Optimización de modelos neuronales para estimar similaridad entre compuestos mediante estrategias basadas en computación evolutiva

Tobías J. Hermann, Leandro D. Vignolo y Matias F. Gerard

Instituto de Investigación en Señales, Sistemas e Inteligencia Computacional, sinc(i), FICH-UNL, CONICET, Ciudad Universitaria UNL, 3000, Santa Fe, Argentina.
{tjhermann,ldvignolo,mgerard}@sinc.unl.edu.ar

Resumen El concepto de similaridad molecular es fundamental para la bioinformática. Sin embargo, calcular la similaridad entre compuestos cuando se desconoce la estructura de alguno de ellos resulta un desafío. Los modelos neuronales en grafos han demostrado eficacia para extraer representaciones a partir de la topología de reacciones químicas. Sin embargo, su diseño y principalmente la elección de hiperparámetros implican evaluar un vasto espacio de posibilidades. Los algoritmos evolutivos surgen como una alternative natural para explorar grandes espacios de búsqueda, como es el caso del espacio de hiperparámetros asociados a las arquitecturas neuronales. Este trabajo propone comparar un el uso de un enfoque clásico de búsqueda de hiperparámetros mediante conocimiento experto frente a una propuesta bioinspirada basada en computación evolutiva para la misma tarea, particularmente en el contexto de estimación de similaridad entre compuestos. Partiendo de una arquitectura predefinida se comparan experimental ambas propuestas y se evalúan en diferentes conjuntos de datos. Los resultados muestran que el enfoque basado en computación evolutiva permite encontrar hiperparámetros adecuados para la arquitectura considerada que permiten alcanzar un desempeño comparable al enfoque basado en conocimiento experto, con la diferencia de no ser necesario el conocimiento humano para esta selección.

Keywords: Redes neuronales en grafo · Computación evolutiva · Similaridad entre compuesto · Vías metabólicas

1. Introducción

El concepto de similaridad molecular es fundamental para la bioinformática, y se basa en el supuesto de que moléculas estructuralmente similares suelen tener propiedades similares. La predicción de propiedades fisicoquímicas mediante la comparación con otros compuestos con estructura similar [5], el cribado virtual de compuestos para la selección de candidatos farmacéuticos [8], y el diseño de vías metabólicas guiado por similaridad entre compuestos [12] son algunos ejemplos de aplicaciones.

En la literatura se dispone de una amplia variedad de métodos para evaluar similaridad entre compuestos, para los cuales se requieren dos elementos claves: representaciones estructurales y métricas para evaluar su similitud [10]. Las

representaciones estructurales pueden basarse en la estructura 2D o 3D. Los descriptores moleculares y las fingerprints, son descripciones típicamente empleadas que se generan a partir de la representación 2D. En cambio, la similitud de forma, las técnicas basadas en campos y las fingerprints 3D son descripciones que se extraen a partir de la estructura tridimensional. Con respecto a las métricas de similitud, las más relevantes son el coeficiente de Tanimoto, Dice, distancia coseno y euclidiana, siendo el coeficiente de Tanimoto el más utilizado [3]. Si bien estos enfoques producen buenos resultados, su uso requiere conocer la estructura de los dos compuestos a comparar.

Recientemente, se ha propuesto un nuevo enfoque para abordar esta problemática en el caso donde los compuestos participan en vías metabólicas [4]. El modelo propuesto emplea una red neuronal de paso de mensajes (MPNN) para aprender representaciones de compuestos químicos a partir de una codificación one-hot inicial y la topología de la red metabólica en la que participan. La MPNN utiliza un mecanismo de atención para combinar las características de cada compuesto con las de sus vecinos. Estas representaciones son luego empleadas por el modelo para inferir la similitud entre pares de compuestos. El modelo se entrena con compuestos de estructura conocida para estimar la similaridad. Para compuestos con estructura desconocida, se infiere la similitud directamente de las representaciones aprendidas por la MPNN, sin requerir información estructural explícita, y se genera un valor de similaridad estimado. Pese a los resultados obtenidos, este modelo posee un importante número de hiperparámetros que deben ser adecuadamente seleccionados. Para tal efecto, debe utilizarse una estrategia experimental rigurosa para lograr un correcto ajuste y buen desempeño, tal y como ocurre con otros modelos basados en aprendizaje profundo. Sin embargo, la exploración completa del espacio de combinaciones inducida por los hiperparámetros y sus posibles valores resulta computacionalmente infactible.

