Universidad Nacional Autónoma de México

Facultad de Ciencias

Implementación de redes neuronales convolucionales para el estudio de interacciones proteína-ligando

T E S I S

QUE PARA OBTENER EL TÍTULO DE

Matemático

PRESENTA

Adrián Antonio Rodríguez Pié

TUTOR

Marcelino Arciniega Castro

Ciudad Universitaria, CDMX, 2019

II

Índice general

Introducción

El descubrimiento de nuevos fármacos es un proceso muy tardado y costoso; e incluso el reposicionamiento de compuestos ya conocidos es una tarea sumamente complicada. El escenario es aún más complejo si se toman en cuenta los miles o millones de moléculas capaces de ser sintetizadas en cada etapa del desarrollo. Para superar estas dificultades, el uso de alternativas computacionales de bajo costo es altamente recomendado y se ha adoptado como la forma estándar de ayuda para el desarrollo de nuevos fármacos.

Una de las metodologias computacionales más usadas para investigar las interacciones proteína-ligando es el acoplamiento molecular. La selección de los ligandos con mayor afinidad utilizando Cribado Virtual basado en Acoplamiento (DBVS, *Docking Based Virtual Screening*) es realizada mediante la inserción de cada elemento de la bibilioteca en una región particular de un receptor objetivo.

En la primera etapa del proceso se procede a una búsqueda heurística en la que miles de posibles inserciones son consideradas. En la segunda etapa, la calidad de la inserción es calificada a través de una función evaluadora. Esta última fase se ha convertido en todo un reto para los científicos computacionales, por la dificultad de decidir de forma determinística si un acoplamiento es bueno o no.

Los sistemas basados en aprendizaje de máquina (ML, *Machine Learning*) han sido empleados con éxito para mejorar la salida del DBVS para incrementar el desempeño de las funciones evaluadoras así como para

construir clasificadores de afinidad de enlace. Una de las principales ventajas de utilizar ML es la capacidad de capturar la dependencia no lineal de las interacciones moleculares entre ligando y receptor.

En este trabajo se propone un acercamiento con una red neuronal convolucional para mejorar el DBVS. El método utiliza los resultados de una simulación de acoplamiento como entrada, en donde automáticamente aprende a extraer características relevantes a partir de datos básicos como tipos de átomo, distancias entre ellos, y su contexto en la rama estructural. La red aprende características abstractas que son útiles para discriminar entre poses válidas y señuelos en un acoplamiento.

CAPÍTULO 1

Sobre inteligencia artifical

Un factor que ha sido central en la historia del ser humano es precisamente el que nos da el nombre como especie *Homo sapiens*: la inteligencia. Durante años se ha buscado entender cómo es que pensamos; es decir, la forma en que un ente puede percibir el entorno y, a partir de ello, entenderlo e incluso manipularlo y hacer predicciones al respecto. La inteligencia artificial (IA) lleva este estudio un paso más adelante: busca no sólo entender sino también construir entes inteligentes.

1.1. La prueba de Turing

Dado que la inteligencia artificial busca crear entes inteligentes, es importante definir primero qué es la inteligencia. Al ser nosotros mismos como especie nuestro modelo de inteligencia, es razonable pensar en la inteligencia en términos de pensamiento humano.

La **prueba de Turing**, propuesta por Alan Turing (1950), fue diseñada para dar una definición de inteligencia en computadoras. Una computadora pasa la prueba si, tras ser interrogada de forma escrita por un evaluador humano, éste no puede distinguir si las respuestas provienen de un humano o una computadora. La discusión sobre si pasar la prueba realmente es señal de una *inteligencia* sale del alcance de esta tesis, pero es importante resaltar que la prueba da cierta noción sobre lo que se

busca en una "inteligencia artificial". Se busca que el ente "inteligente" sea capaz de tener un criterio racional. Es decir, que pueda sacar conclusiones lógicas a partir de premisas dadas mediante silogismos, como lo haría un humano. Pero rara vez toda la información con la que se cuenta es 100 % acertada, lo que echa abajo el enfoque puramente determinista de la inteligencia.

1.2. Agentes

En su concepción más tradicional, la inteligencia artificial gira en torno a **agentes racionales** que, mediante **sensores** pueden percibir su **entorno** y actuar sobre él a partir de un sistema de decisión. Internamente, el agente es una máquina compuesta por un conjunto (finito) de estados, cuyas transiciones están dadas por reglas de inferencias. Cuando se da una transición de estados, entonces es ejecutada una acción. Esta concepción del agente nos permite reducir la IA a un problema de búsqueda de las funciones de transición.

Computacionalmente hablando, la búsqueda de estas funciones en ciertos problemas de IA se vuelve absurdamente compleja, por lo que no es factible hacer la codificación de todos los posibles escenarios en el sistema de inferencia de un agente. Por lo tanto, un agente debe ser capaz de manejar la incertidumbre en su entorno. Además, debe poder *aprender* del entorno y adaptarse a él, generando conocimiento por medio de la experiencia.

El **aprendizaje de máquina** (ML, *Machine Learning*) gira en torno a algorítmos capaces de manejar información estocástica y generar modelos de toma de decisiones a partir de ella. **Un modelo es bueno en la medida de lo acertado que es en términos estadísticos**.

1.3. Tipos de aprendizaje

El aprendizaje de máquina se ha divido en varios subcampos, cada uno atacando un tipo de problema distinto y utilizando diferentes formas de aprendizaje. Se describen a continuación cuatro parámetros bajo los cuales es posible clasificar los distintos paradigmas de aprendizaje.

Supervisado/no supervisado. Dado que el aprendizaje involucra una interacción entre el agente y el ambiente, es posible dividir el aprendizaje con respecto a la naturaleza de dicha interacción. La primera distinción a notar es la diferencia entre aprendizaje supervisado y no supervisado. En abstracto, viendo el aprendizaje como un proceso de "usar la experiencia para ganar *maestría*", el aprendizaje supervisado describe un escenario en el que la "experiencia", un ejemplo de entrenamiento, contiene información significativa que no está contenida en los "ejemplos de prueba" que el sistema aún no ha visto, y en los que se busca aplicar la *maestría* adquirida. En esta configuración, la *maestría* adquirida busca predecir la información faltante de los datos de prueba. Podemos pensar en el ambiente como un profesor que "supervisa" al agente al proporcionarle información extra (una clasificación). Por otro lado, en el aprendizaje no supervisado no hay distinción entre datos de entrenamiento y de prueba. El agente procesa los datos de entrada con el objetivo de generar una síntesis o versión comprimida de los datos que sea representativa.

Agentes pasivos/activos. En los paradigmas de aprendizaje el rol tomado por el agente varia. Es posible distinguir entre agente activo y pasivo. Un agente activo interactúa con el ambiente durante el entrenamiento, haciendo consultas o experimentos, mientras que un agente pasivo sólo observa la información provista por el ambiente sin influir o dirigirla durante el proceso de aprendizaje.

Ayuda del profesor. Cuando se piensa en el aprendizaje humano el proceso, por lo general, involucra a un profesor que está tratando activamente de proveer al agente con la información más útil para lograr el objetivo de aprendizaje. En contraste, cuando un científico aprende sobre la naturaleza o el ambiente, tomando el rol de profesor, puede ser pensado como pasivo: las manzanas caen, las estrellas brillan y la lluvia cae sin cuidado por el aprendizaje del agente. Dichos escenarios son modelados postulando que los datos de aprendizaje (o la experiencia del agente) son generados por un proceso aleatorio.

Protocolo de entrenamiento lineal (o en línea)/por lote. El último parámetro a mencionar es la distinción entre las situaciones en las que el agente tiene que responder de forma lineal; es decir, que para cada ejemplo de entrenamiento tiene que dar una respuesta, y el agente tiene

que mostrar el aprendizaje adquirido después de tener la oportunidad de procesar grandes cantidades de datos.

CAPÍTULO 2

Sobre redes y neuronas

Los modelos de redes neuronales artificiales (ANN *Artificial Neural Network*), uno de los cuales es el *perceptrón* que discutiremos en este capítulo, están inspirados en el cerebro humano. Hay científicos cognitivos y de neurociencia cuya meta es entender el funcionamiento del cerebro y construir modelos que repliquen las redes neuronales naturales del cerebro.

Sin embargo, lo que busca la inteligencia artificial es construir máquinas *útiles* tomando como modelo el cerebro. El cerebro es un dispositivo de procesamiento de información con habilidades asombrosas en muchos campos que sobrepasan con creces a los más grandes esfuerzos de la ingeniería como son visión, aprendizaje y reconocimiento del habla, por nombrar algunos.

2.1. Inspiración en la biología

El cerebro humano es muy diferente de una computadora. Mientras una computadora tiene un número reducido de procesadores, el cerebro está compuesto de una enorme cantidad (10^{11}) de unidades de procesamiento llamadas **neuronas** trabajando en paralelo.

Aunque los detalles son inciertos, se cree que las neuronas son mucho más simples y lentas que un procesador de una computadora [1]. Lo que

hace al cerebro distinto, y le da su gran poder computacional, es su gran conectividad: cada neurona en el cerebro tienen conexiones, llamadas *sinapsis*, con alrededor de otras 10^4 neuronas.

En una computadora, el procesador es activo y la memoria está separada operando de forma pasiva (el procesador accede a ella sólo cuando se requiere); se cree que en el cerebro, tanto el procesamiento como la memoria están distribuidos por toda la red. El procesamiento es efectuado por las neuronas y la memoria se encuentra en las sinapsis entre ellas.

El cerebro es, además, un órgano capaz de adaptarse a las condiciones de su ambiente, ya que constantemente son agregadas nuevas conexiones sinápticas entre neuronas y modificadas las ya existentes. Una vez que una neurona ha emitido una señal eléctrica, las adyacentes reciben la información por medio de canales de transmisión llamados *dendritas*. Estos impulsos son llevados hasta el *núcleo* de la neurona para su procesamiento y, posteriormente, una reacción es transmitida a través del *axón* de la célula [4].

