicho mecanismo se ha adaptado para diferentes aplicaciones dentro de *NLP* [5],[39]. Sin embargo se han encontrado resultados prometedores en distintas áreas como lo son cómputo visual [10],[25],[48] y redes neuronales gométricas [45], [42], por mencionar algunos ejemplos.

Motivados en la creciente popularidad de los mecanismos de atención en varias de las áreas del aprendizaje automático, y a su vez, el problema continuo por conocer propiedades fisicoquímicas en sistemas atómicos, en la presente investigación se propone un doble sistema de atención (*self-attention* y *cross-attention*) para crear un bloque de convolución continuo (utilizando un acercamiento de aprendizaje *Message Passing* ) el cual será utilizado como herramienta principal dentro de una arquitectura capaz de encontrar propiedades escalares de diferentes sistemas atómicos. A dicho modelo se le llamó *Multi-Head Message Passing* o por simplicidad *M-H MP*.

## 1.3. Objetivos general y específicos

El objetivo general el presente trabajo, es encontrar mediciones aproximadas a los resultados de experimentos físicos de sistemas de varios cuerpos utilizando modelos de aprendizaje máquina.

Dentro de los objetivos específicos se proponen los siguientes:

- Lograr modelar interacciones cuánticas utilizando redes neuronales profundas.
- Predecir propiedades moléculares utilizando ambientes químicos virtuales.
- Probar elementos del estado del arte para competir con los resultados de las redes publicadas y los observables documentados.

# 2. Antecedentes y estado del arte

Las técnicas de aprendizaje máquina no solamente han conquistado la industria [11], también se han inmiscuido en otras áreas del conocimiento, formando en los últimos años fuertes alianzas en áreas como la física y la química [3]. Para estas últimas dos áreas el resolver la estructura electrónica para moléculas y materiales es de fundamental importancia. Sin embargo, desde que la teoría cuántica se creyó casi completa, hace casi noventa años, pioneros de la teoría microscópica notaron la necesidad de métodos de aproximación que permitan encontrar soluciones a sistemas complejos, ya que pocos y muy sencillos sistemas (a nivel molecular), podían generar gran complicación matemática [8].

En los últimos años varios trabajos se han publicado cuyas soluciones a sistemas cuánticos se aproximan por métodos conocidos en las áreas de inteligencia artificial ,utilizando un enfoque parecido; creando una base de datos por un método *ab-initio* de partículas elementales o moléculas, para luego entrenar computacionalmente un algoritmo cuya finalidad es encontrar la física detrás de estos sistemas para después utilizar estos resultados con sistemas cuya resolución desde cero se vuelve altamente costosa computacionalmente.

En el artículo *Recurrent neural network wave functions* [16] se utiliza una red neuronal recurrente para representar varias funciones de onda de sistemas de varios cuerpos, optimizando los parámetros variacionales con un acercamiento estocástico. Los investigadores demuestran la efectividad de las redes neuronales recurrentes al calcular propiedades de varios sistemas cuánticos de interés en el área de materia condensada.

En la investigación llamada *Machine learning phases of matter* [4], se demostró que la tecnología desarrollada para aplicaciones como *computer vision* y el procesamiento de lenguaje natural, puede usarse para codificar fases de la materia y discriminar transiciones de fase en sistemas de varios cuerpos. Además se mostró que las redes neuronales permiten codificar información básica de fases no convencionales co-

mo las presentes en el modelo *squared ice.*

Investigadores de los institutos de *Perimeter*, *Vector Institute* y *Center for Computational Quantum Physics* encuentran que los métodos de *Quantum State Tomography* o *QST* resueltos por fuerza bruta requiere una gran capacidad de cómputo. Sin embargo, utilizando un acercamiento por aprendizaje máquina se pueden resolver problemas de *QST* de estados altamente entrelazados con más de mil quibits, con un alto nivel de exactitud [43].

Investigadores de la Universidad de California, en San Diego y del Instituto de Estudios Avanzados en la Universidad Tsinghua [47], muestran una arquitectura que puede desarrollar automáticamente los conceptos de función de onda y encontrar la ecuación de movimiento de Schrödinger utilizando datos provenientes de simulaciones experimentales. Este método de aprendizaje contiene una máquina traductora, cuyo trabajo es realizar un mapeo entre el potencial efectivo de una partícula cuántica y su función de onda utilizando un *auto-encoder* con la finalidad de extraer la información necesaria del traductor; dicha información resulta ser la función de onda y la ecuación de Schrödinger.

También se publicó por investigadores del departamento de Física de la Universidad de Ontario el artículo llamado *Deep learning and the Schrödinger equation* [27], donde se entrenó una red neuronal convolucional para predecir las energías basales de un electrón en cuatro diferentes clases de potenciales bidimensionales. El modelo fue capaz de predecir las energías bases dentro de una exactitud química, con una error de media absoluta de 1.49 mHa.

Una rama cuyas bases se encuentran en la física y química fundamental es la espectroscopía, que además está dentro de las herramientas más poderosas que los científicos tienen a su disposición para conocer la materia a niveles átomicos y moleculares. Estas técnicas espectroscopicas pueden identificar moléculas desconocidas, medir longitudes de enlaces y hasta es posible encontrar constantes de fuerza relacionados con los enlaces químicos [9]. Todas estas propiedades, se basan en transiciones que ocurren entre

diferentes estados energéticos de la materia al momento de interacturar con radiación electromagnética, dicha interacción materia y radiación es posible entenderla dentro de las leyes de la mecánica cuántica.

Investigaciones más recientes, han utilizado modelos y arquitecturas de aprendizaje máquina más complejos e inovadores para crear redes profundas capaces de modelar interacciones cuánticas de varios cuerpos bajo diferentes condiciones de laborario [13]. Arquitecturas capaces de utilizar como herramienta redes profundas con la finalidad de encontar elementos más complejos de los sistemas como la misma función de onda [33], e incluso se han utilizado ya, con los mismos propósitos modelos del estado del arte, como las *graph neural networks* [36].

Las investigaciones anteriores nos guiaron a realizar un trabajo que consistió en utilizar una base de datos artificial que permite medir observables como la energía total de sistemas cuánticos de varios cuerpos utilizando información de estos sistemas, como las cargas nucleares de los átomos que las componen, así como su estuctura topologíca.

Se entrenó un modelo que logró realizar un mapeo entre la estructura documentada de los sistemas moléculares y propiedades básicas de éstos, como: la energía basal, momentos dipolares, energía libre, constantes rotacionales, entre otras. Con el fin de hacer un trabajo más robusto se compararon estos resultados con los reportados por las arquitecturas propuestas en el estado el arte logrando resultados competitivos.

Es posible modelar las interacciones de los sistemas cuánticos de varios cuerpos, utilizando una arquitectura de aprendizaje automático con el fin de aproximar propiedades observables en dichos sistemas creando un mapeo entre descripción nuclear y observables.

En otras palabras, el problema que esta investigación aborda, es el de encontrar una aproximaciones con un error de relatidad química [2] en las propiedades intrínsecas de los sistemas cuánticos de varios cuer-

pos, utilizando la descripción atomística de sistemas reales, los cuales se han encontrado por métodos químicos ab-initio.

# 3. Marco teórico.

## 3.1. Redes neuronales artificiales.

En este apartado se describe el algoritmo básico de las redes neuronales, el cual se puede partir en cuatro partes: propagación, cómputo del error, retropropagación y entrenamiento

## 3.2. Propagación.

La idea básica del aprendizaje profundo, se inspira en las neuronas biológicas de los seres vivos. La neurona McCullin-Pitts se creó en 1943, y fue el primer modelo matemático basado en una neurona, el cual se encargaba de tareas de clasificación. Dicho modelo acepta valores binarios, procesa estos valores de entrada y utiliza un valor umbral, el cual, es encargado de la decisión del valor resultante.

El segundo modelo de cómputo matemático, basado en la neurona biológica se atribuye a Frank Rosenblatt en 1958, este fue llamado Perceptrón. Dicho modelo, es una mejora de la neurona Mcullin-Pitts, donde las entradas no se limitan a un valor binario, sino que, pueden ser cualquier valor dentro del conjunto de los números reales, y cada una de ellas es multiplicada por un valor, conocido como peso. La suma de todos los productos entrada-peso se contrasta con un valor umbral, el cual decide, si el resultado o salida del modelo es un valor positivo o negativo (resultado binario).

Suponiendo un valor umbral igual a $T$, la ecución del Perceptrón se escribe como,

$$y = \begin{cases} 1 & \text{if} \quad x_i w_i \geq T \\ 0 & \text{if} \quad x_i w_i > T. \end{cases} \tag{1}$$

Donde se encuentren variables con el mismo subíndice en forma de producto, durante este trabajo deno-

tará una suma discreta, con límites de cero hasta $n \in \mathbb{R}$, esto es

$$x_i w_i = \sum_{i=0}^{n} x_i w_i. \tag{2}$$

A menos que se indique lo contrario, las variables $x_i$, se utilizarán para denotar los valores de entrada, mientras que, las variables $w_i$ definen los pesos de una red o modelo. Las variables con subíndice cero, se utilizarán para abreviar la notación del término *bias* o sesgo de la neurona, donde $x_0 = 1$ y $w_0 = b$, siendo $b$ , dicho término.

Además de un espectro más amplio en los valores de entrada, el modelo Perceptrón cuenta con una ventaja importante en contraste con la neurona McCullin-Pitts; tener la capacidad de aprender, a través de la actualización o corrección de un error, siendo este, dependiente en primera instancia de la respuesta de salida del Perceptrón. Esto permite crear un hiperplano que divide o clasifica los datos de entrada en dos clases, cada vez que se realiza una actualización del error, el hiperplano se actualiza, logrando una mejor separación de los valores de entrada.

Gracias a la naturaleza líneal de las operaciones realizadas por la máquina Perceptrón, es natural que la separación de clases lograda por ella sea líneal y que falle cuando la distribución de los datos de entrada sea de cualquier orden diferente a este.

Recién se menciona, que la neurona Perceptrón aprende una separación lineal de dos clases en un conjunto de valores de entrada dado un error en el valor de salida. En este momento es normal preguntarse, ¿Cómo se logra dicho aprendizaje? Resulta que, cuanto más grande sea el valor peso que multiplica una entrada, mayor será el impacto de esa entrada (su clase) en la clasificación. La operación de aprendizaje puede escribirse,

$$w_{\text{actualizado}} = w_{\text{anterior}} + \eta \delta x, \tag{3}$$

donde $\delta$, es el error o función error (estos dos nombres serán utilizados indistintivamente a través de esta

investigación), y para el caso del Perceptrón se define,

$$\delta = \hat{y}_i - y_i. \tag{4}$$

La variable $y_i$, con acento circunflejo o simplemente $\hat{y}_i$, es la clase real del valor de entrada $x_i$. Y la letra $\eta$, se le conoce como rango de entrenamiento, del cual se hablará con más detalle en las siguientes secciones de la investigación, por el momento es suficiente mencionar que los valores asignados a esta variable suelen oscilar entre $0 < n \leq 1$ y su fin es acelerar el proceso de aprendizaje.

Finalmente, cada vez que el Perceptrón actualiza sus pesos y por lo tanto, actualiza el hiperplano encargado de separar clases, se máximiza la separación en ambos lados de las clases. Sin embargo, esto no deja de ser una separación lineal, de una distribución binaria en un conjunto de entradas, en otras palabras se tiene la solución a un problema básico y poco frecuente en la vida real.

Con el fin de poder enfrentar problemas más complejos, las máquinas Perceptrón, neuronas o nodos (como se llamará indistintivamente en esta investigación), se utiliza una función de activación, que se opera después de encontrar el producto pesos-entradas,

$$y = \phi\left(w_i x_i\right), \tag{5}$$

sin olvidar que el primer producto de esa suma discreta pertenece al término de sesgo $b$. Conservando la nomenclaura de los modelos anteriores, la única diferencia, hasta el momento será la función de activación $\phi$ **no lineal**, a la suma de productos.

En la actualidad, las redes neuronales modernas, sulen componerse de un numero $l$ de “capas”, donde cada una de ellas se forma de “conjunto apilado” neuronas, es por ello que a estos modelos se les conoce como MLP, por sus nombre en inglés *Multilayer Perceptron*.

Un MLP es un modelo de las llamadas *Artificial Neural Networks* o por simplicidad ANN, y el entendimiento de este, es una piedra angular en esta investigación. Por lo tanto, a fin de desarrollar los siguientes conceptos, se empleará la siguiente Figura 1.

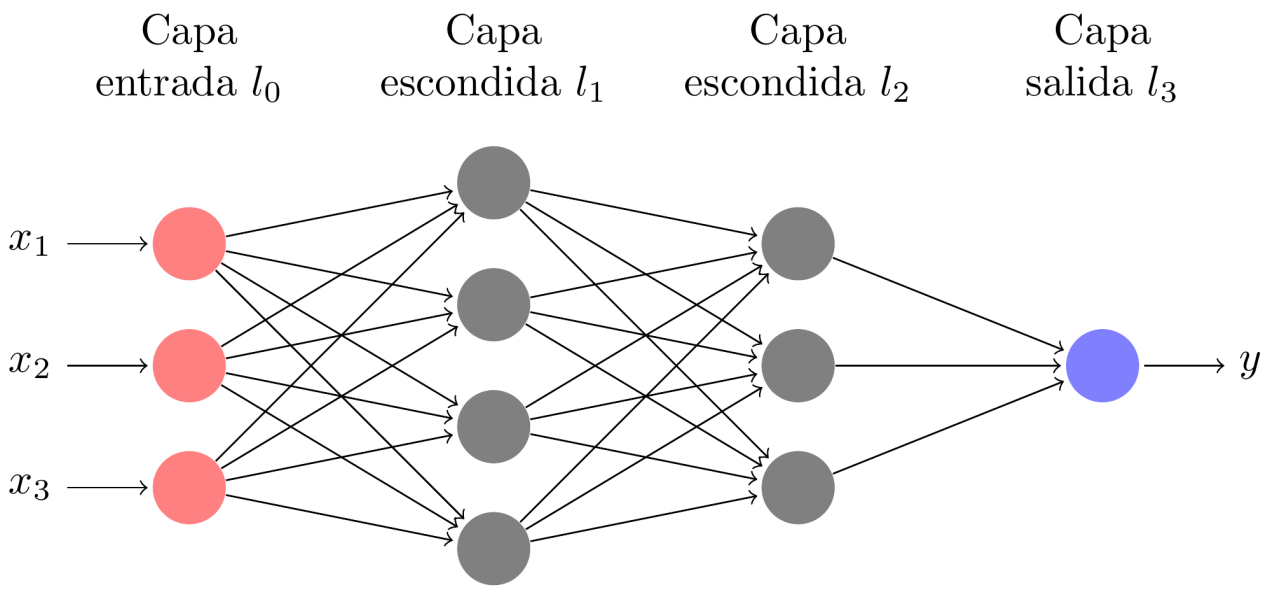


Figura 1: Red neuronal artificial prealimentada, totalmente conectada.

