Modelado de Propiedades Críticas de Sustancias mediante Redes Neuronales y Estructura Molecular

Autor:

Eliana Marcela Mayorga Chaves

Universidad ECCI

Tecnología de Procesos Químicos Industriales

Ingeniería Química

Asesor:

Camilo Andrés López Santamaría

Abril, 2024

Resumen

Este estudio se centra en la aplicación de redes neuronales en ingeniería química para la estimación precisa de propiedades críticas, como la presión y la temperatura crítica, fundamentales en el diseño de plantas químicas. En Colombia, la industria química enfrenta un desafío importante debido a la falta de avances tecnológicos en el diseño de equipos para sectores clave como agricultura, alimentos, farmacéutica y petroquímica. Actualmente, el diseño de equipos se basa en modelos empíricos o determinísticos que, carecen de la información necesaria para realizar estimaciones precisas de las variables pertinentes. Ante esta problemática, se propone el desarrollo de una red neuronal que se enfoca en la temperatura crítica como parámetro principal. Esta red utiliza matrices de distancia y conectividad para representar la información de la estructura molecular de las sustancias. Adicionalmente, usa descriptores como peso molecular y electronegatividad. Los resultados conllevan a estimaciones de propiedades críticas, especialmente para sustancias orgánicas, partiendo de matrices de conectividad. Este enfoque promete contribuir al avance tecnológico en la industria química colombiana al proporcionar herramientas más precisas y eficientes para el diseño de procesos y equipos. Además, abre nuevas oportunidades para investigaciones futuras en el campo de la química computacional y la inteligencia artificial aplicada a la ingeniería química con el fin de satisfacer las necesidades industriales y sociales.

Abstract

This study focuses on the application of neural networks in chemical engineering for the accurate estimation of critical properties, such as pressure and critical temperature, fundamental in the design of chemical plants. In Colombia, the chemical industry faces a major challenge due to the lack of technological advances in the design of equipment for key sectors such as agriculture, food, pharmaceuticals and petrochemicals. Currently, equipment design is based on empirical or deterministic models that lack the necessary information to make accurate estimates of the relevant variables. In view of this problem, we propose the development of a neural network that focuses on the critical temperature as the main parameter. This network uses distance and connectivity matrices to represent the molecular structure information of the substances. Additionally, it uses descriptors such as molecular weight and electronegativity. The results lead to estimates of critical properties, especially for organic substances, based on connectivity matrices. This approach promises to contribute to technological progress in the Colombian chemical industry by providing more accurate and efficient tools for process and equipment design. It also opens new opportunities for future research in the field of computational chemistry and artificial intelligence applied to chemical engineering to meet industrial and societal needs.

Índice de Contenidos

Índice de Figuras

Índice de Gráficas

Índice de Tablas

Introducción

La importancia de las propiedades críticas radica en que son el puente que se tiene para calcular propiedades térmicas y fenomenológicas en sustancias puras y de mezclas. Las propiedades críticas de las sustancias son aquellas que definen su comportamiento físico y químico en condiciones extremas, como temperaturas y presiones críticas elevadas o muy bajas.

Existen modelos que permiten estimar las propiedades de las sustancias utilizando datos característicos, como su estructura molecular. Entre estos modelos, destacan los QSAR/QSPR (Modelos Cuantitativos para la Relación Estructura – Actividad o Modelos Cuantitativos para la Relación Estructura Propiedad), que establecen relaciones entre la estructura química y las propiedades fisicoquímicas de los compuestos. Estos modelos utilizan variables estructurales simples, como números de átomos, peso molecular, momento de inercia, área superficial y volumen molar. (Ahirwar et al., 2022).

La forma más precisa para determinar las propiedades crítica de una sustancia es por medio de análisis experimentales en el laboratorio. Esto implica encontrar los puntos de intercambio de fase bajo diferentes condiciones de temperatura y presión. A menudo, se efectúan experimentos específicos para identificar estas transiciones de fase, como el punto crítico, la presión crítica y la temperatura crítica. En algunos casos, los datos obtenidos en los experimentos pueden ser complementados mediante técnicas de interpolación para estimar valores en condiciones no directamente medidas, esto permite una caracterización más completa y precisa de las propiedades críticas (Prausnitz et al., 2000)

Uno de los modelos ampliamente utilizados en ingeniería química es el modelo UNIFAC, que se basa en la división de moléculas en grupos funcionales más pequeños para caracterizar las interacciones entre ellos. Este método utiliza valores experimentales o relaciones de grupo para describir las interacciones entre pares de grupos y luego implementa ecuaciones termodinámicas, como la ecuación de UNIQUAC, para calcular los coeficientes de actividad de cada componente de la mezcla. El modelo UNIFAC ofrece resultados confiables para una amplia gama de mezclas no ideales con diversos grupos funcionales, lo que lo hace invaluable en la simulación de procesos. Además del modelo UNIFAC, existen otros métodos computacionales, como los basados en dinámica molecular, que ofrecen mayor precisión. Sin embargo, estos métodos pueden ser prohibitivamente costosos en términos computacionales, lo que limita su aplicabilidad en ciertos contextos de ingeniería (Bettina, 2011).

Una solución moderna para esta problemática es la implementación de los algoritmos de inteligencia artificial (IA). Estos algoritmos se definen como un conjunto de instrucciones detalladas diseñadas para realizar una tarea específica, permitiéndoles aprender, razonar y tomar decisiones de manera inteligente (Forero-Corba & Bennasar, 2024).

En esta investigación, se aplica un algoritmo de inteligencia artificial de aprendizaje automático supervisado, utilizando las condiciones críticas de fondo como etiquetas de alimentación. Este enfoque se emplea para estimar la temperatura crítica y descubrir patrones y correlaciones en un conjunto de datos que a menudo no son evidentes para los investigadores humanos.

A nivel general, los modelos de inteligencia artificial mejoran con el tiempo a medida que se ingresan más datos para su aprendizaje, lo que reduce su desviación estándar al reentrenarse y mejora la precisión conforme se dispone de más información. Este aspecto los hace

especialmente útiles en áreas de investigación donde la adquisición de datos es difícil o costosa. Por lo tanto, el modelo propuesto en este proyecto se distingue de otras aplicaciones de sistemas computacionales al estimar la temperatura crítica de las sustancias a partir de su estructura, además de utilizar otros descriptores moleculares.

Justificación

La industria de la ingeniería química se enfrenta a una serie de desafíos cruciales en la actualidad, impulsados por factores como los avances tecnológicos, las preocupaciones ambientales y las cambiantes demandas del mercado. La necesidad de adoptar un enfoque más sostenible, dinámico y adaptable en el diseño y operación de plantas químicas es más apremiante que nunca.

En el ámbito de la ingeniería química, el diseño de equipos y procesos depende en gran medida de la disponibilidad de datos precisos sobre las características, así como sobre las propiedades de las sustancias involucradas en cada etapa de los procesos. Estos datos son fundamentales para desarrollar simulaciones y análisis precisos que permitan optimizar el rendimiento y la seguridad de los procesos químicos.

A pesar de la importancia de contar con datos precisos, la obtención de información como las propiedades críticas, las constantes de Antoine, los coeficientes de interacción entre pares, los calores de formación o los calores específicos puede resultar un desafío significativo en ingeniería química. Estos datos no siempre están disponibles para todas las moléculas o pueden ser difíciles de acceder, ya que se encuentran dispersos en diferentes bases de datos o simplemente no están disponibles públicamente y pueden ser costosos de estimar en el laboratorio. Esta falta de información puede conducir a simulaciones y análisis inexactos, lo que afecta negativamente la toma de decisiones y el rendimiento de los procesos en la industria química. Por lo tanto, abordar la disponibilidad y accesibilidad de estos datos es crucial para mejorar la eficiencia y la precisión en el diseño y operación de sistemas químicos.

Las plantas químicas modernas se enfrentan a la necesidad de adaptarse rápidamente a los cambios en el entorno, como la disponibilidad de materias primas, las condiciones del mercado y

los estándares regulatorios ambientales. Para hacer frente a estos desafíos dinámicos, el diseño de plantas versátiles, modulares y flexibles se vuelve indispensable. En este contexto, las herramientas de simulación de procesos, la optimización y el diseño asistido por computadora juegan un papel fundamental en la creación de plantas químicas que puedan procesar materias primas diversificadas y producir una amplia gama de productos finales. Sin embargo, la integración de tecnologías de la industria 4.0, como la *inteligencia artificial* y el *internet de las Cosas*, ha llevado esta capacidad un paso más allá al establecer una mayor eficiencia, control y toma de decisiones en tiempo real. Por lo tanto, la adopción de estas tecnologías es crucial para garantizar la competitividad y sostenibilidad de las plantas químicas en la era moderna.

Esta aplicación permite superar desafíos y barreras en el diseño de equipos y plantas químicas, contribuyendo a la reducción de costos y fomentando un progreso tecnológico constante. Al entrenar una red neuronal, es posible predecir información sobre sustancias para las cuales no se dispone de datos iniciales, pero que son esenciales en los procesos industriales. Este enfoque innovador no solo mejora la eficiencia y precisión en el diseño y operación de sistemas químicos, sino que también abre nuevas oportunidades para la investigación y desarrollo en la industria química.

Planteamiento del Problema

La estimación precisa de propiedades termodinámicas desempeña un papel fundamental en una amplia gama de aplicaciones en la ingeniería química, desde el diseño de procesos hasta la investigación de nuevos materiales.

En este contexto, el desarrollo de un algoritmo de red neuronal supervisada para la estimación de propiedades críticas de las sustancias emerge como una alternativa prometedora con el potencial de superar las limitaciones mencionadas y abrir nuevas posibilidades. Las redes neuronales tienen la capacidad de aprender las complejas relaciones entre la estructura molecular y las propiedades termodinámicas a partir de grandes conjuntos de datos. Este enfoque simplificará los modelos de dinámica molecular, permitiendo obtener datos precisos para cualquier estructura molecular sin recurrir a simulaciones costosas que pueden llevar meses o incluso años.

Objetivos

Objetivo General

Desarrollar un algoritmo que implemente una red neuronal para estimar la temperatura crítica de sustancias utilizando descriptores topológicos moleculares.

