Universidad Tecnológica de la Mixteca

"Aprendizaje de representaciones de secuencia de aminoácidos utilizando arquitecturas profundas"

tesis

para obtener el grado de

Maestro en Tecnologías de Cómputo Aplicado

presenta:

Ing. Erik Germán Ramos Pérez

Director de tesis

Dr. Raúl Cruz Barbosa

Huajuapan de León, Oax., Febrero de 2016

A Liss

Agradecimientos

Agradezco a *Liss* por estar a mi lado en todo momento, sin su cariño, motivación y amor nada de esto sería posible.

Este trabajo no habría sido posible sin el apoyo y motivación de mi asesor, el Dr. Raúl Cruz Barbosa, agradezco el tiempo y conocimiento brindado.

Agradezco a los sinodales Dra. Lluvia Carolina Morales Reynaga, Dr. Felipe de Jesús Trujillo Romero, Dr. José Anibal Arias Aguilar y al Dr. Santiago Omar Caballero Morales por el tiempo dedicado en la revisión de este trabajo.

A la Universidad Tecnológica de la Mixteca por darme la oportunidad de recibir la formación académica y a sus profesores por el conocimiento recibido.

No puedo terminar de agradecer a todos los familiares, amigos y compañeros de trabajo, en los que siempre encontraré el aliento para seguir adelante.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El Aprendizaje Automático (o computacional) pretende construir un programa de computadora que optimiza una función objetivo usando ejemplos de datos (casos conocidos) o experiencia acumulada. Esto es, busca aprender una función a través de la inducción que sea útil para la aproximación de soluciones. Este tipo de aprendizaje es utilizado cuando se intenta resolver un problema para el cual no existe un algoritmo de manera explícita [Alpaydin, 2010].

Generalmente, el desempeño de los métodos de aprendizaje automático depende en gran medida de la elección de la representación o características de los datos. La obtención y representación de los datos es tan importante como los algoritmos de aprendizaje [Russell and Norvig, 2009], esto significa que una mala representación deriva en un pobre rendimiento de los algoritmos usados en la etapa de aprendizaje. Es por esto, que encontrar una representación adecuada de los datos que se analizan es una tarea clave en el aprendizaje automático.

Mientras que muchas investigaciones buscan las representaciones de forma manual, mediante formulas explícitas, explotando así el conocimiento humano del problema, otros estudios recientes tratan de descubrir buenas representaciones mediante el aprendizaje de éstas a través de los datos. Esto último se conoce como aprendizaje de representaciones, el cual es un conjunto de técnicas de aprendizaje automático que aprenden una transformación de los datos originales de tal forma que dicha representación pueda ser explotada de manera efectiva en tareas como clasificación o agrupamiento, entre otras [Bengio, 2009]. Es por esto que se dice que al aprender representaciones o características se puede llegar a mejoras significativas con respecto a los modelos estándares supervisados en pruebas fuera

de su dominio [Huang and Yates, 2010]. Para esto, el aprendizaje de representaciones usa un enfoque no supervisado, el cual tiene la ventaja de no requerir de información de la clase o categoría de los datos analizados. Además, estas técnicas no necesitan de conocimiento a priori del problema para obtener una buena representación [Schmidhuber, 2015].

Una técnica para el aprendizaje de representaciones se denomina arquitecturas profundas. Aquí, se contruyen varias capas de procesamiento de información, donde cada una es una red neuronal (Máquina de Boltzmann restringida) que se conecta de manera jerárquica con otra [Schmidhuber, 2015, Bengio, 2009].

Un ejemplo clásico de una arquitectura profunda es el cerebro humano, ya que los humanos organizan sus ideas y conceptos de forma jerárquica. Primero, aprenden conceptos simples y después los modifican para poder representar conceptos más abstractos. Supóngase que una persona ve una imagen, en el nivel más bajo sólo observará píxeles, en el siguiente nivel puede notar detectores de bordes, en el que sigue detectar formas y en el siguiente nivel, los objetos. Esto es, en cada nivel podrá observar cosas más especializadas [Bengio, 2009].