Las redes MLP se componen de tres tipos diferentes de capas: entrada, capas escondidas y salida. Donde la primer y última capa, pueden exister una sola vez dentro de una red, mientras que las llamadas capas escondidas, pueden existir $l$ número de ellas.

Para generalizar este modelo, se supone una red donde el número de neuronas en la capa de entrada es $x_i \in \mathbb{R}^d$ , $L$ cantidad de capas, $N$ nodos por capa y con una función de activación para $L-1$ capas, $\phi : \mathbb{R} \to \mathbb{R}$.

Al conocer las operaciónes que se llevan a cabo en el modelo Perceptrón, es posible extrapolar ese conocimiento a una MLP. Para la primera capa escondida, en específico, la $i-$ésima neurona se calcula de la siguiente manera,

$$h_i^{[1]} = \phi^{[1]}\left(w_{i,j}^{[1]} x_j\right), \tag{6}$$

es imperativo recordar que el término $w_{i,0}^{[1]} x_0 = b_i^{[1]}$. La operación para la iésima neurona, de la siguiente

capa escondida se forma como,

$$h_i^{[2]} = \phi^{[2]}\left(w_{i,j}^{[2]} h_j^{[1]}\right), \tag{7}$$

finalmente, el resultado $y$, en la capa de salida viene dado por,

$$y = \phi^{[3]}\left(w_{i,j}^{[3]} h_j^{[2]}\right). \tag{8}$$

Antes de continuar, se generalizará el cálculo de la $i-$ésima neurona en la $l-$ésima capa escondida,

$$h_i^{l} = \phi^{[l]}\left(w_{i,j}^{[l]} h_j^{[l-1]}\right), \tag{9}$$

Es costumbre en el gremio de investigadores de inteligencia artificial, expresar los cálculos de forma vectorial o por capas, en vez de neuronas, por ende, es posible reescribir las ecuaciónes anteriores de la siguiente manera:

$$\mathbf{h}^{[1]} = \phi^{[1]}\left(\mathbf{W}^{[1]}\mathbf{x}\right), \tag{10}$$

$$\mathbf{h}^{[l]} = \phi^{[l]}\left(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]}\right), \tag{11}$$

$$\mathbf{y} = \phi^{l+1}\left(\mathbf{W}^{[l+1]}\mathbf{h}^{[l]}\right). \tag{12}$$

Donde se ha descrito explicitamente, la primera capa escondida y la capa de salida, mientras que se ha generalizado la ecuación (8) para la $l-$ésima capa escondida. Antes de avanzar, se hará una igualdad entre ambas notaciones, para la $l-$ésima capa escondida,

$$\begin{aligned} h_i^{[l]} &= \left[h_1^{[l]}, \ldots, h_n^{[l]}\right] \\ &= \phi^{[l]}\left(\begin{bmatrix} w_{1,1}^{[l]} & \cdots & w_{1,n}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n,1}^{[l]} & \cdots & w_{n,n}^{[l]} \end{bmatrix} \begin{bmatrix} h_1^{[l-1]} \\ \vdots \\ h_n^{[l-1]} \end{bmatrix} + \begin{bmatrix} b_1^{[l]} \\ \vdots \\ b_n^{[l]} \end{bmatrix}\right) \\ &= \phi^{[l]}\left(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]}\right) = \phi^{[l]}\left(\mathbf{z}^{[l]}\right). \end{aligned} \tag{13}$$

Donde la contracción de notación $\mathbf{z} = \mathbf{W}^{[l]}\mathbf{h}^{[l-1]}$, suele usarse por comodidad. Es importante mencionar, un par de resultados que se han mencionado como iguales sin embargo, en la ecuación (8) se denota el resultado de la red MLP como un escalar y en (11) como un vector, esto se debe a que, en la última ecuación se ha generalizado a un número de clasificación $k$ correspondiente a $k$ clases diferentes.

## 3.3. Función de activación.

Una manera simple pero intuitiva de describir a las funciones de activación es pensar en ellas como una compuerta lógica entre un nodos de diferentes capas; basicamente, estas "compuertas" son las encargadas de decidir si la $i-$ésima neurona debe activarse o no, dentro de una capa.

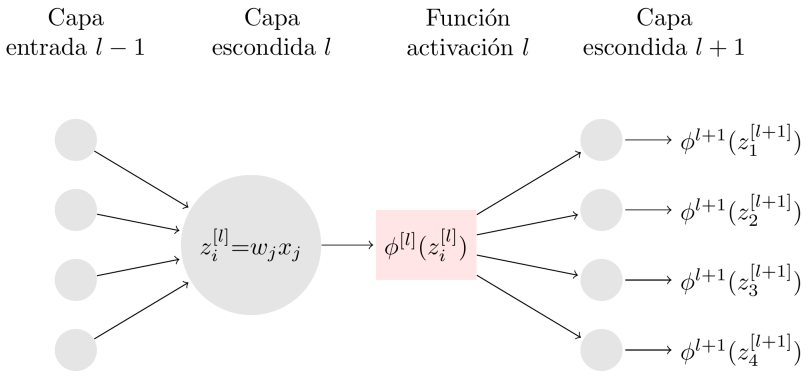


Figura 2: Funcionamiento interno de una neurona.

La Figura 2 separa en dos partes las dos operaciones realizadas dentro de la $i-$ésima neurona, dentro de una capa escondida $l$. Al no agregar una función de activación en las capas de una red, los pesos y el *bias* por si solos, realizan una transformación linear y nada más. Es por ello que las funciones de activación más usadas son funciones no lineales, ya que con ellas es posible modelar por medio del aprendizaje iterativo funciones complejas, y así encontrar predicciones más precisas.

Las funciones de activación no lineales, atacan los siguientes problemas

1. Permiten utilizar un acercamiento de retropropagación, ya que permiten cálcular una primera derivada de dicha función.

2. A diferencia de las funciones lineales, es viable apilar un número $l$ de capas escondidas dentro de una red, logrando el modelado apróximado de cualquier función.

De las siguientes gráficas, es posible visualizar el porque una función constante, sin importar la transformación lineal que tome como argumento, no sirve como una función de activación en una neurona; al calcular el gradiente (derivada), este siempre se desvanecerá.

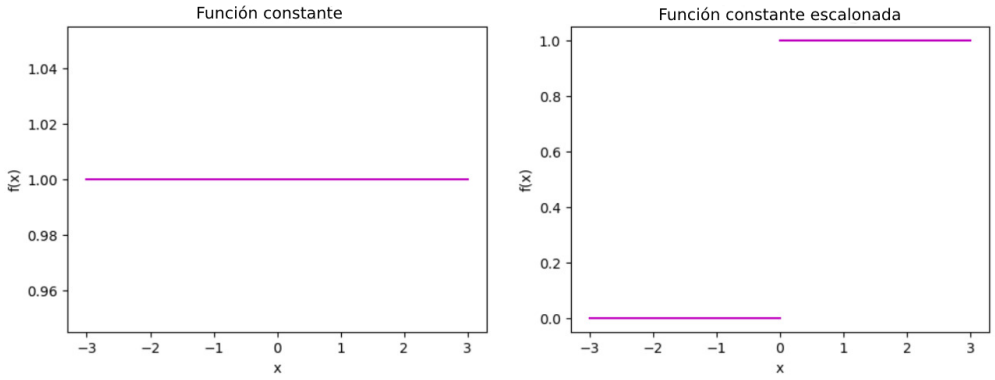


Figura 3: Funciones lineales.

Por lo tanto, se buscan funciones de activación que aporten algo más significativo en el proceso de entrenamiento de una red neuronal.

### 3.3.1. Función sigmoide

Una función sigmoide se utiliza con frecuencia dentro del cómputo profundo como activación neuronal, esta función se define como:

$$\sigma(x) = \frac{1}{1 + \exp[-x]} \tag{14}$$

La razón principal de la popularidad de esta función, es que limita el mapeo de la transformación lineal o argumento a tomar valores dentro del cero y el uno.

Si bien es válido argumentar que una función escalonada marginiza de la misma manera, una sigmoide cuenta con una primera derivada continua y bien comportada. Además se encuentra naturalmente centrada en el origen, siendo así una función que permite crear límite de decisión para tareas de clasificación binarias.

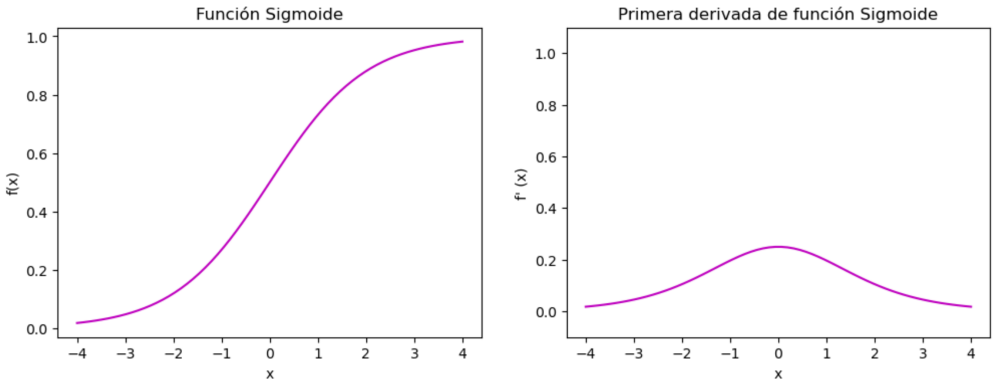


Figura 4: Función sigmoide.

### 3.3.2. ReLU

La función ReLU por las siglas en inglés de *Rectified linear unit*, se define como,

$$\text{ReLU}(x) = \begin{cases} 0 & x < 0 \\ \textbf{max}(x) & x \geq 0. \end{cases} \tag{15}$$

Cuya derivada es también una función escalonada, con valor unitario cuando la variable $x$ es mayor o igual a cero o toma el valor de cero en cualquier otro caso

$$\text{ReLU}'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0. \end{cases} \tag{16}$$

A continuación se gráfica las funciones recién definidas

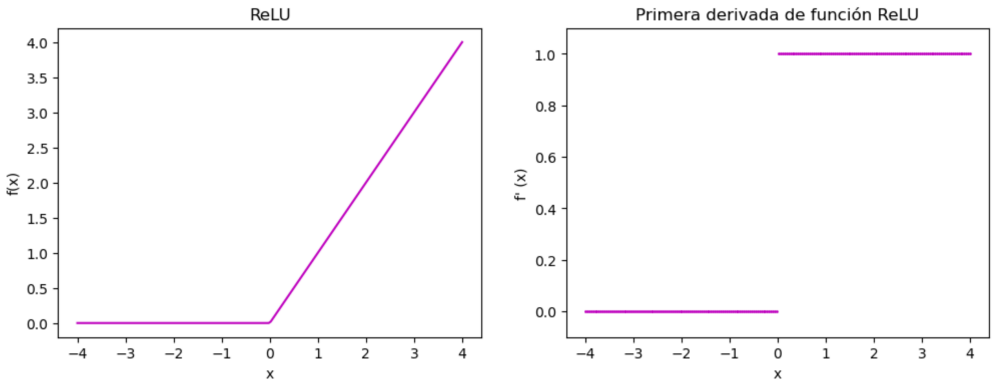


Figura 5: Función ReLU.

Los buenos resultados de esta función en la práctica, la han vuelto popular dentro de las arquitecturas de las redes neuronales. Muestra una más rápida convergencia en comparación con la función sigmoide, además de ser más eficiente computacionalmente [21].

### 3.3.3. Función softmax

Derivado el concepto de la función sigmoide, se generaliza una función que permite una separación multiclase denominada, *softmax*, la cual se define de la siguiente manera,

$$f(x_i) = \frac{\exp[x_i]}{\sum_j^K \exp[x_j]}. \tag{17}$$

Esta función permite que la salida de una red sea una distribución de probabilidad categorizable en $K$ número de clases. En otras palabras, al usar *softmax* como función de activación se produce un vector de probabilidades de acuerdo con la transformación lineal de una neurona.

Como ejemplo, se imagina la operación lineal en la última capa de una red la cual tiene como tarea, una clasificación triple

$$\mathbf{o}^l = \mathbf{W}[h_1^l, h_2^l, h_3^l]^T = \begin{bmatrix} 4.0 \\ 2.3 \\ -1.7 \end{bmatrix}, \tag{18}$$

al utilizar la función softmax sobre este vector, se encuentra

$$Softmax(\mathbf{o}^l) = \begin{bmatrix} 0.8412 \\ 0.1537 \\ 0.0051. \end{bmatrix} \tag{19}$$

Esta función mapea, los valores numéricos de salida en una capa hacia una distribución de probabilidad a lo largo de un número $K$ de clases.

Sin duda, las 3 funciones de activación mencionadas no son las únicas utilizadas en la práctica o encontradas en la literatura, sin embargo, si son de las más utilizadas, además de ser parte de esta investigación.

## 3.4. Costo o función de perdida.

El segundo paso en el algoritmo de predicción utilizando el modelo de redes neuronales, es encontrar el error en el paso progresivo recién descrito o simplemente error en la predicción $y_i$. Hasta el momento se ha mencionado más de una vez, que las entradas $\mathbf{I}_i = \{x_1, \cdots, x_n\}$ describen una instancia u objeto $\hat{y}_i$, la cual suele identificarse con el nombre de etiqueta. La relación que existe entre la predicción y la etiqueta de la instancia, crea un medio para reajustar los parametros "entrenables " del modelo y así, eventualmente, lograr predicciones más acertadas. Existen varios métodos para cuantificar el error obtenido en el paso progresivo en una MLP, cada uno de ellos utilizados en tareas específicas dentro de las disciplinas de aprendizaje máquina. Por convención se utiliza la variable $J(W, W_0)$, para definir la función de costo, pérdida o error.

### 3.4.1. Error cuadrático medio

Una de las funciones de pérdida más usadas dentro de la estadística y el analísis de regresión es el error cuadrático medio. Esta función, se alimenta de dos vectores tal que $\hat{\mathbf{y}}, \mathbf{y} \in \mathbb{R}^N$, donde $N$ es la cantidad instancias disponibles para el entrenamiento. Además, es importante hacer hincapié en el mapeo logrado

al alimentar esta función

$$MSE : \mathbb{R}^N \to \mathbb{R}. \tag{20}$$

Por definición, el error cuadrático medio es,

$$MSE = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\mathbf{y}} - \mathbf{y}||^2 . \tag{21}$$

Ya que la suma, o resta de dos vectores $N$ dimensionales, es otro vector en el espacio $\mathbb{R}^N$, se define un nombre arbitario **E** a la sustracción de la ecuación anterior. La norma de este vector es,

$$||\mathbf{E}|| = \sqrt{\sum_{i,j=1}^{N} E_i E_j \mathbf{e}_i \mathbf{e}_j \delta_{i,j}} = \sqrt{\sum_{i=1}^{N} E_i E_i}. \tag{22}$$

Donde la función $\delta_{i,j}$ se define como,

$$\delta_{i,j} = \begin{cases} 1, & \text{si} \quad i = 1 \\ 0 & \text{otro caso.} \end{cases} \tag{23}$$

Ya que los vectores $\mathbf{e}_{i,j}$, son vectores unitarios, el $i-$ésimo producto entre ellos es unitario. Demostrando así que la función error cuadrático medio mapea de $\mathbb{R}^N \to \mathbb{R}$.

