Universidad Nacional Autónoma de México

Facultad de Ciencias

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

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Matemático

PRESENTA:

Adrián Antonio Rodríguez Pié

TUTOR

Marcelino Arciniega Castro

Ciudad Universitaria, CD. MX., 2018

Índice general

0.1. Abstract v

1. Sobre inteligencia artifical **1**
1.1. La prueba de Turing 1
1.2. Agentes 2
1.3. Tipos de aprendizaje 2

2. Sobre compuertas y neuronas **5**
2.1. Inspiracion en la biología 5
2.2. El perceptrón 7
2.3. Neuronas adaptativas lineales 10
2.4. El perceptrón multicapa 11

3. Sobre redes neuronales **13**
3.1. Descenso por el gradiente 13
3.2. Propagación hacia atrás 15
3.3. Convolución 16

4. Sobre proteínas **17**
4.1. Acoplamiento molecular 17
4.2. Función evaluadora 19
4.3. Archivos PDB 20
4.4. SMILES 20

5. Meta-análisis del acoplamiento **21**
5.1. Deep-pose 21
5.1.1. Contexto de la rama 23
5.1.2. Representación vectorial del contexto de rama . . . 24
5.1.3. Representación de la pose de un complejo proteína-ligando 25
5.1.4. Calificación de la pose 26
5.1.5. Entrenamiento de la red 27

0.1. Abstract

El descubrimiento de nuevos fármacos es un proceso muy tardado y costoso. El desarrollo e incluso el reposicionamiento de compuestos ya conocidos es una tarea sumamente complicada. El escenario es aún más complicado si se toman en cuenta los miles o millones de moléculas capáces 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 te ayuda para el desarrollo de nuevos fármacos.

Una de las metodologias computacionales más usades para investigar estas interacciones es el acoplamiento molecular. La selección de los ligandos más potentes utilizando Cribado Virtual basado en Acoplamiento (DBVS) es realizado a través de realizar la inserción de cada compuesto de la bibilioteca de compuestos en una región particular de un receptor objetivo. En la primera etapa del proceso, una búsqueda heurística es llevada a cabo en la que miles de posibles inserciones son consideradas. En la segunda etapa, la calidad de la inserción es evaluada 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) han sido usados con éxito para mejorar la salida del DBVS para tanto incrementar el desempeño de las funciones evaluadoras, como para construir clasificadores de afinidad de enlace. Una de las principales ventajas de utilizar ML es la capacidad de explicar 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 de acoplamiento y señuelos para un complejo proteína-ligando.

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** 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 humanos.

La **prueba de Turing**, propuesta por Alan Turing (1950), fué 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, este no puede distinguir si las respuestas provienen de un humano o una computadora. La discusión sobre si el pasar la prueba realmente es señal de una *inteligencia* sale del alcance de esta tésis, pero es importante resaltar que la prueba da cierta noción sobre lo

que se busca en una ïnteligencia artificial". Se busca que el ente ïnteligente"sea capáz de tener un razonamiento racional. Es decir, que pueda sacar conclusiones lógicas a partir de premisas dadas a través de silogismos, como lo haría un humano. Pero rara vez toda la información con la que se cuenta es 100 % acertada, lo que le da en la torre al enfoque puramente determinista de la inteligencia.

1.2. Agentes

En su concepción más tradicional, la inteligencia artificial gira en torno a **agentes racionales** que, a través de **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 una se da una transición de estados, entonces es realizada una acción. Esta concepción del agente nos permite reducir la IA en 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 posible hacer la codificación de todos los posibles escenarios en el sistema de inferencia de un agente. Por lo tanto un agente debe ser capáz de manejar la incertidumbre en su entorno. Además, debe poder *aprender* del entorno y adaptarse a él, generando conocimiento a través de la experiencia.