Con las arquitecturas profundas, es posible entonces utilizar el aprendizaje profundo (del inglés Deep Learnig), el cual es un aprendizaje de múltiples niveles de representación. La intención es hallar las características más abstractas en los niveles más altos de la representación, con lo que se espera, sea más fácil separar los diferentes factores explicativos de los datos.

En general, respecto a las aplicaciones posibles ha habido un avance importante usando las arquitecturas profundas, estableciendo mejoras en los rendimientos de clasificación. Entre las más destacadas se encuentran el reconocimiento de voz y reconocimiento de imágenes

La finalidad del presente trabajo de tesis es proponer una arquitectura profunda para obtener una buena representación de secuencias de aminoácidos pertenecientes a los receptores acoplados a proteínas G, los cuales son muy importantes para el desarrollo de nuevos fármacos. Dicha representación se usará para poder clasificar las secuencias y comparar el rendimiento con otras técnicas tradicionales.

1.1. Planteamiento del problema

Un problema relevante en la Biología Computacional y Bioinformática es la clasificación de proteínas dada su secuencia de aminoácidos, en la cual está implícita su función y estructura correspondiente.

Un paso importante para mejorar el rendimiento de un clasificador de secuencias de aminoácidos es encontrar una representación significativa de su secuencia [Weston et al., 2005]. Una representación de secuencias que extrae características basadas en el perfil (regiones bien conservadas) mejora la precisión teniendo en cuenta la información evolutiva extraída de estas. Sin embargo, dichas representaciones conllevan a un elevado costo computacional debido a que impide la aplicación generalizada de estos métodos para una base de datos grande [Liu et al., 2012].

Por otro lado, una representación basada en la información sobre la composición de aminoácidos de su secuencia puede generar los vectores de características con bajo costo computacional. El problema asociado a esto, es que si una proteína está representada sólo por dicha composición, toda la información de su orden y longitud de secuencia queda excluida [Weston et al., 2005, Liu et al., 2012].

Por lo tanto, es necesario encontrar una representación significativa con la menor pérdida de información. En el presente proyecto de tesis, se pretende encontrar representaciones significativas de secuencias de aminoácidos a través del uso de arquitecturas profundas.

1.2. Justificación

La clasificación de proteínas en familias o clases y éstas en tipos y subtipos puede contribuir al avance en el diseño de fármacos y también para una mejor comprensión de los procesos moleculares implicados en la señalización del receptor, tanto en condiciones normales como patológicas [Cruz-Barbosa et al., 2013].

Los métodos existentes para la clasificación de proteínas utilizan diferentes características o propiedades de estas para lograrla. El éxito para obtener una adecuada clasificación depende mucho de la representación de los datos, esto se debe a que las diferentes representaciones pueden excluir y ocultar los diferentes factores explicativos de la variación detrás de los datos.

Estudios de extracción de la información en diferentes áreas han utilizado técnicas de aprendizaje automático, tales como modelos ocultos de Márkov, modelos de máxima entropía, máquinas de soporte vectorial [Bhasin and Raghava, 2004, Bhasin and Raghava, 2005, Papasaikas et al., 2003], entre otras. Estas utilizan, principalmente, transformaciones de la secuencia de aminoácidos original, que sirven como representación de entrada a los algoritmos. En contraste, recientemente, hay una tendencia a extraer las características principales a partir de los datos originales [Qi et al., 2014, Collobert et al., 2011], esto es, sin ningún tipo de ingeniería de características. Lo anterior, conduce a un campo nuevo llamado aprendizaje de representaciones, el cual se puede lograr mediante arquitecturas profundas. Por lo tanto, en este trabajo, se desarrollará una arquitectura profunda para obtener representaciones significativas de secuencias de aminoácidos de receptores acoplados a proteíndas G de la clase C, las cuáles no han sido estudiadas mediante este enfoque.

Con el desarrollo del presente proyecto de tesis, se pretende también generar una línea de investigación de aprendizaje profundo (Deep Learnig), la cuál podría apoyar a posgrado y al cuerpo académico de reconocimiento de patrones de la UTM.