Hay dos excelentes razones para utilizar el error cuadrático medio, como función de costo dentro de un algoritmo de aprendizaje máquina: se asegura manejar valores positivos todo el tiempo, y al estar elevando al cuadrado cada uno de los pares etiqueta-predicción, se hace una diferencia más marcada entre errores grandes y pequeños.

Seguramente, se buscará otras funciones de costo o alguna variación de la función ya descrita en instancias posteriores de la investigación.

## 3.5. Retropropagación.

La retropropagación es el mecanismo usado para la actualización en los parametros de un modelo de redes neuronales, el cual utiliza un algoritmo de optimización conocido como descenso de gradiente; se calcula el gradiente de la función de costo con respecto los parametros o pesos de una red. Este proceso empieza en la salida o predicción del modelo, y atraviesa las capas escondidas $l$, $l-1, \cdots$ , hasta llegar a la primer capa escondida.

Para ejemplificar este proceso, se utiliza la red de la Figura 1, junto con una función de costo genérica $\mathbf{J}(\mathbf{W}, \mathbf{W}_0)$ y se utilizará una función de activación $\phi$.

Como ya se ha mencionado, el algoritmo de retropropagación empieza, por encontrar el gradiente de los parametros en la última capa. Los pesos que influyeron directamente en la capa de salida de nuestra red, son aquellos encontrados en la capa $\mathbf{W}^{[3]}$, por ello se busca la derivada parcial de $J$ con respecto a dichos pesos,

$$\frac{\partial J}{\partial \mathbf{W}^{[3]}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}}, \tag{24}$$

Siguiendo el camino en reversa de la red (parte de *backward*), la actualización en los pesos de la capa escondida esta dado por

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}} = \frac{\partial J}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{h}^{[2]}} \frac{\partial \mathbf{h}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}, \tag{25}$$

para concluir, es necesario encontrar el gradiente de los pesos en la capa de entrada $l_1$ :

$$\frac{\partial J}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathbf{J}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{h}^{[2]}} \frac{\partial \mathbf{h}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{h}^{[1]}} \frac{\partial \mathbf{h}^{[1]}}{\partial \mathbf{I}} \frac{\partial \mathbf{I}}{\partial \mathbf{W}^{[1]}}. \tag{26}$$

Tomando la dirección negativa de gradiente, se empieza a minimizar la función de costo al actualizar los pesos de la siguiente manera,

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \eta \frac{\partial J}{\partial \mathbf{W}^{[l]}}. \tag{27}$$

## 3.6. Entrenamiento.

Sí bien, hasta el momento ya se han actualizado los parametros entrenables, es muy seguro que dichos parametros se encuentren lejos de poder minimizar la función costo, o en otras palabras, la red neuronal no es capaz aún de modelar correctamente la "función" deseada. Para ello hay que hacer parar por la red el número completo de instancias atravez de esta MLP y a su vez, actualizar los pesos con cada iteración.

Si bien, unas líneas atras se mencionó que el error cuadrático medio toma en cuenta todas las instancias para cálcular la función costo, este acercamiento no siempre es el óptimo para todas las tareas. A continuación se mencionan tres enfoques clásicos de optimización de entrenamiento.

1. **Stochastic** *gradient descendent*: Simplifica el cálculo considerando solo una instancia escogida de forma aleatoria cada vez que realiza el cálculo del gradiente para luego actualizar los pesos de la red.

2. **Mini-batch** *gradient descendent*: Se calculan los gradientes y se actualizan los pesos después de haber propagado $N$ instancias.

3. **Batch** *gradient descendent*: Se utilizan todas las instancias para calcular el error y actualizar.

## 3.7. Redes neuronales recurrentes

### 3.7.1. Modelo básico

En un intento por adaptar procesos neurobiológicos a un circuito integrado, se propone un primer algoritmo recurrente para computación neuronal [19]. Desde entonces, los principios de esta arquitectura han sido explotados para resolver problemas donde el orden de alimentación de las instancias a una máquina es determinante en la representación final de un modelo.

El proposito de una red neuronal recurrent o *RNN* (por sus siglas en inglés), es aprender y entregar un estado interno, conocido como *hidden state* el cual será una representación de un conjunto secuencial

de datos. Antes de profundizar en las operaciones que realiza una red recurrente, la siguente Figura 6 muestra el funcionamiento general de estas arquitecturas.

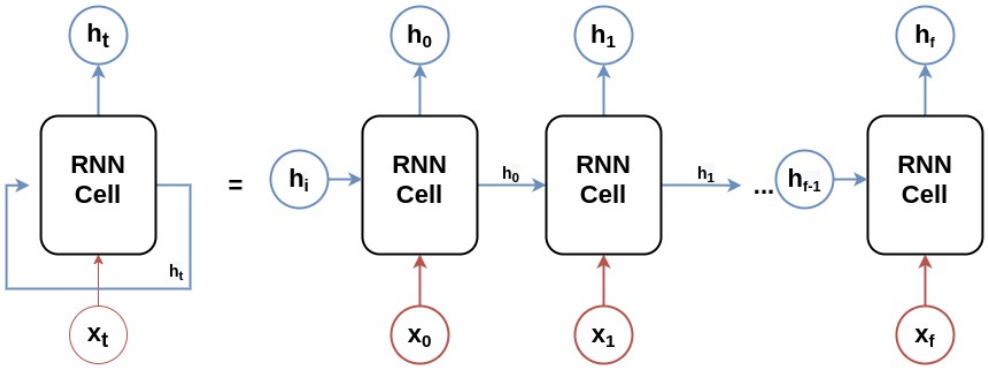


Figura 6: Representación general de una *RNN*.

El primer elemento que aparece en la descripción gráfica es un estado inicial $\mathbf{h}_i$, el cual representa un estado vacío de la secuencia, suele inicializarse aleatoriamente o simplemente como un vector en ceros, cuya dimensión se preserva a lo largo del entrenamiento. A la par que $\mathbf{h}_i$, se alimenta el vector que representa el inicio de una instancia (sequencia) $\mathbf{x}_0$, con estas dos entradas, se calcula el primer estado escondido $\mathbf{h}_0$, el cual, en ese momento representa la sequencia entera y será utilizado como entrada en el siguiente paso $t$, cuando la celda recurrente se alimente con $\mathbf{x}_1$ para producir el siguente estado escondido o representación de la serie $\mathbf{h}_1$.

El proceso anterior se repite, hasta llegar al elemetnto (o conjunto de elementos) final de la sequencia utilizada como dato de entrenamiento $\mathbf{x}_f$, el cual se utiliza, junto con el estado escondido antetior $\mathbf{h}_{f-1}$ para encontrar una representación final de dicha sequencia, conocido como estado final o $\mathbf{h}_f$.

Las operaciones internas dentro de una celda *RNN*, se visualizan en la Figura 7, donde se observa como dentro de dicha celda se encuentran dos capas neuronales totalmente conectadas, las cuales no cuentan con una función de activación. Las ecuaciones, encargas de encontrar un estado escondido son las

siguientes,

$$\begin{aligned}\mathbf{h}_t =& f\left(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}_h + \mathbf{W}_x\mathbf{x}_t + \mathbf{b}_x\right)\\ =& f(\mathbf{t}_x + \mathbf{t}_h) = f(\mathbf{z}).\end{aligned} \tag{28}$$

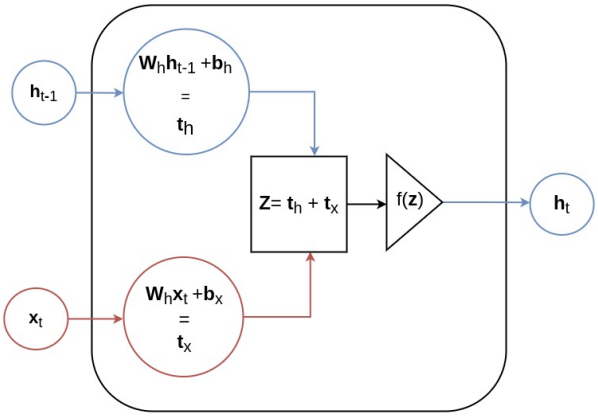


Figura 7: Operaciones internas en una *RNN*.

Donde, $\mathbf{W}_h, \mathbf{W}_x$, $\mathbf{b}_h$ y $\mathbf{b}_x$, son las matrices de pesos y bias que son entrenadas para lograr el aprendizaje del modelo. Se nombra $\mathbf{t}_h$ y $\mathbf{t}_x$ a la suma de las transformaciones lineales en las capas internas y a su vez, $\mathbf{z}$ la suma de ellas. Finalmente $f$ representa la ya acosumbtrada función de activación a la suma anterior, para dar paso a un estado escondido $\mathbf{h}_t$ encargado de representar una secuencia.

### 3.7.2. Gated recurrent units

La capa o celda recurrente recién descrita, actualiza los estados escondidos cada vez que es alimentada por un elemento de una instancia. Sin embargo, fue natural cuestionar si un estado escondido $\mathbf{h}_{t-1}$ puede llegar a ser más representativo en la secuencia que $\mathbf{h}_t$. Utilizando un razonamiento similar, se cuestiona si para un tiempo $t$ un estado escondido representa de mejor manera a la secuencia en cuestión que la misma entrada $\mathbf{x}_t$ y viceversa [6].

En otras palabras, que pasaría si en vez de simplemente sumar los resultados $\mathbf{t}_h$ y $\mathbf{t}_x$ y de ahí generar nuevos estados, se aprendiera una representación escalada de $\mathbf{t}_h$

$$\mathbf{h}_t = f(\mathbf{t}_h + \mathbf{t}_x) \quad \rightarrow \quad \mathbf{h}_t = f(\vec{r} \circ \mathbf{t}_h + \mathbf{t}_x), \tag{29}$$

y al mismo tiempo, se busca una poderación entre estados actuales y anteriores,

$$\mathbf{h}_t = f(\mathbf{t}_h + \mathbf{t}_x) \quad \rightarrow \quad \mathbf{h}^{'} = \mathbf{h}_t(1 - \vec{z}\,) + \vec{z} \circ \mathbf{h}_{t-1}. \tag{30}$$

El nuevo parametro $\vec{z}$ será el encargado de encontrar cuánto pesa en la representación el antiguo estado escondido, mientras que $\vec{r}$ deberá encontrar cuánta información de $\mathbf{t}_h$ se agrega a la suma $\mathbf{z}$. Al fijar la función de activación $f$ como una $\tanh$, además de combinar (29) y (30) se obtiene,

$$\mathbf{h}^{'} = \tanh\left(\vec{r} \circ \mathbf{t}_h + \mathbf{t}_x\right)(1 - \vec{z}\,) + \vec{z} \circ \mathbf{h}_{t-1}. \tag{31}$$

Los parámetros $\vec{r}$ y $\vec{z}$ se conocen como compuertas de reinicio y actualización respectivamento, o *reset gate* y *update gate*. Ambas compuertas son vectores cuyo espacio vectorial corresponde a la dimensión espacial de los estados escondidos. El producto de una compuerta vectorial cualquiera con un estado escondido se define como el producto Hadamard [15]

$$\vec{c} \circ \mathbf{h} = \sum_{i}^{\dim(\mathbf{h})} c_i \cdot h_i \hat{e}_i \tag{32}$$

que, para este caso en particular; es el tirple producto de la $i-$ésima entrada de los dos vectores $\vec{c}, \mathbf{h}$, junto con $\hat{e} \in \mathbb{R}^{\dim \mathbf{h}}$ donde este último es un vector de base canónica. Para lograr las operaciones propuestas, es necesario introducir una tercera compuerta "aspirante ":

$$\vec{n} \equiv \tanh(\vec{r} \circ \mathbf{t}_{hn} + \mathbf{t}_{xn}), \tag{33}$$

donde $\mathbf{t}_{hn}$ y $\mathbf{t}_{xn}$, son los mapeos lineales correspondientes a un *data point* y el estado escondido de la

iteración correspondiente. La compuerta aspirante, necesita de la compuerta de reinicio,

$$\vec{r} = \sigma(\mathbf{t}_{hr} + \mathbf{t}_{xr}). \tag{34}$$

Y al mismo tiempo que $\vec{r}$, se realiza la operación de la compuerta de actualización

$$\vec{z} = \sigma(\mathbf{t}_{hz} + \mathbf{t}_{xz}). \tag{35}$$

Los subíndices en los términos $\mathbf{t}_{ij}$ para $\{i|h, x\}$ y $\{j|r, z, n\}$ son un recordatorio de que, cada uno de estos parámetros cuentan con una matriz de pesos aprendible y un bias. La ecuación (31) se observa gráficamente en la Figura 8

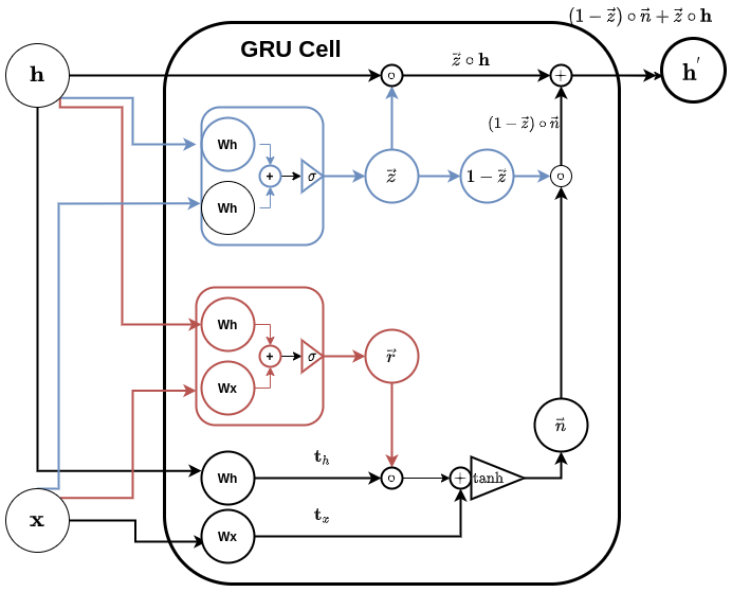


Figura 8: Operaciones internas en una *GRU*.

Una capa *GRU* es una mejora al compararla de forma empírica con una celda recurrente a básica, pero dos observaciones directas se pueden hacer:

- Los estados escondidos se encuentran ligados a una función tangente hiperbólica de activación, la cual asegura a todos ellos dentro de un mismo rango entre [-1,1]
- Al encontrar todos los estados escondidos fijos dentro de un mismo parámetro, los gradientes tam-

bién se fijan y como éstos son actualizados por el algoritmo de retropropagación (multiplicaciones consecutivas por regla de la cadena) es posible caer en el problemas de desvanecimiento de gradiente [17].