El **aprendizaje de máquina** (*machine learning* en inglés) gira en torno a algorítmos capáces de manejar información estocástica y generar modelos de toma de decisiones a partir de ella. Un modelo es bueno en 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 tipos de aprendizaje. Se describen a continuación cuatro diferentes parámetros bajo los cuáles 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 interactua con el ambiente durante el entrenamiento, realizando consultas o experimentos, mientras que un agente pasivo sólo observa la información provista por el ambiente sin influenciarla 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, 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 líneal, es decir que para cada ejemplo de entrenamiento tiene que dar una respuesta, y cuando el agente tiene que mostrar el aprendizaje adquirido después de tener la oportunidad de procesar grandes cantidades de datos.

CAPÍTULO 2

Sobre compuertas y neuronas

Los modelos de redes neuronales artificiales, uno de los cuales es el *perceptrón* que discutiremos en este capitulo, están inspirados en el cerebro humano. Hay científicos cognitivos y de neurociencia cuya meta es entender el funcionamiento del cerebro, y con esto en mente, construir modelos de 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 ingenieria, como son visión, aprendizaje y reconocimiento del habla, por nombrar algunos.

2.1. Inspiracion 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: las neuronas en el cerebro tienen conexiones, llamadas *sinápsis*, a 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 realizado por las neuronas y la memoria se encuentra en las sinápsis entre ellas.

El cerebro es, además, un órgano capás 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].

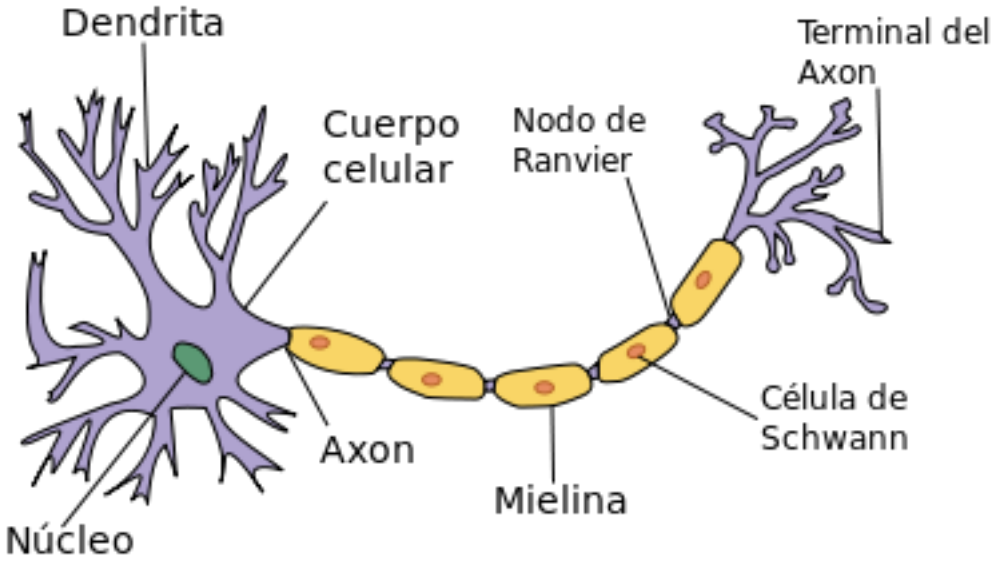


Figura 2.1: Estructura de una neurona. (Tomado de `https://es.wikipedia.org/wiki/Neurona`)

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) [11].

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 publica la primera aproximación al concepto de perceptron basado en el modelo de neuronas MCP [14]. 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 [13].