Objetivos Específicos

- Identificar y seleccionar los descriptores topológicos relevantes que serán utilizados como entradas para el modelo de red neuronal.
- Diseñar las heurísticas necesarias para la implementación del algoritmo, incluyendo la estructura y configuración de la red neuronal, así como otras técnicas definidas.
- Entrenar y ajustar el modelo de red neuronal utilizando datos de entrenamiento adecuados, con el fin de optimizar su rendimiento en la estimación de propiedades críticas

Revisión Literaria

La ingeniería química se encuentra en un momento crucial en el que la inteligencia artificial emerge como un catalizador del cambio, transformando la forma en que se toman decisiones, se optimizan procesos y se diseñan plantas más eficientes y sostenibles (Venkatasubramanian, 2019). Tradicionalmente, las decisiones de diseño se han basado en la experiencia y el conocimiento empírico de los ingenieros. Sin embargo, la inteligencia artificial ofrece un enfoque más sistemático y riguroso al analizar grandes volúmenes de datos y patrones complejos para identificar soluciones óptimas. Esto se refleja en la elección de métodos de separación y purificación de sustancias, la selección de solventes para extracciones líquido-líquido, la implementación de prácticas de química verde para crear plantas sostenibles, la optimización de redes de calor para la generación y transporte de energía, la selección de catalizadores para reducir costos de diseño y recuperación, el control de procesos, la estimación de curvas cinéticas y de selectividad para diferentes reacciones, y el diseño de diagramas de equilibrio químico o de fases, entre otros aspectos.

Por otro lado, es comprensible que los avances en algoritmos en el campo de la ingeniería química no hayan sido tan pronunciados como en otras áreas. Esto se debe a que modelos como las máquinas de soporte vectorial o las redes neuronales, ya sean supervisadas o no supervisadas, dependen en gran medida de datos de entrenamiento, los cuales no son tan accesibles en la nube de información (Zavala, 2023). Por ejemplo, es posible encontrar millones de imágenes etiquetadas o textos legislativos, o cientos de miles de datos para modelos de supervisión, pero los datos necesarios para el diseño de equipos y procesos no son igual de accesibles para la creación de algoritmos de entrenamiento.

Dentro de los modelos empleados en ingeniería química, se encuentra como punto de partida la estimación de propiedades fisicoquímicas de las sustancias utilizadas en las plantas industriales. Estas estimaciones se basan convencionalmente en correlaciones fundamentadas en cinco grupos de datos esenciales, denominados 'propiedades basales' (Prausnitz et al., 2000):

- *Puntos Críticos:* Representan las condiciones de temperatura y presión en las que una sustancia pura cambia de estado líquido a gas, lo que proporciona una comprensión del comportamiento de la sustancia en diferentes condiciones de temperatura y presión.
- *Constantes de Cambios de Fase:* Incluyen la entalpía de vaporización y la entalpía de fusión, que indican la energía requerida para los cambios de fase, como la evaporación, la condensación y la fusión (Prausnitz et al., 2000).
- *Calores de Formación Estándar:* Describen la energía absorbida o liberada durante la formación de un mol de una sustancia pura desde sus elementos en estado estándar, lo que permite calcular el calor de reacción y el equilibrio químico en varios procesos.
- *Coeficientes de Actividad:* Describen la desviación del comportamiento termodinámico de una sustancia en comparación con su estado puro, siendo esenciales para modelar mezclas no ideales, como soluciones electrolíticas y mezclas no azeotrópicas (Prausnitz et al., 2000).
- *Calores Específicos:* Representan la energía necesaria para aumentar la temperatura de un mol de una sustancia en un grado Celsius, lo que es fundamental para analizar la transferencia de calor y diseñar sistemas de intercambio de calor eficientes.

- *Coeficientes de Interacción Entre Pares:* Son coeficientes que ayudan a calcular las propiedades en exceso para mezclas de sustancias.

Estas correlaciones, fundamentadas en datos basales, permiten obtener valores confiables de propiedades como densidad, viscosidad, conductividad térmica, presión de vapor, tensión superficial y solubilidad. La precisión de estas estimaciones depende de la calidad y cantidad de datos disponibles, así como de la complejidad de las correlaciones. A medida que se recopilan más datos y se desarrollan métodos de correlación más avanzados, la precisión de las estimaciones continúa mejorando.

Para obtener los datos basales, se fundamenta en la teoría de Prausnitz y otros investigadores anteriores. Según esta teoría, la información sobre las propiedades de una sustancia es inherente a su estructura molecular. Es decir, si se conoce la forma tridimensional de la molécula, se pueden determinar sus propiedades. La estructura molecular se define por la posición de cada átomo en el espacio y las propiedades atómicas de estos puntos. (Prausnitz et al., 2000)

En algunos casos, para los datos de entrada, se implementa un sistema de codificación molecular simplificado (SMILES). SMILES representa las estructuras moleculares como cadenas de texto, lo que facilita el procesamiento de datos moleculares por parte de las redes neuronales. La teoría de grafos se utiliza para convertir las estructuras moleculares en cadenas de texto SMILES.

La información de los puntos en el espacio se puede representar mediante diversas formas no absolutas. Además de la representación tridimensional tradicional, se pueden emplear otras formas para describir la estructura molecular, como las matrices de distancia y las matrices de

conectividad. Las matrices de distancia proporcionan información sobre las distancias entre todos los pares de átomos en la molécula, mientras que las matrices de conectividad describen cómo están enlazados los átomos entre sí. Estas representaciones alternativas son útiles para ciertos tipos de análisis y pueden facilitar el procesamiento computacional de las estructuras moleculares.

Junto a estas descripciones topológicas, es importante conocer información específica de cada átomo para determinar completamente la estructura de la molécula. Según Rayner Canham (2000), estos aspectos incluyen:

- *Número Atómico:* El número de protones en el núcleo de un átomo, que determina su identidad y posición en la tabla periódica.
- *Peso Atómico:* La masa promedio de los átomos de un elemento, ponderada según la abundancia de sus isótopos naturales.
- *Volumen Atómico:* El volumen ocupado por un mol de átomos de un elemento, que se relaciona con el tamaño y la disposición de los átomos en la estructura cristalina.
- *Electronegatividad:* La capacidad de un átomo para atraer electrones hacia sí mismo en una molécula, influenciando la distribución de electrones en los enlaces químicos.

Al aplicar descriptores en IA, se considera el aprendizaje automático, una rama de la inteligencia artificial que se enfoca en desarrollar algoritmos capaces de aprender de los datos y mejorar su rendimiento con el tiempo. Estos algoritmos pueden identificar patrones y tendencias en los datos, aplicándose en tareas como predicción, clasificación y toma de decisiones (Godoy Viera, 2017).

El aprendizaje automático se basa en disciplinas como matemáticas, estadísticas, teoría de control e informática. Los algoritmos aprenden a partir de ejemplos de entrenamiento, utilizando conjuntos de datos que contienen información sobre variables de entrada y salida. Durante el entrenamiento, identifican relaciones entre estas variables para predecir nuevos datos sin etiquetas.

Existen tres categorías principales de técnicas de aprendizaje automático:

- *Aprendizaje No Supervisado:* Los algoritmos no reciben información sobre las variables de salida y deben identificar patrones y estructuras en los datos por sí mismos (Stack, 2021). Ejemplos incluyen análisis de clústeres y reducción de dimensionalidad.
- *Aprendizaje Supervisado:* Los algoritmos reciben información sobre las variables de salida y se entrenan para realizar predicciones precisas sobre nuevas entradas. Ejemplos incluyen regresión, clasificación y aprendizaje por refuerzo (Stack, 2021).
- *Aprendizaje Semi-Supervisado:* Los algoritmos reciben información sobre algunas variables de salida, pero no todas. Ejemplos incluyen aprendizaje activo y etiquetado múltiple.

En el ámbito de QSPR, las técnicas de aprendizaje automático más utilizadas son:

- *Reglas de Naive Bayes:* Se basan en el teorema de Bayes y asumen independencia entre atributos (López de Castilla Vásquez, 2019). Son eficientes para tareas de clasificación.

- *Algoritmos Bioinspirados:* Resuelven problemas inspirados en procesos naturales o biológicos, como algoritmos genéticos y de enjambre de partículas (Alander, n.d.).
- *Máquinas de Vectores de Soporte (SVM):* Ampliamente utilizadas para clasificación o regresión, encuentran un hiperplano que separa eficientemente los datos (García Díaz & Lozano Martínez, 2006).
- *Redes Neuronales Artificiales (ANN):* Basadas en el cerebro humano, estas redes pueden aprender de los datos y son versátiles en tareas como clasificación, predicción y reconocimiento de patrones (Basheer & Hajmeer, 2000).

Dentro de las redes neuronales artificiales (ANN), una subcategoría prominente son las redes neuronales convolucionales (CNN). Estas CNN son ampliamente reconocidas por su capacidad para extraer características específicas de los datos de entrada, lo que las hace especialmente útiles para analizar estructuras moleculares de proteínas. En particular, las proteínas son moléculas tridimensionales complejas que desempeñan roles fundamentales en los sistemas biológicos, y su función está estrechamente relacionada con su estructura tridimensional. Las CNN pueden analizar eficazmente las características topológicas y estructurales de las proteínas, identificando patrones clave que pueden estar relacionados con su función biológica. Esta capacidad de las CNN ha sido aprovechada en diversos campos de investigación, como la predicción de la estructura de proteínas (por ejemplo, el proyecto AlphaFold), el diseño de fármacos y la identificación de sitios de unión a ligandos (Jumper et al., 2021).

Las CNN funcionan aplicando operaciones de convolución a los datos de entrada. Estas operaciones utilizan 'núcleos', que son matrices de pesos, para detectar patrones y secuencias en

los datos de entrada. Esta capacidad de detección de patrones hace que las CNN sean ideales para analizar la estructura molecular ordenadas.

Algunos estudios han explorado el uso de las ANN y las CNN para este propósito y han logrado resultados prometedores. Por ejemplo, Zhang et al, (2018) presentaron un esquema para simulaciones moleculares conocido como Método de Dinámica Molecular de Potencial Profundo (DPMD), el cual se basa en un potencial de muchos cuerpos y fuerzas interatómicas generadas por una red neuronal profunda, diseñada y entrenada cuidadosamente con datos ab initio. Este enfoque utiliza un modelo de red neuronal que conserva todas las simetrías naturales del problema y está fundamentado en los primeros principios, lo que significa que no incorpora componentes ad hoc aparte del propio modelo de red. Los resultados demuestran que el esquema propuesto proporciona un protocolo eficiente y preciso para una variedad de sistemas, incluyendo materiales a granel y moléculas. En todas estas aplicaciones, el DPMD ofrece resultados esencialmente indistinguibles de los datos originales, con un costo computacional que aumenta linealmente con el tamaño del sistema.

Lie et al (2020)se enfocaron en el diseño de una red híbrida (HNN) que combina una red neuronal convolucional (CNN) con una red neuronal de memoria a corto plazo (LTSM). Su objetivo era extraer características de los materiales para predecir la temperatura crítica (Tc) de superconductores. Alimentaron las redes neuronales con datos de 73,452 compuestos inorgánicos y una representación vectorial de 87 átomos. Entrenaron la HNN con 12,413 superconductores, obteniendo resultados experimentales satisfactorios. La HNN demostró una alta precisión en la predicción de la temperatura crítica al extraer efectivamente las relaciones entre los superconductores y los átomos.

