Universidad Nacional Autónoma de México

Facultad de Ciencias

Evaluación de Arquitecturas de Redes Neuronales Profundas para la Cuantificación de Similitud de Estructuras de Proteínas.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Licenciado en Ciencias de la Computación

PRESENTA:

José Ismael Fernández Martínez

TUTOR

Ricardo Corral Corral

Ciudad Universitaria, Cd. Mx. 2018

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

A mis padres, Gaby y José, por todo el apoyo y paciencia que tuvieron durante este proceso. A mi hermanos, Regina y Sergio, por el interés y la ayuda que me brindaron mientras escribía. A mis amigos quienes fueron pacientes y alentadores cuando desaparecía para redactar y estudiar. Pero en especial a Ricardo por el gran interés de ensenar sin esperar nada a cambio, por contagiarme su pasión por aprender, por toda la paciencia y más importante por ser mi mentor.

Notación

a	Un escalar (entero o real).
$\boldsymbol{a}$	Un vector.
$\boldsymbol{a}_i$	El elemento i del vector $\boldsymbol{a}$.
$\boldsymbol{A}$	Una matriz.
$\boldsymbol{A}^T$	Transpuesta de la matriz $\boldsymbol{A}$.
$\mathbb{A}$	Un conjunto.
$\{0, 1, ..., n\}$	El cojunto de todos los enteros entre 0 y n.
$[a, b]$	El intervalo real incluyendo a y b.
$(a, b]$	El intervalo real excluyendo a pero incluyendo b.
$\frac{\delta x}{\delta y}$	La derivada parcial de y con respecto a x.
$f(\boldsymbol{x}; \boldsymbol{\theta})$	Una función de $\boldsymbol{x}$ parametrizada por $\boldsymbol{\theta}$.
$\sigma(x)$	La función *sigmoid*, $1/(1+e^{-x})$.

Capítulo 1

Introducción

Cuando una compañía farmacéutica desea buscar en una gran base de datos de proteínas para encontrar una que sea similar en forma o actividad a una proteína que se ha descubierto, surge la siguiente pregunta: ¿Cuál es la forma más eficiente de hacerlo? Convencionalmente la similitud entre dos proteínas es calculada con métodos que buscan disminuir la distancia geométrica entre sus correspondientes átomos, desafortunadamente este proceso es muy costoso haciéndolo prácticamente intratable cuando se desea evaluar la similitud entre todos los posibles pares de una gran base de datos como el Banco de Datos de Proteínas (*PDB* por sus siglas en inglés). Shivashankar *et al.* [41] expone algunas de las dificultades particulares de obtener proteínas similares a una proteína dada:

1. Encontrar una representación de estructuras de proteínas la cual capture la inherente relación entre éstas y facilite su comparación.

2. Y desarrollar técnicas que eficientemente aborden los diferentes requerimientos de obtención de estructuras similares.

Un enfoque prometedor que da respuesta a estas dificultades es el uso de una representación de proteínas basada en el Sistema de Clases de Cúmulos de Residuos [13] y modelos de aprendizaje de máquina. Usando este enfoque, específicamente con modelos de Bosques Aleatorios, Corral-Corral *et al.* [13] consiguieron puntajes cerca del estado del arte, incluso superiores, en las tareas de clasificación estructural automática e identificación de vecinos estructurales.

Por otro lado, en años recientes se ha visto que el aprendizaje profundo, una subrama del aprendizaje de máquina, ha conseguido revolucionar en diversas áreas de las cuales destacan:

- Reconocimiento de objectos en imágenes a nivel humano
- Reconocimiento de voz a nivel humano
- Conducción autónoma a nivel humano
- Traducción automática

A diferencia de los modelos de aprendizaje de máquina regular, los modelos de aprendizaje profundo requieren conjuntos de datos relativamente grandes para desempeñarse bien, pero de acuerdo con Ladislav Rampasek *et al.* [36] configurar un

modelo de aprendizaje profundo que maximice su eficacia es una tarea complicada que históricamente se ha reservado solo a expertos en el área.

Asimismo, el aprendizaje profundo ha contribuido significativamente en el problema de predicción de estructura de proteína a partir de su secuencia, ejemplo de esto se puede encontrar en [10] y [34]. Sin embargo, lo mismo no ha sucedido con los problemas relacionados con similitud estructural entre proteínas.

Motivado por el gran éxito que el aprendizaje profundo ha conseguido en diversas áreas históricamente difíciles para el aprendizaje de máquina regular, este trabajo tiene como objetivo evaluar el uso de modelos de aprendizaje profundo en la solución de las tareas de clasificación estructural automática y búsqueda de vecinos estructurales. Específicamente en este trabajo se evalúa el uso de una representación de proteínas basada en el Sistema de Clases de Cúmulos de Residuos y diferentes arquitecturas de redes neuronales profundas en las tareas anteriormente mencionadas. Asimismo, esta investigación muestra un flujo de trabajo para el diseño, optimización y evaluación de hiperparámetros de arquitecturas de aprendizaje profundo. Finalmente, la contribución más significativa de este trabajo es el desarrollo de una técnica de regularización que consiste en el uso de ensambles de modelos de aprendizaje profundo para la tarea de clasificación jerárquica de estructuras de proteínas. Aquí se analiza el rendimiento de diferentes arquitecturas de aprendizaje profundo en la clasificación estructural automática sobre las principales clasificaciones estructurales *CATH* y *SCOP*, y se evalúa el uso de las mismas en la tarea de búsqueda de vecinos estructurales de acuerdo a *SAS* y *SCOP*.

Hipótesis

Existen arquitecturas de redes neuronales profundas que pueden desempeñarse mejor que métodos reportados en tareas de clasificación y cuantificación de similitud estructural de proteínas usando una representación basada en el sistema de Clases de Cúmulos de Residuos.

Objetivos

Objetivos general

Diseñar, implementar y evaluar el desempeño de arquitecturas de redes neuronales profundas usando una representación basada en en el sistema de Clases de Cúmulos de Residuos para tareas de clasificación y cuantificación de similitud estructural de proteínas.

Objetivos específicos

- Encontrar modelos de aprendizaje profundo que se desempeñen mejor que métodos reportados en la tarea de clasificación estructural automática sobre los niveles clase, arquitectura y topología de la clasificación de dominios *CATH*; y superfamilia de la clasificación de dominios *SCOP*.
- Medir la capacidad de identificar vecinos estructurales de proteínas aproximando la similitud estructural de proteínas a partir de modelos de clasificación de superfamilia de *SCOP*.

Capítulo 2

Antecedentes

En primer lugar diversos conceptos relacionados a las estructuras de proteínas son explicados para poder desarrollar las ideas referentes a este trabajo.

2.1. Aminoácido

Un aminoácido es una molécula que consiste de un grupo amino ($-NH_2$), un grupo carboxilo ($-COOH$) y un grupo orgánico R (también conocido como cadena lateral). Todo aminoácido contiene un átomo central carbono (C) llamado carbono alfa, al cual los grupos amino, carboxilo y el grupo orgánico R están unidos, además de un átomo de hidrógeno (H). En la figura 2.1 se ilustra la fórmula general de un aminoácido. Para más información acerca de los aminoácidos véase [39].

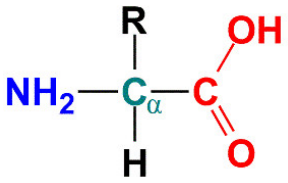

Figura 2.1: Aminoácidos. Recuperado de [1].

2.1.1. Enlace peptídico

Dos aminoácidos pueden ser unidos por una reacción química en la cual un $-OH$ se pierde del grupo carboxilo de un aminoácido junto con un hidrógeno del grupo amino del segundo aminoácido, formando una molécula de agua y dejando a los dos aminoácidos unidos vía un enlace amido llamado enlace peptídico, véase la figura 2.2.

Cuando varios aminoácidos son unidos mediante enlaces peptídicos se dice que constituyen una proteína. Después de que los aminoácidos son incorporados a la proteína, los aminoácidos individuales son conocidos como residuos de aminoácidos y las serie de átomos enlazados de carbono, nitrógeno y oxigeno son conocidos como cadena principal. Típicamente a las proteínas constituidas por menos de 50 aminoácidos se les conoce como péptidos. Para más detalles respecto al enlace peptídico véase [39].

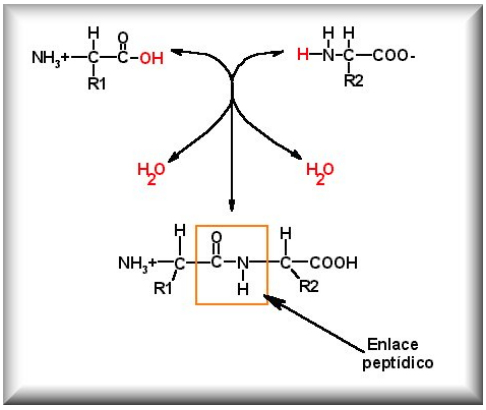

Figura 2.2: Enlace peptídico. Recuperado de [45].

2.2. Estructura de proteínas

Una proteína es una macromolécula compuesta por residuos de aminoácidos. Ésta es caracterizada por:

- Su secuencia de aminoácidos, la cual define los tipos de átomos que la componen y como están conectados.
- Su estructura tridimensional, la cual define como cada átomo está posicionando en el espacio.
- Su función.

Éstas propiedades están relacionadas entre sí, ya que la secuencia de aminoácidos se pliega adoptando la estructura tridimensional de la proteína y la función de ésta depende esa estructura tridimensional.

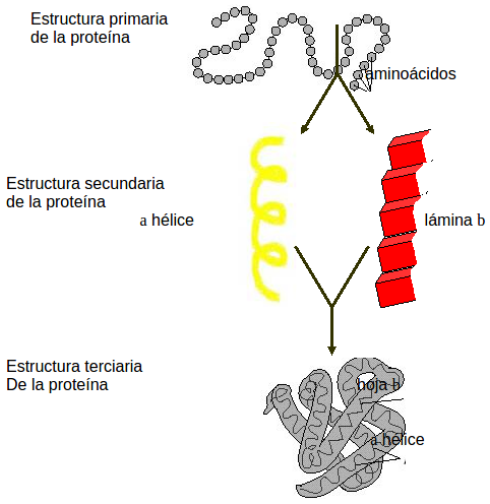

Figura 2.3: Los diferentes niveles de estructura en una proteína. Recuperado de [46].

A la secuencia de aminoácidos de una proteína se le conoce como estructura primaria, mientras que a la estructura tridimensional se le llama estructura terciaria o plegado. La estructura secundaria es la forma de segmentos locales de la proteína, los elementos más comunes de estructura secundaria encontrados en las proteínas son las hélice-α y las hojas y vueltas β, véase la figura 2.3.

Ya que la función de una proteína depende de la estructura tridimensional que tiene, una forma de detectar aquellas proteínas que comparten una función similar es evaluando similitudes entre sus plegados. Un ejemplo comúnmente usado para ilustrar esto es la comparación entre la hemoglobina y la mioglobina ya que ambas proteínas son estructuralmente y funcionalmente muy parecidas, siendo la función de ambas transportar oxigeno, véase la figura 2.4.

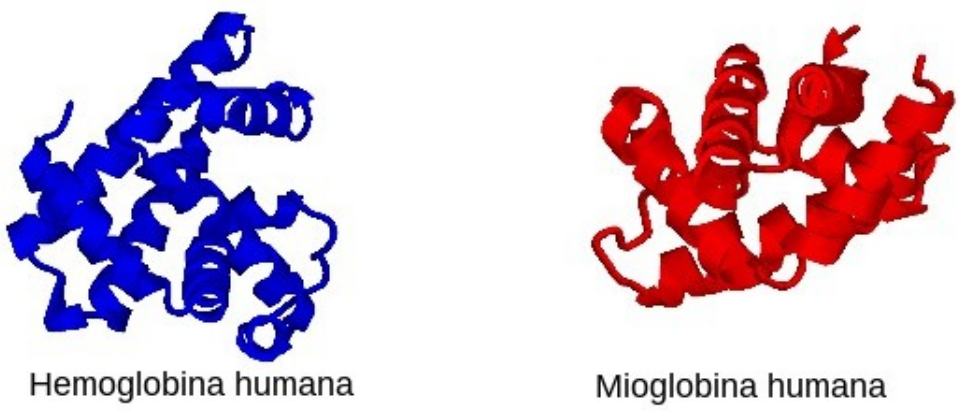

Figura 2.4: Hemoglobina y mioglobina. Recuperado de [9].

Las proteínas juegan un papel muy importante dentro de los organismos ya que realizan gran parte del trabajo en las células. Para más información acerca de las estructuras de proteínas véase [26].

2.2.1. Dominio de proteína

A la unidad estructural de las proteínas se le conoce como dominio de proteína. No existe un consenso de cómo deben ser definidos los dominios, pero en general los dominios de proteínas son una unidad formada por un tramo de una secuencia de aminoácidos que interactúa débilmente con dominios adyacentes. Una proteína puede consistir de varios dominios. La función de la proteína puede ser asociada con uno o más dominios. Para más información acerca de los dominios de proteínas véase [26].

2.2.2. Banco de datos de proteínas

El banco de datos de proteínas [7] (*PDB* por sus siglas en inglés) es un depósito que almacena información descriptiva acerca de estructuras de proteínas y otras macromoléculas como ácidos nucléicos. La información almacenada consiste principalmente en archivos de coordenadas; estos archivos listan los átomos de cada estructura y su posición tridimensional. Un archivo *.pdb* que describe una proteína

consiste de cientos o hasta miles de lineas como las siguientes:

```
# 1HIV.pdb

HEADER    HYDROLASE/HYDROLASE INHIBITOR       12-FEB-92  1HIV
TITLE     CRYSTAL STRUCTURE OF A COMPLEX OF HIV-1 PROTEASE WITH A
TITLE    2 DIHYDROETHYLENE-CONTAINING INHIBITOR: COMPARISONS WITH MOLECULAR
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    N.THANKI,A.WLODAWER
...
REMARK   2 RESOLUTION.   2.00 ANGSTROMS.
REMARK   3 REFINEMENT.
REMARK   3   PROGRAM     : PROLSQ
REMARK   3   AUTHORS     : KONNERT,HENDRICKSON
...
...
ATOM      1  N   PRO A   1      -3.190   7.728  33.820  1.00 21.66           N
ATOM      2  CA  PRO A   1      -2.220   6.922  34.499  1.00 18.48           C
ATOM      3  C   PRO A   1      -0.802   7.080  34.031  1.00 17.67           C
ATOM      4  O   PRO A   1      -0.530   7.806  33.045  1.00 18.49           O
ATOM      5  CB  PRO A   1      -2.727   5.495  34.165  1.00 20.72           C
...

# Tomado del archivo que describe la estructura de un complejo cristalino
# del virus recombinante de la inmunodeficiencia humana tipo 1
```

Donde:

- *HEADER, TITLE y AUTHOR*: proveen información acerca de los investigadores que determinaron la estructura, etc.

- *EXPDTA*: indica la técnica usada para determinar la estructura.

- *REMARK*: provee una anotación en formato libre, pero también puede encontrarse información estandarizada.

- *ATOM*: describe las coordenadas de los átomos que son parte de la proteína.

Para efectos de ilustración solo se muestran algunos campos del archivo *1HIV.pdb*, para más detalles acerca de los archivos *.pdb* véase [35].

En años recientes se ha visto un crecimiento exponencial del *PDB*, lo cual refleja el auge de la investigación que está sucediendo en los laboratorios a través del mundo. Actualmente hay 137,178 proteínas depositadas en el *PDB*.

