UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Cuajimalpa

Aplicaciones del aprendizaje de máquina en la fisicoquímica

Idónea Comunicación de Resultados (ICR)
que

Presenta: **LDMNQ León Francisco Alday Toledo**
Maestría en Ciencias Naturales e Ingeniería
Universidad Autónoma Metropolitana Unidad Cuajimalpa

Director: Dr. Roberto Bernal Jaquez
Departamento de Matemáticas Aplicadas y Sistemas
Universidad Autónoma Metropolitana Unidad Cuajimalpa

Asesor: Dr. Saúl Zapotecas Martínez
Coordinación de Ciencias Computacionales
Instituto Nacional de Astrofísica, Óptica y Electrónica

POSGRADO EN CIENCIAS NATURALES E INGENIERÍA
ESPECIALIZACIÓN MAESTRÍA DOCTORADO

Resumen

El aprendizaje de máquina (*machine learning*, ML) se ha establecido como un campo académico y tecnológico sumamente fértil, y se ha intersectado con múltiples áreas. Una de nuestro particular interés es la fisicoquímica teórica, la cual modela el comportamiento y propiedades de átomos y moléculas a partir del entendimiento humano contemporáneo acerca de la estructura de la materia.

En esta Idónea Comunicación de Resultados (ICR) se ha explorado esta intersección de áreas. A fin de lograr este objetivo, hemos escrito nuestro propio programa que permite la exploración sistemática del aprendizaje de máquina químico, y lo hemos empleado para realizar experimentos de clasificación mediante redes neuronales para la predicción de propiedades termodinámicas de moléculas orgánicas.

También se exploran las bases de datos populares en el área, los optimizadores de gradiente que guían el entrenamiento de los modelos, las funciones de activación que transfieren información a través de un modelo, las representaciones que transforman las moléculas en vectores numéricos, así volviéndolas comprensibles para los modelos ML, los métodos de inicialización que permiten al modelo tener un buen aprendizaje, entre otros aspectos. Se dará una explicación minuciosa acerca de los numerosos componentes de un experimento ML.

Nuestros experimentos se han enfocado hacia la elucidación del carácter de la relación entre diferentes representaciones moleculares y diferentes propiedades moleculares, que es en sí misma, una de las tareas fundamentales de la química. Este conocimiento tiene un significado fisicoquímico, y nos permite entender cuáles son las propiedades estructurales que influyen sobre estas propiedades. Se muestra también una metodología para identificar las representaciones moleculares que resuelven un problema predictivo, y su agregación informada en representaciones mixtas eficientes.

Prefacio

Esta ICR nos lleva a un recorrido minucioso en la intersección del aprendizaje de máquina y la fisicoquímica teórica. Por lo tanto, nuestro trabajo es inherentemente transdisciplinar, dado que emplea técnicas de química computacional, computación científica de alto rendimiento, matemáticas, y aprendizaje de máquina.

Hemos escrito esta ICR buscando crear un documento accesible, uno que no exija un conocimiento profundo sobre estos campos, como requisito para entender nuestras contribuciones.

La estructura de este trabajo es la siguiente:

- el capítulo 1 muestra aspectos fundamentales del ML y las redes neuronales,
- mostramos algunos de los ejemplos prominentes y accesibles de bases de datos moleculares en el capítulo 2,
- el capítulo 3 explica de manera detallada la importancia de los descriptores que codifican las estructuras moleculares para poder realizar predicciones a partir de un modelo ML, y también explora algunos descriptores intuitivos,
- retornamos a aspectos matemáticos y computacionales en el capítulo 4, donde se prepara nuestra base de datos para los experimentos clasificativos, y se revisan los optimizadores de gradiente que usaremos en los entrenamientos,
- en el capítulo 5 es muestran los archivos de entrada y salida de `Dannte`, el programa que se ha escrito para explorar la interfaz del aprendizaje de máquina y la química computacional. Este programa ha sido escrito en el lenguaje `julia`, a totalidad.
- el capítulo 6 muestra nuestros experimentos y las conclusiones y conocimientos que podemos extraer de ellos.

También contamos con dos apéndices,

- el apéndice A muestra cómo obtener las coordenadas cartesianas de una molécula a partir de su representación SMILES, un formato altamente compacto,
- en el apéndice B listamos una fracción de la base de datos GDB-11. Consideramos importante el mostrar su aspecto, dada su gran accesibilidad y su minuciosa elaboración.

Nuestro trabajo ha resultado en una publicación en `arXiv`, bajo el nombre "*Obtaining transferable chemical insight from solving machine-learning classification problems: Thermodynamical properties prediction, atomic composition as good as Coulomb matrix*" [1].

También asistimos a la XX Reunión Mexicana de Fisicoquímica Teórica, celebrada en Cuernavaca, Morelos, en el año 2022, donde participamos en tres carteles:

- Dannte, una implementación accesible del aprendizaje de máquina,
- Composición atómica y AdaBelief: la estequiometría implica propiedades termodinámicas,

- Predicción de las propiedades electrónicas de pigmentos usando redes neuronales.

Este documento fue, por supuesto, escrito en LaTeX, y hace uso liberal de sus artificios para facilitar la lectura. Por ejemplo:

- los hipervínculos a ecuaciones y figuras están escritos en este color, , y hacerles *click* transporta al lector a la página referida,
- los índices tienen hipervínculos a cada capítulo y sección, de color ,
- las referencias son citadas entre corchetes, y escritas de color . De nuevo, hacerles *click* transporta al lector a la referencia pertinente.
- la esquina superior derecha de las páginas muestra este símbolo: , el cual es un hipervínculo al índice.

Hay apenas cuatro elementos en esta ICR que no fueron creados en LaTeX:

- los dos logos visibles en la portada,
- las figuras A.1b y A.2b.

Índice general

Capítulo 1

Introducción y objetivos

Índice de capítulo

1.1 Aprendizaje de máquina y fisicoquímica teórica

Desde su concepción, la meta del aprendizaje de máquina (*machine learning*, ML) ha sido el construir modelos computacionales veloces que aprendan a relacionar algunas propiedades de una muestra (la cual es descrita de manera sencilla), con otras propiedades de complejidad mayor. Por ejemplo, podríamos contar con:

- una gran cantidad de números escritos a mano, y buscamos su clasificación automática [2-4],
- un texto que buscamos traducir a varios idiomas,
- una molécula, descrita por medio de coordenadas cartesianas, y buscamos predecir su energía.

Considerando los dos primeros casos, podría bastar con la intervención humana para completar la labor; para el tercero, existen métodos computacionales que pueden obtener la respuesta. Sin embargo, depender de estos métodos puede ser impracticable en el supuesto de que tengamos una cantidad de datos mayúscula. Es aquí donde los métodos ML adquieren atractivo: podemos delegarles ciertos procesos laboriosos, obteniendo una cantidad de resultados enorme con una calidad comparablemente buena, a un costo computacional o de horas-hombre bastante menor.

Estas técnicas se demostraron viables en teoría desde la década de 1970, pero fue hasta el advenimiento de las computadoras personales y el almacenamiento masivo que se volvieron aplicables en práctica, y se integraron de manera acelerada en múltiples disciplinas. Uno de los casos más sonados en la actualidad es el de ChatGPT, una herramienta del reino del lenguaje natural, capaz de responder a un mensaje escrito con un tono que imita la respuesta de un humano. ChatGPT se asemeja a un asistente personal: puede escribir un correo electrónico, código en distintos lenguajes de programación, una historia de ficción, entre otras tareas [5-7].

Minerva es otra herramienta ML sofisticada, que puede resolver problemas matemáticos verbales, los típicos de un libro de texto [5, 8]. Mientras que los modelos de lenguaje han sido inefectivos para los problemas de razonamiento cuantitativo, Minerva muestra un desempeño prometedor. Los autores exponen [8] que:

> *...we evaluated Minerva 62B on the National Math Exam in Poland and found that it achieves a score of* 57 %*, which happened to be the national average in 2021...The 540B model achieves* 65 %*.*
>
> *...we study the performance of Minerva 540B on simple arithmetic tasks. The model achieves over* 80 % *accuracy on 10-digit addition and over* 20 % *accuracy on 18-digit addition.*

Otro caso muy popular es el de los modelos que generan imágenes elaboradas a partir de descripciones textuales breves, como DALL-E 2 [9, 10], Craiyon [11, 12], y Midjourney [13, 14], los cuales se han vuelto populares dada su habilidad para bosquejar ideas, unir estilos, o ilustrar situaciones graciosas o absurdas.

El aprendizaje de máquina se ha aplicado a resolver otros problemas, como por ejemplo:

- la detección de correos *spam* [15, 16],
- el desarrollo de asistentes de inteligencia artificial para programación [17],
- la identificación de perros por raza [18-21].

El requisito primordial para aplicar el aprendizaje de máquina a una disciplina es que las muestras puedan ser representadas por medio de números. En el caso de la química, el objeto de estudio más usual son las moléculas, que pueden codificarse de varias maneras, por ejemplo las coordenadas cartesianas, la masa molecular, o la cantidad de enlaces, de ciclos, o de sitios donadores y aceptores de puente de hidrógeno. Por lo tanto, la química y el aprendizaje de máquina tienen numerosas oportunidades de traslape.

Dentro de las subdisciplinas de la química, destacamos el caso de la fisicoquímica teórica y la química computacional, cuyos estudios suelen buscar la predicción de alguna propiedad de un sistema atómico a partir de sus coordenadas, usando métodos de alto nivel de teoría, conocidos como *ab initio*, o métodos con menor nivel de teoría y costo computacional, que modelan un sistema usando consideraciones más simples.

Esta es la razón de nuestro particular interés en estas dos subdisciplinas químicas: ambas tienen objetivos comunes con las técnicas ML, incluso si difieren en cuanto al *cómo* realizar las predicciones. Esto es, la descripción inicial del sistema atómico y la búsqueda de una de sus propiedades, son perfectamente transferibles al marco de un estudio ML.

Un axioma común de estas dos subdisciplinas es que las propiedades moleculares están dadas por la estructura. Este resultado proviene de 1824, cuando Wöhler realizó el análisis elemental del cianato de plata, encontrando la fórmula AgCNO [22], mientras que Liebig y Gay-Lussac, independientemente, analizaron el fulminato de plata, encontrando la misma fórmula [23].

Actualmente sabemos que el cianato tiene la estructura $^{-}O{-}C{\equiv}N$, mientras que el fulminato es $^{-}C{\equiv}N^{+}{-}O^{-}$. Sin embargo, en la década de los 1820, estos dos hallazgos parecían contradictorios:

dos sales con múltiples diferencias físicas y químicas eran aparentemente la misma. Este y otros descubrimientos de la época [24] se esclarecieron cuando Berzelius definió el concepto de isomería en 1833, estableciendo así la dependencia de las propiedades de una molécula con su estructura, y no solo con los átomos que le componen [25].

Más adelante, en 1868, Brown y Fraser [26] intentaron entender detalladamente la forma en que la estructura de una molécula determina su actividad fisiológica. Para esto, estudiaron el efecto fisiológico de múltiples compuestos orgánicos. Consideramos que su explicación conceptualiza a los modelos estructura-actividad cuantitativos (*quantitative structure-activity relationship*, QSAR), mucho antes de la existencia de las computadoras y las bases de datos moleculares. Ellos exponen:

> *There can be no reasonable doubt that a relation exists between the physiological action of a substance and its chemical composition and constitution, understanding by the latter term the mutual relations of the atoms in the substance.*
>
> *...composition alone is quite insufficient to explain physiological action, and...constitution must also be taken into account in every attempt to connect the chemistry of substances with their action on the animal body.*
>
> *...[Our proposed method] consists in performing upon a substance a chemical operation which shall introduce a known change into its constitution, and then examining and comparing the physiological action of the substance before and after the change. We may express this in mathematical language thus:—Let C represent the constitution of the original substance and Φ its physiological action. After the operation, C becomes $C + \Delta C$ and Φ, $\Phi + \Delta\Phi$. Here ΔC, Φ, and $\Phi + \Delta\Phi$ are known, and by applying the method to a sufficient number of substances, and by varying ΔC, we might hope to determine what function Φ is of C...[although] we cannot obtain an accurate mathematical definition of f in the equation $\Phi = f(C)$, we may be able, in an approximate manner, to discover the nature of the relation.*

Gran parte de la *praxis* de la química cuántica moderna es también la obtención de propiedades físicas de un sistema a partir de su estructura, y para ello emplea a múltiples operadores asociados a distintas propiedades. De esta manera, el problema de la medición de una propiedad molecular queda reducido a resolver una eigenecuación de la forma $\widehat{O}\mathbf{\Psi} = \lambda\mathbf{\Psi}$.

Los problemas solucionables por la química cuántica, originalmente reducidos en número, se volvieron esencialmente infinitos con la introducción de las computadoras como herramientas de cálculo, dando origen a la química computacional. En la actualidad, el aprendizaje de máquina plantea una segunda revolución o expansión del horizonte y aplicabilidad de los métodos cuánticos. Eso lo podemos ver en la diversidad de estudios que usan tanto al aprendizaje de máquina como a la química computacional:

- el modelado de superficies de energía potencial [27-31],
- asistir el diseño molecular [32],
- predicción de mecanismos de reacción [33], o la interacción fármaco-receptor [34],
- la optimización genética de conjuntos de entrenamiento pequeños altamente efectivos [35],
- explorar el espacio SMILES* [36, 37] para encontrar moléculas con potencial farmacológico [38],
- predecir propiedades termodinámicas a partir de la representación SMILES de la molécula

*El lenguaje SMILES codifica estructuras orgánicas de manera legible y compacta usando caracteres ASCII. Damos una descripción elemental de sus reglas en la página 30 de la sección 2.2.

[39]. Notemos que este método de predicción no requiere de una costosa optimización de la geometría molecular.

La disciplina del aprendizaje de máquina incluye una gran cantidad de métodos, cada uno con un diferente fundamento teórico y especializado en resolver cierto tipo de problema. Por ejemplo, uno de los métodos más robustos para la clasificación de muestras son las máquinas de vectores de soporte (*support vector machine*, SVM) [40], las cuales buscan al hiperplano óptimo que separa a las muestras de diferentes clases.

En el campo de visión por computadora, las redes neuronales convolucionales (*convolutional neural network*, CNN) [20] se han establecido como una opción eficaz para el reconocimiento de objetos en imágenes. Uno de los ejemplos más populares es la sofisticada AlexNet [41], la cual emplea el cómputo GPU como un elemento crucial de su formulación. Esto hizo posible el dar pasos agigantados en términos de desempeño: AlexNet ganó la edición 2012 del concurso *ImageNet Large Scale Visual Recognition Challenge*, separándose notablemente del resto de participantes [41].

Debido a todo lo anterior, consideramos que el horizonte ML es sumamente amplio, y tan solo su intersección con la química presenta una gran cantidad de problemas interesantes. Buscando mantenernos en una región cercana a la química, hemos elegido un método predictivo que ha mostrado su valor en múltiples áreas, incluyendo la química: las redes neuronales (*neural networks*, NN, secciones 1.3 hasta 1.6), y las usamos para la clasificación de moléculas de acuerdo al valor de alguna de sus propiedades, según presentamos en las secciones 4.2 y 4.3. Finalmente, abordamos los resultados de nuestros experimentos en el capítulo 6.

En la siguiente sección explicamos la forma en que delimitamos nuestro proyecto.

1.2 Objetivos y elección de metas

Hay una variedad de formas en que podemos aplicar los métodos ML a distintos problemas químicos, y entonces definir y acotar un proyecto puede ser difícil. Por lo tanto, hemos decidido trabajar con una de las herramientas ML más robustas y estables: las redes neuronales, debido a su gran aplicabilidad y versatilidad, dado que pueden trabajar con muestras de cualquier tipo, y usarse con problemas discretos (como la clasificación de muestras) o continuos (como la predicción de valores numéricos).

Luego, un ángulo atractivo para formular un proyecto de investigación ML es el punto de vista computacional, en el cual nos separamos de las entradas y las salidas para enfocarnos en afinar la etapa intermedia: el modelo que les conecta. Entonces, podríamos buscar los componentes que logran un mejor desempeño: esto significaría experimentar con diferentes optimizadores y sus respectivos hiperparámetros, funciones de activación, arquitecturas, bases de datos, entre otros factores, para finalmente encontrar la configuración experimental que vence alguna marca de la literatura por algunos puntos porcentuales.

Sin embargo, consideramos que este enfoque es insatisfactorio, dado que su logro es simplemente una configuración ML de mejor desempeño: por ejemplo, podríamos haber encontrado que la predicción del momento dipolar de una molécula mejora al entrenar el modelo con el optimizador AdaBelief, mientras que la de la capacidad calorífica es mejor en los modelos con una cantidad impar de capas ocultas. Sin embargo, es probable que este conocimiento no sea interpretable, y no esté acompañado de un entendimiento más profundo sobre la relación entre las entradas y las salidas: no tendríamos explicación del *porqué* una configuración fue la más efectiva y no otra.

Por lo tanto, hemos preferido enfocarnos en las entradas y salidas del modelo, explorando el entorno de descriptores moleculares (representaciones moleculares) en búsqueda de mejorar la capacidad predictiva del modelo para cierta propiedad fisicoquímica. Para esto:

- *trabajaremos con las entradas*, buscando los descriptores que codifican las características

estructurales moleculares relevantes para cierta propiedad objetivo,

- *y con las salidas*,separando al espacio molecular en clases según el valor de la propiedad objetivo, y definiendo un modelo que predice la clase de una molécula en lugar de un valor numérico. De esta manera, en lugar de dar una predicción numérica del valor de una propiedad molecular, como por ejemplo $f(\text{molécula}) = \text{valor} \in \mathbb{R}$, nuestro modelo dirá el intervalo dentro del cual está el valor de la propiedad molecular, es decir, $f(\text{molécula}) \in (y_3, y_4]$, donde $y_3, y_4 \in \mathbb{R}$.

  Una consecuencia natural de este esquema es que podremos analizar el desempeño de cada clase por separado, así como la forma en la que responden al recibir más o menos información química a partir de distintas formas de representar (codificar) las moléculas.

Estas son las dos características principales de nuestro proyecto: el enfoque de clasificación y el experimentar con diferentes representaciones. Consideramos que su conjunción genera un conocimiento mucho más *interpretable*, transferible a experimentos futuros de ML-química, y puede incluso ayudarnos a elucidar el carácter de la relación entre algunas propiedades estructurales y moleculares.

Como explicaremos más adelante, dos de los descriptores más usuales son la matriz de Coulomb ordenada y sus eigenvalores (sección 3.6), las cuales son de cálculo sencillo y definición intuitiva. Por lo tanto, hemos elegido usar estas representaciones como propiedades de entrada de nuestros experimentos (capítulo 6). Luego, dado que hemos encontrado un buen desempeño de estos descriptores para la predicción de propiedades termodinámicas moleculares, las hemos elegido como las propiedades de salida de nuestros modelos.

1.3 Redes neuronales: aproximadores universales

Las redes neuronales son de las técnicas ML más habituales. Están incluidas en varias maquinarias públicas, como `Flux.jl` [42, 43], `scikit-learn` [44], `TensorFlow` [45], y `PyTorch` [46]. Son conocidas como *aproximadores universales*: Cybenko [47] define una función continua σ tal que

$$\sigma(z) = \begin{cases} 1 & \text{cuando } z \to +\infty \\ 0 & \text{cuando } z \to -\infty \end{cases} \tag{1.1}$$

entonces, σ es una función sigmoide. Luego, el teorema 2 de [47] explica que:

> Sea $C(I_n)$ un espacio finito $\subset \mathbb{R}^n$. Entonces, cualquier función sigmoide continua σ puede usarse en sumas finitas de la forma
>
> $$G(\boldsymbol{w}) = \sum_{j=1}^{N} \alpha_j \sigma(\boldsymbol{w} \cdot \boldsymbol{x}_j + \theta_j) \tag{1.2}$$
>
> donde $\alpha_j, \theta_j, \boldsymbol{x}_j$ están fijos, $\alpha_j, \theta_j \in \mathbb{R}$, $\boldsymbol{x}_j, \boldsymbol{w} \in \mathbb{R}^n$.
>
> Y, para cualquier $f \in C(I_n)$ y $\varepsilon > 0$, existe una suma $G(\boldsymbol{w})$ tal que
>
> $$|G(\boldsymbol{w}) - f(\boldsymbol{w})| < \varepsilon \quad \forall\, \boldsymbol{w} \in \text{Dom}(G) \tag{1.3}$$
>
> donde $\text{Dom}(G)$ es el dominio de G.

En otras palabras, las redes neuronales pueden aproximar cualquier función con precisión arbitraria, siempre y cuando no haya restricción en la cantidad de nodos de la red ni en la magnitud de sus pesos.

También, el trabajo de Hornik, Stinchcombe y White [48] también plantea que las redes neuronales de una capa oculta pueden aproximar cualquier función $\mathbb{R}^n \to \mathbb{R}$, donde n es entero:

> *The results given here are clearly only one step in a rigorous general investigation of the capabilities and properties of multilayer feedforward networks. Nevertheless, they provide an essential and previously unavailable theoretical foundation establishing that the successes realized to date by such networks in applications are not just flukes, but are instead a reflection of the general universal approximation capabilities of multilayer feedforward networks.*

Podemos ver que las NN son una herramienta capaz cuando se conocen las variables que influyen en una propiedad de interés, pero no existe una expresión analítica que las conecte. En este caso, podemos entrenar una NN para resolver el problema, lo cual implicará la modificación iterativa de sus parámetros, buscando que sus predicciones sean cada vez más parecidas al valor real.

Consideremos ahora el caso en que la expresión analítica existe, pero su costo computacional es demasiado. Esto sucede con las superficies de energía potencial, las cuales asocian una geometría de un sistema químico con su energía. Esta es un área sumamente fértil de la fisicoquímica teórica, la cual ha modelado estas superficies mediante diversas teorías, que se han implementado en múltiples programas de computadora que obtienen la energía de un sistema químico en tiempo finito. Sin embargo, algunos sistemas pueden ser demasiado grandes como para modelarlos a un nivel de teoría alto. Una solución a este problema es el modelado de superficies de energía potencial mediante técnicas ML [27-31], lo cual puede ser costeable y altamente efectivo.

1.4 Procesamiento de señales

Para entender la forma en que las redes neuronales toman la información de una muestra y dan una predicción, hemos de recurrir a un precursor de las NN, los *perceptrones*, que también son utilizados en casos de regresión numérica o clasificación de muestras.

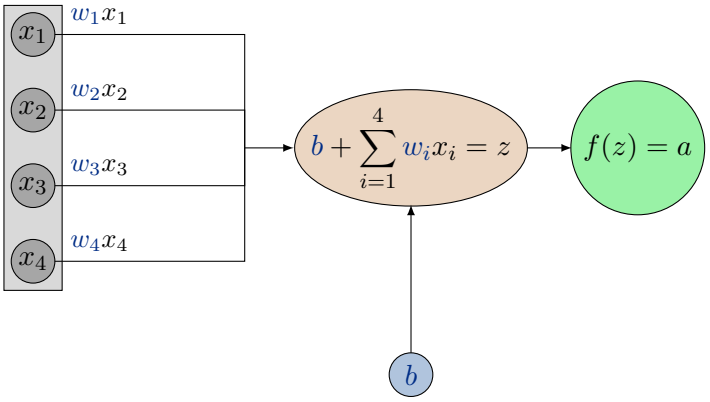
El funcionamiento de un perceptrón está dado por los pasos:

1. Las neuronas de la capa de entrada reciben las características (*features*) de una muestra: $(x_1, x_2, \ldots, x_n)$.
2. Calcular la señal z,
$$z = b + \sum_{i=1}^{n} w_i x_i \tag{1.4}$$
donde los pesos w_i y el incremento b (*bias*) son los parámetros propios del perceptrón, provenientes de haberlo entrenado con un conjunto diverso de muestras.
3. Calcular la propiedad de salida a,
$$f(z) = a \tag{1.5}$$
donde f es la función de activación (sección 1.5). Algunas referencias bibliográficas emplean la notación $\widehat{y}$ en lugar de a.

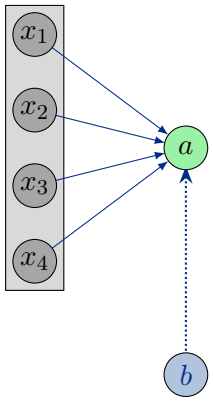
   Simbolizamos a la etiqueta de la muestra (el valor real de su propiedad en la base de datos) como y. En un perceptrón ideal tenemos $a = y$.

Este funcionamiento se ilustra en la figura 1.1 para una muestra con cuatro características.

Al conocer el valor de a sabemos el veredicto del perceptrón sobre la muestra analizada. Por ejemplo, si buscamos conocer si ésta pertenece o no a una clase, entonces es habitual elegir una f tal que $0 < f(z) < 1$. Cuando $f(z) \leq 0.5$ consideramos que la muestra no pertenece a la clase; para



(a) Diagrama explícito.



(b) El perceptrón de la figura 1.1a en representación abreviada. El estilo de flecha es diferente para los pesos que para los incrementos.

Figura 1.1: Dos representaciones de un perceptrón. Los elementos de la red de color ■ son los parámetros, ajustables durante el entrenamiento.

$f(z) > 0.5$, sí. Por otra parte, si buscamos predecir alguna propiedad continua de la muestra, es habitual elegir una f tal que $d < f(z) < \infty$, donde $d \in \mathbb{R}$, y $f(z)$ es la predicción.

Cuando conocemos el valor verdadero de la propiedad buscada, le denominamos y, y la diferencia $y - a$ es una medida del error del modelo para la muestra elegida. La métrica del error del modelo se denomina *costo*, simbolizado por J. Desarrollamos este punto en la sección 1.8.

Revisaremos detalladamente el funcionamiento de las redes neuronales en la sección 1.6. Para esto, primero hemos de ejemplificar las funciones de activación en la sección 1.5.

1.5 Funciones de activación

En la sección 1.3 hemos comentado el carácter de *aproximador universal* de las NN, de acuerdo a los desarrollos de Cybenko [47] y Hornik, Stinchcombe y White [48]. Ambos artículos trabajan con redes neuronales de una capa oculta y una *función de activación* no lineal, que denominamos f. En particular, Cybenko asume que f es sigmoide.

Posteriormente, Hornik [49] generalizó los requisitos que f debe cumplir para que una NN sea un aproximador universal: f debe ser continua, acotada, y no-constante. Otros trabajos de la época imponían sobre f requisitos adicionales como ser integrable, sigmoidal y monotónica, lo cual limitaba las f posibles para una NN.

En su trabajo, Hornik encuentra que el carácter de aproximador universal de una NN se debe a la arquitectura de la red (esto es, al paso de señales entre capas y la presencia de al menos una capa oculta con cierta cantidad de neuronas), y no a la función de activación elegida:

> *...we conclude that it is not the specific choice of the activation function, but rather the* multilayer feedforward architecture *itself which gives neural networks the potential of being universal learning machines.*

aunque Hornik advierte que:

However. it should be emphasized that our results do not mean that all activation functions ψ will perform *equally well in specific learning problems. In applications, additional issues as, for example, minimal redundancy or computational efficiency, have to be taken into account as well.*

donde ψ denota a una función de activación cualquiera.

Muchas de las funciones de activación usadas actualmente tienen dos cotas. Sin embargo, también han surgido otras f con una sola cota (semiacotadas), que también logran resultados meritorios. Hasta donde es de nuestro conocimiento, los modelos de clasificación usan una f de una o dos cotas, mientras que los modelos de regresión eligen f semiacotada.

En esta sección ejemplificaremos algunas funciones de activación comunes en el aprendizaje de máquina. Todas estas funciones tienen una sola variable y alguna cantidad de parámetros. Su variable z es la *señal* explicada en la sección 1.4.

- La más sencilla, *rectified linear unit*, ReLU, con rango $[0, \infty)$,

$$\begin{aligned} \text{ReLU}(z) &= \begin{cases} z & \text{si } z > 0 \\ 0 & \text{si } z \leq 0 \end{cases} \\ &= \max(0, z) \end{aligned} \tag{1.6}$$

- la función *continuously differentiable exponential linear unit*, CELU [50] es una variante de la *exponential linear unit* (ELU) que tiene un parámetro α, cuyo valor por defecto es 1. $\text{CELU}(z) \to \text{ReLU}(z)$ si $z \to \infty$, $\text{CELU}(z) \to -\alpha$ si $z \to -\infty$. Esta función es de derivada suave excepto tal vez en un punto,

$$\text{CELU}(z) = \max(0, z) + \min\left[0, \alpha\left[\exp\left(\frac{z}{\alpha}\right) - 1\right]\right] \tag{1.7}$$

- la función softplus es una función suave semejante a ReLU. Su derivada es continua.

$$\text{softplus}(z) = \ln[1 + \exp(z)] \tag{1.8}$$

- la función logística es, a diferencia de las tres anteriores, una función de tipo sigmoide: su derivada siempre es > 0, y está confinada entre dos reales. Muchas NN clasificativas tienen una f de tipo sigmoide.

$$\text{logística}(z) = \frac{1}{1 + \exp(-z)} \tag{1.9}$$

- la función tangente hiperbólica es un ejemplo más de una función sigmoide,

$$\begin{aligned} \tanh(z) &= \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \\ &= \frac{\exp(2z) - 1}{\exp(2z) + 1} \end{aligned} \tag{1.10}$$

Mostramos estas funciones en la figura 1.2.

1.6 Red neuronal simple y profunda

Hemos revisado las funciones de activación y el procesamiento de señales en un perceptrón, y estamos listos para integrar estos conceptos y abordar el funcionamiento de las redes neuronales sencillas y profundas.

Las diferencias principales entre las redes neuronales y los perceptrones son:

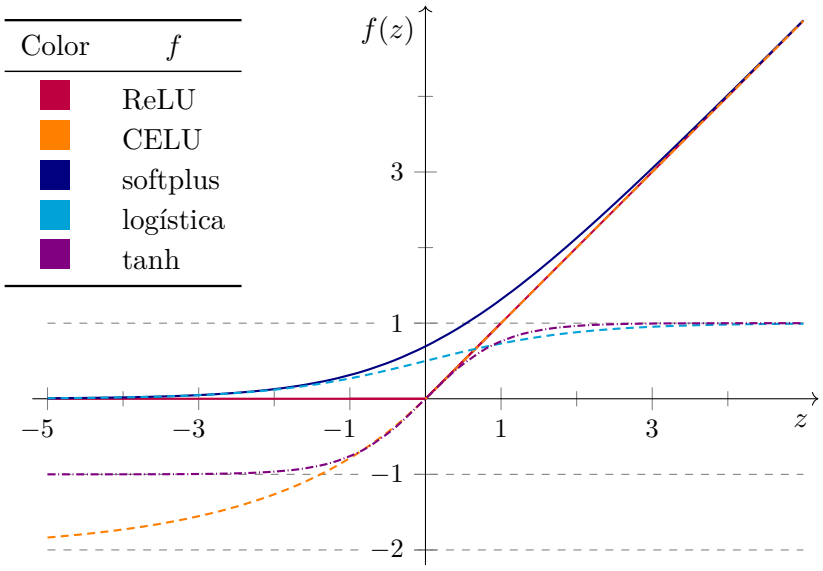


Figura 1.2: Cinco funciones de activación. Las líneas grises - - son rectas paralelas al eje z para indicar la convergencia de las funciones. A fin de distinguir mejor entre las funciones, elegimos que la función CELU tuviera $\alpha = 2$, en lugar de su valor habitual de 1.

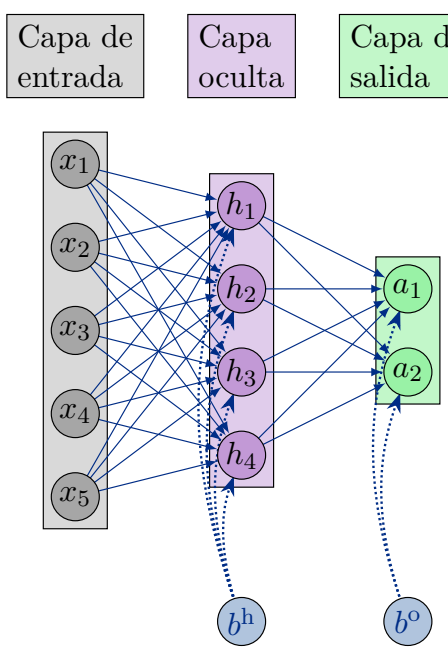
- las NN, en lugar de conectar directamente las entradas con la salida, tienen capas intermedias denominadas *capas ocultas*. Si hay más de una capa oculta, decimos que la red es *profunda*.
- una sola red puede predecir múltiples propiedades de una muestra, en lugar de solamente una.
- las capas ocultas confieren a la red la capacidad de generar 'filtros', los cuales mezclan algunas de las propiedades de entrada. Estas combinaciones de propiedades *concretas* generan descriptores artificiales *abstractos*, cada uno de los cuales correlaciona cierta constelación de entradas dependiendo de las características de las muestras de entrenamiento. Esta es la razón de que las redes profundas puedan abordar problemas de mayor complejidad.

Mostramos dos ejemplos de redes neuronales en la figura 1.3. El proceso de combinación lineal y aplicar la función de activación se representa de manera abreviada, al igual que en la figura 1.1b.

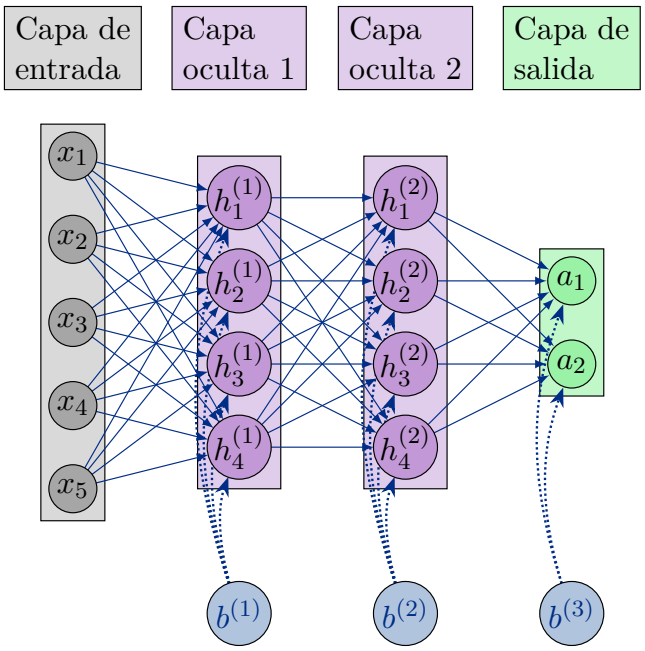
Como consecuencia de la aparición de las capas ocultas, cada una con múltiples nodos (es decir, neuronas), los pesos de las redes neuronales se agrupan en matrices en lugar de vectores; los incrementos se almacenan en vectores en lugar de escalares.

La notación de los componentes de las redes neuronales se vuelve más detallada:

- n, cantidad de características de entrada (*input*).
- c, cantidad de propiedades de salida (*output*).
- x_i, las características de una muestra. $i = 1, 2, ..., n$.
- ℓ, contador que itera sobre las capas ocultas. La *segunda* capa de una NN es la *primer* capa oculta, y la denotamos con $\ell = 1$.
- $z_j^{(\ell)}$, la señal que recibe la neurona j de la capa ℓ. Una vez que $z_j^{(\ell)}$ se opere con la función de activación, tendremos la señal de esta neurona, la cual participa en todas las $z_j^{(\ell+1)}$ de la capa siguiente.
- $b_j^{(\ell)}$, el incremento de la neurona j de la capa ℓ.



(a) Red neuronal con una capa oculta. b^{h} almacena los distintos incrementos de la capa oculta (*hidden*), b^{o} almacena los de la capa de salida (*output*).



(b) Ejemplo simple de una red neuronal profunda.

Figura 1.3: Diagramas abreviados de redes neuronales. En general, las redes neuronales pueden trabajar con miles de características de entrada, pocas decenas de capas ocultas con cientos de unidades en cada una, y decenas de propiedades de salida. También, las NN suelen tener un aspecto piramidal: las capas ocultas subsecuentes tienen cada vez menos neuronas. Los nodos de incrementos almacenan un b distinto para cada neurona.

- $w_{ji}^{(\ell)}$, el peso que conecta a la neurona (o entrada) i de la capa $\ell - 1$ con la neurona j de la capa ℓ.
- $h_j^{(\ell)}$, la neurona j de la capa oculta ℓ.
- a_k, salidas de la red. Son las neuronas de la última capa, que contienen las predicciones de las propiedades de la muestra según la NN. $k = 1, 2, ..., c$.
- y_k, los valores verdaderos de las propiedades de la muestra. Un modelo ideal arroja $a_k = y_k$.

Para aclarar el procesamiento de señales en las NN y el uso de su notación, veremos un ejemplo simbólico siguiendo la figura 1.3b:

- Las neuronas de la primera capa oculta, $h_j^{(1)}$, reciben las señales de su incremento respectivo, $b_j^{(1)}$, y de cada una de las entradas, x_i. Entonces, escribimos el cálculo de las señales $z_j^{(1)}$ de esta forma:

$$z_j^{(1)} = b_j^{(1)} + \sum_{i=1}^{5} w_{ji}^{(1)} x_i \tag{1.11}$$

donde $b_j^{(1)}$ es distinto para cada neurona oculta. En particular, para la primera,

$$z_1^{(1)} = b_1^{(1)} + w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + w_{14}^{(1)} x_4 + w_{15}^{(1)} x_5 \tag{1.11b}$$

donde se marca con color ■ a los índices de la neurona que recibe esta señal.

- Los valores de la primer capa oculta se calculan con f:

$$h_j^{(1)} = f\left(z_j^{(1)}\right) \tag{1.12}$$

En particular, para la primer neurona,

$$h_1^{(1)} = f\left(z_1^{(1)}\right) \tag{1.12b}$$

- Luego, las neuronas de la segunda capa oculta reciben las señales de las cuatro neuronas antecesoras.

$$z_j^{(2)} = b_j^{(2)} + \sum_{i=1}^{4} w_{ji}^{(2)} h_i^{(1)} \tag{1.13}$$

En particular, para la primer neurona,

$$z_1^{(2)} = b_1^{(2)} + w_{11}^{(2)} h_1^{(1)} + w_{12}^{(2)} h_2^{(1)} + w_{13}^{(2)} h_3^{(1)} + w_{14}^{(2)} h_4^{(1)} \tag{1.13b}$$

- Se calculan los valores de la segunda capa oculta,

$$h_j^{(2)} = f\left(z_j^{(2)}\right) \tag{1.14}$$

En particular, para la primer neurona,

$$h_1^{(2)} = f\left(z_1^{(2)}\right) \tag{1.14b}$$

- Por fin, se computan las señales que llegan a las neuronas de la capa de salida.

$$a_1 = f\left(b_1^{(3)} + \sum_{i=1}^{4} w_{1i}^{(3)} h_i^{(2)}\right) \tag{1.15}$$

$$a_2 = f\left(b_2^{(3)} + \sum_{i=1}^{4} w_{2i}^{(3)} h_i^{(2)}\right) \tag{1.15b}$$

donde abreviamos los dos pasos de las capas anteriores en uno solo, al escribir las f directamente sobre la suma, ya sin pasar por el intermediario $z_{1,2}^{(3)}$.