Los algoritmos evolutivos son métodos de optimización y búsqueda, inspirados en la evolución natural, que resultan especialmente adecuados para explorar grandes espacios de soluciones [2]. Estos algoritmos mantienen una población de individuos que representan posibles soluciones a un problema, donde los individuos compiten entre sí para generar descendencia, siguiendo el principio de selección natural. A través de operadores de selección, cruza y mutación, los individuos más aptos sobreviven y se reproducen, generando nuevas soluciones que se evalúan continuamente en función de resolver un problema específico. Estos métodos han sido exitosamente empleados en problemas de optimización [14], búsqueda de representaciones óptimas [16] y selección de características para reconocimiento del habla [15], entre otros. A su vez, debido a su capacidad para explorar de manera inteligente y eficiente extensos espacios de búsqueda, los algoritmos evolutivos han tomado un papel preponderante en el diseño de redes neuronales, en lo que hoy se conocen como métodos NAS (del inglés Neural Architecture Search) [11]. Estos métodos se han empleado exitosamente en diferentes tareas dentro del campo de la visión computacional, entre las que se destacan la detección de objetos, la segmentación de imágenes y las redes generativas adversarias [9]. Las soluciones encontradas por estos enfoques frecuente-

mente producen modelos neuronales más eficientes y con desempeño superior a las mejores arquitecturas diseñadas por expertos.

Por lo expuesto, en este trabajo se propone comparar la capacidad de un modelo neuronal diseñado y optimizado empleando un enfoque tradicional guiado por conocimiento experto, frente a un modelo obtenido mediante una optimización evolutiva de los hiperparámetros. Partiendo de la arquitectura propuesta en [4] y utilizando el mecanismo de atención propuesto en [13], primero se realiza un análisis exploratorio del impacto de los hiperparámetros en el desempeño del modelo y luego se realiza la optimización de los más relevantes mediante una búsqueda en grilla. A su vez, se emplea un algoritmo evolutivo para seleccionar los valores de los hiperparámetros para la misma arquitectura, utilizando el desempeño de cada solución para guiar la exploración.

En adelante, el trabajo se organiza de la siguiente manera. En la Sección 2 se describe el proceso de construcción de los datos, y se detalla la arquitectura neuronal y el esquema general seguido por el algoritmo evolutivo. La Sección 3 explica cómo se construyeron los patrones y que medidas se tienen en cuenta para evaluar el rendimiento de los modelos. Seguidamente, la Sección 4 presenta los resultados obtenidos para el modelo diseñado mediante conocimiento experto y el obtenido empleando el enfoque evolutivo. También se comparan las predicciones de similaridad para ambos modelos frente a compuestos con estructura desconocida. Finalmente, la Sección 5 resume las conclusiones más importantes del trabajo.

2. Algoritmos empleados

2.1. Red neuronal de paso de mensajes

Las redes neuronales de paso de mensajes (MPNN, del inglés Message Passing Neural Network) [6] son arquitecturas de aprendizaje profundo que operan sobre grafos dirigidos o no dirigidos, los cuales pueden tener características asociadas a los nodos y a las aristas. Estos modelos son capaces de aprender a partir de datos estructurados como grafos, mediante el intercambio de información entre los nodos del mismo. Esto permite que las MPNN actualicen la información asociada a los nodos del grafo mediante la agregación de la información de sus nodos vecinos. A su vez, la repetición de este proceso mediante la concatenación de múltiples capas de MPNN, o la realimentación de la salida, permite que los nodos se actualicen con información de vecinos cada vez más lejanos. Dado que la información es transformada en cada paso mediante funciones con parámetros aprendibles a partir de los datos, el modelo resultante puede entrenarse para optimizar las representaciones para cada tarea considerada.

El aprendizaje en estos modelos se realiza en tres pasos: construcción y envío del mensaje, agregación de los mensajes y actualización de la información. Durante la construcción del mensaje, el modelo neuronal actualiza la información $x_i \in R^D$ del i-esimo nodo utilizando la información x_j de los nodos j pertenecientes a la vecindad S_i del nodo i. Para esto, cada nodo j construye un mensaje

de acuerdo a:

$$m_j = M(x_i, x_j, e_{ij}) \tag{1}$$

donde x_i corresponde a la información almacenada en el nodo i, x_j la información asociada al nodo j, y e_{ij} la información que pudiera estar asociada al arco que conecta ambos nodos. Esto se realiza empleando una función $M()$ con parámetros aprendibles, que procesa la información para crear el mensaje. A continuación, los mensajes son enviados al nodo central i, que los recibe y combina empleando la ecuación:

$$X_k = C(m_k) \forall k \in S_i \tag{2}$$

donde $C()$ es una función invariante al orden que combina la información de todos los nodos de la vecindad S_i del nodo i. Luego se construye una versión actualizada x'_i de la información del nodo i, de acuerdo a la ecuación:

$$x'_i = U(x_i, X_k) \tag{3}$$

donde $U()$ es una función que combina la información del nodo central y de su vecindad. Aunque en su versión más sencilla esta función realiza la suma del mensaje a la información del nodo i, versiones más avanzadas pueden realizar transformaciones definidas mediante parámetros aprendibles que empleen la información combinada.

2.2. Arquitectura neuronal propuesta

La arquitectura neuronal empleada en este trabajo se basa en la propuesta presentada en [4]. Este modelo consiste en dos grandes bloques: codificador y regresor. Mientras que el primer bloque se encarga de realizar extracción de características de los datos, el segundo bloque utiliza dichas características para alimentar un modelo neuronal que cumple las funciones de regresor. Dado que todo el proceso se encuentra mediado por funciones con parámetros aprendibles, la propuesta consiste en un enfoque end-to-end que aprende de los datos. La Figura 1 presenta un esquema general de la arquitectura propuesta.