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 activación de función $\phi(\cdot)$ 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 $z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \mathbf{w^T x}$. 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):

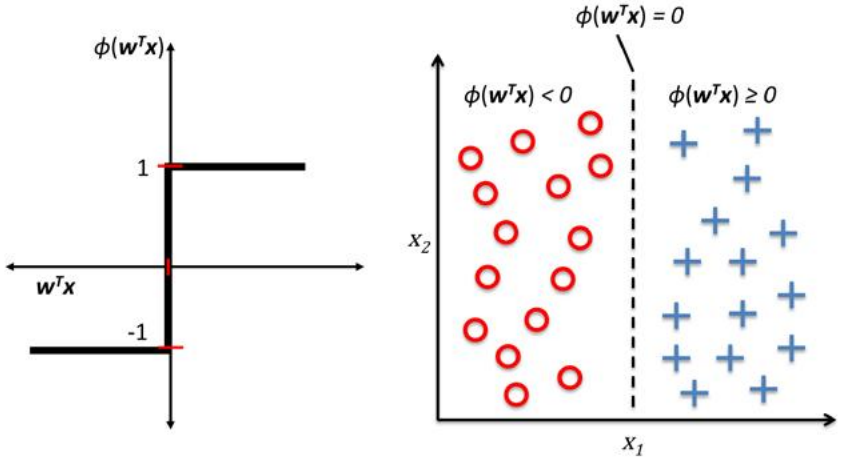


Figura 2.2: *Aplanamiento* de la salida del perceptrón para hacer clasificación. (Tomado de [13])

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 este caso, el valor de salida es la clasificación dada por la función escalón definida previamente, y la actualización simultánea de cada peso w_j en el vector de pesos w puede ser escrito más formalmente cómo:

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

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

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

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.

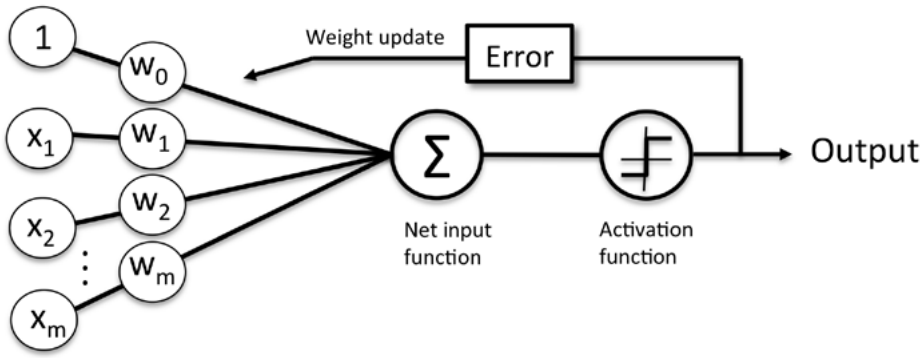


Figura 2.3: Resumen del concepto general de perceptrón. (Tomado de [13])

La figura 2.3 muestra cómo el perceptron 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

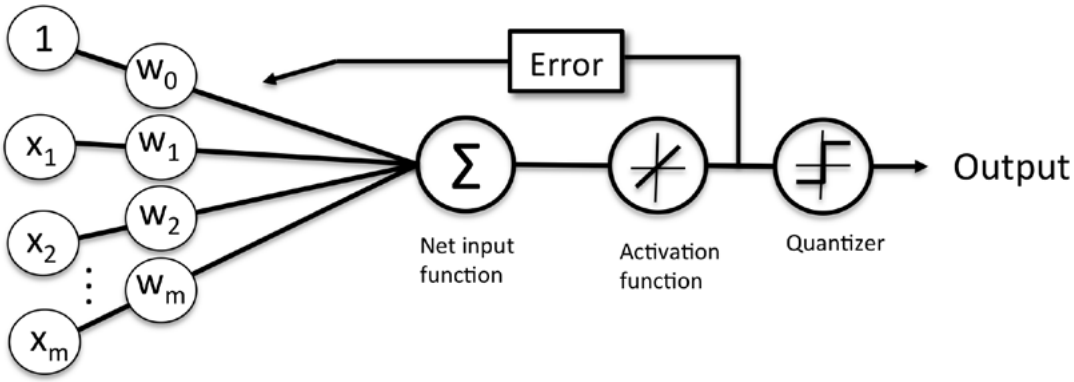


Figura 2.4: Esquema de Adaline. (Tomado de [13])

La neurona adaptativa lineal (**Adaline**) fué publicada unos años después del algoritmo del perceptrón de Frank Rosenblatt, por Bernard Widrow [8] y se considera la evolución natural del perceptrón de Rosenblatt. 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 mas 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 compara la figura 2.3 con la figura 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 más suavizadas y, sobre todo, diferenciables. Esto con el fin de optimizar los parámetros de los modelos más complejos.