El proceso anterior, con iteraciones con respecto a dos índices, y múltiples sumas de la forma $\sum a_i b_i$, puede ser representado por medio de matrices. Las sumas de la ecuación 1.11, por ejemplo,

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \\ z_4^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} & w_{15}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} & w_{25}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} & w_{34}^{(1)} & w_{35}^{(1)} \\ w_{41}^{(1)} & w_{42}^{(1)} & w_{43}^{(1)} & w_{44}^{(1)} & w_{45}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix} \tag{1.16}$$

donde la matriz de pesos tiene 5 columnas (tantas como nodos hay en la capa anterior) y 4 renglones (tantos como nodos hay en la capa actual). Ahora, las sumas de la ecuación 1.13 son,

$$\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ z_4^{(2)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} & w_{14}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} & w_{24}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} & w_{34}^{(2)} \\ w_{41}^{(2)} & w_{42}^{(2)} & w_{43}^{(2)} & w_{44}^{(2)} \end{bmatrix} \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ h_3^{(1)} \\ h_4^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \end{bmatrix} \tag{1.17}$$

donde vemos que, al haber cuatro neuronas en la capa anterior y la actual, la matriz de pesos es cuadrada. Finalmente, el tercer envío de señales, de acuerdo a la ecuación 1.15,

$$\begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} & w_{14}^{(3)} \\ w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} & w_{24}^{(3)} \end{bmatrix} \begin{bmatrix} h_1^{(2)} \\ h_2^{(2)} \\ h_3^{(2)} \\ h_4^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(3)} \\ b_2^{(3)} \end{bmatrix} \tag{1.18}$$

Y finalmente $f\left(z_k^{(3)}\right)$ resulta en las salidas a_k, como vemos en la ecuación 1.15. En nuestra notación matricial:

$$\begin{bmatrix} a_1^{(3)} \\ a_2^{(3)} \end{bmatrix} = f\left(\begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \end{bmatrix}\right) \tag{1.19}$$

Estas a_k son las predicciones del modelo para una muestra dada. Idealmente, tendríamos $a_k = y_k$, esto es, las predicciones son iguales a las propiedades verdaderas de la muestra. Las a_k se computan a partir de las propiedades de la muestra, x_i, y los parámetros del modelo, $\boldsymbol{W}$ y $\boldsymbol{b}$, los cuales representan el aprendizaje que el modelo ha logrado. La meta del entrenamiento es, a cada iteración, refinar los parámetros para reducir el error total y lograr las a_k con el mayor parecido posible a y_k.

Una técnica muy usual para realizar el entrenamiento es la *retropropagación*. Este proceso iterativo se basa en la función de costo, J, la cual toma las a_k y y_k del conjunto de entrenamiento y condensa todas las diferencias $(y_k - a_k)$ en un solo escalar que cuantifica el error del modelo. Entonces, el objetivo del entrenamiento es *minimizar* a J. Este proceso se apoya de un método de optimización, como el descenso de gradiente, o su variante estocástica, o el moderno método Adam (sección 4.4.2, [51, 52]) o su variante AdaBelief (sección 4.4.3, [53]). Ejemplificamos a algunas J en la sección 1.8.

Es ahora claro que los parámetros después del siguiente paso de entrenamiento son más refinados que los anteriores. Sin embargo, no hemos tratado la obtención de los parámetros originales, los cuales provienen de técnicas de inicialización especializadas. Su aplicación no es complicada, pero sí es muy influyente en el éxito del modelo. Desarrollamos este tema en la sección 1.7.

No hemos terminado con nuestra revisión acerca de las redes neuronales. Según el planteamiento de la sección 1.2, haremos experimentos de clasificación (discutidos en el capítulo 6). Este tipo de experimentos requiere tomar algunas decisiones adicionales, pero consideramos oportuno el abordarlas una vez que hayamos elegido una base de datos y las propiedades objetivo de nuestros experimentos. Entonces, cubriremos los ajustes finales de nuestros experimentos en el capítulo 4, particularmente en las secciones 4.2 y 4.3.

1.7 Métodos de inicialización de parámetros

Los arreglos de pesos e incrementos de un modelo, $\boldsymbol{W}$ y $\boldsymbol{b}$, como los mostrados en la ecuación 1.18 se refinan en cada iteración del entrenamiento, tal que al final tenemos las $\boldsymbol{W}$ y $\boldsymbol{b}$ que minimizan la función de costo J. Trataremos a J en la sección 1.8.

En esta sección mostraremos dos maneras de obtener los parámetros iniciales, una para experimentos de clasificación (sección 1.7.1), otra muy usual en experimentos de regresión (sección 1.7.2). Seguir la técnica de inicialización pertinente para nuestro experimento es una etapa de suma importancia, dado que algunos conjuntos de parámetros pueden hacer imposible la optimización del modelo; la técnica correcta permite evitarlos.

Por ejemplo, en varias áreas de las ciencias computacionales es tradicional inicializar una matriz con valores `0` o `1`. Si hiciéramos esto para una NN, tendríamos que la propagación de señales de la capa de entrada a la primer capa oculta (ecuación 1.16) tiene a cada x_i distinto, pero todos los elementos de $\boldsymbol{W}$ son iguales entre sí, así como todos los elementos de $\boldsymbol{b}$. Por lo tanto, todas las $z_j^{(1)}$ son iguales, así como son todas las z_j de capas posteriores. Es decir, cada capa del modelo tiene unidades totalmente simétricas, y entonces el entrenamiento modificará de manera homogénea a todos los elementos de cada $\boldsymbol{W}$ y $\boldsymbol{b}$ del modelo. Esto todavía podría conducir al modelo a un mínimo local más deseable, pero los parámetros finales no representarían un aprendizaje con un significado físico ni tendrían relevancia predictiva: el modelo sería fundamentalmente inútil.

Los métodos de inicialización modernos hacen lo contrario que el ejemplo anterior, dado que, siendo conceptual y computacionalmente sencillos, condicionan al modelo a lograr un aprendizaje adecuado. Sin embargo, antes de su introducción, el panorama de la aplicabilidad de las redes neuronales, especialmente las profundas, era más laborioso y menos esclarecido: A finales de la década que inició en el año 2000, el campo del aprendizaje profundo (*deep learning*) se tornó hacia problemas como la visión por computadora y el procesamiento de lenguaje natural. Estos son problemas complejos, por lo cual se recurrió a redes neuronales profundas (*deep neural networks*, DNN, de ahí el nombre de la subdisciplina), que presentan más de una capa oculta, como la mostrada en la figura 1.3b.

Estas capas adicionales incrementan la flexibilidad funcional y capacidad de abstracción y síntesis de la red; las capas ocultas iniciales se especializan en identificar los descriptores correlacionados, y entonces componer descriptores de más alto nivel. Por otra parte, la última capa oculta interpreta estos descriptores artificiales y refinados, para finalmente predecir las propiedades de la muestra.

Podemos ver que las capas ocultas adicionales brindan a una red profunda la riqueza y complejidad necesarias para abordar problemas mayores, a cambio de un entrenamiento con mayor costo computacional. Lo que es más, las primeras DNN eran de entrenamiento difícil, dado que podrían rehusarse a mejorar sus predicciones durante su entrenamiento, incluso si las inicializáramos usando los métodos de aquella época, los cuales eran exitosos para las redes de una sola capa oculta.

En la práctica, esta cuestión podía sortearse con, por ejemplo, el método de Hinton, Osindero, y Teh [54], donde, una capa a la vez, se realiza un pre-entrenamiento usando una máquina de Boltzmann restringida, la cual es un método semejante a una red neuronal, con solamente dos capas: una capa oculta, y una denominada *visible*, que funge *tanto como capa de entrada* como *de salida* [55]. Erhan *et al.* [56] dan un tratamiento accesible acerca de las diferentes formas en que el pre-entrenamiento de un modelo influye en su desempeño, y evidencian la importancia de este paso en la era previa a las inicializaciones Xavier y Kaiming, las cuales explicamos en las secciones 1.7.1 y 1.7.2.

La exploración del cómo resolver la complejidad de la inicialización de las redes profundas resultó en la inicialización Xavier. Este método absuelve a los experimentos de clasificación de realizar un costoso pre-entrenamiento a fin de inicializar su modelo. El caso de experimentos de regresión o experimentos con funciones de activación semiacotadas fue resuelto andando los años, por la

inicialización Kaiming.

1.7.1 Inicialización Xavier

Este método de inicialización, propuesto por Glorot y Bengio en 2010 [57], robustece los experimentos de clasificación que usan una función acotada superior e inferiormente como las de la figura 1.4. Esta inicialización prácticamente garantiza que el experimento no se verá limitado por un punto de inicio inadecuado, y entonces el aprendizaje podrá proceder hasta donde lo permita el resto de componentes del experimento, a saber: la relevancia de la información aportada por los descriptores, la riqueza del conjunto de entrenamiento, y la elusividad intrínseca de la propiedad objetivo.

Esta técnica también se conoce como 'inicialización Glorot', refiriéndose al apellido del primer autor del método, Xavier Glorot [57]. Hemos optado por el nombre 'inicialización Xavier' a fin de concordar con nuestras referencias.

El problema más usual de las redes profundas contemporáneas a Glorot y Bengio era su tendencia a *saturarse* durante el entrenamiento, a veces de forma irreversible. Esto es, tomando la figura 1.4 como ejemplo, vemos que ambas funciones de activación son prácticamente constantes cuando $|z| \gg 0$. Por lo tanto, un experimento con una red profunda con una mala inicialización, se encontraría que según progresan las señales a lo largo de la red, las z son cada vez más grandes o más pequeñas. Esto tendría dos consecuencias negativas:

- las neuronas de las capas tardías estarían *saturadas*: dada la magnitud acumulada de las señales anteriores, las z de esta capa serían $\approx \pm 1$ o ≈ 0, y entonces el gradiente del costo (∇J) se habría *desvanecido*. Esto significa que,
- el proceso de retropropagación optimiza los parámetros del modelo para reducir el costo (J). Para esto, se calcula el gradiente de la función de costo con respecto a los parámetros, los cuales se actualizan de acuerdo a un optimizador usualmente derivado del descenso de gradiente. Por lo tanto, los parámetros de capas saturadas tomarán muchas iteraciones para lograr una reducción significativa de J.

Glorot y Bengio se enfrentaron a este problema durante sus entrenamientos de redes de una hasta cinco capas ocultas para el reconocimiento de números y de figuras geométricas. Sus modelos fueron inicializados con incrementos 0; los pesos, siguiendo las heurísticas populares de la época, provenieron de la distribución uniforme

$$U\left[-\frac{1}{\sqrt{n_{j-1}}}, \frac{1}{\sqrt{n_{j-1}}}\right] \tag{1.20}$$

donde n_{j-1} es la cantidad de neuronas en la capa anterior. De aquí encontraron que, por ejemplo, la NN de 4 capas ocultas tuvo su última capa oculta saturada en 0 durante aproximadamente 100 iteraciones, eventualmente logrando desaturarse, desencadenando cambios súbitos en todas las matrices de pesos del modelo. Por otra parte, la red de 5 capas ocultas se mantuvo saturada a lo largo de todo el entrenamiento: dada la profundidad de esta red, el método de inicialización original generó parámetros que amplificaron las señales al grado de imposibilitar el aprendizaje.

A fin de resolver este problema, Glorot y Bengio buscaron las condiciones a imponer sobre la inicialización de los modelos, tal que se preserve la magnitud de señal a lo largo de la red, durante la propagación hacia adelante, *y también* la de los gradientes de la función de costo a lo largo de la retropropagación. Esto resolvería el problema del gradiente desvanecido. A fin de encontrar estas condiciones, se establece que la varianza de las señales de un modelo recién inicializado, sea la misma para todas sus capas. También se buscan las condiciones que cumplen que la varianza de la derivada parcial de la función de costo con respecto a las z del modelo, sea constante. Hlav [58] da una explicación gradual y didáctica de la derivación matemática de Glorot y Bengio.

Como resultado de este proceso matemático, obtenemos la media y desviación estándar de una distribución que cumple las condiciones impuestas por Glorot y Bengio:

$$\mu = 0 \tag{1.21}$$

$$\sigma = \sqrt{\frac{2}{n_{j-1} + n_j}} \tag{1.22}$$

donde n_{j-1} y n_j son la cantidad de neuronas en la capa anterior y en la actual. Por lo tanto, n_{j-1} y n_j son también las dimensiones de la matriz de pesos que conecta a las dos capas elegidas.

Hemos obtenido dos propiedades fundamentales de la distribución que buscamos, pero todavía no conocemos su forma o tipo. En la práctica, las distribuciones normal y uniforme han mostrado su habilidad para generar pesos conducentes a un buen entrenamiento. Además, no se conocen razones para preferir definitivamente a una de estas distribuciones o a la otra.

Finalmente, podemos inicializar los pesos de una red usando la distribución normal de Xavier:

$$N\left(0, \frac{2}{n_{j-1} + n_j}\right) \tag{1.23}$$

donde hemos denotado a la distribución normal de la forma $N(\mu, \sigma^2)$. Por otra parte, inicializamos los incrementos $\boldsymbol{b}$ en 0.

Por otra parte, si prefiriéramos usar la distribución uniforme de Xavier, hemos de usar la media y varianza de las ecuaciones 1.21 y 1.22 para obtener los umbrales de la distribución uniforme. Para esto, empleamos dos de sus ecuaciones fundamentales. Sean u_1 y u_2 las cotas de una distribución uniforme, entonces, según explica Weisstein [59], tenemos que

$$\mu = \frac{u_1 + u_2}{2} \tag{1.24}$$

$$\sigma = \frac{(u_2 - u_1)^2}{12} \tag{1.25}$$

A partir de estas ecuaciones podemos llegar a la distribución uniforme de Xavier. Si denotamos a una distribución uniforme de la forma $U[u_1, u_2]$, tenemos que la distribución buscada es:

$$U\left[-\sqrt{\frac{6}{n_{j-1} + n_j}}, \sqrt{\frac{6}{n_{j-1} + n_j}}\right] \tag{1.26}$$

la cual usamos para inicializar las matrices de pesos. Al igual que en la distribución anterior, los incrementos $\boldsymbol{b}$ se inicializan en 0.

Estas dos distribuciones provienen del análisis hecho por los autores [57] para una red neuronal profunda con $\tanh(z)$ como función de activación. Mostramos esta función en la figura 1.4; notemos que las dos f presentadas son acotadas por ambos lados.

La fundamentación de la inicialización Kaiming asume que f es impar y tiene $f'(0) = 1$; $\tanh(z)$ cumple estos requisitos. Sin embargo, nuestros resultados en el capítulo 6 muestran que este método también es compatible con redes clasificativas con $f(z) = \text{logística}(z)$, dado que logran un buen aprendizaje.

1.7.2 Inicialización Kaiming

La inicialización Kaiming es un sucesor de la inicialización Xavier, pensada para entrenamientos que usan una f de la familia ReLU, donde la inicialización Xavier puede saturar irreversiblemente a un modelo. En 2015, los autores de este método, He *et al.* [60], entrenaron redes convolucionales para

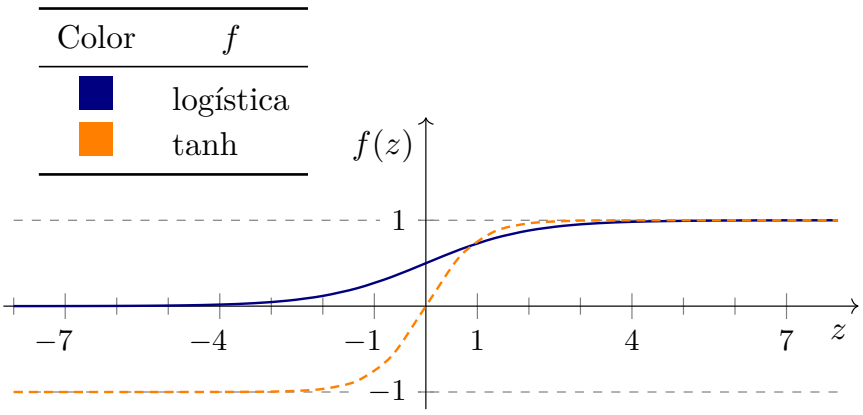


Figura 1.4: Funciones de activación para modelos clasificativos. Mostramos las ecuaciones en la sección 1.5.

clasificar imágenes según los objetos que contienen, usando la base de datos *ImageNet 2012* [61]. La derivación matemática detrás de esta inicialización es explicada de manera didáctica por Hlav [62].

Esta técnica también se conoce como 'inicialización He', refiriéndose al apellido del primer autor del método, Kaiming He. Hemos optado por el nombre 'inicialización Kaiming', al ser una denominación popular del método.

Los experimentos de He *et al.* fueron diferentes a los de Glorot y Bengio:

- usaron redes de hasta 22 y 30 capas ocultas en lugar de solamente 5,
- en lugar de usar la función de activación $\tanh(z)$, trabajaron con funciones de la familia ReLU, semiacotadas, como las que mostramos en la figura 1.5,
- trabajaron con redes convolucionales, en lugar de las redes *feed-forward* ordinarias como las que usamos a lo largo de esta ICR y detallamos en la sección 1.6.

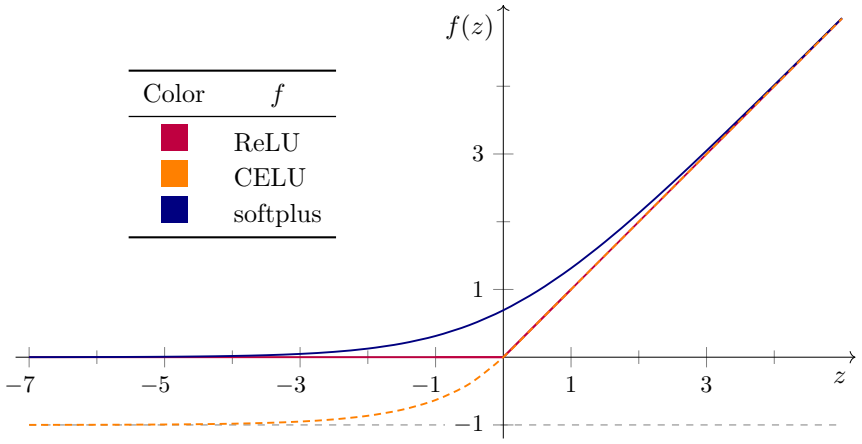


Figura 1.5: Algunas funciones de activación de la familia ReLU, habitualmente usadas en los modelos de regresión. Esta función CELU tiene $\alpha = 1$. Mostramos las ecuaciones en la sección 1.5.

En su artículo, los autores realizan una derivación matemática [60] semejante a la de Glorot y Bengio, esta vez tomando en cuenta las características de las funciones de la familia ReLU. A diferencia de las funciones tipo sigmoide de la sección anterior, estas funciones solo tienen una cota. Por lo tanto, modelos con estas funciones son susceptibles al problema del gradiente desvanecido, y

también al del gradiente detonado (*exploding gradient*), cuando la señal se magnifica al pasar por las capas, resultando en un gradiente gigantesco y un entrenamiento inestable. El método Kaiming busca generar un conjunto de parámetros con poca propensión a estos dos problemas.

La importancia de esta inicialización es especialmente clara en redes profundas con función de activación de la familia ReLU: He *et al.* muestran que una CNN de 22 capas ocultas inicializada con el método de Kaiming tiene un entrenamiento más veloz que esa misma red inicializada con el método Xavier, aunque los desempeños de las dos son prácticamente iguales. Por otra parte, la CNN con 30 capas ocultas inicializada con Kaiming converge rápidamente, mientras que la de Xavier se mantiene saturada durante todo el proceso, nunca logrando aprender.

La derivación de Kaiming *et al.* [60] encuentra la media y varianza de las distribuciones que generan pesos conducentes a un entrenamiento deseable:

$$\mu = 0 \tag{1.27}$$

$$\sigma = \sqrt{\frac{2}{n_{j-1}}} \tag{1.28}$$

donde n_{j-1} es la cantidad de neuronas en la capa anterior.

Podemos tomar esas propiedades para construir la distribución normal de Kaiming,

$$N\left(0, \frac{2}{n_{j-1}}\right) \tag{1.29}$$

donde denotamos a la distribución normal de la forma $N(\mu, \sigma^2)$. Inicializamos los incrementos en 0.

Al igual que con la inicialización Xavier, podemos trabajar con la inicialización Kaiming y preferir una distribución uniforme en lugar de una normal. Siguiendo las ecuaciones 1.24 y 1.25, tenemos que la distribución uniforme de Kaiming es:

$$U\left[-\sqrt{\frac{6}{n_{j-1}}}, \sqrt{\frac{6}{n_{j-1}}}\right] \tag{1.30}$$

Y una vez más inicializamos los incrementos en 0. No se conocen motivos para preferir definitivamente a la distribución normal sobre la uniforme, o viceversa.

1.8 Función de costo, J

El entrenamiento de un modelo ML sucede en una superficie multidimensional, denominada la función de costo J, con tantas dimensiones como parámetros hay en el modelo. J es explorada por un optimizador, iterativamente buscando uno de sus mínimos locales. El punto inicial de la exploración proviene de un método de inicialización, como los mostrados en la sección 1.7.

Al encontrarnos suficientemente cerca del mínimo, consideramos que el modelo ha sido entrenado; sus predicciones para las muestras de entrenamiento ya no pueden mejorar. Idealmente, la capacidad predictiva del modelo empeorará apenas un poco al pasar a muestras futuras, las cuales no formaron parte del entrenamiento, para las cuales el modelo realiza una predicción a partir del conocimiento que ha aprendido.

Entonces, la elección de J es otro paso fundamental antes de poder realizar experimentos ML. En esta sección mostraremos algunas funciones de costo para experimentos de clasificación y regresión.

1.8.1 J para clasificación

La entropía cruzada binaria (*binary cross-entropy*) es una función de costo muy habitual para experimentos de clasificación donde cada muestra pertenece a *una* clase. Podemos calcular el costo de la muestra m con:

$$J_m(\boldsymbol{W}, \boldsymbol{b}) = -\frac{1}{c}\sum_{i=1}^{c}\Big[y_i \ln(a_i) + (1 - y_i)\ln(1 - a_i)\Big] \tag{1.31}$$

donde c es la cantidad de neuronas en la capa de salida, que corresponde a la cantidad de clases en el experimento, y_i es la etiqueta de la clase i, con valor 0 para las clases a la que la muestra no pertenece, 1 para las que sí. Finalmente, a_i son las predicciones del modelo para cada clase.

Debido a su formulación, esta función busca que las a_i se asemejen a elementos de una base canónica de tamaño c, esto es, vectores con c elementos, donde $c - 1$ son 0 y uno es 1.

Por ejemplo, supongamos que nuestro experimento es de tres clases, y m pertenece a la clase 2, entonces su vector de etiquetas es $(0, 1, 0)$, y nuestro modelo actual arroja puntuaciones $(0.12, 0.75, 0.15)$. Entonces, el costo de m es

$$\begin{aligned} J_m(\boldsymbol{W}, \boldsymbol{b}) = -\frac{1}{3}\Big[&\cancel{0\ln(0.12)} + (1-0)\ln(1-0.12) \\ &+1\ln(0.75) + \cancel{(1-1)\ln(1-0.75)} \\ &+\cancel{0\ln(0.15)} + (1-0)\ln(1-0.15) \\ &\Big] \\ = 0.193& \end{aligned}$$

Podemos ver que los dos términos de cada clase siempre tienen un coeficiente 0 y un coeficiente 1, mediante los cuales se toma en cuenta la diferencia entre a_i y 0, o a_i y 1, según el carácter de cada clase.

1.8.2 J para regresión

En un experimento de regresión buscamos reducir la diferencia entre las predicciones a_i del modelo y los valores verdaderos de las muestras, y_i. Una función de costo habitual es el error cuadrado promedio (*mean square error*, MSE). Para la muestra m tenemos el costo:

$$J_m(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{c}\sum_{i=1}^{c}(a_i - y_i)^2 \tag{1.32}$$

donde c es la cantidad de neuronas en la capa de salida, que corresponde a la cantidad de propiedades predichas por nuestro modelo.

Podemos ver que esta función de costo amplifica la contribución de los errores más grandes, y reduce la de errores pequeños. Esta es una propiedad altamente deseable: el entrenamiento dará prioridad a la reducción de los errores mayores, al costo de que los errores menores serán más permisibles.

Otra función de costo es el error absoluto promedio (*mean absolute error*, MAE):

$$J_m(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{c}\sum_{i=1}^{c}|a_i - y_i| \tag{1.33}$$

donde vemos que esta función no introduce el sesgo o priorización que MSE tiene en contra de los errores más grandes. Por lo tanto, un entrenamiento con esta función de costo busca simplemente

reducir el error total a lo largo de todas las muestras, y el modelo resultante podría tener una mayor dispersión en la magnitud de sus errores.

Alternativamente, podemos formular un experimento con MSE como función de costo, y MAE como una métrica secundaria, en lugar de tratarla como objetivo.

1.9 Métodos de validación

El objetivo fundamental detrás del desarrollo de modelos ML es el obtener un modelo capaz de predecir acertadamente las propiedades de muestras ajenas a la base de datos, desconocidas para el modelo pero que siguen las mismas reglas que las muestras usadas en el entrenamiento. Es decir, esperamos entrenar un modelo con un error de generalización muy bajo.

A fin de poder estimar el error de generalización de un modelo, la técnica más usual es dividir la base de datos en un conjunto de entrenamiento (mayoritario) y uno de prueba (minoritario). El primero es usado a lo largo de todo el entrenamiento, mientras que el segundo solo es usado al final. Una vez que el modelo está entrenado, usamos los dos conjuntos para obtener estas medidas:

- el error para el conjunto de prueba,

  es una medida aproximada del error de generalización del modelo (el error que podemos esperar para las muestras futuras).

- la diferencia entre el error de entrenamiento y el de prueba,

  la cual nos indica el grado de *overfit* del modelo: si el modelo logra un error de entrenamiento bajo, mientras que el error de prueba es alto, esto indica que el modelo está sujeto a un *overfit*: el modelo ha aprendido las particularidades de las muestras de entrenamiento, en lugar de reconocer los patrones más profundos y generales que conectan a las entradas con las salidas. Esto indica que debemos reformular el modelo, tal vez añadiendo más neuronas y capas, o diversificando el conjunto de entrenamiento.

A fin de obtener esas medidas (error de generalización y nivel de *overfit*) y evaluar el desempeño de nuestro modelo de forma robusta y comparable con otros métodos semejantes presentes en la literatura o desarrollados por otros grupos de investigación, los experimentos ML suelen realizarse en del marco de un método de *cross-validation* (le llamaremos simplemente *método de validación*).

En esta sección explicamos dos de los métodos más usuales: *holdout* y k-fold.

1.9.1 Validación *holdout*

Este es el método de validación más simple. Sus pasos son:

1. desordenar aleatoriamente la base de datos,
2. dividir la base en un conjunto de entrenamiento y uno de prueba,
3. entrenar al modelo ML,
4. evaluar el desempeño del modelo con las moléculas del conjunto de prueba.

Es usual dividir la base en 80 % para el conjunto de entrenamiento y 20 % para el de prueba, aunque también son comunes las particiones (85 %, 15 %) y (90 %, 10 %).

Una objeción muy natural a esta técnica es que los conjuntos de prueba y entrenamiento son definidos de manera aleatoria, sin algún criterio en particular. Esto significa que algunas muestras importantes podrían ser excluidas del conjunto de entrenamiento, y entonces el modelo tendría un comportamiento pobre con este tipo de muestras.

También, dado que el reordenamiento es aleatorio, las propiedades de los conjuntos de prueba y entrenamiento pueden cambiar de un experimento a otro, incluso si el resto de propiedades se mantiene constante. El método k-fold, que mostramos a continuación, busca resolver estas deficiencias.

1.9.2 Validación k-fold

Comparado con el método *holdout*, el método k-fold[†] es conceptualmente sencillo, y también más sofisticado, robusto, y computacionalmente costoso. Dado que genera resultados más fiables que el método *holdout*, es el método que usamos para nuestros experimentos.

El método k-fold realiza k experimentos usando la misma base de datos y arquitectura, pero diferentes conjuntos de prueba y entrenamiento. Sus pasos son:

1. reordenar aleatoriamente la base de datos,
2. particionar la base en k folds de igual tamaño,
3. entrenar al modelo ML, usando al primer fold como conjunto de prueba y al resto de $(k-1)$ folds como conjunto de entrenamiento,
4. entrenar un segundo modelo, ahora usando al segundo fold como conjunto de prueba y a los $(k-1)$ restantes como conjunto de entrenamiento,
5. el proceso se repite hasta llegar a k experimentos. Al final promediamos las medidas de error y de desempeño de los experimentos.

Mostramos esquemáticamente el funcionamiento del método en la figura 1.6, en el caso particular $k = 5$.

Podemos ver que, debido a su formulación, un experimento con el método k-fold realiza k entrenamientos, mientras que el método *holdout* podría realizar solo uno. Por lo tanto, los experimentos k-fold son más costosos que los experimentos *holdout* más sencillos.

También, podemos ver que *todas* las muestras participan en el conjunto de entrenamiento y en el de prueba. Esta es una forma de sortear la pregunta del porqué asignamos a cierta muestra al conjunto de prueba o al de entrenamiento, al tiempo de robustecer los valores hallados en la evaluación del modelo.

Es convencional el usar $k = 5$ o $k = 10$. Como explican Breiman y Spector [63], elegir $k = 5$ es una configuración aceptable en el caso general.

[†]Escribiremos el nombre del método de una manera poco convencional: k-fold. Normalmente, escribiríamos la palabra 'fold' con itálicas, indicando que la palabra proviene del inglés y se muestra sin traducir. Sin embargo, preferimos la grafía 'k-fold' dado que concentra el énfasis itálico sobre la k, una variable sin valor concreto; si italizáramos también a *fold*, ya no sería instantáneo el identificar a k como variable. Y, buscando ser consistentes, no italizaremos la palabra 'fold' incluso cuando no esté en la vecindad de k.

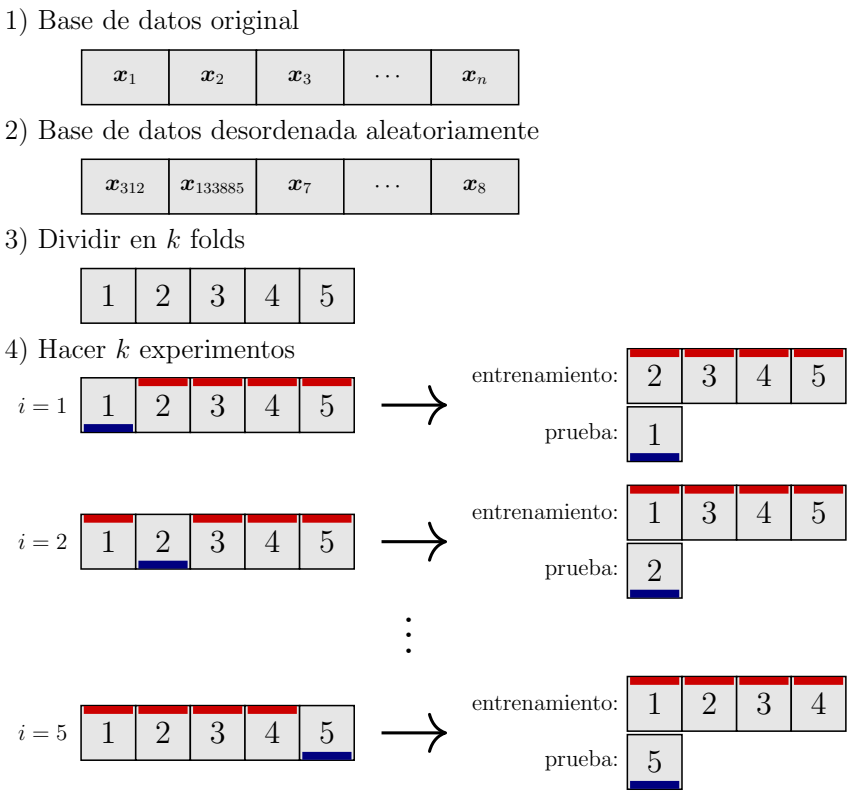


Figura 1.6: Funcionamiento del método de validación k-fold. En este ejemplo tenemos $k = 5$.

Capítulo 2

Bases de datos

Índice de capítulo

2.1 Introducción

La viabilidad de un proyecto o experimento ML depende fuertemente de la existencia de una base de datos que represente de manera fiel al espacio sobre el cual queremos hacer predicciones. En el caso general, la cantidad de muestras posibles es infinita, como pasa en el dominio de traducción de textos o el reconocimiento de objetos en una fotografía. Por lo tanto, la meta de un proyecto ML no es el construir una base de datos exhaustiva que englobe la totalidad del espacio de búsqueda. En su lugar, nos basta con un subconjunto finito, donde el espacio de búsqueda esté bien representado. Con éste, podemos entrenar una máquina que tendrá un desempeño deseable para este conjunto de datos, y también tendrá un desempeño comparablemente bueno para muestras fuera del conjunto de entrenamiento, las cuales quizás fueron generadas mucho después de que el entrenamiento del modelo haya terminado. Cuando esto sucede, decimos que el modelo presenta un *error de generalización* bajo y podemos confiar en sus predicciones.

Los experimentos químicos, sin embargo, pueden presentar algunas diferencias con el caso general. Por ejemplo, supongamos que el espacio de moléculas es nuestro espacio de búsqueda, y que lo acotamos a las moléculas con largo de cadena ≤ 9, que cumplan con ciertos criterios de estabilidad. Pues bien, este espacio es finito y discreto, y entonces podemos calcular algunas propiedades moleculares aplicando métodos de química computacional a todas las moléculas de este espacio. Esto significa que podemos conocer las propiedades de todo este espacio en tiempo finito.

Esta es la razón de que el espacio de moléculas discretas a veces sea peculiar; en casos más allá de la química siempre es posible generar muestras nuevas, inéditas para el modelo, el cual ha de realizar sus predicciones a partir de la información contenida en sí (es decir, hará sus predicciones a partir de su aprendizaje).

Dicho esto, incluso un modelo ML-químico podría encontrarse con datos generados después de la conclusión del entrenamiento: si en lugar de trabajar con el espacio de moléculas, trabajáramos con el de geometrías moleculares, entonces cambios leves en la constelación atómica generan muestras

distintas, cuya estructura y propiedades son ahora distintas (como su energía). Un estudio como este trabaja en un espacio muestral infinito: el modelo eventualmente se encontrará con muestras ajenas a su conjunto de entrenamiento. A fin de robustecer su desempeño y que sea posible tratar muestras inéditas, es necesario trabajar con una base de datos heterogénea y diversa, que represente bien a todas las regiones del espacio muestral.

Tomando en cuenta esta multitud de factores, es claro que el ML químico ha de seguir el mismo camino que el ML general, donde el primer paso es la generación de una base de datos adecuada. En este capítulo veremos algunas bases de datos químicas, públicas, y robustas, con las que podemos realizar experimentos de química computacional serios, y las consideraciones detrás de las mismas. La dificultad de emprender un proyecto ML químico cualquiera disminuye drásticamente debido a la existencia de estas bases.

2.2 Familia GDB

Las bases de datos GDB-11 [64, 65], GDB-13 [66], y GDB-17 [67] son repositorios moleculares públicos y gigantescos de moléculas de hasta 11, 13, o 17 átomos pesados (es decir, $\{\mathrm{C}, \mathrm{N}, \mathrm{O}, \mathrm{F}\}$ con sus correspondientes hidrógenos según la conectividad de cada molécula). Estas bases tienen cardinalidades respectivas de 2.64×10^7, 9.77×10^8, 1.66×10^{11}.

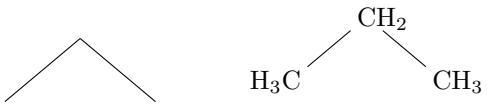
Esta familia de bases de datos son un ejemplo ambicioso del poder de la química matemática. En particular, el objetivo detrás de su creación es la exploración del espacio químico de moléculas orgánicas: la exploración directa a partir de métodos tradicionales como experimentos o simulaciones estará generalmente acotada a los grupos funcionales y bloques estructurales con los que la humanidad está familiarizada. Por otra parte, el fundamento detrás de GDB es matemático y combinatorio, y busca lograr una diversidad estructural más allá de las regiones químicas habituales: la base GDB-11 fue publicada en 2006, y de sus 1208 sistemas de anillos, 538 no estaban definidos en las prominentes bases *CAS Registry* y de Beilstein-Reaxys.

GDB-11, siendo la base más pequeña de la familia, ya presenta una diversidad química promisoria: todas sus moléculas cumplen con las reglas propuestas por Lipinski *et al.* [68] (también conocidas bajo el nombre *regla de 5*), un criterio heurístico que estima la biodisponibilidad de una molécula en el organismo a partir de su masa molecular, coeficiente de partición n-octanol-agua calculado (CLogP), y su cantidad de donadores y aceptores de puente de hidrógeno. Dado que todas las moléculas GDB-11 cumplen con la regla de 5, los autores suponen que todas tienen potencial en el campo farmacológico. También, casi la mitad de la base cumple con las reglas propuestas por Congreve *et al.* [69] (también conocidas como *regla de 3*), otra heurística que identifica las moléculas *lead-like* con potencial farmacológico; sus criterios son similares, pero más estrictos, que los de la regla de Lipinski.

El fundamento detrás de las bases GDB es la teoría de grafos. Esto es, en lugar de crear moléculas ensamblando átomos y grupos funcionales, la construcción se realiza mediante grafos, entidades matemáticas que existen fuera de un contexto químico, constituidos por vértices y aristas en lugar de átomos y enlaces.

El primer paso para el proceso GDB es enlistar todos los grafos con una cantidad de nodos (vértices) menor o igual que, digamos, 11, donde cada nodo tiene una conectividad máxima de 4. Para GDB-11, esta etapa generó 843 335 grafos. Mostramos un grafo ejemplo en la figura 2.1a. La lista de grafos fue filtrada para descartar a los que generarían moléculas inestables, como aquellas en los que un átomo participa en múltiples ciclos pequeños, o estructuras con conectividad imposible. 15 726 grafos pasaron todos los filtros, y se generó el alcano correspondiente a cada uno, como el que mostramos en la figura 2.1b para el grafo de la figura 2.1a.

El siguiente paso es la adición combinatoria de insaturaciones (enlaces dobles y triples) y de heteroátomos $(\mathrm{N}, \mathrm{O}, \mathrm{F})$. Para GDB-11, este paso resultó en más de 1.7×10^9 estructuras. Mostramos



(a) Grafo. (b) Alcano del grafo.

Figura 2.1: Dos etapas iniciales del proceso matemático-químico que genera las moléculas GDB-11, ejemplificadas con un grafo.

en la figura 2.2 algunas de las moléculas derivadas del alcano de la figura 2.1b.

Aunque el paso anterior generó $> 1.7 \times 10^9$ estructuras iniciales, la cardinalidad de GDB-11 es de apenas $\approx 2.64 \times 10^7$. Esto se debe a que los autores implementaron múltiples filtros para descartar las moléculas que ellos consideran inestables, y así acotar la base al espacio de moléculas sintetizables. Si denotamos con R a una cadena orgánica cualquiera*, podemos escribir algunos de los arquetipos moleculares descartados como: las moléculas con enlaces heteroátomo-heteroátomo, o aquellas con alguno de estos grupos funcionales:

- gem-diol, $R_2C(OH)_2$,
- aminales, $R_2C(NR_2)_2$,
- ortoácidos, $RC(OH)_3$,
- fluoruros de acilo, RCOF,
- enoles.

Esta batería de filtros excluye a moléculas como CO_2 u óxidos de nitrógeno como N_2O y NO_2. La figura 2.2 también muestra otras moléculas descartadas, son las que no están indicadas con un cuadro sombreado debajo de la molécula. Para las que cuentan con este símbolo, el índice i dentro del cuadro es su posición en el archivo de tres átomos pesados de GDB-11.

Mostramos cuatro de estos archivos en el apéndice B. Los archivos de todas las bases GDB están disponibles gratuitamente en el sitio web del autor [70].

Dada la inmensa diversidad química contenida en las bases GDB, podríamos pensar que el manejo de la base conlleva requisitos inmensos de espacio en disco. A fin de atenuar este problema, las moléculas son descritas únicamente mediante el lenguaje minimalista SMILES [36] (*simplified molecular-input line-entry system*). Como resultado, los 11 archivos de GDB-11 con sus 2.64×10^7 moléculas ocupan apenas 701 MB en disco.

SMILES codifica a una molécula y su conectividad en una sola línea de texto ASCII. Brevemente: una *string* SMILES recorre el grafo molecular listando uno a uno los átomos pesados de una molécula, los átomos con valencia incompleta son saturados con hidrógenos, las ramificaciones se delimitan con paréntesis, y las insaturaciones se indican con `=` y `#`. Entonces:

- `C` es el metano, CH_4,
- `N` es el amoniaco, NH_3,
- `CCC` es el propano, $CH_3CH_2CH_3$,
- `C(C)(C)(C)` es el neopentano, $C(CH_3)_4$,
- `C=O` es el formaldehido, $H_2C{=}O$,

*Usar a R de esta manera nos permite hablar de estructuras orgánicas en general de manera bastante concisa. Por ejemplo, podemos referirnos a los alcoholes simplemente escribiendo 'R−OH' o 'ROH', en lugar de dar un listado explícito como 'los alcoholes, es decir, CH_3OH, CH_3CH_2OH, $CH_3CH_2CH_2OH$, ...'.