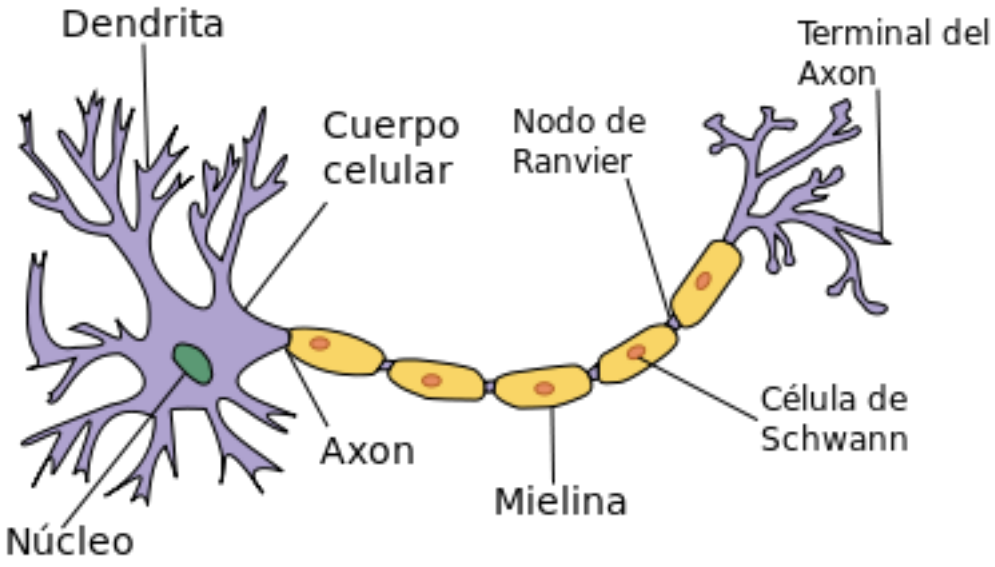

Figura 2.1: Estructura de una neurona. (Tomado de `https://es.wikipedia.org/wiki/Neurona`)

2.2. El perceptrón

En 1943, Warren McCullock y Walter Pitts publican la primera aproximación de una neurona simplificada, tratando de entender cómo funciona el cerebro biológico para el diseño de inteligencia artificial, la llamada neurona McCullock-Pits (MCP) [12].

McCullock y Pitts describen a la neurona como una compuerta lógica sencilla con una salida binaria; múltiples señales llegan a las dendritas para ser integradas al cuerpo de la célula. Si la señal acumulada excede cierto umbral, se genera una señal de salida que se le pasa al axón.

Unos años después, Frank Rosenblatt publicó la primera aproximación al concepto de perceptron basado en el modelo de neuronas MCP [17]. Intuitivamente, el algoritmo aprende automáticamente los coeficientes de pesos óptimos que luego se multiplican con las características de entrada para tomar la decisión de si la neurona se activa o no. Este algoritmo podría ser usado entonces para predecir si una muestra pertenece a una clase o a otra [16].

Formalmente, podemos plantearlo como un problema de clasificación binaria, donde nos referimos a nuestras dos clases, por simplicidad, como 1 (clase positiva) y -1 (clase negativa). Definimos también una *función de activación* $\phi(\mathbf{z})$ que toma una combinación lineal de ciertos valores de entrada $\mathbf{x}$ y un vector de pesos $\mathbf{w}$, donde $\mathbf{z}$ es lo que llamamos *entrada de la red* ($z = w_1x_1 + ... + w_mx_m$):

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

Si la activación de una muestra particular $x^{(i)}$ es mayor que un parámetro definido θ, predecimos la clase 1, y la clase -1 en caso contrario. En el algoritmo del perceptrón de Rosenblatt, la función de activación ϕ es una *función escalón*, que es llamada a veces la *función de Heaviside*:

$$\phi(z) = \begin{cases} 1 & \text{si } z \geq \theta \\ -1 & \text{en otro caso} \end{cases}$$

Por simplicidad, definimos $w_0 = -\theta$ y $x_0 = 1$, escribiendo entonces a z de la forma $z = -\theta + w_1x_1 + \cdots + w_mx_m = \mathbf{w^T x}$. La figura 2.2 ilustra cómo la entrada de la red $z = w^Tx$ es *aplanada* a una salida binaria

(-1 o 1) por la función de activación del perceptrón (izquierda) y cómo puede ser usada para discriminar entre dos clases linealmente separables (derecha):

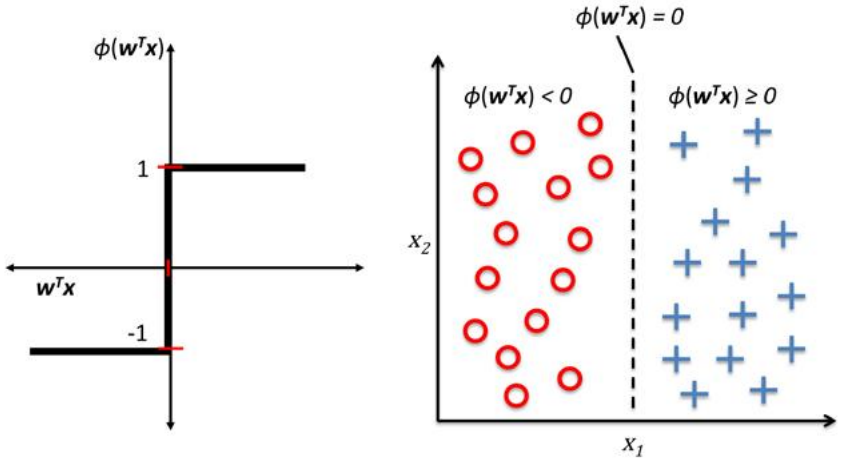

Figura 2.2: *Aplanamiento* de la salida del perceptrón para hacer clasificación. (Tomado de [16])

La idea detrás del modelo de perceptron de Rosenblatt es reducir a una abstracción de cómo funciona una neurona: se activa o no se activa. Así, la regla inicial de Roseblatt es relativamente simple y puede ser resumida en los siguientes pasos:

1. Inicializar los pesos en cero o en números aleatorios cercanos a cero.

2. Para cada muestra de entrenamiento $x^{(i)}$ realizar los siguientes pasos:

   a) Calcular el valor de salida $\hat{y}$.

   b) Actualizar los pesos en w.

En este caso, el valor de salida es la clasificación dada por la función escalón definida previamente (*i.e.* $\hat{y} = \phi(z)$) y la actualización simultánea de cada peso w_j en el vector de pesos w puede ser escrito más formalmente como:

$$w_j := w_j + \Delta w_j \tag{2.1}$$

El valor de Δw_j, que es usado para actualizar el peso w_j, es calculado por la regla de aprendizaje del perceptrón:

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \tag{2.2}$$

Donde η es el índice de aprendizaje (una constante entre 0 y 1), $y^{(i)}$ es la clasificación real de la i-ésima muestra, y $\hat{y}^{(i)}$ es la clasificación dada por la predicción. Es importante recalcar que todos los pesos en el vector de pesos son actualizados de manera simultánea, lo que significa que no recalculamos $\hat{y}^{(i)}$ hasta que todos los pesos Δw_j han sido actualizados.

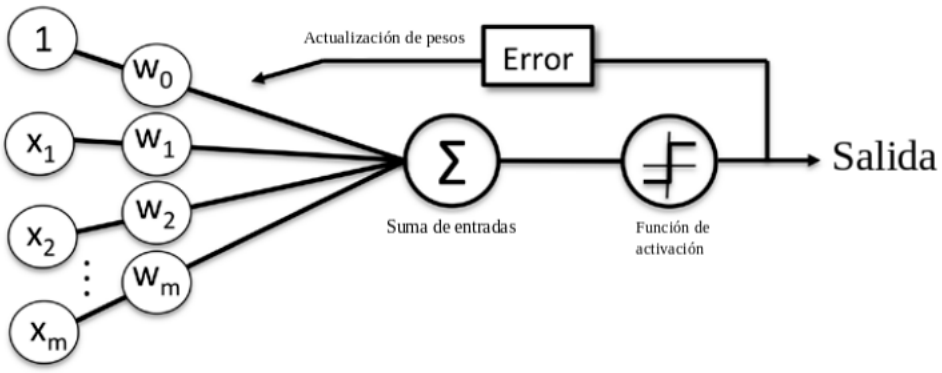

Figura 2.3: Resumen del concepto general de perceptrón. (Tomado de [16])

La figura 2.3 muestra cómo el perceptrón recibe las entradas de una muestra x y las combina con los pesos w para calcular la entrada neta. Esta entrada se le da a la función de activación, que genera una salida binaria (-1 o 1 en este caso), representando la predicción de la clasificación. Durante la fase de aprendizaje, la salida es usada para calcular el error de la predicción y actualizar los pesos.

2.3. Neuronas adaptativas lineales

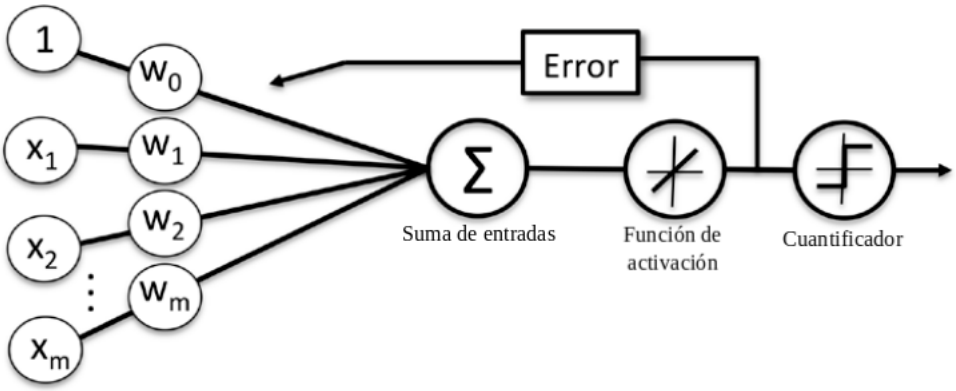

Figura 2.4: Esquema de Adaline. (Tomado de [16])

La neurona adaptativa lineal (**Adaline**) fue publicada por Bernard Widrow [9] unos años después del algoritmo del perceptrón de Frank Rosenblatt, y se le considera su evolución natural.