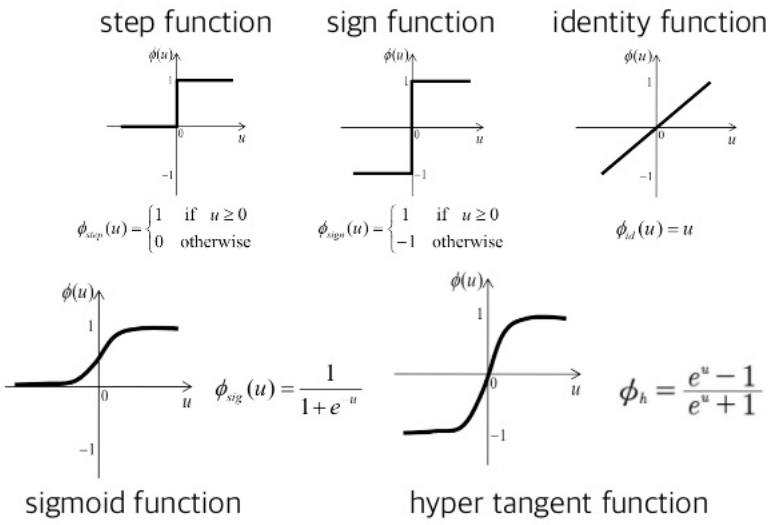


Figura 2.5: Algunos ejemplos de otras funciones de activación. (Tomado de `https://www.slideshare.net/SungJuKim2/multi-layer-perceptron-back-propagation`)

2.4. El perceptrón multicapa

El perceptrón multicapa es una generalización del perceptrón simple de Rosenblatt, como conescuencia de las limitaciones de este ante conjuntos 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* dónde se da la clasificación. Cada capa intermedia, 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* por sus siglas en inglés) es una gráfica $n - partita$ dirigida donde n seria el número de capas.

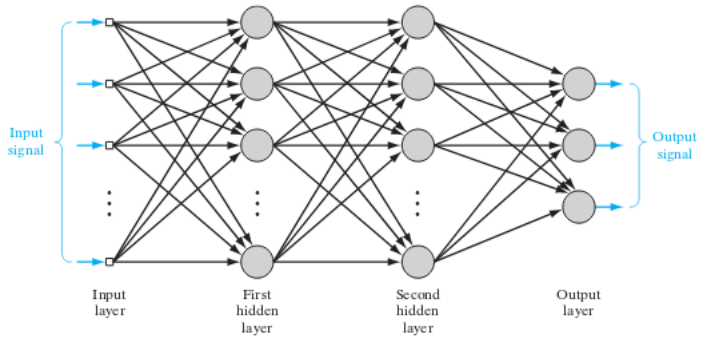


Figura 2.6: Arquitectura de un MLP con dos capas ocultas (Tomado [7])

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* (en inglés, *bias*), del siguiente modo:

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

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

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

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

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

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

CAPÍTULO 3

Sobre redes neuronales

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3.1. 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 cua-**

drados (SSE) entre la salida calculada y el valor real.