El bloque codificador recibe como entrada un grafo G' con una estructura particular: dos nodos centrales (indicados en rojo y verde en la figura), los nodos vecinos para cada uno de ellos, así como las conexiones entre los nodos centrales y sus vecinos. A su vez, los nodos del grafo tienen asociadas un conjunto de características $X' \in R^{nxd}$, donde n es el número de nodos y d el número de características asociadas a cada nodo. Estas características son inicialmente proyectadas a un nuevo espacio, empleando un perceptrón multicapa (MLP) para generar embeddings $H \in R^{nxh}$. El tamaño de estos embeddings se denomina Embedding Size (ES). Luego, estas características son alimentadas a una capa MPNN que utiliza un mecanismo de atención para actualizar las características de cada nodo. A diferencia del modelo original [4], nuestra propuesta emplea el mecanismo de atención propuesto por Veličković *et al.* [13]. Este enfoque emplea un MLP para determinar el valor de atención que debe aplicar a cada elemento. Primero se proyectan los vectores de tamaño ES mediante $\Theta_s x_i$ hacia

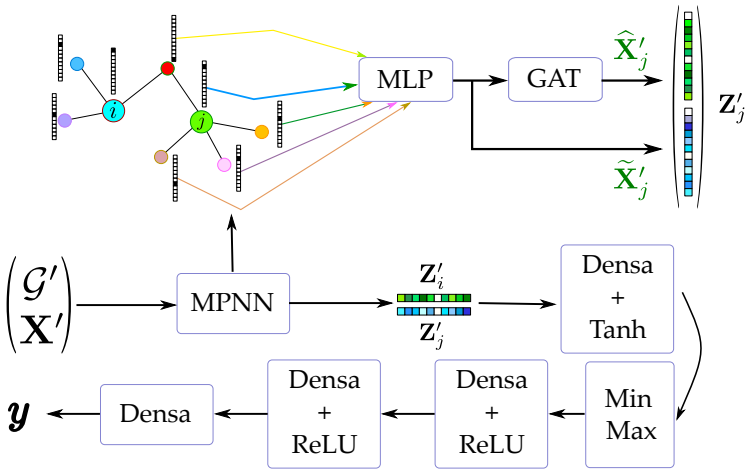

Figura 1: Arquitectura propuesta

un espacio de tamaño GC. Luego, dados los vectores de características x_i y x_j, correspondientes al nodo central i y a un nodo vecino j, este modelo concatena las características y determina el valor α_{ij} de atención que debe aplicar al vector x_j. Así, este permite ajustar el peso de la contribución de las características de los nodos vecinos durante la actualización de las características del nodo central x_i. Cada MLP empleado por este mecanismo para actualizar x_i constituye lo que se conoce como *cabeza de atención*. Al emplear más de un MLP (Head Count HC), es posible obtener múltiples versiones actualizadas de x_i, lo que permite enriquecer la representación de los nodos. Una vez actualizada la información de cada nodo, la MPNN retorna un nuevo vector de características Z_k, obtenido luego de aplicar una función no lineal a la concatenación de las características $\widehat{x}'_k$ y $\widetilde{x}'_k$ de cada nodo k. El tamaño de las características mencionadas se denomina como Hidden Features (HF). Son luego los vectores Z_i y Z_j correspondientes a los nodos centrales del grafo los que se emplean para realizar la inferencia. Primeramente, las características de ambos vectores son concatenadas y espacialmente reacomodadas para mantener la invarianza en el orden en que se analizan los nodos i y j. A continuación, este nuevo vector es procesado por una serie de capas completamente conectadas seguidas de funciones no lineales para obtener el valor de predicción.

2.3. Algoritmo Evolutivo

Los algoritmos evolutivos (AE) son una familia de métodos de búsqueda y optimización que imitan el proceso natural de evolución de los seres vivos. La gran flexibilidad y robustez de estos algoritmos les permite encontrar buenas soluciones en espacios de búsqueda complejos [7], además de presentar un paralelismo implícito que impacta en su aplicabilidad a diversos problemas. Un AE

Algorithm 1 Algoritmo Evolutivo

1: **Inicialización:** Generar una población inicial de individuos aleatorios.
2: **Evaluación:** Calcular la aptitud de cada individuo de la población.
3: **while** Generación actual <Generación máxima **do**
4: **Selección:** Seleccionar individuos para la reproducción basados en su aptitud.
5: **Cruza:** Cruzar los individuos seleccionados para crear descendencia.
6: **Mutación:** Aplicar mutaciones a la descendencia para mantener la diversidad.
7: **Reemplazo:** Reemplazar la población actual con la nueva generación.
8: **Evaluación:** Calcular la aptitud de cada nuevo individuo.
9: **end while**
10: **Resultado:** Devolver el mejor individuo encontrado.