Pertinencia

Para el desarrollo del presente trabajo, se cuenta con un servidor robusto en donde se realizarán las pruebas necesarias, ya sea de métodos de arquitecturas profundas o de clasificadores.

También en la Universidad hay expertos en el área de aprendizaje automático, y en particular, redes neuronales que podrán guiar el desarrollo del trabajo.

1.3. Hipótesis

Las representaciones de secuencias de aminoácidos utilizando arquitecturas profundas ayudan a mejorar el desempeño de exactitud de clasificación de estas.

1.4. Objetivos

Objetivo general

Configurar e implementar una arquitectura profunda para obtener una representación de secuencia de aminoácidos que ayude a mejorar el desempeño de clasificación de estas.

Objetivos particulares

1. Revisar el estado del arte de representaciones de secuencias de aminoácidos y de las arquitecturas profundas.
2. Seleccionar y configurar arquitecturas profundas adecuadas para la representación de secuencias de aminoácidos.
3. Implementar la mejor arquitectura profunda seleccionada en el punto anterior.
4. Comparar el desempeño computacional de la arquitectura profunda con otros tipos de representación de secuencia de aminoácidos.
5. Comparar el rendimiento de un clasificador utilizando la representación obtenida por la arquitectura profunda con otros clasificadores existentes.

1.5. Metas

1. Reporte del estado del arte de representaciones de secuencias de aminoácidos.
2. Reporte del estado del arte de las arquitecturas profundas.
3. Reporte de las arquitecturas profundas seleccionadas adecuadas para la representación de secuencias de aminoácidos.
4. Implementación de la mejor arquitectura profunda seleccionada para la representación de secuencias de aminoácidos en el lenguaje de programación C++.
5. Generación de una biblioteca para arquitecturas profundas.
6. Cuadro comparativo del desempeño computacional de la arquitectura profunda con otros tipos de representación de secuencia de aminoácidos.

7. Cuadro comparativo del rendimiento del clasificador con arquitectura profunda y otros clasificadores.

8. Elaboración de documento de tesis.

9. Publicación de, al menos, un artículo.

1.6. Trabajo relacionado

El aprendizaje de representaciones ha llegado a ser muy importante en la comunidad de aprendizaje automático, logrando tener éxito tanto en la industria como en la academia [Bengio et al., 2013]. Por ejemplo, éste ha tenido un fuerte impacto en el área de reconocimiento de voz con resultados muy importantes en la reducción de tasa de error [Mohamed and Hinton, 2010, Dahl et al., 2012, Mohamed et al., 2012, Seide et al., 2011]. Los algoritmos de aprendizaje de representaciones también se han empleado en la música [Boulanger-Lewandowski et al., 2012] mejorando el error relativo entre 5 % y 30 %.

En otro ámbito, ha habido también un importante avance en el análisis de imágenes, particularmente el reconocimiento de dígitos usando la base de datos MNIST, logrando superar a las máquinas de soporte vectorial que tenían el record de 1.4 % de tasa de error. Actualmente, una versión de arquitectura profunda usando una red convolucional [Rifai et al., 2011] logró 0.27 % de dicha tasa.

Collobert y Weston [Collobert and Weston, 2008] obtuvieron muy buenos resultados en el campo de procesamiento de lenguaje natural. Estos resultados podrían ser usados en el presente trabajo haciendo una analogía con el etiquetado funcional de aminoácidos. Es decir, el lenguaje natural puede ser anotado con etiquetas indicando pares de palabras sinónimas, categorías gramaticales, entidades sintácticas grandes, etc. Estos etiquetados muestran una fuerte dependencia entre tareas y en [Collobert and Weston, 2008] se demuestra que una arquitectura de red neuronal unificada, entrenada simultáneamente con un conjunto de tareas relacionadas, ofrece etiquetados más precisos que los obtenidos si se hubiera realizado con una red entrenada solamente con una tarea.