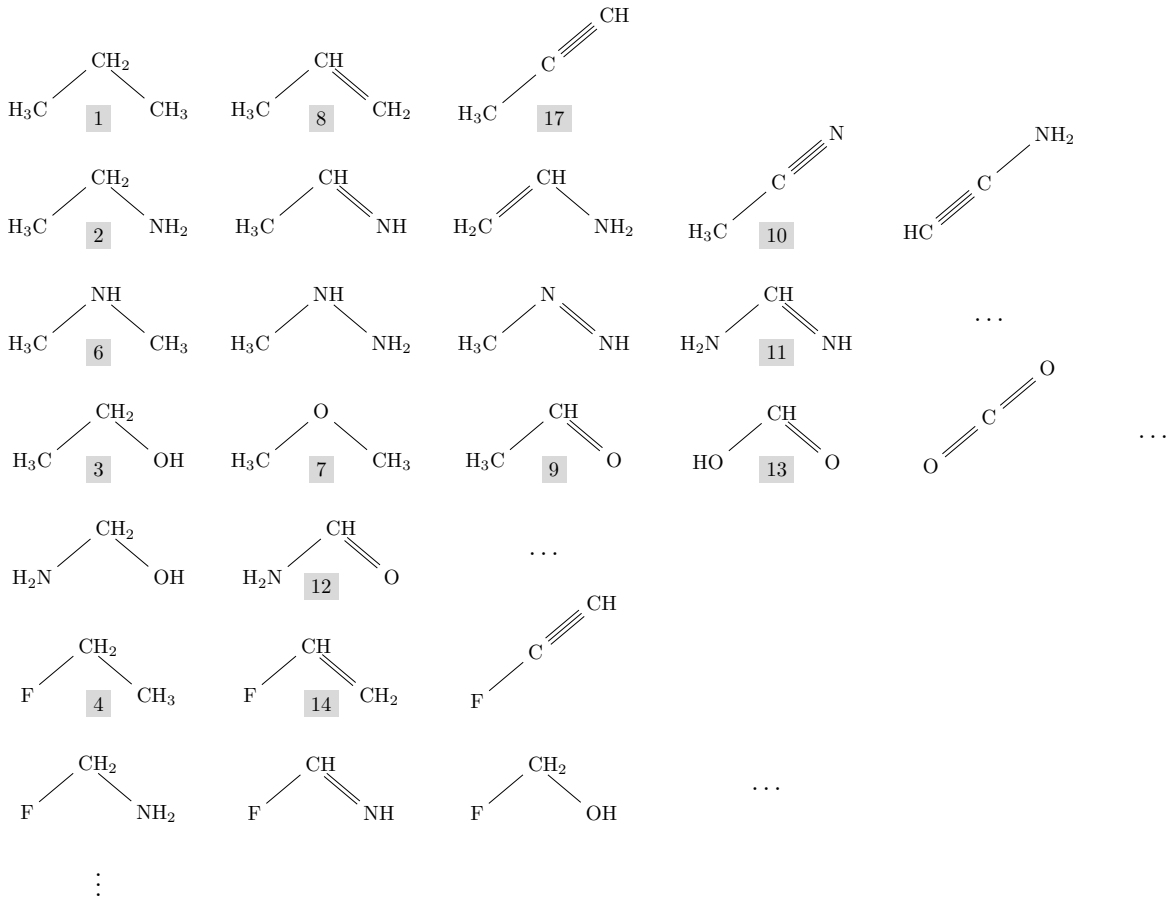


Figura 2.2: Adición de heteroátomos e insaturaciones. Solo algunas de estas moléculas fueron incorporadas a GDB-11, particularmente a la lista de moléculas de tres átomos pesados. Indicamos a las moléculas incluidas con un cuadro sombreado i, indicando su índice i en la lista.

- `C#C` es el acetileno, HC≡CH.

Un recurso que explica a profundidad las reglas del lenguaje SMILES es el sitio web de Daylight [37], una compañía quimioinformática que trabaja de manera cercana con el autor de SMILES.

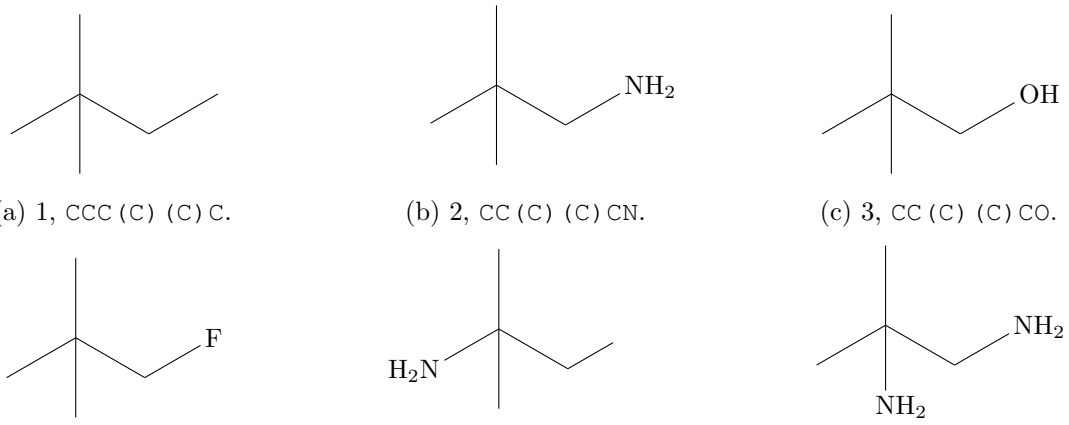
Como ejemplo final, mostramos las primeras seis moléculas de 6 átomos pesados de GDB-11 en la figura 2.3, donde el SMILES de cada molécula está dado al pie. A partir de éste es sencillo dibujar a la molécula, pero obtener propiedades moleculares concretas requiere de otras herramientas computacionales.

Por ejemplo, Open Babel [71] es una manera de obtener las coordenadas atómicas a partir de un SMILES molecular. A partir de éstas se vuelve asequible el realizar cálculos de estructura electrónica, obteniendo así otras propiedades mucho más granulares. Mostramos en el apéndice A un ejemplo de esta conversión.

Existen bases que siguen un enfoque distinto al de la familia GDB, que cambian una cardinalidad inmensa por una menor, y una descripción molecular minimalista por una mucho más detallada. Dos ejemplos importantes son:

- la base QM9 (sección 2.3, [72, 73]), un subconjunto de GDB-17,
- la base PC9 (sección 2.4, [74, 75]) contiene diversas estructuras de moléculas obtenidas experimentalmente, y la mayoría de sus moléculas ($> 80\,\%$) no existe en GDB-17.

Estas dos bases enlistan las coordenadas cartesianas de sus moléculas, y propiedades adicionales como las energías de sus orbitales, momento dipolar, propiedades termodinámicas, entre otras.



(a) 1, `CCC(C)(C)C`. (b) 2, `CC(C)(C)CN`. (c) 3, `CC(C)(C)CO`.

(d) 4, `CC(C)(C)CF`. (e) 5, `CCC(C)(C)N`. (f) 6, `CC(C)(N)CN`.

Figura 2.3: Estructura y SMILES de las moléculas de 6 átomos pesados, de la primera a la sexta, de la base GDB-11 [64, 65].

2.3 QM9: un subconjunto de GDB-17

En 2014, Ramakrishnan *et al.* publicaron la base de datos QM9 [72, 73], con 133 885 moléculas. Esta base, gratuita, accesible y detallada, se ha usado en múltiples estudios de química computacional y aprendizaje de máquina. Por ejemplo, la búsqueda genética de subconjuntos de diferente tamaño que logren predicciones más acertadas que las técnicas convencionales [35], o la búsqueda de propiedades derivadas de la representación SMILES para la predicción de propiedades termodinámicas [39].

QM9 es un derivado de GDB-17 (sección 2.2, [67, 70]), una base minuciosamente desarrollada con técnicas de combinatoria y teoría de grafos. La descripción molecular en GDB-17 es compacta, en el formato SMILES [36, 37], sin coordenadas explícitas ni otras propiedades moleculares.

QM9 es una base mucho más detallada y explícita, y compila de GDB-17 las moléculas de hasta 9 átomos pesados (*non-hydrogen atoms*) C, N, O, F, para describirlas en archivos de texto con coordenadas cartesianas calculadas al nivel B3LYP/6-31G(2df,p), y más información de interés fisicoquímico como las frecuencias vibracionales armónicas, cargas atómicas, momento dipolar, y diversas propiedades electrónicas y termodinámicas.

Esta base es gratuita y accesible; puede descargarse del sitio `Figshare` [73].

2.3.1 Un archivo QM9

La estructura de un archivo QM9 es:

Renglón	Propiedad
1	Cantidad de átomos, n_{at}.
2	Propiedades escalares (ver tabla siguiente).
3, ..., $n_{\text{at}}+2$	De un átomo: Símbolo, coordenadas x, y, z (Å), carga parcial de Mulliken (e)
$n_{\text{at}}+3$	Las frecuencias vibracionales armónicas, en cm^{-1}. Son $3n_{\text{at}}-5$ o $3n_{\text{at}}-6$ según la estructura.

$n_{\text{at}} + 4$	Dos *strings* SMILES, una proveniente de GDB-17 y la otra de la optimización B3LYP. Difieren un poco en moléculas más complejas.
$n_{\text{at}} + 5$	Descriptor InChI de la estructura.

donde Å denota la unidad ångström, tal que $1\,\text{Å} = 1 \times 10^{-10}\,\text{m}$.

Las propiedades escalares, contenidas en el renglón 2 de los archivos, son:

Número	Símbolo	Unidad	Comentario
1	---	---	*String* `gdb`, "*to facilitate extraction*".
2	i	1	Índice de la molécula en QM9. El formato es sin *leading zeros*: en el primer archivo, i es `1` en lugar de `000001`.
3	A	GHz	Constante rotacional, tal que $A \geq B \geq C$.
4	B	GHz	Constante rotacional.
5	C	GHz	Constante rotacional.
6	$\|\boldsymbol{\mu}\|$	debye	Norma del momento dipolar.
7	α	${a_0}^3$	Polarizabilidad isotrópica.
8	E_{HOMO}	E_{h}	Energía del *highest occupied molecular orbital*.
9	E_{LUMO}	E_{h}	Energía del *lowest unoccupied molecular orbital*.
10	E_{gap}	E_{h}	$\Delta E = E_{\text{LUMO}} - E_{\text{HOMO}}$
11	$\langle R^2 \rangle$	${a_0}^2$	*Electronic spatial extent*.
12	ZPVE	E_{h}	Energía vibracional de punto-cero.
13	U_o	E_{h}	Energía interna a 0 K.
14	U	E_{h}	Energía interna a 298.15 K.
15	H	E_{h}	Entalpía a 298.15 K.
16	G	E_{h}	Energía libre de Gibbs a 298.15 K.
17	C_V	$\text{cal}\,\text{mol}^{-1}\,\text{K}^{-1}$	Capacidad calorífica a 298.15 K.

Notas:

- Denotamos a la unidad de energía hartree como E_{h}, tal que $1\,E_{\text{h}} \approx 27.211\,\text{eV} \approx 627.509\,\text{kcal}\,\text{mol}^{-1}$. Luego, a_0 es el radio de Bohr, y $1\,a_0 \approx 5.291 \times 10^{-11}\,\text{m}$.
- El *electronic spatial extent*, según Maity *et al.* [76], *"...is a measure of spread of electron density on the molecule and it is directly proportional to the electronic volume of the molecule."*. También, a valores mayores de polarizabilidad corresponde un $\langle R^2 \rangle$ mayor.

  Por otra parte, Xu *et al.* [77] le denotan ESE, y le definen de la forma *"...the ESE is computed as the expectation extent of the electron density times the distance from the center of mass of a molecule."*.
- SMILES significa *simplified molecular-input line-entry system*. Es un descriptor ASCII compacto que, en una línea, describe una estructura orgánica. Una sola molécula suele tener varios SMILES válidos. Describimos este lenguaje de manera breve en la página 30.
- InChI significa *International Chemical Identifier* [78]. Este descriptor es más complejo y rico que el SMILES, pues ahora contiene una conectividad molecular más explícita, e incluye también otro tipo de información como las cargas atómicas y la isotopía. Dada su mayor complejidad, los InChI ya no esperan ser completamente legibles por un humano; están pensados para computadoras.

Por ejemplo, la molécula 44 de la base es el óxido de propileno (figura 2.4). A continuación su archivo QM9, `dsgdb9nsd_000044.xyz`, donde los números en color ■ son el ennumerado de renglones, de los cuales el 2 y 13 son demasiado largos y los mostramos en varias líneas. El archivo contiene 15 líneas en total:

```
1  10
2  gdb 44 18.21931 6.63877 5.92459 1.812 35.01 -0.2633 0.1052 0.3685 267.2979
   0.085275 -193.039603 -193.035186 -193.034242 -193.065979 14.764
3  C -0.016463427   1.5183070486  0.0077144513 -0.409135
4  C -0.0274322683  0.0165769416 -0.1088974085  0.120723
5  C  0.7729655287 -0.7013188571 -1.1086311523 -0.170074
6  O  1.1721229292 -0.667029688   0.2627882891 -0.233116
7  H  0.9186598033  1.9291421734 -0.3828587993  0.126852
8  H -0.8515801425  1.9545650629 -0.5510945965  0.121174
9  H -0.1148770196  1.8259647016  1.0539777655  0.129656
10 H -0.9358259801 -0.4735060008  0.2468279306  0.090042
11 H  0.4444392202 -1.6720149484 -1.4781062204  0.11283
12 H  1.405382836  -0.1360033938 -1.7927149096  0.111048
13 211.4037 363.5308 406.0832 780.3093 854.5293 904.4036 979.9458 1041.4346
   1131.726 1158.3827 1164.4085 1186.5228 1299.095 1404.3388 1440.1378 1483.8785
   1497.9999 1532.4905 3035.8464 3078.3191 3089.3709 3100.9517 3122.1826
   3164.3029
14 CC1CO1 C[C@H]1CO1
15 InChI=1S/C3H6O/c1-3-2-4-3/h3H,2H2,1H3 InChI=1S/C3H6O/c1-3-2-4-3/h3H,2H2,1H3/t3-/m0/s1
```

A detalle, sus partes son:

```
1  10
2  gdb 44 18.21931 6.63877 5.92459 1.812 35.01 -0.2633 0.1052 0.3685 267.2979
   0.085275 -193.039603 -193.035186 -193.034242 -193.065979 14.764
3  C -0.016463427   1.5183070486  0.0077144513 -0.409135
4  C -0.0274322683  0.0165769416 -0.1088974085  0.120723
5  C  0.7729655287 -0.7013188571 -1.1086311523 -0.170074
6  O  1.1721229292 -0.667029688   0.2627882891 -0.233116
7  H  0.9186598033  1.9291421734 -0.3828587993  0.126852
8  H -0.8515801425  1.9545650629 -0.5510945965  0.121174
9  H -0.1148770196  1.8259647016  1.0539777655  0.129656
10 H -0.9358259801 -0.4735060008  0.2468279306  0.090042
11 H  0.4444392202 -1.6720149484 -1.4781062204  0.11283
12 H  1.405382836  -0.1360033938 -1.7927149096  0.111048
13 211.4037 363.5308 406.0832 780.3093 854.5293 904.4036 979.9458 1041.4346
   1131.726 1158.3827 1164.4085 1186.5228 1299.095 1404.3388 1440.1378 1483.8785
   1497.9999 1532.4905 3035.8464 3078.3191 3089.3709 3100.9517 3122.1826
   3164.3029
14 CC1CO1 C[C@H]1CO1
15 InChI=1S/C3H6O/c1-3-2-4-3/h3H,2H2,1H3 InChI=1S/C3H6O/c1-3-2-4-3/h3H,2H2,1H3/t3-/m0/s1
```

El `listing` anterior hace evidente la semejanza de los archivos QM9 a un archivo `.xyz` convencional: los `.xyz` tienen en la primer línea la cantidad de átomos, permiten que en la segunda el usuario añada comentarios sobre la molécula, y de la línea 3 en adelante listan, para cada átomo, su símbolo y sus tres coordenadas cartesianas.

Claramente, los archivos QM9 cumplen con los requisitos de un archivo `.xyz`; el resto de información es colocada sin interferir. En el `listing` anterior, la información `xyz` está marcada de color ■.

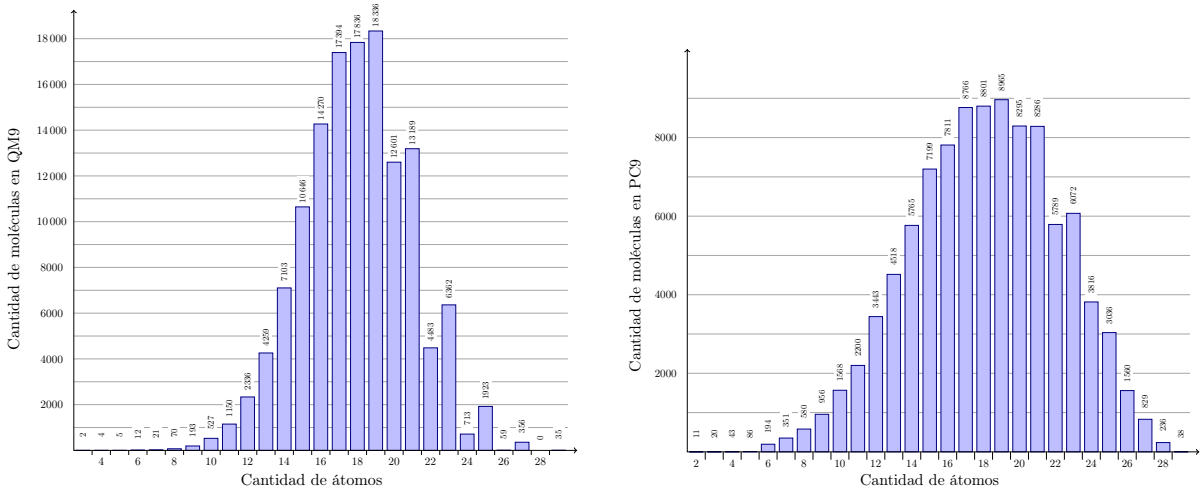
(a) Fórmula taquigráfica. (b) Fórmula desarrollada.

Figura 2.4: Óxido de propileno, molécula 44 de la base de datos QM9 y 6378 de PC9.

El resto de componentes del archivo son:

- , propiedades escalares de la molécula.
- , las cargas parciales de Mulliken.
- , las frecuencias vibracionales.
- , las *strings* SMILES.
- , las *strings* InChI.

La figura 2.5a muestra el conteo atómico de las moléculas QM9.



(a) Base QM9. (b) Base PC9.

Figura 2.5: Histogramas de la cantidad de átomos para las bases QM9 y PC9.

2.4 PC9: una alternativa a QM9

La base PC9 [74, 75] es menos prominente que QM9 (sección 2.3), pero no menos interesante. Al igual que QM9, esta base está compuesta por moléculas orgánicas de ≤ 9 átomos pesados C, N, O, F. Sin embargo, estas bases difieren en el origen de sus moléculas: las de QM9 provienen de la base GDB-17 (sección 2.2), creada desde un enfoque teórico. Por otra parte, PC9 sigue un enfoque experimental: sus moléculas provienen de la base PubChem [79-81], la cual contenía en 2020 a 111 millones de estructuras obtenidas experimentalmente [81]. Luego, el proyecto PubChemQC [82] partió de los

InChI de las moléculas de PubChem, y optimizó a casi tres millones al nivel B3LYP/6-31G(d). Finalmente, PC9 es un subconjunto de esta base.

De esta manera, las estructuras PC9 fueron sintetizadas experimentalmente, en contraste con el proceso teórico y matemáticamente riguroso detrás de GDB-17 y QM9. Como resultado, hay poco traslape entre estas bases: de las 99 234 moléculas PC9, 80 877 no existen en QM9; las moléculas comunes son apenas 18 357.

Los autores de PC9 postulan que su base tiene una mayor diversidad química que QM9. Para esto, presentan los argumentos [74]:

- PC9 presenta una mayor diversidad en distancias interatómicas. Por ejemplo, las distancias N−N en QM9 están concentradas en la región de 130 a 140 pm, mientras que en PC9 la distribución es razonablemente homogénea en el intervalo de 120 a 150 pm. También, las distancias O−N en el rango 135 a 150 pm están bien representadas en ambas bases, pero la región de 120 a 130 pm son una fracción minoritaria de QM9, mientras que en PC9 este intervalo está mejor representado.
- PC9 presenta una mayor diversidad de grupos funcionales: múltiples grupos fueron descartados en la creación de la base GDB-17, y por lo tanto no están presentes en QM9. Por ejemplo, azida, fluoruro de acilo, peróxido, hemiacetal, y nitroso.
- Mientras que todas las moléculas de QM9 son eléctricamente neutras y de capa cerrada, PC9 incluye 4442 radicales y 883 tripletes.
- En general, un modelo ML entrenado con la base QM9, usado para predecir las energías de los orbitales frontera de las moléculas PC9, muestra un desempeño más pobre que el de un modelo entrenado con la base PC9 que predice los orbitales frontera de la base QM9.

Si bien la base PC9 presenta una mayor diversidad química que QM9, su diversidad de propiedades es menor: los archivos PC9 únicamente contienen la geometría del mínimo, su energía total, y las de los niveles HOMO − 1, HOMO, LUMO, LUMO + 1. También, dado que estas moléculas provienen de la base PubChemQC, su nivel de teoría es B3LYP/6-31G(d), menor al de QM9, B3LYP/6-31G(2df,p).

Esta base es gratuita, y puede descargarse del sitio `Figshare` [75].

2.4.1 Un archivo PC9

Los archivos PC9 siguen un formato similar al de QM9:

Renglón	Propiedad
1	Cantidad de átomos, n_{at}.
2	Propiedades escalares (ver tabla siguiente).
3, ..., $n_{\text{at}} + 2$	De un átomo: Símbolo, coordenadas x, y, z (Å), y carga parcial.
$n_{\text{at}} + 3$	Cantidad de átomos pesados, y una secuencia de ceros.
$n_{\text{at}} + 4$	Dos *strings* SMILES.
$n_{\text{at}} + 5$	Descriptor InChI de la estructura.

donde Å denota la unidad ångström, tal que $1\,\text{Å} = 1 \times 10^{-10}\,\text{m}$.

El renglón 2 lista algunas propiedades escalares, así como múltiples ceros, los cuales fueron heredados del formato de los archivos PubChemQC:

Número	Símbolo	Unidad	Comentario
1	---	---	La palabra `PubChemQS`, indicando el origen de los archivos.
2	ID	1	*Compound identification number* de la molécula.
3	$E_{\mathrm{HOMO}-1}$	E_h	Energía del orbital inferior al HOMO.
4	$E_{\mathrm{LUMO}+1}$	E_h	Energía del orbital superior al LUMO.
8	E_HOMO	E_h	Energía del *highest occupied molecular orbital*.
9	E_LUMO	E_h	Energía del *lowest unoccupied molecular orbital*.
10	E_gap	E_h	$\Delta E = E_\mathrm{LUMO} - E_\mathrm{HOMO}$
13	E	E_h	Energía total de la molécula.

Notas:

- Denotamos a la unidad de energía hartree como E_h; $1\,E_\mathrm{h} \approx 27.211\,\mathrm{eV} \approx 627.509\,\mathrm{kcal\,mol^{-1}}$.
- Los archivos PC9 no contienen varias propiedades presentes en QM9, aunque también dan tres propiedades omitidas en QM9: las energías $E_{\mathrm{HOMO}-1}$, $E_{\mathrm{LUMO}+1}$, y la cantidad de átomos pesados.
- Los ID moleculares de la base forman una lista discontinua: Las moléculas de PubChem tienen ID consecutivos, pero los requisitos impuestos por PC9 excluyen a múltiples moléculas, resultando en IDs discontinuos.
- El artículo de PC9 no menciona el método con el que se calcularon las cargas parciales. Hemos escrito a los autores con esta pregunta; no hemos recibido una respuesta.

Por ejemplo, el óxido de propileno (figura 2.4) aparece también en PC9, en el archivo `000006378.xyz`. El renglón 2 es demasiado largo, y tenemos que presentarlo en dos líneas. Este archivo contiene 15 renglones:

```
1  10
2  PubChemQS 6378 -0.2979000000 0.1103000000 0.0 0.0 0.0 -0.2624000000
   0.1058000000 0.3682000000 0.0 0.0 -193.1085113446 0.0 0.0 0.0 0.0
3  C 1.0468809221 -0.0430953618 -0.0259993348 -0.463469
4  C 2.5517486913 -0.1396495157 -0.0252920546  0.111068
5  C 3.2907333080 -0.9157828788 -1.0316774834 -0.109583
6  O 3.2376189938  0.5147043923 -1.1029703840 -0.441777
7  H 0.6105215241 -0.7797711299  0.6593567636  0.149779
8  H 0.6493486073 -0.2206336113 -1.0303926307  0.158050
9  H 0.7235222212  0.9528552202  0.2979876738  0.161543
10 H 3.0314624801  0.0048223090  0.9459634671  0.140355
11 H 2.7404031685 -1.4423016312 -1.8127591892  0.146309
12 H 4.2670494992 -1.3378342518 -0.7901313137  0.147724
13 4.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
14 C[C@H]1CO1 C[C@H]1CO1
15 InChI=1S/C3H6O/c1-3-2-4-3/h3H,2H2,1H3/t3-/m0/s1
```

Podemos ver que el archivo es semejante al formato `xyz` y al de QM9. A detalle, sus partes son:

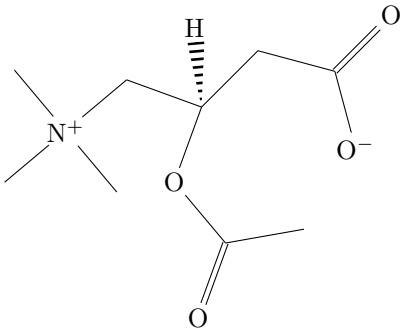
```
1  10
2  PubChemQS 6378 -0.2979000000 0.1103000000 0.0 0.0 0.0 -0.2624000000
   0.1058000000 0.3682000000 0.0 0.0 -193.1085113446 0.0 0.0 0.0 0.0
3  C 1.0468809221 -0.0430953618 -0.0259993348 -0.463469
4  C 2.5517486913 -0.1396495157 -0.0252920546  0.111068
5  C 3.2907333080 -0.9157828788 -1.0316774834 -0.109583
6  O 3.2376189938  0.5147043923 -1.1029703840 -0.441777
7  H 0.6105215241 -0.7797711299  0.6593567636  0.149779
8  H 0.6493486073 -0.2206336113 -1.0303926307  0.158050
9  H 0.7235222212  0.9528552202  0.2979876738  0.161543
10 H 3.0314624801  0.0048223090  0.9459634671  0.140355
11 H 2.7404031685 -1.4423016312 -1.8127591892  0.146309
12 H 4.2670494992 -1.3378342518 -0.7901313137  0.147724
13 4.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
14 C[C@H]1CO1 C[C@H]1CO1
15 InChI=1S/C3H6O/c1-3-2-4-3/h3H,2H2,1H3/t3-/m0/s1
```

La codificación de colores es la misma que la usada para el archivo QM9:

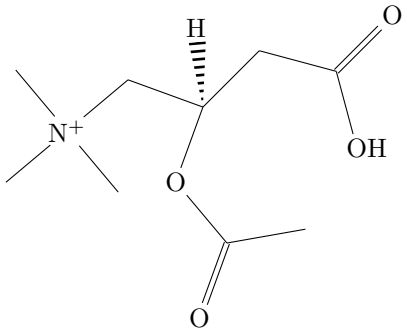
- , información parte del formato `xyz`.
- , propiedades escalares de la molécula.
- , cargas parciales.
- , *strings* SMILES.
- , *string* InChI.

Las discontinuidades en los índices moleculares son un distintivo de PC9. Por ejemplo, el óxido de propileno tiene ID `6378`, pero es la molécula 313 de la base PC9. A fin de ilustrar este fenómeno, mostramos las primeras cuatro moléculas de PubChem en la figura 2.6. Todos estos átomos son válidos dentro del espacio PC9, pero las primeras tres moléculas tienen demasiados átomos pesados (respectivamente 14, 14, y 11), y solo la 4 cumple con este criterio. Las moléculas 5 hasta la 21 tampoco existen dentro de PC9; la molécula 22 de PubChem es apenas la segunda en PC9.

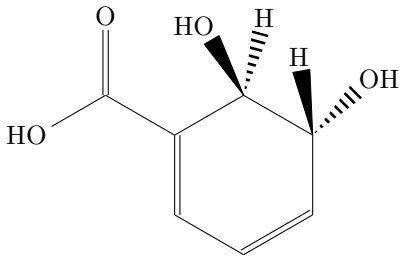
La figura 2.5b muestra el conteo atómico de las moléculas PC9.



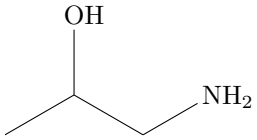
(a) Acetil-L-carnitina, molécula 1 de PubChem.



(b) *1-Propanaminium, 2-(acetyloxy)-3-carboxy-N,N,N-trimethyl-, inner salt*, molécula 2 de PubChem. Es la forma hidrogenada de la acetil-L-carnitina.



(c) Ácido 5,6-dihidroxiciclohexa-1,3-dieno-1-carboxílico, molécula 3 de PubChem.



(d) 1-Aminopropan-2-ol, molécula 4 de PubChem.

Figura 2.6: Moléculas 1 a 4 de PubChem. Las 1 y 2 tienen 14 átomos pesados, y la 3 tiene 11. Solo la 4 forma parte de PC9, dado que tiene cinco átomos pesados, y todos sus átomos $\in \{\mathrm{H}, \mathrm{C}, \mathrm{N}, \mathrm{O}, \mathrm{F}\}$.

Capítulo 3

Descriptores moleculares

Índice de capítulo

3.1 Introducción

Una vez que hemos elegido una base de datos que representa al espacio químico a explorar, nuestra siguiente decisión es la representación molecular. Debemos elegir un descriptor compatible con todas las moléculas de nuestra base.

Esta decisión debe satisfacer consideraciones de carácter químico y matemático-computacional: A nivel matemático-computacional, buscamos que la representación esté compuesta, fundamentalmente, de elementos que se puedan representar con un tipo de dato intrínseco de los lenguajes de programación, como los números reales que se representan aproximadamente mediante los `Float64`, o variables binarias que se representan mediante los `Bool`. También, la dimensionalidad de la representación debe ser tal que los experimentos tengan un costo computacional aceptable, tanto en tiempo de cálculo como en uso de memoria.

Por otra parte, tenemos las consideraciones químicas. Al cumplirlas, los experimentos se mantienen fieles a la química que rodea al problema, y son consistentes con la realidad, y sus resultados se vuelven más interpretables. Por ejemplo, si nuestro objetivo es realizar predicciones energéticas o termodinámicas de moléculas aisladas, la estereoquímica de los centros quirales es indistinta. Por lo tanto, esperamos que el descriptor considere iguales a un par de enantiómeros: si las representaciones

fueran numéricamente distintas, entonces las predicciones del modelo, que también son numéricas, serán distintas. Este es un ejemplo de discordancia entre la química de un problema y la maquinaria ML usada para abordarlo.

También, los modelos ML más habituales son de arquitectura estática. Esto es, un modelo entrenado para, digamos, representaciones de 40 elementos, es incompatible con representaciones de 41 elementos, incluso si este nuevo elemento tuviera un significado químico cercano al de los demás; sería necesario entrenar a un modelo nuevo para poder trabajar con esta representación. Por lo tanto, el tamaño del descriptor debe ser constante para todas las moléculas de nuestra base de datos. Esto descalifica a algunos descriptores convencionales como las coordenadas cartesianas, cuyo tamaño escala según el tamaño del sistema.

Algunos descriptores conceptualmente sencillos que cumplen con estos requisitos son el conteo de átomos por tipo (la composición atómica, sección 3.7), o el de enlaces. Algunas representaciones más complicadas son los índices de Kier y Hall [83] de conectividad molecular ${}^{m}\chi_{t}$ y de forma ${}^{m}\kappa$, los cuales provienen de la teoría de grafos química, y fueron originalmente concebidos como variables para modelos QSAR.

Es valioso el conocer el carácter preciso de la relación entre las entradas y las salidas que elegimos, puesto que de esta manera podemos realizar una mejor interpretación de los resultados del experimento. Esto, sin embargo, no es un requisito indispensable de los experimentos. Basta con informadamente sospechar que la representación molecular elegida está relacionada a las propiedades objetivo. Luego, podríamos elucidar el carácter del vínculo entre las entradas y las salidas a partir del análisis de los resultados del experimento.

Podemos considerar que un descriptor que cumple con los requisitos mencionados anteriormente es un *buen descriptor*. Behler [84] aborda este tema de forma didáctica, y propone una serie de requisitos a cumplir, que veremos a detalle en la siguiente sección.

3.2 Propiedades de un buen descriptor

El artículo de Behler y Parrinello [85] propone una serie de funciones simetrizantes como descriptor para experimentos ML aplicados a sistemas químicos. En particular, se muestra el desempeño de esta representación para la predicción de energías de dinámicas moleculares de silicio en bulto. Este método, que posteriormente sería nombrado *high-dimensional neural network potential* (HDNNP, sección 3.4), muestra resultados cercanos a los del método de referencia DFT, mientras que otros métodos comparables quedan más lejanos, como los potenciales de Bazant, Lenosky, o de Tersoff. Además, el método HDNNP fue más veloz que el DFT por cerca de 5 órdenes de magnitud, para el sistema de 64 átomos que se estudió.

En un artículo posterior, Behler [84] añade otras tres funciones de simetrización, y les denomina G_i^j, donde $j = \{1, 2, \ldots, 5\}$. Con estas cinco familias, un investigador puede definir una serie de funciones que describen el entorno radial y angular de los diferentes átomos en un sistema. Le llamaremos *atomic environment vector* (AEV, sección 3.4) a este arreglo de funciones con distintos coeficientes y formas funcionales. Tratamos estas funciones con más detalle en la sección 3.5.

Behler también expone las características que estas funciones de simetrización cumplen, y que las hacen una opción sensata para estudios ML. Podemos extender el criterio de Behler, y considerar que cumplir con este conjunto de requisitos es una condición necesaria, pero no suficiente, para que un descriptor sea considerado válido:

- *ser continuo*, un requisito crucial para la compatibilidad de las funciones con las redes neuronales.
- *ser diferenciable*. Este es un requisito de carácter químico: el cálculo de fuerzas atómicas requiere la diferenciabilidad del descriptor.

- *tener derivada de cálculo asequible*. Esto permite implementar los descriptores en programas rápidos.
- *decaer hacia* 0 *para distancias interatómicas grandes*. Si las funciones de simetrización cumplen con este último punto, entonces su comportamiento es semejante al de las interacciones atómicas: decrecen a distancias mayores. Este requisito es de carácter químico, y cumplirlo reduce la dificultad del entrenamiento de la red.
- *ser de tamaño constante, independientemente de*:
  - *el tamaño del sistema*, para lograr un predictor extendible a diferentes sistemas,
  - *el número de coordinación de cada átomo*. Este requisito permite que la NN pueda predecir reacciones y experimentos de dinámica molecular. En ambos casos, el número de coordinación de los átomos cambia a lo largo del tiempo.
- *y el descriptor debe ser invariante ante*:
  - traslaciones,
  - rotaciones,
  - intercambio de átomos en el listado de coordenadas,

  dado que estas acciones no modifican las propiedades fisicoquímicas moleculares, pero pueden modificar algunas representaciones como la cartesiana.

3.3 Diacuoaluminio(III): álgebra contra NN

Hay precursores a Behler y Parrinello que han estudiado los problemas y sistemas químicos desde el punto de vista del aprendizaje de máquina. Un ejemplo interesante es el de Gassner *et al.* [27], quienes en 1998 buscaron la predicción de energías para el diacuoaluminio(III), $[Al(H_2O)_2]^{+3}$, mediante redes neuronales.

El grupo de Gassner se enfrentó al mismo problema que describimos anteriormente. Este sistema químico debe representarse de forma numérica para que la red pueda realizar predicciones. Como ya comentamos, la representación cartesiana no es viable, dado que no es invariante ante procesos como la rotación o traslación del sistema.

La solución de Gassner es ingeniosa, y la tratamos a detalle en la sección 3.3.1. En resumen, ellos parten de su sistema, $[Al(H_2O)_2]^{+3}$, y cuidadosamente construyen un arreglo de coordenadas internas que pueda representar a cualquier geometría de este sistema. Bajo este esquema, cualquier geometría tiene *una sola* representación, y queda resuelto el problema fundamental de la representación cartesiana que mencionamos anteriormente.

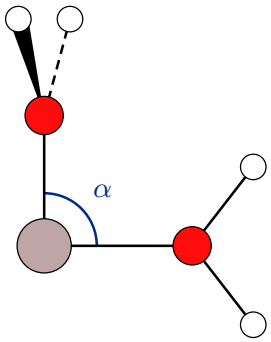
El sistema $[Al(H_2O)_2]^{+3}$ es particular, dado que la interacción Al^{+3}–H_2O es tan fuerte que modelar al sistema únicamente mediante interacciones por pares resulta en predicciones inexactas. La decisión de la forma de la función a usar es de la mayor importancia, dado que en caso de equivocarnos, el modelo estaría condenado a tener un mal desempeño sin importar el tamaño de nuestra base de datos o cuánto tiempo de cálculo se dedique al ajuste numérico. Por otra parte, si usamos redes neuronales para modelar el sistema, la flexibilidad intrínseca de éstas es una respuesta natural al problema de no conocer el carácter verdadero de las interacciones en el sistema; el problema de la construcción del potencial se delega a la red, y confiamos en que expandir la base de datos o aumentar su nivel de cálculo mejore la calidad del modelo.

Gassner considera dos tipos de conformaciones, mostradas en la figura 3.1. Sus propiedades son: en el tipo 1,

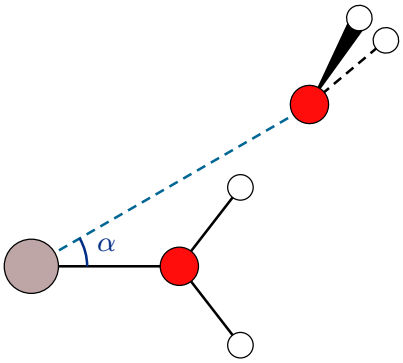
- ambas aguas forman parte de la primer esfera de solvatación del Al^{+3},
- ambas aguas orientan su oxígeno al aluminio,
- por lo tanto, la energía está dada principalmente por el ángulo O−Al−O y las distancias Al−O,
- la interacción de tres cuerpos es repulsiva.

mientras que en el tipo 2,

- un agua pertenece a la primera esfera de solvatación, la otra a la segunda,
- el oxígeno interno apunta hacia el aluminio; el externo puede apuntar hacia un hidrógeno interno o hacia el aluminio,
- por lo tanto, la energía depende también de las distancias H−H y O−H,
- la interacción de tres cuerpos es atractiva cuando se forma un puente de hidrógeno.



(a) Tipo 1. $\alpha > 45°$, $r_{\mathrm{Al-O_{(1,2)}}} \in [1.7\,\text{Å}, 2.5\,\text{Å}]$.



(b) Tipo 2. $\alpha < 45°$, $r_{\mathrm{Al-O_{(1)}}} = 1.8\,\text{Å}$, $r_{\mathrm{Al-O_{(2)}}} \in [2\,\text{Å}, 5.3\,\text{Å}]$.

Figura 3.1: Conformaciones presentes en el diacuoaluminio(III), $[Al(H_2O)_2]^{+3}$, sistema modelado por Gassner *et al.* [27] y Bakker *et al.* [86].