en su forma más básica consiste en tres operadores: selección, operación genética y reemplazo. La población está formada por un grupo de individuos cuya información se codifica en cromosomas, que se inicializan de forma aleatoria, y de ella se eligen los candidatos para la solución de un problema. El desempeño de cada individuo dentro de la población se mide mediante un valor de aptitud, que se obtiene calculando la función objetivo en base a su información genética. Esta función es la que simula a la presión selectiva del ambiente en la evolución natural. Como se puede ver en el algoritmo 1, en cada iteración del algoritmo se selecciona un grupo de individuos, cada uno con una probabilidad proporcional a su aptitud, para generar la descendencia mediante las operaciones genéticas de cruzamiento y mutación. La población actual es luego reemplazada por su descendencia, para lo cual existen diferentes estrategias. Una de ellas es el elitismo, que consiste en tomar los mejores individuos de la generación actual, para que formen parte de la siguiente generación sin sufrir variaciones. El ciclo de un AE es repetido hasta que se alcanza un criterio de finalización deseado.

2.4. Estrategia de optimización del modelo neuronal

La elección de lo hiperparámetros adecuados para el modelo neuronal propuesto en la Sección 2.2 requiere la exploración de un espacio de configuraciones de elevada dimensionalidad. Esto lleva a que cada posible configuración deba ser evaluada para determinar la que produce el mejor desempeño. Sin embargo, la exploración de todas las posibles combinaciones en un tiempo razonable no resulta factible.

En este contexto se propone el uso de un algoritmo evolutivo para realizar la exploración inteligente del espacio de posibilidades. El proceso de optimización sigue el esquema clásico descripto en el algoritmo 1. Los individuos de la población se construyen mediante la combinación de los distintos valores que pueden tomar cada uno de los hiperparámetros que son considerados en la optimización del modelo neuronal. La evaluación del fitness corresponde a la construcción del modelo neuronal con los hiperparámetros definidos por cada individuo y la predicción de valores en una partición de test, luego de haber entrenado el modelo con datos independientes de entrenamiento. Esto permite guiar la búsqueda ha-

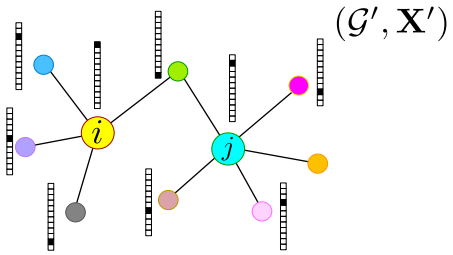

Figura 2: Patrón empleado en el entrenamiento del modelo neuronal. Los nodos centrales i,j corresponden a los compuestos entre los que estimar la similaridad. Cada compuesto del grafo posee un vector de características one-hot único.

cia configuraciones de mayor desempeño. La exploración del espacio viene así dada por el fitness asociado a los individuos y los operadores de selección y variación que combinan las soluciones disponibles.

3. Datos de entrenamiento y medidas de evaluación

En esta Sección se describe cómo se construye el dataset de entrenamiento y cuáles son las medidas empleadas para evaluar el desempeño de los modelos neuronales. Para facilitar la lectura, el proceso de construcción del dataset es dividido en dos etapas: el patrón a partir del cual aprender las características para realizar la inferencia, y el valor real asociado a dicho patrón que se busca aprender. Estas serán descriptas en detalle en subsecciones independientes.

3.1. Construcción de patrones

Una forma común de modelar las vías metabólicas es a través del uso de grafos de compuestos [1]. Dado un grafo $G = v, e$, es posible modelar cada compuesto de la vía como un vértice v, y establecer una conexión o arista e entre dos compuestos, si estos participan como sustrato y producto en una misma reacción. A partir de esta representación es posible generar un grafo G', requerido para alimentar el modelo neuronal descripto en la Sección 2.2, extrayendo de G los dos nodos entre los que se busca estimar la similaridad, sus primeros vecinos, y las conexiones entre ellos. La Figura 2 presenta un ejemplo de subgrafo.

A partir de este procedimiento se construyeron 2 conjuntos de datos para evaluar los diferentes modelos. El primero, denominado ecoli30, se construyó empleando las 29 reacciones que forman parte de la vía metabólica de la glucólisis en la bacteria E. coli. El grafo inducido a partir de estos datos cuenta con 76 nodos y 117 aristas. Para la construcción del segundo conjunto de datos, denominado eco6path, se emplearon todas las reacciones conocidas para 6 vías metabólicas de referencia: glicólisis (rn00010), ciclo de Krebs (rn00020), pentosas fosfato (rn00030), metabolismo de la sacarosa y del almidón (rn00500),

metabolismo del piruvato (rn00620), y metabolismo del propanoato (rn00640). En este caso, el grafo resultante contó con 273 nodos y 533 aristas. En ambos casos, las reacciones fueron obtenidas de la base de datos KEGG. Para cada conjunto de datos, se creó una lista de los nodos que lo conforman, seguido de la construcción de vectores de características one-hot para cada nodo, derivados de su posición en esta lista. Estos vectores fueron empleados como vectores de características en los experimentos posteriores. Finalmente, cada patrón quedó constituido por el subgrafo G' inducido a partir de dos nodos i,j centrales, y el conjunto X' los vectores de características para los nodos participantes. Luego del procesamiento, ecoli30 cuenta con 787 patrones, mientras que el conjunto de datos eco6path quedó conformado por 7764 patrones.