Una posible solución, a lo anterior es tomar en cuenta estados escondidos simultáneos, los cuales no compartan función de activación.

### 3.7.3. *Long Short-Term Memory*

La llamada (fines prácticos) *LSTM*, es una celda neuronal de tipo recurrente, la cual toma en cuenta dos acercamientos de estados escondidos:
Un primer y ya trabajado estado ligado, que se encuentra limitado por una función de activación $\tanh$, él cual será generado por una celda recurrente genérica y se conoce como **g**, estado aspirante de la red *LSTM* ,

$$\mathbf{g} = \tanh\left(\mathbf{t}_{hg} + \mathbf{t}_{xg}\right). \tag{36}$$

Y un nuevo estado propuesto libre de ligaduras (funciones de activación), nombrado celda de estado **c**. Dicho estado inicializa de manera arbitraria y se actualiza como una suma ponderada entre él y **g**,

$$\mathbf{c}^{'} = \vec{f} \circ \mathbf{c} + \vec{i} \circ \mathbf{g} \tag{37}$$

Donde $\vec{i}$ es la compuerta de entrada y $\vec{f}$ compuerta de olvido. Finalmente los nuevos estados escondidos se calculan como,

$$\mathbf{h}^{'} = \tanh(\mathbf{c}^{'}) \circ \vec{o}. \tag{38}$$

Donde la compueta $\vec{o}$ se conoce como compuerta de salida. La celda de estado **c** corresponde a *long-term memory*, mientras que el estado escondido **h** a *short-term memory* [18]. Se observa gráficamente las operaciones en una celda *LSTM* en la Figura 9.

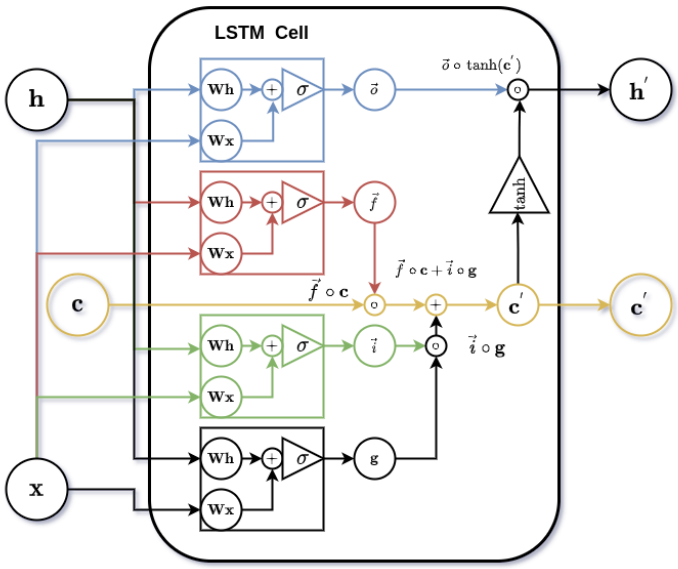


Figura 9: Operaciones internas en una *LSTM*.

Incluso cuando cambia el número de compuertas y las operaciones internas, los argumentos de entrada entre una celda *GRU* y una *LSTM* son los mismos. Sin embargo, los argumentos de salida si cambian, ya que una celda *LSTM* regresa dos estados simultáneos; un estado escondido y un estado celda.

Dos vértices en un grafo se dicen adjacentes sí y sólo sí, son puntos finales de una arista, mientras que una arista se dice incidente a cada uno de sus puntos finales. El número de aristas incidentes a un vértice se denomina el grado de un vértice y se utiliza la notación $\deg(v_i)$.

## 3.8. Arquitectura *Encoder-Decoder*

La arquitectura *encoder-decoder* es la combinación de los modelos: *encoder* y *decoder*.

### 3.8.1. *Encoder*

El objetivo del *encoder* es generar una nueva representación de una sequencia, o dicho de otra manera, codificar una sequencia. Una forma comunmente utilizada en arquitecturas de Procesamiento del Lenguaje Natural o *NLP* para codificar sequencias es utilizar redes neuronales recurrentes [46]. A continuación se muestra un diagrama de esta arquitectura

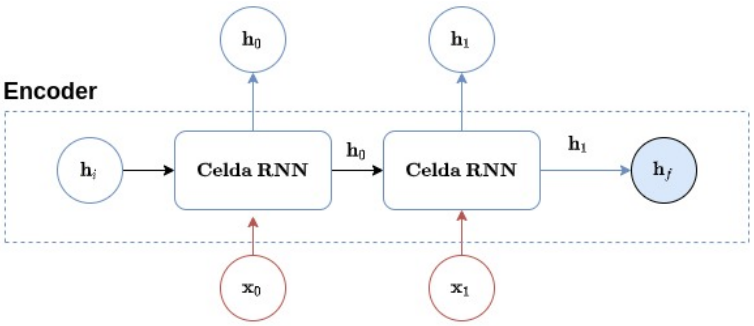


Figura 10: Arquitectura *Encoder*

### 3.8.2. Decoder

El fin del *decoder* es generar una sequencia que construya el objetivo o *target* del problema. Esta segunda parte de la arquitectura se alimenta de la siguiente manera:

- Se inicializa una celda recurrente con el estado final generado por el encoder ($\mathbf{h}_f$, Figura 10).
- La primera iteración, mencionada anteriormente, se alimenta también con el *data point* correspondiente al estado $\mathbf{h}_f$.
- Los estados escondidos generados por la celda recurrente dentro del decoder se alimentan a una multicapa perceptrón, la cual genera la sequencia objetivo.

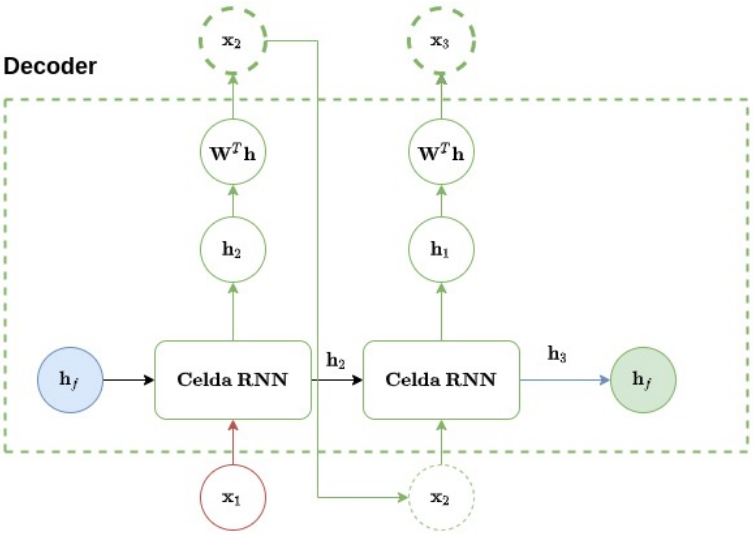


Figura 11: Arquitectura *Decoder*

De lo anterior se observa que toda la secuencia de entrada (la cual alimenta a al *encoder*) se ve representada solamente por el estado final.

### 3.8.3. *Teacher forcing*

El acercamiento anterior puede llegar a encontrar problemas en el tiempo de entrenamiento [23]. Al comiezo del entrenamiento, en lo general un modelo encuentra predicciones alejadas del objetivo, el problema se agrava en esta arquitectura ya que esos resultados se utilizan como instancia de entrada en los pasos subsequentes.

Una manera de acelerar el proceso de aprendizaje es utilizar la información del *target* como entrada en el *decoder* para así alimentar el modelo con la respuesta correcta, a esto se le conoce como *teacher forcing*. Obviamente, el alimentar con la respuesta correcta a cada instancia minimiza la función de pérdida en el entrenamiento pero empobrece resultados al en el proceso de *testing*

Para solucionar los inconvenientes mencionados que resultan de este acercamiento, se utiliza un proceso

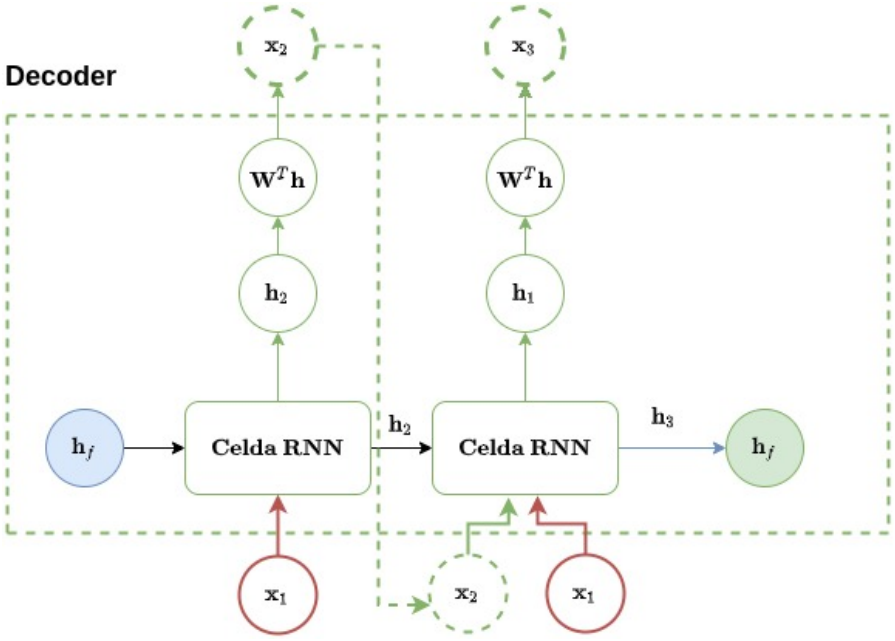


Figura 12: *Teacher forcing Decoder*

aleatorio, en el cual se fija una probabilidad $P$ de ocurrencia en el acercamiento de *teacher forcing*, y un

límite $l$ en el cuál se utilizará la predicción o el objetivo real:

$$\text{Input} = \begin{cases} \text{Objetivo} \quad \text{sí} \quad P < l \\ \text{Predicción} \quad \text{sí} \quad P \geq l. \end{cases} \tag{39}$$

## 3.9. Mecanismo de atención

Los mecánismos de atención utilizandos en arquitecturas como la desarrollada anteriormente permiten al *decoder* decidir que estados escondidos de la sequencia de entrada debe tomar en cuenta para la predicción del objetivo y no solamente el estado escondido final $h_f$. Para dicha decisión se utiliza un vector conocido como *context vector*. A continuación se describen los elementos y el proceso por el cual pasan estos elementos para construir un *context vector* o vector de contexto como se mencionará indistintamente.

### 3.9.1. *Values*

En un proceso de atención se conoce a los estados escondidos producidos por un *encoder* como *values* (**V**). La multiplicación de un *value* por el correspondiente *score attention* se denomina como *alignment vector* y a la suma de los *alignment vectors* se denomina *context vector*:

$$\textit{Context vector} = \alpha_0 V_0 + \alpha_1 V_1 = \sum \alpha_i V_i. \tag{40}$$

donde $\alpha_i V_i$ son los $i-$ésimos *alignment vectors*. En el siguiente apartado se muestra el origen de los *attention score* ($\alpha_i$).

### 3.9.2. *Keys* y *Queries*

Los *attention scores* se encuentran al empatar la transformación lineal de estados escondidos generados por el *decoder* con cada uno de las transformaciones lineales de los estados escondidos generados por el *encoder*. Para esto se necesitan dos definiciones más: a las transformaciones de los estados escondidos

del *encoder* se renombran como *Keys* (**K**). Mientras que las transfromaciones de los estados escondidos generados por el *decoder* se nombran *query* (**Q**).

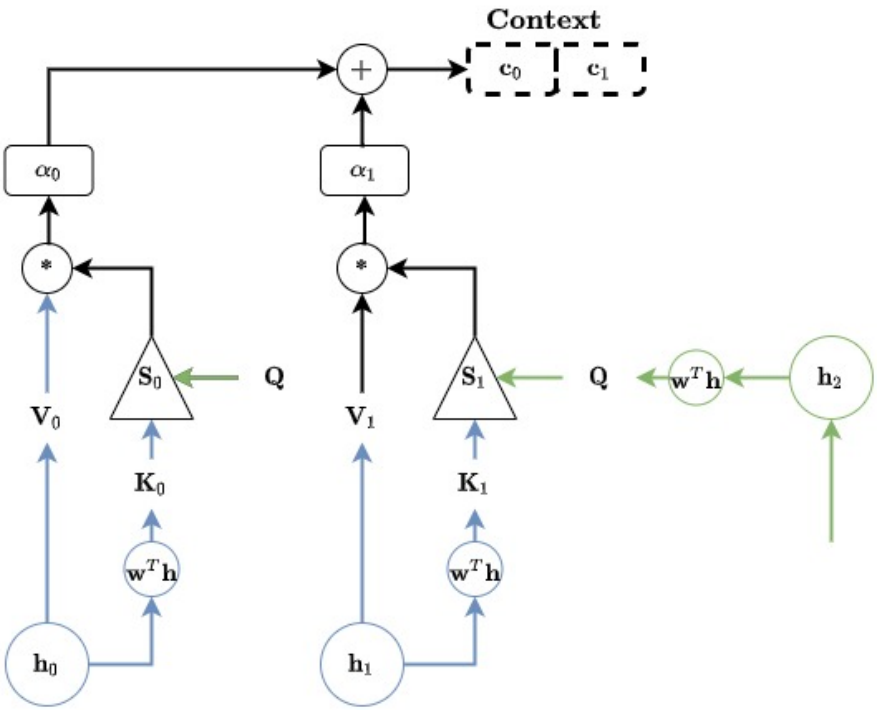


Figura 13: Mecanismo de atención

El diagrama de la Figura 13 muestra como el *Query* de un estado escondido del encoder se empata con las *Keys* de todos los estados escondidos del encoder.

### 3.9.3. Attention score y *context vector*

El método *scoring* es un proceso en el cual se determina que tan bien empatan los **K** con un **Q** o la similaridad entre ellos. Un método común (pero no único) para ello es buscar la similitud coseno entre dichos vectores. Esta similitud se define,

$$\cos\theta = \frac{\sum_i Q_i K_i}{\sum_{ij} \sqrt{Q_i^2}\sqrt{K_j^2}}. \tag{41}$$

Como $\theta$ es el ángulo que separa a los vectores **Q** y **K**, se infiere que dos vectores tendrán una similitud máxima cuando éstos sean paralelos entre ellos ($\cos 0^\circ = 1$), ambos vectores serán de similaridad

totalmente diferente cuando sean antiparalelos ($\cos 180° = -1$) y finalmente habrá similitud inexistente cuando ambos vectores sean ortogonales entre ellos ($\cos 90° = -1$).