Este conjunto de diferencias en la física de las interacciones dificulta la concepción de aproximaciones algebraicas a la energía que sean versátiles. Por ejemplo, el estudio de Bakker *et al.* [86] trabaja con sistemas $[Al(H_2O)_2]^{+3}$ (en conformaciones de tipo 1 y 2) y $[Al(H_2O)_6]^{+3}$ (con simetría $\boldsymbol{S_4}$). Ellos calculan la energía del sistema sumando las energías de las interacciones de un, dos, tres, ..., n cuerpos. Uno de sus resultados es una expresión algebraica para la energía de tres cuerpos del diacuoaluminio(III):

$$E^{(3)} = 74.85 \cdot \left[0.06413 + (\pi - \alpha)^2\right] \exp\left[-0.2465\left(r^2_{\mathrm{Al-O_{(1)}}} + r^2_{\mathrm{Al-O_{(2)}}}\right)\right] \tag{3.1}$$

donde las r son las distancias en Å entre el aluminio y los oxígenos, α es el ángulo O−Al−O en radianes, y la energía tiene unidades $\mathrm{kJ\,mol^{-1}}$.

Notemos que esta expresión asume que la interacción de tres cuerpos siempre es repulsiva, y por lo tanto no es compatible con geometrías del tipo 2. Según explican los autores:

In contrast to the type 1 configurations, the type 2 triplets show a stronger dependence on the orientation of the dipole moment of both *water molecules. Expressing three-body energies in an analytical form thus needs more parameters than the ion−water distances and the O−Al−O angle only.*

Por otra parte, el modelo de NN de Gassner no tiene una forma funcional rígida, y es compatible con ambos tipos de geometrías.

3.3.1 Funciones de simetrización de Gassner

Una opción conceptualmente sencilla para representar al $[Al(H_2O)_2]^{+3}$ es transformar sus coordenadas cartesianas a una matriz Z, un arreglo de coordenadas internas de distancias, ángulos, y diedros. Esta representación es invariante ante la rotación y traslación, pero no es invariante ante el reordenamiento de átomos en la lista de coordenadas.

La solución presentada por Gassner es el uso de funciones de simetrización, un proceso minucioso que considera todos los tipos de distancia interatómica en el sistema. En este caso, tenemos O−O, Al−O, Al−H, O−H, H−H.

La distancia más sencilla es O−O, puesto que es una sola, y se puede representar directamente.

Luego, hay dos distancias Al−O. Se construyen las métricas:

$$r^{(1)}_{\mathrm{Al-O}} = \left| r_{\mathrm{Al-O}_{(1)}} + r_{\mathrm{Al-O}_{(2)}} \right| \qquad r^{(2)}_{\mathrm{Al-O}} = \left| r_{\mathrm{Al-O}_{(1)}} - r_{\mathrm{Al-O}_{(2)}} \right| \tag{3.2}$$

donde $O_{(i)}$ denota al átomo de oxígeno de la molécula i. Notemos que estas distancias cumplen *todas* las invarianzas anteriormente mencionadas.

Luego, hay cuatro distancias Al−H y O−H. Mostramos la primera de ambas series,

$$r^{(1)}_{\mathrm{Al-H}} = \left| r_{\mathrm{Al-H}_{(11)}} + r_{\mathrm{Al-H}_{(12)}} \right| + \left| r_{\mathrm{Al-H}_{(21)}} + r_{\mathrm{Al-H}_{(22)}} \right| \tag{3.3}$$

$$r^{(1)}_{\mathrm{O-H}} = \left| r_{\mathrm{O}_{(1)}-\mathrm{H}_{(21)}} + r_{\mathrm{O}_{(1)}-\mathrm{H}_{(22)}} \right| + \left| r_{\mathrm{O}_{(2)}-\mathrm{H}_{(11)}} + r_{\mathrm{O}_{(2)}-\mathrm{H}_{(12)}} \right| \tag{3.4}$$

donde denotamos $H_{(12)}$ al hidrógeno 1 del agua 2.

Las distancias H−H se definen mediante cuatro valores. Dado que éstas consideran a los cuatro hidrógenos, las expresiones son más complicadas. A continuación, la primera. $r^{(4)}_{\mathrm{H-H}}$ contiene la misma cantidad de términos, mientras que $r^{(2)}_{\mathrm{H-H}}$ y $r^{(3)}_{\mathrm{H-H}}$ son del doble de tamaño,

$$\begin{aligned} r^{(1)}_{\mathrm{H-H}} = &\left(\left| r_{\mathrm{H}_{(22)}-\mathrm{H}_{(21)}} + r_{\mathrm{H}_{(22)}-\mathrm{H}_{(11)}} \right| + \left| r_{\mathrm{H}_{(21)}-\mathrm{H}_{(11)}} + r_{\mathrm{H}_{(21)}-\mathrm{H}_{(12)}} \right| \right) \\ &+ \left(\left| r_{\mathrm{H}_{(22)}-\mathrm{H}_{(12)}} + r_{\mathrm{H}_{(21)}-\mathrm{H}_{(12)}} \right| + \left| r_{\mathrm{H}_{(21)}-\mathrm{H}_{(11)}} + r_{\mathrm{H}_{(22)}-\mathrm{H}_{(11)}} \right| \right) \end{aligned} \tag{3.5}$$

Podemos ver que, si bien las funciones de simetrización de Gassner son una interfaz adecuada entre las NN y el sistema $[M(H_2O)_2]^{+n}$, su fundamentación es delicada, y su extensibilidad es difícil, dado que intentar trasladar la técnica a un sistema molecular distinto requiere definir un nuevo conjunto de funciones, cuya cantidad y complejidad se incrementan rápidamente al añadir átomos, especialmente si se introducen átomos nuevos en el sistema, dado que esto resulta en nuevos tipos de distancias interatómicas.

Habiendo cubierto este antecesor, podemos tratar a las funciones de Behler-Parrinello. Solo que, antes de hacerlo, veremos la técnica HDNNP, que es el entorno en el cual se formularon las funciones de Behler, y en el cual éstas son usadas para modelar las interacciones interatómicas.

3.4 *High-dimensional neural network potential*

Un componente integral de la química computacional es la formulación y desarrollo de modelos aproximados cuyo menor nivel de teoría permita tratar sistemas más grandes que los compatibles con métodos más rigurosos y computacionalmente costosos. Ejemplos usuales son los modelos de campos de fuerza clásicos y métodos semiempíricos.

Un método reciente, nacido en la intersección del aprendizaje de máquina con la química, es el *high-dimensional neural network potential*, o HDNNP, aquí denominado NNP [84, 85]. Este método modela sistemas químicos a partir de funciones de simetrización, que son análogas a los campos de fuerza clásicos y sus funciones cuadráticas, o los métodos semiempíricos y sus hamiltonianos con algunos elementos constantes o calculados mediante álgebra en lugar de integraciones numéricas.

Esta técnica fue originalmente planteada para describir dinámicas moleculares de silicio en bulto [85], pero la familia de potenciales ANI ha adaptado esta metodología a la predicción de energías de moléculas orgánicas discretas [29, 87-89], alcanzando desempeños a la par que métodos semejantes. Por ejemplo, ANI-1 [29], el primer modelo, usa como referencia energías y estructuras calculadas al nivel de teoría ωB97X/6-31G(d); sus predicciones son más cercanas a los valores de referencia que los de *Density-Functional Based Tight Binding* (DFTB), y que los semiempíricos AM1 y PM6 [90, 91]. Además, el costo computacional de los potenciales ANI es comparable a los campos de fuerza clásicos.

Mostramos el funcionamiento de un NNP en la figura 3.2b. Damos a continuación una descripción de sus pasos:

1. Recibir las coordenadas atómicas.
2. Para cada átomo, computar, con diferentes funciones simetrizantes G, los vectores de entorno atómico (*atomic environment vector*, AEV).

   Los AEV son un conjunto de funciones simetrizantes evaluadas para el átomo i (el átomo de interés), tomando en cuenta la cercanía y arreglo de sus vecinos (el resto de átomos).

   En la figura 3.2b, este es el paso de la primer a la segunda columna. Notemos que cada átomo tiene una sola flecha negra hacia su propio AEV; el resto de átomos participa en ese AEV formando parte del entorno (flecha café). Es convencional que el AEV total tenga un bloque para las interacciones entre el átomo i y los átomos de hidrógeno, luego una sección para el átomo i y los átomos de carbono, ...

   Dado que, en general, un archivo de coordenadas posee únicamente la identidad del átomo, pero no su carga ni su hibridación, los NNP se suelen entrenar únicamente distinguiendo entre átomos por su símbolo. Por lo tanto, la construcción de un NNP depende (y necesita) que el conjunto de entrenamiento sea amplio y tenga una riqueza muestral, y así el NNP podrá predecir energías de átomos en entornos químicos diversos.
3. Con los AEV como características, cada NNP calcula energías atómicas.

   Los AEV son la capa de entrada para las redes neuronales. Cada uno de los múltiples elementos del AEV pasan a las neuronas de la primer capa oculta de la misma forma que en cualquier red neuronal (sección 1.6).

   Cada NNP se usa según haya átomos de ese tipo en el sistema. En general, cada NNP tiene parámetros y arquitectura diferentes. En el caso de la figura 3.2b, el NNP que calcula las energías de ambos hidrógenos es el mismo. La única diferencia es que, dado que los elementos de $\boldsymbol{G}_{\mathrm{H}}$ son diferentes a los de $\boldsymbol{G}_{\mathrm{H}}$, entonces las energías E_{H} y E_{H} son distintas.

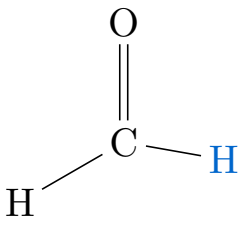
   Siguiendo la notación de la sección 1.6, podemos expresar la energía de un átomo de manera análoga a, por ejemplo, la ecuación 1.15. Si tenemos un NNP con dos capas ocultas, la energía

del átomo i es

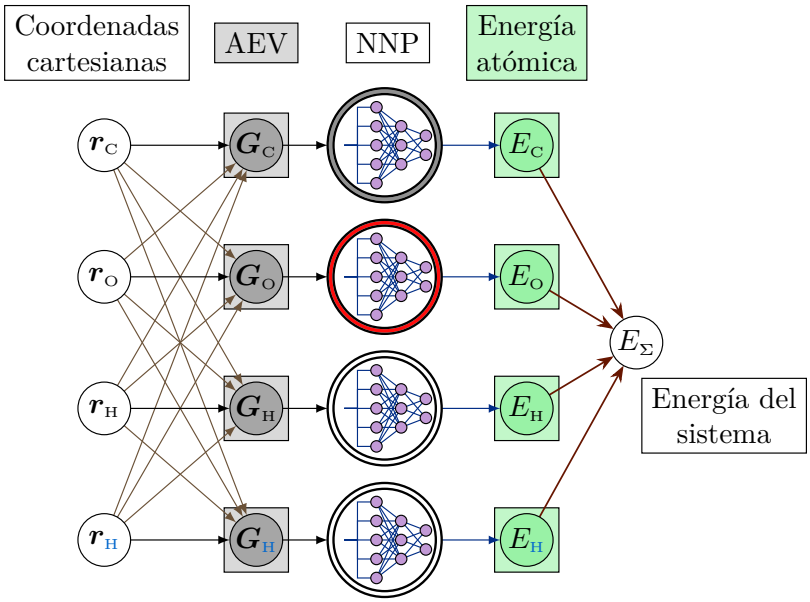
$$E_i = f\left(b_1^{(3)} + \sum_j w_{1j}^{(3)} f\left(b_j^{(2)} \sum_k w_{jk}^{(2)} f\left(b_k^{(1)} + \sum_p w_{kp}^{(1)} G_p\right)\right)\right) \tag{3.6}$$

4. Sumar todas las energías atómicas.

   El resultado es la energía del sistema. Este paso se representa en la figura 3.2b con las flechas que llegan a la energía total.



(a) Molécula de formaldehído. Su simetría es $\boldsymbol{C_s}$ en lugar de $\boldsymbol{C_{2v}}$ dado el desplazamiento del segundo hidrógeno. Se colora a los hidrógenos de diferente manera para distinguirlos en la siguiente figura.



(b) Funcionamiento de un NNP. El esquema de colores para las capas de entrada y salida es consistente con el de las figuras 1.1 y 1.3. Cada vector de las diferentes funciones G es un *atomic environment vector*, AEV. Las flechas que conectan las energías atómicas con la energía del sistema indican la suma de energías.

Figura 3.2: Molécula de formaldehído y predicción de su energía mediante un NNP. Su cálculo emplea tres NNP distintos (codificados por su color de borde), uno para cada tipo de átomo, cada uno con sus parámetros y arquitectura particulares.

3.5 Funciones de simetrización de Behler

Ahora veremos las funciones simetrizantes G que constituyen a los AEV. Para éstas, hemos de considerar a un átomo de interés i, a partir del cual consideraremos todos los pares atómicos (i, j), para las funciones radiales, y todas las tríadas (i, j, k), para las funciones angulares.

Hay un componente común en todas las funciones G de Behler, la función de corte f_c:

$$f_c(R_{ij}) = \begin{cases} 0.5\left[\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1\right] & \text{si } R_{ij} \leq R_c \\ 0 & \text{si } R_{ij} > R_c \end{cases} \tag{3.7}$$

donde R_c es el radio de corte. El aspecto de esta función se muestra en la figura 3.3. Es claro que f_c y su derivada son 0 cuando $R_{ij} > R_c$; todas las funciones G involucran una multiplicación por $f_c(R_{ij})$. La consecuencia de esto es que las G adquieren esta propiedad de f_c, y entonces la forma de las G es químicamente realista para $R_{ij} > R_c$, y entonces el NNP solo considera en su predicción energética las interacciones atómicas a distancias $< R_c$.

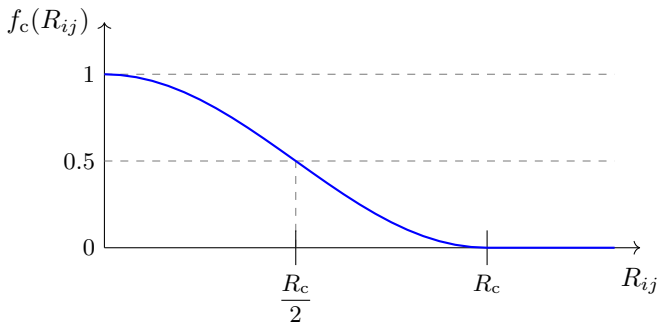


Figura 3.3: f_c, función de corte de Behler (ecuación 3.7).

3.5.1 Funciones radiales

Behler [84] propone tres tipos:

$$G_i^1 = \sum_{\substack{j=1 \\ j\neq i}}^{n_{at}} f_c(R_{ij})$$

$$G_i^2 = \sum_{\substack{j=1 \\ j\neq i}}^{n_{at}} \exp[-\eta(R_{ij} - R_s)^2] \cdot f_c(R_{ij})$$

$$G_i^3 = \sum_{\substack{j=1 \\ j\neq i}}^{n_{at}} \cos(\kappa R_{ij}) \cdot f_c(R_{ij})$$

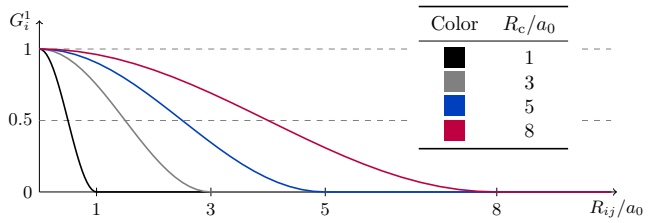
donde n_{at} es la cantidad de átomos en la molécula. Estas funciones son sumas sobre todos los vecinos de i, y proporcionan información sobre el número de coordinación de i y su conectividad a diferentes distancias. G_i^2 (figura 3.4b, 3.4c) es quizás el ejemplo más claro al respecto, dado que se pueden elegir múltiples valores de R_s y así explorar el entorno radial de i. G_i^3 (figura 3.4d) son cosenos amortiguados, cuyo período está dado por κ, y el entorno de i se puede explorar mediante distintos valores de κ.

3.5.2 Funciones angulares

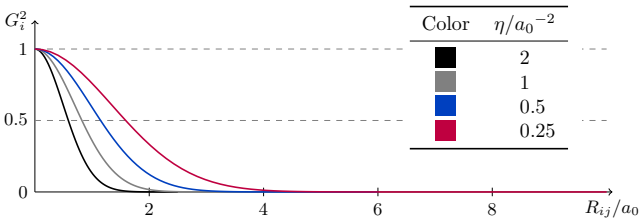
Las funciones angulares proporcionan información acerca del arreglo de tríadas de átomos. Behler [84] propone sumas de cosenos de θ_{ijk}, tal que

$$\theta_{ijk} = \arccos\left(\frac{\boldsymbol{R_{ij}} \cdot \boldsymbol{R_{ik}}}{R_{ij} \cdot R_{ik}}\right) \tag{3.8}$$

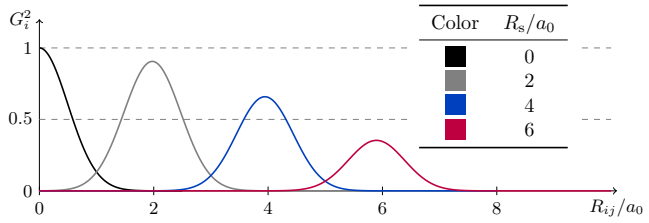
donde se mide el ángulo del átomo i con sus vecinos j y k. $\boldsymbol{R_{il}}$ es el vector que une al átomo i con el l.



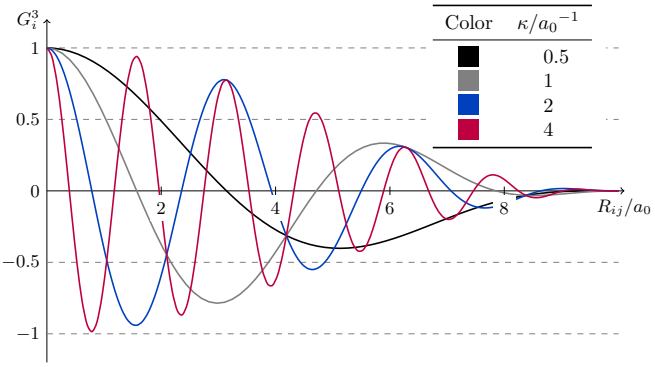
(a) G_i^1. Dado que el sistema atómico de referencia es de dos átomos, la función graficada es la misma de la figura 3.3. Aquí se muestra el efecto de diferentes R_c.



(b) G_i^2 con $R_s = 0\,a_0$, $\eta > 0\,a_0^{-2}$.



(c) G_i^2 con $R_s \geq 0\,a_0$, $\eta = 2\,a_0^{-2}$.



(d) G_i^3. Esta es la única función radial con valores < 0.

Figura 3.4: Adaptación de la figura 3 de [84]. Funciones radiales simetrizantes $G_i^{1,2,3}$ para un sistema de dos átomos. $R_c = 10\,a_0 \approx 5.29\,\text{Å}$.

Las dos funciones propuestas por Behler son:

$$G_i^4 = 2^{1-\zeta} \sum_{\substack{j=1 \\ j\neq i}}^{n_{at}-1} \sum_{\substack{k=j+1 \\ k\neq i}}^{n_{at}} (1 + \lambda \cos\theta_{ijk})^\zeta \cdot \exp[-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)] \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}) \tag{3.9}$$

$$G_i^5 = 2^{1-\zeta} \sum_{\substack{j=1 \\ j\neq i}}^{n_{at}-1} \sum_{\substack{k=j+1 \\ k\neq i}}^{n_{at}} (1 + \lambda \cos\theta_{ijk})^\zeta \cdot \exp[-\eta(R_{ij}^2 + R_{ik}^2)] \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \tag{3.10}$$

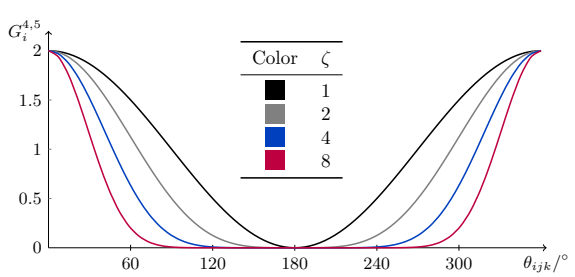
donde n_{at} es la cantidad de átomos en la molécula.

Una vez más, las funciones deben preservar la simetría química. En este caso, para una tríada de átomos, un $\theta_{ijk} = 179°$ es equivalente a $\theta_{ijk} = 181°$. Esto restringe los valores posibles para λ a 1 y -1; esto es visible en la figura 3.5. Por otra parte, la anchura de las funciones depende de ζ: A valores pequeños, la función se vuelve más difusa.

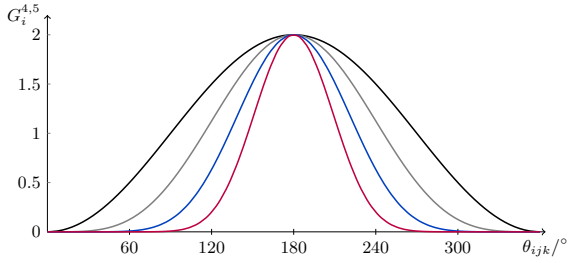
Podemos notar que la parte angular es idéntica para ambas funciones, pero sus partes radiales difieren. El efecto de estas diferencias se muestra en la figura 3.6: G_i^4 es ≈ 0 para más ángulos que G_i^5: esto se debe al factor adicional $f_c(R_{jk})$ de G_i^4. El máximo de ambas funciones decae rápidamente para incrementos leves de distancia: para un $R_c = 10\,a_0 \approx 5.29\,\text{Å}$, incrementos del orden de $0.2\,\text{Å}$ dividen el valor de la función a la mitad.

De manera análoga a las funciones radiales, se puede definir una colección de funciones con

diversos valores (λ, ζ, R_c) para muestrear la abundancia de ángulos en el entorno del átomo i.

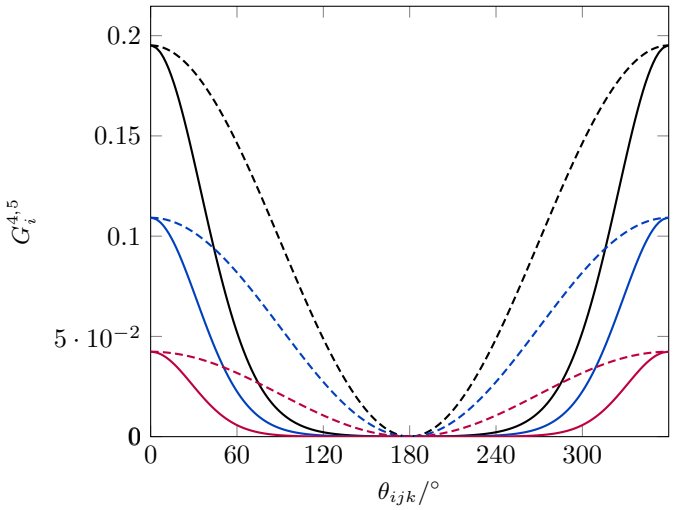


(a) Caso $\lambda = 1$.



(b) Caso $\lambda = -1$. Las ζ y su coloración son iguales a las de la subfigura anterior.

Figura 3.5: Adaptación de la figura 4 de [84]. Las funciones G_i^4 y G_i^5 son idénticas al haber fijado sus partes radiales en 1.



Tipo de línea	Función
——	G_i^4
- - - - -	G_i^5

Color	$R_{ij} = R_{ik}$/Å	Distancia de
■	1.204	HC≡CH
■	1.344	$H_2C{=}CH_2$
■	1.547	$H_3C{-}CH_3$

Figura 3.6: Modificación de la figura 6 de [84]: producto de parte radial y angular de las funciones angulares simetrizantes $G_i^{4,5}$. $R_c = 10\,a_0 \approx 5.29$ Å, $\eta = 2\,{a_0}^{-2}$, $\lambda = 1$. Distancias de enlace tomadas de Pauling y Brockway [92].

La familia de potenciales ANI [29, 87-89] define sus AEV a partir de la función radial G_i^2, y una función angular G_i^5 modificada:

$$G_i^{5\text{mod}} = 2^{1-\zeta} \sum_{\substack{j=1 \\ j\neq i}}^{n_{\text{at}}-1} \sum_{\substack{k=j+1 \\ k\neq i}}^{n_{\text{at}}} [1 + \lambda \cos(\theta_{ijk} - \theta_s)]^\zeta \cdot \exp\left[-\eta \left(\frac{R_{ij} + R_{ik}}{2} - R_s\right)^2\right] \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \tag{3.11}$$

3.6 Matriz de Coulomb ordenada y eigenvalores

Hemos visto a lo largo de este capítulo (secciones 3.2–3.5) ejemplos de descriptores que codifican sistemas químicos para su uso en redes neuronales o modelos ML en general. Ahora veremos una

familia más, la matriz de Coulomb ordenada (*sorted Coulomb matrix*, SCM) y sus eigenvalores (EV), cuya aplicabilidad es versátil, comparable a la de las funciones de Behler-Parrinello. Sin embargo, son más intuitivas a nivel conceptual, dado que codifican la geometría de manera más simple; y a nivel de fundamentación, dado que carecen de parámetros que el usuario o investigador tenga que decidir antes de poder realizar experimentos.

Mientras que las funciones de Behler están enfocadas a la predicción de energías y modelado de superficies de energía potencial, estos descriptores fueron concebidos para la predicción de propiedades termodinámicas de moléculas orgánicas. Por lo tanto, estos son los descriptores que usaremos en los experimentos del capítulo 6.

Estas representaciones cumplen con casi todos los requisitos planteados por Behler (sección 3.2), con la debatible excepción de ser invariantes ante el tamaño del sistema (su cantidad de átomos): el tamaño del descriptor para una sola molécula, fuera del contexto de algún conjunto de moléculas o base de datos, depende directamente del tamaño del sistema. Sin embargo, una vez que elegimos una base de datos, queda también establecida una máxima cantidad de átomos: de esta manera, todos los descriptores pueden dimensionarse al mismo tamaño, independientemente del tamaño de cada molécula.

3.6.1 Construcción de la matriz de Coulomb

Antes de tratar a la matriz ordenada, hemos de hablar de la *matriz de Coulomb* [93, 94] (CM), un arreglo de rango 2 que codifica la geometría y estequiometría molecular:

$$C_{ij} = \begin{cases} \frac{1}{2} Z_i^{2.4} & \text{si } i = j \\ \frac{Z_i Z_j}{\|\boldsymbol{R}_i - \boldsymbol{R}_j\|} & \text{si } i \neq j \end{cases} \tag{3.12}$$

donde Z_k es el número atómico del átomo k; $\boldsymbol{R_k}$ son sus coordenadas cartesianas en bohr (a_0).

En el contexto QM9, hay 35 moléculas con 29 átomos, las más grandes en toda la base de datos. Por lo tanto, un estudio ML que describa a todas las moléculas QM9 a partir de la matriz de Coulomb, trabajará con matrices 29×29, donde moléculas de tamaño < 29 definirán sus valores en la esquina superior izquierda de una matriz 29×29; el resto de elementos serán cero.

Notemos que la representación CM es efectivamente un listado de coordenadas internas, y es invariante ante la rotación o traslación del sistema. Sin embargo, no es así ante el reetiquetado de átomos en la lista de coordenadas. Además, no existe algún criterio químico que pueda, en el caso general, establecer un orden inambiguo en los átomos de una lista de coordenadas. Por lo tanto, la misma estructura tiene múltiples CM posibles.

Las dos representaciones siguientes (EV y SCM) resuelven esta objeción. Hay, sin embargo, una más, que no tiene solución dentro de los derivados de la matriz de Coulomb: el significado físico de los descriptores tiende a cambiar de una molécula a otra: incluso si existiera un 'orden natural' para ordenar los átomos en un listado de coordenadas, como pasa en la matriz de Coulomb ordenada, los átomos 1 de dos moléculas distintas pueden tener entornos químicos, e incluso identidades, completamente distintas. Por otra parte, un AEV de funciones de Behler puede tener un significado concreto para todas las moléculas de una base: por ejemplo, el elemento 1 puede describir el entorno radial cercano de los átomos de carbono, mientras que el elemento 2 describe al entorno lejano.

A pesar de estos señalamientos de inconsistencia, las representaciones derivadas de la matriz de Coulomb son herramientas muy socorridas en los estudios ML, dado que codifican la geometría molecular de manera conveniente, y obtienen resultados aceptables:

- los eigenvalores de la matriz ordenada de Coulomb (sección 3.6.3) tienen, para la predicción de energías de atomización, un mejor desempeño que otros métodos establecidos como el *bond counting* de Benson [95] o el método semiempírico PM6 [91].

- la matriz ordenada de Coulomb y matrices de Coulomb ordenadas aleatoriamente (sección 3.6.2) obtienen resultados mejores que la representación de eigenvalores [96].

3.6.2 Matriz ordenada de Coulomb

Cronológicamente, la matriz ordenada de Coulomb (*sorted Coulomb matrix*, SCM) [97] fue presentada después que la representación EV. La SCM posee todas las ventajas de la matriz de Coulomb original, añadiendo una más: invarianza ante el reetiquetado de átomos en las coordenadas. De esta manera, todas las representaciones cartesianas posibles para un mínimo particular generan la misma matriz.

Una vez que tenemos la CM de una molécula, su SCM se obtiene de ordenar los renglones y las columnas de acuerdo al orden no-decreciente de sus normas.

La SCM de una molécula, así como todas sus CM posibles, son matrices simétricas (ecuación 3.12). Esto es un alivio para los modelos ML, puesto que el escalamiento de la capa de entrada se reduce considerablemente: en lugar de tener n_{at}^2 entradas para n_{at} átomos, tenemos solamente $0.5\, n_{\text{at}}(n_{\text{at}}+1)$.

Incluso con la adición de este orden inambiguo, la representación SCM todavía no tiene un significado consistente para sus elementos. Por ejemplo, los elementos respectivos al primer átomo pueden describir a un nitrógeno para algunas muestras, y a un oxígeno para otras. Además, consideremos grupos funcionales semejantes, como un carbonilo y un alcohol o un alcano y un alquino: cambios leves en la geometría podrían alterar el orden de los átomos involucrados, y esto dificulta el reconocimiento de semejanzas estructurales. Por añadidura, una NN entrenada con geometrías optimizadas a un nivel de teoría dado, podría tener resultados sorprendentemente pobres para moléculas optimizadas con un nivel de teoría distinto.

Una forma de mitigar este problema es mediante las matrices de Coulomb ordenadas aleatoriamente (*randomly-sorted Coulomb matrices*, rSCM) [97]. Esta técnica toma la SCM de una molécula, y añade una cantidad pequeña de ruido a las normas de los renglones y columnas. Si esta perturbación altera el orden original, la SCM se reordena, generando una rSCM nueva, y ahora la molécula inicial se describe mediante una matriz más. Esto generaliza la aplicabilidad de un modelo ML, puesto que le otorga algo de independencia con respecto al nivel de cálculo de las estructuras. Por otra parte, este descriptor conlleva un incremento al costo computacional del entrenamiento.

Los autores de este descriptor han escrito un paquete en el *scripting language* `python` para facilitar el cálculo de las SCM, llamado `QML` [98].

3.6.3 Eigenvalores de SCM

Mientras que las matrices de Coulomb ordenadas aleatoriamente buscan representar las moléculas de forma más granular y detallada que las SCM, la representación de eigenvalores (EV) [93] es más pequeña y menos granular. Esta reducción de dimensionalidad no es necesariamente sinónimo de un empeoramiento en los modelos, puesto que una descripción más sencilla siempre aligera el costo de los entrenamientos, y puede incluso introducir una *regularización* en el descriptor que descarta información poco relevante. Esto significa que un modelo con un descriptor más sencillo podría, en algunas circunstancias, tener mejor desempeño que uno con un descriptor más granular.

El tamaño de esta representación es igual al tamaño molecular más grande para la base de datos elegida. Entonces, la representación EV de las moléculas QM9 toma apenas 29 elementos, mientras que la SCM requiere $0.5 \cdot 29 \cdot (29+1) = \frac{29 \cdot 30}{2} = 435$. Esta diferencia se vuelve más pronunciada al pasar a moléculas más grandes: para moléculas de 70 átomos, las longitudes son 70 y 2485. Entonces, podemos ver que el escalamiento de las representaciones SCM y rSCM es problemático para sistemas grandes.

Por otra parte, el escalamiento de EV es benigno, y, su falta de granularidad puede compensarse mediante definir un descriptor EV aumentado, donde se incluyan otros descriptores adicionales con

propiedades estructurales relevantes para la propiedad objetivo. Un ejemplo de estas representaciones auxiliares es la composición atómica, que tratamos en la siguiente sección.

Esta representación hereda las invarianzas traslacionales y rotacionales de la CM, y su carácter de eigenvalor le confiere una más: dado que cualquier reordenamiento de los renglones o columnas de la CM es fundamentalmente una transformada de similitud, la representación EV es invariante al reetiquetado de átomos.

3.7 Composición atómica

La *composición atómica* (AC) es un descriptor sencillo, que básicamente adapta la estequiometría molecular al contexto ML. A pesar de su simpleza, este descriptor tiene implicaciones interesantes, y le usaremos en nuestros experimentos del capítulo 6. La formulación aquí usada se debe a Tchagang y Valdés, quienes en 2019 [99] mostraron que las representaciones tradicionales matriz de Coulomb ordenada (SCM, sección 3.6.2) y sus eigenvalores (EV, sección 3.6.3), pueden enriquecerse al ser incluida la composición atómica. Los descriptores resultantes son más grandes y ricos que los originales, y disponen de manera explícita información que en el descriptor original no es inmediata.

Esto se refleja en la calidad de los entrenamientos realizados por Tchagang y Valdés, quienes predijeron energías de atomización para la base QM7 [100], la cual contiene 7165 moléculas compuestas de hasta siete átomos pesados $\{\mathrm{C}, \mathrm{N}, \mathrm{O}, \mathrm{S}\}$, más sus correspondientes hidrógenos. El mejor de sus modelos SCM tiene un error de $4.4\,\mathrm{kcal\,mol^{-1}}$, mientras que el mejor modelo emplea a la representación SCM + AC, y tiene un error de $3.0\,\mathrm{kcal\,mol^{-1}}$.

También, esta técnica cuenta con precursores:

- los métodos de contribución de grupos son un área longeva, que desde 1949 [101] han buscado predecir propiedades de moléculas orgánicas a partir de diferentes consideraciones estructurales, tomando en cuenta a los átomos, enlaces, y grupos funcionales que componen a la molécula.

  De acuerdo a los términos planteados por Benson [102], la composición atómica es análoga a la ley de aditividad de propiedades atómicas, la cual es la aproximación de orden cero a la ley de aditividad de propiedades moleculares. Benson muestra que las diferentes aproximaciones a esta ley pueden predecir propiedades termodinámicas como C_P, S°, $\Delta H_\mathrm{f}^\circ$.

  Otros métodos de contribución de grupos usan métodos semejantes para predecir la tríada crítica $(P_\mathrm{c}, T_\mathrm{c}, V_\mathrm{c})$ de compuestos puros [103-108], temperaturas de fusión y ebullición [106, 107, 109], e incluso los coeficientes de actividad de compuestos en mezclas no-ideales binarias y terciarias [110].

- Burden y Winkler [111, 112] usan un versión más granular de la composición atómica para predecir refractividad molar, hidrofobicidad, y actividad biológica de moléculas *drug-like*.

Consideramos que estos estudios son evidencia de la capacidad de AC como descriptor en un modelo ML.

La representación AC considera la totalidad de elementos presentes en la base, en lugar de considerar a cada molécula por separado. Por ejemplo, el óxido de propileno, $\mathrm{C_3H_6O}$, es la molécula 44 de la base QM9 (sección 2.3), la cual contiene los átomos $\{\mathrm{H}, \mathrm{C}, \mathrm{N}, \mathrm{O}, \mathrm{F}\}$. Por lo tanto, su representación AC en el contexto QM9 es $(6, 3, 0, 1, 0)$.

Notemos que AC considera exclusivamente la estequiometría molecular: no puede distinguir entre isómeros. Esto es una debilidad; AC aparentemente deshace el logro de Berzelius quien plantea la isomería [25]. Sin embargo, no consideramos que esto demerite al descriptor: Tchagang y Valdés muestran que AC es particularmente valiosa al integrarse en una representación mixta, y dependiendo del resto de representaciones componente, esta insensibilidad puede desaparecer: SCM + AC hereda de SCM la capacidad de distinguir entre isómeros estructurales y conformacionales. También,

EV + AC es generalmente capaz de distinguir entre diferentes estructuras para una estequiometría común [113, 114].

3.8 Resolución de una representación

A lo largo de este capítulo hemos revisado varias representaciones moleculares, y es oportuno que presentemos el concepto de *resolución* de una representación. Usamos este término para comparar la cantidad de información estructural contenida en una representación, particularmente en el capítulo 6.

Este concepto es informal y cualitativo, pero también lo consideramos valioso, dado que simplifica la discusión y comparativa sobre experimentos con diferentes representaciones.

No cualquier par o conjunto de representaciones puede abordarse mediante este término: solo podemos comparar la resolución de ciertas representaciones cuando éstas codifican aspectos estructurales parecidos. Por ejemplo:

- podemos intentar comparar dos vectores de funciones G de Behler-Parrinello (sección 3.5). Ambos vectores podrían representar distintos aspectos estructurales, usando una diferente cantidad y selección de funciones G, con diferentes parámetros. En el supuesto de que un vector esté constituido exclusivamente de G radiales, mientras que el otro tenga una parte radial semejante, pero incluya también una serie de funciones angulares, entonces decimos que el segundo vector (la segunda representación) es de resolución mayor.

  Si un vector de G es puramente radial y otro es puramente angular, no es claro que podamos decir que un vector tiene mayor resolución que el otro, dado que los dos vectores codifican aspectos estructurales distintos. Podríamos argumentar que, dado que las funciones angulares G contienen términos radiales, mientras que las G radiales no contienen información angular, el vector angular es el de resolución mayor. Sin embargo, este planteamiento podría ser engañoso, dado que un vector angular corto podría contener menos información que un vector radial minucioso. Por otra parte, un vector angular minucioso carente de elementos radiales podría presentar información difícil de interpretar por la red, en caso de que los elementos radiales aporten información acerca de la geometría y enlaces en el arreglo atómico.

  En general, podemos decir que, un vector G puramente radial y un vector G puramente angular no son directamente comparables. También, en la práctica, los vectores de funciones G incluyen una parte radial y una angular; el argumento recién hecho postuló a vectores puramente radiales y puramente angulares para ejemplificar la aplicabilidad del concepto de resolución.

- Siguiendo el ejemplo del punto anterior, un vector de funciones G no es comparable con una matriz de Coulomb.

- La composición atómica (AC, sección 3.7) es de menor resolución que la matriz de Coulomb ordenada (SCM, sección 3.6.2) o que sus eigenvalores (EV, sección 3.6.3).

  - La composición lista exclusivamente la estequiometría molecular, mientras que la matriz incluye esta información y también una descripción de la geometría molecular.
  - Los eigenvalores de SCM condensan la estequiometría *y también* la geometría contenidas en la matriz ordenada. También, el tamaño de EV es típicamente mayor al de AC. Para QM9 (sección 2.3), EV es de tamaño 29, mientras que AC es de tamaño 5.

- La matriz de Coulomb ordenada es de mayor resolución que sus eigenvalores.

  Como mencionamos en el punto anterior, la representación EV condensa la toda la información

de SCM en un vector de longitud mucho menor. Sin embargo, es importante aclarar que estas reducciones de tamaño no son inherentemente malas, ni significan que la representación acortada sea inferior a la original. Como explican los autores de la matriz de Coulomb [97]:

> *Computing the eigenspectrum of a molecule reduces the dimensionality from* $(3d-6)$ *degrees of freedom to just* d*. In machine learning, dimensionality reduction can sometimes positively influence the prediction accuracy by providing some regularization. However, such a drastic dimensionality reduction can cause loss of information and introduce unfavorable noise (see Moussa [113] and Rupp* et al. *[114]), like any coarse-grained approach.*

La cita anterior menciona al trabajo de Moussa [113], el cual plantea una objeción interesante a la representación EV: la formulación de grano grueso de EV (es decir, su resolución menor a la de SCM) ocasiona que geometrías distintas pueden tener la misma representación. Moussa plantea un sistema de acetileno (figura 3.7), cuyas geometrías cambian de acuerdo al valor de θ, pero las representaciones EV son constantes.