El algoritmo de Adaline es particularmente interesante porque ilustra el concepto clave de definir y minimizar funciones de costo, lo que sienta las bases para algoritmos más avanzados de clasificación, como la regresión logística o las máquinas de vectores de apoyo.

La principal diferencia entre las reglas de Adaline (también llamada de *Widrow-Hoff*) y la del perceptrón de Rosenblatt es que los pesos son actualizados con base en una función de activación lineal, en lugar de una función escalón. En Adaline, esta función de activación $\phi(z)$ es simplemente la función identidad de la entrada neta, así $\phi(w^T x) = w^T x$.

Mientras que la función de activación lineal es usada para la actualización de pesos, un *cuantificador*, similar a la función escalón descrita anteriormente, puede ser usado para hacer la clasificación, como se ilustra en la figura 2.4.

Si se comparan las figuras 2.3 y 2.4, la diferencia es que se usa la salida con valores continuos de la función lineal de activación para calcular el error y actualizar los pesos, en lugar de hacer una clasificación binaria.

Cabe destacar que actualmente se buscan funciones de activación con curvas más suavizadas y, sobre todo, diferenciables. Esto con el fin de optimizar los parámetros de los modelos más complejos.

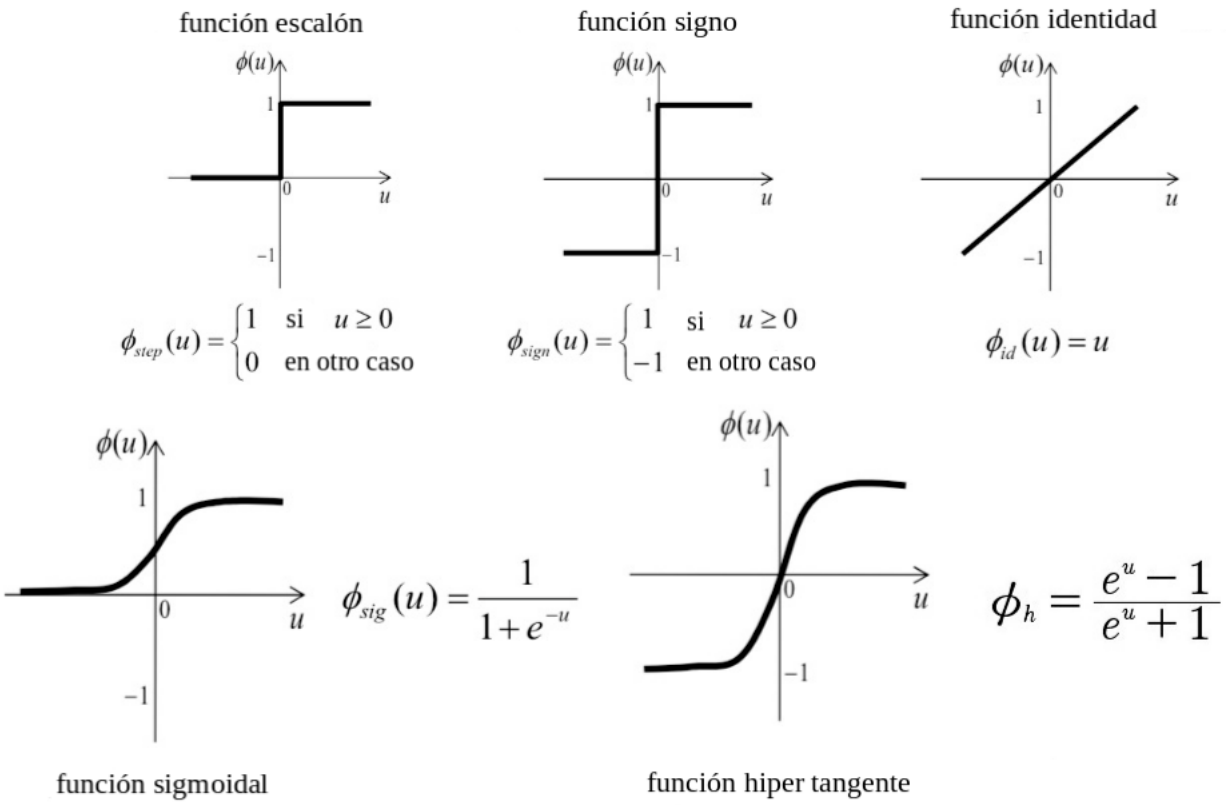

Figura 2.5: Algunos ejemplos de otras funciones de activación. (Tomado de `https://www.slideshare.net/SungJuKim2/multi-layer-perceptron-back-propagation`)

2.4. El perceptrón multicapa

El perceptrón multicapa es una generalización del perceptrón simple de Rosenblatt, como consecuencia de las limitaciones de éste ante con-

juntos de datos que no son linealmente separables. Lo que se hace es combinar varios perceptrones en una *red neuronal de propagación hacia adelante*, con una estructura de *capas*.

La información fluye de capa en capa en el mismo sentido, desde la *capa de entrada* donde están los datos sin procesar, hasta la *capa de salida* donde se da la clasificación. Cada una de las capas intermedias, llamadas *capas ocultas*, consta de un conjunto de neuronas sin conexiones entre ellas, pero totalmente conectadas a las capas inmediatamente anterior y posterior. Visto como una gráfica, un **perceptrón multicapa** (MLP, *Multilayer Perceptron*) es una gráfica n-partita dirigida donde n sería el número de capas.

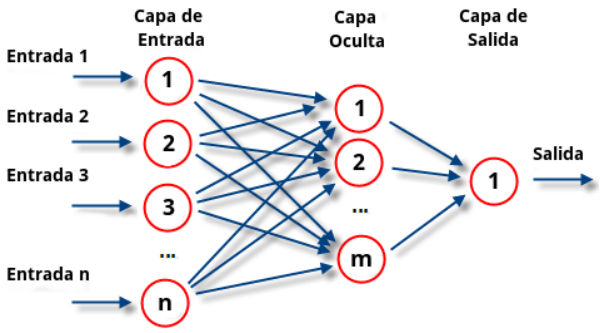

Figura 2.6: Arquitectura de un MLP con una capa oculta (Tomado de `https://es.wikipedia.org/wiki/Perceptron_multicapa`)

Cada neurona oculta actua como un *detector de características*. Conforme va avanzando el proceso de aprendizaje, las neuronas ocultas comienzan a "descubrir" gradualmente las características más sobresalientes de los datos de entrenamiento. Esto se logra a través de una serie de transformaciones no lineales, dadas por las funciones de activación y pesos específicos de cada neurona.

Inicialmente, a cada neurona se le asigna un peso aleatorio. Dada la estructura en capas de la red, es mucho más fácil visualizar y manipular estos pesos acomodandolos en una matriz W, que llamaremos *matriz de pesos*, donde W_i son los pesos de la i-ésima capa de la red. Entonces, dado un vector de entrada x, la primera capa oculta h_1 calcula su valor a partir de W_1 y una traslación b_1, que llamamos *sesgo* (*bias*, inglés), del siguiente modo:

$$h_1 = \phi(W_1 x + b_1) \tag{2.3}$$

donde ϕ es una función de activación no lineal y diferenciable. Así la $i+1$-ésima capa oculta computa su valor como

$$h_{i+1} = \phi(W_{i+1}h_i + b_{i+1}) \tag{2.4}$$

Por último, la salida, suponiendo que hay n capas ocultas, sería

$$\hat{y} = \phi(W_{n+1}h_n + b_{n+1}) \tag{2.5}$$

donde $\hat{y}$ es el vector de clasificación.

2.5. Descenso por el gradiente

Uno de los puntos clave del aprendizaje de máquina supervisado es definir una función objetivo que deberá ser optimizada durante el proceso de aprendizaje. Esta función objetivo usualmente es una **función de costo, pérdida o error** que se quiere minimizar.

Definimos la función de costo J como la **Suma de los errores cuadrados** (SSE, *Squared Sum Error*) entre la salida calculada y el valor real.

$$J(w) = 1/2n \sum_i (\hat{y}_i - y_i^2) \tag{2.6}$$

donde w es el conjunto de los parámteros de la red, es decir $w := \bigcup W_i, b_{i i=1}^n$, n el número de capas de la red, y y_i la i-ésima etiqueta de entrenamiento. La principal ventaja de esta función de error, además de ser lineal es que la función de costos se vuelve diferenciable.

La no-linealidad de las funciones de activación provoca que no se garantice la convexidad de las funciones de error más comunes, es decir, que no existe un método analítico para encontrar el mínimo de la función de error. El entrenamiento de la red se basa en métodos iterativos que van reduciendo paulatinamente ese error.

Sabemos que la derivada es útil para minimizar funciones porque, dada una función $y = f(x)$, nos dice cómo cambiar x para hacer una pequeña mejora en y, en otras palabras $f(x - \varepsilon f'(x)) < f(x)$ para un $\varepsilon \in \mathbb{R}$ suficientemente pequeño. Podemos entonces minimizar $f(x)$ al mover a x en pequeños *pasos* con el signo opuesto de la derivada. A esta técnica se le conoce como **descenso por el gradiente**.