Para el éxito del trabajo mencionado anteriormente fue esencial el uso de una red neuronal profunda, la cual es capaz de aprender una jerarquía de características que son relevantes para las tareas utilizando entradas muy básicas. Con la arquitectura multitarea

profunda es posible ignorar el complicado proceso de la incorporación manual de características para cada tarea. Finalmente una parte fundamental de la metodología usada por [Collobert and Weston, 2008] es el uso de una tarea de modelado de lenguaje, en la cual la red aprende a discriminar entre oraciones genuinas y oraciones generadas sintéticamente.

En otro trabajo realizado por [Qi et al., 2014], utilizan una arquitectura profunda para la extracción de información basada en caracteres, en particular del lenguaje chino. En el texto chino no existen espacios que delimiten una palabra de otra, lo cual se asemeja mucho a la secuencia de aminoácidos.

La propuesta en [Qi et al., 2014] consiste en un sistema unificado de extremo a extremo que, dada una cadena de caracteres, proporciona varias capas de extracción de características y predice las etiquetas para cada caracter. Las características relevantes para el objetivo de extracción de la información son aprendidas automáticamente por retro-propagación en las capas más profundas del modelo de red.

1.7. Metodología

Para abordar el tema de representación de secuencias de aminoácidos, se realizará un estudio sobre las representaciones existentes y los métodos que se han utilizado para obtener dichas representaciones, es decir, los métodos más relevantes o más usados.

Los datos de las secuencias de aminoácidos que serán utilizados para el desarrollo de este proyecto de tesis se obtendrán de una base de datos pública denominada GPCRDB (Base de Datos de Receptores Acoplados a Proteínas G) [Vroling et al., 2011]. Dicha base de datos divide la súper-familia GPCR en cinco clases principales (de la A a la E) basados en el tipo de ligando, funciones y similitud de secuencias.

Para llevar a cabo la extracción automática de características de las secuencias se desarrollará una arquitectura profunda, en donde se tendrá que definir la configuración necesaria para poder obtener las características más convenientes. Para poder construir dicha arquitectura, es necesario realizar una investigación exhaustiva de en dónde se han aplicado, las ventajas que ofrecen y el rendimiento comparado con otras representaciones.

Una vez obtenidas las características es necesario probar si con ellas se logra una clasificación con buen rendimiento, lo cual se realizará utilizando dos clasificadores.

La documentación correspondiente del presente trabajo se irá redactando durante y conforme se vayan concluyendo las etapas y logrando las metas propuestas.

Capítulo 2

Fundamento teórico

En este capítulo se explica qué es el aprendizaje automático, y los tipos de aprendizaje más usados. También, se explica en qué consiste el aprendizaje de representaciones y por qué utilizarlo, se da una introducción de las arquitecturas profundas y las más empleadas. Finalmente, se explican las transformaciones convencionales de secuencias de aminoácidos más relevantes.

2.1. Aprendizaje automático

El aprendizaje automático ha sido muy importante en la revolución tecnológica basada en el uso inteligente de la información. Este es una rama de la inteligencia artificial donde se pretende crear algoritmos capaces de generalizar comportamientos y reconocer patrones usando ejemplos de datos (casos conocidos) o experiencia acumulada. Es decir, métodos que permiten a través de la inducción encontrar soluciones aproximadas. Este tipo de aprendizaje es utilizado cuando se intenta resolver un problema para el cual no existe un algoritmo explícitamente [Alpaydin, 2010].

Existen diferentes tipos de aprendizaje automático:

Aprendizaje supervisado. En este tipo de aprendizaje los ejemplos de datos (entradas) se encuentran etiquetados (salidas), y consiste de pares (x_i, y_i), donde x_i es la entrada obtenida de una distribución de probabilidad desconocida, y y_i es la salida conocida que se asocia con la entrada x_i (ver figura 2.1). Esto es, con la información anterior se pretende que después de la fase de entrenamiento el algoritmo de aprendizaje proporcione salidas (soluciones) adecuadas ante entradas nuevas o desconocidas por éste.