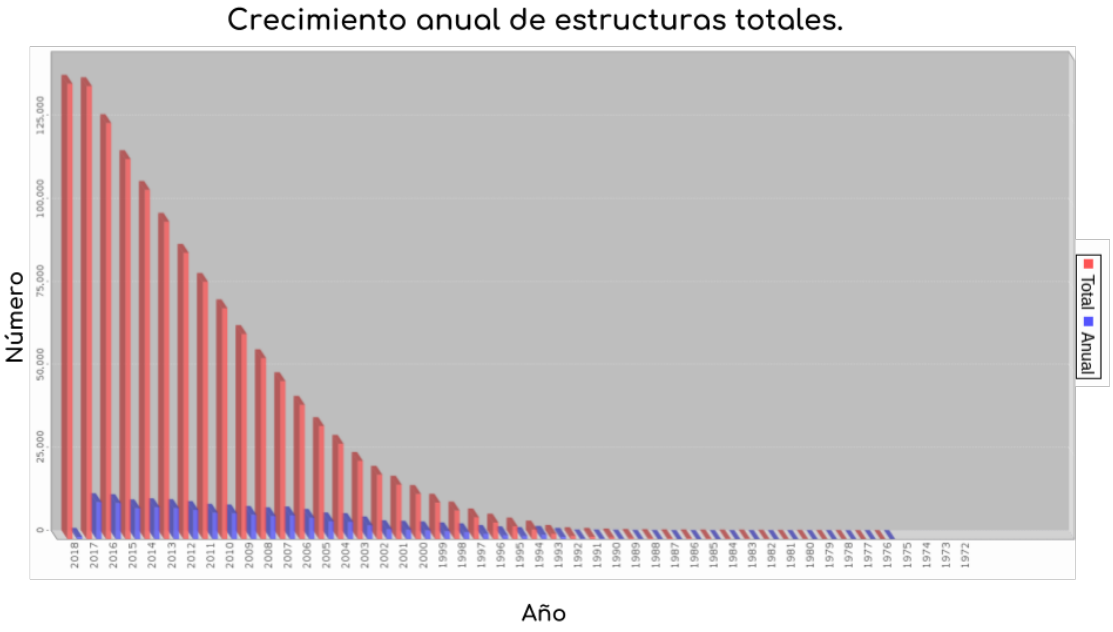

Figura 2.5: Crecimiento anual del *PDB*. Recuperado de [48]

2.2.3. Alineamiento estructural

El alineamiento estructural es el proceso más usado para cuantificar similitud entre dos estructuras de proteínas. En un alineamiento estructural la similitud entre dos estructuras es calculada con métodos que buscan disminuir la distancia geométrica entre los átomos de la pareja de estructuras, véase la figura 2.6. Generalmente el resultado de un alineamiento estructural es: una superposición de los átomos de la pareja de estructuras y la mínima raíz de la desviación cuadrática media (*RMSD* por sus siglas en inglés) entre las estructuras.

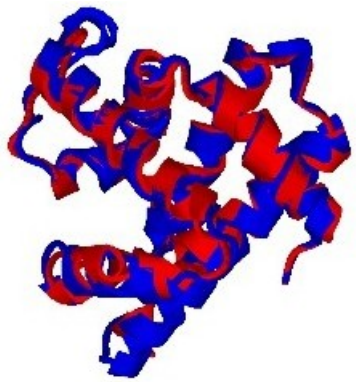

Figura 2.6: Ilustración de un alineamiento estructural entre la hemoglobina y la mioglobina de la figura 2.4. Recuperado de [9]

Existen múltiples métodos de alineamiento estructural. En general alinear dos estructuras es un proceso computacional caro, ya que determinar la transformación que posiciona de forma óptima a las dos estructuras es una tarea no trivial; y aunque se ha demostrado que un alineamiento estructural puede realizarse en tiempo

polinómico $O(n^{10})$ [25], alinear una estructura nueva contra todas las estructuras contenidas en alguna base de datos como el *PDB*, es una tarea sumamente costosa. Para más información acerca del alineamiento estructural véase [26].

2.2.4. Clasificación estructural

A fin de proveer una vista ordenada de las grandes cantidades de datos presentes en el *PDB* varios métodos de clasificación estructural han sido desarrollados. Estos métodos facilitan la comparación de estructuras de proteínas y proveen una mejor comprensión estructural y funcional de las mismas. Las principales clasificaciones estructurales son *CATH* y *SCOP*, las cuales mantienen clasificaciones jerárquicas de estructuras de proteínas conocidas.

CATH

CATH [33] es una clasificación jerárquica semi-automática de dominios de proteínas que facilita la asignación de relaciones estructurales, funcionales y evolutivas para proteínas con estructura conocida. Los cuatro niveles principales de clasificación de *CATH* son:

- ***C****lass.*- Clase, es el nivel más simple y esencialmente describe la composición de estructuras secundarias de cada dominio.
- ***A****rchitecture.*- Arquitectura, resume la forma revelada por las orientaciones de las unidades de estructura secundaria, tales como *barrels* y *sandwiches.*
- ***T****opology.*- Topología, en este nivel la conectividad secuencial es considerada tal que los miembros de la misma arquitectura pueden tener diferentes topologías.
- ***H****omologous superfamily.*- Superfamilia homóloga, agrupa estructuras que corresponden al mismo nivel T y que comparten gran similitud estructural y funcional, se asume que las proteínas están evolutivamente relacionadas.

CATH asigna dominios mediante un procedimiento de consenso basado en varios algoritmos de reconocimiento de dominios. Cuando los algoritmos generan el mismo resultado, la asignación de dominios es en forma automática. Sin embargo, cuando no existe consenso evidente, se realiza una asignación manual mediante inspecciones visuales.

SCOP

La Clasificación Estructural de Proteínas [32] (*SCOP* por sus siglas en inglés) es una clasificación de dominios de proteínas, la cual fue construida en su mayoría en forma manual y provee una descripción detallada de las relaciones estructurales y evolutivas de las proteínas con estructura conocida. *SCOP* está organizada en niveles jerárquicos que representan las relaciones evolutivas y estructurales:

- *Class.*- Clase, es el nivel más simple y agrupa los tipos de plegados en clases.
- *Fold.*- Plegado, reúne los diferentes tipos de formas de los dominios dentro de una clase.

- *Superfamily.*- Superfamilia, conjunta proteínas cuyas características funcionales sugieren que un origen evolutivo común es probable.

- *Family.*- Familia, agrupa proteínas que tienen un origen común evolutivo detectable en secuencia.

En general las clasificaciones estructurales varían debido a que sus estrategias de clasificación dependen de decisiones específicas, algoritmos, parámetros y muchas veces intervención humana; por consecuencia, ofrecen diferentes vistas ordenadas de las estructuras de proteínas. Las clasificaciones estructurales son muy importantes debido a que ofrecen atajos en la identificación de características estructurales, además son ampliamente usadas en la predicción de función de proteínas. Para más información acerca de las clasificaciones estructurales véase [26].

2.2.5. Espacio de plegados

El espacio de plegados es un marco conceptual en el que cualquier estructura de proteínas puede ser representada. En este espacio se relacionan estructura, función y evolución de las proteínas. Existen diversas representaciones de este espacio, algunas se basan en espacios vectoriales y en ellas cualquier estructura de proteína es un punto en el espacio. En general buscan que proteínas similares se encuentren juntas. Algunas de las interpretaciones, como la de Jingtong Hou *et al.* [21], ajustan las posiciones de las estructuras de manera que las distancias entre ellas aproximen alguna medida de similitud.

Espacio de plegados basado en Clases de Cúmulos de Residuos

Un cúmulo de residuos se define como el conjunto de residuos de aminoácidos que están en proximidad en la representación tridimensional de una estructura de proteína. Se dice que dos residuos están en proximidad si estos comparten al menos una pareja de átomos, que no sean de hidrógeno, a no más de ϵ Å (Ångströms) en la representación tridimensional. Cúmulos de tamaño 3 hasta 6 son considerados para definir un sistema de 26 clases, donde diferentes clases son asignadas a los cúmulos de acuerdo a la contigüidad en la secuencia de la proteína. Así que, para cualquier proteína existen 3 clases para cúmulos de tamaño 3 ([1,1,1], [1,2], [3]), 5 clases para cúmulos de tamaño 4 ([1,1,1,1], [1,1,2], [2,2], [1,3], [4]), 7 clases para cúmulos de tamaño 5 ([1,1,1,1,1], [1,1,1,2], [1,2,2], [1,1,3], [2,3], [1,4], [5]) y 11 clases para cúmulos de tamaño 6 ([1,1,1,1,1,1], [1,1,1,1,2], [1,1,2,2], [2,2,2], [1,1,1,3], [1,2,3], [3,3], [1,1,4], [2,4], [1,5], [6]). Y usando este sistema es posible construir una representación vectorial del espacio de plegados, de manera que toda proteína sea representada por un vector de 26 posiciones. En cada entrada este vector describe el número de ocurrencias de cada clase definida en el sistema. Dicha representación del espacio de plegados naturalmente segrega estructuras de proteínas similares en regiones definidas y es aprendible por cualquier algoritmo de clasificación de aprendizaje de máquina. Para más detalles del espacio de plegados basado en Clases de Cúmulos de Residuos, *RCC* por sus siglas en inglés, véase [13].

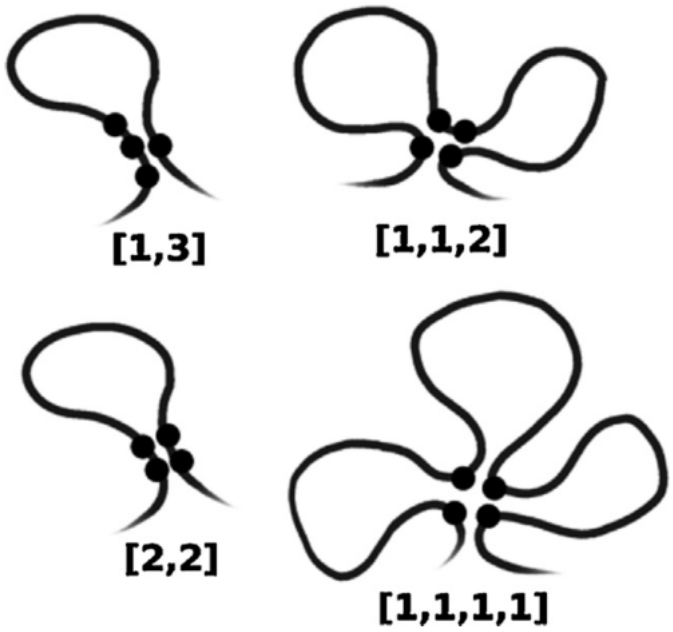

Figura 2.7: La curva representa la cadena principal de la proteína, y los puntos negros representan residuos siendo considerados en un cúmulo. Se muestran cuatro distintas clases ([1,3], [1,1,2], [2,2], [1,1,1,1]) que surgen de un cúmulo de tamaño 4. El nombre de la clase se define por el número de residuos en cada segmento que define *RCC*. Por convención los números son escritos en orden ascendente. Recuperado de [13]

Capítulo 3

Trabajo relacionado

3.1. Clasificación estructural automática

Diversos métodos han sido desarrollados para automatizar la clasificación estructural, de los cuales destacan:

- En el año 2010 usando una representación de dominios de proteínas comprendida por descriptores de estructura y secuencia, Jain *et al.* [22] entrenaron modelos de bosques aleatorios sobre subconjuntos de datos de *SCOP* (versiones 1.69 y 1.73), obteniendo puntajes de precisión muy alentadoras sobre los diferentes niveles de clasificación definidos en *SCOP*. Con lo cual mostró el gran potencial de la clasificación estructural automática.
- Más adelante, en el año 2015, Corral-Corral *et al.* [13] usando modelos de Bosques Extremadamente Aleatorios y una representación de estructuras basada en el sistema *RCC*, reportó los puntajes de validación cruzada más altos hasta ese momento en la tarea de clasificación estructural sobre los niveles clase, pliegue, superfamilia y familia de *SCOP* versión 1.75B. Asimismo, reportó puntajes de validación cruzada en clasificación estructural para los niveles C, CA y CAT de *CATH* versión 3.5.

3.2. Obtención de vecinos estructurales

La necesidad de buscar estructuras similares de una nueva proteína en bases de datos como las definidas por *CATH* y *SCOP*, motiva el problema de obtención de vecinos estructurales descrito por Xuefeng Cui *et al.* [14]:

> "Dada una estructura de proteína q y una base de datos de estructuras de proteínas D, encontrar todas las estructuras en D que son similares a q."

Según Budowski-Tal *et al.* [8], el método de obtención de vecinos estructurales comúnmente propuesto para resolver este problema, es el de cuantificar la similitud entre todos los posibles pares de estructuras de proteínas en D usando un alineamiento estructural. De acuerdo a Jinbo Xu *et al.* [47] el tiempo de ejecución que toma alinear un par de proteínas toma desde 10 minutos hasta 1 hora, por lo que realizar un alineamiento estructural entre una proteína y todas las demás de alguna gran base de datos como el *PDB* es una tarea demasiado costosa. Y así el obtener rápidamente vecinos estructurales de una proteína se ha convertido en un reto.

3.2.1. *FragBag*

En respuesta a esta limitante, Budowski-Tal *et al.* [8] en 2010 propusieron *FragBag*, un método donde es posible representar estructuras de proteínas como vectores. Cada componente en este vector corresponde al número de ocurrencias de un fragmento de secuencia de proteína contigua particular. Usando esta representación midieron similitud entre estructuras de proteínas basados en la similitud entre sus correspondientes vectores. De esta forma la obtención de vecinos estructurales puede realizarse eficientemente. Sin embargo, la precisión de este método cae cuando dos estructuras son similares en varios fragmentos de secuencia locales pero difieren significativamente en su estructura general.

3.2.2. *ContactLib*

Xuefeng Cui *et al.* [14] para resolver este problema, en 2014, desarrollaron *ContactLib*, una biblioteca que permite identificar grupos de contacto de estructuras de proteínas. Un grupo de contacto se define como un conjunto de residuos representados por sus respectivos átomos C_α, tal que todos los átomos C_α se encuentran localizados dentro de una esfera de radio r. El número de grupos de contactos similares que comparten dos estructuras de proteínas es usado como indicador de similitud estructural. Cada grupo de contacto es representado por un vector, estos vectores luego son indexados y usados en una búsqueda basada en una tasa de aciertos. Hasta hace poco *ContactLib* había reportado los mejores resultados en la solución de la tarea de vecinos estructurales.

3.2.3. *RCC* y bosques extremadamente aleatorios

En 2015 Corral-Corral *et al.* [13] usando la separabilidad del espacio de plegados basado en *RCC*, superaron los resultados reportados por *FragBag*, en la tarea de obtención de vecinos estructurales valiéndose únicamente de la distancia euclidiana entre las estructuras en dicho espacio. Sin embargo, usando la misma representación y un enfoque de aprendizaje supervisado lograron superar en términos generales los puntajes reportados por *ContactLib*. El enfoque usado consistió en aprender la clasificación estructural del nivel Superfamilia de *SCOP* con un modelo compuesto por 250 Bosques Extremadamente Aleatorios, y para toda estructura q que se desease encontrar sus vecinos estructurales, se predijo su clase y se ordenó el resto de las estructuras con base a las log-probabilidades de pertenecer a dicha clase.