La idea detrás del descenso por el gradiente es disminuir el gradiente hasta encontrar un mínimo (local o global) de la función de costo. En

cada iteración, se reduce un *paso* del gradiente, donde cada *paso* está determinado por el valor del índice de aprendizaje, así como por la pendiente del gradiente.

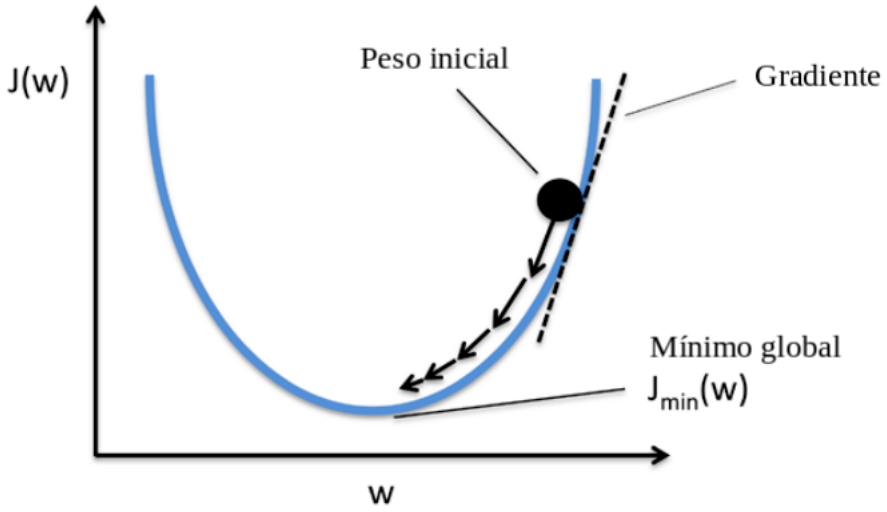

Figura 2.7: Diagrama de descenso por el gradiente. (Tomado de [16])

Usando el descenso por el gradiente, se pueden actualizar los pesos quitandole un *paso* al gradiente $\nabla J(w)$ de la función de costos $J(w)$:

$$w := w + \Delta w \tag{2.7}$$

En donde el cambio de peso Δw se define como el gradiente negativo multiplicado por el índice de aprendizaje η:

$$\Delta w := -\eta \Delta J(w) \tag{2.8}$$

Calculamos la derivada parcial de la función de costos SSE con respecto al j-ésimo peso de la siguiente manera:

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\ &= \frac{1}{2}\sum_i 2(y^{(i)} - \phi(z^{(i)}))\frac{\partial}{\partial w_j}(y^{(i)} - \phi(z^{(i)})) \\ &= \sum_i (y^{(i)} - \phi(z^{(i)}))\frac{\partial}{\partial w_j}(y^{(i)} - \phi(z^{(i)})) \\ &= \sum_i (y^{(i)} - \phi(z^{(i)}))(-x_j^{(i)}) \\ &= -\sum_i (y^{(i)} - \phi(z^{(i)}))x_j^{(i)}\end{aligned}$$

2.6. Descenso estocástico por el gradiente

Si se considera el caso en el que se tiene un conjunto de datos con millones de puntos con datos, correr un entrenamiento con descenso por el gradiente puede ser un proceso sumamente costoso computacionalmente ya que se requiere reevaluar todo el conjunto de datos cada vez que se toma un *paso* hacia el mínimo global.

Una alternativa popular al algoritmo de descenso por el gradiente es el **descenso estocástico por el gradiente**, llamado también descenso por el gradiente iterativo. En lugar de actualizar los pesos basado en la suma de los errores acumulados de todas las muestras $x^{(i)}$:

$$\Delta w = \eta \sum_i (y^{(i)} - \phi(z^{(i)}))x^{(i)} \tag{2.9}$$

se actualizan los datos de manera incremental para cada muestra del entrenamiento:

$$\Delta w = \eta (y^{(i)} - \phi(z^{(i)}))x^{(i)} \tag{2.10}$$

Aunque el descenso estocástico por el gradiente podría ser considerado una aproximación del descenso por el gradiente, por lo general converge mucho más rápido debido a las actualizaciones tan frecuentes de los pesos. Como cada gradiente se calcula basado en un sólo ejemplo de entrenamiento, el error tiende a tener más entropía, lo que permite que el

descenso estocástico por el gradiente pueda *escapar* más fácilmente de un mínimo local. Para obtener buenos resultados con el descenso estocástico por el gradiente es importante que se tomen los datos de forma aleatoria.

Otra ventaja del descenso estocástico por el gradiente es que se puede usar para hacer *aprendizaje en línea*. Esto quiere decir que el modelo es entrenado al momento, mientras más y más datos van llegando. Esto es especialmente útil cuando se están acumulando grandes cantidades de datos. Usando entrenamiento en línea, el sistema puede adaptarse inmediatamente a los cambios y los datos de entrenamiento pueden ser descartados después de actualizar el modelo, si el espacio de almacenamiento fuera un problema.

Las redes neuronales son modelos complejos que por lo general tienen un alto número de mínimos locales, volviendo al descenso por el gradiente poco efectivo como mecanismo de entrenamiento, incluso utilizando la variante estocástica. Sin embargo, redes altamente eficientes usualmente son entrenadas utilizando descenso por el gradiente. ¿Cómo explicar esta aparente contradicción?

Recientemente, evidencia empírica [6] ha dado indicios de que las ANN tienen la mayoría de sus mínimos locales cerca del mínimo global. Esto quiere decir que **el descenso por el gradiente estocástico funciona "suficientemente bien"**, ya que puede *escapar* de los mínimos locales sin alejarse demasiado del mínimo global. Hay otros métodos más avanzados que son empleados para entrenar ANNs, pero en este trabajo utilizamos descenso por el gradiente por su simplicidad e interpetación tan intuitiva.

2.7. Retropropagación

Para utilizar de forma efectiva el descenso por el gradiente, es necesario una forma eficiente de computar el gradiente de una función de error. La propagación hacia atrás o retropropagación (*Backpropagation*) nos permite hacerlo.

Definimos para cada neurona j en la capa l la salida $o_j^{(l)}$ como

$$o_j^{(l)} = \phi(a_j^{(l)}) = \phi(\sum_{k=1}^{n} w_{kj} o_k^{(l-1)}) \tag{2.11}$$

donde ϕ es la función de activación y $a_j^{(l)}$ se refiere a la suma de los pesos de las salidas de las neuronas de la capa inmediata anterior. Comunmente, llamamos a $a_j^{(l)}$ la *activación* de la neurona.

Supongamos ahora que queremos encontrar la derivada parcial de la función de errir con respecto a w_{ij}. Para encontrar cómo cambia una función de error J cambia con respecto al peso w_{ij}, aplicamos la regla de la cadena:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial o_j^{(l)}} \frac{\partial o_j^{(l)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ij}} \tag{2.12}$$

A partir de 2.11, podemos ver que $a_j^{(l)} = \sum_{k=1}^{n} w_{kj} o_k^{(l-1)}$. Esto implica que $\frac{\partial a_j^{(l)}}{\partial w_{ij}} = o_i^{(l-1)}$. Definimos entonces

$$\delta_j^{(l)} = \frac{\partial J}{\partial o_j^{(l)}} \frac{\partial o_j^{(l)}}{a_j^{(l)}} \tag{2.13}$$

Así, tenemos que

$$\frac{\partial J}{\partial w_{ij}} = \delta_j^{(l)} o_i^{(l-1)} \tag{2.14}$$

Resulta entonces que tenemos una identidad recursiva para $\delta_j^{(l)}$, llamada la fórmula de la retropropagación

$$\delta_j^{(l)} = \phi'(a_j^{(l)}) \sum_m w_{jm} d_m^{(l+1)} \tag{2.15}$$

donde la suma es sobre las neuronas conectadas a la j-ésima neurona de la capa l.

Podemos entonces derivar esta identidad utilizando la regla de la cadena para escribir $\frac{\partial J}{\partial o_j^{(l)}}$ en términos de $\frac{\partial J}{\partial a_m^{(l+1)}}$ y $\frac{\partial a_m^{(l+1)}}{\partial o_j^{(l)}}$.

$$\begin{aligned}
\delta_j^{(l)} &= \frac{\partial J}{\partial o_j^{(l)}} \frac{\partial o_j^{(l)}}{\partial a_j^{(l)}} \\
&= \frac{\partial o_j^{(l)}}{\partial a_j^{(l)}} \frac{\partial J}{\partial o_j^{(l)}} \\
&= \phi'(a_j^{(l)}) \sum_m \frac{\partial J}{\partial o_m^{(l+1)}} \frac{\partial o_m^{(l+1)}}{\partial a_m^{(l+1)}} \frac{\partial a_m^{(l+1)}}{\partial o_j^{(l)}} \\
&= \phi'(a_j^{(l)}) \sum_m \delta_m^{(l+1)} w_{jm}
\end{aligned} \tag{2.16}$$

Notemos que esto nos permite computar capas anteriores a δ_j a través de las capas posteriores, de ahí el nombre de retropropagación. Podemos computar δ_j directamente, si j es una capa de salida, así que este proceso eventualmente termina.