3.2. Cálculo de la similaridad real

Los valores de similaridad se obtuvieron experimentalmente a partir de la estructura molecular correspondiente a los dos nodos centrales de cada patrón. Empleando la base de datos PubChem se descargó la estructura molecular para cada compuesto, cuando se encontraba estaba disponible. Luego, dichas estructuras fueron procesadas para obtener las fingerprints correspondientes que describen a cada compuesto. Estas fueron construidos utilizando el algoritmo de Morgan y la librería RDKit. En todos el algoritmo se configuró para utilizar radio $r = 3$ y 2048 bits para cada estructura. Finalmente, el cálculo de similaridad se realizó utilizando el coeficiente de Tanimoto, definido como:

$$T(m_i, m_j) = \frac{\sum_k (m_i^k \wedge m_j^k)}{\sum_k (m_i^k \vee m_j^k)} \tag{4}$$

donde $\wedge$ y $\vee$ son los operadores binarios *and* y *or*, respectivamente, y m_i y m_j son las representaciones binarias de las fingerprints para las estructuras de los compuestos i y j.

3.3. Medidas de desempeño

El desempeño de las arquitecturas propuestas se evaluó empleando dos medidas: error cuadrático medio (MSE) y el coeficiente de correlación de Pearson (ρ). El MSE se calcula de acuerdo a la ecuación:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{5}$$

donde y_i es el valor de similaridad predicho para el patrón i, e $\hat{y}_i$ es el valor real calculado a partir de las estructuras moleculares y el coeficiente de Tanimoto. Por su parte, el coeficiente de Pearson se calcula como:

$$\rho = \frac{\mathrm{cov}(X, Y)}{\sqrt{\mathrm{var}(X) \cdot \mathrm{var}(Y)}} \tag{6}$$

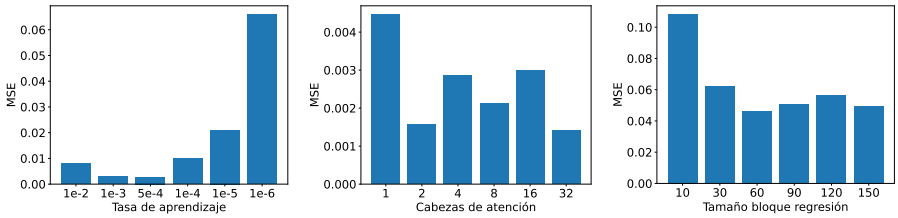

Figura 3: Resultados de la exploración preliminar de hiperparámetros. a) Efecto de la tasa de aprendizaje. b) Número de cabezas de atención. c) Tamaño de características del bloque de regresión. En todos los casos se muestra el efecto de la variación del error en función de los valores que adoptan los hiperparámetros.

donde X e Y, var(X) y var(Y) corresponden a la varianza de las variables X e Y, y cov(X, Y) es la covarianza de las variables. Esta medida varía en el rango $[-1, +1]$, e indica una correlación perfecta entre ambas variables cuando ésta toma alguno de los valores extremos. En particular, el valor de ρ se empleó para caracterizar la calidad de predicción del modelo, al determinar cuán correlacionadas se encontraban los valores de similaridad predichos y reales.

4. Resultados

4.1. Optimización de hiperparámetros basada en conocimiento

El primer paso para la optimización de hiperparámetros consistió en un análisis exploratorio para determinar la influencia sobre la arquitectura. Se identificaron 5 hiperparámetros para realizar el estudio preliminar: tasa de aprendizaje LR, tamaño de características del bloque de regresión HF, embedding size ES, tamaño del vector de entrada al mecanismo de atención GC y cantidad de cabezas de atención H. Los experimentos consistieron en tomar una arquitectura con parámetros iniciales seleccionados a partir de pruebas preliminares, y analizar el impacto producido al variar cada uno de ellos de forma individual. Esta arquitectura inicial posee los siguientes hiperparámetros: LR=5E-4, HF=50, ES=100, GC=100 y HC=1.

El modelo fue implementado utilizando las librerías Pytorch y Pytorch Geometric. El entrenamiento se realizó utilizando el optimizador Adam y MSE como función de costo. Se fijó un número máximo de épocas de 3000, y un término de paciencia de 400 para realizar la detención temprana. En todos los experimentos se emplearon los mismos datos de entrenamiento y validación (80 % del dataset), y una partición adicional para evaluar los modelos ya entrenados (20 %). Los experimentos se realizaron empleando el dataset ecoli30, empleando una PC equipada con un microprocesador intel i7-2600k y 16 GB de RAM.