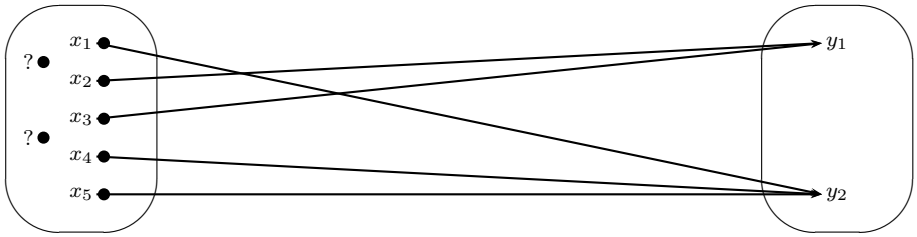

Figura 2.1: Aprendizaje supervisado

Aprendizaje no-supervisado. En este tipo de aprendizaje los ejemplos de datos (entradas) no se encuentran etiquetados, esto es, sólo se conocen los datos de entrada x_i. Con este tipo de aprendizaje se busca encontrar propiedades o estructuras ocultas en los datos (ver figura 2.2).

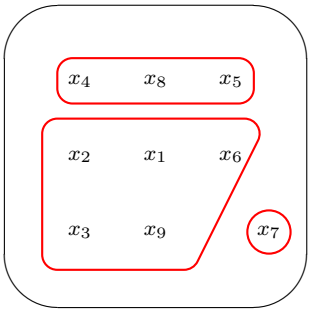

Figura 2.2: Aprendizaje no supervisado

Aprendizaje semi-supervisado. En este tipo de aprendizaje se conocen muy pocos ejemplos de datos (entradas) que se encuentran etiquetados, y un gran número de éstos sin información de clase o etiqueta. Aquí, se pueden realizar tareas tanto de tipo no supervisado como supervisado (ver figura 2.3).

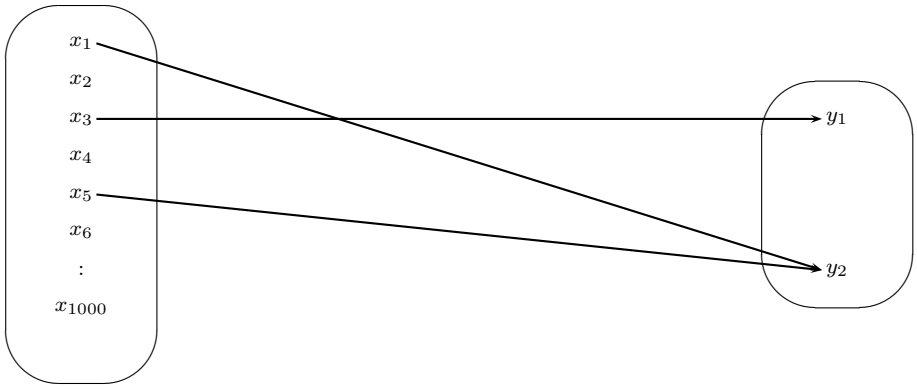

Figura 2.3: Aprendizaje semi-supervisado

Generalmente, el desempeño de los métodos de aprendizaje automático depende en gran medida de la elección de la representación de los datos o características. La obtención, y representación de los datos es tan importante como los algoritmos de aprendizaje [Russell and Norvig, 2009], lo cual significa que una mala representación deriva en un pobre rendimiento de los algoritmos usados en la etapa de aprendizaje.

2.2. Aprendizaje de representaciones

Antes de explicar en que consiste el aprendizaje de representaciones, primero se muestra qué es una representación. Esta es un sistema formal el cual "hace explícitas ciertas entidades y tipos de información" [Marr, 1982], las cuales pueden ser operadas por un algoritmo con el fin de alcanzar alguna meta al procesar dicha información.

Ahora, el aprendizaje de representaciones es un conjunto de técnicas de aprendizaje automático que aprenden las características desde un bajo nivel (esencial) hasta un nivel superior (abstracto) de los datos de entrada original, las cuales forman una representación que puede ser explotada de manera efectiva en una tarea de aprendizaje tal como clasificación o agrupamiento [Bengio, 2009].