Por otro lado, Scalia et al (2020) se basaron en el aprendizaje de máquina y los modelos en gráficos para modelar la temperatura crítica, la presión y el volumen. Implementaron técnicas como bootstrap, ensemble y dropout para cuantificar la incertidumbre de la predicción. Su estudio se enmarca en los avances recientes en la predicción de propiedades moleculares basada en redes neuronales profundas (DNN), con un enfoque particular en las redes neuronales convolucionales gráficas (GCNN), que presentan el rendimiento más avanzado para esta tarea. El desafío principal abordado por Scalia et al (2020) es la cuantificación de la incertidumbre en las predicciones, lo cual es crucial para evaluar la confiabilidad de los modelos, especialmente cuando se trata de moléculas fuera del dominio de entrenamiento. En su estudio, comparan cuantitativamente técnicas escalables para la estimación de la incertidumbre en GCNNs, como MC-dropout, Deep Ensembles y bootstrapping. Los resultados indican que tanto Deep Ensembles como bootstrapping superan consistentemente a MC-dropout en la reducción de errores relacionados con la incertidumbre, aunque cada uno presenta pros y contras específicos en diferentes contextos. Esto destaca la importancia de abordar la incertidumbre en la predicción de propiedades moleculares y ofrece una visión más completa sobre su impacto en la precisión de los modelos.

A partir de estudios previos, se ha confirmado que las CNN y las ANN son herramientas fundamentales para la estimación de propiedades de las sustancias. Estas redes tienen la capacidad de aproximar con precisión la representación molecular o estructural correspondiente a cada propiedad, lo que las hace ideales para diversas aplicaciones de QSPR.

Sin embargo, la precisión de los modelos QSPR basados en redes neuronales convolucionales (CNN) y redes neuronales artificiales (ANN) depende en gran medida de la calidad y confiabilidad de los datos de entrenamiento. A menudo, los datos reportados en tablas o

bases de datos abiertas pueden no ser completamente confiables, ya que pueden contener errores, inconsistencias o sesgos. Estos errores pueden surgir de diversas fuentes, como la variabilidad en los métodos experimentales utilizados para medir las propiedades fisicoquímicas, la falta de estandarización en la presentación de los datos o incluso la presencia de datos incorrectos o mal etiquetados. Por lo tanto, se vuelve indispensable realizar una minería de datos exhaustiva para evaluar la calidad y confiabilidad de los datos de entrenamiento antes de utilizarlos para entrenar la red neuronal. Esto implica no solo la verificación de la precisión de los datos reportados, sino también la identificación y corrección de cualquier error o inconsistencia. Además, puede ser necesario complementar los datos existentes con experimentos adicionales o cálculos teóricos para obtener una representación más completa y precisa de las propiedades fisicoquímicas de las moléculas. Solo mediante un proceso riguroso de recopilación y validación de datos se puede garantizar que los modelos QSPR basados en CNN y ANN proporcionen resultados confiables y útiles para la predicción de propiedades moleculares.

Metodología

El modelado de la red neuronal en esta investigación aborda la dispersión de los datos mediante una regresión adaptativa, personalizando así la estimación de la temperatura crítica. A diferencia de los datos con clasificaciones binarias, la dispersión de los datos no sigue una función sigmoidal, lo que exige una adaptación específica de la red. Durante cada ciclo de entrenamiento, la red realiza operaciones matriciales para alcanzar predicciones cercanas a los valores esperados. Esta metodología asegura la integridad y precisión de los métodos empleados en el proceso de estimación.

Para llevar a cabo este estudio de investigación, se dividió en diferentes etapas como se observa en la Figura 1, con el fin de garantizar el cumplimiento de los requisitos funcionales y no funcionales de la red neuronal.

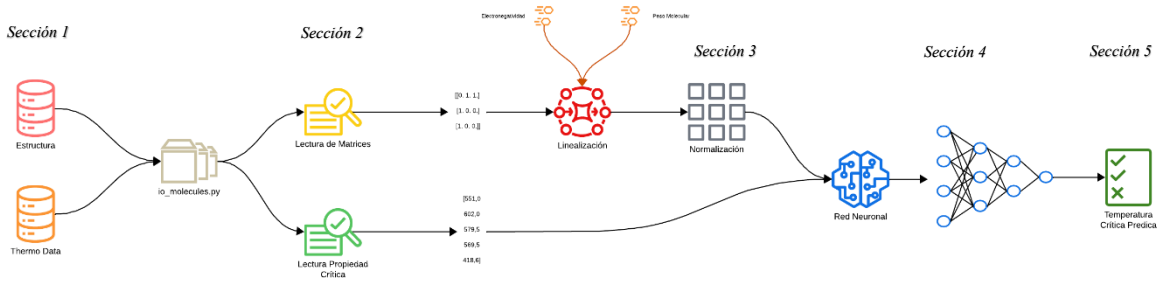


Figura 1: Esquema General de la Metodología

Recopilación de Datos (Sección 1)

Se trabajó con un conjunto de 435 sustancias con diversas características. Para alimentar y entrenar la red neuronal, fue necesario obtener su información matricial, incluyendo la matriz de

distancia y la matriz de conectividad. Estas matrices se obtuvieron de la base de datos de la Universidad de Alcalá (UAH) de Madrid en formato .mol. Un ejemplo de la estructura del archivo .mol para el amoníaco se muestra en la Figura 2:

```
molecula
__Jmol-14_03182322313D 1   1.00000     0.00000     0
Jmol version 14.6.4_2016.10.26  2016-10-26 12:26 EXTRACT: ({0:3})
  4  3  0  0  0  0              1 V2000
   0.00000   0.00000   0.05500 N   0  0  0  0  0  0
  -0.01280  -0.96010  -0.25500 H   0  0  0  0  0  0
   0.83790   0.46900  -0.25500 H   0  0  0  0  0  0
  -0.82510   0.49110  -0.25500 H   0  0  0  0  0  0
  1  2  1  0  0  0
  1  3  1  0  0  0
  1  4  1  0  0  0
M  END
```

Figura 2: Estructura del Archivo .mol para el Amoniaco

Posteriormente, se determinaron las propiedades críticas de las sustancias, con un enfoque especial en la temperatura. Estos datos se recopilaron en una base de datos en formato .xlsx para proporcionar a la red neuronal la información de los datos de salida, con el objetivo de que sus predicciones se acercaran lo más posible a los valores reales o esperados.

Clasificación de las Sustancias

Dentro del archivo .xlsx, se clasificaron las sustancias en orgánicas e inorgánicas, con un total de 396 sustancias orgánicas y 39 sustancias químicas inorgánicas. En el caso de las sustancias orgánicas, se realizó una segunda clasificación basada en el número de carbonos y los grupos funcionales presentes. El rango para el número de carbonos fue de 1 a 21, y se identificaron las siguientes subcategorías de grupos funcionales:

- Alcoholes: 29 sustancias
- Aromáticos: 37 sustancias
- Haluros: 34 sustancias
- Cetonas y Aldehídos: 13 sustancias
- Ácidos (Ésteres, Ácidos Carboxílicos, Amidas): 44 sustancias
- Otros (Alcanos, Alquenos, Alquinos, etc): 239 sustancias

La clasificación de las sustancias se llevó a cabo con el respaldo de referencias bibliográficas reconocidas en el campo de la química orgánica. Se utilizaron criterios establecidos en la literatura especializada para identificar y categorizar las sustancias según su composición química y grupos funcionales presentes.

Definición del Lenguaje de Programación y Editor de Código

Para el diseño de la red neuronal, se optó por utilizar Python 3.11.2 como lenguaje de programación principal. Python se seleccionó debido a su reputación como un lenguaje de alto nivel, reconocido por su sintaxis simple y clara, lo que facilitó el enfoque en el desarrollo de la red en sí, en lugar de lidiar con las complejidades inherentes a otros lenguajes de programación. Además, Python ofrece una amplia gama de bibliotecas y frameworks especializados en aprendizaje automático y redes neuronales, como TensorFlow, PyTorch, Keras y scikit-learn. Estas bibliotecas proporcionan funciones predefinidas para tareas fundamentales como el manejo de datos, la construcción de modelos, el entrenamiento y la evaluación de redes neuronales, simplificando en gran medida el proceso de desarrollo.

Sin embargo, dada la naturaleza de las sustancias químicas y la necesidad de diseñar una red neuronal específica para capturar las correlaciones presentes en los datos, fue necesario crear la red desde cero. Aunque Python ofrece una amplia gama de herramientas preconstruidas, el diseño personalizado de la red neuronal permitió adaptarla específicamente a las características de los datos químicos.

Para facilitar el proceso de codificación, se utilizó Sublime Text como editor de código. Sublime Text es una herramienta ligera y versátil que ofrece una interfaz intuitiva para el desarrollo de software. Su capacidad para manejar grandes archivos y códigos complejos simplificó significativamente el proceso de codificación y depuración.

Estructura del Código (Sección 2 y 3)

El código se organiza en dos partes principales: la primera parte se refiere a los diferentes métodos donde se ejecuta la red neuronal con diversas configuraciones para predecir la propiedad crítica de las sustancias. Las posibles configuraciones incluyen:

- Matriz de Distancia y Sustancia Orgánicas
- Matriz de Distancia y Sustancia Inorgánicas
- Matriz de Conectividad y Sustancia Orgánicas
- Matriz de Conectividad y Sustancia Inorgánicas

En la segunda parte se encuentran los datos y la infraestructura necesarios para que la aplicación funcione. Esta sección alberga los métodos encargados de extraer información de los archivos .mol y .xlsx, así como los métodos que intervienen en cada uno de los procesos de la red

neuronal. Se mantiene una interacción constante entre estos métodos para garantizar el funcionamiento adecuado del software.

Entrenamiento de la Red (Sección 4)

Para entrenar la red neuronal, se implementó un proceso de validación conocido como validación cruzada cuádruple (Naranjo et al., 2013), el cual sigue el siguiente mecanismo:

1. El conjunto de datos se divide aleatoriamente en cuatro subconjuntos iguales.
2. Se entrena el modelo de red neuronal utilizando tres de los subconjuntos como conjunto global de entrenamiento y el cuarto subconjunto como conjunto de validación.
3. Este proceso se repite cuatro veces, utilizando cada subconjunto como conjunto de validación una vez.
4. El rendimiento promedio del modelo en los cuatro conjuntos de validación se utiliza como una estimación del rendimiento general del modelo.

Esta técnica de validación cruzada es útil para evaluar el rendimiento del modelo desarrollado y evitar el sobreajuste. Al entrenar el modelo en diferentes subconjuntos de datos, se obtiene una estimación más precisa de cómo se generalizaría o comportaría el modelo con nuevos datos no vistos.