Los resultados obtenidos en el análisis preliminar se presentan en la Figura 3. Como puede observarse, la variación del error en función de la tasa de aprendizaje

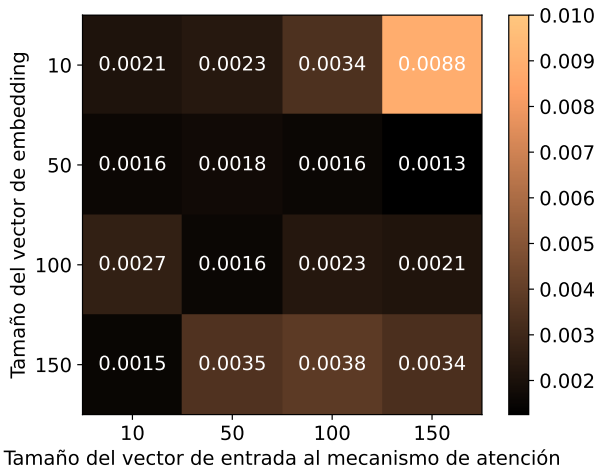

Figura 4: Cada valor en el mapa de colores representa el MSE elevado al cuadrado, para cada par de valores de HF y ES.

presenta un mínimo en el rango [1E-3,5E-4], siendo mínimo en el extremo inferior del rango, y aumentando luego al reducir el valor de LR. Esto puede deberse a que se emplea un número máximo de épocas que, aunque presenta un valor elevado, puede ser insuficiente cuando la tasa de aprendizaje cae por debajo de un determinado valor. Por su parte, no se aprecia una tendencia clara respecto a cómo varía el error en función del número de cabezas de atención. Sin embargo, nótese que cuando H=2 se obtiene el menor valor de error. Sólo se logra reducir el error al incorporar 32 cabezas de atención al modelo, pero logrando sólo una pequeña mejora a costa de aumentar considerablemente el número de parámetros aprendibles al modelo. En lo referente al tamaño del bloque de regresión, la Figura 3.c muestra que el menor error se logra para HF=60. En lo referente al tamaño del embedding ES y al tamaño del vector de entrada al mecanismo de atención GC, los resultados no fueron concluyentes. Por lo tanto, para estudiar el efecto que estos dos hiperparámetros producen sobre el modelo, se realizó una búsqueda en grilla. Se consideraron los valores para tamaño del embedding ES={10, 50, 100 y 150}, y GS={10, 50, 100 y 150}. Los resultados obtenidos se presentan en la Figura 4.

En la Figura 4 podemos observar los resultados de la búsqueda en grilla realizada para los parámetros GC y ES. Cada celda corresponde al MSE que logra la partición de testeo para cada par de valores de GC y ES. Como puede apreciarse, se obtienen los mejores resultados con ES = 50, seguido por ES=100. Al aumentar o disminuir el valor de ES, respecto de los valores anteriormente mencionados, se observa una tendencia a un mayor MSE. Con respecto a GC no hay una conclusión clara. Al aumentar el tamaño de GC, los resultados tienden

a mejorar para un ES=50, sin embargo, para los demás casos ocurre lo contrario, al aumentar el tamaño, el rendimiento disminuye. Finalmente, en base a estos resultados se concluye que la mejor combinación es ES = 50 y GC = 150, con un MSE de 0,0012535.

4.2. Optimización de hiperparámetros mediante computación evolutiva

El algoritmo empleado para llevar adelante la optimización de hiperparámetros mediante computación evolutiva se implementó utilizando la librería Deap. Los individuos se construyeron de manera de representar todas las posibilidades para cada hiperparámetro empleando codificación binaria. La cantidad de cabezas de atención se codificó empleando 2 bits para representar los valores H={1,2,4,8}. Para los parámetros ES, HF y GC se utilizaron 4 bits para representar el rango de valores [10,160], con intervalos de 10. Dado que se emplea un total de 14 bits, el espacio de búsqueda inducido contiene $2^{14} = 16{,}384$ posibles modelos a evaluar. Claramente, este espacio es mucho mayor al evaluado en la Sección 4.1, donde solo se consideraron 16 valores para la búsqueda en grilla. El modelo codificado por cada individuo fue entrenado y luego evaluado en una partición de test. El valor de MSE en esta partición fue empleado como valor de fitness del individuo y utilizado para guiar la búsqueda. En promedio, la evaluación de una población completa requiere entre 30 y 240 minutos, dependiendo del número de parámetros aprendibles que poseen los modelos en una dada generación. Los experimentos se realizaron empleando el operador de selección por torneo, empleando 3 individuos para la competencia. Además, se utilizó el operador de cruza en un punto con probabilidad 0,9. Las posiciones fueron elegidas al azar pero de manera de preservar la integridad de los genes. Las corridas se llevaron a cabo empleando elitismo con un único individuo que reemplaza aleatoriamente a otro en la nueva población. Se empleó un máximo de 100 generaciones, con un tamaño de población constante de 100 individuos y un esquema de selección de 20 padres para generar la descendencia. La probabilidad de mutación es de 1 gen por cromosoma, lo que representa una probabilidad de 7,14 % de mutar cada gen. Para controlar la convergencia prematura de la población, se introduce un mecanismo que aumenta la probabilidad de mutación si la varianza de la población es menor a 1E-5, y retorna a valores normales cuando se supera nuevamente el umbral anterior. Además, se incluye un mecanismo de reinicio de la población en caso de convergencia prematura. Los experimentos se ejecutaron en un Ryzen 9 5950X con 64GB de RAM. En promedio, las 100 generaciones de los 4 experimentos, requieren de 38 horas. Los hiperparámetros mencionados del algoritmo evolutivo fueron elegidos teniendo en cuenta valores utilizados típicamente en la literatura.