Sin embargo la posición geométrica no es todo lo que se debe tomar en cuenta, la magnitud de los vectores, dictada en este caso por la suma de los cuadrados de los componentes en *Keys* y el vector *Query*, es muy importante también. Para lograr eso se multiplica ambos lados de la ecuación anterior por denominador de la parte izquierda de ésta,

$$\sum_{ij} Q_i K_{ij} = \sum_{j} |\mathbf{Q}|\,|\mathbf{K}| \cos\theta. \tag{42}$$

Resultando que la similaridad coseno multiplicada por la norma de los vectores es simplemente el producto punto entre ellos. El producto de la ecuación (42) se conoce como *alignment scores*.

Para conocer el *attention score* es necesario alimentar el vector *alignment* a una función softmax como se muestra a continuación

$$\mathbf{S}_i = \text{Softmax}\left(\sum_{ij} Q_i K_{ij}\right). \tag{43}$$

El vector $\mathbf{S}_i$ será una suma ponderada a la unidad. Y con ello el mecanismo de atención permite elegir el porcentaje necesario de atención para cada estado escondido, representado por $\mathbf{V}_i$ para cada $\mathbf{Q}$.

Al concatenar *alignment vector* se encuentra el *context vector*.

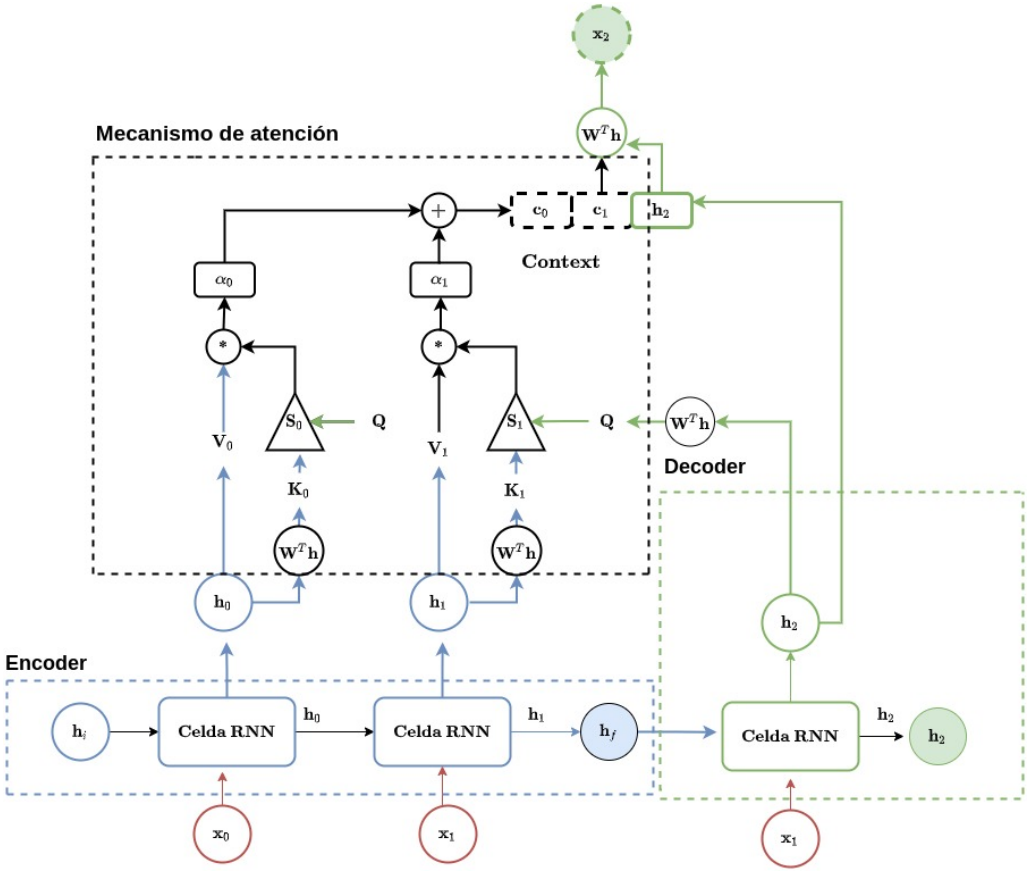


Figura 14: EncoderDecoder + Mecanismo de atención

La imagen anterior muestra un diagrama del mecanismo de atención incorporado a una arquitectura total Encoder-Decoder. La cual permite conocer la cantidad necesaria de información de todos los estados escondidos producidor por el *encoder* a la misma instancia que produce un estado escondido en la arquitectura *decoder* y así encontrar una predicción más cercana al objetivo original.

## 3.10. Cabezas de atención

No hay motivo teórico (excepto el costo computacional) para limitar una arquitectura a un solo mecánismo de atención o como también se le conoce; cabeza de atención. Todas las cabezas utilizadas en este tipo de modelos se alimenta con la totalidad de sus estados escondidos y produce un vector de contexto de la misma dimensión, los cuales son combinados utilizando una transformación lineal para encontrar solo vector de contexto.

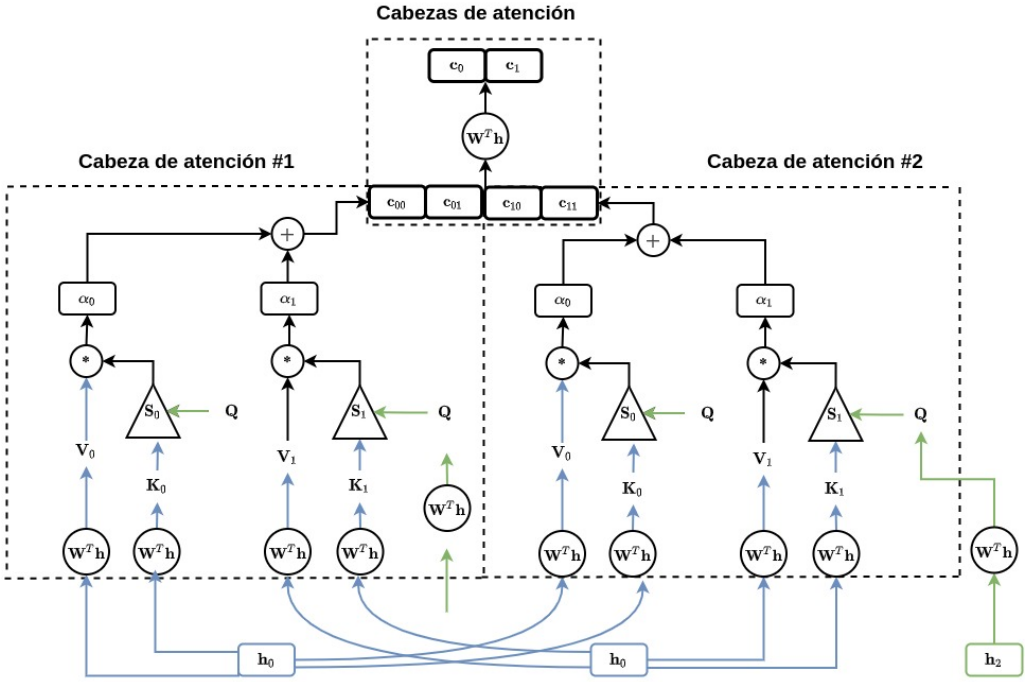


Figura 15: EncoderDecoder + Mecanismo de atención

Lo que a priori se podrá pensar como información redundante ya que una misma instancia alimenta cada una de las cabezas, hay que recordar que las transformaciones lineales son independientes en cada una de ellas, logrando así encontrar distintos patrones.

## 3.11. Codificación y decodificación paralela (Self-Attention)

En el afamado artículo *'Attention is all you need '* [44], se propone reemplazar las capas recurrentes por más mecanismos de atención. El principal motivo de esto, es evitar el computo sequencial que la misma recurrencia necesita para generar un estado escondido final.

### 3.11.1. *Self-Attention Encoder*

En esta arquitectura *Encoder* cada instancia se transforma, también en **Q** y por lo tanto produce su propio vector de contexto utilizando *alignment vectors* para cada instancia de la sequencia de entrada, incluida a ella misma.

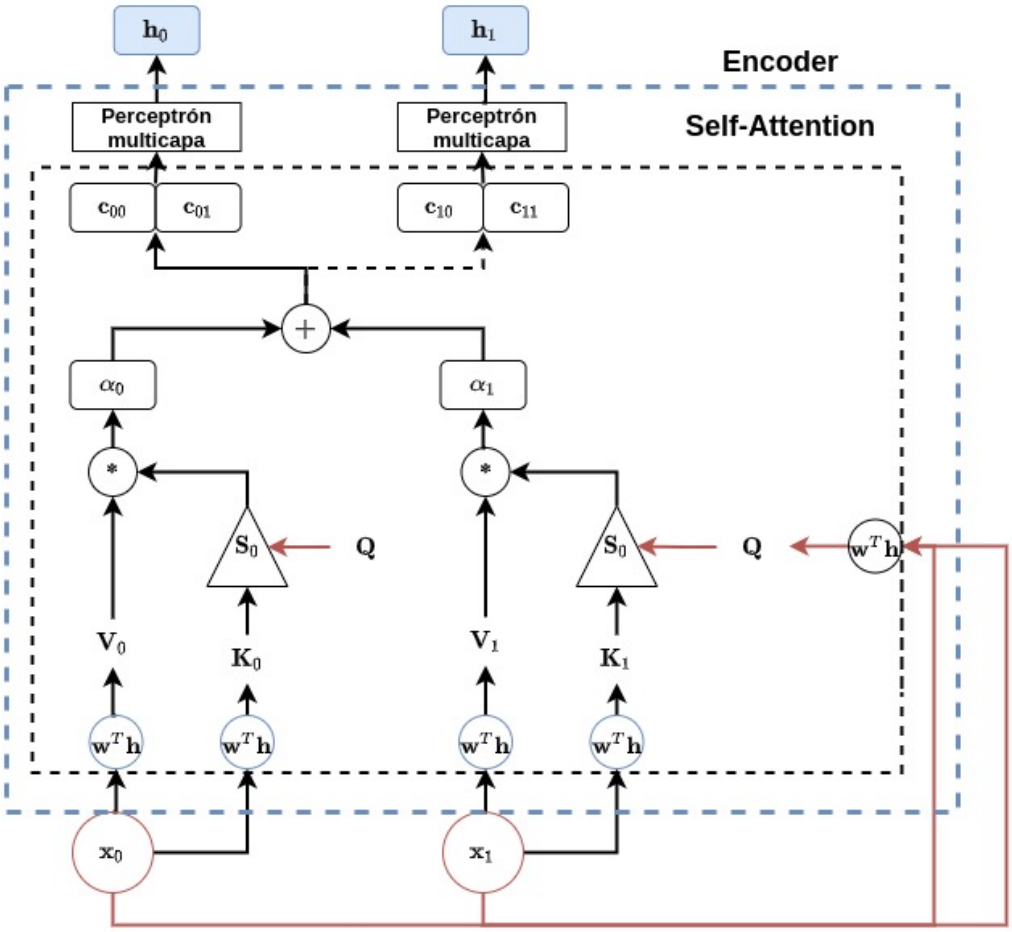


Figura 16: EncoderDecoder + Mecanismo de atención

Lo primero que realiza esta arquitectura al ser alimentada por una sequencia, es tomar la instancia $\mathbf{x}_i$ y se realiza una transformación lineal, resultando $\mathbf{Q}_i$. Cada *Query* encontrado se empatan con cada una de las diferentes $\mathbf{K}_i$ generadas. El producto anterior (dividido por un factor de normalización) se pasa por una función de activación resultando en los *attention scores* o *alphas*. Para el caso particular del diagrama anterior.

$$\alpha_0, \alpha_1 = \text{Softmax}\left(\frac{\mathbf{Q}_0\mathbf{K}_0}{\sqrt{2}}, \frac{\mathbf{Q}_1\mathbf{K}_1}{\sqrt{2}}\right) \tag{44}$$

El siguiente paso del algoritmo, es realizar el producto de las *alphas* con los *values*, para encontrar así un vector de contexto.

$$\textit{context vector} = \alpha_0\mathbf{V}_0 + \alpha_1\mathbf{V}_1. \tag{45}$$

Finalmente, el vector de contexto por la suma de los *alignment vectors* se alimenta a un multicapa perceptrón, creando un estado escondido $\mathbf{h}_0$ correspondiente a la instancia $\mathbf{x}_0$.

### 3.11.2. Cross-Attention

Un mecánismo de *cross-attention* se conoce a aquel mencionado en la sección (3.10), en la cual una arquitectura *decoder* provee los vectores $\mathbf{Q}$. Además, en este acercamiento, el estado escondido generado por el *decoder* en cuestión es concatenado al vector de contexto resultante.

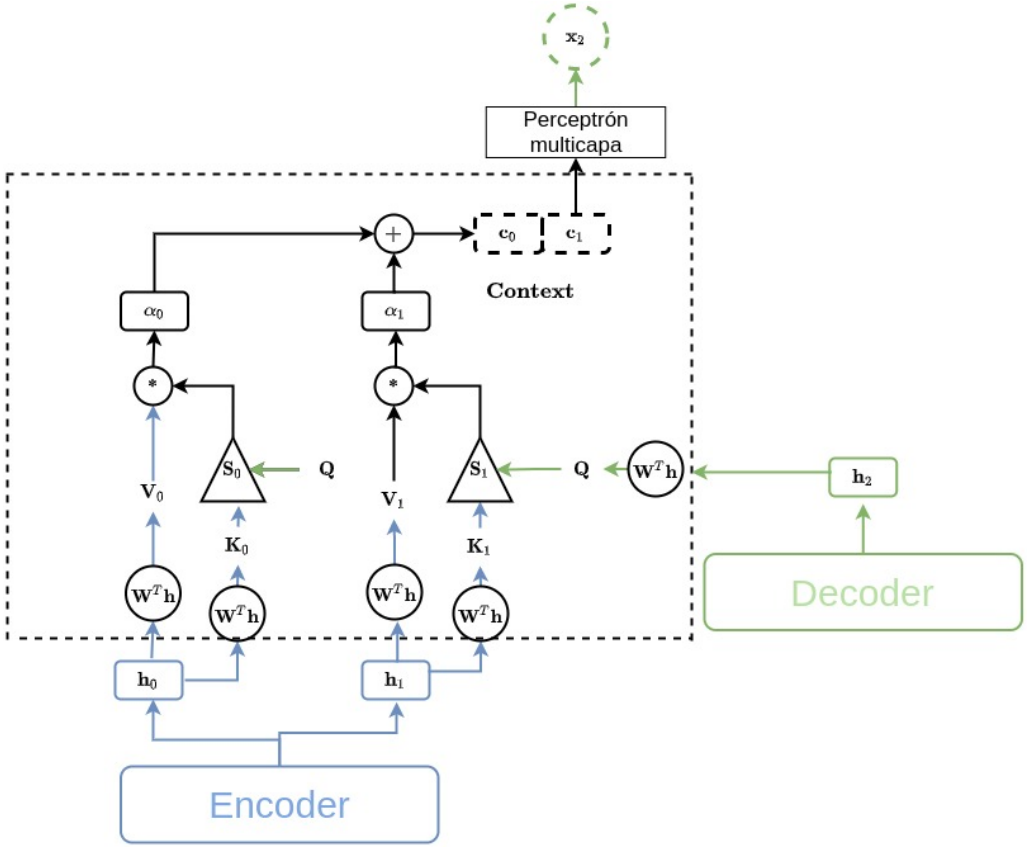


Figura 17: *Decoder self-attention*

El modelo propuesto como *EncoderDecoder* utilizando *self-attention* como generador de estados escondidos, no se ve en la necesidad de concatenar el vector contexto con estos últimos estados, sino que, utiliza un perceptrón multicapa en el vector contexto para general las predicciones.