Salida de Datos (Sección 5)

Al finalizar el entrenamiento, el programa proporciona como salida un archivo .xlsx que contiene:

- El listado de las sustancias empleadas para la predicción.
- La temperatura crítica original de cada sustancia.
- La temperatura crítica predicha por el modelo.
- Para el caso de sustancias orgánicas, se incluyen el número de carbonos y los grupos funcionales correspondientes.
- El error por diferencia de valores entre la temperatura crítica original y la predicha.
- El error porcentual de la predicción.

Esta información permite analizar el comportamiento de las sustancias en función del número de carbonos o grupos funcionales, así como la desviación estándar respecto al valor meta.

Documentación del Código

Parte 1

El archivo que permite la interacción con la red neuronal se llama test.py y su estructura es la siguiente:

Librerías y Métodos

```
import pandas as pd
import numpy as np
import io_molecules
import seaborn as sns
from molecule_nn import NeuralNetwork
from sklearn.model_selection import train_test_split
from io_molecules import atomic_electronegativity, atomic_weight
```

- Pandas: Permite la creación de documentos en Excel, facilitando la visualización de la salida de los datos.
- Numpy: Establece el mecanismo para realizar cálculos numéricos y analizar datos mediante la creación de listas o arrays dentro del código.
- Io_molecules: En este caso, se importan todos los métodos del archivo io_molecules para la búsqueda de los archivos .mol y el archivo .xlsx que contiene las propiedades críticas de las sustancias.
- From molecule_nn import NeuralNetwork: Llama a la red neuronal para proporcionarle los datos de entrenamiento y predicción.
- From sklearn.model_selection import train_test_split: Esta librería importa el método que permite dividir las sustancias en cuatro grupos iguales para el entrenamiento.
- From io_molecules import atomic_electronegativity, atomic_weight: En este caso, se llaman los diccionarios que contienen los valores de electronegatividad y peso

molecular de los elementos de cada sustancia, que funcionan como descriptores numéricos en las operaciones matriciales de la red neuronal.

Dirección

```
path = "src/Estructura/"
```

Esta línea establece la ubicación dentro del computador donde se encuentran los 435 archivos .mol que contienen la información de las matrices de conectividad y distancia.

División de Datos

```
def split_data(matrix, element_lists, temperatures, names, test_size=0.25, random_state=None):
    matrix_train, matrix_test, elements_train, elements_test, temperatures_train, temperatures_test, names_train, names_test = train_test_split(matrix, element_lists, temperatures, names, test_size=test_size, random_state=random_state)
    return matrix_train, matrix_test, elements_train, elements_test, temperatures_train, temperatures_test, names_train, names_test
```

El método split_data divide los datos para llevar a cabo el proceso de validación cruzada cuádruple. Toma como entrada las matrices, las listas de elementos, las temperaturas y los nombres de las sustancias, tanto para la predicción como para el entrenamiento. La división se realiza utilizando el método train_test_split, que divide los datos en conjuntos de entrenamiento y prueba, con el tamaño especificado para el conjunto de prueba. El conjunto de entrenamiento se utiliza para entrenar el modelo de red neuronal, mientras que el conjunto de prueba se utiliza para evaluar su rendimiento. Esto se repite cuatro veces para cada combinación de conjuntos de entrenamiento y prueba, lo que permite una evaluación más robusta del modelo y ayuda a evitar el sobreajuste. Finalmente, el

método devuelve los conjuntos de datos divididos para su posterior uso en el proceso de entrenamiento y predicción.

Propiedad Crítica

```
names, temperatures = io_molecules.get_data_substances('CRITICAL TEMPERATURA')
```

En esta línea, se utiliza el método get_data_substances del módulo io_molecules para obtener los nombres y temperaturas críticas de las sustancias que se encuentran en la base de datos en formato .xlsx. El parámetro 'CRITICAL TEMPERATURE' indica que se están extrayendo los datos específicamente relacionados con la temperatura crítica de las sustancias. Los nombres y temperaturas críticas se asignan a las variables names y temperatures, respectivamente, para su posterior uso en el proceso de entrenamiento y predicción de la red neuronal.

Matriz de Conectividad o Distancia

```
matrix = []
element_lists = []
for name in names:
    conectivity_matrix, elements = io_molecules.get_conectivity_matrix(name)
    #distance_matrix, elements = io_molecules.get_distances_matrix(name)
    matrix.append(conectivity_matrix.astype(float))
    #matrix.append(distance_matrix.astype(float))
    element_lists.append(elements)

matrix = np.array(matrix, dtype=object).reshape(-1,1)
element_lists = np.array(element_lists, dtype=object)
temperatures = np.array([temperatures]).T
```

- En este bloque de código, se realiza un bucle sobre la lista de nombres de las sustancias para obtener las matrices de conectividad correspondientes a cada una de ellas utilizando el método get_conectivity_matrix del módulo io_molecules.

- Posteriormente, se agregan estas matrices de conectividad a la lista matrix, mientras que los elementos asociados a cada matriz se agregan a la lista element_lists.
- Las líneas comentadas indican que es posible cambiar la configuración del proceso, permitiendo alternar entre la obtención de matrices de conectividad y matrices de distancia según sea necesario.
- Una vez completada la iteración, se convierten las listas matrix y element_lists en matrices NumPy y se modifican sus dimensiones según sea necesario para su posterior procesamiento. Además, se convierte la lista temperatures en un array NumPy y se ajusta su forma para que quede en formato de columna.
- Estas líneas de código preparan los datos para ser utilizados en el proceso de entrenamiento y predicción de la red neuronal

Llamado de la Red Neuronal

```
if __name__ == "__main__":

    matrix_train, matrix_test, elements_train, elements_test, temperatures_train, temperatures_test, names_train, names_test = split_data(
        matrix, element_lists, temperatures, names)

    nn = NeuralNetwork(hidden_size=4, kind='regression')
    nn.train(matrix_train, elements_train, temperatures_train, epochs=30000, learning_rate=0.000105)

    matrix_test_e= nn.linealize_forward(matrix_test, elements_test, grade=1, property_type=atomic_electronegativity)
    matrix_test_w = nn.linealize_forward(matrix_test_e, elements_test, grade=2, property_type=atomic_weight)

    Xtest = np.array(matrix_test_w)
    predictions = nn.predict(Xtest)
```

Este bloque de código verifica si el script actual está siendo ejecutado como el programa principal (__name__ == "__main__"). Si es así, se ejecutan las siguientes instrucciones:

1. Los datos se dividen en conjuntos de entrenamiento y prueba utilizando la función split_data.
2. Se instancia un objeto NeuralNetwork con un tamaño de capa oculta de 4 y un tipo de red neuronal de regresión.
3. Se entrena la red neuronal utilizando los datos de entrenamiento (matrix_train, elements_train, temperatures_train) con un número específico de épocas (30000) y una tasa de aprendizaje de 0.000105.
4. Se realiza una transformación lineal de los datos de prueba (matrix_test) utilizando la función linealize_forward de la red neuronal para calcular las características específicas de los elementos, como la electronegatividad y el peso atómico.
5. Se realizan las predicciones utilizando la red neuronal entrenada en los datos de prueba transformados (Xtest), y las predicciones se almacenan en la variable predictions.

Determinación de los Errores

```
errores = [abs(orig - pred) for orig, pred in zip(temperatures_test.flatten(), predictions.flatten())]
porcentaje_error = [(error / orig) * 100 if orig != 0 else 0 for orig, error in zip(temperatures_test.flatten(), errores)]
carbon_number, functional_group = io_molecules.get_data_substances(None, names_test)
```

En este fragmento de código se calculan los errores y el porcentaje de error entre las temperaturas críticas reales (temperatures_test) y las temperaturas predichas (predictions). Además, se obtienen los números de carbono y los grupos funcionales de las sustancias de prueba.

1. errores: Se calcula la diferencia absoluta entre cada temperatura crítica real y su predicción correspondiente.
2. porcentaje_error: Se calcula el porcentaje de error dividiendo el error absoluto entre la temperatura crítica real, multiplicado por 100. Si la temperatura crítica real es cero, el porcentaje de error se establece en cero para evitar divisiones por cero.
3. carbon_number, functional_group: Se obtienen los números de carbono y los grupos funcionales de las sustancias de prueba utilizando el método get_data_substances de io_molecules, pasando como argumento None para indicar que se desean obtener todos los datos disponibles y los nombres de las sustancias de prueba (names_test).

Salida de Datos

```
    data = {'Sustancia': names_test.flatten(),
            'Temperatura Original': temperatures_test.flatten(),
            'Temperatura Predicha': predictions.flatten(),
            'Número de Carbonos': carbon_number.flatten(),
            'Grupo Funcional': functional_group.flatten(),
            'Error': errores,
            '% Error': porcentaje_error}

    df = pd.DataFrame(data)
    df.to_excel('resultados_prediccion.xlsx', index=False)
    print(df)

max_error_idx = df['% Error'].idxmax()
min_error_idx = df['% Error'].idxmin()

max_error_substance = df.loc[max_error_idx, 'Sustancia']
min_error_substance = df.loc[min_error_idx, 'Sustancia']

max_error_value = df.loc[max_error_idx, '% Error']
min_error_value = df.loc[min_error_idx, '% Error']

# Imprime los resultados minímos y máximos de errores
print("Máximo % Error:", max_error_value, "para la sustancia:", max_error_substance)
print("Mínimo % Error:", min_error_value, "para la sustancia:", min_error_substance)
```

En este bloque de código se crea un DataFrame de Pandas (df) con la información recopilada durante el proceso de predicción y se guarda en un archivo Excel llamado "resultados_prediccion.xlsx". Luego, se identifican los índices correspondientes a las mayores y menores discrepancias porcentuales entre las temperaturas críticas originales y las predicciones. Finalmente, se imprimen los detalles de las sustancias con los máximos y mínimos errores porcentuales.
Es importante tener en cuenta que max_error_idx y min_error_idx son los índices de las filas en el DataFrame df que contienen los máximos y mínimos errores porcentuales, respectivamente.

Parte 2

La segunda parte del sistema consta de dos archivos principales: molecule_nn.py y io_molecules.py.

1. **molecule_nn.py:** En este archivo se encuentra la estructura de la red neuronal. Contiene las definiciones de la clase NeuralNetwork que se encarga de la creación, entrenamiento y predicción de la red neuronal. También pueden incluirse otras clases o funciones auxiliares relacionadas con el funcionamiento interno de la red neuronal.
2. **io_molecules.py:** Aquí se encuentran los métodos de interacción con los archivos .mol y .xlsx. Estos métodos se encargan de leer y procesar la información contenida en los archivos de entrada, como las matrices de conectividad, los datos de temperatura crítica y otros descriptores de las sustancias. Además, pueden contener funciones para manipular y transformar los datos antes de ser utilizados por la red neuronal.