En la Tabla 1 se presentan los resultados obtenidos en 4 ejecuciones del algoritmo evolutivo. En cada caso se muestra el desempeño alcanzado y los hiperparámetros del modelo correspondiente. Por comparación, al final de la tabla se presentan los mismos resultados alcanzados con el mejor modelo de la Sección anterior. Puede observarse que los 4 modelos presentan un buen desempeño, con

Tabla 1: Desempeño obtenido por los modelos optimizados empleando el enfoque basado en conocimiento y el enfoque basado en computación evolutiva.

Modelo	**MSE**	**HC**	**ES**	**HF**	**GC**
Evolutivo 1	0.00077	1	90	160	100
Evolutivo 2	0.00087	1	50	160	110
Evolutivo 3	0.00081	4	90	110	10
Evolutivo 4	0.00068	2	20	140	100
Conoc. experto	0.00125	2	50	60	150

un valor de MSE menor que el modelo diseñado empleando conocimiento experto. Además, el modelo evolutivo tiene a emplear valores de HF y GC en la parte superior del rango de posibilidades, favoreciendo la representación interna de los compuestos. También resulta llamativo que se emplea diferente número de cabezas de atención, pero siempre tendiendo a que dicho valor sea reducido. Otro aspecto interesante es que los hiperparámetros elegidos presentan algunas diferencias importantes con el modelo encontrado en la Sección anterior.

Para realizar una comparación justa de ambas propuestas, el modelo diseñado mediante conocimiento experto y el modelo Evolutivo 4 (que presenta el menor valor de MSE) se entrenaron y evaluaron empleando el dataset eco6path, utilizando el 80 % de los datos para entrenar y 20 % para evaluar. Los resultados muestran que el primer modelo produce un MSE=0,00093, con un valor R=0,987, mientras que el enfoque evolutivo produce un MSE=0,00104, y un valor R=0,985. Estos resultados muestran que ambos modelos presentan desempeños similares, dado que los errores en ambos casos son pequeños y existe una elevada correlación entre los valores de similaridad predichos y los calculados a partir de la estructura.

4.3. Comparación frente a compuestos con estructura desconocida

Teniendo en cuenta que ambos enfoques producen modelos con desempeños similares, se evaluó de forma cualitativa la calidad de las predicciones sobre los compuestos de una reacción que contiene compuestos con estructura desconocida. Para esto se utilizaron los compuestos de la reacción R01196 de KEGG, que posee compuestos con estructura conocida y desconocida. En particular, aunque se desconoce la estructura de los compuestos C00138 y C00139, se conoce de la literatura que ambos compuestos presentan una elevada similaridad. La Figura 5 presenta un esquema de la reacción analizada.

Empleando los modelos entrenados con el conjunto de datos eco6path se realizó la predicción de la similaridad de todos los compuestos indicados en la figura. Nótese que para los compuestos con estructura conocida, las similaridades estimadas con los modelos son similares a las calculadas a través de la estructura. Por ejemplo, la similaridad entre los compuestos C00024 y C00022 es baja y se predice correctamente (real: 0.041, manual: 0.042, evolutivo: 0.030), mientras que C00024 presenta una elevada similaridad con C00010 (real: 0.855, manual:

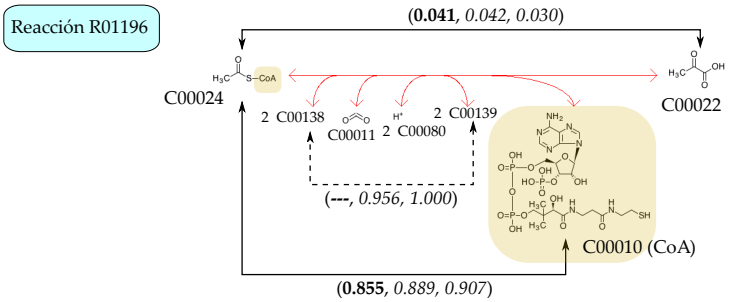

Figura 5: Comparación de los valores de similaridad calculados con diferentes modelos para una reacción que incluye compuestos con estructura desconocida. Las flechas indican los pares de compuestos entre los que se estima la similaridad. Entre paréntesis se muestra la similaridad real, la predicha con el modelo optimizado con el enfoque basado en conocimiento experto y con el enfoque evolutivo.