Capítulo 4

Aprendizaje profundo

En este capítulo las técnicas de Aprendizaje Profundo empleadas en este proyecto son estudiadas. Aprendizaje Profundo (*Deep Learning* en inglés) es un tipo particular de aprendizaje de máquina, el cual consiste en usar arquitecturas de redes neuronales profundas en la solución de problemas. Dichas arquitecturas tienen como finalidad encontrar, dentro de un espacio predefinido de posibilidades (espacio de hipótesis), representaciones de datos abstractas y útiles.

4.1. Redes neuronales

Las redes neuronales son estructuras de procesamiento de información. Una de sus propiedades más importante es su habilidad de aprender de los datos. Éstas son llamadas neuronales debido a que son modelos que fueron inspirados por la estructura neuronal del cerebro. Una red neuronal está compuesta por varias capas computacionales que procesan datos, cada capa está compuesta por múltiples unidades que actúan en paralelo. Cada neurona recibe una entrada desde muchas otras neuronas y calcula su valor de activación.

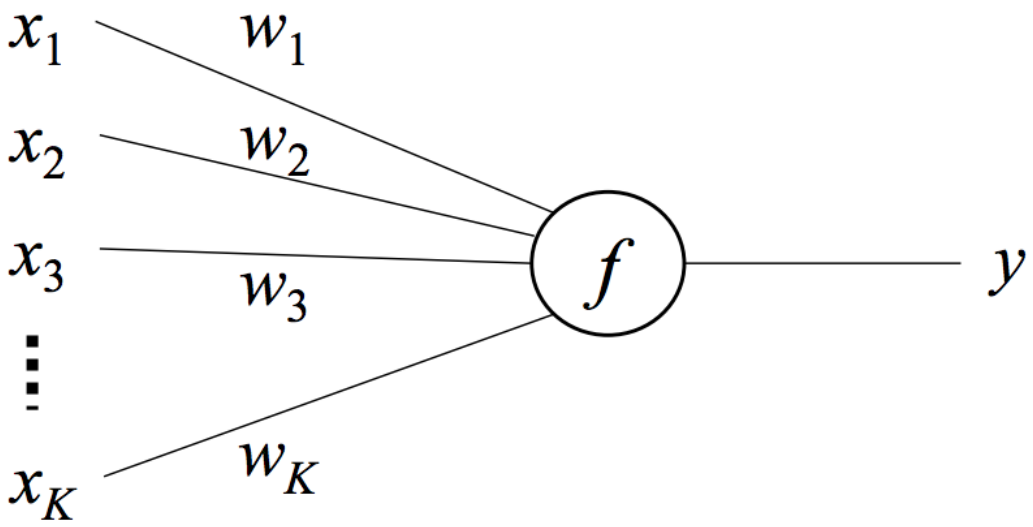

Figura 4.1: Unidad. Recuperado de [31]

En general, las unidades están diseñadas para tomar un conjunto $\boldsymbol{x}$ de n valores de entrada $(x_1, x_2, \ldots, x_n)$, aprender un conjunto $\boldsymbol{W}$ de pesos $(w_1, w_2, \ldots, w_n)$ y producir una salida $\boldsymbol{y}$.

4.2. Arquitecturas de redes neuronales

Aunque una neurona puede resolver varios tipos de tareas de aprendizaje, el poder de las redes neuronales viene cuando varias neuronas son conectadas en una arquitectura. La arquitectura de una red neuronal define la estructura general de la red, es decir la manera en que sus unidades están conectadas entre sí. La profundidad de una red se refiere al número de capas ocultas que componen a la red neuronal, mientras que la anchura determina el número de unidades que compondrá cada capa oculta.

De acuerdo con François Chollet [11], la arquitectura de una red restringe el espacio sobre el cual se buscan las representaciones de datos, a una serie específica de operaciones que mapean datos de entrada a datos de salida. Lo cual resulta en la búsqueda de un buen conjunto de valores para los pesos $\boldsymbol{W}$ involucrados en dichas operaciones.

En esta sección dos tipos importantes de arquitecturas son presentados.

4.2.1. Redes *feedforward*

Las redes *feedforward* son redes neuronales compuestas por una capa de entrada, una o varias capas intermedias y una de salida. Las capas intermedias también son conocidas como capas ocultas.

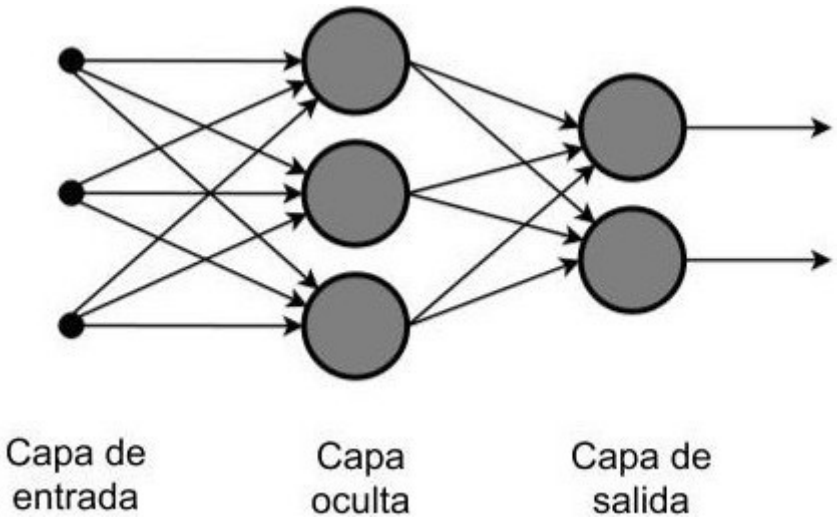

Figura 4.2: Red Neuronal Feedforward. Recuperado de [2]

En estas redes la información fluye en una sola dirección: desde la capa de entrada, a través de las capas ocultas y finalmente por la capa de salida; sus conexiones son no cíclicas. Las redes *feedforward* procesan datos de manera jerárquica, es decir la salida de una capa se convierte en la entrada de la siguiente.

La capa de entrada únicamente representa a los datos de entrada, por lo que las unidades pertenecientes a esta capa no tienen entradas pero sí generan una salida la cual es lineal. La salida de las unidades pertenecientes a las demás capas generalmente es calculada como una función no lineal (f) de una transformación afín de sus valores de entrada:

$$H(\boldsymbol{x},\ \boldsymbol{W}, \boldsymbol{b}) = f(\boldsymbol{W}^T \cdot \boldsymbol{x} + \boldsymbol{b})$$

Por simplicidad el parámetro de intersección $\boldsymbol{b}$ (*bias* en inglés) no se define explicitamente. Aumentando una entrada a cada uno de los parámetro $\boldsymbol{x}$ y $\boldsymbol{W}$, y estableciendo siempre en 1 la entrada extra en $\boldsymbol{x}$ $(x_1, x_2, \ldots, x_n, 1)$, la entrada extra en $\boldsymbol{W}$ jugará el papel del parámetro $\boldsymbol{b}$ $(w_1, w_2, \ldots, w_n, b)$:

$$H(\boldsymbol{x}, \boldsymbol{W}) = f(\boldsymbol{W}^T \cdot \boldsymbol{x})$$

De acuerdo con Geoffrey Hinton si una red neuronal está formada por más de una capa de oculta, se dice que es una red profunda [19]. Las redes neuronales profundas son capaces de comportamientos significativamente más complejos que sus contrapartes no profundas. Para más información acerca de las redes *feedforward* véase el capítulo 6 [16].

4.2.2. Redes *highway*

Propuestas por Srivastava *et al.* [44], las redes *highway* son una modificación a las redes *feedforward* profundas que facilitan el flujo de información entre capas mediante un mecanismo adaptativo de compuertas que permite caminos (*highways* en inglés) por los que la información puede fluir a través de muchas capas sin atenuación. A diferencia de una red *feedforward*, una red *highway* define dos transformaciones no lineales adicionales $T(\boldsymbol{x}, \boldsymbol{W}_T)$ y $C(\boldsymbol{x}, \boldsymbol{W}_C)$, T y C por sus siglas en inglés de *transfer gate* y *carry gate* respectivamente, tal que:

$$\boldsymbol{y} = H(\boldsymbol{x}, \boldsymbol{W}_H) \cdot T(\boldsymbol{x}, \boldsymbol{W}_T) + \boldsymbol{x} \cdot C(\boldsymbol{x}, \boldsymbol{W}_C)$$

donde T determina la magnitud de $\boldsymbol{y}$ que es producida por la transformación H de la entrada $\boldsymbol{x}$ y C determina la magnitud de la entrada cruda que fluye a la siguiente capa. Por simplicidad en este trabajo se estable $C = 1 - T$ tal que:

$$\boldsymbol{y} = H(\boldsymbol{x}, \boldsymbol{W}_H) \cdot T(\boldsymbol{x}, \boldsymbol{W}_T) + \boldsymbol{x} \cdot (1 - T(\boldsymbol{x}, \boldsymbol{W}_T))$$

y T es definida como:

$$T(\boldsymbol{x}, \boldsymbol{W}_T) = \sigma(\boldsymbol{W}_T^T \cdot \boldsymbol{x} + \boldsymbol{b}_T)$$

Es importante notar que la dimensionalidad de $\boldsymbol{x}$, $\boldsymbol{y}$, $H(\boldsymbol{x}, \boldsymbol{W}_H)$ y $T(\boldsymbol{x}, \boldsymbol{W}_T)$ es la misma.

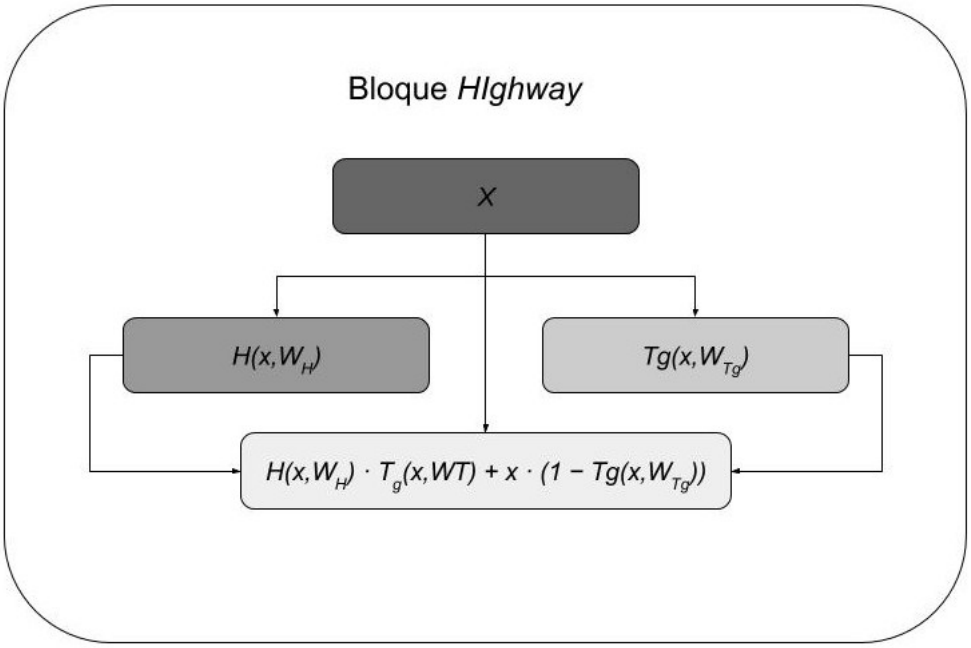

Figura 4.3: Representación capa *Highway*

4.3. Entrenamiento de una red neuronal

El objetivo de una red neuronal es aproximar alguna función g. Sea g' una red neuronal que aproxima g, g' está definida por:

$$g(\boldsymbol{x}) \approx g'(\boldsymbol{x};\boldsymbol{\theta})$$

donde $\boldsymbol{\theta}$ denota la colección de parámetros de la red conformada por los pesos $\boldsymbol{W}$ pertenecientes a las unidades. El entrenamiento de una red neuronal consiste en mostrarle datos a la red para calcular iterativamente las diferencias entre los valores producidos por $g'(\boldsymbol{x};\boldsymbol{\theta})$ con los valores reales $g(\boldsymbol{x})$, esto con el fin de modificar $\boldsymbol{\theta}$ basado en las diferencias y en cada paso disminuir éstas. Las diferencias son calculadas usando alguna función $J(\boldsymbol{\theta})$ conocida como función de costos. En cada paso del entrenamiento los parámetros $\boldsymbol{\theta}$ de la red reciben actualizaciones usando el algoritmo de propagación de errores hacia atrás [40] de acuerdo al gradiente de la función de costos y los valores de $\boldsymbol{\theta}$ en ese paso. La idea general es comenzar con suposiciones iniciales para $\boldsymbol{\theta}$, después en cada paso del entrenamiento, $\boldsymbol{\theta}$ se actualizará con la siguiente regla:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \frac{\delta}{\delta \boldsymbol{\theta}} J(\boldsymbol{\theta})$$

donde α es un valor ajustable llamado tasa de aprendizaje, el cual controla que tan grande es el cambio sobre $\boldsymbol{\theta}$. La finalidad del entrenamiento es reducir $J(\boldsymbol{\theta})$ hasta un mínimo local o global, con el propósito de aprender los valores $\boldsymbol{\theta}$ que resulten en la mejor aproximación a g. A la tarea de minimizar $J(\boldsymbol{\theta})$ la llamamos optimizar y en general las redes neuronales son entrenadas usando optimizadores basados en descenso por el gradiente [28].

Para más detalles del entrenamiento de una red neuronal, véase el capítulo 6 de [16].

4.4. Conceptos básicos de una red neuronal

4.4.1. Tipos de unidades

Las unidades de una red neuronal son definidas por la función de activación f que le aplican a la transformación afín de sus valores de entrada, la elección de las funciones de activación usadas en una red es fundamental, ya que generalmente introducen las no linealidades. Los tipos de unidades mayormente usados son:

- **Lineales** $f(x) = x$
  Estas son las unidades más simples, son fáciles de calcular pero tienen serias limitaciones, ya que una red neuronal compuesta únicamente por unidades lineales sólo calculará combinaciones lineales de sus valores de entrada. Y ésto resulta en un problema, ya que las unidades ocultas ayudan a la red a aprender relaciones complejas, por lo que es necesario usar algún tipo de no linealidad.

- ***ReLu*** $f(x) = max(0, x)$
  Unidades lineales rectificadas (*ReLu* por sus siglas en inglés) son la elección por defecto ya que son muy fáciles de optimizar, debido a que son muy similares a las unidades lineales. Simplemente son un umbral en el cero.

- ***Sigmoid*** $f(x) = 1/(1 + e^{-x})$
  Antes de la introducción de las unidades lineales rectificadas, la elección por defecto en la mayoría de las redes neuronales eran las unidades *sigmoid*. Intuitivamente en estas unidades cuando la magnitud de su entrada es muy pequeña, la salida es cercana a 0, cuando su entrada es muy grande la salida es muy cercana a 1.

- ***Tanh*** $f(x) = tanh(x)$
  Las unidades tangente hiperbólicas (*tanh* por sus siglas en inglés) usan un tipo de no linealidad similar a las *sigmoid* pero en lugar de tener un rango de $[0, 1]$, su salida se encuentra en el intervalo $[-1, 1]$. En la práctica las unidades *tanh* son preferidas a las *sigmoid*.

- ***Softsign*** $f(x) = \frac{x}{1+|x|}$
  Las unidades *softsign* han surgido como una alternativa a las unidades *tanh*.

- ***Softplus*** $f(x) = log(e^x + 1)$
  Las unidades *softplus* son una aproximación de las unidades *ReLu*.