Molecule_nn.py

Librerías y Métodos

```
import numpy as np
from io_molecules import get_elements_properties_single
from io_molecules import atomic_electronegativity, atomic_weight
```

Este fragmento de código importa las siguientes librerías y funciones desde los archivos correspondientes:

1. numpy as np: Importa la biblioteca NumPy con el alias np. NumPy es una biblioteca fundamental para la computación científica en Python y proporciona soporte para matrices y operaciones matemáticas avanzadas
2. from io_molecules import get_elements_properties_single: Importa la función get_elements_properties_single desde el archivo io_molecules.py. Esta función se utiliza para obtener las propiedades de los elementos químicos, como la electronegatividad y el peso atómico, a partir de un solo elemento.
3. from io_molecules import atomic_electronegativity, atomic_weight: Importa las librerías atomic_electronegativity y atomic_weight desde el archivo io_molecules.py.

Función Sigmoidal

```
def sigmoid(X):
   return 1 / (1 + np.exp(-X))
```

La función sigmoid implementa la función de activación sigmoide, que transforma la entrada lineal en un rango de valores entre 0 y 1. Esta función es comúnmente utilizada en redes neuronales para introducir no linealidad y para mapear los valores de salida a una distribución de probabilidad.

Lista de Sustancias

```
def preprocess_element_lists(element_lists):
    processed_lists = [np.array(element_list) for element_list in element_lists]
    return processed_lists
```

Esta función preprocess_element_lists toma una lista de listas de elementos y la procesa para convertirla en una lista de matrices NumPy

- element_lists: Es una lista de listas que contiene los elementos de cada sustancia.

La función itera sobre cada lista de elementos en element_lists y la convierte en un array NumPy utilizando np.array(). Luego, devuelve la lista resultante de matrices NumPy.

El propósito de esta función es preparar los datos de los elementos para ser utilizados en el entrenamiento o la predicción de la red neuronal. Al convertir las listas de elementos en matrices NumPy, los datos se vuelven más fáciles de manipular y procesar en operaciones posteriores.

Normalización de Datos

```
def normalize_minmax(data):
    min_val = np.min(data)
    max_val = np.max(data)
    normalized_data = (data - min_val) / (max_val - min_val)
    return normalized_data
```

Esta función normalize_minmax toma un conjunto de datos y realiza una normalización min-max sobre ellos.

- data: Los datos que se van a normalizar.

La función calcula el valor mínimo y máximo en el conjunto de datos utilizando np.min() y np.max() respectivamente. Luego, aplica la fórmula de normalización min-max a cada dato para obtener su valor normalizado. Finalmente, devuelve los datos normalizados.

La normalización min-max es una técnica común utilizada para escalar los datos en un rango específico (generalmente entre 0 y 1), lo que puede ayudar a mejorar el rendimiento de algunos algoritmos de aprendizaje automático al hacer que los datos sean más comparables y estables.

Creación de Clase

```
class NeuralNetwork:
    def __init__(self, hidden_size, activation_function=None,
kind='classification'):
        self.hidden_size = hidden_size
        self.activation_function = activation_function
        self.kind = kind

    def initWeights(self, input_size, hidden_size, output_size):
        # Inicialización de los pesos de las capas oculta y de salida
        self.hidden_weights = np.random.randn(input_size, self.hidden_size)
        self.output_weights = np.random.randn(self.hidden_size, output_size)

        # Inicialización de los sesgos de las capas oculta y de salida
        self.hidden_bias = np.zeros((1, self.hidden_size))
        self.output_bias = np.zeros((1, output_size))

        # Inicialización de los pesos de electronegatividad y peso molecular
        self.WE = np.random.rand()
        self.WM = np.random.rand()
```

Esta clase NeuralNetwork es la estructura principal para definir una red neuronal. Aquí están los métodos y atributos principales:

- __init__: El método constructor que inicializa los atributos de la red neuronal. Toma tres argumentos obligatorios: hidden_size (tamaño de la

capa oculta), activation_function (función de activación) y kind (tipo de red, por defecto es 'clasificación').

- initWeights: Un método para inicializar los pesos de la red neuronal. Toma tres argumentos: input_size (tamaño de la capa de entrada), hidden_size (tamaño de la capa oculta) y output_size (tamaño de la capa de salida). Inicializa los pesos y sesgos tanto de la capa oculta como de la capa de salida, así como los pesos de electronegatividad (WE) y peso molecular (WM).

Propagación Hacia Adelante

```
def linealize_forward(self, matrices, element_lists, grade, property_type, alpha=0.000105):
    if property_type == atomic_electronegativity:
        W = self.WE
    elif property_type == atomic_weight:
        W = self.WM

    output_scalars = []
    salida = []

    for i in range(len(matrices)):
        P = get_elements_properties_single(element_lists[i], property_type)

        if grade == 1:
            output_scalar = np.sum((matrices[i] + np.identity(len(matrices[i]))) * W * P)
        elif grade == 2:
            output_scalar = np.sum(matrices[i] * W * P)
            salida.append(output_scalar)

        output_scalars.append(output_scalar)

    # Output_array almacena la segunda salida de la función (los valores escalres de las
    # sustancias) para retornarla de forma directa en el condicional
    if grade == 2:
        output_array = np.array([salida])
        output_array = normalize_minmax(output_array)
        return output_array.T

    return  output_scalars
```

El método linealize_forward realiza una operación de propagación hacia adelante en la red neuronal. Toma varios argumentos: matrices, element_lists, grade, property_type, y alpha. Calcula una salida escalar para cada matriz de entrada en base a la propiedad especificada (electronegatividad o peso molecular). Si el grado de la operación es 1, se realiza un cálculo especial de la salida escalar utilizando la matriz y la propiedad. Si el grado de la operación es 2, se realiza otro cálculo de salida escalar y se almacenan en una lista salida. Después de calcular las salidas escalares, se normalizan utilizando la función normalize_minmax. La salida normalizada se devuelve en forma de matriz transpuesta si el grado de la operación es 2; de lo contrario, se devuelve una lista de salidas escalares. Este método es esencial para la propagación hacia adelante en la red neuronal y maneja la entrada de datos y el cálculo de salidas de acuerdo con el tipo de propiedad especificada.

Retropropagacion

```
def linealize_backward(self, output_matrix, Y, learning_rate=0.000105):
    m = len(output_matrix)
    error = output_matrix - Y

    delta_WE = (1/m) * np.sum(error)
    delta_WM = (1/m) * np.sum(error)

    self.WE -= learning_rate * delta_WE
    self.WM -= learning_rate * delta_WM
```

El método linealize_backward realiza la retropropagación de errores en la red neuronal. Toma tres argumentos: output_matrix, Y y learning_rate. Calcula el error entre la salida predicha y la salida real. Luego, calcula la tasa de cambio de los pesos WE y WM (utilizados para la electronegatividad y el peso molecular,

respectivamente) con respecto al error. Estos cálculos se realizan utilizando el descenso de gradiente estocástico. Finalmente, actualiza los pesos WE y WM con la tasa de aprendizaje y las tasas de cambio calculadas. Este método es esencial para ajustar los pesos de la red neuronal durante el proceso de entrenamiento para minimizar el error entre las salidas predichas y las salidas reales.

Paso Hacia Delante de Red

```
        def forward(self, X):
            if self.activation_function:
                # Paso hacia adelante de la redd neuronal
                hidden_layer = self.activation_function(X @ self.hidden_weights +
self.hidden_bias)
                output_layer = self.activation_function(hidden_layer @
self.output_weights + self.output_bias)
            else:
                # Paso hacia adelante de la redd neuronal
                hidden_layer = X @ self.hidden_weights + self.hidden_bias
                output_layer = hidden_layer @ self.output_weights + self.output_bias

            return hidden_layer, output_layer
```

El método forward realiza el paso hacia adelante de la red neuronal. Toma un argumento X, que representa los datos de entrada. Si se especifica una función de activación, aplica esta función a la combinación lineal de los datos de entrada, los pesos de la capa oculta y los sesgos de la capa oculta. Luego, aplica la función de activación nuevamente a la combinación lineal de la capa oculta, los pesos de salida y los sesgos de salida para obtener la salida de la red neuronal. Si no se especifica una función de activación, realiza los mismos cálculos sin aplicar una función de activación. Finalmente, devuelve las activaciones de la capa oculta y la

salida de la red neuronal. Este método es esencial para propagar los datos de entrada a través de la red neuronal y obtener la salida correspondiente.

Paso Hacia Atrás de la Red

```
        def backward_regression(self, X, Y, hidden_layer, output_layer,
learning_rate=0.000105):
            # Paso hacia atrás de la redd neuronal (actualización de pesos y sesgos)
            m = len(X)

            error = output_layer - Y

            delta_output_weihts = (1/m) * hidden_layer.T @ error
            delta_output_bias = (1/m) * np.sum(error)
            delta_output_bias = np.float64(delta_output_bias)
            delta_hidden_weights = (1/m) * X.T @ error
            delta_hidden_bias = (1/m) * np.sum(error)
            delta_hidden_bias = np.float64(delta_hidden_bias)

            self.hidden_weights -= learning_rate *
delta_hidden_weights.astype(float)
            self.hidden_bias -= learning_rate *  delta_hidden_bias
            self.output_weights -= learning_rate * delta_output_weihts.astype(float)
            self.output_bias -= learning_rate *  delta_output_bias
```

El método backward_regression realiza el paso hacia atrás de la red neuronal para el caso de regresión, lo que implica la actualización de los pesos y sesgos en función del error entre la salida real y la salida predicha. Toma los datos de entrada X y las etiquetas Y, así como las activaciones de la capa oculta hidden_layer y la salida de la red output_layer. Calcula el error entre la salida predicha y la salida real. Luego, calcula los gradientes de los pesos y sesgos de la capa de salida y de la capa oculta. Finalmente, actualiza los pesos y sesgos de ambas capas utilizando el gradiente descendente y la tasa de aprendizaje especificada. Este método es esencial para entrenar la red neuronal mediante el ajuste de los pesos y sesgos en función del error de la salida.

Paso Atrás Clasificación

```
        def backward_classification(self, X, Y, hidden_layer, output_layer,
learning_rate):
            # Paso hacia atrás de la redd neuronal (actualización de pesos y sesgos)
            error = Y - output_layer
            delta_output = error * ((output_layer) * (1 - output_layer))
            delta_hidden = delta_output @ self.output_weights.T * (hidden_layer *
(1 - hidden_layer))

            self.output_weights += hidden_layer.T @ delta_output * learning_rate
            self.output_bias += np.sum(delta_output, axis=0) * learning_rate
            self.hidden_weights += X.T @ delta_hidden * learning_rate
            self.hidden_bias += np.sum(delta_hidden, axis=0) * learning_rate
```

El método backward_classification realiza el paso hacia atrás de la red neuronal en el caso de clasificación, lo que implica la actualización de los pesos y sesgos en función del error entre las etiquetas reales y las predicciones de salida. Toma los datos de entrada X, las etiquetas Y, así como las activaciones de la capa oculta hidden_layer y la salida de la red output_layer. Calcula el error entre las etiquetas reales y las predicciones de salida. Luego, calcula los gradientes de los pesos y sesgos de la capa de salida y de la capa oculta utilizando la función de activación de la capa de salida (derivada de la función de activación sigmoidal). Finalmente, actualiza los pesos y sesgos de ambas capas utilizando el gradiente descendente y la tasa de aprendizaje especificada. Este método es crucial para entrenar la red neuronal en problemas de clasificación, ya que ajusta los pesos y sesgos en función del error de clasificación.