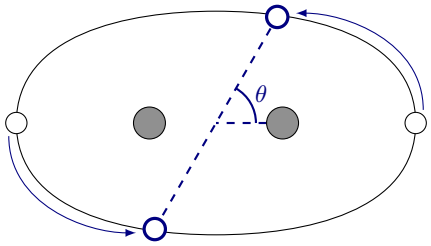


Figura 3.7: Figura 2 de Moussa [113]. Las diferentes geometrías del acetileno HC≡CH tienen diferente energía, pero su representación de eigenvalores es constante.

Luego, el segundo trabajo citado, de Rupp *et al.* [114], es la respuesta al trabajo de Moussa, y contiene propuestas para robustecer el entrenamiento de modelos a partir de la representación EV.

- Si dos representaciones A y B codifican diferente información estructural molecular, entonces no son comparables. Podríamos crear una representación mixta, la cual incorpora toda la información de sus representaciones componente. Para esto, simplemente hemos de concatenar los vectores de A y B. Denotamos a esta representación mixta como $A+B$, donde el operador $+$ indica la concatenación de representaciones y no la suma de escalares o vectores.

  Trabajar con la representación mixta $A+B$ tiene algunas consecuencias interesantes:

  - podemos comparar su resolución con la de sus componentes, incluso si éstos no son comparables entre sí.
  - podemos también comparar el desempeño de modelos entrenados con la representación mixta o con sus componentes por separado. De este análisis podríamos encontrar información valiosa:

    1. Si el espacio de muestras está delimitado en clases, entonces el desempeño de cada representación componente para una clase particular es un indicador de la *relevancia informacional* de esta representación para dicha clase. Esta propiedad es cualitativa, y podemos usarla para comparar a A con B, ya no en términos de resolución, sino en términos de efectividad predictiva.

2. En general, esperamos que $A + B$ tenga un mejor desempeño que sus componentes, dado que la representación mixta contiene más información que cualquiera de sus componentes. Luego, independientemente de si nuestro experimento trabaja con clases, podemos comparar la diferencia entre el desempeño de $A + B$ y el de A, con la diferencia entre $A + B$ y B. Esto nos indica cuál de las componentes es la más influyente en el desempeño de $A + B$.

   También, si encontráramos que el desempeño de la representación mixta es muy superior al de todos sus componentes, esto nos indica que algunas de las componentes tienen información con un sentido físico importante, pero que tal vez no abarca toda la base de datos. Por otra parte, la representación mixta agrega estas informaciones parciales, resultando en una mejor descripción de la propiedad objetivo.

   Por otra parte, si la representación mixta tiene un desempeño muy parecido al de todos sus componentes, entonces sabemos que las diferentes componentes codifican información muy parecida, y son fundamentalmente semejantes.

La inclusión de las representaciones mixtas en nuestros experimentos nos ofrece múltiples oportunidades de evaluar la cercanía entre representaciones moleculares y propiedades fisicoquímicas, y es uno de los axiomas detrás de los experimentos mostrados en el capítulo 6. Este enfoque nos permite extraer conocimiento interpretable de carácter fisicoquímico a partir de nuestros experimentos ML.

Capítulo 4

Diseño experimental y las herramientas del *machine learning*

Índice de capítulo

Como hemos expuesto en la sección 1.2, nuestros experimentos buscan entrenar redes neuronales para la clasificación de moléculas, de acuerdo a la predicción de sus propiedades moleculares a partir de su estructura. Desde el momento que hicimos esa afirmación, hasta ahora, hemos explorado el entorno del aprendizaje de máquina y su intersección con la química, desarrollando los diferentes aspectos que debemos entender antes de poder proponer un experimento:

- funciones de activación (sección 1.5),
- métodos ML predictivos, de los cuales elegimos a las redes neuronales (sección 1.6),
- métodos de inicialización de parámetros (sección 1.7),
- función de costo (sección 1.8),
- métodos de validación (sección 1.9),
- bases de datos químicas (capítulo 2),
- descriptores moleculares (capítulo 3).

Pues bien, nuestro recorrido ha madurado y ya es posible y necesario integrar los conceptos anteriores para formular nuestros experimentos.

Para esto, hay unas decisiones operacionales que debemos tomar, como la delimitación del espacio químico en clases, o el optimizador que minimizará a la función de costo. Abordaremos estos temas en el resto de este capítulo: la discretización del espacio en la sección 4.3, y discutiremos tres optimizadores de gradiente en la sección 4.4.

Hemos incorporado estos conceptos en nuestro programa `Dannte`, el cual usamos para todos nuestros experimentos. Explicaremos su interfaz y el significado de los archivos de entrada y de salida en el capítulo 5.

Finalmente, en el capítulo 6, analizaremos los resultados de nuestros entrenamientos y extraeremos el conocimiento de carácter fisicoquímico proveniente de nuestra metodología clasificativa.

Dado su contenido, la mayor parte de este capítulo está enfocada a la computación aplicada y al aprendizaje de máquina y no a la química computacional.

4.1 Elección de propiedades objetivo

Hemos decidido ya varias de las características de nuestros experimentos, como nuestro enfoque clasificativo, el uso de redes neuronales, entre otras propiedades. En esta sección elegiremos las propiedades objetivo, las cuales intentaremos predecir con nuestros modelos.

También, considerando las representaciones mostradas en el capítulo 3, hemos elegido a la matriz ordenada de Coulomb y sus eigenvalores, dada su definición intuitiva, y a la composición atómica, dada su formulación simple y efectividad en los experimentos mostrados por los autores.

Luego, hemos preferido a la base QM9 en lugar de la PC9 (secciones 2.3 y 2.4) dada su mayor diversidad de de propiedades moleculares. Luego, hemos hecho entrenamientos auscultativos a fin de encontrar la dificultad predictiva relativa de estas propiedades (las 'propiedades escalares', contenidas en la línea 2 de los archivos QM9).

De esta manera, hemos encontrado que las propiedades termodinámicas entalpía (H), energía libre de Gibbs (G), energía vibracional de punto cero (ZPVE) y capacidad calorífica a volumen constante (C_V) son las que mejor responden a las representaciones moleculares elegidas (esto es, los modelos predictores de estas propiedades tienen un error bajo comparado al resto de propiedades).

El buen desempeño de estas cuatro propiedades termodinámicas es consistente con el propósito original de las matrices de Coulomb: predecir la energía de atomización de moléculas orgánicas [93].

4.2 Tratamiento y estandarización de datos

Nuestros experimentos involucran diferentes descriptores moleculares y propiedades de salida, y cada uno presenta su propio intervalo y distribución. Por lo tanto, el uso de las propiedades tal cual incrementa la dificultad del entrenamiento, dado que el modelo tendría que cuidadosamente ajustar sus pesos para escalar las entradas y las salidas tal que todos los descriptores puedan aportar a la predicción.

Corremos también el riesgo de que, dependiendo de los parámetros iniciales, los descriptores con valores absolutos menores sean numéricamente insignificantes desde la primer iteración, y el modelo nunca considere la información contenida en ellos.

A fin de sortear esta dificultad, es habitual realizar una transformación sobre los datos involucrados en nuestro experimento. Una opción común es ajustar cada familia de datos al intervalo lineal $[0, 1]$. Otra muy común, que usamos en nuestros experimentos, es estandarizar cada familia de datos, tal que su distribución nueva tenga $\langle x \rangle = 0$ y $\sigma = 1$. Realizamos este proceso para cada familia de datos por separado:

- Cada propiedad molecular (H, G, ZPVE, C_V) fue estandarizada por separado.
- Por separado también, la composición atómica (sección 3.7) de cada tipo de átomo (hidrógenos, carbonos, ...).
- La matriz de Coulomb ordenada (sección 3.6.2) fue considerada como una sola entidad, dado que no consideramos que sus elementos tengan significados físicos distintos. Por lo tanto, esta estandarización toma en cuenta a $435 \cdot 133\,885 = 5.82 \times 10^7$ valores.
- Seguimos el mismo proceso para los eigenvalores de SCM (sección 3.6.3), y su estandarización considera a $29 \cdot 133\,885 = 3.88 \times 10^6$ valores.

Realizamos el escalamiento mediante la fórmula

$$x_i^{\mathrm{G}} = \frac{x_i - \langle x \rangle}{\sigma} \tag{4.1}$$

donde x_i^{G} es el valor escalado a la distribución gaussiana, $\langle x \rangle$ el promedio, y σ la desviación estándar, la cual se calculó con

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \Big(x_i - \langle x \rangle \Big)^2} \tag{4.2}$$

Listamos algunas propiedades de las distribuciones originales en las tablas 4.1, 4.2, y 4.3.

	min	max	$\langle x \rangle$	σ
n(H)	0	20	9.188	2.829
n(C)	0	9	6.323	1.250
n(N)	0	7	1.044	1.099
n(O)	0	5	1.404	0.884
n(F)	0	6	0.025	0.220

Tabla 4.1: Parámetros de las distribuciones de la composición atómica en QM9 (sección 2.3).

	min	max	$\langle x \rangle$	σ
EV	0	202.664	13.758	27.182
SCM	0	97.533	1.918	7.039

Tabla 4.2: Parámetros de las distribuciones de EV y SCM en QM9.

	min	max	$\langle x \rangle$	σ
H/E_{h}	−714.56	−40.48	−411.53	40.06
G/E_{h}	−714.60	−40.50	−411.58	40.06
ZPVE$/E_{\mathrm{h}}$	0.0160	0.274	0.149	0.0333
$C_V/\mathrm{cal\,mol^{-1}\,K^{-1}}$	6.002	46.969	31.601	4.062

Tabla 4.3: Parámetros de las distribuciones de H, G, ZPVE y C_V en QM9.

4.3 Redes neuronales para clasificación

Hemos revisado el paso de señales en una red neuronal en la sección 1.6. Sin embargo, nuestros planes de clasificación molecular introducen conceptos y decisiones adicionales que debemos de revisar antes de dar por terminada nuestra revisión de los componentes detrás de una red neuronal. Cubriremos estos temas faltantes en esta sección: determinaremos la cantidad y amplitud de cada clase, y fijaremos un criterio para identificar las moléculas bien clasificadas.

Hemos de discretizar el espacio químico en clases. Para esto, dividimos los valores de las propiedades estandarizadas en 25 intervalos, los cuales agrupamos en clases buscando tener cardinalidades homogéneas. Siguiendo este proceso, obtenemos cinco clases para H y G (tabla 4.4), y siete para ZPVE y C_V (tabla 4.5). Mostramos los intervalos y sus agrupamientos en las figuras 4.1 y 4.2. Al

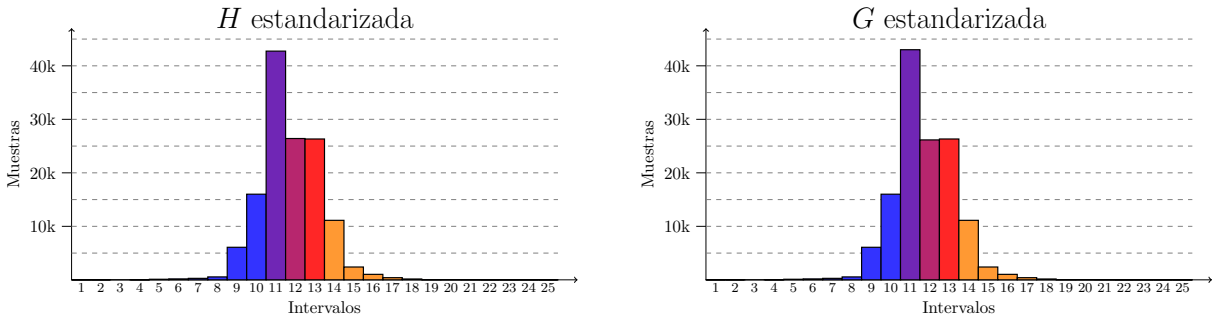
definir las clases, hemos también decidido la cantidad de neuronas en la capa de salida de las redes: las de H y G tienen cinco neuronas, las de ZPVE y C_V tienen siete.

	H			G		
Clase	intervalos	valores/E_h	proporción	intervalos	valores/E_h	proporción
c_1	< 11	$[-714.56, -444.93)$	0.1734	< 11	$[-714.60, -444.96)$	0.1734
c_2	11	$[-444.93, -417.96)$	0.3193	11	$[-444.96, -418.00)$	0.3212
c_3	12	$[-417.96, -391.00)$	0.1973	12	$[-418.00, -391.03)$	0.1953
c_4	13	$[-391.00, -364.04)$	0.1966	13	$[-391.03, -364.07)$	0.1966
c_5	> 13	$[-364.04, -40.48]$	0.1134	> 13	$[-364.07, -40.50]$	0.1134

Tabla 4.4: Población y amplitud de las clases de H y G. Las columnas 'proporción' indican la proporción de moléculas QM9 que pertenecen a cada clase.

	ZPVE			C_V		
Clase	intervalos	valores/E_h	proporción	intervalos	valores/$\mathrm{cal\,mol^{-1}\,K^{-1}}$	proporción
c_1	< 11	$[0.0160, 0.1191)$	0.1839	< 14	$[6.002, 27.305)$	0.1351
c_2	11	$[0.1191, 0.1295)$	0.1021	14	$[27.305, 28.944)$	0.1151
c_3	12	$[0.1295, 0.1398)$	0.1248	15	$[28.944, 30.582)$	0.1517
c_4	13	$[0.1398, 0.1501)$	0.1227	16	$[30.582, 32.221)$	0.1646
c_5	14	$[0.1501, 0.1604)$	0.1279	17	$[32.221, 33.860)$	0.1503
c_6	$15, 16$	$[0.1604, 0.1811)$	0.1653	18	$[33.860, 35.498)$	0.1170
c_7	> 16	$[0.1811, 0.2739]$	0.1733	> 18	$[35.498, 46.969]$	0.1662

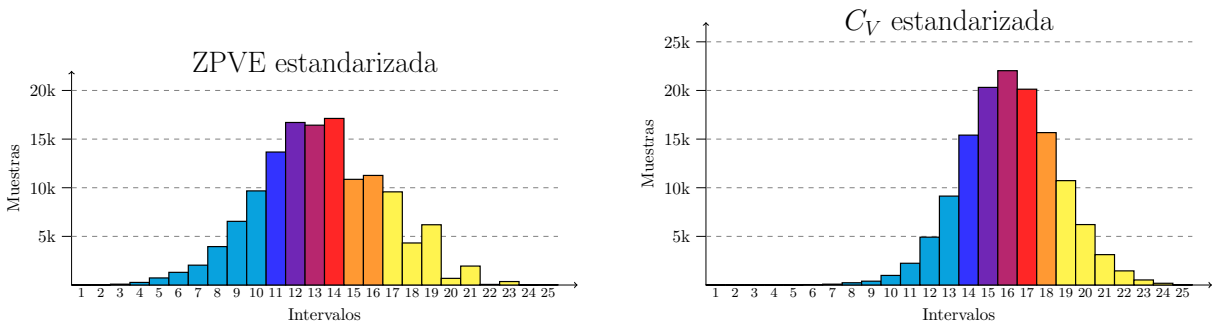
Tabla 4.5: Población y amplitud de las clases de ZPVE y C_V. Las columnas 'proporción' indican la proporción de moléculas QM9 que pertenecen a cada clase.



(a) H. (b) G.

Figura 4.1: Histogramas de H y G estandarizadas. Sus 5 clases se indican con colores, y se detallan en la tabla 4.4.

Reconocemos que estas clases tienen una amplitud mayor a los estándares de precisión de la fisicoquímica teórica. Sin embargo, no consideramos que esto demerite nuestros experimentos, ni haga menos valioso el conocimiento que obtengamos a partir de sus resultados: nuestro objetivo no es encontrar una configuración experimental perfecta; dado que nuestros experimentos son inherentemente transdisciplinares, nuestro objetivo es encontrar un mejor entendimiento de la forma en que se relacionan sus diversos componentes, de carácter fisicoquímico, computacional, y matemático.



(a) ZPVE.

(b) C_V.

Figura 4.2: Histogramas de ZPVE y C_V estandarizadas. Sus 7 clases se indican con colores, y se detallan en la tabla 4.5.

Hay una decisión más que debemos tomar, la cual es una consecuencia directa del enfoque clasificativo de nuestros experimentos: *¿Qué significa que una molécula se haya clasificado correctamente?* Esta pregunta no existe en el marco de un experimento de regresión, el cual se centra en una sola medida: la función de costo. El modelo ideal tiene error 0 para todas las muestras, y el error total de nuestro modelo entrenado es una medida de su idealidad.

Por otra parte, los experimentos de clasificación consideran dos medidas: la función de costo *y* la tasa de moléculas clasificadas correctamente. Esta vez, la tasa de clasificación es prioritaria, mientras que la función de costo pierde el primer plano y simplemente es el medio en que llegamos a un mejor desempeño. El modelo clasificativo ideal tiene un error 0 para todas las muestras, y arroja puntuaciones 0 para las clases a las que la muestra no pertenece, y 1 para las que sí*.

Sin embargo, podríamos tener un modelo que arroja puntuaciones 0.1 y 0.9 en lugar de 0 y 1. Estas puntuaciones son diferentes a las ideales, pero resultan en el mismo veredicto. ¿Consideraríamos ideal a un modelo con puntuaciones 0.2 y 0.8? Debemos establecer un criterio antes de poder evaluar a nuestros modelos. Dada la formulación de nuestros experimentos, todas las muestras pertenecen estrictamente a *una* clase, y por lo tanto, podemos simplemente considerar que la clase con la puntuación más alta es la predicción del modelo.

Otro criterio posible es: el modelo arroja puntuaciones $< a$ para las clases incorrectas, y $> b$ para la correcta, donde podríamos elegir, digamos, $a = b = 0.5$. Los criterios de esta serie suelen ser más estrictos que el del párrafo anterior, y solamente son viables cuando conocemos la clase verdadera de las muestras. A pesar de esta limitante, estos criterios pueden ser auxiliares útiles de auscultación durante la fase de entrenamiento, dado que generan información complementaria valiosa.

4.4 Optimizadores de gradiente

Desde el punto de vista matemático, el entrenamiento de un modelo ML se puede considerar como un simple problema de optimización: Se busca encontrar los parámetros $\boldsymbol{W}$ y $\boldsymbol{b}$ tal que la función de costo $J(\boldsymbol{W}, \boldsymbol{b})$ tenga un valor mínimo.

Por lo tanto, la discusión sobre los métodos de optimización de parámetros durante un entrenamiento ML suele realizarse en un contexto más abstracto, ajeno al aprendizaje de máquina; los

*Esto es el caso particular de un modelo donde la función de activación es $f(z) = \text{logística}(z)$. Si usáramos $f(z) = \tanh(z)$, las puntuaciones serían -1 y 1, en lugar de 0 y 1, debido al rango de tanh. Mostramos estas funciones en las figuras 1.2 y 1.4.

optimizadores son suficientemente generales como para poder abordar a funciones más allá del ML. Entonces, en esta sección denotaremos a $J(\boldsymbol{W}, \boldsymbol{b})$ de manera más genérica: $f(\boldsymbol{\theta}) = f(\theta_1, \theta_2, \ldots, \theta_N)$.

A continuación mostraremos algunos de los optimizadores más populares en ML, particularmente la familia de derivados del método *descenso de gradiente* (GD). Estos optimizadores requieren la derivada *de primer orden* de la función a optimizar (es decir, debe existir y ser calculable), y entonces están catalogados como *optimizadores de primer orden.*

El funcionamiento general de estos algoritmos es:

1. Sea f la función objetivo, N-dimensional, continua, y derivable.
2. Definir un punto de inicio $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_N)$, el cual puede obtenerse aleatoriamente o provenir de un experimento anterior.
3. Iterar sobre estos cuatro pasos:
   - 3.1 Calcular el gradiente de f en el punto $\boldsymbol{\theta}$.
   - 3.2 El optimizador combina de cierta manera la información del gradiente actual y los anteriores para encontrar la dirección que minimiza a f.
   - 3.3 Dar un paso de tamaño $\eta \approx 0$ en la dirección encontrada en el paso anterior.
   - 3.4 Guardar el punto actual, al que llegamos en el paso anterior, en $\boldsymbol{\theta}$, y regresar al paso 3.1.
4. Finalmente, tenemos que $f(\boldsymbol{\theta})$ ha disminuido durante la optimización. Si f es convexa, el valor final de $\boldsymbol{\theta}$ es el mínimo global. En caso contrario, $\boldsymbol{\theta}$ es un mínimo local.

Por lo tanto, podemos ver que la los métodos de gradiente sondean la topología a f a partir de un solo explorador. Otros métodos, por ejemplo, los algoritmos genéticos [115], exploran el espacio de búsqueda mediante *poblaciones* de decenas o cientos de exploradores, y no requieren que f sea derivable, ni continua. A cambio de esta mayor versatilidad, los algoritmos genéticos suelen incurrir en un costo computacional mayor al de los métodos de gradiente, excepto en los casos en los que la derivada de f existe pero su cálculo es muy costoso.

4.4.1 Descenso de gradiente

Este método de optimización, *gradient descent* (GD), es una consecuencia natural del concepto de la derivada de una función multivariada $f(\theta_1, \theta_2)$: si su derivada es conocida, el cálculo de su gradiente en el punto (θ_1, θ_2) indica la dirección en la cual el cambio en f es más pronunciado. Por lo tanto, podemos acercarnos a un mínimo de f al desplazarnos en la dirección $-\nabla f$, dando un paso de tamaño $\eta \approx 0$. De esta manera, la posición nueva es $\left(\theta_1 - \eta\frac{\partial f}{\partial \theta_1}, \theta_2 - \eta\frac{\partial f}{\partial \theta_2}\right)$. Podemos aproximarnos a un mínimo de f iterando sobre este proceso.

En general, el descenso de gradiente se puede expresar de esta manera:

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i} \tag{4.3}$$

donde $\theta_{a,i}$ es el parámetro i del modelo ML en la iteración a, $g_{a,i}$ es la derivada de $f(\boldsymbol{\theta})$ con respecto a $\theta_{a,i}$, y η es el tamaño de paso o *learning rate*. Notemos que este parámetro es constante para *todas* las iteraciones del experimento, y es usado para actualizar *todas* las $\theta_{t,i}$ del modelo. Esto puede provocar que, cuando f depende de múltiples variables, el gradiente de f ya haya convergido en algunas direcciones, para las cuales η es demasiado grande, y demasiado chica para otras. Este inconveniente puede mitigarse si el usuario monitorea el decaimiento de f y reduce a η cuando f deja de decaer.

Cuando f es convexa y continua, el método GD basta para minimizarle, y una vez que realicemos suficientes iteraciones estaremos en la vecindad del mínimo global. Sin embargo, los problemas

abordados con aprendizaje de máquina suelen presentar funciones con múltiples mínimos y topologías menos regulares. Por lo tanto, se suele preferir a métodos como el descenso de gradiente estocástico (SGD) o a los métodos adaptativos, de los cuales el optimizador Adam es una de las alternativas más populares, la cual abordaremos en la sección 4.4.2. Una de sus variantes más interesantes es AdaBelief, que cubriremos en la sección 4.4.3.

4.4.2 Adam

El método Adam [51, 52] es ubicuo en el aprendizaje de máquina. Es un descendiente directo de los métodos AdaGrad [116] y RMSProp [117], los cuales son modificaciones de GD. De AdaGrad, Adam hereda la capacidad de trabajar con gradientes de regiones muy planas; de RMSProp proviene su capacidad de tratar a problemas no estacionarios.

Adam es un método *adaptativo*, debido a que emplea tamaños de paso independientes para cada variable de f, cuyas magnitudes son ajustadas dinámicamente a lo largo de la optimización: paso a paso se toma en cuenta el gradiente actual y el de las iteraciones más recientes.

Cuando el gradiente de f con respecto a θ_i es grande, tenemos que su tamaño de paso efectivo se reduce, para compensar el abrupto paisaje local de f. Luego, cuando el gradiente es pequeño, el tamaño de paso efectivo se incrementa, a fin de buscar salir de esta región plana y llegar a una con un gradiente mayor. Estas dos etapas pueden presentarse múltiples veces durante el entrenamiento, y el tamaño de paso efectivo de cada variable responderá a estos cambios. Esta reactividad es característica de los métodos adaptativos.

Si $f = f(\theta_1, \theta_2, \ldots, \theta_N)$, la forma más simple de registrar los gradientes de los puntos anteriores es almacenarlos en una matriz $N \times t$, donde t es la cantidad de iteraciones del proceso. Sin embargo, Adam usa un método más refinado y elegante: los valores de los N gradientes de f, a lo largo de t iteraciones, son almacenados en un vector de tamaño N. Esto es posible a través del uso de los *exponential moving average* (EMA), los cuales ya se usaban en métodos predecesores como AdaDelta y RMSProp. Adam usa dos: m_t almacena los gradientes anteriores; v_t almacena sus cuadrados. Enfoquémonos en la variable θ_i. Sus EMA son:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{4.4}$$

donde g_t es el gradiente de f en la iteración t; m_t es el *momentum* (o inercia) presente en el explorador; v_t es la velocidad del explorador. β_1 y β_2 son hiperparámetros de Adam con valores ≈ 1.

Notemos que m_t y v_t almacenan los gradientes a lo largo de todas las iteraciones, dando más peso al gradiente de las iteraciones pasadas que al de la actual. También, si consideramos a m_t como una combinación lineal de los gradientes encontrados durante el experimento, entonces el gradiente de la iteración $t-1$ (la anterior) es el término con el coeficiente más grande, y los coeficientes de las iteraciones previas son sucesivamente más pequeños. De esta manera se almacenan todos los gradientes anteriores de θ_i en un solo valor; basta con un vector de tamaño N para almacenar los N gradientes de $\boldsymbol{\theta}$ a lo largo de t iteraciones.

Como podemos ver en la ecuación 4.4, el valor de m_t y v_t evoluciona según avanza el experimento. Esta es la razón detrás de la palabra *moving* en el nombre *exponential moving average*. Una consecuencia más de este artificio es que el movimiento del explorador en la superficie f emula la forma en que una esfera desciende por una superficie, donde la inercia acumulada en las iteraciones pasadas tiene más importancia que el gradiente del punto actual; a la siguiente iteración, este gradiente adquiere una importancia mucho mayor.

En la primera iteración, $g_t = g_1$ se calcula a partir de la ubicación inicial del explorador. En experimentos ML, es habitual iniciar el experimento en un punto aleatorio (como explicamos en la sección 1.7), el cual es probablemente lejano a un mínimo de f. Por otra parte, los EMAs son

inicializados en 0 (es decir, $m_0 = v_0 = 0$). Esto, aunado a que $\beta_1 \approx \beta_2 \approx 1$, significa que durante las primeras iteraciones tenemos m_t y v_t sesgados hacia 0. Buscando revertir este efecto, se realizan las correcciones a los momentos:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

donde $\beta_{1,2}^t$ es $\beta_{1,2}$ a la potencia t.

Finalmente, el nuevo valor de θ_i se computa:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\widehat{v}_t} + \varepsilon}\widehat{m}_t$$

Los autores de Adam [52] sugieren este conjunto de hiperparámetros: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1 \times 10^{-8}$.

Adam está implementado en múltiples paquetes de aprendizaje de máquina, incluyendo a `Flux.jl` [42, 43], que usamos en este proyecto.

4.4.3 AdaBelief

AdaBelief [53] es una modificación ingeniosa de Adam. La premisa fundamental de AdaBelief es que el EMA del gradiente, acumulado en las iteraciones pasadas, es aproximadamente igual al valor del gradiente en la iteración actual. Para cada iteración se calcula la diferencia entre estos dos vectores, la cual determina nuestro nivel de confianza en el gradiente observado (es decir, el tamaño de nuestro *belief* en él), y por lo tanto el tamaño de paso efectivo de cada iteración: cuando el gradiente actual es similar a su EMA, confiamos en ese gradiente y damos un paso más grande. Por otra parte, según aumenta la diferencia, desconfiamos que la dirección del gradiente actual nos acerque a un mínimo, y entonces el tamaño de paso de esta iteración es más pequeño.

Los autores argumentan que AdaBelief logra estas tres metas:

- la convergencia veloz característica de los métodos adaptativos,
- buen desempeño de generalización, característico del descenso de gradiente estocástico (SGD, *stochastic gradient descent*) y métodos derivados,
- entrenamientos estables incluso en problemas complejos como las redes generativas adversariales (GAN, *generative adversarial network*).

La razón de esta tríada de objetivos es la ambición unificadora de AdaBelief: la mayoría de optimizadores usados en experimentos ML pertenece a una de dos familias: la familia adaptativa, y la SGD. Diferentes áreas del ML prefieren recurrir a una familia o a otra según las características de su problema. Por ejemplo:

- en las redes neuronales convolucionales (CNN) es habitual usar a los métodos adaptativos, dado que su entrenamiento es típicamente más veloz que el de los métodos SGD. Sin embargo, es habitual que su error de generalización sea mayor.
- Wilson *et al.* [118] comparan el desempeño de estas dos familias en experimentos de visión por computadora, modelado de lenguaje, y análisis sintáctico. Habitualmente, los métodos SGD presentan un aprendizaje más lento en las etapas tempranas, pero su error de generalización es mejor, incluso si los adaptativos alcanzan un error de entrenamiento menor.
- Adam es un optimizador deseable para entrenar GANs [119], dado que provee entrenamientos estables. Las GAN son métodos complejos donde se entrenan dos redes adversarias: G busca

generar muestras ficticias semejantes a las del conjunto de entrenamiento, y D busca distinguir las muestras originales de las generadas. En una GAN ideal, G logra generar muestras ficticias con aspecto perfectamente realista, tal que ningún experto humano es capaz de distinguirlas de las verdaderas; solamente D es capaz de identificar el origen de cada muestra.

Entonces, AdaBelief busca establecerse como una opción competitiva para los problemas diversos mencionados anteriormente. Hemos incorporado este optimizador a nuestros experimentos debido a su elegante formulación teórica, y también dado su buen desempeño para nuestros problemas de clasificación molecular. Hemos encontrado que en experimentos con descriptores sencillos, el rendimiento de sus modelos es inalcanzable por Adam. Discutimos estos resultados en el capítulo 6.

Para abordar el funcionamiento de AdaBelief, consideremos la siguiente comparativa, la cual es una modificación de la mostrada por los autores [53], donde `niter` es la variable con la cantidad de iteraciones a realizar, y el símbolo ← indica asignación de un valor. Las líneas que empiezan con # son comentarios.

Optimizador Adam

$\boldsymbol{m}_0 \leftarrow 0$, $\boldsymbol{v}_0 \leftarrow 0$

`for` i `in 1:niter`

$\boldsymbol{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_t)$

$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$

$\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$

#Corregir el sesgo hacia cero en

$\boldsymbol{m}_t$ y $\boldsymbol{v}_t$ cuando $t \approx 0$

$\widehat{\boldsymbol{m}}_t \leftarrow \dfrac{\boldsymbol{m}_t}{1-\beta_1^t}$

$\widehat{\boldsymbol{v}}_t \leftarrow \dfrac{\boldsymbol{v}_t}{1-\beta_2^t}$

#Actualizar los parámetros del modelo

$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \dfrac{\eta \widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{v}}_t} + \varepsilon}$

`end for`

Optimizador AdaBelief

$\boldsymbol{m}_0 \leftarrow 0$, $\boldsymbol{s}_0 \leftarrow 0$

`for` i `in 1:niter`

$\boldsymbol{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_t)$

$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$

$\boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t - \boldsymbol{m}_t)^2 + \varepsilon$

#Corregir el sesgo hacia cero en

$\boldsymbol{m}_t$ y $\boldsymbol{v}_t$ cuando $t \approx 0$

$\widehat{\boldsymbol{m}}_t \leftarrow \dfrac{\boldsymbol{m}_t}{1-\beta_1^t}$

$\widehat{\boldsymbol{s}}_t \leftarrow \dfrac{\boldsymbol{s}_t}{1-\beta_2^t}$

#Actualizar los parámetros del modelo

$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \dfrac{\eta \widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{s}}_t} + \varepsilon}$

`end for`

Las partes color ■ indican las diferencias entre los métodos: $\boldsymbol{v}_t$ es reemplazada por $\boldsymbol{s}_t$, en la cual se computa la diferencia entre el gradiente actual, $\boldsymbol{g}_t$, y su EMA, $\boldsymbol{m}_t$. Cuando estas dos magnitudes difieren, $\|\boldsymbol{s}_t\| \gg 0$. Esto determina lo que sucede en la actualización de parámetros, donde el tamaño de paso efectivo disminuye. Si, por otra parte, $\boldsymbol{g}_t \approx \boldsymbol{m}_t$, el tamaño de paso se incrementa.

También podemos notar que AdaBelief usa los mismos hiperparámetros que Adam. Los valores sugeridos por los autores son iguales a los de Adam: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1 \times 10^{-8}$.

AdaBelief está implementado en el paquete `Flux.jl` [42, 43], que usamos en este proyecto.

Capítulo 5

`Dannte`, una plataforma para la exploración sistemática del aprendizaje de máquina

Índice de capítulo

A lo largo de esta ICR hemos dado un recorrido por los diversos aspectos teóricos componentes de un experimento de aprendizaje de máquina. En resumen:

- hemos abordado aspectos fundamentales del aprendizaje de máquina en el capítulo 1,
- dimos un recorrido minucioso por algunas de las bases más populares en el aprendizaje de máquina químico (capítulo 2),
- explicamos la forma en que una molécula se codifica tal que pueda ser manipulada por un modelo ML en el capítulo 3,
- revisamos algunos aspectos de aprendizaje de máquina más especializados en el capítulo 4, lo cual constituye el primer paso en la realización de nuestros experimentos.

Finalmente, en este capítulo explicaremos la implementación que hemos realizado para incorporar los conceptos y técnicas mencionados anteriormente, y ahora dar el paso a la *praxis*.

Hemos escrito nuestro propio programa para la exploración sistemática del aprendizaje de máquina químico, al que hemos llamado `Dannte` [120], con el cual hemos realizado los experimentos que discutiremos en el capítulo 6.

Nuestro programa está escrito en el lenguaje `julia` [121], un lenguaje enfocado a la programación científica, con una comunidad sumamente activa que ha desarrollado una gran cantidad de paquetes, como por ejemplo `Flux.jl` [42, 43], el cual usamos activamente en `Dannte`, así como otros paquetes y bibliotecas [122-130].

5.1 Configuración inicial de `Dannte`

Es indispensable contar con una instalación de `julia` para poder ejecutar nuestro código. Para esto:

- los instaladores para múltiples lenguajes están alojados en https://julialang.org/downloads/,
- y las instrucciones de instalación están en https://julialang.org/downloads/platform/.

Habiendo instalado el lenguaje, podemos instalar los paquetes empleados por `Dannte`. Para lograrlo, basta con guardar el siguiente bloque de texto a un archivo con extensión `jl` y ejecutarlo:

```
import Pkg

Pkg.add("Random")
Pkg.add("Flux")
Pkg.add("Zygote")
Pkg.add("Parameters")
Pkg.add("Distributions")
Pkg.add("DelimitedFiles")
Pkg.add("ProgressMeter")
Pkg.add("Plots")
Pkg.add("BenchmarkTools")
Pkg.add("Dates")
Pkg.add("StatsBase")
Pkg.add("Statistics")
Pkg.add("HDF5")
Pkg.add("Formatting")
```

Luego, necesitaremos establecer una estructura de carpetas que aproveche el carácter modular de nuestro código: los diferentes experimentos que realizaremos tendrán varios componentes comunes, como la base de datos, los optimizadores, la arquitectura, o la cantidad de iteraciones. Por lo tanto, podemos simplificar los archivos de entrada de nuestros cálculos escribiendo en *archivos de configuración* las variables comunes para toda una familia de experimentos. Luego, simplemente hemos de cargar estos archivos desde el archivo de entrada.

A continuación mostramos la estructura de carpetas que hemos usado durante nuestros experimentos, y posteriormente veremos ejemplos de los archivos que hemos de almacenar en ellas:

- La carpeta raíz de los experimentos, en la cual almacenamos los archivos de entrada (o *jobfiles*), los archivos de código que constituyen a `Dannte`, y el resto de carpetas con archivos de configuración y las que recibirán las salidas de los cálculos.

  Las carpetas de configuración son:

  - `Holder-h5/`. Esta carpeta contiene nuestros reempacamientos de la base QM9 (sección 2.3), codificada según los descriptores de las secciones 3.6 y 3.7. Usamos el formato `HDF5`, el cual es altamente eficiente en términos de espacio en disco, especialmente para arreglos numéricos grandes como los derivados de QM9, que tienen una cardinalidad múltiplo de 133 885.

    Por ejemplo, nuestros primeros experimentos leían las matrices de Coulomb contenidas

en archivos de texto con 435 valores, almacenados en una carpeta. Esta representación requería 859 MB. Por otra parte, la codificación `h5` guarda a esos $435 \cdot 133\,885 = 5.82 \times 10^7$ valores en *un solo* archivo, ocupando apenas 153 MB.

La extensión `h5` indica que estos archivos siguen el formato *Hierarchical Data Format 5* [131], el cual está pensado para manejar datos a gran escala, y es compatible con una variedad de lenguajes de programación como `C`, `Fortran`, `julia`, `haskell`, el *scripting language* `python`, entre otros. Este formato es usado por grupos de investigación en la NASA [132] y el CERN [133].

Empleamos al paquete `HDF5.jl` [129] para escribir y leer archivos de este formato.

- `Optimizer-grad-lists/`. Un objetivo central de `Dannte` es la auscultación del horizonte de optimizadores para el entrenamiento de un modelo dado. Por lo tanto, esta carpeta almacena listas de optimizadores que pondremos a prueba para diferentes descriptores y propiedades objetivo.

  Estos archivos son de extensión `jl`, sencillos, legibles y editables por humanos. Los discutimos más adelante, en la sección 5.2.1.

- `Pools-TT/`. Esta carpeta almacena archivos de texto con listas de enteros, indicando los índices que formarán parte de un experimento. Mostramos la sintaxis de estos archivos en la sección 5.2.2.

5.2 Archivos de entrada del cálculo

La concepción de 'archivo de entrada' de `Dannte` es agresivamente modular. Esto permite simplificar los archivos de entrada principales, dado que podemos definir algunas variables del cálculo en archivos externos, que incorporamos al archivo de entrada principal, simplemente mencionando sus nombres. Este enfoque también facilita el enviar una familia de cálculos a fin de comparar diferentes representaciones, arquitecturas, optimizadores, ...

Si, por otra parte, tuviéramos interés en mandar un solo cálculo, esto también es posible: podríamos incorporar las variables mostradas en esta sección en un solo archivo de entrada, y ejecutarlo.

5.2.1 Archivo de optimizadores

Hacemos uso de los numerosos optimizadores de gradiente definidos por `Flux.jl` [42, 43] para escribir este archivo de entrada: simplemente definimos una lista de optimizadores, `optilist`, los cuales escribimos con hiperparámetros explícitos de acuerdo al manual del paquete [134].

Definimos también a la lista `optinoms`, un arreglo con elementos `String` que contiene los nombres 'humanos' con que hemos denominado a los optimizadores. `Dannte` toma estos nombres para referirse a cada optimizador del experimento y para nombrar las carpetas con los resultados de cada uno.

Este archivo ejemplo es `list-optimizers-grad-k5.jl`, y lo almacenamos en la carpeta `Optimizer-grad-lists/`.

```
#= Una lista elemental de optimizadores de gradiente
   21-08-21
=#

const optilist = [
Flux.Optimise.ADAM(0.001, (0.9, 0.999)),
Flux.Optimise.AdaBelief(0.001, (0.9, 0.999))
```

```
]
const optinoms = [
"Adam-d",
"Adabelief-d"
]
const noptimizers = length(optinoms)
```

Hemos nombrado estos dos optimizadores con el sufijo `-d`, dado que usan los parámetros `default`, los sugeridos por sus respectivos autores. Tratamos a estos dos optimizadores con más detalle en las secciones 4.4.2 y 4.4.3.

5.2.2 Archivo de índices de las muestras del experimento

En un experimento k-fold, como los mostrados en el capítulo 6, las muestras de la base de datos tienen un rol doble: participan tanto en el conjunto de prueba como el de entrenamiento. Por lo tanto, es absurdo el describir por separado a un conjunto de prueba y uno de entrenamiento. En su lugar, trabajamos con una lista conjunta, que hemos denominado `TTpool` (*training-and-testing sample pool*).

Estos archivos pueden contener valores enteros, o también `UnitRanges` de `julia`, artificios elegantes que *declaran* un intervalo en lugar de listar explícitamente cada uno de los elementos. Mostramos uno en el ejemplo a continuación.