- ***Softmax*** $f(z_i) = \frac{e_i^z}{\sum_j e_j^z}$
  Las unidades *softmax* representan la distribución de probabilidad sobre n diferentes clases. A diferencia de los tipos de unidades vistos anteriormente, la salida de las unidades *softmax* depende de las salidas de todas las otras unidades pertenecientes a la misma capa, esto debido a que las unidades *softmax* requieren que la suma de todas las salidas de la capa sea igual a 1. Las unidades *softmax* son frecuentemente usadas en la capa de salida para la tarea de

clasificación multiclase, es decir cuando la tarea de clasificación involucra más de dos clases.

Unidades de salida

Las unidades de salida son las unidades pertenecientes a la capa de salida. Al igual que las unidades ocultas son definidas por la función de activación que le aplican a sus valores de entrada. En principio cualquier tipo de unidad puede ser usado como unidad de salida, pero generalmente el tipo de la unidad de salida está directamente influenciado por la elección de la función de costos y el tipo de tarea a resolver. Las unidades de salida proveen alguna transformación adicional sobre los datos que fluyen en la red para completar la tarea que la red debe desempeñar. Para más información acerca de las unidades de salida véase el capitulo 6 de [16].

4.4.2. No linealidades

De acuerdo con François Chollet [11], una red neuronal sin no linealidades solo puede aprender transformaciones lineales de los datos de entrada, es decir el espacio de hipótesis sobre el cual buscará representaciones útiles y abstractas para los datos de entrada será el conjunto de todas de todas las posibles transformaciones lineales sobre los datos de entrada. Siendo dicho espacio muy restrictivo, un modelo de múltiples capas sin no linealidades no se beneficiaría de sus múltiples capas ya que la pila de capas lineales seguiría siendo una operación lineal sobre los datos de entrada, es decir el añadir más capas no extenderá el espacio de hipótesis. Por lo que las redes neuronales profundas que contienen múltiples capas ocultas con unidades no lineales son modelos muy expresivos que pueden aprender relaciones complejas entre sus entradas y salidas.

4.4.3. Inicialización de parámetros

Anteriormente se mencionó que al inicio del entrenamiento se suponen valores para $\boldsymbol{\theta}$. Según Goodfellow *et al.*, [16] el entendimiento que se tiene de cómo los valores iniciales influyen en el entrenamiento es mínimo. Por lo que se podría pensar que la manera trivial de abordar este requerimiento es inicializar todos los parámetros a cero y esperar que el algoritmo de optimización modifique los valores durante el entrenamiento buscando la mejor configuración de estos. Pero lo anterior resulta en un error debido a que las unidades que tienen la misma función de activación y reciben valores de entrada de las mismas unidades, calcularán la misma salida, luego calcularán el mismo gradiente durante propagación hacia atrás y por tanto se someterán a exactamente la misma actualización, lo que podría derivar en que la red se quede estancada en algún punto silla. Por lo anterior se requiere que los parámetros iniciales rompan simetría.

Usualmente es mejor inicializar cada unidad para calcular una salida diferente de todas las demás unidades, lo que motiva la inicialización aleatoria de los parámetros $\boldsymbol{\theta}$. Además la inicialización aleatoria usando una distribución con alta entropía es computacionalmente barata y difícilmente asignará un mismo valor a diferentes unidades con lo que evitaremos calcular una misma salida.

En general el entrenamiento de redes neuronales es fuertemente afectado por la inicialización de los parámetros. Además la suposición inicial de $\boldsymbol{\theta}$ puede determinar

si el algoritmo convergerá y que tan rápido lo hará, por otro lado con algunas suposiciones iniciales el algoritmo puede encontrar dificultades numéricas y fallar por completo.

Típicamente los parámetros de intersección $\boldsymbol{b}$ son constantes heurísticamente escogidas y solo los pesos $\boldsymbol{W}$ son inicializados aleatoriamente. Casi siempre los pesos son valores aleatorios tomados de una distribución gaussiana (normal) o uniforme. Aunque en la práctica ha sido indistinta la elección entre dichas distribuciones, la escala de la distribución ha tenido gran efecto en la habilidad de la red para generalizar.

Algunas heurísticas disponibles para la elección de la escala inicial de pesos son:

- Escalar por la raíz cuadrada del número de entradas m, algunas veces es multiplicada una constante k, usada en [18], [24] y [27]:

$$\sqrt{\frac{k}{m}}$$

- Escalar por la raíz cuadrada de la suma del número de entradas m y el número de salidas n, algunas veces es multiplicada una constante k, usada en [15]:

$$\sqrt{\frac{k}{m+n}}$$

4.5. Clasificación con redes neuronales

En aprendizaje de máquina la tarea de clasificación comprende el uso de los datos disponibles para aprender una función que asigna una categoría a una entrada [29]. Dicha función es descrita por la ecuación:

$$y_i = g(x_i)$$

Donde:

- x_i es alguna entrada
- $\{y_1, y_2, \ldots, y_n\}$ es el conjunto de categorías a las que x_i puede pertenecer.
- Generalmente no se sabe como $\boldsymbol{x}$ y $\boldsymbol{y}$ están matemáticamente relacionadas.
- Se asume que la función g existe.

Entonces una red neuronal g' que aproxima g está definida por:

$$y_i = g'(x_i; \boldsymbol{\theta})$$

Para resolver la tarea de clasificación el objetivo principal es el de diseñar una red neuronal g' y encontrar sus respectivos parámetros $\boldsymbol{\theta}$ tal que minimicen las diferencias con g.

4.6. Regularización de una red neuronal

Las redes neuronales con muchos parámetros pueden dar solución a una gran cantidad de tareas, pero aunque se desempeñen bien con los datos disponibles de alguna tarea en especifico, no quiere decir que sean un buen modelo para dicha tarea. Comúnmente existe tal grado de libertad en el modelo que puede describir cualquier tipo de conjunto de datos de cualquier tamaño, es decir memorizarlo, sin capturar alguna visión genuina del fenómeno subyacente. Cuando lo anterior sucede, el modelo trabajará bien con los datos usados para entrenarlo pero fallará generalizando ante nuevas situaciones (nuevos datos). Este fenómeno es mejor conocido como sobreajuste. Cuando un modelo sufre de sobreajuste se debe a que está ajustado a características muy específicas de los datos usados para entrenarlo y que no tienen relación causal con la función objetivo. Por lo que la verdadera prueba de un modelo es su habilidad de hacer predicciones sobre situaciones a las que no ha sido expuesto anteriormente.

La regularización de una red neuronal se define como cualquier modificación que se haga sobre esta, la cual intente reducir el error ante nuevas situaciones, el error de generalización, pero no sobre los datos usados para entrenarla. Existen múltiples métodos de regularización. Algunos añaden restricciones sobre los valores de los parámetros de la red otros añaden términos extra sobre la función de costos.

4.6.1. Detención temprana

Durante el entrenamiento de una red neuronal los parámetros $\boldsymbol{\theta}$ del modelo se actualizarán iterativamente buscando la mejor configuración de estos para la solución de alguna tarea, sin embargo llegado a cierto punto el modelo dejará de mejorar ante situaciones nuevas (fuera del conjunto entrenamiento) y pasado ese punto la capacidad de generalización del modelo disminuirá. La detención temprana es un método de regularización usado para evitar el sobreajuste de modelos de aprendizaje de máquina durante el entrenamiento ya que provee un mecanismo para saber cuántas iteraciones se pueden ejecutar antes de que el modelo sea sobreentrenado.

En la práctica, la detención temprana es usada al tomar un pequeño subconjunto $\mathbb{V}$ del conjunto original de datos entrenamiento y no usarlo durante el entrenamiento. La idea es correr el algoritmo de optimización hasta que el error sobre un conjunto $\mathbb{V}$ haya dejado de mejorar para alguna cantidad de tiempo t. El error sobre el conjunto $\mathbb{V}$ es usado como una representación del error de generalización, con lo que podemos determinar cuando el sobreajuste ha comenzado. Para más detalles acerca de detención temprana véase el capítulo 7.8 de [16].

4.6.2. *Dropout*

El método *dropout* [43] provee un método de regularización computacionalmente barato pero muy poderoso. La idea central detrás de *dropout* es apagar aleatoriamente durante el entrenamiento unidades de la red neuronal. Por apagar una unidad, se entiende el temporalmente removerla de la red, así como sus conexiones de entrada y de salida. Esto evita que las unidades se adapten demasiado. La elección de qué unidad apagar es aleatoria. En la práctica cada unidad es retenida con cierta probabilidad p independiente de las otras unidades. Aplicar *dropout* a una red neuronal se puede pensar como tomar una muestra reducida de si misma, la red reducida

consiste de todas las unidades que sobrevivieron al *dropout*. Redes neuronales con *dropout* se entrenan de manera similar a las redes estándar, la única diferencia es que para cada ejemplo en el entrenamiento, este se entrena sobre una red reducida de la red original. Esto significativamente reduce el sobreajuste y nos da mejoras sobre otros métodos de regularización.

4.6.3. Ensamble de modelos

Es una técnica para reducir el error de generalización la cual consiste en entrenar individualmente varios modelos para después de alguna manera combinar las predicciones de estos. El método de ensamble mayormente usado es la agregación *bootstrap*, la cual consiste en entrenar varios modelos diferentes de manera separada, para después combinarlos ya sea promediando el resultado en el caso de regresión o votando para clasificación. Para más detalles acerca del ensamble de modelos o la agregación *bootstrap* véase el capítulo 7.11 de [16].

4.7. Hiperparámetros

Los modelos de aprendizaje profundo tienen muchos hiperparámetros los cuales controlan diferentes aspectos del comportamiento del modelo. Es importante entender las diferencia entre los parámetros del modelo y los hiperparámetros. Los parámetros del modelo son aquellos valores $\boldsymbol{\theta}$ que una red neuronal aprende durante el entrenamiento; mientras que los hiperparámetros son aquellos valores que se ajustan, es decir no son adaptados por el algoritmo de aprendizaje en sí; ejemplo de hiperparámetros son: el valor para α (la tasa de aprendizaje), los tipos de unidades ocultas, la distribución elegida para inicializar los parámetros, la probabilidad p de *dropout* usada en cada capa, etc.

Los modelos de aprendizaje profundo pueden desempeñarse bien con solo ajustar pocos hiperparámetros pero con frecuencia se benefician significativamente de ajustar múltiples hiperparámetros. De acuerdo con Goodfellow *et al.* [16] los algoritmos de aprendizaje profundo dependen de un buen ajuste de hiperparámetros para alcanzar su máxima efectividad. En general existen dos enfoques a la hora de ajustar los hiperparámetros de un modelo: seleccionar los valores a mano u optimizarlos automáticamente.

Ajustar los hiperparámetros manualmente puede funcionar muy bien cuando el usuario tiene un buen punto de inicio tal como alguno determinado por otros que hayan trabajado sobre el mismo tipo de aplicación y arquitectura o cuando el usuario tiene mucha experiencia en el uso de modelos de aprendizaje profundo en tareas similares. Sin embargo para muchas aplicaciones este tipo de puntos iniciales no siempre están disponibles, en estos casos la optimización de hiperparámetros puede encontrar valores útiles. La optimización de hiperparámetros es el problema de encontrar en el espacio de hiperparámetros la configuración que maximiza el desempeño de un modelo sobre un conjunto de datos. El espacio de hiperparámetros representa todas las posibles configuraciones o valores que pueden adquirir los hiperparámetros de un modelo, la dimensión de este espacio depende de cómo el experimentador parametrice el modelo y de qué valores escoja para tener valores fijos.

4.7.1. Optimización automática de hiperparámetros

Debido a los avances computacionales, actualmente es posible evaluar múltiples intentos de optimización, lo que ha permitido la viabilidad en usar enfoques automáticos para encontrar mejores resultados. La manera natural de desarrollar algoritmos de optimización de hiperparámetros es que envuelvan al algoritmo de aprendizaje y escojan sus hiperparámetros. Los algoritmos de optimización de hiperparámetros no solo deben optimizar sobre los valores de los hiperparámetros, sino también simultáneamente deben decidir qué hiperparámetros optimizar. A diferencia del entrenamiento de una red neuronal, el mapeo entre la configuración de hiperparámetros y el desempeño de la red sobre un conjunto de datos no siempre puede ser descrito por una fórmula, por lo que el gradiente no siempre estará disponible y las técnicas de optimización como descenso por el gradiente no podrán ser aplicadas.

Grid search

Grid Search es un algoritmo que sugiere configuraciones de hiperparámetros al buscar exhaustivamente sobre todo el espacio de hiperparámetros definido. *Grid Search* entrena un modelo para cada especificación de hiperparámetros existente en el producto cartesiano de los conjuntos de posibles valores de cada hiperparámetro. Al final el modelo que tenga la mejor generalización es escogido por haber encontrado la mejor configuración de hiperparámetros. El problema con *Grid Search* radica en que el costo computacional de éste crece exponencialmente con el número de hiperparámetros, por lo que en la práctica comúnmente es usado cuando hay pocos hiperparámetros que optimizar.

Random search

Random Search es una alternativa a *Grid Search* la cual generalmente converge a buenos valores de hiperparámetros a un menor costo y de manera más rápida. A diferencia de *Grid Search*, *Random Search* sugiere configuraciones aleatorias de hiperparámetros del espacio de hiperparámetros definido. Bergstra *et al.* [3] mostraron que *Random Search* encuentra en menos evaluaciones configuraciones con resultados similares que *Grid Search*, solo que no es claro cuántas evaluaciones son necesarias para un problema en particular.

Optimización secuencial basada en modelo

Para compensar la falta de gradiente y mitigar el costo de evaluar la función que mide el desempeño de los hiperparámetros (función de aptitud) ya que el evaluarla puede implicar grandes cantidades de cómputo y memoria, la Optimización Secuencial Basada en Modelo (*SMBO* por sus siglas en inglés) pretende construir modelos que aproximan la función de aptitud basados en un historial de observaciones, y después escoger hiperparámetros para evaluar.

La idea general de estos algoritmos es, en un ciclo fuera del entrenamiento de una red neuronal, optimizar una función sustituta de la función de aptitud, dicha función sustituta será una aproximación más barata de evaluar de la función de aptitud; y el punto que maximice la función sustituta se convertirá en una propuesta

de configuración de hiperparámetros, sólo las propuestas más prometedoras serán evaluadas en la función original.

Existen diversos enfoques de *SMBO* los cuales difieren en el criterio por el cual optimizan la función sustituta y en la manera en que construyen la función sustituta dado el historial de observaciones:

- **Optimización bayesiana:** El enfoque de optimización bayesiana se centra en un modelo de probabilidad $P(puntaje|configuracion)$ para modelar la función sustituta, el cual se obtiene al actualizar una probabilidad anterior de un historial de parejas $(configuracion, puntaje)$. En este enfoque comúnmente procesos de Gauss son usados como modelo de probabilidad y típicamente este enfoque optimiza el criterio de Mejora Esperada (*Expected Improvement* en inglés) el cual es la probabilidad esperada de que nuevas propuestas mejoren la mejor configuración de hiperparámetros actual.

- **Estimador [árbol-estructurado parzen:** A diferencia de la optimización bayesiana, el enfoque de estimador árbol-estructurado parzen (*TPE* por sus siglas en inglés) usa dos modelos separados: $P(configuracion|puntaje)$ y $P(puntaje)$, para modelar $P(puntaje|configuracion)$. Aquí generalmente se define el espacio de hiperparámetros como un espacio árbol-estructurado, donde las hojas (por ejemplo: el número de unidades para una capa oculta) sólo son definidas cuando los nodos (por ejemplo: el número de capas ocultas a usar) toman valores particulares. Estos modelos separados son modelados usando densidades no paramétricas, los nuevos puntos serán muestreados de la región que se considere óptima (criterio de Mejora Esperada).