CAPÍTULO 3

Sobre proteínas

Las proteínas son macromoléculas formadas por cadenas lineales de aminoácidos, cada una de las cuales se denomina polipéptido. El término **proteína** se origina del griego *proteios* que significa "primario" o "de primer orden". El nombre fue adoptado por Jöns Berzelius en 1838 para enfatizar la importancia de esta clase de moléculas. Las proteínas juegan un rol crucial en el mantenimiento de la vida, ya que son el soporte para la arquitectura del tejido muscular, ligamentos, tendones, huesos, piel, cabello, órganos y glándulas. Proveen los servicios fundamentales de transporte y almacenamiento como en el caso del oxígeno y hierro en células musculares y eritrocitos. También participan en muchos procesos regulatorios esenciales, como reacciones de catálisis, funciones inmunológicas y hormonales, así como en la coordinación de actividades neuronales y diferenciación celular.[19]

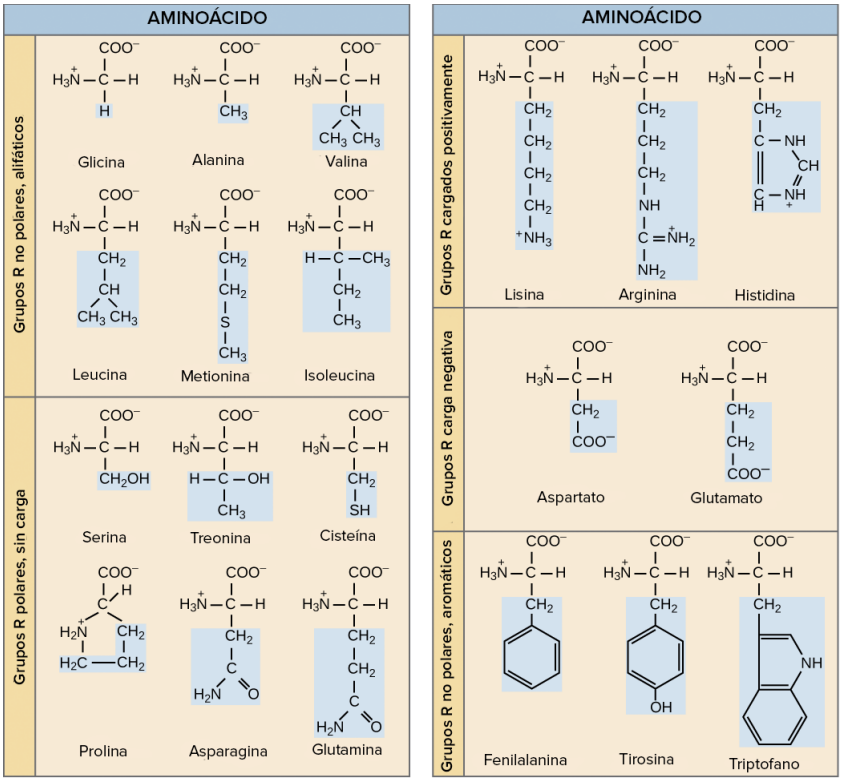

Figura 3.1: Existen sólo 20 tipos de aminoácidos en las proteínas (Tomado de `https://cnx.org/contents/GFy_h8cu@11.9:2zzm1QG9`)

3.1. Ligandos

La comunicación entre moléculas es dada por medio de señales químicas. Estas señales químicas son mandadas en forma de moléculas pequeñas, usualmente volátiles o solubles, llamadas ligandos. Un **ligando** es una molécula que se une a otra molécula específica, en algunos casos, mandando una señal en el proceso. Los ligandos interactuan con proteínas en células objetivos. Son a estas proteínas a las que llamamos **receptores**.

Las interacciones de una proteína con otras moléculas, como ligandos, ácidos nucléicos u otras proteínas, son críticas para su función bioquímica. Usualmente, no todos los residuos en la superficie de una proteína

participan en estas interacciones; las interacciones ocurren en áreas definidas, que llamamos **sitios de activación o activos** de las proteínas.

Las interacciones proteína-ligando y proteína-proteína juegan un papel fundamental en el descubrimiento de fármacos: el objetivo de la creación de fármacos es encontrar un agente que pueda interactuar con una molécula objetivo y modificar su actividad. Estas moléculas objetivo son usualmente proteínas, encargadas de la mayoría de las tareas necesarias para mantener a las células vivas. Por otro lado, los agentes son ligandos que interactuan con el sitio activo de la proteína y pueden inhibirla o reactivarla, de ahí que a este tipo de ligandos se le llamen **inhibidores**. Es por esto que se han desarrollado una variedad de metodologías para investigar dichas interacciones. En este trabajo nos enfocamos en el acoplamiento molecular.

3.2. Acoplamiento molecular

El campo del **acoplamiento molecular** o **docking** surge a lo largo de las últimas tres décadas gracias a la necesidad de la biología molecular en cuanto al descubrimiento de inhibidores basado en estructuras. Este campo ha evolucionado considerablemente a lo largo de los últimos años gracias al crecimiento dramático en la disponibilidad y poder de las computadoras, además del incremento en las bases de datos de proteínas y moléculas, y la facilidad de acceso a ellas.[10]

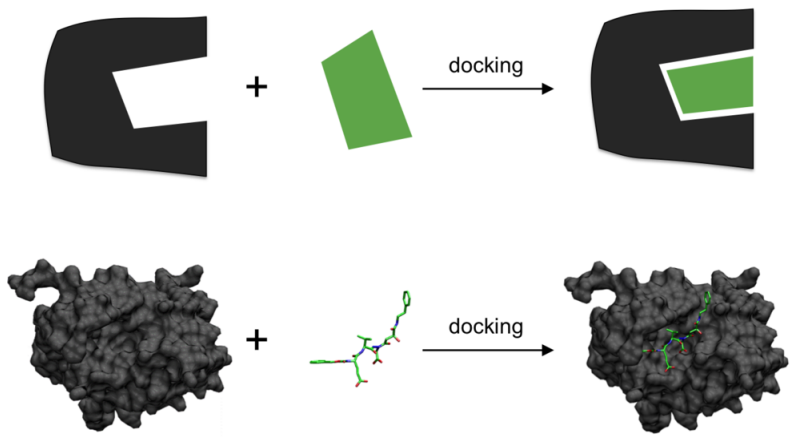

Figura 3.2: Representación esquemática del *docking*. (Tomado de `https://en.wikipedia.org/wiki/Docking_(molecular)`)

El objetivo de un programa de acoplamiento molecular automatizado es comprender y predecir reconocimiento molecular, tanto estructuralmente, encontrando posibles *poses* de acoplamiento, como energéticamente, prediciendo la afinidad del enlace. El acoplamiento molecular usualmente se realiza entre una molécula pequeña, un ligando, y una macromolécula objetivo, una proteína en nuestro caso. Es importante que este acoplamiento se haga en el sitio activo del receptor, ya que es ahí donde se espera que se dé la interacción.

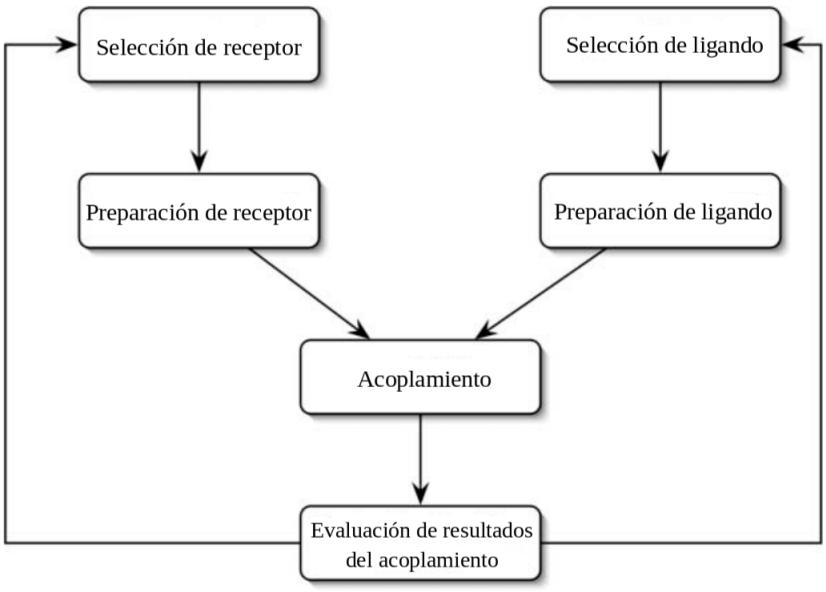

Figura 3.3: Diagrama de flujo para un acoplamiento usual. (Tomado de [10])

La figura 3.3 muestra los pasos clave que son comunes en todos los protocolos. El acoplamiento consiste en encontrar las poses de unión más favorables de un ligando hacia una proteína objetivo. La pose de unión de un ligando puede ser caracterizado de forma única por sus variables de estado. Éstas consisten en su posición (traslaciones sobre los ejes x, y, z), orientación (ángulos de Euler o cuaterniones) y, si el ligando es flexible, su conformación (los ángulos de torsión para cada enlace de rotación). Cada una de las variables de estado describe un grado de libertad en un espacio de búsqueda multidimensional.

3.3. Función evaluadora

Todos los métodos de acoplamiento requieren una función de evaluación para calificar las poses de unión de los candidatos, y un método de búsqueda para explorar las configuraciones de las variables de estado. En

general, el éxito de un acoplamiento se mide en términos de la **desviación media cuadrática** (RMSD, *Root-mean-square deviation*) de las coordenadas cartesianas de los átomos del ligando en las conformaciones del acoplamiento, comparadas con las cristalográficas. Un acoplamiento se considera exitoso si el RMSD es menor a 2Å.

CAPÍTULO 4

Sobre meta-análisis del acoplamiento

Como se mencionaba en el capitulo anterior, cada acoplamiento es evaluado a través de una función evaluadora. El detalle es que esta fase de evaluación se ha convertido en todo un reto para los científicos computacionales por la dificultad de decidir de forma determinística si un acoplamiento es bueno o no. Esto se traduce en que la pose que se evalua como la "mejor" está realmente alejada de la pose cristalográfica, y la pose más cercana a la cristalográfica esta mucho más abajo en el ranking de poses "buenas" dada por esta evaluación.

En este trabajo se propone un acercamiento con una red neuronal convolucional para hacer un meta-análisis de dichos acoplamientos, comparando las poses cristalográficas con las dadas por el acoplamiento simulado.