La maquinaria de lectura de los archivos `TTpool` construye un vector de elementos `Bool` con la lista de muestras consideradas y las descartadas. Por lo tanto, basta con indicar una sola vez que una muestra dada debe considerarse en el experimento, pero no hay problema si una muestra se listara múltiples veces, tanto en enteros como en `UnitRanges`.

Este archivo ejemplo es `pool-TT-QM9.dat`, y lo almacenamos en la carpeta `Pools-TT/`.

```
1:133885
```

Los números del `UnitRange` anterior comprenden desde la primera hasta la última muestra de la base QM9 (sección 2.3); nuestro experimento considerará a todas sus moléculas.

5.2.3 El archivo de entrada (*jobfile*)

En esta sección mostraremos un archivo de entrada que generó uno de los experimentos discutidos en el capítulo 6 (específicamente, los experimentos 'EV + AC-ZPVE' de la sección 6.6).

Cada sección experimental del capítulo de resultados comprende los resultados de cinco *jobfiles* distintos:

- los experimentos de resolución baja, usando los descriptores
  1. AC (sección 3.7)
  2. EV (sección 3.6.3)
  3. EV + AC
- y los experimentos de resolución alta,
  4. SCM (sección 3.6.2)
  5. SCM + AC

luego, cada *jobfile* considera a los optimizadores Adam y AdaBelief (secciones 4.4.2 y 4.4.3), y cada propiedad objetivo tiene su propia péntada de archivos de entrada.

Todos estos experimentos tienen algunas propiedades en común, como su cantidad de iteraciones, su `TTpool`, o la lista de optimizadores. Por lo tanto, hemos trabajado con archivos de entrada ensamblables, donde un archivo `incore` define las variables comunes (sección 5.2.3.1), mientras que el archivo principal define aspectos particulares de cada experimento (sección 5.2.3.2).

5.2.3.1 Archivo `incore` para variables compartidas

Los archivos `incore` simplifican la construcción de los archivos de entrada, dado que almacenan las variables comunes, compartidas a lo largo de varios experimentos. Así, basta con editar un solo archivo para modificar las propiedades experimentales compartidas; el archivo de entrada principal solo enlista las características particulares del experimento.

Hemos nombrado a este tipo de archivos como `incore`, al ser el *core* de un archivo de entrada. Todos los archivos de este tipo deben contener la palabra `incore` en su nombre.

Este archivo ejemplo es `incore-mckh-zpve.jl`, y lo almacenamos en la carpeta raíz.

```
1  const descrJob = "Experimento kfold para para propiedades termodinámicas"
2  const descrModel = "NN con 3 capas ocultas"
3
4  # Elegir la propiedad de salida (zpve)
5  const nomprop = "zpve"
6  const labelsfam = "labelsgau"
7  const readstyle = 'h'
8
9  # Definir la cantidad de iteraciones
10 const niter = 2000
11
12 # Elegir la lista de optimizadores de este experimento
13 const fileopts = "list-optimizers-grad-k5.jl"
14 include("Optimizer-grad-lists/"*fileopts)
15
16 # Elegir los índices del conjunto de prueba y entrenamiento
17 const fileindicesTTpool = "pool-TT-QM9.dat"
18 const indicesTTpool = readpoolvec("Pools-TT/"*fileindicesTTpool)
19 const descrTTpool = "QM9 set, \""*fileindicesTTpool*"\""
20 const nmolecTTpool = length(indicesTTpool)
21
22 # Variables del método de validación
23 const validationm = "kfold"
24 const nfolds = 5
25 const nRepeat = 1
26
27 # Calcular medidas de error adicionales
28 const cart_xent = false
29 const cart_mae = false
30 const cart_mse = false
31
32 # Elegir el formato de los archivos de salida adicionales
33 const outstyle_decay = "h5"
34 const outstyle_corr = "h5"
35 const outstyle_par = "h5champions"
```

El significado de las líneas del archivo es:

- 1–2, estas variables almacenan descripciones del experimento a realizar y el modelo,
- 4–7, este bloque determina la propiedad objetivo (ZPVE, con valores normalizados), y la forma en que se hará la lectura de datos (leer de un archivo `h5`),
- 9–10, define la cantidad de iteraciones del cálculo. Hemos trabajado con una cantidad de

iteraciones fija, en lugar de un criterio de convergencia basado en un estancamiento de la función de costo, dado que, como es claro en las figuras del capítulo 6 (por ejemplo, 6.1), los entrenamientos suelen tener un decaimiento acelerado, luego una etapa de estabilización, y finalmente decaen una vez más. Implementar un criterio de terminación temprana podría evitarnos el llegar a estos decaimientos posteriores.

- 12–14, este bloque nombra y carga el archivo con la lista de optimizadores, el cual mostramos en la sección 5.2.1,
- 16–20, da el nombre del archivo con la `TTpool` de este experimento (sección 5.2.2), lo lee, almacena una descripción elemental del mismo, y cuenta la cantidad de moléculas,
- 22–25, dan el método de validación y el valor de k. En `nRepeat` guardamos la cantidad de experimentos a realizar para cada optimizador. Si cambiáramos su valor a `2`, entonces para cada optimizador se realizarían dos experimentos k-fold con $k = 5$ (es decir, cada optimizador entrenaría 10 modelos en total). Hemos hablado de los métodos de validación en la sección 1.9.
- 27–30, determinan si se calcularán medidas de error adicionales: entropía cruzada (`X-entropy`, `xent`), promedio del error absoluto (`mae`), y promedio del error cuadrado (`mse`). Tratamos brevemente a estas funciones de costo en la sección 1.8.
- 32–35, dan el formato de los archivos de salida especializados, que contienen información más minuciosa y detallada. A fin de ahorrar espacio en disco, elegimos el formato `h5`. La primer opción determina la forma en que se guardan los decaimientos de las medidas de error, la segunda es referente a las moléculas clasificadas *correctamente*, y la tercera es sobre los parámetros finales de cada experimento (es decir, los modelos ya entrenados). Detallamos estos archivos en la sección 5.3.2.

  Los modelos finales son probablemente el resultado más valioso de todo experimento ML, dado que a partir de éstos se pueden calcular múltiples métricas acerca del desempeño del modelo, así como ponerlo a prueba con cualquier selección de datos, incluso una fuera del espacio de entrenamiento.

5.2.3.2 Archivo principal para variables particulares

El archivo de entrada de un cálculo es ensamblable. Para experimentos únicos, podemos definir un solo archivo que lista cada una de las variables necesarias. Si, por el contrario, planeáramos realizar un arreglo de experimentos buscando comparar diferentes descriptores y propiedades objetivo, entonces podemos dividir el archivo en:

- una serie de archivos `incore` que contienen variables comunes (sección 5.2.3.1),
- y archivos de entrada particulares que contienen las variables particulares del experimento.

A continuación mostramos un ejemplo de estos archivos de entrada reducidos.

Este archivo ejemplo es `mckh-kf-a-nn3-zpve-evac-QM9.jl`, y lo almacenamos en la carpeta raíz.

```
1  # Experimento kfold para propiedades termodinámicas
2
3  include("machinery-input.jl")
4
5  include("incore-mckh-zpve.jl")
6
7  const folderOptResults = "KH-kf-NN3-a-zpve-evac-QM9/"
8
9  const nneur_h1 = 26
```

```
10 const nneur_h2 = 20
11 const nneur_h3 = 14
12
13 # Elegir el descriptor molecular
14 const vecfams = ["evscmgau","atomcompgau"]
15
16 const jobfile = Base.@__FILE__ # No tocar esta línea
17
18 # Por fin ejecutar el cálculo
19 include("cothon-nn3.jl")
```

El significado de las líneas del archivo es:

- 1, este bloque puede contener tantas líneas de comentario como sea necesario, explicando la meta de este cálculo, o su rol dentro del arreglo de experimentos al que pertenece,
- 3, esta línea debe ser la primera después del bloque de comentarios, y define algunas funciones elementales que se usarán para definir el resto de variables. Esto sucede todavía mucho antes de definir la red neuronal o hacer la lectura de la base de datos.
- 5, carga el código del archivo `incore` de la sección 5.2.3.1,
- 7, nombra la carpeta de salida de este experimento, la cual almacenará los archivos de salida con los resultados del experimento. Los enlistamos en la sección 5.3.
- 9–11, define la cantidad de neuronas en cada capa oculta de la red. Hemos planeado que el modelo de nuestros experimentos sea una NN con tres capas ocultas, entonces es obligatorio definir las tres variables de este bloque.
- 13–14, elegimos los descriptores a usar en este experimento (en este caso, EV + AC),
- 16, esta línea es obligatoria, debe ser parte del archivo principal en lugar de trasladarse a un `incore`. Esta variable tiene fines de documentación y seguimiento. Por ejemplo, el nombre del *jobfile* principal será guardado en un archivo de texto en la carpeta de salida.
- 18–19. Es obligatorio que estas líneas sean las últimas. Aquí es donde finalmente se reúnen las especificaciones dadas en las variables anteriores, y por fin emprenderemos el cálculo. Notemos que el nombre del archivo `cothon` solicitado debe corresponder a la profundidad de red deseada (en este caso, tres capas ocultas). Por el momento, `Dannte` puede entrenar modelos de cero a cinco capas ocultas; no hay restricciones en la cantidad de neuronas ocultas en cada capa.

5.3 Archivos de salida del cálculo

Hay diferentes archivos de salida, cada uno con un diferente propósito. En esta sección explicaremos los nombres, propósito, contenido y formato de cada uno.

5.3.1 Archivos de perspectiva general

En la línea 7 del archivo de entrada principal (sección 5.2.3.2) definimos el valor de `folderOptResults`, `KH-kf-NN3-a-zpve-evac-QM9/`. Esta carpeta es creada al inicio del entrenamiento, dentro de la cual se guardan archivos de carácter global que veremos en esta sección, y archivos particulares de cada optimizador, que son almacenados en su propia carpeta y veremos en la sección 5.3.2.

Los contenidos de la carpeta `folderOptResults` son:

- `cp-mckh-kf-a-nn3-zpve-evac-QM9.jl`, este archivo es una copia del archivo de entrada original. Lo guardamos en la carpeta de salida para facilitar la modificación o repetición del

cálculo.

- `incore-mckh-zpve.jl`. También son copiados todos los archivos `incore` mencionados en el archivo de entrada original, a fin de asegurar la repetibilidad de nuestros cálculos.
- `logprocess.log`. Este es un archivo de seguimiento. `Dannte` lo actualiza según el avance del experimento; lo mostramos con más detalle a continuación.
- `readme-experimentsettings.txt` y `z-constantsThisExp.h5`. Estos dos archivos resumen las características del experimento. El primer archivo es de texto, y lo mostramos más adelante. El segundo es un archivo `h5`, el cual dará información a cualquier programa de análisis *a posteriori* que busque extraer alguna información de esta carpeta.
- `timeeachmethod-log.dat`, un listado con el tiempo en minutos que tardó el experimento de cada optimizador. Mostramos un ejemplo a continuación.
- También, hay una carpeta para los resultados de cada optimizador, los cuales mostramos en la sección 5.3.2. Para este experimento, tenemos a:
  - `Adabelief-d/`
  - `Adam-d/`

  Podemos ver que las carpetas han sido nombradas según los valores del vector `optinoms` del archivo de optimizadores (sección 5.2.1).

5.3.1.1 Archivos ejemplo

Este archivo, `logprocess.log`, evoluciona durante el cálculo. Para este experimento, tenemos:

```
1  Process started at 2022-05-01 09:24:33
2  Initial checks passed! Experiment starts now!!
3  ====================================
4  2022-05-01 09:25:38
5  TIME FOR A NEW OPT!
6  Adam-d
7  #1 out of 2, underway now...
8  DONE!
9  Job ETA: 4 hours, 41 minutes, 14 seconds, 2 milliseconds
10 Estimated job end time: 2022-05-01 14:06:52
11
12 2022-05-01 11:46:15
13 TIME FOR A NEW OPT!
14 Adabelief-d
15 #2 out of 2, underway now...
16 ====================================
17 Please pretend this is a clever Gaussian-like quote
18 AND, IT IS DONE
19 Normal termination of Dannte at 2022-05-01 14:06:12
20 Training time:    4 hours, 40 minutes, 34 seconds
```

La evolución del archivo es:

- el archivo de entrada fue ejecutado unos segundos antes de las `09:24:33`. Eventualmente, las funciones más tempranas fueron compiladas, y se revisó la existencia y validez de las variables fundamentales del cálculo. Entonces, a las `09:24:33` aparecen las líneas 1–3 del archivo, e inician procesos como la lectura de datos, o la inicialización de las matrices de pesos.

- toda la maquinaria de entrenamiento está lista a las `09:25:38`. Da inicio el experimento con el primer optimizador, y aparecen las líneas 4–7 de este archivo.
- a las `11:46:15` ha concluido el experimento con el primer optimizador, se han guardado todos sus archivos de salida, y entonces se crea la carpeta del segundo optimizador, la cual está vacía por el momento. En este momento aparece el bloque de líneas 8–15.

  Para este momento ya podemos examinar los archivos de salida del primer optimizador, dado que este experimento es independiente del resto de optimizadores.
- el cálculo ha terminado a las `14:06:12`. Aparecen las líneas 16–20 de este archivo, los archivos propios del segundo experimento, y el resto de archivos listados en la sección 5.3.1.

El archivo `readme-experimentsettings.txt` tiene el aspecto

```
1  == Experimento kfold para para propiedades termodinámicas ==
2  model:    NN con 3 capas ocultas
3  cvalidationm: k
4  nneur_h:  26 20 14
5  niter:    2000
6  feats:    (2 <- 2) evscmgau+atomcompgau (29+5=34 <- 29+5=34)
7  thesep:   ["all", "all"]
8  labels:   zpve (7 clases)
9  nfolds:  5
10 nRepeat: 1
11 poolset: QM9 set, "pool-TT-QM9.dat"
12 opt list: list-optimizers-grad-k5.jl
13 cart (xent,mae,mse)   = F F F
14 osty (decay,corr,par) = h h c
15 began: 2022-05-01 09:25:38
16 ended: 2022-05-01 14:06:02
17 walltime: 4 hours, 40 minutes, 34 seconds
18 hostname: mixtli
19 jobfile:  /home/leon/QM9-experimentos/QM9-herramientas/mckh-kf-a-nn3-zpve-evac
   -QM9.jl
20 julia v.: 1.6.1
```

Podemos ver que mucha de la información aquí listada forma parte del archivo de entrada principal o de los `incore`. Este archivo reúne todas estas variables, y lista información adicional como el nombre de la computadora donde se realizó el cálculo, el tiempo de inicio y final del experimento, y la versión de `julia` usada.

El archivo `timeeachmethod-log.dat` lista el tiempo de entrenamiento, en minutos, para cada optimizador.

```
1  140.618359535325
2  139.965543256225
```

5.3.2 Archivos de cada optimizador

La carpeta `folderOptResults` contiene subcarpetas, cada una con los resultados de los entrenamientos de los diferentes optimizadores. En el caso de este cálculo ejemplo, la carpeta principal es `KH-kf-NN3-a-zpve-evac-QM9/`. Luego, hemos nombrado a los optimizadores en las líneas 9–12 de la lista de optimizadores (sección 5.2.1). Por lo tanto, las subcarpetas de este cálculo son `KH-kf-NN3-a-zpve-evac-QM9/Adam-d/` y `KH-kf-NN3-a-zpve-evac-QM9/Adabelief-d/`. Ejemplificaremos esta sección con los archivos de `Adabelief-d/`.

- El siguiente par de archivos da información acerca del valor de la función de costo J, la entropía cruzada binaria (sección 1.8.1), que abreviamos como `bix` (*binary X-entropy*). Mostramos a este par más adelante.

  - `bix-eachexperiment-TrainTest.dat`. Este archivo tabula renglón a renglón los valores finales de J de cada experimento individual de los 5 que componen a nuestro experimento k-fold. La primer columna da el error para el conjunto de entrenamiento, la segunda da el error para el conjunto de prueba.
  - `bix-avgdecay.dat`, este archivo muestra el promedio de la evolución de J para todos los experimentos, a lo largo de cada iteración del entrenamiento. Por lo tanto, la longitud de este archivo está dada por el valor de `niter` que establecimos en la sección 5.2.3.1.

    Estos archivos ayudan a comparar el desempeño de los modelos. De hecho, todas las gráficas del capítulo 6 están hechas a partir de estos vectores.

- En el caso general, un experimento con un optimizador dado comprende a múltiples entrenamientos independientes, los cuales forman una distribución. El archivo `bix-avgdecay.dat` contiene al decaimiento promedio, y su graficación es muy valiosa. Ahora bien, el propósito de los archivos aquí descritos es informarnos acerca de la distribución de los decaimientos, con la esperanza de poder cuantificar la dispersión de la distribución, y así poder detectar alguna propiedad objetivo peculiar donde el decaimiento promedio tenga buen aspecto, pero en realidad haya experimentos muy pobres dispersos entre algunos más adecuados.

  Por lo tanto, cada uno de estos archivos guarda uno de los decaimientos de interés. Les denominamos 'campeones' dado que cada uno maximiza o minimiza un criterio particular. Entonces, esta serie de archivos nos da una visión condensada sobre la población de decaimientos; si tuviéramos un minucioso interés en cada decaimiento por separado, podemos consultar el archivo `z-mats-errordecays.h5`.

  Algunos de estos archivos contienen las letras `N` y `M` (`normal` y `max`), las cuales son referentes a distintos criterios de clasificación, como los mencionados en la sección 4.3. En particular, el criterio `N` considera que una molécula fue clasificada correctamente cuando su clase correcta tiene puntuación > 0.5, y las incorrectas tienen < 0.5; `M` es más laxo, y solo exige que la clase correcta tenga la puntuación más grande de entre todas las clases.

  - `cp-logdecayBix-trainbest.dat` y `cp-logdecayBix-trainworst.dat`. Estos decaimientos tuvieron el mejor y peor valor final de J para el conjunto de entrenamiento.
  - `cp-logdecayBix-testbest.dat` y `cp-logdecayBix-testworst.dat`. Análogos a los dos anteriores, esta vez considerando el conjunto de prueba.
  - `cp-logdecayBix-bestclasfN.dat` y `cp-logdecayBix-bestclasfM.dat`. Estos decaimientos tienen los mejores desempeños clasificativos para los criterios `N` y `M`.
  - `cp-logdecayBix-ovfitbix.dat`. Este es el decaimiento del experimento con el mayor grado de *overfit* para la función de costo. Un experimento tiene un *overfit* cuando un modelo muestra un desempeño prometedor para el conjunto de entrenamiento, pero su desempeño para el conjunto de prueba es mucho más pobre. En términos de la función de costo, este es el experimento con el valor máximo de $(J_{\text{prueba}} - J_{\text{entrenamiento}})$, donde $J \geq 0 \; \forall \; \boldsymbol{W}, \boldsymbol{b}$, y un modelo ideal tiene $J_{\text{prueba}} = J_{\text{entrenamiento}} = 0$.
  - `cp-logdecayBix-ovfitN.dat` y `cp-logdecayBix-ovfitM.dat`. El criterio de estos decaimientos es análogo al del archivo anterior, esta vez considerando a los experi-

mentos con altas tasas de clasificación para el conjunto de entrenamiento, y pobres para el conjunto de prueba, considerando al criterio `N` y al `M`.

- Estos archivos resumen información referente a los decaimientos 'campeones' mencionados en el punto anterior.
  - `rangeofmethod.txt`, es uno de los archivos de salida más valiosos, y lo discutimos con más detalle en la sección 5.3.2.1. Este archivo de texto tiene un renglón para cada campeón y uno más con el promedio de todos los experimentos.
  - `noteworthy-indices.dat`, da un listado de los campeones de acuerdo a su índice en la lista de experimentos realizados. A partir de este vector podemos consultar los demás archivos de salida y extraer información de interés referente a algún campeón.
- Estos archivos contienen información minuciosa acerca de cada experimento. Su formato es definido en las líneas 32–35 del archivo `incore` de ejemplo (sección 5.2.3.1).
  - `z-corrclasfd-LNSM.h5`, lista las tasas de clasificación para el conjunto de prueba y entrenamiento, considerando a cuatro criterios: `N`, `M`, y dos adicionales: `L` y `S` (`l`axo y estricto). `L` exige > 0.4 para la clase correcta y < 0.6 para las incorrectas; `S` intercambia estos valores.

    Este archivo contiene la información presentada en todas las tablas del capítulo 6.
  - `z-mats-errordecays.h5`, contiene los decaimientos de J de cada uno de los experimentos, así como cualquier otra medida adicional de error que haya sido solicitada en las líneas 27–30 del archivo `incore` (sección 5.2.3.1).
  - `z-paramsEnd.h5`, contiene los parámetros finales de los modelos, es decir, sus matrices de pesos $\boldsymbol{W}$ y sus vectores de incrementos $\boldsymbol{b}$ al final del entrenamiento. A partir de este archivo se pueden realizar predicciones para muestras nuevas, o comparar nuestros modelos con otros presentes en la literatura.

5.3.2.1 Archivos ejemplo

El archivo `rangeofmethod.txt` ilustra en un vistazo la distribución y regularidad de los modelos entrenados con la configuración experimental elegida. A partir de este archivo podríamos detectar varias situaciones de interés, como por ejemplo:

- el nivel de *overfit* de los experimentos,
- estimar el error de generalización global,
- verificar la variabilidad en las capacidades predictivas de los modelos.

Este archivo lista diez renglones y siete columnas. La primer columna da el índice del experimento de interés. En el resto de columnas, las pares son referentes al conjunto de entrenamiento; las impares al de prueba. Las columnas 2 y 3 dan los J finales del modelo; las 4 y 5 dan la proporción de moléculas clasificadas correctamente (bajo `N`), y las 6 y 7 son referentes a `M`.

Luego, los renglones 1–4 dan los campeones que maximizan cierto criterio deseable, el 5 da el promedio de las propiedades para todos los experimentos, y los renglones 6–10 listan los campeones que maximizan cierto criterio indeseable (y entonces, les llamamos 'anticampeones'). Estos nueve criterios son los mismos mencionados en la serie de archivos `cp-logdecayBix` de la sección 5.3.2. Los renglones de este archivo siguen los criterios:

- 1, `trainbest`, menor valor de J para el conjunto de entrenamiento,

- 2, `testbest`, menor valor de J para el conjunto de prueba,
- 3, `bestclasfN`, mayor cantidad de moléculas clasificadas correctamente, según `N`,
- 4, `bestclasfM`, mayor cantidad de moléculas clasificadas correctamente, según `M`,
- 5, renglón con el promedio de las propiedades,
- 6, `trainworst`, mayor valor de J para el conjunto de entrenamiento,
- 7, `testworst`, mayor valor de J para el conjunto de entrenamiento,
- 8, `ovfitbix`, experimento que maximiza la diferencia entre un J de entrenamiento bajo y un J de prueba alto,
- 9, `ovfitN`, experimento que maximiza la diferencia entre una tasa de clasificación de entrenamiento alta, y una tasa de clasificación de prueba baja,
- 10, `ovfitM`, análogo al experimento anterior, ahora bajo el criterio `M`.

Este es un archivo `rangeofmethod.txt`:

```
  3   0.043823 0.044742   0.946223 0.944617   0.949173 0.947828
  5   0.043992 0.043455   0.945457 0.946895   0.948482 0.949733
  5   0.043992 0.043455   0.945457 0.946895   0.948482 0.949733
  5   0.043992 0.043455   0.945457 0.946895   0.948482 0.949733
avg   0.047211 0.047674   0.945672 0.944807   0.948581 0.946577
  2   0.049164 0.050030   0.944682 0.942040   0.948164 0.946633
  2   0.049164 0.050030   0.944682 0.942040   0.948164 0.946633
  3   0.043823 0.044742   0.946223 0.944617   0.949173 0.947828
  2   0.049164 0.050030   0.944682 0.942040   0.948164 0.946633
  1   0.043866 0.044470   0.946409 0.944878   0.949313 0.947716
```

El archivo `bix-eachexperiment-TrainTest.dat` tiene tantos renglones como haya modelos entrenados en un experimento. La primer columna da el J de entrenamiento, la segunda da el de prueba:

```
0.04386592512138025     0.04447032518943494
0.04916386754728918     0.050030132752180906
0.043822602065064394    0.04474221377770278
0.04826246030286565     0.048289731691170026
0.04399182252832678     0.04345537036334406
```

Este archivo, `bix-avgdecay.dat`, es promedio de los decaimientos de J para todos los experimentos. Estos son los vectores graficados a lo largo del capítulo de resultados, por ejemplo la figura 6.2.

```
0.7345168060961853
0.7118633518417105
0.6859258947018679
0.658361752273442
0.6304016961068036
...
0.04542531536302081
0.04540523139274812
0.04538444958800587
0.04536580098726681
```

Capítulo 6

Resultados y conclusiones

Índice de capítulo

6.1 Introducción y metas experimentales

A lo largo de esta ICR hemos revisado los diferentes componentes de un experimento ML. Por fin, en este capítulo pondremos a prueba la capacidad predictiva de diferentes redes neuronales para la clasificación molecular de acuerdo al valor de sus propiedades moleculares, y discutiremos el significado e implicaciones de los resultados. Estos entrenamientos fueron posibles gracias a la diligente asistencia de Luis Alarcón, quien habilitó y configuró las computadoras `labred250` y `mixtli` del Departamento de Matemáticas Aplicadas y Sistemas (DMAS).

Los experimentos aquí mostrados son los mismos que los que reportamos en `arXiv` [1]; en esta ICR los tratamos con mayor detalle.

Hemos realizado nuestros experimentos en el programa `Dannte` ([120], capítulo 5), el cual desarrollamos como un marco de investigación para explorar problemas aplicados de aprendizaje de máquina, en este caso a la química computacional. El programa está enteramente escrito en el lenguaje `julia` [121], el cual está orientado a la programación científica. Parte de nuestra maquinaria más fundamental de aprendizaje de máquina emplea al paquete `Flux.jl` [42, 43]. También usamos otros paquetes y bibliotecas [122-130].

El objetivo de estos experimentos es comparar la dificultad predictiva de cuatro propiedades termodinámicas (H, G, ZPVE, C_V) y la efectividad de cinco descriptores (AC, EV, SCM, y sus

concatenaciones). Cada una de estas representaciones tiene diferente tamaño, y codifica la estructura molecular de diferente manera, a distintos niveles de resolución. De acuerdo al planteamiento de la sección 3.8, consideramos que AC y EV son de *resolución baja*, mientras que SCM es de *resolución alta*.

Luego, denotamos a la concatenación de EV y AC como EV + AC. Esta representación es *mixta* y de resolución baja, mientras que la concatenación de SCM y AC, SCM + AC, es de resolución alta.

Esperamos que el orden de desempeño de las representaciones corresponda a su orden por resolución. También, esperamos que las representaciones aumentadas EV + AC y SCM + AC tengan mejor desempeño que sus componentes, que denominamos representaciones *standalone*. ¿Qué tan mejor? Si la representación EV + AC alcanzara un desempeño semejante al de SCM, entonces tendríamos evidencia de que las representaciones EV y AC describen aspectos estructurales distintos, y que su unión alcanza una riqueza semejante a la de SCM. Bajo este supuesto, habríamos encontrado que EV + AC presenta una calidad cercana a la de SCM, a un costo computacional significativamente menor. Esto significaría que para ciertas aplicaciones, la representación SCM es innecesaria, y es suficiente con EV + AC para entrenar a nuestros modelos.

Por otra parte, comparar el desempeño de las dos representaciones de resolución alta nos informa acerca del traslape entre las representaciones SCM y AC: si la representación mixta tiene un desempeño significativamente mejor, entonces el traslape es pequeño y las dos representaciones aportan información independiente y valiosa.

Este es el tipo de deducciones que esperamos poder encontrar acerca de las propiedades de entrada y salida del modelo: conocimiento de carácter fisicoquímico como el vínculo entre las propiedades moleculares objetivo y los descriptores y la forma en que codifican la geometría, la relevancia informacional de cada descriptor en una representación mixta, el tipo de molécula con una mejor o peor relación con cierto descriptor o propiedad objetivo, etcétera.

Dado el *modus operandi* de la fisicoquímica teórica, consideramos que este tipo de resultados son más redituables a largo plazo que, digamos, encontrar la arquitectura ideal de una NN que predice alguna propiedad objetivo con un buen desempeño, puesto que nuestros experimentos generan conocimiento *interpretable* y podemos transferirlo a experimentos futuros que exploren configuraciones experimentales semejantes, tal vez con metas predictivas más estrictas, diferentes optimizadores o descriptores, o una base de datos más ambiciosa y diversa.

En la siguiente sección enlistamos el resto de componentes de los experimentos, y el motivo detrás de cada uno, y la sección de esta ICR donde explicamos el tema a mayor profundidad.

6.2 Configuración e hiperparámetros de los experimentos

Los componentes de nuestros experimentos son:

1. **Método ML**, hemos elegido a las redes neuronales dada su versatilidad y eficacia en diferentes áreas de la ciencia, incluyendo la química computacional. Las explicamos en las secciones 1.3 hasta 1.6.

2. **Función de activación**, dado que hemos decidido trabajar con experimentos de clasificación, consideramos a la función de activación logística(z) (sección 1.5).

3. **Método de inicialización de parámetros**. Dado el carácter clasificativo de nuestros modelos, y la identidad de nuestra función de activación, usamos la inicialización Xavier (sección 1.7.1).

4. **Función de costo** $J(\boldsymbol{W}, \boldsymbol{b})$. En experimentos de clasificación como los nuestros estamos obligados elegir a la entropía cruzada binaria (sección 1.8.1) dado que es una de las mejores opciones.

5. **Método de validación**. A fin de robustecer nuestros resultados, usamos la validación k-fold con $k = 5$ (sección 1.9.2) en lugar del más simple método *holdout*.
6. **Base de datos**. Usamos la base QM9 dada su diversidad química, riqueza de propiedades, y accesibilidad (sección 2.3).
7. **Descriptores moleculares**.
   - **Resolución alta**: Dada su formulación transparente y clara, hemos elegido a la matriz ordenada de Coulomb (SCM, sección 3.6.2), y la representación mixta SCM + AC, la cual concatena a SCM y AC.
   - **Resolución baja**: los eigenvalores de SCM (EV, sección 3.6.3), la composición atómica (AC, sección 3.7), y EV + AC, la concatenación de EV y AC.
8. **Propiedades objetivo**. Nuestras cuatro propiedades objetivo son termodinámicas, dado que los descriptores EV y SCM fueron originalmente concebidos para la predicción de una propiedad termodinámica: la energía de atomización. Además, estas cuatro propiedades han mostrado buenos desempeños para los descriptores moleculares que elegimos:
   - Entalpía, H, sección 6.4,
   - Energía libre de Gibbs, G, sección 6.5,
   - Energía vibracional de punto cero, ZPVE, sección 6.6,
   - Capacidad calorífica a volumen constante, C_V, sección 6.7.
9. **Tratamiento de datos y discretización del espacio de propiedades**. Hemos estandarizado las distribuciones de las propiedades objetivo (sección 4.2), luego dividido el espacio en 25 intervalos, resultando en 5 clases para H y G, 7 para ZPVE y C_V (sección 4.3).
10. **Criterio de clasificación**. Como mencionamos en la sección 4.3, consideramos que una molécula está bien clasificada cuando la NN asigna a la clase correcta la puntuación más grande de entre todas las clases.
11. **Optimizadores y sus hiperparámetros**. Hemos usado dos optimizadores de gradiente para entrenar nuestros modelos:
    - Adam (sección 4.4.2), dada su formulación teórica robusta y gran popularidad en el área,
    - AdaBelief (sección 4.4.3), una variante reciente de Adam, con una fundamentación elegante pensada en acelerar los entrenamientos,
    - ambos optimizadores usan los mismos hiperparámetros, y los autores de ambos coinciden en los valores sugeridos, que hemos optado por usar: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1 \times 10^{-8}$.
12. **Cantidad de iteraciones**. Hemos encontrado que 2000 iteraciones tienen un costo computacional aceptable, y bastan para obtener un aprendizaje significativo, tras el cual podemos comparar los diferentes experimentos, descriptores, y optimizadores.
13. **Arquitectura de las redes neuronales**.

    Aspectos generales

    La arquitectura de la red a entrenar, es decir, su cantidad de capas y el número de neuronas de cada una, es otro aspecto de crucial importancia en el éxito de un experimento. Esta no es una decisión trivial: no existe un método analítico que arroje la arquitectura ideal para un

problema dado. Una solución conveniente es el seguir el ejemplo de estudios predecesores, si existieran. Por otra parte, una solución metódica es la disciplina *neural network architecture search* (NAS), que engloba diferentes métodos para automatizar esta exploración [135, 136].

Una forma más es el uso de criterios heurísticos, con los cuales podemos obtener arquitecturas candidatas razonables. Ejemplos de estos criterios son:

- la NN debe tener forma de embudo, donde la cardinalidad de las capas disminuye según nos acercamos a la capa de salida, minimizando así las diferencias en tamaño entre capas vecinas, y añadiendo capas ocultas según sea necesario para atenuar la reducción de cardinalidad.
- un modelo entrenado debe tener un desempeño semejante para el conjunto de entrenamiento y el de prueba. Cuando esto sucede, sabemos que el modelo no está sujeto a un *overfit* ni a un *underfit* significativo, y ha logrado aprender los patrones y características en las entradas que influyen en las salidas.

Si cumplimos con estos criterios, podemos considerar que la arquitectura elegida es adecuada para nuestro problema.

Arquitecturas elegidas

Al decidir cuáles descriptores usaremos en nuestros experimentos, estamos eligiendo también el tamaño de la capa de entrada. En nuestro caso, tenemos longitudes de 5 a 440. Este intervalo es demasiado amplio como para usar una arquitectura común en todos los experimentos. Por lo tanto, hemos seguido los criterios heurísticos recién mencionados, y tras múltiples experimentos hemos llegado a modelos con tres capas ocultas, cuyas arquitecturas dependen del tamaño de la capa de entrada:

- AC, `inp:12:10:8:out`,
- EV y EV + AC, `inp:26:20:14:out`,
- SCM y SCM + AC, `inp:240:120:60:out`.

donde `inp` es la longitud de la representación (5, 29, y 435 para las representaciones *standalone*), y `out` es la cantidad de clases de la propiedad a predecir (5 para H y G, 7 para ZPVE y C_V).

6.3 Herramientas de análisis

6.3.1 Análisis de un experimento

Dos de los resultados más importantes de un experimento ML de clasificación, debido a la información que aportan, son:

- la *learning curve* (LC), por ejemplo la figura 6.2, es la gráfica de la evolución del costo para el conjunto de entrenamiento, a lo largo del avance del entrenamiento. Al graficar experimentos con diferentes optimizadores, descriptores, o arquitecturas, podemos comparar la velocidad del entrenamiento de cada experimento, su desempeño al final del entrenamiento, las regiones donde el descenso de J puede estancarse (a lo cual hemos denominado barreras de error) y si fue posible vencer estas barreras, entre otros indicadores de desempeño.

  Por ejemplo, la figura 6.1 muestra tres LC ejemplo para las representaciones A, B, C. Podemos ver una barrera de error en la región $J \approx 0.4$. Apegándonos al vocabulario habitual de la literatura ML, hemos denominado la cantidad de iteraciones de entrenamiento con la palabra *epoch*.

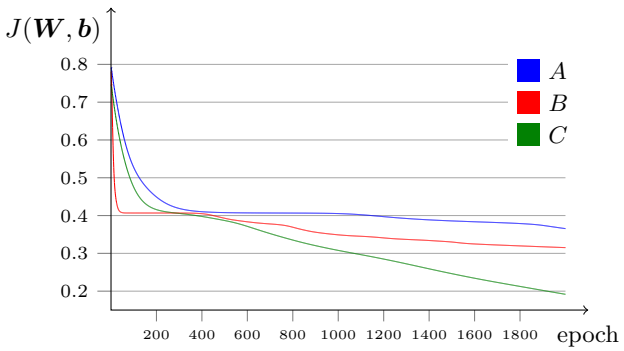


Figura 6.1: Ejemplo de *learning curves* de tres modelos para una propiedad dada.

- la tabla de tasas clasificativas para el conjunto de prueba al final del entrenamiento. Mientras que la *learning curve* muestra cuánto fue posible aprender del conjunto de entrenamiento, la tabla muestra algo más general: el nivel de efectividad del modelo para un conjunto de moléculas ajenas al entrenamiento. Esto es, por lo tanto, una aproximación al *error de generalización* del modelo, la medida del error esperado para predicciones de muestras futuras, algunas de las cuales podrían obtenerse años después de finalizado el entrenamiento.

  Por ejemplo, la tabla 6.1 compara diferentes descriptores y optimizadores para una propiedad dada. Por otra parte, podríamos encontrarnos con un experimento que muestra una gran dispersión en las tasas globales, como pasa con la columna de Opt_1 de la tabla 6.1. Esto nos indica que podemos adquirir información fisicoquímica a partir de una presentación más granular. Esto sucede en la tabla 6.2, en la cual mostramos las tasas de cada clase por separado. Esta representación puede revelar patrones y peculiaridades indetectables en la tabla original.

	Optimizador	
Descriptor	Opt_1	Opt_2
A	0.5754	0.6891
B	0.9315	0.9934
C	0.8783	0.9218

Tabla 6.1: Ejemplo de tabla de tasas clasificativas para el conjunto de prueba, para una propiedad objetivo dada.

	Opt_1				
Descriptor	tasa g.	c_1	c_2	c_3	c_4
A	0.5754	0.5058	0.3281	0.6831	0.7341
B	0.9315	0.7139	0.5532	0.9347	0.9912
C	0.8783	0.6839	0.4522	0.8543	0.8943

Tabla 6.2: Ejemplo de tabla de tasas clasificativas globales (tasa g.) y de cada clase, para el conjunto de prueba, para una propiedad objetivo dada y el optimizador Opt_1.

La comparación entre los resultados recién mencionados puede ser muy reveladora: un modelo ideal presenta un error final muy bajo en su LC, y una tasa clasificativa alta. Sin embargo, si

obtuviéramos errores bajos en la LC, pero la tabla mostrara tasas bajas, entonces concluimos que el modelo está experimentando un *overfit*, pues ha aprendido las correlaciones particulares entre las entradas y las salidas, pero se ha especializado en las muestras que hemos usado durante el entrenamiento, en lugar de tener un aprendizaje general, uno con un mayor nivel de abstracción, que le permita dar un buen tratamiento a muestras futuras. Este problema se puede solucionar al incrementar la cantidad de neuronas y capas del modelo, o enriquecer el conjunto de entrenamiento, mediante aumentar su tamaño o diversificar sus muestras. Una opción más es que el descriptor usado no esté suficientemente relacionado con las propiedades de salida, o tal vez no tenga la resolución suficiente para describir fielmente la propiedad objetivo.

6.3.2 Métodos de comparación entre experimentos

Si estamos interesados en estudiar la dificultad predictiva de incluso pocas propiedades, tomando en cuenta múltiples descriptores y sus diferentes concatenaciones, el escalamiento de la cantidad de gráficas y tablas se volverá inmanejable y su interpretación será difícil. A fin de mitigar este factor, hemos usado dos técnicas para agilizar la extracción de información fisicoquímica y llegar a *metapredicciones* sobre experimentos futuros, las cuales explicaremos más adelante.

La primera de estas herramientas debe su nombre al contexto químico de nuestros experimentos, y le denominamos *análisis Hammond-like*; la segunda es el *análisis per-class*. Las explicamos a continuación y les damos uso en las siguientes secciones.

Análisis *Hammond-like* (AHL)

Sea A una representación *standalone* y M una representación mixta que incluye a A. Podemos usar al análisis *Hammond-like* para estimar cuán influyente es A en M, lo cual es una medida de la originalidad de la información de A, comparada a las demás representaciones $\in M$.

Este análisis es conceptualmente sencillo, pero nos brinda información químicamente valiosa. Su formulación proviene del postulado de Hammond [137], una hipótesis muy usada en estudios de reactividad química,

> *If two states, as for example, a transition state and an unstable intermediate, occur consecutively during a reaction process and have nearly the same energy content, their interconversion will involve only a small reorganization of the molecular structures.*

En la práctica, el postulado suele emplearse en forma de este corolario:

> Cuando un estado de transición (TS) conecta a dos mínimos, TS tiene un mayor parecido estructural al mínimo cuya energía es la más cercana a la de TS.