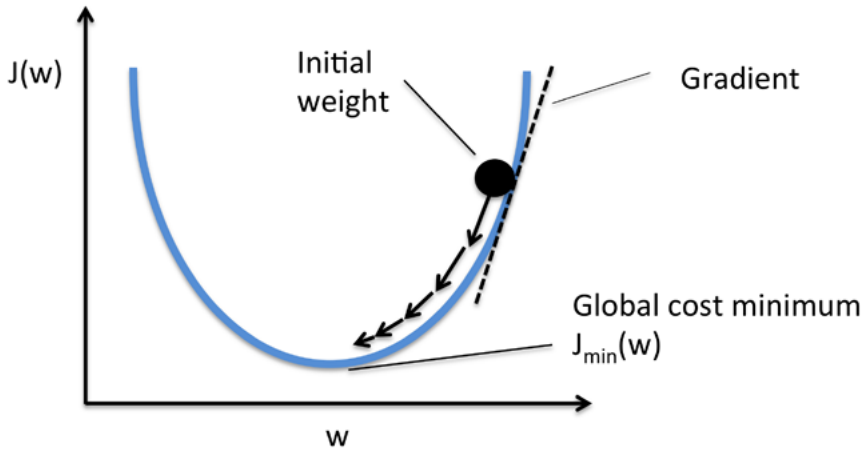
$$J(w) = 1/2n \sum_i (\hat{y}_i - y_i^2) \tag{3.1}$$

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.



Usando el descenso por el gradiente, se pueden actualizar los pesos

quitandole un *paso* al gradiente $\nabla J(w)$ de la función de costos $J(w)$:

$$w := w + \Delta w \tag{3.2}$$

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

$$\Delta w := -\eta \Delta J(w) \tag{3.3}$$

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

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

Las redes neuronales (ANN por sus siglas en inglés) 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. Sin embargo, redes altamente eficientes por lo general son entrenadas utilizando descenso por el gradiente. ¿Cómo explicar esta aparente contradicción?

Recientemente, evidencia empírica [5] 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 funciona "suficientemente bien"**. Hay otros métodos más avanzados que son empleados para entrenar ANNs, pero este trabajo utiliza descenso por el gradiente por su simplicidad e interpetación tan intuitiva.

3.2. Propagación hacia atrás

FALTA SECCIÓN [7] 129 [4] 36 http://www.iro.umontreal.ca/ pift6266/H10/notes/mlp.

3.3. Convolución

FALTA SECCIÓN

CAPÍTULO 4

Sobre proteínas

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 (~~life-sustaining~~). Las proteínas proveen el soporte para la arquitectura de tejido musculoso, ligamentos, tendones, huesos, piel, cabello, órganos y glándulas. Las proteínas también proveen los servicios fundamentales de transporte y almacenamiento como en el caso del oxígeno y hierro en células musculares y eritrocitos. Las proteínas también juegan un rol crucial en muchos procesos regulatorios esenciales para la vida, como reacciones de catálisis (e.g. digestión); funciones inmunológicas y hormonales; y la coordinación de actividades neuronales, crecimiento de células y hueso, y diferenciación celular.[16]

4.1. Acoplamiento molecular

Basado en [9] El campo del **acoplamiento molecular** o **docking** surge a lo largo de las últimas tres décadas gracias a la necesidad de la biología molecular estructural y el descubrimiento de hinibidores basado en estructuras. Ha podido evolucionar considerablemente gracias al crecimiento dramático de disponibilidad y poder de las computadoras, y al

creciente acceso a bases de datos de proteínas y moléculas.

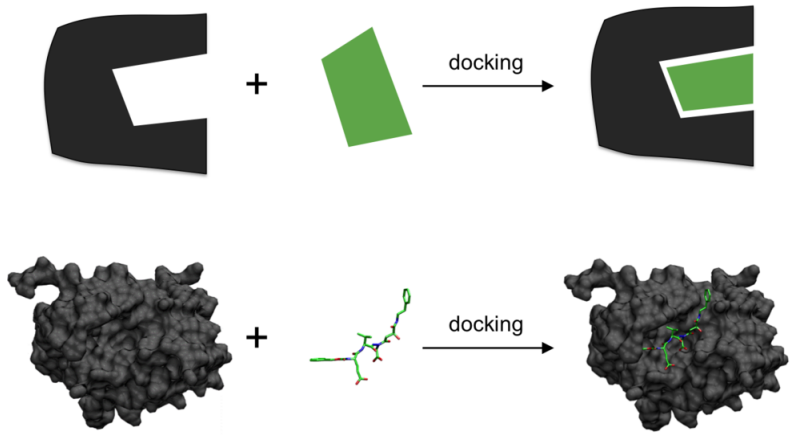


Figura 4.1: Representación esquemática del *docking*. (Tomado de `https://en.wikipedia.org/wiki/Docking_(molecular)`)

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, llamada ligando, y una macromolécula objetivo, una proteína en nuestro caso.