4.1. Preparación de la base de datos

La base de datos de la cual partimos es la de el Banco de Datos de Proteínas (PDB, *Protein Data Bank*), que proporciona acceso a resultados de estudios estructurales de macromoléculas biológicas[1]. Este banco está compuesto poralrededor de 150,000 moléculas tridimensionales de-

[1] `https://www.rcsb.org/`

terminadas experimentalmente a las que se tiene libre acceso. El formato utilizado para almacenar esta información (.pdb) contiene elementos como coordenadas de átomos, nombres de moléculas e información sobre estructuras primarias y secundarias. Es con este formato con el que se trabajó durante el proyecto.

Inicialmente, se descargaron todas las proteínas alojadas en el banco y se fue haciendo un filtrado de ellas del siguiente modo:

1. Nos interesa estudiar acoplamientos, entonces se trabaja únicamente con proteínas que contuvieran ligandos. El resto queda descartado.

2. Se eliminan todas las proteínas que tuvieran un peso molecular menor a 300Da. Debajo de este valor se encuentran principalmente iones y moléculas de bufer de cristalización.

3. Se quitan de las proteínas las moléculas solventes como DMSO, PEG y MES, los cuales no son de relevancia biológica.

4. Se filtran de las proteínas las cadenas de ADN o ARN.

5. Se quitan también todos los metales pesados de las proteínas.

6. Finalmente, se transforma el archivo .pdb en un archivo .pdbqt. Esto quiere decir que se definen las cargas de la proteína así como sus libertades de torsión (a las que más adelante llamaremos *ramas*). Para esto se utilizó el programa MGLTools[2]

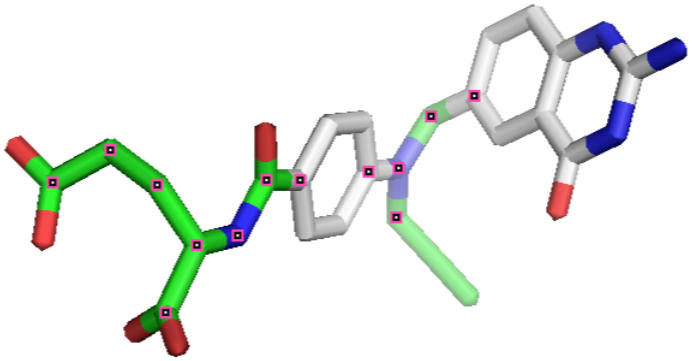

Figura 4.1: Ligando CB3 con sus ejes de torsión marcados en rosa.

[2] `http://mgltools.scripps.edu/`

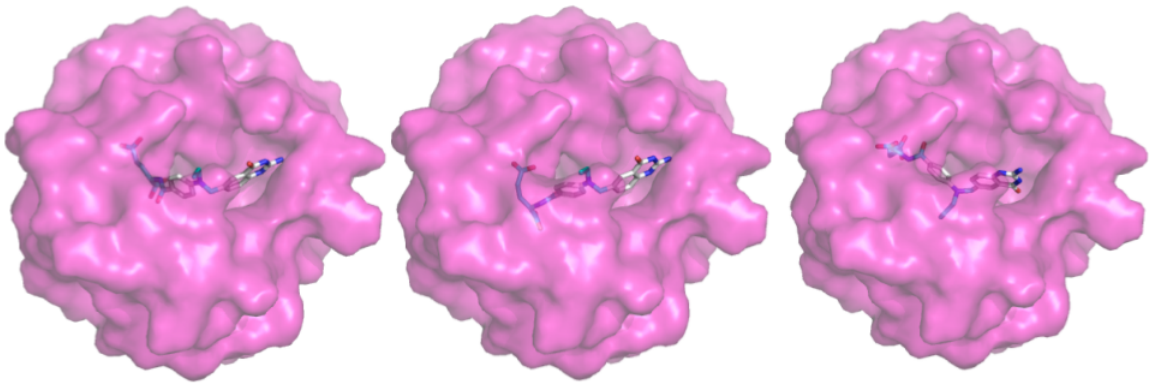

Figura 4.2: Primeras tres poses de acoplamiento del ligando CB3 con el receptor 4EIL.

Después, se hace la separación en cada proteína entre ligando y receptor, y para cada pareja se realiza un acoplamiento virtual utilizando **AutoDock Vina 1.1.2**[3], teniendo así una base de datos de 24,964 acoplamientos. Esto genera un listado de poses, donde cada pose es un archivo .pdbqt, junto con una calificación asociada, dada por la función evaluadora de AutoDock Vina, de qué tan "buena" es la pose dada. Con esto, se hace una tabulación de cada pose y su calificación contra el RMSD a la pose cristalográfica original. Esta tabla sirve como entrada para *Deep-pose*.

[3] `http://vina.scripps.edu/index.html`

Pose	Clasificación (según AutoDock Vina)	Calificación	RMSD
4EIL_CB3_A	1	-10.2	3.08
4EIL_CB3_A	2	-10.0	3.02
4EIL_CB3_A	3	-9.8	3.02
4EIL_CB3_A	4	-9.5	1.31
4EIL_CB3_A	5	-9.3	3.0

Cuadro 4.1: Extracto de la tabulación de poses contra RMSD a la información cristalográfica. Se puede apreciar que la mejor pose es la que Autodock Vina colocó en el lugar 4, siendo que es la que tiene el menor RMSD. El formato de nombre de las poses es {compuesto}_{ligando acoplado}_{cadena}.

4.2. Deep-pose

Deep-pose es una red neuronal convolucional profunda que toma la información de la pose de un acoplamiento en un complejo proteína-ligando como entrada y produce una calificación de qué tan viable es dicha pose. Primero, dada una entrada de un complejo proteína-ligando x, se extrae información del contexto local de cada rama. El *contexto* de una rama está dado por información estructural básica (*tipo de rama* y distancia). Después, cada una de estas características básicas de cada rama es convertida en vectores característicos que utiliza la red para aprender. Luego, una capa convolucional es empleada para sintetizar la información de todos los contextos de todas las ramas del ligando y generar una representación vectorial del complejo. Posteriormente se pasa a una capa oculta para síntesis y procesamiento del vector-ligando. Finalmente, en la última capa, la representación del complejo es dada como entrada a un clasificador *softmax*, que es responsable de producir el puntaje.

A continuación se presenta un pseudo-código de alto nivel del proceso de la red:

Algorithm 1 Deep-pose

Entrada: complejo proteína-ligando x, donde el ligando tiene n *ramas*
Dados:
$W^{b\_type} \in \mathbb{R}^{N\times|B|}, W^{b\_dist} \in \mathbb{R}^{N\times|B|}$,
$W^{conv} \in \mathbb{R}^{|z_i|\times cf}, W^3 \in \mathbb{R}^{cf\times h}$,
$W_{out} \in \mathbb{R}^{h\times 2}$,
$b^{conv} \in \mathbb{R}^{cf}, b^3 \in \mathbb{R}^h$,
$b^{out} \in \mathbb{R}^2$
$Z \leftarrow [..]$
for $i \leftarrow 1, n$ **do**
    $z_{b\_type} \leftarrow$ columnas de W_{b_type} correspondientes a los tipos de ramas de los vecinos de la rama i
    $z_{b\_dist} \leftarrow$ columnas de W_{b_dist} correspondientes a las distancias de los vecinos de la rama i
    $z_i \leftarrow \{z_{b\_type}, z_{b\_dist}\}$
    $Z.add(z_i)$
end for
// U es inicializada con ceros
$U \leftarrow [..] \in \mathbb{R}^{cf\times n}$
// Capa convolucional
for $i \leftarrow 1, n$ **do**
    $U[:, i] \leftarrow f(W^{conv} Z[i] + b^{conv})$
end for
// *max-pooling* por columnas
$r \leftarrow$ máx$(U, axis = 1)$
//Capas oculta y de salida
$score \leftarrow W^{out}(W^3 r + b^3) + b^{out}$
// Regresa el puntaje normalizado
return $\frac{e^{score[1]}}{e^{score[0]}+e^{score[1]}}$

Veamos a detalle cada una de las partes de la red.

4.2.1. Contexto de la rama

Pereira, Caffaren y dos Santos [15] consideran al átomo como una entidad ligada íntimamente a su contexto. Bajo esta premisa, crean una

red que toma como entrada a cada átomo del ligando con su contexto codificado, entendiendo contexto del átomo como las carácteristicas de éste y de los átomos más cercanos. Pereira, Caffaren y dos Santos buscaban encontrar valores de acoplamiento que correspondieran con la eventual medición de la energía de enlace, sin importar cual fuera la conformación necesaria para poder realizarlo. Nuestro enfoque busca dar preferencia a la identificación de poses que correspondan con una eventual pose cristalográfica.

Partiendo de la idea del átomo ligado a su contexto, y considerando que lo que buscamos es una propiedad puramente estructural, tomamos como unidad básica del ligando a los segmentos con libertad rotacional, a la que llamaremos *rama*. Así, nuestro *contexto de átomo* se convierte en **contexto de rama**, siendo éste la combinación del tipo de rama, los de las ramas más cercanas y la distancia a cada una de ellas.

4.2.2. Codificación del contexto de la rama

SMILES (*Simple Molecular Input Line Entry System*) es un sencillo lenguaje químico que permite describir moléculas y reacciones utilizando únicamente caracteres ASCII[4] que representan símbolos de átomos y enlaces. Una cadena SMILES contiene la misma información que una tabla de conexiones extendida, pero con varias ventajas: es sumamente compacta y puede ser canonizada de tal manera que puede ser usada como identificador universal para una estructura química dada.[5]

Se divide cada receptor y ligando en sus respectivas ramas y cada una de estas ramas se codifica usando la respresentación SMILES. Esta codificación, representa de forma única a cada rama distinta; es a esto a lo que llamamos **tipo de rama**. Se enlistan todos los tipos de rama, asociando a cada uno un índice, generando así lo que llamamos el *diccionario de ramas*.