Entrenamiento

```
def train(self, matrix, element_lists, Y, epochs, learning_rate):

    input_size = Y.shape[1]
    output_size = Y.shape[1]

    self.initWeights(input_size, self.hidden_size, output_size)

    element_lists_en = preprocess_element_lists(element_lists)
    element_lists_mw = preprocess_element_lists(element_lists)
    X_en = self.linealize_forward(matrix, element_lists_en, 1, atomic_electronegativity)
    X_mw = self.linealize_forward(X_en, element_lists_mw, 2, atomic_weight)

    # Entrenamiento de la redd neuronal
    for epoch in range(epochs):


        hidden_layer, output_layer = self.forward(X_mw)

        if self.kind == 'classification':
            self.backward_classification(X, Y, hidden_layer, output_layer, learning_rate)
        elif self.kind == 'regression':
            self.linealize_backward(X_mw, Y)
            self.backward_regression(X_mw, Y, hidden_layer, output_layer, learning_rate)

        # Cálculo del error en cada iteración
        error = np.mean(np.abs(Y - output_layer))
        print(f"Epoch {epoch+1}/{epochs}, Error: {error}")
    print('output_weights', self.output_weights)
    print('output_bias', self.output_bias)
    print('hidden_weights', self.hidden_weights)
    print('hidden_bias', self.hidden_bias)
```

- El método train se encarga de entrenar la red neuronal. Recibe como entrada la matriz de datos matrix, las listas de elementos element_lists, los valores objetivo Y, el número de épocas epochs y la tasa de aprendizaje learning_rate.
- Primero, inicializa los pesos de la red neuronal utilizando el método initWeights. Luego, realiza una serie de preprocesamientos sobre las listas de elementos para obtener dos conjuntos de características linealizados. Después, inicia el bucle de entrenamiento sobre el número de épocas especificado.
- Dentro del bucle de entrenamiento, calcula las salidas de la red neuronal utilizando el método forward. Dependiendo del tipo de red neuronal especificada

(classification o regression), realiza la retropropagación adecuada utilizando los métodos backward_classification o backward_regression, respectivamente.

- Finalmente, imprime el error medio absoluto en cada época para monitorear el progreso del entrenamiento y muestra los pesos y sesgos de las capas ocultas y de salida al final del entrenamiento.

Predicción

```
def predict(self, X):
    _, predictions = self.forward(X)
    # Predicción con la redd neuronal entrenada
    return predictions.flatten()
```

El método predict se encarga de realizar predicciones utilizando la red neuronal entrenada. Recibe como entrada un conjunto de datos de entrada X. Primero, utiliza el método forward para calcular las salidas de la red neuronal para los datos de entrada proporcionados. Luego, devuelve las predicciones aplanadas como un arreglo unidimensional.

io_molecules.py

Librerías

```
import numpy as np
import pandas as pd
```

En estas líneas se importa numpy para las operaciones matemáticas y pandas para interactuar con el documento en Excel

Dirección

```
molecules_folder = "src/Estructura/"
```

Esta línea establece la ubicación dentro del computador donde se encuentran los 435 archivos .mol que contienen la información de las matrices de conectividad y distancia.

Diccionarios

```
atomic_number = {
  "H": 1,
  "He": 2,
  "B": 5,
  "C": 6,
  "N": 7,
  "O": 8,
  "F": 9,
  "Ne": 10,
  "Si": 14,
  "P": 15,
  "S": 16,
  "Cl": 17,
  "Ar": 18,
  "Br": 35,
  "Kr": 36,
  "I": 53,
  "Xe": 54
}

atomic_weight = {
  "C": 12.011,
  "H": 1.00784,
  "O": 15.999,
  "Cl": 35.453,
  "F": 18.998403,
  "N": 14.0067,
  "Ar": 39.948,
  "B": 10.811,
  "Br": 79.904,
  "S": 32.065,
  "He": 4.002602,
  "I": 126.90447,
  "Kr": 83.798,
  "Ne": 20.1797,
  "P": 30.973762,
  "Si": 28.0855,
  "Xe": 131.293
}

atomic_electronegativity = {
  "C": 2.55,
  "H": 2.2,
  "O": 3.44,
  "Cl": 3.16,
  "F": 3.98,
  "N": 3.04,
  "Ar": 0,
  "B": 2.04,
  "Br": 2.96,
  "S": 2.58,
  "He": 0,
  "I": 2.66,
  "Kr": 3,
  "Ne": 0,
  "P": 2.19,
  "Si": 1.9,
  "Xe": 1.9
}
```

Estos diccionarios contienen información sobre propiedades de elementos químicos comunes. atomic_number contiene los números atómicos de varios elementos, atomic_weight contiene los pesos atómicos, y atomic_electronegativity contiene valores de electronegatividad.

Matriz de Distancia

```
def get_distances_matrix(molecule_name):
  path_file = molecules_folder + molecule_name + ".mol"
  file = open(path_file, "r")
  text = file.readlines()
  file.close()

  # Lee los elementos y las posiciones que hay en la molécula
  elements = []
  elements_position = []
  for line in text:
    if len(line.split()) == 10:
      elements.append(line.split()[3])
      elements_position.append([float(line.split()[0]),
        float(line.split()[1]), float(line.split()[2])])

  # Crea una matriz de 0s, y luego coloca las distancias
  # entre todos los átomos.
  distances_matrix = np.zeros((len(elements), len(elements)))
  for row in range(len(elements)):
    for column in range(row, len(elements)):
      element_1_position = elements_position[row-1]
      element_2_position = elements_position[column-1]

      distance = (element_1_position[0]-element_2_position[0])**2 + \
              (element_1_position[1]-element_2_position[1])**2 + \
              (element_1_position[2]-element_2_position[2])**2

      distances_matrix[row-1][column-1] = distance
      distances_matrix[column-1][row-1] = distance

  return distances_matrix, elements
```

Este fragmento de código busca calcular una matriz de distancias entre los átomos de una molécula utilizando las coordenadas espaciales de los átomos especificadas en un archivo de formato .mol.

- **Lectura del Archivo:** La función toma el nombre de una molécula como entrada y utiliza esa información para construir la ruta del archivo ".mol" correspondiente.
- **Extracción de Datos:** Luego, la función abre el archivo, lee su contenido línea por línea y almacena las líneas relevantes que contienen información sobre los elementos de la molécula, incluyendo las posiciones de cada átomo en el espacio tridimensional.
- **Procesamiento de Datos:** La función analiza estas líneas para extraer los elementos presentes en la molécula y las coordenadas de cada átomo. Estas coordenadas se almacenan en una lista de listas llamada elements_position, donde cada sublista contiene las coordenadas (x, y, z) de un átomo.
- **Cálculo de Distancias:** Con las coordenadas de los átomos, la función calcula las distancias entre cada par de átomos utilizando la fórmula de distancia euclidiana en tres dimensiones. Estas distancias se almacenan en una matriz cuadrada llamada distances_matrix, donde cada entrada representa la distancia entre dos átomos.
- **Retorno de Datos:** Finalmente, la función devuelve la matriz de distancias junto con la lista de elementos presentes en la molécula.

Matriz de Conectividad

```
def get_conectivity_matrix(molecule_name):
  path_file = molecules_folder + molecule_name + ".mol"
  file = open(path_file, "r")
  text = file.readlines()
  file.close()

  elements = []
  line_text = 4
  for line in text:
    if len(line.split()) == 10:
      elements.append(line.split()[3])
      line_text += 1
  conectitivity_matrix = np.zeros((len(elements), len(elements)))
  for line in text[line_text:-1]:
    element_1 = int(line.split()[0])
    element_2 = int(line.split()[1])
    conectitivity_matrix[element_1-1][element_2-1] = 1
    conectitivity_matrix[element_2-1][element_1-1] = 1

  return conectitivity_matrix, elements
```

Este fragmento de código está destinado a extraer la matriz de conectividad de una molécula a partir de un archivo .mol.

- **Lectura del Archivo:** La función toma el nombre de una molécula como entrada y utiliza esa información para construir la ruta del archivo ".mol" correspondiente.
- **Extracción de Datos:** Luego, la función abre el archivo, lee su contenido línea por línea y almacena las líneas relevantes que contienen información sobre los elementos de la molécula.
- **Procesamiento de Datos:** La función analiza estas líneas para extraer los elementos presentes en la molécula y construye una lista de estos elementos.
- **Construcción de la Matriz de Conectividad:** Después de identificar los elementos, la función crea una matriz de ceros del tamaño adecuado

para representar la matriz de conectividad entre los elementos. Luego, recorre las líneas del archivo ".mol" que contienen información sobre la conectividad entre los átomos y actualiza la matriz de conectividad con los valores correspondientes

- **Retorno de Datos:** Finalmente, la función devuelve la matriz de conectividad junto con la lista de elementos presentes en la molécula.

Vector de Propiedades

```
def get_elements_properties_single(element_list, property):
    return np.array([property[element] for element in element_list])
```

Esta función sirve para obtener las propiedades de los elementos individuales de una lista de elementos dada.

Propiedades Críticas

```
def get_data_substances(critical_property=None, nombres=None):
  file = 'src/Thermo_Data.xlsx'
  db = pd.read_excel(file, skiprows=0)

  if nombres is None and critical_property is not None:
     names = db[db['CONECTIVITY MATRIX'] == 'X'][db['ORGANICS'] == 'X']['COMPONENT']
     property_type = db[db['COMPONENT'].isin(names)][critical_property]
     names = np.array(names)
     property_type = np.array(property_type)
     return names, property_type
  else:
     # Retornar el número de carbonos y grupos funcionales para cada sustancia en la lista de nombres
     num_carbons = []
     functional_groups = []

     for nombre in nombres:
       carbon_count = db[db['COMPONENT'] == nombre]['CARBON NUMBER'].values[0]
       functional_group = db[db['COMPONENT']== nombre]['FUNCTIONAL GROUP'].values[0]
       num_carbons.append(carbon_count)
       functional_groups.append(functional_group)
     num_carbons = np.array(num_carbons)
     functional_groups = np.array(functional_groups)
     return num_carbons, functional_groups
```

PROEsta función get_data_substances obtiene los datos específicos de sustancias a partir de un archivo de Excel.