### 3.11.3. *Decoder self-attention*

Al crear un modelo *decoder* capaz de generar estados escondidos sin la necesidad de compuertas recurrentes, se mantiene la capacidad de un cómputo paralelo.

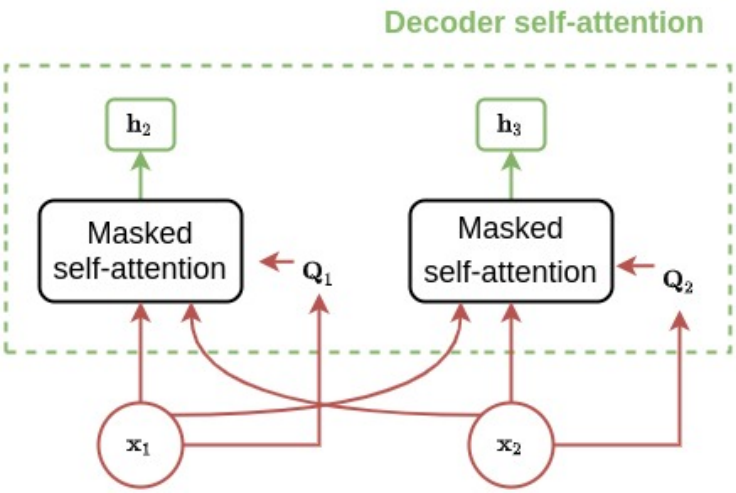


Figura 18: *Cross-Attention*

El mecanismo interno de atención del *decoder* lleva dos notables diferencias en comparación con el análogo del *encoder*. La primera, se visualiza gracias a los diagramas de las Figuras (18 y 19) en donde no existe un perceptrón multicapa en la parte superior del vector contexto para crear estados escondidos. Esto se debe a que la predicción del objetivo se realiza directamente en el mecanismo *cross-attention* mientras que la funcionalidad del *decoder* se límita a crear estados escondidos para la generacion de *Queries*.

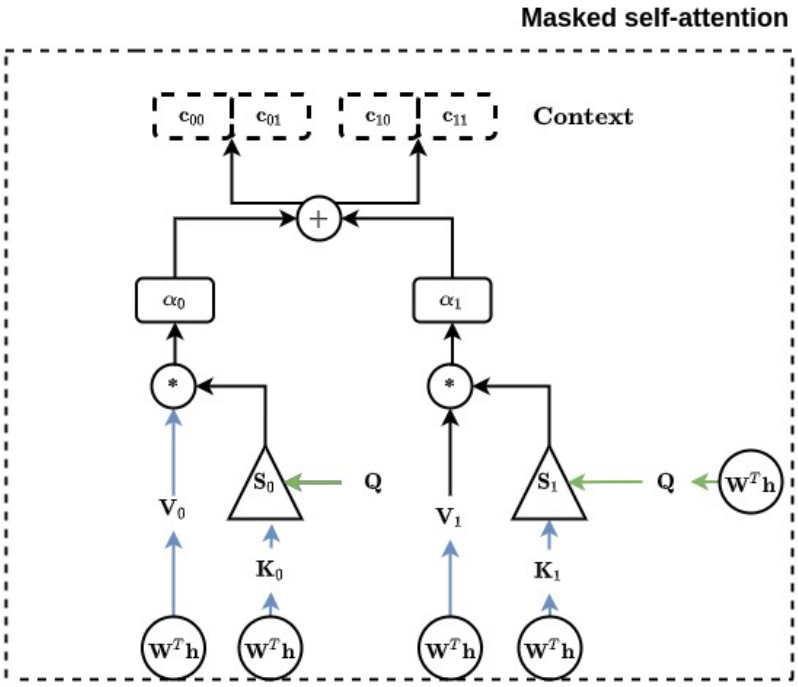


Figura 19: *Masked self-attention*

La segunda diferencia, y también motivo por el cual cambia el nombre del mecanismo de atención dentro de un *decoder*, *Masked Attention Scores*, es que se utiliza el número de instancias ejemplificadas en los diagramas. El primer *hidden state* generado por el *decoder* esta definido por

$$\alpha_{21}, \alpha_{22} = \text{Softmax}\left(\frac{\mathbf{Q}_1\mathbf{K}_1}{\sqrt{2}}, \frac{\mathbf{Q}_1\mathbf{K}_2}{\sqrt{2}}\right), \tag{46}$$

hasta el momento, no hay nada diferente, pero, al calcular el vector de contexto, se tiene que

$$\textit{context vector}_2 = \alpha_{21}\mathbf{V}_1, \alpha_{22}\mathbf{V}_2. \tag{47}$$

El problema es, que no es posible utilizar $\mathbf{K}_2$ y $\mathbf{V}_2$, vectores creados de una misma instancia, la cual se esta intentando predecir [23]. Para solucionar esto, se realiza un proceso de desvanecimiento de *alphas* mejor conocido como *Target Masking*. Este proceso consiste en crear una máscara, que bloquee la información

de las alfas correspondientes al vector de contexto de la instancia que se intenta predecir, esto es

$$\begin{aligned}\mathbf{h}_2 =& \mathbf{h}_2(\alpha_{21}, 0)\\ \mathbf{h}_3 =& \mathbf{h}_2(\alpha_{31}, \alpha_{32}, 0)\end{aligned} \tag{48}$$

donde los ceros, corresponden a los *alphas* $\alpha_{22}$ y $\alpha_{33}$. Resuelto el problema pasado se han descrito todos los elementos de una arquitectura *EncoderDecoder* utilizando un acercamiento *self-attention* para la generación paralela de estados escondidos.

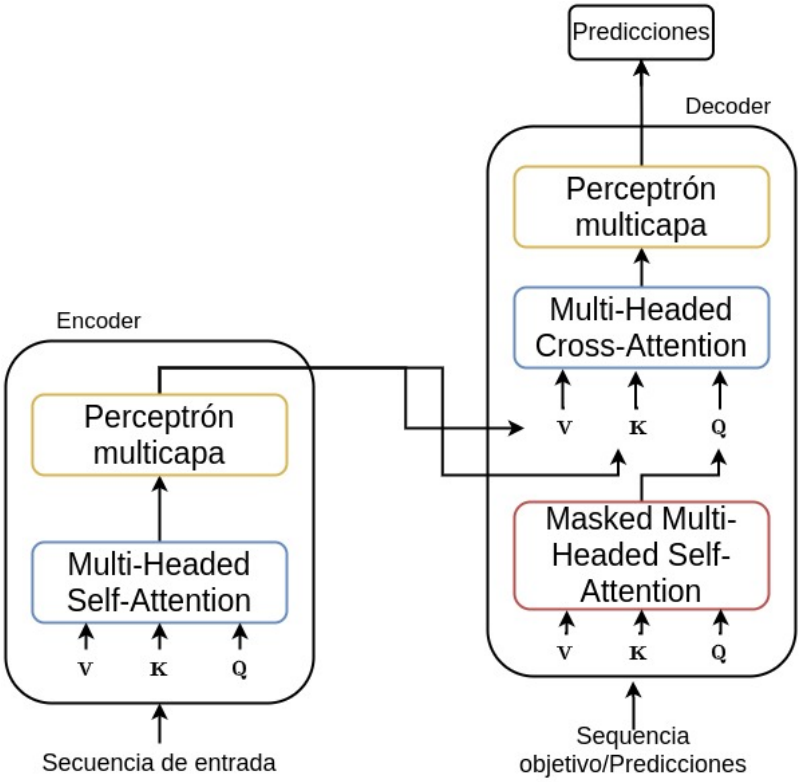


Figura 20: *Masked self-attention*

### 3.11.4. *Positional encoding*

El principal logro de las arquitecturas *EncoderDecoder* utilizando mecanismos *self-attention* es la mencionada computación paraleta. En otras palabras, se ha perdido la secuencialidad obligada por la escencia de las compuertas recurrentes. Lo anterior mejora el problema de cómputo, por otro lado se pierde infor-

maciíon natural del orden en las instancias y eso, por lo general afecta los entrenamientos [44].

Con el fin de generar un valor único para cada valor en los datos de entrada, los investigadores del artículo recén citado proponen utilizar el siguiente par de funciones cíclicas,

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), \tag{49}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right), \tag{50}$$

donde $i$ es la dimensión de la posición de los estados escondidos entrantes a *Positional encoding* y *pos* es la posición actual de la sequencia. Una vez encontrado vector término *Positional embedding* se suma a las características originales de las instancias, esto es

$$\mathbf{x}_i^{\text{sec}} = \mathbf{x}_i + \mathbf{PE}. \tag{51}$$

Donde el superíndice «sec» índica una instancia sequencializada.

### 3.11.5. *Message Passing*: *Message phase*

La fase *Message Passing* se corre a lo largo de $T$ iteraciones, donde en cada iteración se actualiza los estados escondidos $\mathbf{h}_v^t$ correspondientes a los nodos o átomos de las moléculas. Dicha actualización se basa en $m_v^{t+1}$

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}), \tag{52}$$

$$\mathbf{h}_v^{t+1} = E_t\left(m_v^{t+1}\right). \tag{53}$$

Donde la suma sobre todo vertice $w$ en $N(v)$, denota la suma discreta de los estados escondidos del vertice $v$. Siguiendo el estado del arte, en esta investigación se utiliza *encoder self-attention* de $n$ cabezas como función de actualización. Donde $E_t$ es un sistema *encoder* el cual se alimenta de $m_v^{t+1}$ para crear

cada una de las cabezas de atención, esto es

$$E_t\left(\text{multi-Head}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right)\right) = E_t\left(\sum_i \text{head}_i(m_v^{t+1})\right). \tag{54}$$

La función de actualización o *encoder* se ha implementado con 3 cabezas de atención con mecanismos *self-attention*. Dimensionalidad de modelo de 64 neuronas, 10 neuronas para 2 capas de perceptrón, activadas por una función ReLU.

# 4. Metodología

## 4.1. Marco de trabajo

### 4.1.1. *Framework*

La experimentación cuantitativa en esta investigación utilizó como marco de trabajo la libreria **PyTorch** como medio de aceleración en la reproducción y creación de algoritmos y arquitecturas de aprendizaje máquina.

La interpretación química; como lo fue el pasar de una cadena de caracteres directa de la base de datos cruda, a un objeto molécular se logró con la libreria *Atomic Simulation Environment (**ASE**)*.

Con el fin de encontrar propiedades moleculares, necesarias para la base de datos final, también se utilizó el paquete de informatica química **RDKit**: *Open-Source Cheminformatics Software*. Esta librería permitió encontrar enlaces moleculares (simples, dobles, triples y aromáticos) y el tipo de hibridiación orbital en las interacciones interatómicas.

Finalmente, esta investigación se ha apoyado de un ecosistema basado en PyTorch conocido como *PyTorch Geometric* (**PyG**). Esta libreria facilita la creación y entrenamiento de redes neuronales geométricas.

## 4.2. Representación

### 4.2.1. Representacion inicial

Al tener la información de las instancias mostrada en la Tabla 1, especifícamente, el número de átomos por molécula o número de cuerpos en el sistema atómico $n_a$, la geometría cartesiana del sistema y las cadenas de caracteres *SMILES* y *InChI* se utilizan, junto con los dos sistemas químicos asistidos por

computadora ***ASE*** y ***RDKit***, para encontrar:

- el número átomico o carga nuclear de cada uno de los átomos que componen el iésimo sistema

$$\mathbf{Z}_i = \left\{ \mathbf{Z}_i \middle| \mathbf{Z}_i \in \mathbb{R}^{1\times n_a} \right\}$$

- La geometría cartesiana de los sistemas en estado basal

$$\mathbf{R}_i = \left\{ \mathbf{R}_i \middle| \mathbf{R}_i \in \mathbb{R}^{n_a\times 3} \right\}$$

- El número y tipo de enlace entre átomos a lo largo del sistema (simple, doble, triple y aromático )

$$\mathbf{L}_i = \left\{ \mathbf{L}_i \middle| \mathbf{L}_i \in \mathbb{R}^{1\times \mathcal{L}} \right\}$$

  analísisanalísisanalísisanalísis donde $\mathcal{L}$ es el número de enlaces existentes en el iésima molécula.

Los arreglos $\mathbf{Z}_i$, $\mathbf{R}_i$ y $\mathbf{L}_i$, junto con el *target* deseado, Tabla 2 y Tabla 3, son información suficiente para crear el conjunto de datos que define a los sistemás atómicos,

$$\mathbf{x}_i^0 = \mathbf{Z}_i + \mathbf{R}_i + \mathbf{L}_i \tag{55}$$

### 4.2.2. Representación Grafo-Molecular

Se realiza una transformación de los tensores que componen el conjunto de datos, con el fin de convertir cada sistema o instancia a una estructura grafo, tal que

$$\mathbf{x}_i = G\left(\mathbf{V}_i(\mathbf{Z}_i), \mathbf{E}_i(\mathbf{R}_i, \mathbf{L}_i)\right), \tag{56}$$

donde $f$ es una función que mapea, tres tensores ($\mathbf{Z}$, $\mathbf{R}$, $\mathbf{L}$) a un objeto grafo $G$. Cada uno de los átomos dentro de un sistema, se vuelve un vertice $v$ en $G_i$ y la carga nuclear $\mathbf{Z}_i$ es el peso asignado. Mientras que una arista se le asignaran los dos siguientes valores:

1. **Enlace**: 0 sencillo, 1 doble, 2 triple, 3 aromático.

2. **Distancia**: el radio cartesiano entre la posición geomética de vértices incidentes (vértices donde exista enlace):

$$r = \sqrt{v_i^2 - v_j^2}. \tag{57}$$

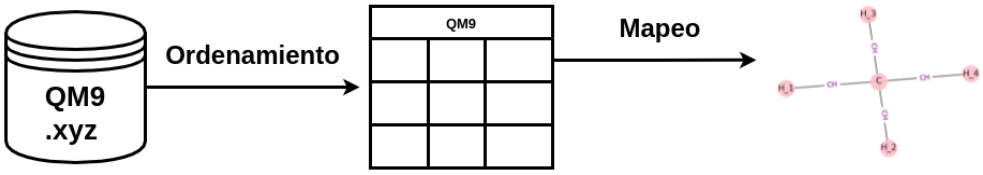


Figura 21: Representación gráfica de las transformaciones realizadas

Finalmente, al conjunto de datos (ya en una representación grafo) se le debe realizar un proceso de codificación, mejor conocido como *One-Hot Encoding*, por lo tanto se tendrá para

$$\mathbf{x}_i \rightarrow (\mathbf{h}_v^t, \mathbf{e}_{vw})_i, \tag{58}$$

donde $\mathbf{h}_v^t$ y $\mathbf{e}_{vw}$ se nombra como estados escondidos de los átomos (nodos) y estados escondidos de los enlaces (aristas). El super índice $t$ índica la tésima fase de la red.