Del mismo modo, se segmentan los rangos de distancia encontrados en compartimentos, y a cada uno de estos se les asigna también un índice, generando así un *diccionario de distancias*.

[4]El American Standard Code for Information Interchange es un estándar de codificación de caracteres para la comunicación informática.

[5]`http://www.daylight.com/smiles/`

SMILES	Idx	Rango de distancia (Å)	Idx
NC1=N[C](=NC=C1)=O	93	3.0526 - 3.2631	6
C1CCCCC1	94	3.2632 - 3.4736	7
CNC=O	95	3.4737 - 3.6842	8
NC=N	96	3.6843 - 3.8947	9
CC=C	97	3.8948 - 4.1052	10

Cuadro 4.2: Fragmento de los diccionarios de ramas y de distancias

A partir de estos diccionarios, se asocia a una rama del ligando con las cinco ramas del receptor más cercanas codificadas a través de sus tipos y sus distancias a las ramas dadas. Lo que se genera entonces es un vector con dos tuplas, donde cada elemento de las tuplas son índices de tipos de rama y distancia respectivamente; a este vector le llamamos el *vector de rama*. El conjunto de vectores de rama de un ligando genera la *matriz de ligando*, que será la entrada de la red.

OP(O)O	Å
N	5.794664
C1CC1	5.691862
NC1=N[C](=NC=C1)=O	4.449922
NC=N	3.785496
O	3.747894

$$\downarrow$$

$$OP(O)O = \begin{bmatrix}(2, 13, 93, 96, 4) & (4, 2, 2, 6, 4)\end{bmatrix}$$

Cuadro 4.3: Traducción de una rama a su representación vectorial

Tanto para los tipos de rama, como para las particiones de distancia, se generan las matrices W^{b_type} y W^{b_dist} respectivamente. Estas matrices constituyen los pesos de la primera capa de la red y son inicializadas con valores aleatorios.

4.2.3. Representación vectorial del contexto de rama

La primera capa de la red toma cada matriz de ligando y la transforma en una matriz en $\mathbb{R}$. Cada columna en $W^{b\_type} \in \mathbb{R}^{N \times |B|}$ corresponde al vector característico de un tipos de rama, donde B es el conjunto de tipo de ramas y N es la dimensión de los vectores característicos, quedando como un hiperparámetro a definir. Dado el contexto de una rama, la red transforma cada tipo de rama en su respectivo vector característico, utilizando los índices ya generados, y luego concatena los vectores para generar la representación vectorial del tipo de rama z_{b_type}. Analogamente, se genera el vector z_{b_dist} en el contexto de la rama objetivo.

$$Rama = \left[OP = O, \;\; OC = O, \;\; C = O, \;\; OPO, \;\; OP = O\right]$$

$W_{b\_type} =$

OP=O	OC=O	OPO	C=O	NC=O	OPO	OP=O

$\downarrow$

$z^T_{b\_type} =$ OP=O $\bullet$ OC=O $\bullet$ C=O $\bullet$ OPO

Cuadro 4.4: Representación de la construcción del tipo de rama (z_{b_type}). El símbolo $\bullet$ representa la operación de concatenación.

Finalmente, la representación del contexto de la rama b se define como $z_b = z_{b\_type} \cdot z_{b\_dist}$. El objetivo es que, a partir de características básicas contextuales, la red pueda aprender a distinguir rasgos abtractos de las ramas que permitan la discriminación entre poses válidas y señuelos.

4.2.4. Representación de la pose de un complejo proteína-ligando

La segunda capa en la red es una capa convolucional encargada de extraer más características abstractas de las representaciones de los con-

textos de ramas y sintetizar la información de todas ellas en un vector r de longitud fija.

El problema esencial que resuelve el uso de una capa convolucional es la capacidad de manejar entradas de distintas dimensiones. Dado que la cantidad de ramas por ligando es variable, el uso de una capa convolucional posibilita el procesamiento de complejos de distintos tamaños.

Dado un complejo x conformado por n ramas, la entrada de la capa convolucional es una lista de vectores $\{z_1, z_2, ..., z_n\}$ donde z_i es la representación vectorial del contexto de la i-ésima rama del ligando. En la primera etapa de la capa, la extracción de características más abstractas de cada vector z_i está dada por

$$u_i = f(W_{z_i}^{conv} + b^{conv}) \tag{4.1}$$

donde $W^{conv} \in \mathbb{R}^{cf \times h_1}$ es la matriz de pesos correspondiente a la capa convolucional, b^{conv} es el sesgo, f es la función tangente hiperbólica y $u_i \in \mathbb{R}^{cf}$. El número de unidades o filtros (cf) en la capa convolucional es un hiperparámetro definido por el usuario.

La segunda etapa en la capa convolucional correspondiente a la etapa de agrupación (*pooling* en inglés) se encarga de sintetizar las características de los contextos de rama. La entrada consiste en un conjunto de vectores $\{u_1, u_2, ..., u_n\}$. Utilizamos una capa de *max-pooling* que produce un vector $r \in \mathbb{R}^{cf}$, donde el valor del j-ésimo elemento está definido como el máximo de los j-ésimos elementos del conjunto de vectores de entrada:

$$[r]_j = \max_{1 \leq i \leq n} [u_i]_j \tag{4.2}$$

El vector resultante r de esta etapa es la representación de la pose del complejo proteína-ligando. De este modo, la red puede aprender a generar una representación vectorial que sintetice la información del complejo que sea relevante para discriminar poses válidas de señuelos.

4.2.5. Calificación de la pose

El vector r es procesado por dos capas neuronales básicas más: una tercera capa oculta que representa un nivel más de abstracción y una cuarta y última capa de salida, que computa una calificación para cada una de las posibles clasificaciones de la pose: (0) pose señuelo y (1) pose

válida. Formalmente, dada la representación r de la pose del complejo x, la capa oculta y la de salida se computan de la siguiente manera:

$$s(x) = W^{out}(w^3 r + b^3) + b^{out} \tag{4.3}$$

donde $W^3 \in \mathbb{R}^{hxcf}$ es la matriz de pesos de la tercera capa oculta, $W^{out} \in \mathbb{R}^{2xh}$ es la matriz de pesos de la capa de salida, y $b^3 \in \mathbb{R}^h$ y $b^{out} \in \mathbb{R}^2$ son sesgos. El número de unidades en la capa oculta, h, es un hiperparámetro que está definido por el usuario. $s(x) \in \mathbb{R}^2$ es un vector que contiene las calificaciones de cada una de las dos clases.

Sean $s(x)_0$ y $s(x)_1$ las calificaciones de las clases 0 y 1, respectivamente, transformamos estas calificaciones en una distribución de probabilidad utilizando la función *softmax*:

$$p(0|x) = \frac{e^{s(x)_0}}{e^{s(x)_0} + e^{s(x)_1}} \tag{4.4}$$

$$p(1|x) = \frac{e^{s(x)_1}}{e^{s(x)_0} + e^{s(x)_1}} \tag{4.5}$$

donde decimos que $p(0|x)$ y $p(1|x)$ es la probabilidad condicional de que la pose del compuesto sea válida o sea señuelo, respectivamente, dados los datos obtenidos a partir del acoplamiento del complejo proteína-ligando.

CAPÍTULO 5

Entreamiento de la red y resultados

5.1. Entorno de trabajo

Para los cómputos de entrenamiento y predicción de la red se utilizó una tarjeta gráfica (GPU, *Graphics Processing Unit*) *Quadro K5000* de la marca *NVIDIA*[1]. Tanto *Deep-pose* como todas las herramientas utilizadas para el procesamiento de datos fueron creadas con el lenguaje de programación **Python** (versión 2.7) y con las bibliotecas **Numpy** (versión 1.14.0)[2] para el manejo de los datos y **Theano** (versión 0.9.0)[3] para la creación de la red.

5.2. Entrenamiento de la red

Deep-pose se entrenó utilizando un algoritmo de **descenso estocástico por el gradiente** (SGD, *Stochastic Gradient Descent*). En este caso, SGD se utiliza para minimizar la función de costos sobre un conjunto de entrenamiento D que contiene poses tanto válidas como señuelos. En

[1] `https://www.nvidia.com/en-us/design-visualization/quadro-desktop-gpus/`

[2] `https://www.numpy.org/`

[3] `http://deeplearning.net/software/theano/`

cada iteración, un nuevo complejo $(x, y) \in D$ es elegido al azar, donde $y = 1$ si la pose es válida y $y = 0$ en caso contrario. Después la red, junto con los parámetros $\theta = \{W^{b_{type}}, W^{b_{dist}}, W^{conv}, W^3, W^{out}, b^{conv}, b^3, b^{out}\}$ es utilizada para estimar la probabilidad $p(y|x, \theta)$. Finalmente, el error en la predicción se computa como la probabilidad logarítmica negativa, $-\log(p(y|x, \theta))$, y los parámetros de θ son actualizados utilizando retropropagación.

$$\theta \longmapsto \sum_{(x,y)\in D} -\log p(y|x, \theta) \tag{5.1}$$

En este trabajo, se utilizaron *minilotes* de complejos, y tomamos el promedio del error de la predicción para realizar la retropropagación.

La red se entrenó con un conjunto de datos de **24,957 poses** que se dividió en 19,965, 2,495 y 2,497 muestras para entrenamiento, pruebas y validación respectivamente. Éstas poses al dividirlas en ramas, resultaron en **5,216 distintas ramas**, que son las utilizadas para crear el diccionario de ramas.

Hiperparámetro	Descripción	Valor
N	Dimensión del vector característico	80
cf	Unidades en la capa convolucional	150
h	Unidades en la capa oculta	60
bs	*Tamaño de los minilotes*	20
λ	Índice de aprendizaje	0.1

Cuadro 5.1: Hiperparámetros utilizados para entrenar *Deep-pose*.