- **Carga de Datos:** La función carga un archivo de Excel llamado "Thermo_Data.xlsx" utilizando la biblioteca pandas.
- **Selección de Datos:** Dependiendo de los argumentos proporcionados, la función selecciona y devuelve diferentes conjuntos de datos:
  - Si se proporciona la propiedad crítica (por ejemplo, temperatura crítica), la función busca en la base de datos las sustancias que tienen información sobre esa propiedad y devuelve los nombres de las sustancias junto con los valores de esa propiedad.
  - Si se proporcionan nombres de sustancias, la función busca en la base de datos el número de carbonos y los grupos funcionales correspondientes a esas sustancias y los devuelve.
- **Manipulación de Datos:** La función utiliza bucles y consultas a la base de datos para extraer los datos específicos requeridos y los almacena en listas.
- **Retorno de Datos:** Los datos extraídos se convierten en matrices numpy y se devuelven como resultado de la función.

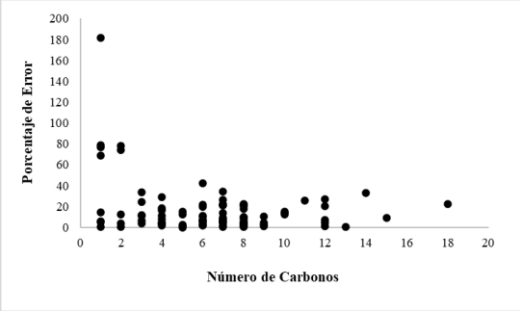
Resultados y Discusión

Sustancias Orgánicas

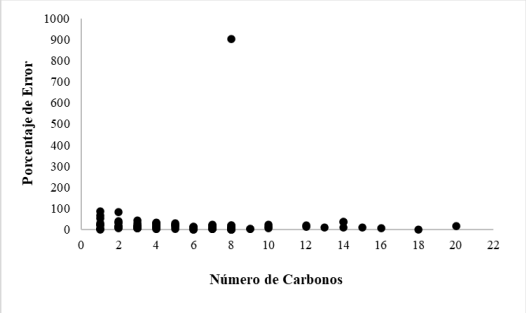
La Gráfica 1 presenta el porcentaje de error en la estimación de la temperatura crítica de las sustancias en función del número de átomos de carbono en las moléculas. Estos datos se obtuvieron utilizando la red neuronal previamente descrita, entrenada con las matrices de conectividad. La matriz de conectividad proporciona información sobre la conectividad local de los átomos en una molécula, incluyendo el número y tipo de enlaces que cada átomo forma con otros átomos. Además, la polaridad de una molécula está influenciada por la distribución de la carga eléctrica, la cual, a su vez, depende de la conectividad local de los átomos (Ottow & Weinmann, 2008).

La ANN demostró habilidad para aprender las relaciones complejas entre la conectividad local de los átomos y las propiedades fisicoquímicas de las moléculas, lo cual sugiere que la red neuronal capturó de manera efectiva la información relevante de la matriz de conectividad. La tendencia general de disminución del porcentaje de error con el aumento del número de átomos de carbono puede explicarse por el hecho de que las moléculas con un mayor número de átomos de carbono tienen una mayor conectividad local, es decir, cada átomo está conectado a más átomos vecinos (Ottow & Weinmann, 2008). Esta mayor conectividad local proporciona a la red más información para aprender relaciones complejas entre la estructura molecular y las propiedades fisicoquímicas.

Es importante destacar que las moléculas con un mayor número de átomos de carbono pueden exhibir una mayor diversidad estructural, lo que permite a la red neuronal aprenda una gama más amplia de relaciones entre la estructura y las propiedades.



Gráfica 1: Porcentaje de Error en Función del número de Carbonos, A partir de la Matriz de Conectividad



Gráfica 2: Porcentaje de Error en Función del número de Carbonos, A partir de la Matriz de Distancia

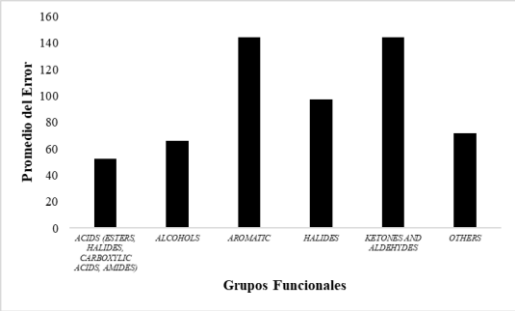
La Gráfica 2 también muestra el porcentaje de error en función del número de átomos de carbono. Los datos se obtuvieron utilizando la ANN con la matriz de distancia como datos de entrada. Este tipo de matriz proporciona información sobre la distancia entre pares de átomos en una molécula, lo que puede ser útil para identificar interacciones moleculares, como enlaces de hidrógeno o fuerzas de Van der Waals (Bonilla & Herrera, 2006). Estas interacciones pueden influir en propiedades como el punto de ebullición. La red neuronal pudo retroalimentarse con la información relevante de la matriz de distancia. Sin embargo, se observa que el porcentaje de error en la Gráfica 2 es generalmente mayor que en la Gráfica 1. Esto sugiere que la red neuronal se ajusta mejor al comportamiento de la conectividad de las moléculas en comparación con la distancia entre átomos.

Las Gráficas 3 y 4 proporcionan información sobre el rendimiento de la red neuronal utilizando la matriz de conectividad y la matriz de distancia como datos de entrada.

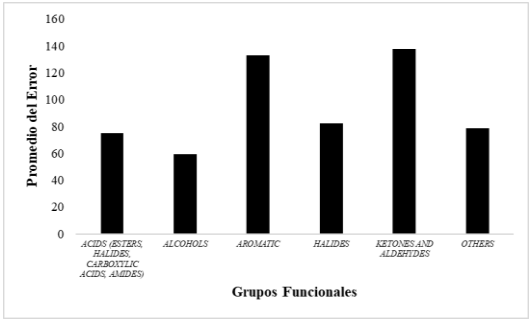
En la matriz de conectividad, se observan los siguientes resultados:

Acids (Esters, Halides, Carboxylic Acids, Amides): Se registra el menor error promedio para este grupo funcional, atribuido a la presencia de enlaces

característicos y grupos funcionales bien definidos, como el grupo carbonilo (-C=O) en cetonas y aldehídos, o el grupo carboxilo (-COOH) en ácidos carboxílicos (McMurry, 2016). Estos grupos funcionales proporcionan información química clara y distintiva que la ANN aprende de manera efectiva.



Gráfica 3: Porcentaje de Error en Función de los Grupos Funcionales, A partir de la Matriz de Conectividad



Gráfica 4: Porcentaje de Error en Función de los Grupos Funcionales, A partir de la Matriz de Distancia

Alcohols: Se observa un error promedio moderado, relacionado con la presencia del grupo hidroxilo (-OH), que introduce polaridad significativa en las moléculas. Aunque la ANN captura algunas relaciones relacionadas con la polaridad, la complejidad estructural de las moléculas, como la presencia de diferentes cadenas carbonadas y grupos sustituyentes, puede dificultar la extracción precisa de información.

Aromatic: Presenta el error promedio más alto, asociado con la complejidad estructural y la diversidad de los compuestos aromáticos. La estructura cíclica y la presencia de anillos de benceno pueden introducir desafíos para la ANN al aprender relaciones precisas entre la estructura y las propiedades.

Halides: El error promedio es similar al de los alcoholes, posiblemente debido a la presencia de átomos de halógeno (F, Cl, Br, I) en las moléculas (McMurry, 2016).

Estos átomos tienen características electrónicas distintivas, pero la ANN puede enfrentar dificultades para aprender estas relaciones debido a cambios estructurales y aumentos en la desviación estándar.

Ketones and Aldehydes: El error promedio es similar al de los compuestos aromáticos debido a la complejidad estructural y la presencia del grupo carbonilo (-C=O) en diferentes posiciones de la cadena carbonada (McMurry, 2016).

Others: El error promedio para este grupo depende de las características estructurales específicas de cada molécula. La ANN muestra buen rendimiento para moléculas con estructuras simples y grupos funcionales bien definidos, pero puede tener un rendimiento inferior para moléculas complejas con estructuras diversas.

Por otro lado, en la matriz de distancia se observan los siguientes resultados:

Acids (Esters, Halides, Carboxylic Acids, Amides): Se registra un error promedio mayor en comparación con la matriz de conectividad, indicando que la información de distancia entre pares de átomos no es tan específica para la estimación de propiedades fisicoquímicas en este grupo funcional. La información de distancia no captura adecuadamente las características estructurales como la conectividad local y la disposición espacial de los grupos funcionales.

Alcohols: La matriz de distancia captura algunas relaciones útiles para los alcoholes, posiblemente relacionadas con la disposición espacial del grupo hidroxilo (-OH) en la molécula (McMurry, 2016). Sin embargo, la información incompleta de la matriz de distancia limita la precisión de las estimaciones.

Aromatic: No se observa una mejora significativa en el error al cambiar a la matriz de distancia, debido a la falta de información sobre la distancia entre pares de átomos y los cambios estructurales.

Halides, Others, Ketones and Aldehydes: El error promedio para estos grupos es similar al de la matriz de conectividad, lo que sugiere que la matriz de distancia no tiene un impacto significativo en el rendimiento para este grupo funcional. La presencia de átomos de halógeno y otros grupos funcionales no se refleja adecuadamente en la información de distancia entre pares de átomos.

Sustancias Inorgánicas

En la Tabla 1 se presentan los resultados del porcentaje de error en la predicción química de las sustancias utilizando la matriz de conectividad como dato de entrada:

Sustancia	% Error
HYDROGEN FLUORIDE	16,16922301
NEON	768,2448651
NITROGEN TRIFLUORIDE	71,08459969
BORON TRICHLORIDE	10,70857255
PHTHALIC ANHYDRIDE	19,88079611
SILICON TETRAFLUORIDE	58,1008499
BROMINE	33,88487444
FLUORINE	167,2877697
HYDROGEN BROMIDE	7,031838049
CARBON MONOXIDE	1075,690945

Tabla 1: Predicción de las Sustancias Inorgánicas, A partir de la Matriz de Conectividad

Moléculas con Conectividad Clara y Átomos Centrales Bien Definidos

- *HIDROGEN FLUORIDE (HF) y HIDROGEN BROMIDE (HBr):* El error es inferior al 30% para estas moléculas diatómicas debido a la simplicidad de su estructura, con un enlace covalente claro entre el átomo central (H) y el átomo de halógeno (F o Br) (Housecroft & Sharpe, 2006). La matriz de conectividad captura adecuadamente esta conectividad simple, permitiendo que la ANN aprenda las relaciones precisas de estas sustancias.
- *BORON TRICHLORIDE (BCl_3):* El error no supera el 11% debido a la disposición espacial trigonal planar que tiene el compuesto con tres átomos de cloro alrededor del átomo central de boro. La matriz de conectividad refleja esta tridimensionalidad, proporcionando la información necesaria en cada etapa de la red neuronal.
- *NITROGEN TRIFLUORIDE (NF_3):* El error moderado (aproximadamente 71%) puede atribuirse a la presencia de enlaces covalentes polares entre el átomo central de nitrógeno y los tres átomos de flúor, lo que dificulta que la ANN capture la polaridad de los enlaces.