Finalmente, en nuestro contexto ML, el principio detrás del análisis *Hammond-like* es:

> Si, considerando a todas las representaciones $\{A, B, C, \ldots\}$ que componen a M, tenemos que el desempeño de A es el más parecido al de M, entonces A es el descriptor que aporta la información más original o significativa presente en M.

Análisis *per-class* (APC)

Este análisis presenta de manera más granular a las tasas clasificativas. El APC es valioso cuando las puntuaciones de las clases presentan una dispersión grande. Esperamos que las mejores clases contengan las estructuras y grupos funcionales en las que el descriptor tiene una relación más cercana con la propiedad objetivo; las peores clases indican lo contrario. También, esta tabulación detallada es de suma utilidad si estuviéramos interesados en ensamblar una representación mixta,

dado que es más productivo enfocarnos en las clases más débiles que en la tasa global; los incrementos en el desempeño de un modelo al incluir representaciones nuevas son mucho más visibles en la representación granular, mientras que en la global podrían pasar inadvertidos.

6.4 Entalpía, H

Si consideramos la forma en que hemos dividido en clases a la población de la entalpía de QM9, y observamos los resultados de los modelos de entalpía (figura 6.2 y la tabla 6.3), podemos decir que esta propiedad está muy correlacionada con las representaciones moleculares usadas. Es decir, los modelos de H muestran un desempeño bastante alto: los errores finales son ≈ 0 y varias tasas clasificativas son > 0.99. A fin de facilitar la discusión de nuestros resultados, decimos que H es una propiedad *benigna*.

Comparemos a los dos optimizadores. A partir del decaimiento de J y la tabla clasificativa vemos que:

- las *learning curves* de AdaBelief son coherentes con sus resultados clasificativos: este optimizador llega a desempeños consistentemente buenos sin importar el descriptor usado.
- Adam tiene desempeños comparablemente buenos para los descriptores de resolución alta, pero es particularmente pobre para los otros tres descriptores.

Estos resultados ilustran la importancia de la elección del optimizador para experimentos de resolución baja: el optimizador correcto puede lograr, con los descriptores más pequeños, desempeños casi a la par de los descriptores más detallados. Notemos también, los modelos de AdaBelief y resolución baja tardan cerca de 1800 iteraciones en llegar a errores ≈ 0, mientras que los de resolución alta tardan ~ 800 iteraciones. Esto es, los modelos más pequeños requieren de más iteraciones para terminar su aprendizaje, pero estas iteraciones tienen un costo computacional menor, dadas las diferencias en la cantidad de neuronas en los modelos (y por lo tanto, en parámetros a almacenar y optimizar a lo largo del entrenamiento).

Vemos también que la representación AC es bastante capaz para estos experimentos con H: para los modelos Adam, la tasa clasificativa de AC es mejor que la de EV; con AdaBelief está casi a la par de SCM y SCM + AC. También, los decaimientos del error de las representaciones mixtas son más acelerados que los de EV y SCM, lo cual puede ahorrar tiempo de cálculo.

Como corolario, tenemos que la elección de un mejor optimizador es una manera sencilla de lograr mejores entrenamientos desde el punto de vista matemático. Luego, si fuera impracticable entrenar con otro optimizador, podemos proceder desde un ángulo fisicoquímico y crear una representación mixta tomando descriptores establecidos y concatenándole otros descriptores que proporcionan información nueva y físicamente relevante para nuestra propiedad objetivo. Esto es especialmente claro en la columna de Adam de la tabla 6.3: la representación mixta EV + AC logra una tasa bastante mejor que la de sus dos representaciones componentes.

En conclusión, los dos optimizadores son valiosos, de diferente manera:

- los entrenamientos de AdaBelief son estrictamente superiores,
- Adam es sumamente valioso si buscamos conocer la relevancia de cada una las representaciones *standalone* en una representación mixta.

  Por ejemplo, Adam nos muestra que la puntuación de AC es bastante más cercana a la de EV + AC que la de EV. Siguiendo el argumento *Hammond-like*, concluimos que AC es más importante que EV para la predicción de H. Por otra parte, es imposible realizar este argumento si solo hubiéramos experimentado con AdaBelief, dadas sus puntuaciones tan homogéneas > 0.99.

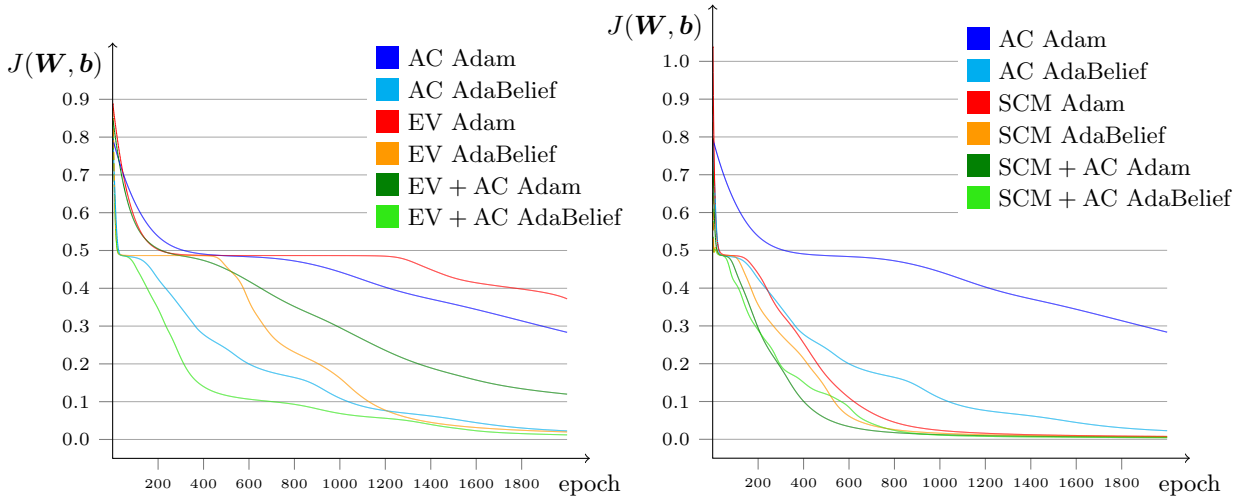


Figura 6.2: Decaimiento del error de los modelos H, para el conjunto de entrenamiento.

	Optimizador	
Descriptor	Adam	AdaBelief
AC	0.7279	0.9921
EV	0.5754	0.9915
EV + AC	0.8783	0.9939
SCM	0.9943	0.9947
SCM + AC	0.9948	0.9953

Tabla 6.3: Tasa de clasificación de los modelos H, para el conjunto de prueba.

6.5 Energía libre de Gibbs, G

Tenemos algo de información *a priori* acerca de la distribución de G en QM9: la tabla 4.4 y la figura 4.1b muestran una gran semejanza entre las distribuciones de H y de G. Esta es la razón de que ambas propiedades tengan la misma cantidad de clases (5), y de que sus cotas sean tan parecidas. Partiendo de esta información, podríamos hacer la hipótesis que los resultados de G sean idénticos a los de H.

Ahora, habiendo concluido los experimentos, podemos verificar esta hipótesis. Encontramos que hay semejanzas entre G y H: los descriptores de resolución alta logran tasas clasificativas > 0.99 para ambos optimizadores y errores ≈ 0. En particular, los experimentos de resolución baja con AdaBelief tienen un desempeño casi tan bueno como los de resolución alta, mientras que los de Adam quedan visiblemente detrás, con una clara dispersión dependiendo del descriptor usado. Dado este desempeño tan alto, podemos decir que G es una propiedad benigna, al igual que hicimos para H.

Por otra parte, encontramos algunas diferencias entre G y H:

- Mientras que el modelo AC entrenado con Adam tiene un desempeño parecido para H y G (tasas de clasificación de 0.73 y 0.76), el modelo EV-Adam muestra una tasa clasificativa para G muy superior a la de H, pasando de 0.5754 a 0.8987. Esto es, AC rebasa a EV cuando se trata de la predicción de G, invirtiendo los papeles que encontramos en H.

Este fenómeno también sucede con AdaBelief, aunque es menos perceptible. Esto sugiere que la representación EV está más relacionada con G que con H, mientras que AC se mantiene constante para las dos propiedades.

- Dado que AC es igualmente bueno para ambas propiedades, mientras que EV es mucho más relevante para G, tenemos también que el desempeño de EV + AC es mejor para G que para H. Por lo tanto, el análisis *Hammond-like* muestra que EV es la representación más importante dentro de EV + AC.

Estos cambios en los desempeños clasificativos, aunados a las diferencias en los decaimientos entre H y G (especialmente los de EV-Adam), muestran que nuestra hipótesis sobre la equivalencia de H y G es falsa: estas dos propiedades no son idénticas, aunque es cierto que sus distribuciones son similares.

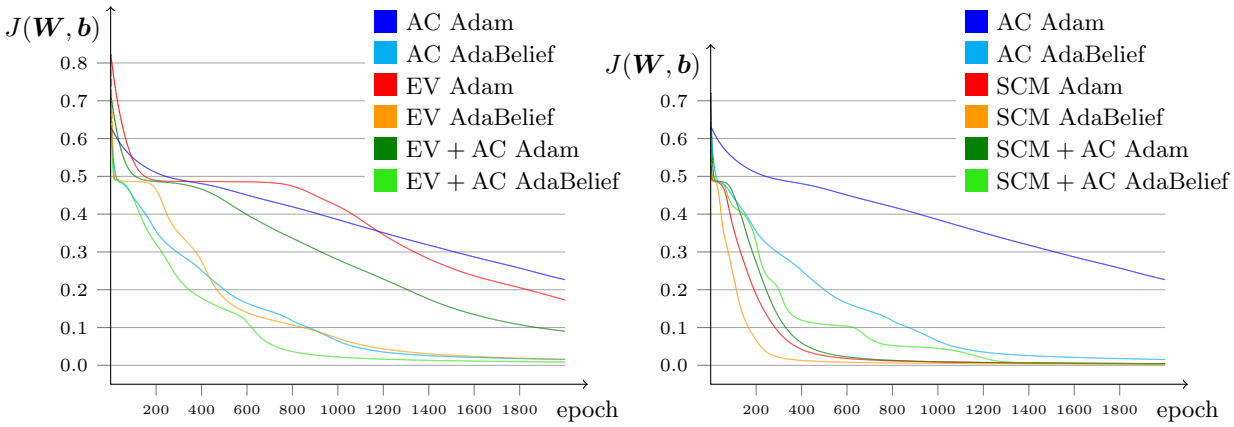


Figura 6.3: Decaimiento del error de los modelos G, para el conjunto de entrenamiento.

	Optimizador	
Descriptor	Adam	AdaBelief
AC	0.7580	0.9913
EV	0.8987	0.9932
EV + AC	0.9359	0.9952
SCM	0.9954	0.9955
SCM + AC	0.9959	0.9959

Tabla 6.4: Tasa de clasificación de los modelos G, para el conjunto de prueba.

6.6 Energía vibracional de punto cero, ZPVE

Esta propiedad presenta una mayor dificultad que las dos anteriores, y está dividida en siete clases en lugar de cinco (tabla 4.5, figura 4.2a).

ZPVE presenta algunas semejanzas cualitativas con las dos propiedades anteriores: los modelos de resolución alta tienen el mejor desempeño, mientras que los de resolución baja obtienen resultados mucho mejores con AdaBelief que con Adam. Finalmente, el análisis *Hammond-like* muestra que AC es la representación más significativa dentro de EV + AC.

Por otra parte, la mayor dificultad de esta propiedad se manifiesta tanto en las tasas clasificativas (ninguno de los modelos llega a > 0.99) y las *learning curve* (ninguno de los optimizadores o descriptores llega a valores tan bajos como para H o G). Los modelos EV-Adam y EV-AdaBelief son de particular interés: tomando en cuenta a H, G, y ZPVE, este último presenta la única tasa clasificativa < 0.5. Esto es, es más probable que una molécula sea clasificada incorrectamente que correctamente.

Las tasas finales muestran una dispersión mucho mayor a las dos propiedades anteriores. A fin de poder entender las clases detrás de estas puntuaciones más bajas, emprendemos el análisis *per-class*: tabulamos las tasas globales y también las de las dos clases con puntuaciones más bajas (tabla 6.5).

Esta presentación granular nos muestra el desempeño de las representaciones con respecto tanto a la tasa global como las clases difíciles, que son las más influyentes en que las puntuaciones globales sean comparablemente bajas. Revisándoles podemos identificar las representaciones que complementan con información nueva a una representación existente. En este caso, tenemos que EV tiene un mejor desempeño para la clase 6 (c_6) que para todo QM9, al contrario de AC. Esto sugiere que EV es una representación valiosa para añadir a otras representaciones. Esto se ve reflejado en el renglón de EV + AC, que casi alcanza a las representaciones de resolución alta en su tasas c_6 y global.

Esta técnica, el APC, tiene dos resultados generales:

- *un criterio para refinar las representaciones usadas*, una consecuencia natural de esta tabulación es el identificar las fortalezas de cada representación, y las concatenaciones más benéficas que podemos obtener con ellas. Este conocimiento de mejores representaciones halladas es *transferible* a experimentos futuros, los cuales pueden tener fines exploratorios y poner a prueba otras arquitecturas u optimizadores. Alternativamente, los experimentos podrían tener un mayor grado de ambición, y tratar otras bases de datos de mayor cardinalidad o diversidad química; o un mayor grado de precisión, y trabajar con más clases, o metas predictivas más estrictas.
- *entender de mejor manera la interacción entre algunas clases moleculares y las propiedades del experimento*, una consecuencia natural de nuestros experimentos clasificativos es el encontrar las clases con menor o mayor relación a los descriptores moleculares del experimento. Las clases con puntuaciones particularmente bajas o altas pueden contener moléculas con características estructurales comunes; identificarlas puede resultar en un entendimiento profundo de la relación entre un grupo funcional dado y alguno de los descriptores, optimizadores, o propiedades objetivo.

Estas predicciones sobre experimentos futuros son las de los resultados más valiosos que podemos obtener de nuestros experimentos. Les denominamos *metapredicciones* dado que a lo largo de esta ICR hemos usado la palabra *predicción* para referirnos a un proceso computacional de la forma $y = \text{modelo}(\boldsymbol{x})$ donde $y \in \mathbb{R}$ y conocemos su valor. Tratamos este tema con más detalle en la sección 6.7.2.

6.7 Capacidad calorífica a volumen constante, C_V

C_V es más difícil que las propiedades anteriores. Esto nos ha llevado a uno de los planteamientos más valiosos en esta ICR: un método para informada y sistemáticamente reunir los descriptores que resuelven el problema de la predicción de propiedades. Veremos este tema en la sección 6.7.2; en la 6.7.1 veremos los resultados de C_V análogos a la mayoría de resultados presentados para las propiedades anteriores.

Al igual que ZPVE, C_V está dividida en siete clases (tabla 4.5, figura 4.2b). Presentamos las habituales curva de aprendizaje en la figura 6.5 y la tabulación de tasas en la tabla 6.6, las cuales

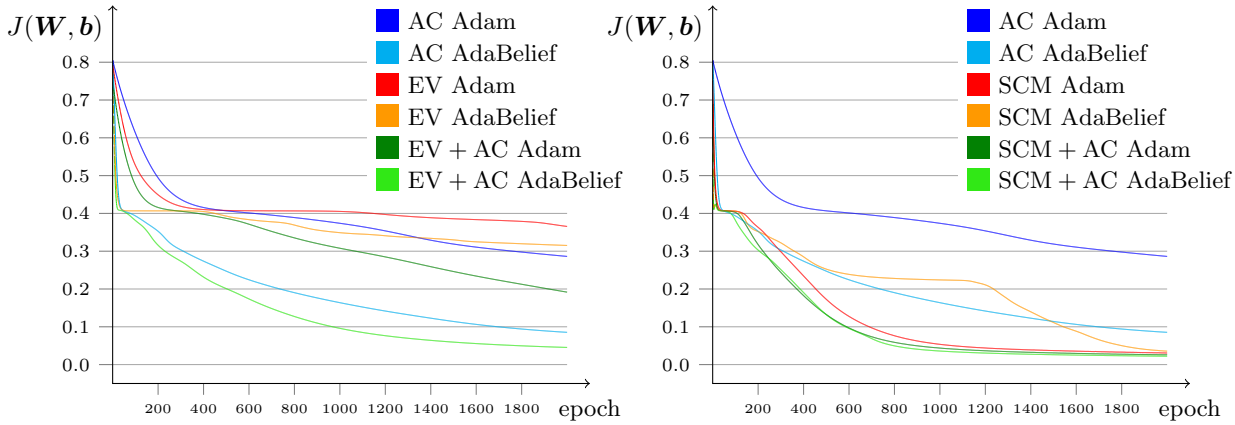


Figura 6.4: Decaimiento del error de los modelos ZPVE, para el conjunto de entrenamiento.

	Tasa global		AdaBelief	
Descriptor	Adam	AdaBelief	c_5	c_6
AC	0.5069	0.9106	0.8878	0.6686
EV	0.3460	0.4302	0.2830	0.4717
EV + AC	0.7867	0.9480	0.8959	0.8821
SCM	0.9573	0.9550	0.9161	0.9070
SCM + AC	0.9631	0.9644	0.9307	0.9349

Tabla 6.5: Tasa de clasificación de los modelos ZPVE, para el conjunto de prueba. c_5 y c_6 son las clases mostradas en la tabla 4.5.

presentan una mayor dispersión que las vistas en las propiedades anteriores. Por lo tanto, tabulamos también a cada una de las clases bajo el optimizador AdaBelief en la tabla 6.7.

6.7.1 Predicciones a partir de las salidas

C_V es la propiedad con más particularidades. Empecemos por las semejanzas con las propiedades anteriores: AdaBelief tiene mejores desempeños que Adam, especialmente para los modelos de resolución baja. También, con la excepción de G, el análisis *Hammond-like* muestra que EV es un descriptor más relevante que AC.

En cuanto a las diferencias, la dificultad de C_V es mayor, y esto se refleja en tasas clasificativas menores y errores de predicción mayores, ambos de los cuales presentan una dispersión mayor, tanto para Adam como AdaBelief, y esta variabilidad logra que los resultados de AdaBelief también sean compatibles con el AHL, no solo Adam. En términos concretos, ninguno de los modelos de resolución baja alcanza errores de 0.2; los de resolución alta se mantienen > 0.15, y las tasas de clasificación son $\in (0.25, 0.74)$, mientras que para H y G la mayoría de tasas eran > 0.99, y para ZPVE eran > 0.91.

Finalmente, C_V es más sensible a los descriptores: las propiedades anteriores tienen puntuaciones semejantes para EV + AC, SCM, y SCM + AC, a pesar de las grandes diferencias en granularidad y tamaño entre los descriptores. Esto es especialmente claro bajo AdaBelief, donde las diferencias para las propiedades anteriores son $< 1\,\%$. Para C_V, por otra parte, la tabla 6.6 muestra una brecha evidente entre EV + AC y las dos representaciones de resolución alta, para *ambos* optimizadores.

Esta variabilidad mayor nos da la oportunidad de extraer más información de los resultados y entender de manera más profunda la relación entre C_V y los diferentes descriptores. Para esto, extenderemos el formato aumentado mostrado en la tabla 6.5: obtenemos la tabla 6.7 donde mostramos clase a clase las tasas de AdaBelief. A partir de esta presentación granular podemos realizar el análisis *per-class*.

Aquí podemos ver que, por ejemplo, AC y EV tienen sus mejores tasas para las clases extremas c_1 y c_7 ($> 68\,\%$). Luego, las clases intermedias muestran un desempeño muy reducido, siempre con tasas < 0.5 e incluso llegando a valores < 0.1.

Basta con la tabla global (6.6) para percatarnos de que la representación mixta EV + AC tiene un desempeño notablemente mejor que sus dos representaciones componente. Luego, si quisiéramos entender la razón de la mejora, hemos de tornarnos hacia la tabla granular (6.7), donde podemos notar que:

- las puntuaciones de EV + AC son mejores que las de sus dos componentes, para todas las clases,
- las clases extremas alcanzan un desempeño casi a la par de las representaciones de resolución alta,
- las clases intermedias excepto c_6 mejoran significativamente, alcanzando tasas > 0.51,
- c_6 pasa de puntuaciones ≈ 0 a casi 0.44.

De aquí concluimos que las moléculas de las clases extremas tienen una relación más clara entre su C_V y sus eigenvalores y composición atómica, mientras que la relación es más nebulosa para las clases intermedias. Se podría atribuir este comportamiento a la amplitud de las clases: las extremas son más amplias y permisivas, y basta con estas representaciones sencillas para predecir aproximadamente su C_V. Luego, las clases intermedias son más estrechas, y es necesario considerar más factores para poder realizar una predicción. Este es un ejemplo conveniente de las deducciones de carácter fisicoquímico que podemos extraer de la tabulación granular.

Estos desempeños más pobres, en realidad, son sumamente valiosos, dado que nos dan información nueva que podemos usar para elucidar experimentos futuros. Detallamos el cómo en la siguiente sección.

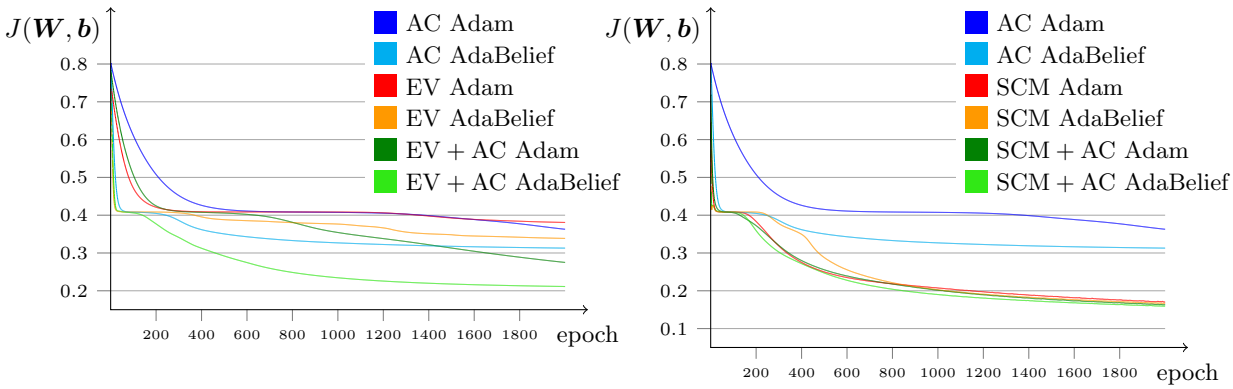


Figura 6.5: Decaimiento del error de los modelos C_V, para el conjunto de entrenamiento.

	Optimizador	
Descriptor	Adam	AdaBelief
AC	0.3217	0.4174
EV	0.2548	0.3683
EV + AC	0.5287	0.6185
SCM	0.7143	0.7213
SCM + AC	0.7285	0.7371

Tabla 6.6: Tasa de clasificación de los modelos C_V, para el conjunto de prueba.

	AdaBelief							
Descriptor	tasa g.	c_1	c_2	c_3	c_4	c_5	c_6	c_7
AC	0.4174	0.6884	0.0879	0.4956	0.3786	0.3423	0.0178	0.7424
EV	0.3683	0.7422	0.0367	0.2732	0.2274	0.3816	0.0000	0.7680
EV + AC	0.6185	0.8349	0.5670	0.5500	0.5363	0.5156	0.4398	0.8409
SCM	0.7213	0.8734	0.6768	0.6676	0.6511	0.6489	0.6320	0.8755
SCM + AC	0.7371	0.8804	0.6856	0.6871	0.6828	0.6598	0.6374	0.8963

Tabla 6.7: Tasa de clasificación global (tasa g.) y de cada clase, para el conjunto de prueba, de los modelos C_V entrenados con AdaBelief.

6.7.2 Metapredicciones a partir de los descriptores

C_V nos da la oportunidad de hallar varias *metapredicciones* valiosas acerca de experimentos futuros. Usamos el término *metapredicción* dado que a lo largo de esta ICR hemos usado la palabra *predicción* para referirnos a un proceso computacional de la forma $y = \text{modelo}(\boldsymbol{x})$ donde $y \in \mathbb{R}$ y conocemos su valor. Por otra parte, las metapredicciones son pronósticos que esperamos observar en experimentos *futuros*, y no son entidades tangibles o comparables, como sí lo es la predicción $y \in \mathbb{R}$ de la ecuación anterior.

La tabulación granular (6.7) nos presenta algunas de estas deducciones. Por ejemplo, ¿cuáles son las fortalezas y debilidades de nuestros descriptores? Esta información tiene implicaciones fisicoquímicas profundas, dado que esto nos permite informadamente construir una representación mixta de calidad, donde se aprovechen las capacidades de las diferentes representaciones componente.

Observemos las clases c_2 y c_6: ambas tienen tasas muy bajas para EV y AC, pero son las que mejor responden a la representación mixta EV + AC, casi alcanzando a las representaciones de resolución alta. En otras palabras, estas clases son las más influyentes en la mejoría predictiva de EV + AC. Podemos ver que los decaimientos de J (figura 6.5) son consistentes con esta conclusión: EV + AC rápidamente se separa de las curvas de sus componentes. Incluso, la curva de AdaBelief alcanza la vecindad de $J = 0.2$, al igual que las representaciones de resolución alta. Sin embargo, notemos que la *learning curve* no nos permite observar el vínculo entre la mejoría en el desempeño de EV + AC y las clases c_2 y c_6, pero sí nos permite confirmar este hallazgo.

Si bien hemos tenido éxito en el análisis de las representaciones por separado, como al encontrar que la clase c_6 de ZPVE sugiere que EV es una representación benéfica para adición a otras representaciones, vemos con este caso de c_2 y c_6 de C_V que pronosticar el desempeño de representaciones mixtas a partir de solamente el desempeño de varias representaciones componentes por separado, puede hacernos descartar algunos conjuntos sinérgicos de descriptores.

También hay una metapredicción-hipótesis, que puede guiar experimentos futuros: *todas* las

representaciones de nuestros experimentos tienen mejor desempeño en las clases extremas que en las intermedias. Por lo tanto, experimentos futuros podrían sondear representaciones nuevas, buscando alguna con mejor desempeño para alguna clase intermedia. Esperamos que unirla con alguna de las aquí probadas generará una representación mixta altamente efectiva.

Considerando particularmente a c_2 y c_6, podríamos probar la unión de EV o AC con esta representación nueva, a la que denominaremos B. Dado que EV y AC son cercanas a las otras tres representaciones para el resto de clases, la B correcta podría lograr representaciones AC+B y EV+B con un desempeño comparable al de las tres mejores representaciones de nuestros experimentos. Este es un ejemplo de las perspectivas que pueden otorgarnos un entendimiento estructural muy valioso sobre cuáles son las representaciones que mejor conectan a una clase molecular dada y una propiedad fisicoquímica particular.

6.8 Conclusiones y trabajo a futuro

Características de nuestros experimentos

A lo largo de esta ICR hemos explorado la intersección del aprendizaje de máquina con la química computacional. También, hemos prestado particular atención a la relación entre cuatro propiedades termodinámicas y los descriptores 'matriz de Coulomb ordenada' y 'composición atómica'; hemos realizado múltiples experimentos de clasificación involucrando a estas propiedades.

Nuestros experimentos revelan información fisicoquímica valiosa: podemos tomar las *predicciones* de los modelos para entender, por ejemplo, la efectividad y riqueza informacional de cada descriptor molecular, sus concatenaciones efectivas para una propiedad objetivo dada, la dificultad relativa de cada propiedad, y la aptitud de cada optimizador. También, el análisis conjunto de estos resultados nos provee de *metapredicciones*, conocimiento de carácter general, transferible a experimentos futuros que podemos proponer a partir del entendimiento aquí obtenido. Estos experimentos futuros pueden explorar una región más grande del espacio de compuestos orgánicos, buscar metas de precisión más altas, o buscar predicciones numéricas continuas en lugar de clasificar mediante predicciones discretas.

Por ejemplo, nuestros experimentos muestran que los descriptores de resolución baja son efectivos para nuestras metas predictivas: H y G presentan tasas de clasificación $> 99\,\%$ para sus cinco clases. ZPVE y sus siete clases alcanzan tasas $> 90\,\%$. Listamos nuestros descriptores en el punto 7 de la sección 6.2.

Resultados experimentales

También, los optimizadores son influyentes en la calidad de los resultados: los resultados anteriormente mencionados son característicos de AdaBelief, mientras que Adam presenta más variabilidad en sus resultados. En particular, sus modelos son comparables a los de AdaBelief únicamente para los descriptores de resolución alta. Esto nos muestra que, para las metas predictivas que hemos elegido, AdaBelief hace innecesario el hacer experimentos con los costosos descriptores de resolución alta: es suficiente con usar los descriptores de grano grueso. Por otra parte, si buscamos un desempeño alto, y no estamos seguros de cuál optimizador usar, hemos visto que usar descriptores de resolución alta hace menos importante la elección del optimizador.

Nuestros experimentos también han mostrado el poder de las representaciones mixtas, las cuales son más descriptivas y de efectividad típicamente superior a la de sus representaciones componente. Además, esta técnica no se ve afectada por la calidad del optimizador, y su construcción es sencilla: simplemente hemos de concatenar distintas representaciones que, a nuestro criterio, brindan información nueva y relevante para nuestra propiedad objetivo.

C_V es nuestra cuarta propiedad objetivo, la cual presenta la mayor dificultad. Por ejemplo, el mejor de los modelos C_V es AdaBelief-SCM + AC, apenas llegando a un desempeño clasificativo de 0.7371 para el conjunto de prueba. Explorar esta propiedad requiere la creación de otras representaciones mixtas, redes neuronales más profundas, entre otras posibilidades.

Hemos también mostrado dos métodos para la creación de representaciones mixtas refinadas:

- el análisis *Hammond-like* revela la importancia relativa de cada descriptor que forma parte de una representación mixta. Por ejemplo, el AHL nos ha mostrado que EV es más significativo que AC para G, mientras que H presenta el caso contrario. Este hallazgo muestra diferencias claras entre H y G que no habríamos esperado al ver las distribuciones de estas propiedades, como mostramos en la tabla 4.4 y las figuras 4.1a y 4.1b.
- el análisis *per-class* es más revelador que una tabulación global, y nos permite encontrar las clases más difíciles para cada descriptor. Por ejemplo, vemos que los modelos C_V de resolución baja tienen especial dificultad para las clases c_2 y c_6; el mejor desempeño de los modelos EV + AC, SCM, y SCM + AC es parcialmente debido a estas clases.

Explicamos estos métodos a mayor detalle en la sección 6.3.2.

Perspectivas para experimentos futuros

Considerando a todos nuestros experimentos, Adam tiene desempeños típicamente peores a los de AdaBelief. Sin embargo, esto no significa que Adam sea obsoleto: la mayor dispersión en sus tasas y errores facilita discriminar entre descriptores y clases, y este conocimiento también puede ayudar a definir experimentos futuros. Esto significa, también, que nuestros experimentos pueden resultar en conocimiento valioso incluso si nuestra selección de optimizadores fuera limitada y no estuviéramos enterados de AdaBelief.

Algunas bases que podríamos explorar a futuro son:

- PC9 (sección 2.4), cuyas moléculas son semejantes a las de QM9 (todas tienen ≤ 9 átomos pesados $\{\mathrm{C}, \mathrm{N}, \mathrm{O}, \mathrm{F}\}$) y tienen una mayor diversidad funcional,
- algún subconjunto de las bases GDB (sección 2.2), las cuales son ejemplos mayúsculos de bases; la base mayor contiene moléculas de hasta 17 átomos pesados.

Los experimentos futuros podrían también explorar experimentos con más clases, o modelos de regresión con metas predictivas progresivamente más estrictas. Estas últimas podrían aplicarse a C_V: hemos visto que los descriptores de resolución alta logran mejores desempeños que los de resolución baja, y la diferencia entre ellos es la codificación explícita de la conectividad molecular. Esto nos sugiere explorar el desempeño de otros descriptores que incluyen esta información, como los índices de conectividad molecular de Kier y Hall ${}^{m}\chi_{\mathrm{t}}$ [83], *Bag of Bonds* [138], *bonds-and-angles machine learning* (BAML) [139], histogramas de distancias, ángulos, y ángulos dihedrales [140], funciones simetrizantes de Behler-Parrinello (sección 3.5, [84, 85]) o de Smith [29].

6.9 Productividad científica

El trabajo realizado durante la maestría ha resultado en una publicación en `arXiv`, bajo el nombre "*Obtaining transferable chemical insight from solving machine-learning classification problems: Thermodynamical properties prediction, atomic composition as good as Coulomb matrix*" [1], en el cual exponemos los resultados recién mostrados a lo largo de este capítulo.

También hemos asistido a la XX Reunión Mexicana de Fisicoquímica Teórica, celebrada en Cuernavaca, Morelos, en el año 2022, donde participamos en tres carteles:

- Dannte, una implementación accesible del aprendizaje de máquina,

- Composición atómica y AdaBelief: la estequiometría implica propiedades termodinámicas,
- Predicción de las propiedades electrónicas de pigmentos usando redes neuronales.

Apéndice A

Convertir SMILES a coordenadas cartesianas con Open Babel

Índice de apéndice

A.1 Introducción

Open Babel [71] es un programa para la interpretación y traducción de distintos formatos químicos, como los sistemas quimioinformáticos SMILES, InChI, o MOL, o archivos de entrada para programas de estructura electrónica como GAMESS, Gaussian, o MOPAC, entre otros formatos.

Este programa es compatible con Windows, Linux, y MacOS. Sus instrucciones de instalación están disponibles en la documentación del programa, https://open-babel.readthedocs.io/en/latest/Installation/install.html.

La aplicabilidad de Open Babel es diversa. Por ejemplo, si buscamos trabajar con los archivos de una base de datos escritos en un formato particular, tal vez en coordenadas fraccionales en lugar de cartesianas, Open Babel puede traducir las moléculas a un formato compatible con las herramientas que usaremos. O, supongamos que estamos trabajando con las bases GDB (sección 2.2) que únicamente enlistan las moléculas en formato SMILES [36], un lenguaje conciso que representa a la molécula como un grafo sin estructura tridimensional explícita. Si tuviéramos interés en las coordenadas cartesianas de algunas moléculas GDB, Open Babel es una forma conveniente de conseguirlas.

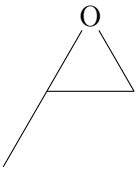
En un ejemplo a menor escala, supongamos que nos interesa estudiar y modelar la superficie de energía potencial de una sola molécula a un nivel de teoría muy alto. Como primer paso hemos de obtener una geometría aproximada para su subsecuente optimización con un programa de estructura electrónica al nivel de teoría deseado. Una forma habitual para obtener las coordenadas aproximadas es escribir la matriz Z de la molécula, lo cual es un problema de geometría euclidiana potencialmente laborioso. También podemos usar una herramienta gráfica para construir la estructura, pero esto puede requerir de un programa comercial.

Un método alternativo, y probablemente más expedito, es escribir el SMILES de la estructura y usar a Open Babel para convertirlo a coordenadas cartesianas. En la siguiente sección mostramos cómo realizar este proceso.

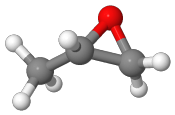
A.2 Traducción mediante Open Babel

Por ejemplo, si estuviéramos interesados en estudiar al óxido de propileno, podemos dibujar su estructura en notación taquigráfica con facilidad (figura A.1a), y luego recorrer su grafo listando cada nodo y su conectividad, resultando en el SMILES `CC1CO1`, donde el listado de átomos empieza por el metilo, y el enlace C−O faltante se indica etiquetando con `1` a estos dos átomos.

Hemos dado una explicación elemental del funcionamiento del lenguaje SMILES en la sección 2.2, específicamente en la página 30. Para un tratamiento a profundidad, sugerimos un sitio mantenido por el autor de SMILES [37].



(a) Fórmula taquigráfica.



(b) Coordenadas optimizadas con Open Babel y MMFF94, graficadas con el visualizador `Jmol` [141].

Figura A.1: Óxido de propileno, molécula 44 de la base de datos QM9 y 6378 de PC9.

Luego, podemos obtener las coordenadas cartesianas aproximadas de la molécula con:

```
$:obabel -:"CC1CO1" -ismi -oxyz --ff=MMFF94 --gen3d slowest -c
10

C         -1.14831        0.29381        0.36795
C          0.33501        0.19465        0.38594
C          1.08829       -0.55000       -0.68286
O          0.98374        0.88122       -0.69022
H         -1.58035       -0.40036        1.09442
H         -1.56405        0.05622       -0.61705
H         -1.46288        1.30723        0.63550
H          0.76326        0.28871        1.37496
H          0.53481       -1.09417       -1.43819
H          2.05048       -0.97731       -0.43045
1 molecule converted
```

donde la primer línea del bloque anterior es una línea de código ejecutada en una terminal, y el resto de líneas son las coordenadas generadas por Open Babel, en el formato `xyz` tradicional. Graficamos esta geometría en la figura A.1b.

Los argumentos del comando anterior tienen el significado:

- `-:"CC1CO1"`, el SMILES a convertir,
- `-ismi`, indica que el formato de entrada es un SMILES,
- `-oxyz`, formato de salida: `xyz`,
- `--ff=MMFF94`. Open Babel genera internamente un *guess* de las coordenadas mediante reglas generales de distancias y ángulos de enlace. Estas coordenadas se pueden todavía refinar mediante una optimización breve al nivel campo de fuerzas (*force field*, FF). En este caso, elegimos a MMFF94 [142, 143],
- `--gen3d`, solicita la generación de las coordenadas tridimensionales de la estructura,
- `slowest`, indica la exigencia del proceso de optimización. `slowest` es la opción con la mayor cantidad de iteraciones y el mejor refinamiento. Otras opciones son `fastest` (no realizar

optimización), `med` (la opción por defecto), y `slow`. Dado el bajo costo de las optimizaciones FF, sugerimos la opción `slowest`, a fin de ahorrar tiempo de cómputo en cálculos posteriores que tengan un costo computacional mayor,

- `-c`, fijar el centro de masa molecular en $(0, 0, 0)$.

Los FF disponibles en Open Babel son:

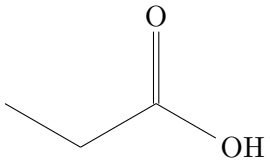
- MMFF94, *Merck Molecular Force Field* [142, 143], un FF versátil, pensado para moléculas orgánicas incluyendo azufre, fósforo, halógenos, y algunos iones típicos,
- MMFF94s, una variante de MMFF94 especializada en estructuras cristalinas [144],
- GAFF, *Generalized Amber Force Field* [145], desarrollado para biomoléculas,
- Ghemical, pensado para moléculas orgánicas y polipéptidos [146],
- UFF, *Universal Force Field* [147]. Mientras que la mayoría de campos de fuerzas se enfocan en algunos átomos en particular y no son compatibles con otros, UFF es compatible con todos, de ahí su designación *universal*.

En general, los autores de Open Babel sugieren GAFF o MMFF94 para moléculas orgánicas, MMFF94s para cristales, y UFF para cualquier otra molécula. Sin embargo, el campo de fuerzas por defecto es Ghemical.

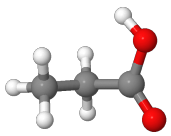
Como ejemplo final, mostramos la conversión del ácido propanoico (figura A.2) con el campo de fuerzas UFF. Indicamos el enlace doble con el signo igual:

```
$:obabel -:"CCC(=O)O" -ismi -oxyz --ff=UFF --gen3d slowest -c
11

C         -1.09130       -0.51666       -0.39030
C          0.41595       -0.45352       -0.25141
C          0.80850        0.42835        0.90474
O          1.00500        0.02052        2.03896
O          0.89263        1.73668        0.57711
H         -1.37172       -1.16848       -1.22322
H         -1.51270        0.47701       -0.57454
H         -1.55253       -0.90872        0.52238
H          0.88360       -0.08606       -1.17118
H          0.82755       -1.45315       -0.07293
H          0.69503        1.92403       -0.35961
1 molecule converted
```



(a) Fórmula taquigráfica.



(b) Coordenadas optimizadas con Open Babel y UFF, graficadas con el visualizador `Jmol` [141].

Figura A.2: Ácido propanoico.