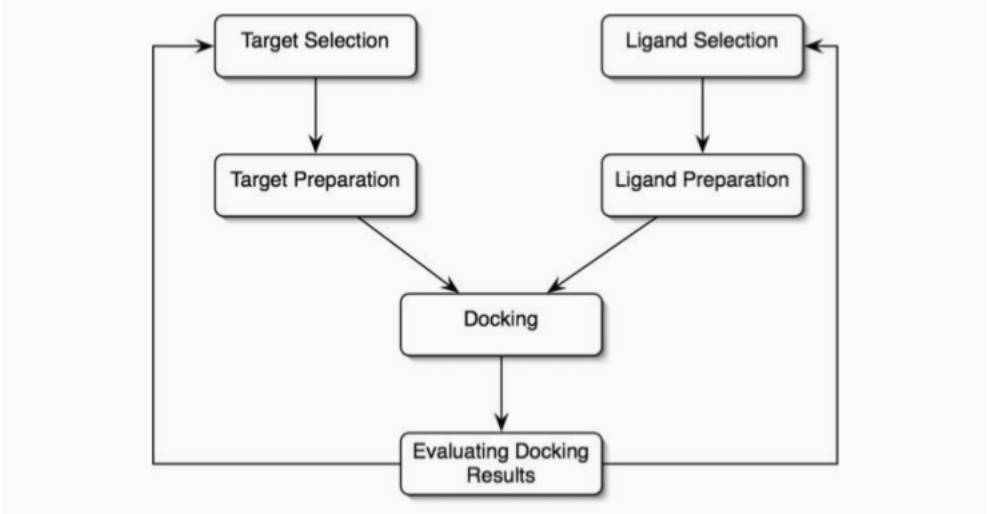


Figura 4.2: Diagrama de flujo para un acoplamiento usual. (Tomado de [9])

La figura 4.2 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. Estas 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 torción para cada enlace de rotación). Cada una de las variables de estado describe un grado de libertad en un espacio de búsqueda multidimensional.

4.2. 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) 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.

4.3. Archivos PDB

~~¿Esto está bien aquí? ¿Habría que profundizar?~~ El banco de datos de proteínas es un archivo de estructuras de macromoléculas biológicas determinadas experimentalmente. El formato utilizado para almacenar está información 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. [1]

4.4. SMILES

~~¿Vale la pena poner esto?~~ SMILES (Simple Molecular Input Line Entry System) es un sencillo lenguaje químico que permite describir moléculas y reacciones utilizando únicamente caracteres ASCII 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.[2]

[1] `ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v33_Letter.pdf`

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

CAPÍTULO 5

Meta-análisis del acoplamiento

~~Se tomó una base de datos cristalográfica de acoplamientos ya realizados y sobre esos ligandos y proteinas se corrió Docking con AutoDockVina n.m. Después se hace una tabla comparando los scores que les asigna AutoDockVina con el RMSDI cristalográfico. Pero los scores de autodockvina no siempre son muy buenos y pasa que muchas veces la que es la mejor pose la manda al 4^o o 5^o lugar del ranking.~~

5.1. 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 genera una representación vectorial del complejo. Después se pasa a dos capas ocultas para sintetización y procesamiento del vector-ligando.