Para más información acerca de la optimización secuencial basada en modelo véase [6] y [4].

Capítulo 5

Enfoque

En este capítulo se describen las técnicas usadas para la solución de las tareas de clasificación estructural automática y búsqueda de vecinos estructurales con modelos de aprendizaje profundo y el sistema de *RCC*. De igual forma se describe el proceso usado para el diseño de los modelos de aprendizaje profundo. Asimismo son analizados los retos presentes en la selección y evaluación de los modelos de aprendizaje de profundo. También se especifican los detalles de los conjuntos de datos usados, lo cuales fueron proporcionados por los autores de Clases de Cúmulos de Residuos [13].

5.1. Clasificación estructural automática

El objetivo en esta tarea es el de reproducir las clasificaciones asignadas a los dominios reportados por *CATH* en los niveles de clase, arquitectura y topología. Así que se aprendieron las clasificaciones de los dominios reportados sobre el espacio de plegados basado en el sistema *RCC*. El aprendizaje se realizó mediante el entrenamiento de clasificadores basados en modelos de redes neuronales profundas.

5.1.1. Conjunto de datos

Para esta tarea se usaron los 235,858 dominios reportados en *CATH* versión 4.0. Donde éstos se encuentran clasificados en: 4 clases, 40 arquitecturas, 1,375 topologías y 2,738 superfamilias homólogas. A continuación se muestra un fragmento de este conjunto de datos:

```
# Dominio,              Vector de 26 posiciones,              Clasificacion
1oaiA00,0,0,2,17,27,0,5,0,0,1,2,2,0,8,11,0,0,0,0,0,0,0,0,0,0,16,1_10_8_10
1iu5A00,0,0,1,27,8,5,21,2,0,2,8,3,1,9,6,0,0,0,0,0,1,0,0,1,0,0,2_20_28_10
1mynA00,0,0,1,2,7,0,2,1,2,2,9,1,2,8,2,0,1,1,0,0,5,0,1,0,3,10,3_30_30_10
125d000,2,1,8,11,4,1,13,3,0,0,4,0,0,4,1,0,0,0,0,0,0,0,0,0,0,5,4_10_240_10
...
```

Aquí cada renglón describe a un dominio representado por: su nombre, un vector de 26 posiciones y una etiqueta de clasificación de acuerdo a *CATH*. El vector de 26 posiciones denota el número de ocurrencias de cada clase definida en el sistema de *RCC* y en el cual se consideró una distancia de 5 Å en la definición de los cúmulos

de residuos. La etiqueta especifica cada nivel jerárquico de clasificación de acuerdo a *CATH* de la siguiente manera:

- *c*: especifica el nivel clase.
- *c_a*: especifica el nivel arquitectura.
- *c_a_t*: especifica el nivel topología.
- *c_a_t_h*: especifica el nivel superfamilia homologa.

5.2. Búsqueda de vecinos estructurales

Antes que nada varias notas son dadas para la mejor explicación del enfoque usado:

- **Nota 1:** Se dice que dos estructuras de proteínas se encuentran en la vecindad *SAS* X, si la mejor superposición entre los átomos de éstas resulta en un Puntaje de Alineamiento Estructural (*SAS* por sus siglas en inglés) menor a X Ångströms.
- **Nota 2:** Dos estructuras de proteínas son consideradas como vecinas si se encuentran en la misma vecindad *SAS* y ambas pertenecen a la misma superfamilia, de acuerdo a la definición de *SCOP* [14].
- **Nota 3:** En el espacio de plegados basado en el sistema *RCC*, la proximidad entre dos estructuras de proteínas no siempre implica que éstas sean vecinas [13].

Ya que dos estructuras son consideras vecinas si, además de encontrarse en la misma vecindad *SAS*, ambas pertenecen a la misma superfamilia, se aprendieron las superfamilias definidas por *SCOP* sobre un espacio basado en el sistema *RCC*. Y una vez que se desea encontrar a los vecinos de una estructura q en una base de datos D, se predice la superfamilia de la estructura q y se ordenan las estructuras en D con base en su probabilidad de pertenencia a la superfamilia predicha, para más detalles véase el algoritmo 1.

Algoritmo 1: Búsqueda de vecinos estructurales

```
aux ← [] ;
superfamilia ← predice(q) ;
para p in D hacer
    proba ← predice_proba(p, superfamilia) ;
    aux.inserta((p, proba));
fin
vecinos ← ordena_probas(aux);
```

Al igual que en la clasificación estructural automática, el aprendizaje se realizó mediante el entrenamiento de clasificadores basados en modelos de redes neuronales profundas. El clasificador trabajará correctamente si los vecinos de q están hasta arriba en la lista ordenada.

5.2.1. Análisis *ROC*

Para evaluar el desempeño de un modelo de aprendizaje profundo en la identificación de vecinos estructurales se uso la curva Característica Operativa del Receptor o *ROC* por sus siglas en inglés. Típicamente usada sobre algoritmos de clasificación binaria donde los ejemplos a clasificar son divididos en positivos y negativos, la curva *ROC* es una gráfica de la proporción de verdaderos positivos sobre el eje y contra la proporción de falsos positivos sobre el eje x. Una curva *ROC* puede ser interpretada gráficamente así como numéricamente.

El desempeño de un clasificador para correctamente identificar positivos (en este caso vecinos) y negativos (no vecinos) es medido por el área bajo la curva *ROC* (*AUC* o *AUROC* por sus siglas en ingles). Un área de 1 representa a un clasificador perfecto, mientras que un área de 0.5 se dice que corresponde a un clasificador aleatorio. Áreas bajo la curva mayores a 0.5 representan buenos resultados de clasificación, es decir mejor que el azar, mientras que áreas menores a 0.5 se consideran peores que el azar. [42]

5.2.2. Conjunto de datos

Para esta tarea, este proyecto usa los subconjuntos *SCOP30* y *SCOPtrain1* de *SCOP* versión 1.75B. *SCOP30* contiene 3,290 estructuras de proteínas, mientras que *SCOPtrain1* es un muestreo aleatorio que contiene 136,300 estructuras. A continuación se muestra un fragmento de este conjunto de datos:

```
# Dominio,              Vector de 26 posiciones,              Clasificacion
d101ma_,16,1,1,44,86,0,12,5,0,3,4,0,7,7,63,0,0,0,0,0,0,1,0,1,0,31,a.1.1.2
d32c2a1,8,3,5,17,21,16,56,5,2,1,25,11,4,21,4,0,0,0,0,0,2,0,0,0,1,1,b.1.1.1
d1gu7a2,16,4,6,80,99,5,70,4,0,2,12,6,5,18,34,0,0,0,0,0,0,0,0,0,1,31,c.2.1.1
d2gu3a2,2,1,3,14,10,1,19,5,0,0,12,2,3,16,2,0,0,0,0,0,0,0,0,0,4,5,d.17.1.6
...
```

Aquí cada renglón describe a un dominio representado por: su nombre, un vector de 26 posiciones y una etiqueta de clasificación de acuerdo a *SCOP*. El vector de 26 posiciones denota el número de ocurrencias de cada clase definida en el sistema de *RCC* y en el cual se consideró una distancia de 5 Å en la definición de los cúmulos de residuos. La etiqueta especifica cada nivel jerárquico de clasificación de acuerdo a *SCOP* de la siguiente manera:

- c: especifica el nivel clase.
- $c.p$ especifica el nivel plegado.
- $c.p.s$: especifica el nivel superfamilia.
- $c.p.s.f$ especifica el nivel familia.

5.3. Diseño de las redes neuronales

Toda red neuronal tiene una capa de entrada, la cual debe tener por número de neuronas el número de características de los datos usados como entrada. Asimis-

mo, toda red neuronal tiene una capa de salida y su tamaño es determinado por la tarea que desea resolver, en el caso de una tarea de clasificación la capa de salida tendrá una neurona por cada posible categoría. Por tanto el reto de diseñar una arquitectura para la tarea de clasificación se centra en cómo definir las capas ocultas. Anteriormente se mencionó que el definir una buena arquitectura de red no es suficiente en la solución de un problema, muchas veces la red neuronal depende de hiperparámetros bien configurados para alcanzar su máxima eficacia. Es importante saber que no existen reglas definidas para escoger los valores de los hiperparámetros ya que es imposible predecir de antemano qué configuraciones funcionarán mejor para cada problema. Por lo que la experimentación es necesaria para llegar a una configuración ideal. Generalmente el proceso de selección consiste en prueba y error, intuyendo qué configuración puede trabajar bien y evaluando su desempeño.

5.3.1. Definición del espacio de hiperparámetros

Se parametrizaron diversos aspectos referentes a la arquitectura como el número de capas ocultas, el número de neuronas por capa y el uso de capas *highway*. Los valores correspondientes al número de neuronas por capas fueron sugeridos a partir de la experimentación manual. Debido a que las redes neuronales son diseñadas para resolver tareas clasificación, el tipo de las unidades de salida no fue parametrizado y la función *softmax* fue elegida para poder representar la salida de cada red como la distribución de probabilidad sobre las n diferentes categorías a clasificar. La manera natural de medir la distancia entre dos vectores de probabilidad es la entropía cruzada, por lo que ésta fue la elección como función de costos $J(\theta)$. La entropía cruzada mide las diferencias entre los valores reales de g y los valores calculados por g'.

Sea t_i los valores reales de g y sea y_i los valores calculados por g', la entropía cruzada E está definida por:

$$E = -\sum_i t_i \cdot log(y_i)$$

Donde E representa el error, t_i la repuesta correcta y y_i el valor computado.

En general descenso por el gradiente ha sido considerado lento, por lo que las redes neuronales en la práctica son entrenadas usando optimizadores basados en descenso por el gradiente. Para entrenar las redes neuronales este proyecto usa $ADAM$ como optimizador. ADAptive Moment Estimation ó $ADAM$ (por sus siglas en inglés) es un algoritmo de optimización adaptativo de la tasa de aprendizaje α [23].

5.3.2. Optimización de hiperparametros

Ajustar hiperparámetros es una tarea difícil debido a que es computacionalmente caro, muy raramente es tratable hacer una búsqueda manual y no es posible estimarlos directamente de los datos usados para entrenar el modelo. Además la intuición general de un humano para sugerir buenos parámetros es bastante limitada y generalmente es peor que hacer una búsqueda aleatoria sobre las posibles configuraciones de estos. Considerando lo anterior se decidió usar una estrategia semi automática en el proyecto.

Hyperopt

Para la optimización de hiperparametros el proyecto usó *hyperopt* [5]. *Hyperopt* es una biblioteca escrita en Python diseñada para optimizar sobre espacios de búsqueda complicados. Actualmente hyperopt provee dos algoritmos de optimización: *Random Search* y *TPE*. *Hyperopt* provee una interfaz de optimización en la cual se define el espacio de configuración y una función de evaluación. La manera de usar *hyperopt* es describiendo:

1. La función de evaluación, es decir la función objetivo a minimizar. En el caso de la optimización de hiperparametros de una red neuronal la función deberá entrenar a la red neuronal y regresar el valor de la métrica evaluada:

```
from hyperopt import STATUS_OK, fmin

def objective(params, **kwargs):
    # Aqui ocurre el entrenamiento de la red neuronal
    return {'loss': metric_score, 'status': STATUS_OK}
```

2. El espacio de configuraciones sobre el cual buscar (*Random Search* o *TPE*):

```
from hyperopt import hp

neurons = [16, 192, 512, 1024]
initializers = ['uniform','normal']
activations = ['relu', 'tanh', 'sigmoid']

space = {
        'hidden_1': hp.choice('hidden_1', neurons),
        'hidden_1_init': hp.choice('hidden_1_init',initializers),
        'dropout_1': hp.uniform('dropout_1', 0.0, 1.0),
        'hidden_2': hp.choice('hidden_2', neurons),
        'hidden_2_act': hp.choice('hidden_2_act', activations),
        'dropout_2': hp.uniform('dropout_2', 0.0, 1.0)
        }
```

3. El algoritmo de optimización a usar:

```
from hyperopt import tpe, rand

best = fmin(objective, space, algo=tpe.suggest, max_evals=100)
```

Para más información respecto a *hyperopt* véase a la documentación oficial de *hyperopt* [5].

5.3.3. Evaluación y selección del modelo

Modificar hiperparámetros cuando se entrena un modelo de aprendizaje profundo sobre un mismo conjunto de datos, resulta en diferentes modelos. Con el fin de poder comparar los modelos generados durante la optimización se eligió como métrica

cuantitativa de su desempeño la precisión. La precisión es definida como el número de predicciones correctas sobre el número total de ejemplos evaluados:

$$acc = \frac{VP}{NT}$$

Donde $NT = VP + FP$, VP representa el número de verdaderos positivos y FP el número de falsos positivos.

Para saber cómo se desempeña un modelo sobre datos nunca antes vistos, la evaluación de la precisión debe realizarse sobre datos diferentes a los de entrenamiento, de lo contrario esto puede producir un sobreajuste del modelo.

Típicamente el conjunto de datos disponibles para resolver una tarea se divide en tres partes: el conjunto de datos de entrenamiento, el conjunto de datos de prueba y el conjunto de datos de validación. El primero, es usado para entrenar al modelo, mientras que los datos de validación son usados para evaluar el desempeño del modelo en cada iteración de la optimización y, hasta que sea haga la elección del modelo con mejor desempeño sobre los datos de validación, se corrobora la capacidad de generalización del modelo seleccionado con el conjunto de datos de prueba.

Es todo un debate la elección de los tamaños de estos conjuntos de datos, ya que entre más grande sea cada uno, más precisas serán las evaluaciones sobre su desempeño. En este proyecto usando un simple proceso de submuestreo al azar, se asignó 10 % del total de datos disponibles al conjunto de datos de validación, 10 % al conjunto de datos de prueba y 80 % al conjunto de datos de entrenamiento.

Además el conjunto de datos de validación fue usado para realizar detención temprana, es decir para medir el error en cada iteración durante el entrenamiento. El algoritmo de optimización correrá hasta que el error sobre el conjunto de validación haya dejado de mejorar para alguna ventana de tiempo t, ($t = 10$ para $CATH$ y $t = 20$ para $SCOP$). Y así, el error sobre el conjunto de validación es usado como una representación del error de generalización, con lo que es posible determinar cuándo el sobreajuste ha comenzado.

Una vez escogidas las arquitecturas que mejor se desempeñaron en cada uno de los niveles de clasificación estructural respectivos a los conjuntos de datos, es decir de clase, arquitectura y topología para el conjunto de datos $CATH$ y superfamilia para el conjunto de datos $SCOP$, se entrenó cada arquitectura para pruebas de validación cruzada estratificada con 10 iteraciones, esto con el fin de evaluar la capacidad de cada modelo para aprender reglas generales sobre los conjuntos de datos.

Validación Cruzada

La idea detrás de validación cruzada es que todos los ejemplos del conjunto de datos disponible sean usados para la estimación de la métrica deseada, esto a expensas de un incremento en el costo computacional. Para ejemplificar validación cruzada es fácil imaginar que para evaluar un modelo sobre un conjunto de datos de tamaño 10, se entrena dicho modelo 10 veces, y en cada entrenamiento se deja fuera un elemento del conjunto de entrenamiento, este elemento diferente en cada iteración; y dicho elemento es usado para evaluar el modelo al final de cada iteración. Al final el promedio de los valores obtenidos en cada evaluación representará el desempeño global de modelo.