Apéndice B

Algunos archivos GDB-11

Índice de apéndice

Hemos revisado en la sección 2.2 la teoría detrás de las bases GDB, especialmente GDB-11. Estas bases presentan una diversidad química impresionante dentro del espacio de las moléculas orgánicas pequeñas. Las bases GDB son relativamente manejables en términos de espacio en disco, considerando su inmensa cardinalidad. Por ejemplo, GDB-11 separa sus 2.64×10^7 estructuras en 11 archivos según la cantidad de átomos pesados de las moléculas. Las estructuras son representadas en SMILES [36, 37], un lenguaje minimalista constituido por caracteres ASCII. Por lo tanto, todas las estructuras son legibles tanto por computadoras como por humanos, y la base solamente requiere 701 MB de almacenamiento en disco.

Hemos dado una explicación elemental del funcionamiento del lenguaje SMILES en la sección 2.2, específicamente en la página 30. Para un tratamiento a profundidad, sugerimos un sitio mantenido por el autor de SMILES [37].

A fin de ilustrar el aspecto y accesibilidad de estas bases, en este capítulo mostraremos los primeros cuatro archivos de GDB-11. Los archivos de todas las bases GDB están disponibles gratuitamente en el sitio web del autor [70].

Archivo	Tamaño	Cardinalidad	
1	24 B	4	4
2	67 B	9	9
3	187 B	20	20
4	907 B	80	80
5	4.9 kB	352	352
6	30 kB	1850	1850
7	195 kB	1.06×10^4	10 568
8	1.4 MB	6.67×10^4	66 706
9	11 MB	4.44×10^5	444 313
10	77 MB	3.11×10^6	3 114 041
11	613 MB	2.28×10^7	22 796 628
total	701 MB	2.64×10^7	26 434 571

Tabla B.1: Tamaño y cardinalidad de los archivos GDB-11.

B.1 Moléculas de 1 átomo pesado

El archivo `gdb11_size01.smi` lista 4 moléculas.

```
C       1       1
N       2       1
O       3       1
F       4       1
```

B.2 Moléculas de 2 átomos pesados

El archivo `gdb11_size02.smi` lista 9 moléculas.

```
CC      1       1
CN      2       1
CO      3       1
CF      4       1
FF      5       1
C=C     6       1
C=O     7       1
O=O     8       1
C#C     9       1
```

B.3 Moléculas de 3 átomos pesados

El archivo `gdb11_size03.smi` lista 20 moléculas.

```
CCC     1       1
CCN     2       1
CCO     3       1
CCF     4       1
FCF     5       1
CNC     6       1
COC     7       1
CC=C    8       1
```

```
CC=O    9       1
CC#N    10      1
NC=N    11      1
NC=O    12      2
OC=O    13      1
FC=C    14      1
NN=C    15      1
ON=C    16      1
CC#C    17      1
C1CC1   18      1
C1CN1   19      1
C1CO1   20      1
```

B.4 Moléculas de 4 átomos pesados

El archivo `gdb11_size04.smi` lista 80 moléculas.

```
CC(C)C  1       1
CC(C)N  2       1
CC(C)O  3       1
CC(C)F  4       1
CC(F)F  5       1
FC(F)F  6       1
CN(C)C  7       1
CC(C)=C 8       1
CC(C)=O 9       1
CC(N)=N 10      1
CC(N)=O 11      2
CC(O)=O 12      1
CC(F)=C 13      1
NC(N)=N 14      1
NC(N)=O 15      1
OC(O)=O 16      1
FC(F)=C 17      1
CCCC    18      1
CCCN    19      1
CCCO    20      1
CCCF    21      1
NCCN    22      1
NCCO    23      1
NCCF    24      1
OCCO    25      1
OCCF    26      1
FCCF    27      1
CCNC    28      1
CCOC    29      1
CCC=C   30      1
CCC=O   31      1
CCC#N   32      1
NCC=C   33      1
OCC=C   34      1
OCC=O   35      1
OCC#N   36      1
FCC=C   37      1
FCC=O   38      1
FCC#N   39      1
CNC=N   40      2
CNC=O   41      2
NNC=N   42      2
NNC=O   43      2
ONC=N   44      2
ONC=O   45      2
CNN=C   46      1
COC=C   47      1
COC=O   48      1
CON=C   49      1
CCC#C   50      1
NCC#C   51      1
OCC#C   52      1
FCC#C   53      1
C=CC=C  54      1
C=CC=O  55      1
C=CC#N  56      1
O=CC=O  57      1
O=CC#N  58      1
N#CC#N  59      1
C=NN=C  60      1
C=CC#C  61      1
O=CC#C  62      1
C#CC#N  63      1
C#CC#C  64      1
CC=CC   65      1
CC=CF   66      1
```

```
FC=CF   67      1
CC=NN   68      1
CC=NO   69      1
CC#CC   70      1
CC1CC1  71      1
NC1CC1  72      1
OC1CC1  73      1
FC1CC1  74      1
CC1CN1  75      1
CC1CO1  76      1
CN1CC1  77      1
C1CCC1  78      1
C1CNC1  79      1
C1COC1  80      1
```

Bibliografía

[1] Leon Alday-Toledo *et al.* *Obtaining transferable chemical insight from solving machine-learning classification problems: Thermodynamical properties prediction, atomic composition as good as Coulomb matrix*. 2022. DOI: 10.48550/arxiv.2212.01420. URL: https://arxiv.org/abs/2212.01420.

[2] Yann LeCun *et al.* "Gradient-Based Learning Applied to Document Recognition". En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. DOI: 10.1109/5.726791.

[3] Kenneth J. Wilder. "Decision Tree Algorithms for Handwritten Digit Recognition". Tesis doct. United States of America: University of Massachusetts, 1998. URL: https://scholarworks.umass.edu/dissertations/AAI9823791/.

[4] Daniel Keysers *et al.* "Deformation models for image recognition". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.8 (2007), págs. 1422-1435. DOI: 10.1109/TPAMI.2007.1153.

[5] Anil Ananthaswamy. "In AI, is bigger always better?" En: *Nature* 615 (2023), págs. 202-205. DOI: 10.1038/d41586-023-00641-w.

[6] A. M. Hopkins *et al.* "Artificial intelligence chatbots will revolutionize how cancer patients access information: ChatGPT represents a paradigm-shift". En: *JNCI Cancer Spectrum* (2023). DOI: 10.1093/jncics/pkad010.

[7] Jeff Prosise. *Understanding ChatGPT*. 2023. URL: https://www.atmosera.com/ai/understanding-chatgpt/ (visitado 26-03-2023).

[8] Aitor Lewkowycz *et al.* *Solving Quantitative Reasoning Problems with Language Models*. 2022. DOI: 10.48550/arXiv.2206.14858. arXiv: 2206.14858 [cs.CL].

[9] OpenAI. *DALL·E 2*. 2023. URL: https://openai.com/product/dall-e-2.

[10] Gary Marcus, Ernest Davis y Scott Aaronson. *A very preliminary analysis of DALL-E 2*. 2022. DOI: 10.48550/arXiv.2204.13807. arXiv: 2204.13807 [cs.CV].

[11] Craiyon LLC. *Craiyon, AI Image Generator*. 2023. URL: https://www.craiyon.com/.

[12] Audra Schroeder. *AI program DALL-E mini prompts some truly cursed images*. 2022. URL: https://www.dailydot.com/unclick/dall-e-mini-memes/ (visitado 02-02-2023).

[13] Midjourney, Inc. *Midjourney*. 2023. URL: https://www.midjourney.com.

[14] Martin Ruskov. *Grimm in Wonderland: Prompt Engineering with Midjourney to Illustrate Fairytales*. 2023. DOI: 10.48550/arXiv.2302.08961. arXiv: 2302.08961 [cs.CL].

[15] Ola Amayri y Nizar Bouguila. "A study of spam filtering using support vector machines". En: *Artificial Intelligence Review* 34.1 (2010), págs. 73-108. DOI: 10.1007/s10462-010-9166-x.

[16] Aditya Shrivastava y Rachana Dubey. "Classification of Spam Mail using different machine learning algorithms". En: *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*. IEEE, 2018, págs. 1-10. ISBN: 978-1-5386-5367-8. DOI: 10.1109/ICACAT.2018.8933787. URL: https://ieeexplore.ieee.org/document/8933787/.

[17] Dominik Sobania, Martin Briesch y Franz Rothlauf. *Choose Your Programming Copilot: A Comparison of the Program Synthesis Performance of GitHub Copilot and Genetic Programming*. 2021. DOI: 10.48550/arXiv.2111.07875. URL: https://doi.org/10.48550/arXiv.2111.07875.

[18] David Hsu. *Using Convolutional Neural Networks to Classify Dog Breeds*. 2015. URL: http://cs231n.stanford.edu/reports/2015/pdfs/fcdh_FinalReport.pdf.

[19] Zalán Ráduly *et al.* "Dog Breed Identification Using Deep Learning". En: *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*. 2018, págs. 000271-000276. DOI: 10.1109/SISY.2018.8524715.

[20] Wenting Shi *et al. Dog Breed Identification*. 2018. URL: http://noiselab.ucsd.edu/ECE228_2018/Reports/Report18.pdf.

[21] Punyanuch Borwarnginn *et al.* "Knowing Your Dog Breed: Identifying a Dog Breed with Deep Learning". En: *International Journal of Automation and Computing* (2020). ISSN: 17518520. DOI: 10.1007/s11633-020-1261-0.

[22] Friedrich Wöhler. "Analytische Versuche über die Cyansäure". En: *Annalen der Physik* 77.5 (1824), págs. 117-124. DOI: 10.1002/andp.18240770506. URL: https://doi.org/10.1002/andp.18240770506.

[23] Justus Liebig y Joseph Louis Gay-Lussac. "Analyse du Fulminate d'argent". En: *Annales de chimie et de physique* 25 (1824), págs. 285-311. URL: https://gallica.bnf.fr/ark:/12148/bpt6k65688009/f295.item.

[24] Soledad Esteban. "Liebig-Wöhler Controversy and the Concept of Isomerism". En: *Journal of Chemical Education* 85.9 (2008), págs. 1201-1203. DOI: 10.1021/ed085p1201. URL: https://doi.org/10.1021/ed085p1201.

[25] Jacob Berzelius. "Isomerie, Unterscheidung von damit analogen Verhältnissen". En: *Jahres-Bericht über die Fortschritte der physischen Wissenschaften* 12 (1833), págs. 63-67. URL: https://zs.thulb.uni-jena.de/rsc/viewer/jportal_derivate_00223636/NT_328_l_Seite_074.tiff?logicalDiv=jportal_jparticle_00297008.

[26] Alexander Crum Brown y Thomas Richard Fraser. *On the Connection between Chemical Constitution and Physiological Action. Part I.—On the Physiological Action of the Salts of the Salts of the Ammonium Bases, derived from Strychnia, Brucia, Thebaia, Codeia, Morphia, and Nicotia*. Vol. 25. Royal Society of Edinburgh, 1868, págs. 151-203. URL: https://www.biodiversitylibrary.org/item/126701.

[27] Helmut Gassner *et al.* "Representation of intermolecular potential functions by neural networks". En: *Journal of Physical Chemistry A* 102.24 (1998), págs. 4596-4605. DOI: 10.1021/jp972209d.

[28] Jörg Behler. "Constructing high-dimensional neural network potentials: A tutorial review". En: *International Journal of Quantum Chemistry* 115.16 (2015), págs. 1032-1050. DOI: 10.1002/qua.24890.

[29] J. S. Smith, O. Isayev y A. E. Roitberg. "ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost". En: *Chemical Science* 8.4 (2017), págs. 3192-3203. DOI: 10.1039/C6SC05720A.

[30] Stefan Chmiela *et al.* "Machine learning of accurate energy-conserving molecular force fields". En: *Science Advances* 3.5 (2017). ISSN: 23752548. DOI: 10.1126/sciadv.1603015. arXiv: 1611.04678.

[31] Julian J. Kranz *et al.* "Generalized Density-Functional Tight-Binding Repulsive Potentials from Unsupervised Machine Learning". En: *Journal of Chemical Theory and Computation* 14.5 (2018), págs. 2341-2352. ISSN: 15499626. DOI: 10.1021/acs.jctc.7b00933.

[32] Daniel C. Elton *et al.* "Deep learning for molecular design - A review of the state of the art". En: *Molecular Systems Design and Engineering* 4.4 (2019), págs. 828-849. ISSN: 20589689. DOI: 10.1039/c9me00039a. arXiv: 1903.04388.

[33] Tetsuhiko Takabatake *et al.* "Artificial intelligence-designed stereoselective one-pot synthesis of trans-β-lactams and its application to cholesterol absorption inhibitor SCH 47949 synthesis". En: *Heterocycles* 100.1 (2020), págs. 60-84. DOI: 10.3987/COM-19-14130.

[34] Qifan Kuang *et al.* "An eigenvalue transformation technique for predicting drug-target interaction". En: *Scientific Reports* 5 (2015), págs. 1-9. ISSN: 20452322. DOI: 10.1038/srep13867.

[35] Nicholas J. Browning *et al.* "Genetic Optimization of Training Sets for Improved Machine Learning Models of Molecular Properties". En: *Journal of Physical Chemistry Letters* 8.7 (2017), págs. 1351-1359. DOI: 10.1021/acs.jpclett.7b00038.

[36] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". En: *Journal of Chemical Information and Computer Sciences* 28.1 (1988), págs. 31-36. DOI: 10.1021/ci00057a005. URL: https://doi.org/10.1021/ci00057a005.

[37] Daylight Chemical Information Systems, Inc. *3. SMILES - A Simplified Chemical Language*. 2022. URL: https://daylight.com/dayhtml/doc/theory/theory.smiles.html.

[38] Yongbeom Kwon y Juyong Lee. "MolFinder: an evolutionary algorithm for the global optimization of molecular properties and the extensive exploration of chemical space using SMILES". En: *Journal of Cheminformatics* 13.24 (2021), págs. 1-14. DOI: 10.1186/s13321-021-00501-7.

[39] Gabriel A. Pinheiro *et al.* "Machine Learning Prediction of Nine Molecular Properties Based on the SMILES Representation of the QM9 Quantum-Chemistry Dataset". En: *Journal of Physical Chemistry A* (2020). DOI: 10.1021/acs.jpca.0c05969.

[40] Dennis Decoste y Schölkopf. "Training Invariant Support Vector Machines". En: *Machine Learning* 46 (2002), págs. 161-190. DOI: 10.1023/A:1012454411458.

[41] Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". En: *Communications of the ACM* 60.6 (2017), págs. 84-90. DOI: 10.1145/3065386.

[42] Mike Innes. "Flux: Elegant machine learning with Julia". En: *Journal of Open Source Software* 3.25 (2018), pág. 602. DOI: 10.21105/joss.00602.

[43] Michael Innes *et al. Fashionable Modelling with Flux*. 2018. DOI: 10.48550/arXiv.1811.01457. URL: https://arxiv.org/abs/1811.01457.

[44] Fabian Pedregosa *et al.* "Scikit-learn: Machine Learning in Python". En: *Journal of Machine Learning Research* 12.85 (2011), págs. 2825-2830. ISSN: 15337928. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[45] Martín Abadi *et al.* "TensorFlow: A system for large-scale machine learning". En: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016* (2016), págs. 265-283. arXiv: 1605.08695.

[46] Adam Paszke *et al.* "PyTorch: An imperative style, high-performance deep learning library". En: *Advances in Neural Information Processing Systems* 32 (2019), págs. 8024-8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[47] G. Cybenko. "Approximation by superpositions of a sigmoidal function". En: *Mathematics of Control, Signals and Systems* 2 (1989), págs. 303-314. DOI: 10.1007/BF02551274.

[48] Kurt Hornik, Maxwell Stinchcombe y Halbert White. "Multilayer feedforward networks are universal approximators". En: *Neural Networks* 2.5 (1989), págs. 359-366. DOI: 10.1016/0893-6080(89)90020-8. URL: https://doi.org/10.1016/0893-6080(89)90020-8.

[49] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". En: *Neural Networks* 4.2 (1991), págs. 251-257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: https://doi.org/10.1016/0893-6080(91)90009-T.

[50] Jonathan T. Barron. "Continuously differentiable exponential linear units". En: *arXiv* 3 (2017), págs. 1-2. arXiv: 1704.07483.

[51] Sebastian Ruder. "An overview of gradient descent optimization algorithms". En: (2016), págs. 1-14. URL: https://arxiv.org/abs/1609.04747.

[52] Diederik P. Kingma y Jimmy Ba. "Adam: A Method for Stochastic Optimization". En: (2014). DOI: 10.48550/arXiv.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[53] Juntang Zhuang *et al.* "AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients". En: (2020). DOI: 10.48550/arXiv.2010.07468. URL: https://arxiv.org/abs/2010.07468.

[54] Geoffrey E. Hinton, Simon Osindero y Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". En: *Neural Computation* 18.7 (2006), págs. 1527-1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. eprint: https://direct.mit.edu/neco/article-pdf/18/7/1527/816558/neco.2006.18.7.1527.pdf. URL: https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf.

[55] Asja Fischer y Christian Igel. "Training restricted Boltzmann machines: An introduction". En: *Pattern Recognition* 47.1 (2014), págs. 25-39. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2013.05.025. URL: https://doi.org/10.1016/j.patcog.2013.05.025.

[56] Dumitru Erhan *et al.* "The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training". En: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. por David van Dyk y Max Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 2009, págs. 153-160. URL: https://proceedings.mlr.press/v5/erhan09a.html.

[57] Xavier Glorot y Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". En: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Yee Whye Teh y Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, págs. 249-256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[58] Ester Hlav. *Xavier Glorot Initialization in Neural Networks – Math Proof*. 2022. URL: https://towardsdatascience.com/xavier-glorot-initialization-in-neural-networks-math-proof-4682bf5c6ec3.

[59] Eric W. Weisstein. *Uniform Distribution*. 2004. URL: https://mathworld.wolfram.com/UniformDistribution.html (visitado 02-04-2023).

[60] Kaiming He *et al.* *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. DOI: 10.48550/arXiv.1502.01852. URL: https://doi.org/10.48550/arXiv.1502.01852.

[61] Olga Russakovsky *et al.* *ImageNet Large Scale Visual Recognition Challenge*. 2015. DOI: 10.48550/arXiv.1409.0575. URL: https://doi.org/10.48550/arXiv.1409.0575.

[62] Ester Hlav. *Kaiming He Initialization in Neural Networks – Math Proof*. 2023. URL: https://towardsdatascience.com/kaiming-he-initialization-in-neural-networks-math-proof-73b9a0d845c4.

[63] Leo Breiman y Philip Spector. "Submodel Selection and Evaluation in Regression. The X-Random Case". En: *International Statistical Review / Revue Internationale de Statistique* 60.3 (1992), págs. 291-319. DOI: 10.2307/1403680. URL: http://www.jstor.org/stable/1403680.

[64] Tobias Fink, Heinz Bruggesser y Jean-Louis Reymond. "Virtual Exploration of the Small-Molecule Chemical Universe below 160 Daltons". En: *Angewandte Chemie International Edition* 44.10 (2005), págs. 1504-1508. DOI: 10.1002/anie.200462457. URL: https://doi.org/10.1002/anie.200462457.

[65] Tobias Fink y Jean-Louis Reymond. "Virtual Exploration of the Chemical Universe up to 11 Atoms of C, N, O, F: Assembly of 26.4 Million Structures (110.9 Million Stereoisomers) and Analysis for New Ring Systems, Stereochemistry, Physicochemical Properties, Compound Classes, and Drug Discovery". En: *Journal of Chemical Information and Modeling* 47.2 (2007), págs. 342-353. DOI: 10.1021/ci600423u. URL: https://doi.org/10.1021/ci600423u.

[66] Lorenz C. Blum y Jean Louis Reymond. "970 Million druglike small molecules for virtual screening in the chemical universe database GDB-13". En: *Journal of the American Chemical Society* 131.25 (2009), págs. 8732-8733. DOI: 10.1021/ja902302h.

[67] Lars Ruddigkeit *et al.* "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17". En: *Journal of Chemical Information and Modeling* 52.11 (2012), págs. 2864-2875. DOI: 10.1021/ci300415d.

[68] Christopher A. Lipinski *et al.* "Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings". En: *Advanced Drug Delivery Reviews* 23.1 (1997). In Vitro Models for Selection of Development Candidates, págs. 3-25. DOI: 10.1016/S0169-409X(96)00423-1.

[69] Miles Congreve *et al.* "A 'Rule of Three' for fragment-based lead discovery?" En: *Drug Discovery Today* 8.19 (2003), págs. 876-877. DOI: 10.1016/S1359-6446(03)02831-9. URL: https://www.sciencedirect.com/science/article/pii/S1359644603028319.

[70] Daniel Probst y Jean-Louis Reymond. *Download Chemical Databases*. 2020. URL: https://gdb.unibe.ch/downloads/.

[71] Noel M. O'Boyle *et al.* "Open Babel: An open chemical toolbox". En: *J. Cheminform.* 3.1 (2011), pág. 33. DOI: 10.1186/1758-2946-3-33.

[72] Raghunathan Ramakrishnan *et al.* "Quantum chemistry structures and properties of 134 kilo molecules". En: *Scientific Data* 1 (2014). DOI: 10.1038/sdata.2014.22.

[73] Raghunathan Ramakrishnan *et al. Quantum chemistry structures and properties of 134 kilo molecules*. 2019. DOI: 10.6084/m9.figshare.c.978904.v5. URL: https://springernature.figshare.com/collections/Quantum_chemistry_structures_and_properties_of_134_kilo_molecules/978904/5.

[74] Marta Glavatskikh *et al.* "Dataset's chemical diversity limits the generalizability of machine learning predictions". En: *Journal of Cheminformatics* 11.1 (2019). DOI: 10.1186/s13321-019-0391-2.

[75] Thomas Cauchy, Benoit Da Mota y Marta Glavastkikh. *PC9_data.zip*. 2019. URL: https://figshare.com/articles/dataset/PC9_data_zip/9033977.

[76] Rakesh Maity, Debkumar Mandal y Ajay Misra. "Effect of donor acceptor substitution position on the electrical responsive properties of azulene system: a computational study". En: *Molecular Physics* 117.14 (2019), págs. 1781-1789. DOI: 10.1080/00268976.2018.1543902.

[77] Fan Xu *et al.* "Effect of electronic spatial extents (ESE) of ions on overpotential of lithium ion capacitors". En: *Electrochimica Acta* 115 (2014), págs. 234-238. DOI: 10.1016/j.electacta.2013.10.175.

[78] Stephen Heller *et al.* "InChI - the worldwide chemical structure identifier standard". En: *J. Cheminform.* 5.7 (2013). DOI: 10.1186/1758-2946-5-7.

[79] Yanli Wang *et al.* "PubChem: a public information system for analyzing bioactivities of small molecules". En: *Nucleic Acids Research* 37 (2009), W623-W633. DOI: 10.1093/nar/gkp456. URL: https://doi.org/10.1093/nar/gkp456.

[80] Sunghwan Kim *et al.* "PubChem 2019 update: improved access to chemical data". En: *Nucleic Acids Research* 47.D1 (2018), págs. D1102-D1109. DOI: 10.1093/nar/gky1033. URL: https://doi.org/10.1093/nar/gky1033.

[81] Sunghwan Kim *et al.* "PubChem in 2021: new data content and improved web interfaces". En: *Nucleic Acids Research* 49.D1 (2020), págs. D1388-D1395. DOI: 10.1093/nar/gkaa971. URL: https://doi.org/10.1093/nar/gkaa971.

[82] Maho Nakata y Tomomi Shimazaki. "PubChemQC Project: A Large-Scale First-Principles Electronic Structure Database for Data-Driven Chemistry". En: *Journal of Chemical Information and Modeling* 57.6 (2017), págs. 1300-1308. DOI: 10.1021/acs.jcim.7b00083. URL: https://doi.org/10.1021/acs.jcim.7b00083.

[83] Lowell H. Hall y Lemont B. Kier. "The Molecular Connectivity Chi Indexes and Kappa Shape Indexes in Structure-Property Modeling". En: *Reviews in Computational Chemistry*. John Wiley & Sons, Ltd, 1991. Cap. 9, págs. 367-422. DOI: 10.1002/9780470125793.ch9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470125793.ch9.

[84] Jörg Behler. "Atom-centered symmetry functions for constructing high-dimensional neural network potentials". En: *Journal of Chemical Physics* 134.7 (2011). DOI: 10.1063/1.3553717.

[85] Jörg Behler y Michele Parrinello. "Generalized neural-network representation of high-dimensional potential-energy surfaces". En: *Physical Review Letters* 98.14 (2007), págs. 1-4. DOI: 10.1103/PhysRevLett.98.146401.

[86] Albert Bakker *et al.* "Interaction of aluminum(III) with water. An Ab initio study". En: *International Journal of Quantum Chemistry* 75.4-5 (1999), págs. 659-669. DOI: 10.1002/(SICI)1097-461X(1999)75:4/5<659::AID-QUA33>3.0.CO;2-O.

[87] Justin S. Smith *et al.* "Less is more: Sampling chemical space with active learning". En: *Journal of Chemical Physics* 148.24 (2018), pág. 241733. DOI: 10.1063/1.5023802. URL: https://doi.org/10.1063/1.5023802.

[88] Justin S. Smith *et al.* "The ANI-1ccx and ANI-1x data sets, coupled-cluster and density functional theory properties for molecules". En: *Scientific Data* 7.1 (2020), págs. 1-10. DOI: 10.1038/s41597-020-0473-z.

[89] Christian Devereux *et al.* "Extending the Applicability of the ANI Deep Learning Molecular Potential to Sulfur and Halogens". En: *Journal of Chemical Theory and Computation* 16.7 (2020), págs. 4192-4202. DOI: 10.1021/acs.jctc.0c00121. URL: https://doi.org/10.1021/acs.jctc.0c00121.

[90] Michael J. S. Dewar *et al.* "Development and use of quantum mechanical molecular models. 76. AM1: a new general purpose quantum mechanical molecular model". En: *Journal of the American Chemical Society* 107.13 (1985), págs. 3902-3909. DOI: 10.1021/ja00299a024. URL: http://doi.org/10.1021/ja00299a024.

[91] James J. P. Stewart. "Optimization of parameters for semiempirical methods V: Modification of NDDO approximations and application to 70 elements". En: *Journal of Molecular Modeling* 13.12 (2007), págs. 1173-1213. DOI: 10.1007/s00894-007-0233-4.

[92] Linus Pauling y L. O. Brockway. "Carbon-Carbon Bond Distances. The Electron Diffraction Investigation of Ethane, Propane, Isobutane, Neopentane, Cyclopropane, Cyclopentane, Cyclohexane, Allene, Ethylene, Isobutene, Tetramethylethylene, Mesitylene, and Hexamethylbenzene. Revised Values of Covalent Radii". En: *Journal of the American Chemical Society* 59.7 (1937), págs. 1223-1236. DOI: 10.1021/ja01286a021.

[93] Matthias Rupp *et al.* "Fast and accurate modeling of molecular atomization energies with machine learning". En: *Physical Review Letters* 108.5 (2012), págs. 1-5. DOI: 10.1103/PhysRevLett.108.058301.

[94] Pavlo O. Dral, O. Anatole von Lilienfeld y Walter Thiel. "Machine learning of parameters for accurate semiempirical quantum chemical calculations". En: *Journal of Chemical Theory and Computation* 11.5 (2015), págs. 2120-2125. DOI: 10.1021/acs.jctc.5b00141.

[95] Sidney W. Benson. "Bond Energies". En: *Journal of Chemical Education* 42.9 (1965), págs. 502-518. DOI: 10.1021/ed042p502.

[96] Grégoire Montavon *et al.* "Learning Invariant Representations of Molecules for Atomization Energy Prediction". En: *Advances in Neural Information Processing Systems* 25 (2012), págs. 449-457. URL: https://proceedings.neurips.cc/paper/2012/file/115f89503138416a242f40fb7d7f338e-Paper.pdf.

[97] Katja Hansen *et al.* "Assessment and validation of machine learning methods for predicting molecular atomization energies". En: *Journal of Chemical Theory and Computation* 9.8 (2013), págs. 3404-3419. DOI: 10.1021/ct400195d.

[98] Anders S. Christensen *et al. qmlcode/qml: Release v0.3.1*. Ver. 0.3.1. 2017. DOI: 10.5281/zenodo.817332. URL: https://doi.org/10.5281/zenodo.817332.

[99] Alain B. Tchagang y Julio J. Valdés. "Prediction of the Atomization Energy of Molecules Using Coulomb Matrix and Atomic Composition in a Bayesian Regularized Neural Networks". En: *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*. Ed. por Igor V. Tetko *et al.* Cham: Springer International Publishing, 2019, págs. 793-803. ISBN: 978-3-030-30493-5. DOI: 10.1007/978-3-030-30493-5_75.

[100] Matthias Rupp. "Machine learning for quantum mechanics in a nutshell". En: *International Journal of Quantum Chemistry* 115.16 (2015), págs. 1058-1073. DOI: 10.1002/qua.24954. URL: https://doi.org/10.1002/qua.24954.

[101] L. Riedel. "Ein neues Verfahren zur Abschätzung unbekannter kritischer Drucke von organischen Verbindungen". En: *Zeitschrift für Elektrochemie und angewandte physikalische Chemie* 53.4 (1949), págs. 222-228. DOI: 10.1002/bbpc.19490530411. URL: https://onlinelibrary.wiley.com/doi/10.1002/bbpc.19490530411.

[102] Sidney W. Benson y Jerry H. Buss. "Additivity Rules for the Estimation of Molecular Properties. Thermodynamic Properties". En: *The Journal of Chemical Physics* 29.3 (1958), págs. 546-572. DOI: 10.1063/1.1744539. URL: https://doi.org/10.1063/1.1744539.

[103] Aksel Lydersen. *Estimation of Critical Properties of Organic Compounds*. 1955.

[104] D. Ambrose. *Correlation and Estimation of Vapor-Liquid Critical Properties: I. Critical Temperatures of Organic Compounds*. NPL Rep. Chem. 92. National Physical Laboratory, Teddington, 1978, corregido 1980.

[105] D. Ambrose. *Correlation and Estimation of Vapor-Liquid Critical Properties: II. Critical Pressures and Volumes of Organic Compounds*. NPL Rep. Chem. 98. National Physical Laboratory, Teddington, 1979.

[106] K. G. Joback y R. C. Reid. "Estimation of Pure-Component Properties From Group-Contributions". En: *Chemical Engineering Communications* 57.1-6 (1987), págs. 233-243. DOI: 10.1080/00986448708960487. URL: https://doi.org/10.1080/00986448708960487.

[107] Leonidas Constantinou y Rafiqul Gani. "New group contribution method for estimating properties of pure compounds". En: *AIChE Journal* 40.10 (1994), págs. 1697-1710. DOI: 10.1002/aic.690401011. URL: https://doi.org/10.1002/aic.690401011.

[108] Yash Nannoolal, Jürgen Rarey y Deresh Ramjugernath. "Estimation of pure component properties: Part 2. Estimation of critical property data by group contribution". En: *Fluid Phase Equilibria* 252.1 (2007), págs. 1-27. DOI: 10.1016/j.fluid.2006.11.014. URL: https://doi.org/10.1016/j.fluid.2006.11.014.

[109] Yash Nannoolal *et al.* "Estimation of pure component properties: Part 1. Estimation of the normal boiling point of non-electrolyte organic compounds via group contributions and group interactions". En: *Fluid Phase Equilibria* 226 (2004), págs. 45-63. DOI: 10.1016/j.fluid.2004.09.001. URL: https://doi.org/10.1016/j.fluid.2004.09.001.

[110] Aage Fredenslund, Russell L. Jones y John M. Prausnitz. "Group-contribution estimation of activity coefficients in nonideal liquid mixtures". En: *AIChE Journal* 21.6 (1975), págs. 1086-1099. DOI: 10.1002/aic.690210607. URL: https://doi.org/10.1002/aic.690210607.

[111] Frank R. Burden. "Using Artificial Neural Networks to Predict Biological Activity from Simple Molecular Structural Considerations". En: *Quantitative Structure-Activity Relationships* 15.1 (1996), págs. 7-11. DOI: 10.1002/qsar.19960150103. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/qsar.19960150103.

[112] Frank R. Burden y David A. Winkler. "Robust QSAR Models Using Bayesian Regularized Neural Networks". En: *Journal of Medicinal Chemistry* 42.16 (1999), págs. 3183-3187. DOI: 10.1021/jm980697n. URL: https://doi.org/10.1021/jm980697n.

[113] Jonathan E. Moussa. "Comment on 'fast and accurate modeling of molecular atomization energies with machine learning'". En: *Physical Review Letters* 109.5 (2012), pág. 059801. DOI: 10.1103/PhysRevLett.109.059801. arXiv: 1208.1085.

[114] Matthias Rupp *et al.* En: *Physical Review Letters* 109.5 (2012), pág. 059802. DOI: 10.1103/PhysRevLett.109.059802.

[115] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. Wiley, 2001. ISBN: 9780471873396.

[116] John Duchi, Elad Hazan y Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". En: *Journal of Machine Learning Research* 12 (2011), págs. 2121-2159.

[117] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. 2013. DOI: 10.48550/arXiv.1308.0850. URL: https://arxiv.org/abs/1308.0850.

[118] Ashia C. Wilson *et al. The Marginal Value of Adaptive Gradient Methods in Machine Learning*. 2018. DOI: 10.48550/arXiv.1705.08292. URL: https://arxiv.org/abs/1705.08292.

[119] Ian J. Goodfellow *et al. Generative Adversarial Networks*. 2014. DOI: 10.48550/arXiv.1406.266. URL: https://doi.org/10.48550/arXiv.1406.266.

[120] Leon Alday-Toledo, Roberto Bernal-Jaquez y Saúl Zapotecas-Martínez. *Dannte suite – Trabajo en progreso*. 2022.

[121] Jeff Bezanson *et al.* "Julia: A Fresh Approach to Numerical Computing". En: *SIAM Review* 59.1 (2017), págs. 65-98. DOI: 10.1137/141000671. URL: https://doi.org/10.1137/141000671.

[122] Michael Innes. *Don't Unroll Adjoint: Differentiating SSA-Form Programs*. 2018. DOI: 10.48550/arxiv.1810.07951. URL: https://arxiv.org/abs/1810.07951.

[123] Mathieu Besançon *et al.* "Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem". En: *Journal of Statistical Software* 98.16 (2021). DOI: 10.18637/jss.v098.i16. URL: https://doi.org/10.18637%2Fjss.v098.i16.

[124] Dahua Lin *et al. JuliaStats/Distributions.jl*. 2022. DOI: 10.5281/zenodo.2647458. URL: https://doi.org/10.5281/zenodo.2647458.

[125] Tom Breloff. *Plots.jl*. Ver. v1.31.7. 2022. DOI: 10.5281/zenodo.6989218. URL: https://doi.org/10.5281/zenodo.6989218.

[126] Tim Holy *et al. ProgressMeter.jl*. Ver. 1.7.1. 2021. URL: https://github.com/timholy/ProgressMeter.jl.

[127] Jarrett Revels *et al. BenchmarkTools.jl*. Ver. 1.2.2. 2021. URL: https://github.com/JuliaCI/BenchmarkTools.jl.

[128] Dahua Lin *et al. StatsBase.jl*. Ver. 0.33.14. 2022. URL: https://github.com/JuliaStats/StatsBase.jl.

[129] Tim Holy *et al. HDF5.jl*. Ver. 0.16.1. 2022. URL: https://github.com/JuliaIO/HDF5.jl.

[130] John M. Kuhn *et al. Formatting.jl*. Ver. 0.4.2. 2020. URL: https://github.com/JuliaIO/Formatting.jl.

[131] The HDF Group. *The HDF5® library & file format*. 2017. URL: https://www.hdfgroup.org/solutions/hdf5/.

[132] Earth Science Data and Information System Project. *HDF5 Data Model, File Format and Library—HDF5 1.6*. 2020. URL: https://www.earthdata.nasa.gov/esdis/esco/standards-and-practices/hdf5.

[133] N. Rees *et al.* "Developing HDF5 for the Synchrotron Community". En: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, 17-23 October 2015*. (Melbourne, Australia). International Conference on Accelerator and Large Experimental Physics Control Systems 15. Geneva, Switzerland: JACoW, 2015, págs. 845-848. ISBN: 978-3-95450-148-9. DOI: doi:10.18429/JACoW-ICALEPCS2015-WEPGF063. URL: http://jacow.org/icalepcs2015/papers/wepgf063.pdf.

[134] Michael Innes *et al. Optimiser Reference*. 2023. URL: https://fluxml.ai/Flux.jl/stable/training/optimisers/#Optimiser-Reference (visitado 22-09-2023).

[135] Thomas Elsken, Jan Hendrik Metzen y Frank Hutter. "Neural Architecture Search: A Survey". En: *Journal of Machine Learning Research* 20.55 (2019), págs. 1-21. URL: http://jmlr.org/papers/v20/18-598.html.

[136] Martin Wistuba, Ambrish Rawat y Tejaswini Pedapati. *A Survey on Neural Architecture Search*. 2019. DOI: 10.48550/arXiv.1905.01392. URL: https://arxiv.org/abs/1905.01392.

[137] George S. Hammond. "A Correlation of Reaction Rates". En: *Journal of the American Chemical Society* 77.2 (1955), págs. 334-338. DOI: 10.1021/ja01607a027. URL: https://doi.org/10.1021/ja01607a027.

[138] Katja Hansen *et al.* "Machine Learning Predictions of Molecular Properties: Accurate Many-Body Potentials and Nonlocality in Chemical Space". En: *The Journal of Physical Chemistry Letters* 6.12 (2015), págs. 2326-2331. DOI: 10.1021/acs.jpclett.5b00831. URL: https://doi.org/10.1021/acs.jpclett.5b00831.

[139] Bing Huang y O. Anatole von Lilienfeld. "Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity". En: *The Journal of Chemical Physics* 145.16 (2016), pág. 161102. DOI: 10.1063/1.4964627. URL: https://doi.org/10.1063/1.4964627.

[140] Felix A. Faber *et al.* "Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error". En: *Journal of Chemical Theory and Computation* 13.11 (2017), págs. 5255-5264. DOI: 10.1021/acs.jctc.7b00577.

[141] Jmol development team. *Jmol*. Ver. 14.6.4. 152016. URL: http://jmol.sourceforge.net/.

[142] Thomas A. Halgren. "Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94". En: *Journal of Computational Chemistry* 17.5-6 (1996), págs. 490-519. DOI: 10.1002/(SICI)1096-987X(199604)17:5/6<490::AID-JCC1>3.0.CO;2-P. URL: https://doi.org/10.1002/(SICI)1096-987X(199604)17:5/6<490::AID-JCC1>3.0.CO;2-P.

[143] Thomas A. Halgren. "Merck molecular force field. V. Extension of MMFF94 using experimental data, additional computational data, and empirical rules". En: *Journal of Computational Chemistry* 17.5-6 (1996), págs. 616-641. DOI: 10.1002/(SICI)1096-987X(199604)17:5/6<616::AID-JCC5>3.0.CO;2-X. URL: https://doi.org/10.1002/(SICI)1096-987X(199604)17:5/6<616::AID-JCC5>3.0.CO;2-X.

[144] Thomas A. Halgren. "MMFF VI. MMFF94s option for energy minimization studies". En: *Journal of Computational Chemistry* 20.7 (1999), págs. 720-729. DOI: 10.1002/(SICI)1096-987X(199905)20:7<720::AID-JCC7>3.0.CO;2-X. URL: https://doi.org/10.1002/(SICI)1096-987X(199905)20:7<720::AID-JCC7>3.0.CO;2-X.

[145] Junmei Wang *et al.* "Development and Testing of a General Amber Force Field". En: *Journal of Computational Chemistry* 25.9 (2004), págs. 1157-1174. DOI: 10.1002/jcc.20035. URL: https://doi.org/10.1002/jcc.20035.

[146] Tommi Hassinen y Mikael Peräkylä. "New energy terms for reduced protein models implemented in an off-lattice force field". En: *Journal of Computational Chemistry* 22.12 (2001), págs. 1229-1242. DOI: 10.1002/jcc.1080. URL: https://doi.org/10.1002/jcc.1080.

[147] A. K. Rappe *et al.* "UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations". En: *Journal of the American Chemical Society* 114.25 (1992), págs. 10024-10035. DOI: 10.1021/ja00051a040. URL: https://doi.org/10.1021/ja00051a040.