Debido a la importancia del aprendizaje de representaciones, algunos investigadores han llegado a considerarlo como un campo del aprendizaje automático, y con la actividad científica tan intensa que ha tenido actualmente ha llegado a tener buen éxito tanto académico como en la industria [Mohamed and Hinton, 2010, Dahl et al., 2012, Mohamed et al., 2012, Seide et al., 2011].

Mientras que muchas investigaciones buscan las representaciones de forma manual, mediante fórmulas explícitas, explotando así el conocimiento humano del problema, otros estudios recientes tratan de descubrir buenas representaciones mediante el aprendizaje de éstas a través de los datos. Esto último se conoce como aprendizaje de representaciones. Al utilizar este tipo de aprendizaje se puede llegar a mejoras significativas con respecto a los modelos estándares supervisados en pruebas fuera de su dominio [Huang and Yates, 2010]. Para esto, el aprendizaje de representaciones usa un enfoque no supervisado, el cual tiene la ventaja de no requerir información de clase. Además, estas técnicas no necesitan de conocimiento a priori del problema para obtener una buena representación [Bengio, 2009].

Entre las aplicaciones más importantes en donde el aprendizaje de representaciones ha sido aplicado recientemente, se encuentran las siguientes:

- **Reconocimiento de voz y procesamiento de señal**: El reconocimiento de voz fue una de la primeras aplicaciones de las redes neuronales y con el reciente interés que ha surgido de nuevo por éstas, el aprendizaje profundo y el aprendizaje de representaciones ha tenido un fuerte impacto en esta área. Por ejemplo, Microsoft® liberó en el 2012 una nueva versión de MAVIS [Seide et al., 2011] (Microsoft Audio Video Indexing Service), el cual basa su reconocimiento de voz en aprendizaje profundo, reduciendo la tasa de error de 27.4 % a 18.5 %.
- **Reconocimiento de dígitos**: En los inicios del aprendizaje profundo uno de los problemas que se abordaron fue la clasificación de imágenes de dígitos utilizando la base de datos pública MNIST. En 2006 y 2007, finalmente se logra superar el dominio de las máquinas de soporte vectorial (SVM por sus siglas en inglés), las cuales tenían el mejor rendimiento para este problema. Actualmente, el mejor rendimiento lo tiene una red profunda de arquitectura convolucional [Schmidhuber, 2012]. Por otro lado, en cuanto al reconocimiento de imágenes de la base de datos ImageNet se ha reducido la tasa de error del 26.1 % al 15.3 % usando una red profunda [Krizhevsky et al., 2012].
- **Procesamiento de lenguaje natural**: Las aplicaciones de procesamiento de lenguaje natural se basan en el aprendizaje de representaciones distribuidas de palabras, denominado word embedding. En 2011 se aplica esta idea en conjunto con una arquitectura convolucional para desarrollar el sistema SENNA el cual iguala o sobrepasa al estado del arte en tareas de análisis sintáctico, etiquetamiento de categorías gramaticales, entre otros. La ventaja es que es mucho más veloz que otros resultados del estado del arte [Collobert et al., 2011].

2.3. Razones para utilizar el aprendizaje de representaciones

Algunas razones o circunstancias específicas para utilizar el aprendizaje de representaciones se presentan a continuación [Bengio et al., 2013]:

1. Usualmente, los sistemas de aprendizaje automático utilizan cuidadosamente las características o representaciones de los datos de entrada diseñadas *ad hoc* (esto

debido a que existen muchas personas que cuentan con enorme experiencia para el diseño de características). Esto consume mucho tiempo e incluso puede llegar a ser susceptible a errores e incompletitud. Por lo tanto, una opción viable es aprender buenas características que se obtengan de los propios datos.

2. La necesidad de representaciones distribuidas es deseable, generalizar localmente requiere de ejemplos representativos para todas las posibles variantes.

3. Otra propiedad importante es que el aprendizaje de características se realiza de manera no supervisada. Actualmente, las aplicaciones de aprendizaje automático más prácticas requieren de grandes cantidades de datos de entrenamiento etiquetado, sin embargo, en la vida real, es poco probable obtenerlos.