Validación cruzada con k iteraciones es un caso especial de validación cruzada donde se itera sobre el conjunto de datos k veces. Para cada iteración, el conjunto

de datos es dividido en k partes: una parte es usada para la evaluación del modelo, mientras que las $k-1$ restantes son usadas, como un solo conjunto, para el entrenamiento. La métrica deseada entonces es estimada al calcular el promedio de los valores evaluados a través de las k iteraciones. Para más información acerca de validación cruzada y validación cruzada con k iteraciones véase [37].

Estratificación

Generalmente para dividir el conjunto de datos disponibles en los subconjuntos necesarios, ya sea para los conjuntos de entrenamiento, validación y prueba o para los de validación cruzada con k iteraciones, se usa un simple proceso de muestreo aleatorio: aleatoriamente se elige cierto porcentaje del conjunto total para cada subconjunto. Y para este proceso de muestreo aleatorio se hace la suposición de que los subconjuntos obtenidos serán representativos, pero en la práctica no siempre es correcto ya que comúnmente los conjuntos de datos disponibles contienen clases desequilibradas, es decir habrá algunas clases con mayor número de representantes que otras; y con lo cual existe la posibilidad de que alguno de los conjuntos de datos pueda contener ningún representante de alguna clase minoritaria. Para atacar dicha problemática comúnmente se hace la división del conjunto de datos de manera estratificada, es decir aleatoriamente se divide el conjunto de datos buscando que las clases sean correctamente representadas en los diferentes subconjuntos.

5.3.4. Apilamiento de modelos jerárquicos

El apilamiento es una técnica de ensamble de modelos, la cual combina información de múltiples modelos de aprendizaje de máquina para generar un nuevo modelo. Generalmente el apilamiento consiste en un ensamble de dos capas, donde varios modelos individuales forman la primera capa y alimentan sus salidas (predicciones) al segundo nivel (típicamente otro modelo) el cual hace las predicciones finales. Para mas detalles acerca del apilamiento de modelos, véase [17] o el capítulo 7 de [38].

Buscando reducir el error de generalización sobre los diferentes niveles de clasificación en *CATH*, se desarrolló un método de apilamiento de modelos basado en la estructura jerárquica que establece la definición de *CATH*. Dicho método se denominó Apilamiento de Modelos Jerárquicos y consiste en que una vez diseñadas las arquitecturas correspondientes a los diferentes niveles de clasificación clase, arquitectura, topología, estas son apiladas en un ensamble de 3 niveles:

1. El primero consiste del modelo correspondiente al nivel de clase, éste es entrenado de manera normal y, una vez entrenado, sus parámetros $\boldsymbol{\theta}$ son congelados, es decir estos parámetros no podrán ser actualizados.

2. El segundo, se compone por el modelo diseñado para el nivel de clasificación arquitectura y, a diferencia del primero, éste es entrenado usando las predicciones del anterior, concatenadas con la entrada original. Al igual que para el primer nivel, una vez entrenado éste, sus parámetros son congelados.

3. Por último, el tercero consta del modelo correspondiente al nivel de topología. Durante el entrenamiento de éste, las predicciones de los niveles uno y dos son concatenadas con la entrada original.

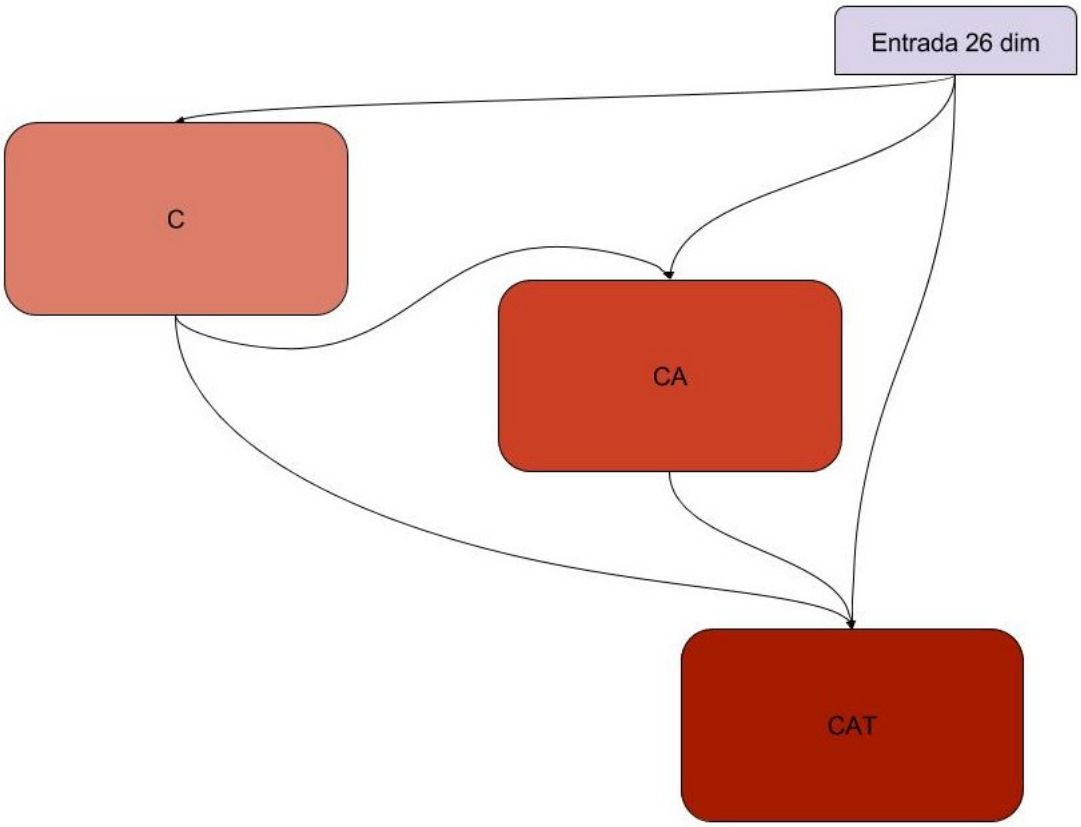

Figura 5.1: Intuición del apilamiento de modelos jerárquicos

El primer y segundo nivel son congelados para evitar un sobreajuste durante el entrenamiento de sus niveles superiores, y para evitar el problema del desvanecimiento de gradiente [20]. Es importante notar que a diferencia de las técnicas de apilamiento comunes, el conjunto de datos de entrenamiento es el mismo para todos los niveles.

5.3.5. *Keras*

Implementar modelos de redes neuronales profundas que se desempeñen bien en diferentes problemas había sido una tarea no trivial, y generalmente, reservada solo a expertos en el campo. Con el gran entusiasmo que existe en la investigación y desarrollo de modelos de aprendizaje profundo, actualmente se encuentran disponibles una gran cantidad de bibliotecas para el diseño y entrenamiento de redes neuronales.

Este proyecto utiliza *Keras* [12], una biblioteca minimalista y altamente modular para el diseño de redes neuronales, escrita en Python y capaz de correr sobre *Tensorflow* [30], lo cual permite el uso de CPU y GPU. *Keras* cuenta con un API funcional que brinda mucha flexibilidad y permite crear modelos tan complejos como aquellos que usan múltiples entradas/salidas o comparten capas, etc.

Redes *Highway*

Usando el API funcional de *Keras*, implementar una simple red neuronal *highway* es tan fácil como se muestra a continuación:

```
from keras.layers import Input, Dense, Activation, Lambda
from keras.models import Model

dim = 10

a = Input(shape=(dim,))
b = Dense(units=dim)(a)
c = Dense(units=dim)(a)
c = Activation(activation='sigmoid')(c)

highway = Lambda(function=lambda x: x[1] * x[2] + x[0] * (1. - x[2]),
   output_shape=lambda s: s[0])
d = highway([a, b, c])

model = Model(inputs=a, outputs=d)
```

En este ejemplo la mayoría de los parámetros definidos en el API de *Keras* son ignorados, para más información véase la documentación oficial de *Keras* [12].

Congelar parámetros

Keras permite congelar capas para excluirlas del entrenamiento de una red neuronal. Por lo que para congelar un modelo completo es posible definir:

```
def make_trainable(net, val):
    net.trainable = val
    for l in net.layers:
        try:
            l.trainable = val
        except AttributeError:
            pass
```

Capítulo 6

Experimentos y resultados

Diferentes experimentos fueron realizados para resolver la tarea de clasificación estructural sobre los diferentes niveles clase, arquitectura y topología de *CATH*, y supéramilia de *SCOP*. Este capítulo muestra los resultados de esos experimentos. Por último el modelo con mejor desempeño en la tarea de clasificación estructural automática sobre el nivel superfamilia de *SCOP* es usado para evaluar la tarea de búsqueda de vecinos estructurales.

6.1. Clasificación estructural *CATH*

6.1.1. Clase

Para diseñar la red neuronal específica a la tarea de clasificación del nivel clase, primero se realizó la experimentación y evaluación manual de arquitecturas de redes neuronales. Esta experimentación estableció las bases sobre cómo parametrizar los modelos evaluados posteriormente. Después se realizaron dos tipos de optimizaciones una no dirigida y otra dirigida.

Nombre del Hiperparámetro	**Distribución**	**Valores**
Número de capas ocultas	Categórica	$x \in \{2, 3, 4, 5, 6\}$
Número de unidades en la capa oculta i (En caso de estar definida)	Categórica	$x \in \{16, 128, 192, 256, 512, 1024\}$
Función de activación en capas ocultas	Categórica	$x \in \{$*softplus, softsign, tanh, sigmoid, ReLu, hard_sigmoid*$\}$
Estrategia de Inicialización de parámetros en capas ocultas	Categórica	$x \in \{$*uniform, glorot_normal, lecun_uniform, normal, orthogonal, glorot_uniform, he_normal, he_uniform*$\}$
Dropout en capas ocultas	Categórica	$x \in \{0.0, 0.1, 0.25, 0.5, 0.75, 0.9\}$

Cuadro 6.1: Definición del espacio de hiperparámetros para la optimización no dirigida

Optimización no dirigida

La optimización no dirigida consistió en el uso del algoritmo *Random Search* sobre el espacio de hiperparametros definido en el cuadro 6.1. Esta optimización resultó en el entrenamiento y evaluación de 509 redes neuronales diferentes, de las cuales destaca la red con mejor desempeño la cual obtuvo un puntaje de presicion de 0.9607818 sobre el conjunto de datos de validación.

Optimización dirigida

Al no tener claro cuántas evaluaciones de diferentes modelos son necesarias para encontrar aquel con el mejor desempeño usando el algoritmo de *Random Search* y considerando el gran costo computacional que entrenar redes neuronales requiere, se decidió probar una estrategia de optimización diferente. La estrategia elegida fue la Optimización Secuencial Basada en Modelo, específicamente usando el algoritmo *TPE*. Para esta estrategia se usó el espacio definido en el cuadro 6.1.1 como espacio de hiperparametros. Todos los modelos en esta optimización fueron entrenados usando *ADAM* como optimizador.

Nombre del Hiperparámetro	**Distribución**	**Valores**
Número de capas ocultas	Categórica	$x \in \{2,3,4,5\}$
Número de unidades en la capa oculta i (En caso de estar definida)	Categórica	$x \in \{16, 128, 192, 256, 512, 1024\}$
Función de activación en la capa oculta i (En caso de estar definida)	Categórica	$x \in \{$*softplus, softsign, tanh, sigmoid, ReLu, hard_sigmoid*$\}$
Estrategia de Inicialización de parámetros en la capa oculta i (En caso de estar definida)	Categórica	$x \in \{$*uniform, glorot_normal, lecun_uniform, normal, orthogonal, glorot_uniform, he_normal, he_uniform*$\}$
Dropout capa oculta i	Uniforme	$x \in [0,1]$
Highway capa oculta i	Categórica	$x \in \{$**False, True**$\}$

Cuadro 6.2: Definición del espacio de hiperparámetros

Tipo de Optimización	**Algoritmo**	**# de Modelos Evaluados**	**Mejor Puntaje de Precisión Obtenido**
No Dirigida	Búsqueda Aleatoria	509	0.96078
Dirigida	*TPE* (*SMBO*)	125	0.9672

Cuadro 6.3: Cuadro comparativo entre las optimizaciones dirigida y no dirigida

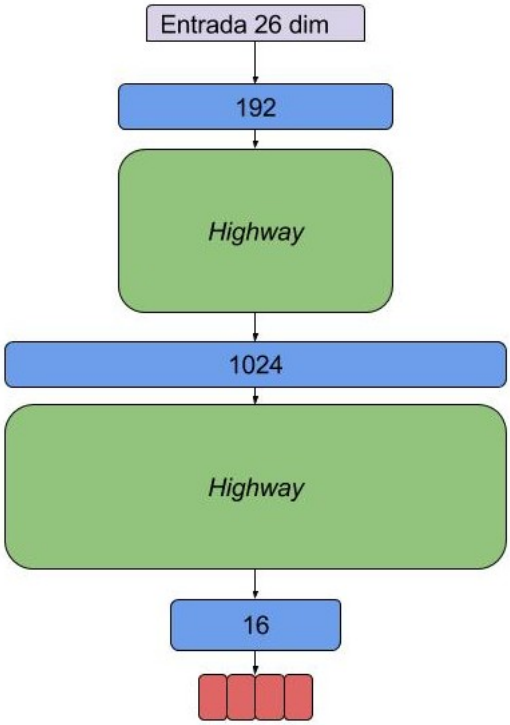

Figura 6.1: Se muestra la red neuronal con el mejor desempeño sobre los datos de validación para el nivel clase de la clasificación estructural *CATH*.

La red encontrada, véase la figura 6.1, se compone de 5 capas ocultas de las cuales la segunda y cuarta son capas *highway* y las restantes son *feedforward* comunes. La primer capa oculta se compone de 192 neuronas, la tercera se compone de 1024 neuronas y la quinta se compone de 16 neuronas. Como se mencionó anteriormente la capas capas *highway* requieren que la dimensionalidad de $\boldsymbol{x}$, $\boldsymbol{y}$, $H(\boldsymbol{x}, \boldsymbol{W}_H)$ y $T(\boldsymbol{x}, \boldsymbol{W}_T)$ sea la misma, por lo que esta dimensionalidad es dada por la capa anterior a cada capa *highway*. La salida de la última capa de la red se alimenta a un *softmax* de 4 vías que produce una distribución sobre las 4 etiquetas correspondientes al nivel clase. La no linealidad *sigmoid* es usada como la transformación T en ambas capas *highway*. Las no linealidades *softsign*, *tanh* y *softplus* se aplican a la salida de la primera, tercera y quinta capa oculta respectivamente. La técnica de regulación *dropout* fue usada durante el entrenamiento sobre la primera, tercera y quinta capa oculta con probalidades de 0.08913, 0.6354, 0.0016 respectivamente.

Una vez encontrada esta arquitectura se realizaron pruebas de validación cruzada estratificada con 10 iteraciones para evaluar la capacidad de la red para aprender reglas generales del nivel clase sobre el conjunto de datos *CATH*. Los resultados son reportados en el cuadro 6.4.