Elementos Monoatómicos y Gases Nobles

- *NEON (Ne) y FLUORINE (F):* El error experimental supera el 100% debido a que la matriz de conectividad no es adecuada para representarlos como elementos monoatómicos, ya que no tienen una conectividad molecular real (Housecroft & Sharpe, 2006).
- *HELIO-4 (He) y ARGON (Ar):* El error supera el 1000% para este compuesto diatómico heteronuclear debido a la complejidad de su enlace

químico. El enlace CO se considera una mezcla de enlace covalente y enlace triple, lo que dificulta que la matriz capture la naturaleza del enlace y su relación con las propiedades fisicoquímicas.

Moléculas Inorgánicas Complejas

- *CARBON MONOXIDE (CO):* El error supera el 1000% para este compuesto diatómico hetero nuclear y esto puede ser por la complejidad de su enlace químico. En enlace CO se considera como una mezcla de enlace covalente y enlace triple, lo que dificulta que la matriz capture la naturaleza del enlace y su relación con las propiedades fisicoquímicas

En la Tabla 2 se presentan los resultados del porcentaje de error en la predicción química de las sustancias utilizando la matriz de distancia como dato de entrada:

Sustancia	% Error
DEUTERIUM OXIDE	42,82687342
SILICON TETRACHLORIDE	24,57524631
HYDRAZINE	43,43061324
SILICON TETRAFLUORIDE	199,5071174
BORON TRICHLORIDE	17,51789367
SULFUR TRIOXIDE	24,47596647
HELIUM-4	6992,387796
ARGON	144,0947789
HYDROGEN IODIDE	13,08412417
CARBON MONOXIDE	1019,101977

Tabla 2:Predicción de las Sustancias Inorgánicas, A partir de la Matriz de Distancias

Para el caso de las sustancias *SILICON TETRACHLORIDE ($SiCl_4$)* y *BORON TRICHLORIDE (BCl_3)*, se observan errores similares en ambas matrices, lo que indica que la distancia interatómica puede ser suficiente para capturar información en moléculas con simetría regular.

El *CARBON MONOXIDE (CO)* presenta porcentajes de error casi idénticos a los de la matriz de conectividad, confirmando que la complejidad de su enlace no puede relacionarse adecuadamente con ningún tipo de matriz.

Con los resultados de las tablas, se observa que la matriz de distancia no proporciona una descripción óptima para las moléculas diatómicas, ya que la distancia interatómica es fija y no proporciona suficiente información para la estimación de propiedades fisicoquímicas.

Conclusiones

La ANN desarrollada demostró una capacidad significativa para aprender las relaciones que existen entre la estructura molecular y las propiedades fisicoquímicas, evidenciando que la red puede predecir con precisión el comportamiento de las sustancias orgánicas e inorgánicas, incluso en presencia de estructuras moleculares complejas y diversos grupos funcionales.

Tanto la matriz de conectividad como la matriz de distancia proporcionaron información útil para la red, sin embargo, los resultados fueron variables dependiendo de las sustancias y sus características estructurales. La elección del tipo de datos de entrada puede influir en el rendimiento de la red, por lo que es necesario investigar las características específicas de las moléculas empleadas.

Se observaron diferencias significativas en el rendimiento de la red neuronal supervisada entre las sustancias orgánicas e inorgánicas. Mientras que para algunas sustancias orgánicas la predicción fue precisa, para las sustancias inorgánicas, especialmente en elementos monoatómicos y los gases nobles, los resultados fueron menos satisfactorios. Esto sugiere que la ANN tiene limitaciones en la predicción para ciertos tipos de sustancias donde su comportamiento no tiene similitudes con otros.

La complejidad estructural de las moléculas influyó en el rendimiento de la red. Moléculas con estructuras simples y grupos funcionales definidos obtuvieron datos más precisos en comparación con moléculas con estructuras complejas y diversidad de grupos funcionales. Esto resalta la importancia de considerar la complejidad estructural en los modelos de predicción.

Aunque la red neuronal mostró un rendimiento aceptable, aún existen áreas de mejora. La predicción precisa de propiedades fisicoquímicas, como la temperatura crítica en las pruebas

experimentales, sigue siendo un desafío debido a la complejidad de las interacciones moleculares y la variabilidad de las estructuras químicas. Además, la selección adecuada de características y la optimización de los hiperparámetros son aspectos críticos que influyen en el rendimiento del modelo y requieren análisis constante en investigaciones futuras.

Recomendaciones

Realizar un análisis exhaustivo de los datos empleados, incluyendo aspectos como la calidad de los datos, la distribución de las propiedades fisicoquímicas y la diversidad de las moléculas en el conjunto de datos. Esto puede brindar apoyo al identificar posibles sesgos o áreas de mejora en la recopilación de datos.

Para mejorar la estimación de las propiedades de sustancias inorgánicas complejas o elementos monoatómicos, puede explorarse otro tipo de descriptores moleculares que capturen mejor la estructura parcial y la naturaleza de los enlaces, como la teoría del funcional de la densidad (DFT) o métodos basados en cargas parciales.

Probar y ajustar diferentes arquitecturas para la red neuronal, incluyendo variaciones en las capas ocultas, funciones de activación y otros hiperparámetros que mejoren el rendimiento del modelo. Esto podría implicar el uso de arquitecturas más complejas o técnicas de regularización para reducir el sobreajuste de los datos.

Considerar técnicas para aumentar la base de datos con el fin de incrementar la diversidad del conjunto de datos y mejorar la capacidad del modelo para generalizar las muestras. Esto podría incluir la generación de nuevas muestras sintéticas o la aplicación de transformaciones de datos para introducir variabilidad adicional en el conjunto de entrenamiento.

Referencias

Ahirwar, H., Kurmi, G., Khan, R., Khare, B., Jain, A., Jain, P. K., & Thakur, B. S. (2022). Review on QSAR using Anticancer Drug. *Asian Journal of Dental and Health Sciences*, *2*(4), 59–63. https://doi.org/10.22270/ajdhs.v2i4.27

Alander, J. T. (n.d.). On optimal population size of genetic algorithms. *CompEuro 1992 Proceedings Computer Systems and Software Engineering*, 65–70. https://doi.org/10.1109/CMPEUR.1992.218485

Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: Fundamentals, computing, design, and application. *Journal of Microbiological Methods*, *43*(1), 3–31. https://doi.org/10.1016/S0167-7012(00)00201-3

Bettina, M. (2011). *ESTRATEGIA DE PARAMETRIZACIÓN DEL MODELO A-UNIFAC*. https://repositoriodigital.uns.edu.ar/bitstream/handle/123456789/2295/GARRIGA-Tesis.pdf?sequence=1&isAllowed=y

Bonilla, B., & Herrera, J. N. (2006). *Revisando la ecuación de van der Waals*.

Forero-Corba, W., & Bennasar, F. N. (2024). Techniques and applications of Machine Learning and Artificial Intelligence in education: a systematic review. *RIED-Revista Iberoamericana de Educacion a Distancia*, *27*(1), 209–253. https://doi.org/10.5944/ried.27.1.37491

García Díaz, E. E., & Lozano Martínez, F. (2006). Boosting Support Vector Machines. *Revista de Ingeniería*, *24*, 62–70. https://doi.org/10.16924/revinge.24.8

Godoy Viera, Á. F. (2017). Técnicas de aprendizaje de máquina utilizadas para la minería de texto. *Investigación Bibliotecológica. Archivonomía, Bibliotecología e Información*, *31*(71), 103. https://doi.org/10.22201/iibi.0187358xp.2017.71.57812

Housecroft, C. E., & Sharpe, A. G. (2006). *Química inorgánica.* Pearson Educacion.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., … Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Li, S., Dan, Y., Li, X., Hu, T., Dong, R., Cao, Z., & Hu, J. (2020). Critical temperature prediction of superconductors based on atomic vectors and deep learning. *Symmetry*, *12*(2). https://doi.org/10.3390/sym12020262

López de Castilla Vásquez, C. (2019). Redes bayesianas con algoritmos basados en restricciones, scores e híbridos aplicados al problema de clasificación. *Anales Científicos*, *80*(1), 15. https://doi.org/10.21704/ac.v80i1.1370

McMurry, J. (2016). Organic Chemistry. *Cengage Learning* . https://dl.iranchembook.ir/ebook/organic-chemistry-419.pdf

Naranjo, V. M., Zagal, J. C., Roger, M., & Plaza, B. (2013). *ENTRENAMIENTO DE UNA RED NEURONAL PARA LA DETECCIÓN DE DAÑOS EN UNA VIGA USANDO FRECUENCIAS DE ANTI-RESONANCIA CRUZADAS*.

Ottow, E. (Eckhard), & Weinmann, Hilmar. (2008). *Molecular Descriptors for Chemoinformatics*. Wiley-VCH.

Prausnitz, J., Lichtenthaler, R., & Gomes de Azevedo, E. (2000). Termodinámica Molecular en Equilibrio de Fases. *Isabel Capella*.

Rayner Canham, G. (2000). *Quimica Inorganica Descriptiva, 2da Edición*. www.FreeLibros.me

Scalia, G., Grambow, C. A., Pernici, B., Li, Y. P., & Green, W. H. (2020). Evaluating Scalable Uncertainty Estimation Methods for Deep Learning-Based Molecular Property Prediction. *Journal of Chemical Information and Modeling*, *60*(6), 2697–2717. https://doi.org/10.1021/acs.jcim.9b00975

Stack, P. (2021). Métodos de Aprendizaje Supervisado y no Supervisado para la Estimación de Microestructura Cerebral en Datos de DWMR. *Centro de Investigación En Matemáticas, A.C*. https://cimat.repositorioinstitucional.mx/jspui/bitstream/1008/1129/1/TE%20835.pdf

Venkatasubramanian, V. (2019). The promise of artificial intelligence in chemical engineering: Is it here, finally? *AIChE Journal*, *65*(2), 466–478. https://doi.org/10.1002/aic.16489

Zavala, V. M. (2023). Outlook: How i Learned to Love Machine Learning (A Personal Perspective on Machine Learning in Process Systems Engineering). In *Industrial and Engineering Chemistry Research* (Vol. 62, Issue 23, pp. 8995–9005). American Chemical Society. https://doi.org/10.1021/acs.iecr.3c01565

Zhang, L., Han, J., Wang, H., Car, R., & Weinan, E. (2018). Deep Potential Molecular Dynamics: A Scalable Model with the Accuracy of Quantum Mechanics. *Physical Review Letters*, *120*(14). https://doi.org/10.1103/PhysRevLett.120.143001