Finalmente, en la última capa, la representación del complejo es dada como entrada a un clasificador *softmax*, quien 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:
$W^{b\_type} \in \mathbb{R}^{h\times|B|}, W^{b\_dist} \in \mathbb{R}^{h\times|B|}$,
$W^{conv} \in \mathbb{R}^{|z_i|\times cf}, W^3 \in \mathbb{R}^{cf\times h}$,
$W_{out} \in \mathbb{R}^{h\times 2}$,
$b^{conv} \in \mathbb{R}^{cf}, b^3 \in \mathbb{R}^h$,
$b^{out} \in \mathbb{R}^2$
$Z \leftarrow [..]$
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.

5.1.1. Contexto de la rama

Pereira, Caffaren y dos Santos [12] 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 este y de los átomos más cercanos. Pereira buscaba encontrar el acoplamiento que generara la mayor cantidad de energía, sin importar cual fuera la conformación necesaria para poder realizarlo; nuestro enfoque busca precisamente encontrar dicha conformación.

Partiendo de la ídea del átomo ligado a su contexto, y considerando que lo que buscamos encontrar 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 este la combinación de el tipo de rama, los de las ramas más cercanas y la distancia a cada una de ellas.

IMAGEN HISTOGRAMA DE DISTRIBUCIÖN DE TIPOS DE RAMA

Se divide cada receptor y ligando en sus respectivas ramas y cada una de estas ramas se codifica usando la respresentación SMILES [1]. 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*.

SMILES	Id	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 5.1: Fragmento de los diccionarios de ramas y de distancias

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

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 5.2: 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.

5.1.2. 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 $W^{b\_type} \in \mathbb{R}^{h\times|B|}$ corresponde al vector característico de un tipo de rama, donde B es el conjunto de tipo de ramas y h 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 a 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 5.3: 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 $z_b = z_{b\_type} \cdot z_{b\_dist}$. La idea 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.

5.1.3. 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 contextos de ramas y sintetizar la información de todas ellas en un vector r de longitud fija. La preocupación subyaciente que ataca el uso de una capa convolucional es la capacidad de manejar entradas de distintas dimensiones. La cantidad de ramas por ligando es variable, entonces la capa convolucional nos permite procesar 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

$$u_i = f(W^{conv}_{z_i} + b^{conv}) \tag{5.1}$$

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 en la capa convolucional (cf) es un hiperparámetro definido por el usuario.

La segunda etapa en la capa convolucional correspondiende 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:

$$[r]_j = \max_{1 \leq i \leq n} [u_i]_j \tag{5.2}$$

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.

5.1.4. 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:

$$s(x) = W^{out}(w^3 r + b^3) + b^{out} \tag{5.3}$$

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*:

$$p(0|x) = \frac{e^{s(x)_0}}{e^{s(x)_0} + e^{s(x)_1}} \tag{5.4}$$

$$p(1|x) = \frac{e^{s(x)_1}}{e^{s(x)_0} + e^{s(x)_1}} \tag{5.5}$$

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.

5.1.5. Entrenamiento de la red

Deep-pose se entrenó utilizando un algoritmo de **descenso estocástico por el gradiente** (SDG por sus siglas en inglés). En este caso, SDG se utiliza para minimizar la función de costos sobre un conjunto de entrenamiento D que contiene poses tanto válidas como señuelos. En 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 ~~backpropagation~~.

$$\theta \longmapsto \sum_{(x,y)\in D} -\log p(y|x, \theta) \tag{5.6}$$

En este trabajo, se utilizaron *minilotes* de complejos, 30 por lote, y tomamos el promedio del error de la predicción para realizar el ~~backpropagation~~. La red se implementó utilizando la biblioteca Theano[2].

[2]http://deeplearning.net/software/theano/

Bibliografía

[1] Ethem Alpaydin. *Introduction to Machine Learning*. 2nd. 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] Anna Choromanska y col. "The loss surfaces of multilayer networks". En: *Artificial Intelligence and Statistics*. 2015, págs. 192-204.

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

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

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

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

[10] 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.

[11] 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).

[12] 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).

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

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

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

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

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