6.1.2. Arquitectura

Para el diseño de la red para el nivel de clasificación arquitectura directamente se usó una optimización de hiperparámetros usando el algoritmo *TPE*. Esta optimización se realizó sobre el espacio de parámetros definido en el cuadro 6.1.1 y la cual resultó en el entrenamiento y evaluación de 150 modelos diferentes de redes neuronales. La red con el mejor desempeño consiguió un puntaje de precisión sobre el conjunto de datos de validación de 0.9022. Esta red, véase figura 6.2, está compuesta de 5 capas ocultas y al igual que la red para el nivel clase, su segunda y cuarta son capas *highway*, las restantes son *feedforward*. La primer, tercer y quinta capa oculta se compone de 192, 1024 y 512 neuronas respectivamente. La salida de la última capa se alimenta a un *softmax* de 40 vías que produce una distribución sobre las 40 etiquetas de clase correspondientes al nivel arquitectura. La no linealidad *sigmoid* es usada como la transformación T en ambas capas *highway*. Las no linealidades *tanh*, *softplus* y *ReLu* se aplican a la salida de la primera, tercera y quinta capa oculta respectivamente.Y una probabilidad de *dropout* de 0.4663, 0.1746, 0.0989 fue utilizada durante el entrenamiento sobre la primera, tercera y quinta capa oculta respectivamente. Asimismo esta arquitectura realizó pruebas validación cruzada estratificada con 10 iteraciones, los resultados son reportados en el cuadro 6.4.

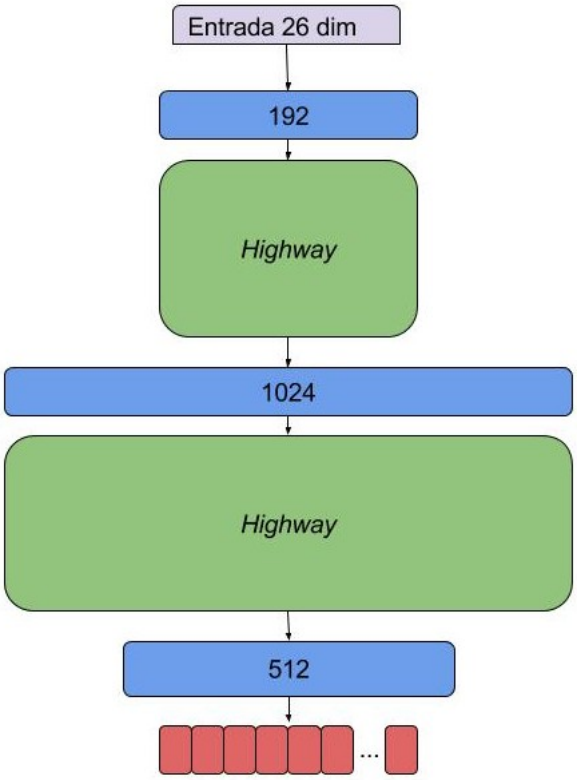

Figura 6.2: Se muestra la red neuronal con el mejor desempeño sobre los datos de validación para el nivel arquitectura de la clasificación estructural *CATH*.

6.1.3. Topología

De la misma forma que para el nivel de arquitectura, únicamente se realizó una optimización usando el algoritmo *TPE* sobre el espacio de parámetros definido en el cuadro 6.1.1. Esta resultó en la evaluación de 232 modelos de redes neuronales, de las cuales la red con el mejor resultado obtuvo un puntaje de precisión de 0.8680 sobre el conjunto de datos de validación. Esta red, véase figura 6.3, se compone de 4 capas ocultas de las cuales la segunda y cuarta son capas *highway*; las dos capas restantes son *feedforward* y se componen de 1024 neuronas. Las dimensión de las capas *highway* es determinada por la capa anterior a cada una de ellas y ambas usan la no linealidad *sigmoid* como transformación T. La salida de la última capa se alimenta a un *softmax* de 1375 vías que produce una distribución sobre las 1375 etiquetas de clase correspondientes al nivel topología. Las no linealidades *sigmoid* y *ReLu* se aplican a la salida de la primera y tercera capa oculta respectivamente y ambas capas fueron entrenadas usando *dropout* con probabilidades de 0.5543 y 0.2135 respectivamente. Al igual que para los modelos correspondientes a los niveles clase y arquitectura, esta realizó pruebas de validación cruzada estratificada con 10 iteraciones, los resultados son reportados en el cuadro 6.4.

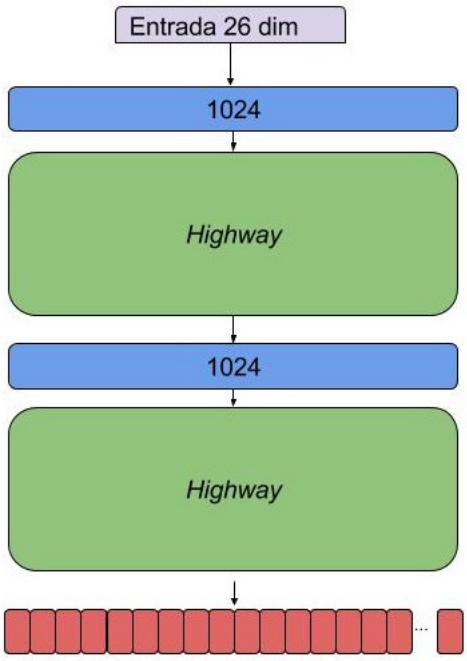

Figura 6.3: Se muestra la red neuronal con el mejor desempeño sobre los datos de validación para el nivel topología de la clasificación estructural *CATH*.

6.1.4. Modelo apilado

Una vez diseñadas las arquitecturas referentes a cada nivel de clasificación de acuerdo a *CATH*, estas fueron apiladas de manera jerárquica siguiendo el método descrito en la Sección 5.4.1. El modelo resultante, véase el figura 6.4, fue entrenado para pruebas de validación cruzada estratificada con 10 iteraciones. Los resultados son presentados en el cuadro 6.4. Es importante notar que el número de épocas usadas para entrenar cada nivel del modelo apilado es diferente que sus contrapartes no apiladas debido a que fue evaluada la detención temprana en el modelo apilado y resultó en un número diferente de épocas.

Nivel	**# de Épocas**	**Apilado**	**Puntaje de Precisión en Validación Cruzada**
C	88	No	0.9713
CA	108	No	0.9093
CA	126	Si	0.9107
CAT	62	No	0.8762
CAT	101	Si	0.8803

Cuadro 6.4: Resumen resultados validación cruzada, clasificación *CATH*

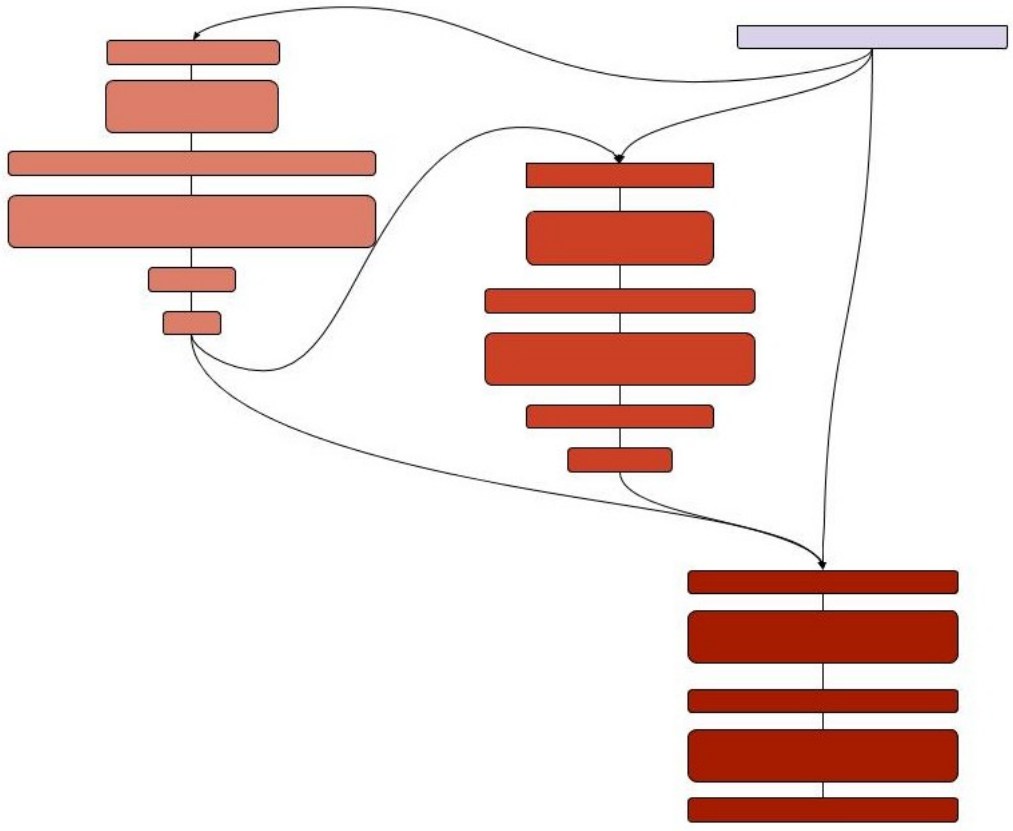

Figura 6.4: Se muestra el apilamiento de modelos jerárquicos para los modelos de los niveles clase, arquitectura y topología de la clasificación estructural *CATH*.

6.2. Clasificación estructural *SCOP*

Esta sección usa el conjunto de datos *SCOPtrain1* descrito en la Sección 5.1.

6.2.1. Superfamilia

Al igual que en el diseño de las arquitecturas para los niveles de clasificación de *CATH*, para la red del nivel Superfamilia de *SCOP* se realizó una optimización de hiperparámetros usando el algoritmo *TPE*. Esta optimización se realizó sobre el espacio de hiperparámetros definido en el cuadro 6.5 que a diferencia del espacio del cuadro 6.1.1, restringe el número de unidades ocultas a 4 y 5 capas ocultas y el número de unidades ocultas por capa oculta a 16,192, 512 y 1024, además de incorporar el uso de las no linealidades *elu* y *selu* [24].

Nombre del Hiperparámetro	**Distribución**	**Valores**
Número de capas ocultas	Categórica	$x \in \{4,5\}$
Número de unidades en la capa oculta i (En caso de ser definida)	Categórica	$x \in \{16, 192, 512, 1024\}$
Función de activación en la capa oculta i (En caso de ser definida)	Categórica	$x \in \{$ *softplus, softsign, ReLu, tanh, sigmoid, selu, elu, hard_sigmoid*$\}$
Estrategia de Inicialización de parámetros en la capa oculta i (En caso de ser definida)	Categórica	$x \in \{$*uniform, glorot_normal, lecun_uniform, normal, orthogonal, glorot_uniform, he_normal, he_uniform*$\}$
Dropout capa oculta i	Categórica	$x \in [0,1]$
Highway capa oculta i	Categórica	$x \in \{$**False, True**$\}$

Cuadro 6.5: Espacio de hiperparámetros usado para la optimización del nivel superfamilia de *SCOP*

Dicha optimización resultó en la evaluación de 1109 modelos de redes neuronales de los cuales destaco la red con el mejor desempeño sobre el conjunto de datos de validación de 0.8741. Esta red, véase figura 6.5, está compuesta de 4 capas ocultas, de la cuales la segunda y cuarta son capas *highway*, las restantes son *feedforward*. La primer y tercer capa oculta se compone de 512 y 1024 neuronas respectivamente. La salida de la última capa se alimenta a un *softmax* de 2217 vías que produce una distribución sobre las 2217 etiquetas de clase correspondientes al nivel Superfamilia. La no linealidad *sigmoid* es usada como la transformación T en ambas capas *highway*. Las no linealidades *tanh* y *sigmoid* se aplican a la salida de la primera y tercera capa oculta respectivamente.Y una probabilidad de *dropout* de 0.7233 y 0.4701 fue utilizada durante el entrenamiento sobre la primera y tercera capa oculta respectivamente. De la misma manera que los modelos evaluados para *CATH*, esta arquitectura realizó pruebas de validación cruzada estratificada con 10 iteraciones, donde el puntaje de precisión en validación cruzada fue de 0.8821 (el promedio de los puntajes de cada una de las 10 iteraciones).

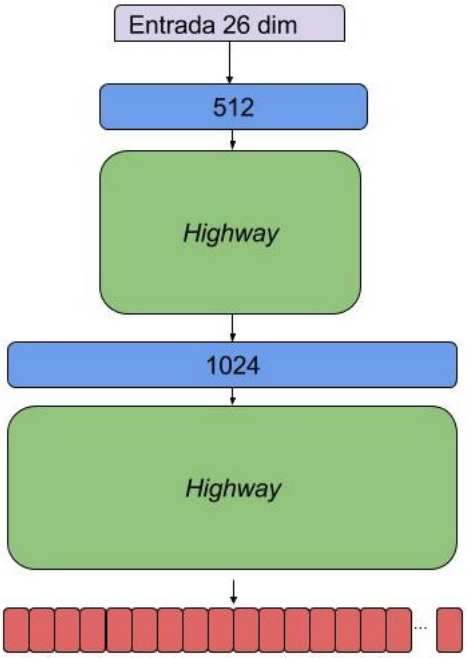

Figura 6.5: Se muestra la red neuronal con el mejor desempeño sobre los datos de validación para el nivel Superfamilia de la clasificación estructural *SCOP*.

6.3. Vecinos estructurales

Una vez elegida la red para resolver la tarea de clasificación en el nivel superfamilia de acuerdo a *SCOP*, se evaluó la capacidad de la misma para identificar vecinos estructurales usando el algoritmo descrito en la Sección 5.3 sobre el conjunto de datos *SCOP30*. Se comparó la lista de estructuras ordenadas resultante del algoritmo con la lista de vecinos para tres vecindades *SAS* (2.0, 3.5 y 5.0). Para realizar la evaluación se calculó el *AUROC* para cada estructura en *SCOP30*. En el cuadro 6.6 se resumen los resultados de la evaluación para cada vecindad *SAS*.

Vecindad *SAS*	**Promedio *AUROC***
2.0	0.9004
3.5	0.7264
5.0	0.5922

Cuadro 6.6: Resultados evaluación búsqueda de vecinos estructurales

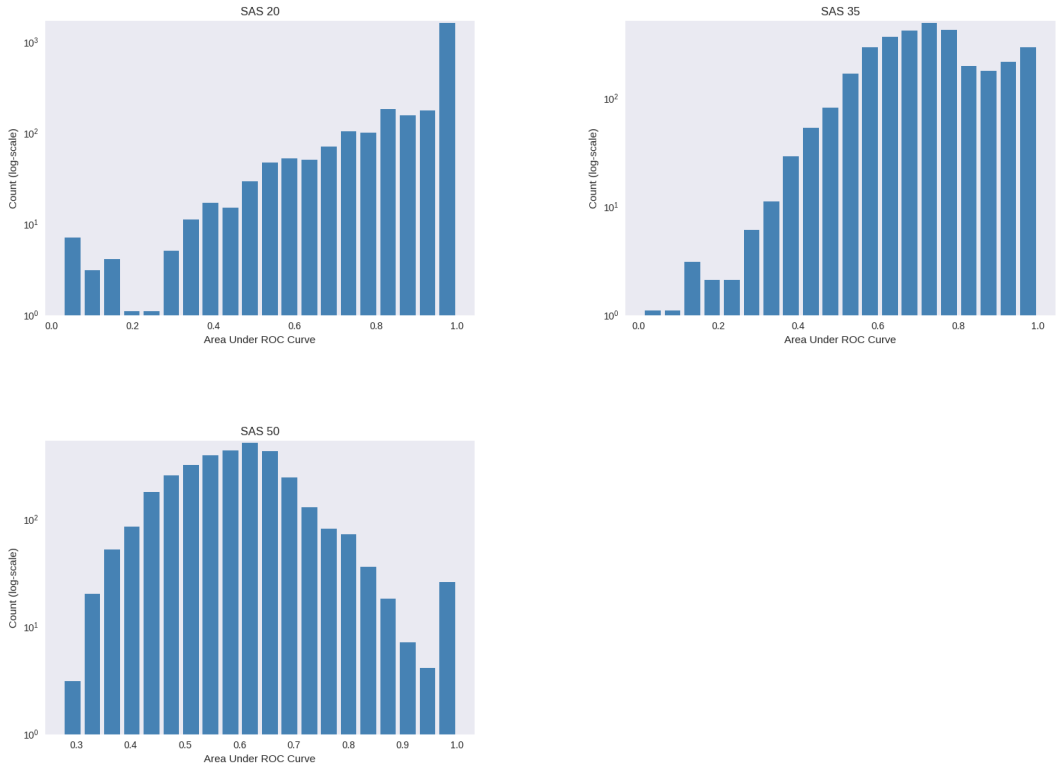

Figura 6.6: Histogramas de los *AUROC* por nivel *SAS*

Capítulo 7

Discusión y Conclusiones

El objetivo de este trabajo fue el de explorar el uso de arquitecturas de redes neuronales profundas para la cuantificación de similitud de estructuras de proteínas mediante la solución de las tareas de clasificación estructural automática y obtención de vecinos estructurales usando una representación basada en el sistema *RCC*.