### 4.2.3. Capa atómica

La primer capa entrenable del modelo es una capa densa (totalmente conectada), que se aplica de forma separada a la representación de cada átomo , esto es

$$\mathbf{h}_v^t = \mathbf{W}^t \mathbf{h}_v^t + \mathbf{b}^t. \tag{59}$$

Ya que los pesos $\mathbf{W}^l$ y bias $\mathbf{b}^l$ se comparten para todos los átomos, la arquitectura permanece invariante, sin importar el número de átomos por sistema (molécula).

### 4.2.4. Convolución vía Message Passing

El proceso de *Graph Convolution* o convolución vía *Message Passing* se divide en dos pasos

- Fase *Message Passing*: Se corre a lo largo de $t$ iteraciones y se define en términos de funciones *Message Passing* $M_t$ y una función de actualización de vértices $U_t$. Durante esta fase, los estados escondidos $\mathbf{h}_{vi}^t$ de cada vértice es actualizado de acuerdo a,

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}) \tag{60}$$

$$\mathbf{h}_v^{t+1} = U_t(m_v^{t+1}). \tag{61}$$

Donde la suma sobre todo vértice $w$ en $N(v)$, denota la suma discreta de los estados escondidos del vértice $v$. La función *Message Passing* se define como

$$M_t(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}) = A_{e_{vw}}\mathbf{h}_w^t, \tag{62}$$

donde $A_{e_{vw}}$ es una función entrenable, una para cada arista incidente **e**. En esta investigación se utiliza para dicha función dos capas escondidas totalmente conectadas de 64 y 1,024 neuronas respectivamente, entre ellas una función activación ReLU.

La función de actualización $U_t$, también necesita ser una función entrenable, y en este trabajo se propuso utilizar para dicha función una arquitectura *self-attention encoder*. *Self-attention encoder* se implementa con 3 cabezas de autoatención seguida de una doble capa perceptrón con dimensionalidad de modelo de 64 neuronas y 10 neuronas en la segunda capa densa activadas por una función ReLU y una capa de salida de 64 neuronas. Dicha arquitectura se puede ver en la Figura 22. Los estados resultantes de la última capa se definen como $\mathbf{h}_v^{t+1}$.

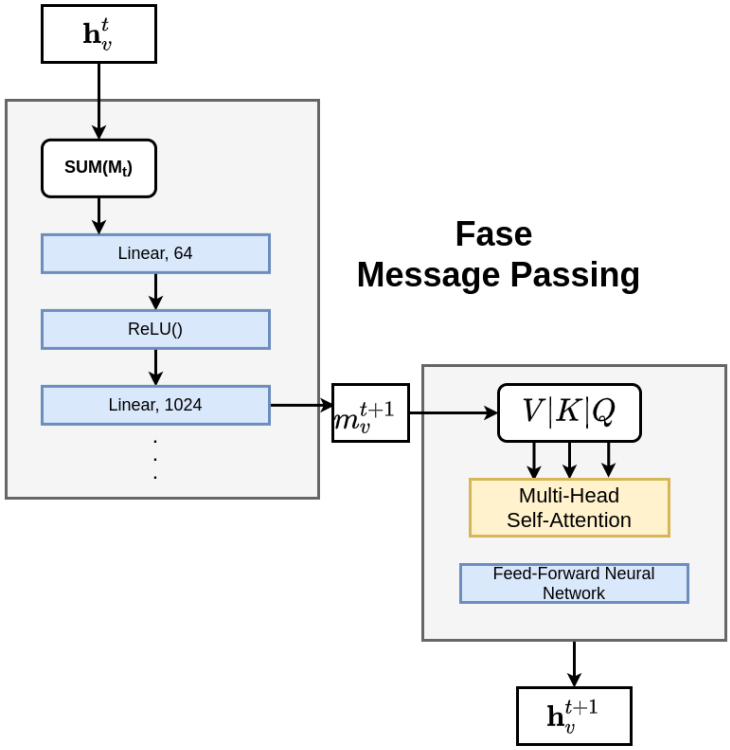


Figura 22: Diagrama a bloques de la fase *Message Passing*

- Fase *readout*: Cálcula un tensor característico $\vec{m}_i$ para todo el grafo (tensor molecular), usando una función *readout* tal que

$$\vec{m}_i = R(\{\mathbf{h}_v^{t+1} | v \in G\}). \tag{63}$$

En este trabajo, se propone utilizar un mecanismo *self-attention decoder* como función *readout*. Se implementa con los mismos parámetros que el *encoder*; 3 cabezas de autoatención, una dimensionalidad del modelo de 64 neuronas, 10 neuronas en doble capa densa, activada por una función ReLU y una capa de salida de 64 neuronas. Los tensores $\vec{m}_i$ al representar una molécula pueden ser utilizados directamente para realizar la tarea de aproximación. En esta investigación se realiza una degradación de dimensiones utizando dos capas densas; la primera de 32 neuronas activada por una función ReLU y la segunda de una sola unidad, encargada de realizar la regresión.

A todo el modelo descrito en esta seccion se le nombró como MultiHead - Message Passing y se muestra en la Figura 24

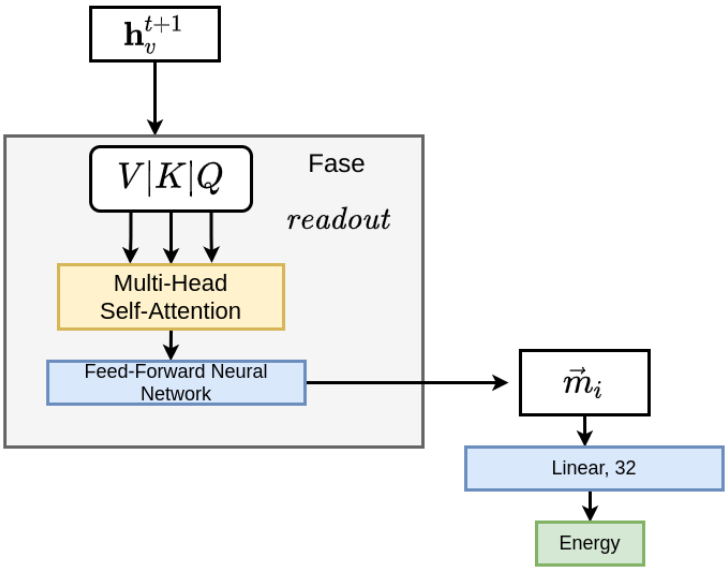


Figura 23: Diagrama a bloques de la fase *readout* + predicción

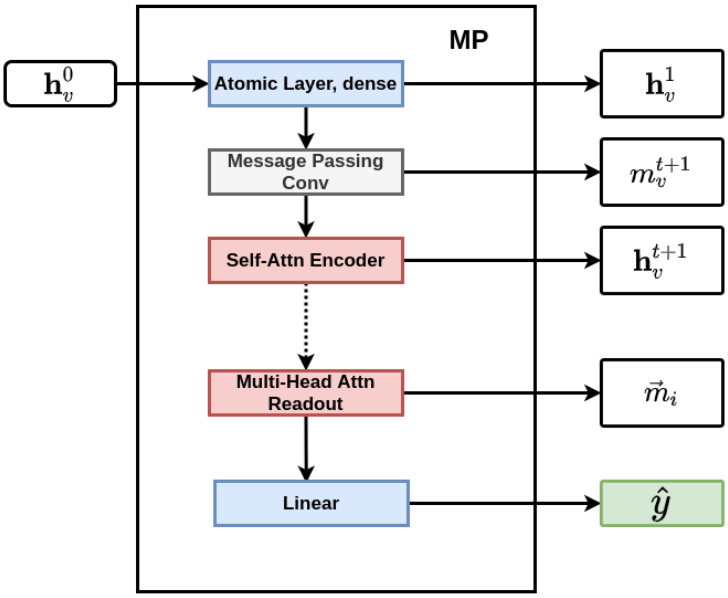


Figura 24: Diagrama a bloques de la fase *readout* + predicción

# 5. Resultados

## 5.1. Base de datos

En la búsqueda de modelos de aprendizaje computacional capaces de aproximar propiedades fisicoquímicas de sistemas atómicos, es común utilizar como punto de referencia base de datos parteaguas, QM9 es una de ellas [29]. Esta base de datos almacena cerca de 133, 000 moléculas orgánicas, en las cuales se reporta: geometrías, energías, configuración electrónica y propiedades termodinámicas.

Los sistemas reportados en este marco son moléculas amorfas, creadas con el conjunto de átomos CHONF (carbono, hidrógeno, oxígeno, nitrógeno y flúor). Cada instancia se encuentra en archivos nombrados `dataset_index.xyz`, y cuentan con el formato generalizado de siguiente Tabla.

Tabla 1: **Estructura de los sistemas atómicos en formato .xyz**

| **Iésimo sistema molecular** | |
|---|---|
| 1 | Número de átomos por molécula $n_a$ |
| 2 | Propiedades escalares |
| 3,..., $n_a + 2$ | Elemento, coordenada $(x, y, z)$ en $\mathring{A}$, cargas parciales Mulliken en $e$ por átomo |
| $n_a + 3$ | Frecuencias armónicas vibracionales en $cm^{-1}$ |
| $n_a + 4$ | Cadena *SMILES* de GDB$_{-17}$ y B$_3$LYP |
| $n_a + 5$ | Cadena *InChI* (International Chemical Identifier) |

Las propiedades escalares ordenadas en la segunda línea de los archivos .xyz, junto con las unidades que han sido creadas se muestran en la siguiente Tabla:

Tabla 2: **Información de propiedades escalares 1-9**

| **Propiedades escalares** | | | |
|---|---|---|---|
| **$N_o$** | **Propiedad** | **Unidad** | **Descripción** |
| 1 | Etiqueta | - | Cadena de caracteres “gdb9” |
| 2 | i | - | Identificador entero base-1 |
| 3 | A | GHz | Constante rotacional |
| 4 | B | GHz | Constante rotacional |
| 5 | C | GHz | Constante rotacional |
| 6 | $\mu$ | D | Momento dipolar |
| 7 | $\alpha$ | $a_0$ | Polarizabilidad isotrópica |
| 8 | $\epsilon_{\text{HOMO}}$ | Ha | Energía HOMO |
| 9 | $\epsilon_{\text{LUMO}}$ | Ha | Energía LUMO |

Tabla 3: **Información de propiedades escalares 10-17**

| **Propiedades escalares** | | | |
|---|---|---|---|
| **$N_o$** | **Propiedad** | **Unidad** | **Descripción** |
| 10 | $\epsilon_{\text{gap}}$ | Ha | $\epsilon_{HOMO} - \epsilon_{LUMO}$ |
| 11 | $\langle R^2 \rangle$ | $a_0^2$ | Extensión espacial electrónica |
| 12 | zpve | Ha | Punto de energía vibracional cero |
| 13 | $U_0$ | Ha | Energía interna a 0 K |
| 14 | $U$ | Ha | Energía interna a 298.15 K |
| 15 | $H$ | Ha | Entalpia a 298.15 K |
| 16 | $G$ | Ha | Energía libre |
| 17 | $C_v$ | $\frac{\text{mol}}{\text{molK}}$ | Capacidad calorífica |

## 5.2. Configuración del modelo

La primer propiedad que se utilizó como experimento de aproximación fue la energía en estado basal, $U_0$, por lo tanto se fija $\hat{y} = U_0$.

A continuación se menciona la configuración necesaria utilizada en el experimento.

1. Normalizar el conjunto $\hat{y}$, esto es, transformar el *target* de la base de datos a fin de que la media sea igual a cero y la desviación estandar igual a la unidad.

2. Se dividieron los conjuntos de entrenamiento, validación y prueba en 15,000, 4,3000 y 2,200 instancias, respectivamente. El ordenamiento de los tres conjuntos anteriores se formó de manera aleatoria .

3. Se utilizó el optimizador *ADAM* con los paraámetors colocados por defecto, con excepción del *learning rate*.

4. Se inicializó el entrenamiento con un *learning rate* igual a $10^{-3}$ y un factor de decaimiento $f = 0.7$ cada cinco épocas, y un límite inferior de $10^{-5}$.

5. Se entrenó durante 200 épocas utilizando un *batch size* igual a 8.

6. Se utilizó un equipo de cómputo el cual cuenta con una targeta de video **nvidia** RTX 2060.

## 5.3. Resultados Obtenidos para $U_0$ con el modelo MultiHead-Message Passing

En este apartado se muestras las gráficas del error absoluto medio en la predicción del observable escalar energía en estado basal $U_0$. Este primer resultado se logró utilizando la arquitectura propuesta de MultiHead-Message Passing basada en un algoritmo *Message passing* y compuesta por un *Encoder self-attention*.

En la Figura se puede observar como el error se mantiene con pocas variaciones hasta llegar a las iteraciones 75-90. Pasado ese intervalo se visualiza un decaimiento en el error, lo que índica una mejora en el proceso de aprendizaje. El siguiente punto y el más importante en este resultado, es que el conjunto de

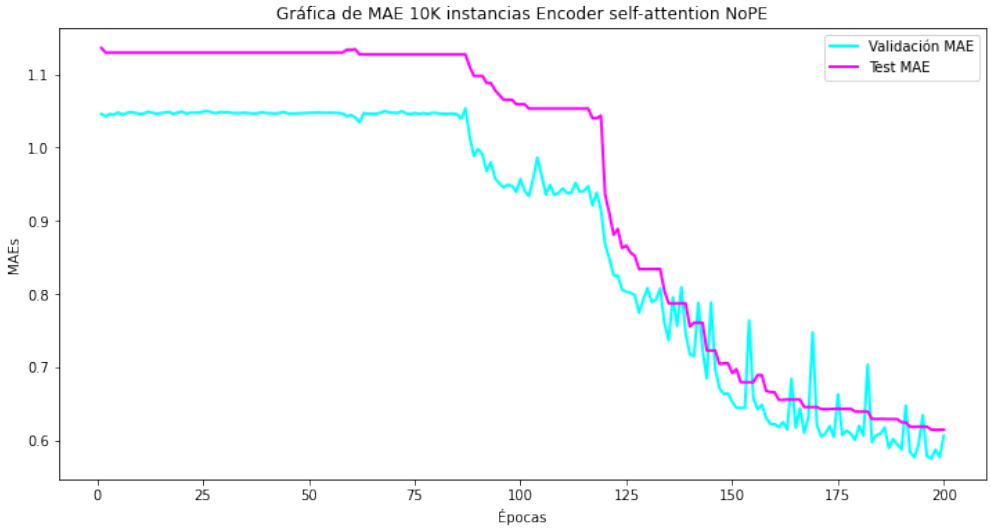


Figura 25: Gráfica del error absoluto medio en los conjuntos de validación (turquesa) y entrenamiento (magenta) para una arquitectura con algoritmo *Message Passing* y *self-attention Encoder* o MultiHead-Message Passing

validación no da seales de haber alcanzado un mínimo global, el ruido y patrón de crecimiento negativo índica que en este modelo y con el mismo número de instancias es posible encontrar mejores resultados con más iteraciones en el proceso de entrenamiento.