0.889, evolutivo: 0.907) dado que su estructura representa la mayor parte del compuesto. A su vez, algo similar ocurre con los compuestos C00138 y C00139, ya que se predicen valores de similaridad elevados con ambos modelos (manual: 0.956, evolutivo:1.0), lo que se condice con lo que se indica en la literatura.

5. Conclusiones y trabajos futuros

En este trabajo se propuso un enfoque basado en computación evolutiva para la optimización de los hiperparámetros de una arquitectura neuronal. El enfoque se comparó con una propuesta basada en conocimiento experto para la elección de los hiperparámetros, en el contexto de estimación de similaridad entre compuestos. El enfoque presentado se ha utilizado de manera similar en otras áreas, sin embargo, en optimización de modelos que se apliquen en vías metabólicas no se encuentran estudios similares. Los enfoques se evaluaron empleando un conjunto de datos pequeño para seleccionar los hiperparámetros de un modelo reportado en la literatura. Los mejores modelos fueron reentrenados luego con un conjunto de datos más amplio para evaluar la efectividad de la propuesta. Los resultados indican que los modelos optimizados empleando ambas estrategias poseen desempeños similares, y que ambos son capaces de predecir con buen desempeño la similaridad entre compuestos con estructura conocida y potencialmente, también entre compuestos para los que se desconoce la misma. Esto muestra que el enfoque evolutivo es una opción valiosa para ser considerada por usuarios no expertos para la optimización de hiperparámetros de sus modelos. Una de las limitaciones del enfoque es la codificación binaria, la cual no permite tener una densidad numérica infinita como la codificación real. Sin embargo, utilizar una codificación real amplía el espacio de búsqueda y con ello el costo computacional. Para estudios futuros se propone extender el espacio de

búsqueda para evaluar configuraciones con una mayor granularidad, y extender el estudio a vías metabólicas de mayor tamaño. Además, se propone evaluar experimentalmente los hiperámetros del algoritmo evolutivo para encontrar los más adecuados. También se considerará la utilización de la técnica de validación cruzada con el método k-fold en la evaluación de cada solución, para realizar una mejor estimación del desempeño del modelo y reducir el sesgo inducido por el uso de una partición en particular.

Referencias

1. Arita, M.: From Metabolic Reactions to Networks and Pathways, pp. 93–106 (2012)
2. Baeck, T., Fogel, D., Michalewicz, Z.: Evolutionary Computation 1: Basic Algorithms and Operators. CRC Press (2000)
3. Bajusz, D., Rácz, A., Héberger, K.: Why is tanimoto index an appropriate choice for fingerprint-based similarity calculations? Journal of Cheminformatics **7**(1), 20 (2015)
4. Borzone, E., Di Persia, L.E., Gerard, M.: Neural model-based similarity prediction for compounds with unknown structures. In: Applied Informatics. pp. 75–87 (2022)
5. Fujimoto, K.J., Minami, S., Yanai, T.: Machine-learning- and knowledge-based scoring functions incorporating ligand and protein fingerprints. ACS Omega **7**(22), 19030–19039 (2022)
6. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. Proceedings of Machine Learning Research, vol. 70, pp. 1263–1272 (2017)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
8. Hutter, M.C.: Differential multimolecule fingerprint for similarity searchmaking use of active and inactive compound sets in virtual screening. Journal of Chemical Information and Modeling **62**(11), 2726–2736 (2022)
9. Kang, J.S., Kang, J., Kim, J.J., Jeon, K.W., Chung, H.J., Park, B.H.: Neural architecture search survey: A computer vision perspective. Sensors **23**(3) (2023)
10. Kumar, A., Zhang, K.Y.J.: Advances in the development of shape similarity methods and their application in drug discovery. Frontiers in Chemistry **6** (2018)
11. Kyriakides, G., Margaritis, K.: Evolving graph convolutional networks for neural architecture search. Neural Computing and Applications **34**(2), 899–909 (2022)
12. Rahman, S.A., Advani, P., Schunk, R., Schrader, R., Schomburg, D.: Metabolic pathway analysis web service (Pathway Hunter Tool at CUBIC). Bioinformatics **21**(7), 1189–1193 (2004)
13. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2018)
14. Vignolo, L.D., Milone, D.H., Scharcanski, J.: Feature selection for face recognition based on multi-objective evolutionary wrappers. Expert Systems with Applications **40**(13), 5077–5084 (2013)
15. Vignolo, L.D., Rufiner, H.L., Milone, D.H.: Multi-objective optimisation of wavelet features for phoneme recognition. IET Signal Processing **10**(6), 685–691 (2016)
16. Vignolo, L.D., Rufiner, H.L., Milone, D.H., Goddard, J.C.: Evolutionary splines for cepstral filterbank optimization in phoneme classification. EURASIP Journal on Advances in Signal Processing **2011**, 8:1–8:14 (2011)