Este trabajo ha implementado diversas técnicas para el diseño, optimización y evaluación de varias arquitecturas de redes neuronales profundas. Se ha mostrado empíricamente que el uso de una Optimización Secuencial Basada en Modelos es más eficiente para la optimización de hiperparámetros de un modelo de aprendizaje profundo, que una Optimización Aleatoria. Además se introdujo un método de regularización efectivo que consiste en el apilamiento de modelos basado en la estructura jerárquica que establece la definición de *CATH*.

Los puntajes de precisión de las arquitecturas de redes neuronales profundas mostradas en el cuadro 6.5 muestran un buen desempeño en las pruebas de validación cruzada para la tarea de clasificación estructural automática, siendo de 0.971 para el nivel clase, 0.910 para arquitectura y 0.880 para topología de *CATH*; éstos pueden ser directamente comparados con los reportados por Corral-Corral *et al.* [13] usando modelos de Bosques Extremadamente Aleatorios de 0.969, 0.883 y 0.849, respectivamente. Asimismo, un resultado análogo se puede ver en la clasificación estructural automática del nivel Superfamilia de *SCOP* cuando se compara el puntaje de precisión de 0.882 obtenido por el modelo de aprendizaje profundo contra el reportado por Corral-Corral *et al.* [13] de 0.8781. Sin embargo, lo mismo no sucede a la hora de evaluar la capacidad de identificar verdaderos vecinos, siendo los Bosques Extremadamente Aleatorios significativamente superiores en dicha tarea que la red neuronal presentada en este trabajo. No obstante, existen buenas razones para no sobre generalizar los resultados presentados en la solución de dicha tarea usando el algoritmo descrito en la Sección 5.2 y arquitecturas de redes neuronales profundas, ya que existen un número infinito de grados de libertad para el diseño de éstas, además de arbitrariedad. También es importante considerar que la fracción de dominios que pertenecen a la misma clasificación en SCOP y se encuentran en la misma vecindad *SAS* es pequeña para el nivel superfamilia, véase el cuadro 7.1.

Nivel *SAS*	**Familia**	**Superfamilia**	**Plegado**	**Clase**
2.0	0.1288	0.3837	0.8417	0.9198
3.5	0.0140	0.0571	0.1116	0.7640
5.0	0.0032	0.0138	0.0311	0.4437

Cuadro 7.1: Clasificación *SCOP* y coincidencia de puntaje *SAS*. La tabla muestra la fracción de pares de dominios alineados en cada nivel SAS (2.0, 3.5, 5.0) que pertenece a la misma clasificación *SCOP*. Recuperado de [13].

Por último los puntajes de validación cruzada presentados en este trabajo pueden servir como nuevos puntos de comparación para aprender las clasificaciones estructurales de *CATH* y *SCOP* sobre el espacio de plegados basado en el sistema de *RCC*.

Para facilitar el uso de las tecnicas expuestas en este trabajo a terceros se ha implementado una API RESTful disponible en http://docs.neuralprotein.space/.

Bibliografía

[1] *Aminoácidos*. 2018. URL: http://www.omundodaquimica.com.br/academica/aminoacidos.

[2] *Artificial Neural Networks/Feed-Forward Networks*. URL: https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Feed-Forward_Networks.

[3] James Bergstra JAMESBERGSTRA y Umontrealca Yoshua Bengio YOSHUABENGIO. «Random Search for Hyper-Parameter Optimization». En: *Journal of Machine Learning Research* 13 (2012), págs. 281-305. ISSN: 1532-4435. DOI: 10.1162/153244303322533223. arXiv: 1504.05070.

[4] J. Bergstra, D. Yamins y D. D. Cox. «Making a Science of Model Search». En: (2012), págs. 1-11. arXiv: 1209.5111. URL: http://arxiv.org/abs/1209.5111.

[5] James Bergstra, Dan Yamins y David D Cox. «Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms». En: *12th PYTHON IN SCIENCE CONF. (SCIPY 2013)* Scipy (2013), págs. 13-20. ISSN: 1749-4699. DOI: 10.1088/1749-4699/8/1/014008. URL: http://hyperopt.github.io/hyperopt/%7B%5C%%7D5Cnhttps://github.com/jaberg/hyperopt%7B%5C%%7D5Cnhttp://www.youtube.com/watch?v=Mp1xnPfE4PY.

[6] James Bergstra y col. «Algorithms for Hyper-Parameter Optimization». En: *Advances in Neural Information Processing Systems (NIPS)* (2011), págs. 2546-2554. ISSN: 10495258. DOI: 2012arXiv1206.2944S. arXiv: 1206.2944.

[7] H M Berman y col. «The protein data bank.» En: *Nucleic acids research* 28.1 (2000), págs. 235-242. ISSN: 0305-1048. DOI: 10.1093/nar/28.1.235.

[8] I. Budowski-Tal, Y. Nov y R. Kolodny. «FragBag, an accurate representation of protein structure, retrieves structural neighbors from the entire PDB quickly and accurately». En: *Proceedings of the National Academy of Sciences* 107.8 (2010), págs. 3481-3486. ISSN: 0027-8424. DOI: 10.1073/pnas.0914097107. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.0914097107.

[9] Saramita Chakravarti. *Protein Structure, Databases and Structural Alignment*. 2013. URL: https://www.slideshare.net/SaramitaDeChakravart/protein-structure-databases-and-structural-alignment.

[10] Jianlin Cheng, Allison N. Tegge y Pierre Baldi. «Machine Learning Methods for Protein Structure Prediction». En: *IEEE Reviews in Biomedical Engineering* 1 (2008), págs. 41-49. ISSN: 19411189. DOI: 10.1109/RBME.2008.2008239. arXiv: arXiv:1011.1669v3.

[11] François Chollet. *Deep Learning with Python*. Shelter Island, NY: Manning Publications Co., 2018.

[12] François Chollet y col. *Keras*. https://github.com/keras-team/keras. 2015.

[13] Ricardo Corral-Corral, Edgar Chavez y Gabriel Del Rio. «Machine Learnable Fold Space Representation based on Residue Cluster Classes». En: *Computational Biology and Chemistry* 59 (2015), págs. 1-7. ISSN: 14769271. DOI: 10.1016/j.compbiolchem.2015.07.010. URL: http://dx.doi.org/10.1016/j.compbiolchem.2015.07.010.

[14] Xuefeng Cui y col. «Fingerprinting protein structures effectively and efficiently». En: *Bioinformatics* 30.7 (2014), págs. 949-955. ISSN: 14602059. DOI: 10.1093/bioinformatics/btt659.

[15] Xavier Glorot y Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks». En: *Pmlr* 9 (2010), págs. 249-256. ISSN: 15324435. DOI: 10.1.1.207.2059. arXiv: arXiv:1011.1669v3. URL: http://machinelearning.wustl.edu/mlpapers/paper%7B%5C_%7Dfiles/AISTATS2010%7B%5C_%7DGlorotB10.pdf.

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

[17] Ben Gorman. *Guide to Model Stacking (i.e. Meta Ensembling)*. 2016. URL: https://gormanalysis.com/guide-to-model-stacking-i-e-meta-ensembling/.

[18] Kaiming He y col. «Delving deep into rectifiers: Surpassing human-level performance on imagenet classification». En: *Proceedings of the IEEE International Conference on Computer Vision* 2015 International Conference on Computer Vision, ICCV 2015 (2015), págs. 1026-1034. ISSN: 15505499. DOI: 10.1109/ICCV.2015.123. arXiv: 1502.01852.

[19] Geoffrey Hinton. *Neural Networks for Machine Learning*. https://www.coursera.org/learn/neural-networks.

[20] Josef Hochreiter. «Untersuchungen zu dynamischen neuronalen Netzen». En: *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen* (1991), págs. 1-71. URL: http://www.bioinf.jku.at/publications/older/3804%7B%5C_%7D2.pdf%7B%5C%%7D0Ahttp://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Untersuchungen+zu+dynamischen+neuronalen+Netzen%7B%5C#%7D0.

[21] J. Hou y col. «A global representation of the protein fold space». En: *Proceedings of the National Academy of Sciences* 100.5 (2003), págs. 2386-2390. ISSN: 0027-8424. DOI: 10.1073/pnas.2628030100. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.2628030100.

[22] Pooja Jain y Jonathan D Hirst. «Automatic structure classification of small proteins using random forest.» En: *BMC bioinformatics* 11 (2010), pág. 364. ISSN: 1471-2105. DOI: 10.1186/1471-2105-11-364. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-77954053524%7B%5C&%7DpartnerID=tZOtx3y1.

[23] Diederik Kingma y Jimmy Ba. «ADAM: A Method for Stochastic Optimization». En: *Proc. of the 3rd International Conference for Learning Representations* (2015), págs. 1-15. arXiv: 1412.6980v8. URL: http://arxiv.org/abs/1412.6980.

[24] Günter Klambauer y col. «Self-Normalizing Neural Networks». En: (2017). DOI: 1706.02515. arXiv: 1706.02515. URL: http://arxiv.org/abs/1706.02515.

[25] Rachel Kolodny y Nathan Linial. «Approximate protein structural alignment in polynomial time.» En: *Proceedings of the National Academy of Sciences of the United States of America* 101.33 (2004), págs. 12201-6. ISSN: 0027-8424. DOI: 10.1073/pnas.0404383101. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=514457%7B%5C&%7Dtool=pmcentrez%7B%5C&%7Drendertype=abstract.

[26] Rachel Kolodny y col. «On the Universe of Protein Folds». En: *Annual Review of Biophysics* 42.1 (2013), págs. 559-582. ISSN: 1936-122X. DOI: 10.1146/annurev-biophys-083012-130432. URL: http://www.annualreviews.org/doi/10.1146/annurev-biophys-083012-130432.

[27] Yann A. LeCun y col. «Efficient backprop». En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7700 LECTURE NO (2012), págs. 9-48. ISSN: 03029743. DOI: 10.1007/978-3-642-35289-8-3. arXiv: 9809069v1 [arXiv:gr-qc].

[28] Yann LeCun y col. «Gradient Based Learning Applied to Document Recognition». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. ISSN: 00189219. DOI: 10.1109/5.726791. arXiv: 1102.0183. URL: http://ieeexplore.ieee.org/abstract/document/726791/.

[29] SPANDAN MADAN. *An end to end implementation of a Machine Learning pipeline*. 2017. URL: https://spandan-madan.github.io/DeepLearningProject/.

[30] Martin Abadi y col. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[31] Alex Minnaar. *The Single Neuron Model*. 2015. URL: http://alexminnaar.com/deep-learning-basics-neural-networks-backpropagation-and-stochastic-gradient-descent.html.

[32] Alexey G. Murzin y col. «SCOP: A structural classification of proteins database». En: *Nucleic Acids Research* 27.1 (1999), págs. 254-256. ISSN: 03051048. DOI: 10.1093/nar/27.1.254. arXiv: arXiv:0912.3967v1.

[33] Ca Orengo y col. «CATH - a hierarchic classification of protein domain structures». En: *Structure* 17.March (1997), págs. 1093-1109. ISSN: 09692126. DOI: 10.1016/S0969-2126(97)00260-8. URL: http://discovery.ucl.ac.uk/170914/.

[34] Kuldip Paliwal, James Lyons y Rhys Heffernan. «A short review of deep learning neural networks in protien structure prediction problems». En: *Advanced Techniques in Biology and Medicine* 3.3 (2015), pág. 1000139. DOI: 10.4172/2379-1764.

[35] *Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description*. URL: http://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html.

[36] Ladislav Rampasek y Anna Goldenberg. «TensorFlow: Biology's Gateway to Deep Learning?» En: *Cell Systems* 2.1 (2016), págs. 12-14. ISSN: 24054720. DOI: 10.1016/j.cels.2016.01.009. URL: http://dx.doi.org/10.1016/j.cels.2016.01.009.

[37] Sebastian Raschka. «Model evaluation , model selection , and algorithm selection in machine learning Performance Estimation : Generalization Performance Vs . Model Selection». En: January (2016), págs. 1-13.

[38] Sebastian Raschka y Vahid Mirjalili. *Python Machine Learning, Second Edition*. Packt Publishing Ltd., 2017.

[39] Michael K. Reddy. *Amino acid*. 2018. URL: https://www.britannica.com/science/amino-acid.

[40] David E. Rumelhart, Geoffrey E. Hinton y Ronald J Williams. *Learning representations by back-propagating errors*. 1986. URL: https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf.

[41] S. Shivashankar y col. «Multi-view methods for protein structure comparison using latent dirichlet allocation». En: *Bioinformatics* 27.13 (2011), págs. 61-68. ISSN: 13674803. DOI: 10.1093/bioinformatics/btr249.

[42] Paolo Sonego, András Kocsor y Sándor Pongor. «ROC analysis: Applications to the classification of biological sequences and 3D structures». En: *Briefings in Bioinformatics* 9.3 (2008), págs. 198-209. ISSN: 14675463. DOI: 10.1093/bib/bbm064.

[43] Nitish Srivastava y col. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». En: *Journal of Machine Learning Research* 15 (2014), págs. 1929-1958. ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807.

[44] Rupesh Kumar Srivastava, Klaus Greff y Jürgen Schmidhuber. «Highway Networks». En: (2015). arXiv: 1505.00387. URL: http://arxiv.org/abs/1505.00387.

[45] Edgar Vázquez-Contreras. *Enlace peptídico*. 2003. URL: http://laguna.fmedic.unam.mx/~evazquez/0403/enlace/%20peptidico.html.

[46] Edgar Vázquez-Contreras. *Los diferentes niveles de estructura en una proteína*. 2003. URL: http://laguna.fmedic.unam.mx/~evazquez/0403/introduccion%5C%20estructura%5C%20proteinas.html.

[47] Jinbo Xu, Feng Jiao y Bonnie Berger. «A parameterized algorithm for protein structure alignment.» En: *Journal of computational biology : a journal of computational molecular cell biology* 14.5 (2007), págs. 564-77. ISSN: 1066-5277. DOI: 10.1089/cmb.2007.R003. URL: http://www.ncbi.nlm.nih.gov/pubmed/17683261.

[48] *Yearly Growth of Protein Structures*. 2018. URL: http://www.rcsb.org/pdb/statistics/contentGrowthChart.do?content=molType-protein.