Los siguientes resultados a analizar son de la misma arquitectura *message passing* compuesta con un *Encoder self-attention* y un proceso de *Positional Encoding* en las instancias antes de entrar al *Encoder*. En el par de conjuntos (validación y test) de este experimento se observa muy poco ruido después del intervalo 25-50 de iteraciones. Además de eso se observa que después de ese intervalo se alcanza la convergencia de este experimento, por lo que es natural pensar en alimentar esa arquitectura con los mismos parametros pero más instancias y buscar mejores resultados, pero dificilmente se encontrarán mejores resultados con más iteraciones y las mismas instancias.

El tercer experimiento a visualizar, es la arquitectura propuesta en el artículo *Neural Message Passing for Quantum Chemestry* donde se emplea el acercamiento operacional *Message Passing*, mismo que le da el nombre a este. Se alimentó con el mismo número de datos que los dos experimentos pasados, la diferen-

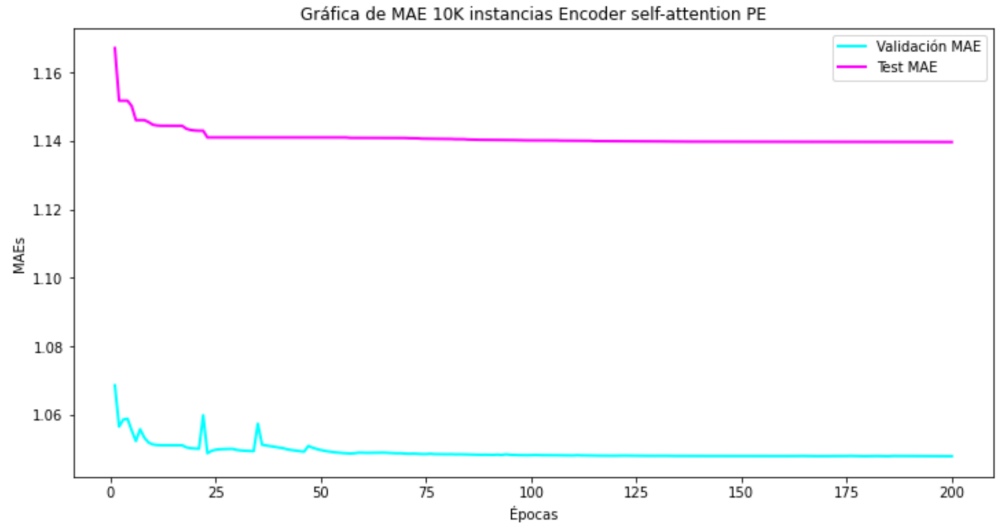


Figura 26: Gráfica del error absoluto medio en los conjuntos de validación (turquesa) y entrenamiento (magenta) para una arquitectura con algoritmo *Message Passing* y *self-attention Encoder + Positional Encoding*

cia central es que en la fase *Message update* se utiliza una unidad *GRU* unidireccional. En la recreación

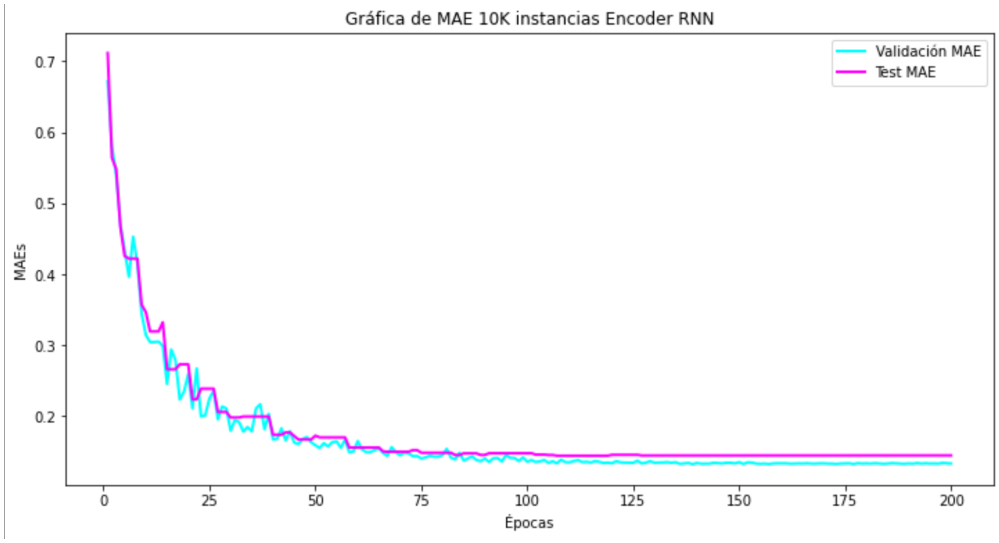


Figura 27: Gráfica del error absoluto medio en los conjuntos de validación (turquesa) y entrenamiento (magenta)para una arquitectura con algoritmo *Message Passing* y codificación vía *GRU*

de este experimento se observa una convergencia a la mitad del entrenamiento, además de llegar a un error menor que los dos anteriores. Sin embargo, también sabemos que para esta configuración general este experimento ya ha alcanzado sus mejores predicciones, es decir es un modelo pre-entrenado y aquí solamente se estan utilizando los valores de sus pesos para un ajuste fino.

Finalmente se agruparon los tres errores absoulutos medios para los conjuntos de prueba o *test*.

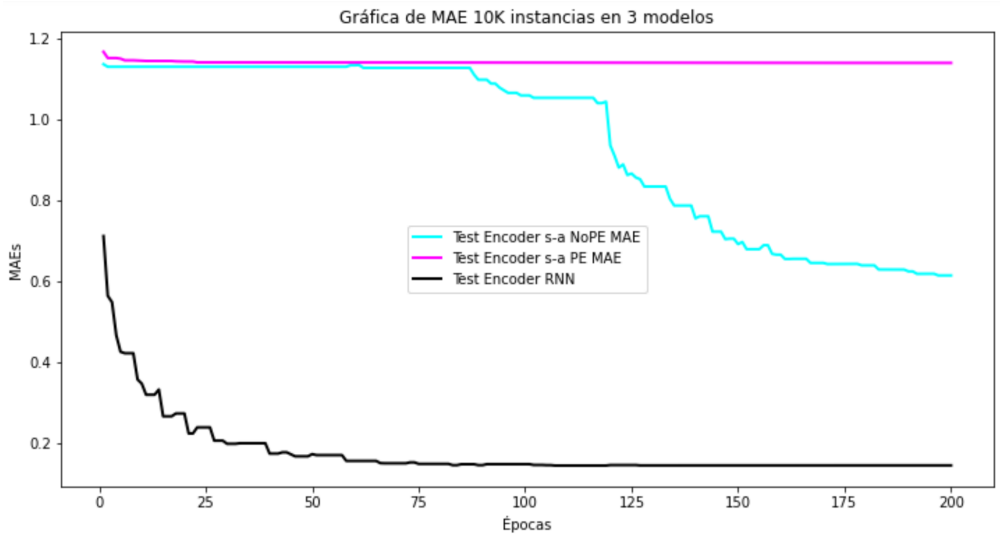


Figura 28: Gráfica del errores absolutos en el conjunto de entrenamiento de las tres arquitecturas analizadas

Se puede observar claramente la superioridad del modelo *Message Passing* y codificación vía *GRU* para la configuración utilizada. El modelo *Message Passing* y *self-attention Encoder + Positional Encoding* utiliza un *Positional Encoding* en los estados escondidos antes de alimentar al *Encoder* y se encuentra lejos de poder competir con la arquitectura original. El agregar esta información al modelo no le favoreció por la naturaleza del problema en el cual la geometría de la molécula pierde al parecer relevancia. Finalmente, al eliminar el *Positional Encoding* de la arquitectura propuesta en esta investigación tiene un mejor aprendizaje con la desventaja de ser más lento. Posiblemente si se utilizara un mayor número de iteraciones sería posible encontrar mejores resultados, pero las gráficas indican que el error va disminuyendo conforme avanza el entrenamiento.

# 6. Conclusiones

En este trabajo de investigación se propuso un método novedoso basado en redes neuronales que incorpora diferentes mecánismos de atención enfocados a la predicción de propiedades moleculares. A al modelo propuesto se le nombró *Multi-Head Message Passing* o *M-H MP*. Particularmente, los mecanismos de atención que se utilizan son *Multi-head self-attention* y *cross-attention*. Además, la *M-H MP* emplea una estructura de grafos la cual permite incorporar elementos importantes de la geometría molecular. Adicionalmente se implementa todo bajo el concepto de *Message Passing* para representar adecuadamente propiedades fisicoquímicas de las moléculas.

Hasta el momento, los resultados logrados con la arquitectura propuesta son prometedores ya que en el experimento donde se utiliza sin el proceso de *Positional Encoding* no ha mostrado sobreentrenamiento pero pareciera que el modelo no ha logrado converger aún. Hace sentido que el modelo original, armando con una *RNN GRU* encuentre en menos iteraciones sus mejores resultados ya que cuenta con un número menor de parametros en comparación con el implementado en *M-H MP* el cual cuenta con un *encoder* de múltiples cabezas de atención.

Sin embargo, en ninguno de los experimentos hechos hasta el momento, se ha logrado llegar a un error químico (*MAE*$\approx .04$), el cual modelos como la Schnet [35] logra en aproximadamente 2400 épocas, al utilizar menos de 110,000 instancias de entrenamiento. Por lo tanto, el siguiente y último paso en esta investigación será realizar experimentos con configuraciones análogas a la propuesta en este trabajo de investigación.

# 7. Referencias


[1] Alpaydin, E. (2014). Introduction to machine learning ethem alpaydin. *Introduction to machine learning*.

[2] Bogojeski, M., Vogt-Maranto, L., Tuckerman, M. E., Müller, K.-R., and Burke, K. (2020). Quantum chemical accuracy from density functional approximations via machine learning. *Nature communications*, 11(1):1–11.

[3] Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., and Zdeborová, L. (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002.

[4] Carrasquilla, J. and Melko, R. G. (2017). Machine learning phases of matter. *Nature Physics*, 13(5):431–434.

[5] Cheng, Z., Bai, F., Xu, Y., Zheng, G., Pu, S., and Zhou, S. (2017). Focusing attention: Towards accurate text recognition in natural images. In *Proceedings of the IEEE international conference on computer vision*, pages 5076–5084.

[6] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[7] Combes, J.-M., Duclos, P., and Seiler, R. (1981). The born-oppenheimer approximation. In *Rigorous atomic and molecular physics*, pages 185–213. Springer.

[8] Dirac, P. A. M. (1929). Quantum mechanics of many-electron systems. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 123(792):714–733.

[9] Engel, T. (2006). *Quantum chemistry and spectroscopy*. Pearson Education India.

[10] Fukui, H., Hirakawa, T., Yamashita, T., and Fujiyoshi, H. (2019). Attention branch network: Learning of attention mechanism for visual explanation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10705–10714.

[11] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

[12] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

[13] Grisoni, F., Moret, M., Lingwood, R., and Schneider, G. (2020). Bidirectional molecule generation with recurrent neural networks. *Journal of chemical information and modeling*, 60(3):1175–1183.

[14] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[15] Hiai, F. and Lin, M. (2017). On an eigenvalue inequality involving the hadamard product. *Linear Algebra and its Applications*, 515:313–320.

[16] Hibat-Allah, M., Ganahl, M., Hayward, L. E., Melko, R. G., and Carrasquilla, J. (2020). Recurrent neural network wave functions. *Physical Review Research*, 2(2):023358.

[17] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[18] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[19] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

[20] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

[21] Kamath, U., Liu, J., and Whitaker, J. (2019). *Deep learning for NLP and speech recognition*, volume 84. Springer.

[22] Klein, M. J. (1961). Max planck and the beginnings of the quantum theory. *Archive for History of Exact Sciences*, 1(5):459–479.

[23] Lamb, A. M., ALIAS PARTH GOYAL, A. G., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29.

[24] LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

[25] Li, Y. (2020). A calibration method of computer vision system based on dual attention mechanism. *Image and Vision Computing*, 103:104039.

[26] Lubbers, N., Smith, J. S., and Barros, K. (2018). Hierarchical modeling of molecular energies using a deep neural network. *The Journal of chemical physics*, 148(24):241715.

[27] Mills, K., Spanner, M., and Tamblyn, I. (2017). Deep learning and the schrödinger equation. *Physical Review A*, 96(4):042113.

[28] Parr, R. G. (1983). Density functional theory. *Annual Review of Physical Chemistry*, 34(1):631–656.

[29] Ramakrishnan, R., Dral, P. O., Rupp, M., and von Lilienfeld, O. A. (2014). Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1.

[30] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

[31] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. (2020). Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR.

[32] Schrödinger, E. (1926). Quantisierung als eigenwertproblem. *Annalen der physik*, 385(13):437–490.

[33] Schütt, K., Gastegger, M., Tkatchenko, A., Müller, K.-R., and Maurer, R. J. (2019). Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions. *Nature communications*, 10(1):1–10.

[34] Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. (2017a). Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8(1):1–8.

[35] Schütt, K. T., Kindermans, P.-J., Sauceda, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. (2017b). Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *arXiv preprint arXiv:1706.08566*.

[36] Shui, Z. and Karypis, G. (2020). Heterogeneous molecular graph neural networks for predicting molecule properties. *arXiv preprint arXiv:2009.12710*.

[37] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[38] Slater, J. C. (1951). A simplification of the hartree-fock method. *Physical review*, 81(3):385.

[39] Sperber, M., Neubig, G., Niehues, J., and Waibel, A. (2019). Attention-passing models for robust and data-efficient end-to-end speech translation. *Transactions of the Association for Computational Linguistics*, 7:313–325.

[40] 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. (2020). A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.

[41] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[42] Thekumparampil, K. K., Wang, C., Oh, S., and Li, L.-J. (2018). Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*.

[43] Torlai, G., Mazzola, G., Carrasquilla, J., Troyer, M., Melko, R., and Carleo, G. (2018). Neural-network quantum state tomography. *Nature Physics*, 14(5):447–450.

[44] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[45] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

[46] Vinyals, O., Bengio, S., and Kudlur, M. (2015). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

[47] Wang, C., Zhai, H., and You, Y.-Z. (2019). Emergent schrödinger equation in an introspective machine learning architecture. *Science Bulletin*, 64(17):1228–1233.

[48] Wang, Y., Zhang, J., Kan, M., Shan, S., and Chen, X. (2020). Self-supervised equivariant attention mechanism for weakly supervised semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12275–12284.