El ajuste de los hiperparámetros se hizo de manera *artesanal*, haciendo varias iteraciones de entrenamiento mientras se iban ajustando los hiperparámetros. El mejor conjunto de hiperparámetros que se encontró es el que se muestra en la tabla 5.1.

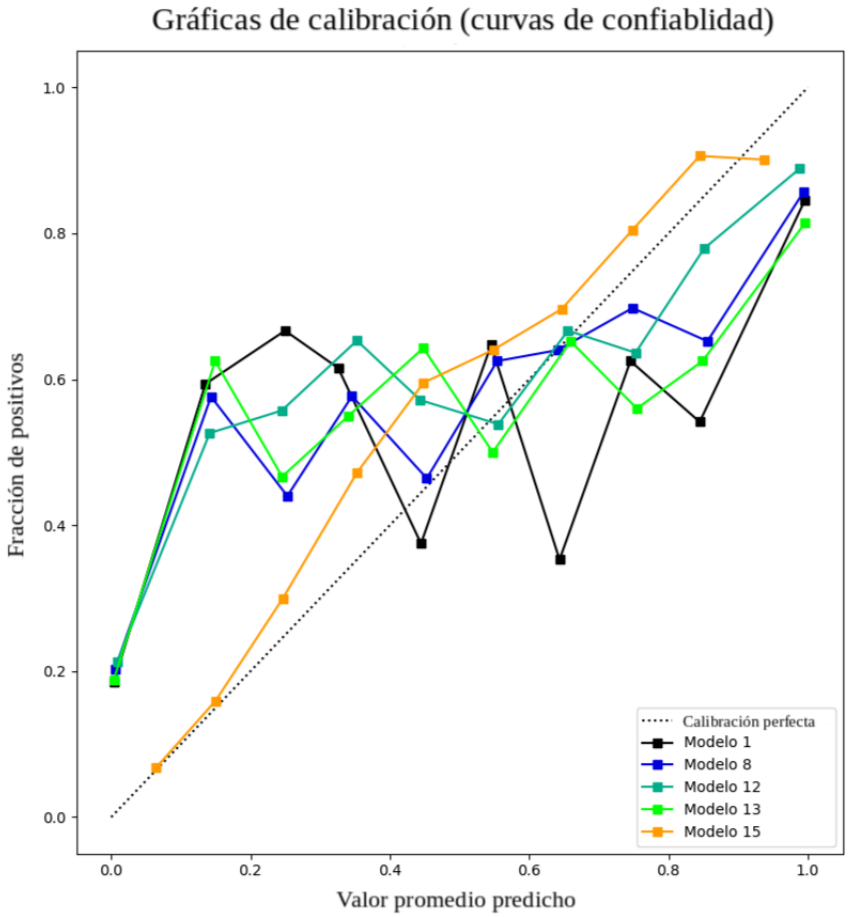

Figura 5.1: Calibracion de los hiperparámetros, siendo la iteración número 15 la mejor encontrada.

5.3. Resultados

Podemos ver en la figuras 5.2 y 5.3 que *Deep-pose* alcanza una precisión de **82 %** al momento de determinar si una pose es válida o no, lo cual muestra que es una opción efectiva para determinar la fiabilidad de un acoplamiento virtual.

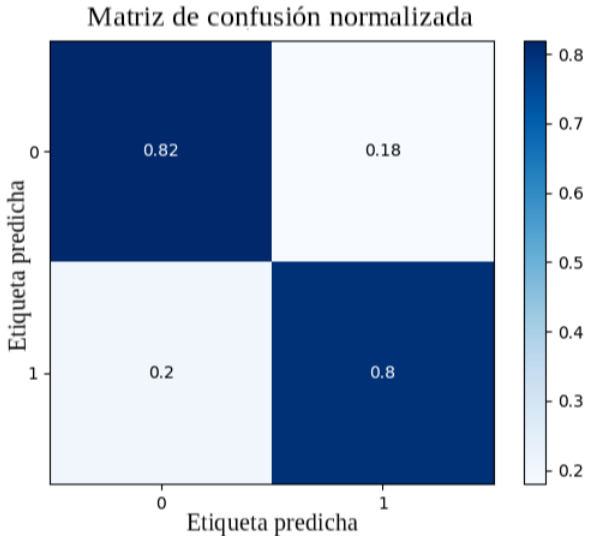

Figura 5.2: Matriz de confusión de las predicciones de la red, donde la etiqueta 1 representa una pose válida y 0 una inválida. Los puntajes son adquiridos a partir del conjunto de validación usado durante el entrenamiento.

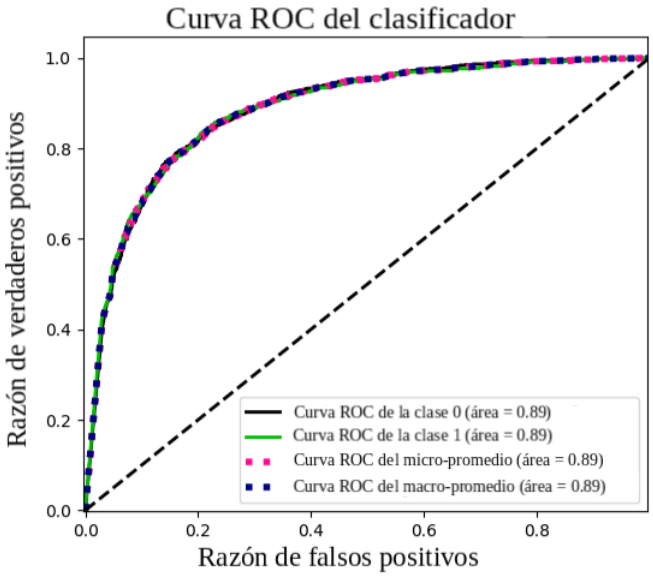

Figura 5.3: Curva característica operativa del receptor (ROC, *Receptive Operating Characteristic*) mostrando las razones de verdaderos y falsos positivos.

CAPÍTULO 6

Conclusiones

En este trabajo se diseña y se crea *Deep-pose*: una red de aprendizaje profundo que busca hacer ua evaluación de acoplamientos virtuales. *Deep-pose* está compuesta por una capa convolucional, una capa oculta y una de clasificación y toma como entrada la salida de acoplamientso virtuales generados por AutoDock Vina. Estos acoplamientos son traducidos en una codificación que transforma las poses de acoplamiento en vectores de enteros mediante asociaciones con tablas de búsqueda, y sirven como entrada a la red.

Esta arquitectura, permitó obtener un 82 % de precisión al momento de discernir entre poses cercanas a las cristalográficas. Este resultado, aunado a que (1) *Deep-pose* no requiere características definidas por un humano, a diferencia del proceso *artesanal* que se sigue usualmente para determinar cuáles acoplamientos son buenos, y que (2) alcanza buenos resultados a partir de la salida de un sólo programa de docking, hacen que *Deep-pose* sea atractivo para usarse como complemento al acoplamiento virtual usado normalmente. Además, este trabajo propone ideas novedosas sobre cómo modelar complejos proteínas-ligandos e introduce una propuesta que demuestra ser efectiva para codificar configuraciones moleculares, que podría ser usada para otros fines en alguna otra red de aprendizaje profundo aplicada a bioinformática.

Bibliografía

[1] Ethem Alpaydin. *Introduction to Machine Learning*. 2.ª ed. The MIT Press, 2010.

[2] Marcelino Arciniega y Oliver F. Lange. "Improvement of virtual screening results by docking data feature analysis". En: *Journal of chemical information and modeling* (mayo de 2014).

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] Albert Manuel Orozco Camacho. "Generación automática de memes de Internet a través de una red neuronal profunda". Tesis de mtría. Universidad Nacional Autónoma de México, 2018.

[5] Nuno MFSA Cerqueira y col. "Receptor-based virtual screening protocol for drug discovery". En: *Archives of biochemistry and biophysics* 582 (2015), págs. 56-67.

[6] Anna Choromanska y col. "The loss surfaces of multilayer networks". En: *Artificial Intelligence and Statistics*. 2015, págs. 192-204.

[7] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[8] Simon Haykin. *Neural Networks and Learning Machines*. 3.ª ed. Prentice Hall, 1999.

[9] Stanford University. Stanford Electronics Laboratories y col. *Adaptive "adaline" neuron using chemical "memistors"*. 1960.

[10] Erik R. Lindahl y col. *Molecular Modeling of Proteins*. Ed. por Andreas Kukol. Humana Press, 2008.

[11] Valère Lounnas y col. "Current progress in structure-based rational drug design marks a new mindset in drug discovery". En: *Computational and structural biotechnology journal* 5.6 (2013), e201302011.

[12] Warren S. McCulloch y Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". En: *The bulletin of mathematical biophysics* (dic. de 1943).

[13] Britta Nisius, Fan Sha y Holger Gohlke. "Structure-based computational analysis of protein binding sites for function and druggability prediction". En: *Journal of biotechnology* 159.3 (2012), págs. 123-134.

[14] OpenStax. *Biology*. OpenStax, 2013.

[15] Janaina Cruz Pereira, Ernesto Raúl Caffarena y Cicero Nogueira dos Santos. "Boosting docking-based virtual screening with deep learning". En: *Journal of chemical information and modeling* (nov. de 2016).

[16] Sebastian Raschka. *Python Machine Learning*. Packt Publishing, 2015.

[17] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[18] Stuart Russell y Peter Norvig. *Artificial Intelligence: a modern approach*. 3.ª ed. Prentice Hall, 2010.

[19] Tamar Schlick. *Molecular Modeling and Simulation - An Interdisciplinary Guide*. Springer, 2002.

[20] Hastie Trevor, Tibshirani Robert y Friedman JH. *The elements of statistical learning: data mining, inference, and prediction*. 2009.