4. Aprender distintos niveles de representación es otra propiedad fundamental. Este aprendizaje está inspirado en la biología del cerebro humano, ya que los seres humanos aprenden primero los conceptos más simples y después otros más complejos. Por lo tanto, se pueden compartir componentes en una arquitectura profunda, es decir, cada capa se utiliza para una representación más especializada.

5. El aprendizaje multi-tarea también es deseable. Las arquitecturas profundas aprenden buenas representaciones intermedias que se pueden compartir a través de distintas tareas. Esto es, representaciones que extraen factores subyacentes de variación tienen sentido para muchas tareas, debido a que cada tarea se refiere a un subconjunto de estos.

2.4. Arquitecturas profundas

Se puede decir que las redes neuronales de una o dos capas ocultas son *poco profundas* y se encuentran limitadas en cuanto a las funciones que pueden representar, como las funciones de muchas variables. Es por eso que surgió la necesidad de estudiar algoritmos para entrenar modelos profundos con varios niveles de abstracción, con el fin de encontrar mejores representaciones de dichas funciones.

A la composición de muchas capas de componentes adaptativos no lineales se le denomina arquitectura profunda [Bengio, 2009]. Estas arquitecturas permiten la representación de una extensa familia de funciones de manera más compacta que las arquitecturas poco profundas. Con las arquitecturas profundas se busca representar funciones simples de manera sencilla, las cuales son extremadamente díficiles de representar con una aquitectura

poco profunda. El uso de una arquitectura inadecuada hace que estas funciones simples sean muy complicadas de aprender y por lo tanto, se necesite una enorme cantidad de datos para poder aproximarlas eficazmente.

La idea central de las arquitrecturas profundas consiste en el uso de una etapa de entrenamiento previo ayudado por una heurística ávida sin supervisión, para aprender una jerarquía de características en un nivel a la vez. Esto se realiza utilizando aprendizaje de características para aprender una nueva transformación en cada nivel, la cual es integrada con la transformación aprendida previamente en el nivel anterior. Cada iteración del aprendizaje de características no supervisado agrega una capa de pesos a una red neuronal profunda. Finalmente, el conjunto de capas utilizado se puede combinar para inicializar un predictor supervisado profundo.

Los modelos utilizados normalmente para el entrenamiento de una arquitectura profunda son [Bengio, 2009, Bengio et al., 2013, Schmidhuber, 2015, Arnold et al., 2011]:

- Máquinas de Boltzmann restringidas
- Auto-codificadores
- Redes neuronales convolucionales

2.4.1. Máquinas de Boltzmann restringidas

La máquina de Boltzmann restringida (MBR) es uno de los modelos que más se utilizan en las arquitecturas profundas. Este tipo de arquitecturas forman parte de los modelos basados en energía, en los que se asocia un valor escalar (denominado energía) a distintas configuraciones de las variables analizadas de un problema. Esta energía puede ser entendida como una medida de compatibilidad entre las variables. En general, se asocia un valor pequeño al escalar de energía para representar una alta compatibilidad entre las variables y un valor elevado para configuraciones de variables que son altamente incompatibles [Smolensky, 1986].

El aprendizaje en este tipo de modelos consiste principalmente en encontrar una función de energía, tal que para configuraciones correctas de las variables analizadas la energía sea mínima.

Una MBR es una red neuronal recurrente de conexiones simétricas, cuyas neuronas son activadas estocásticamente, y permite aprender regularidades complejas presentes en los datos de entrenamiento. La MBR, cuenta con ciertas restricciones en sus conexiones, y se forma por una capa de neuronas (binarias) ocultas y otra capa de neuronas visibles, que pueden tener valores binarios o reales. Además, las conexiones entre ambas capas son simétricas y no se permiten conexiones intracapas (es decir, entre unidades del tipo visible-visible u oculta-oculta). De esta manera se forma un grafo bipartito (ver figura 2.4).

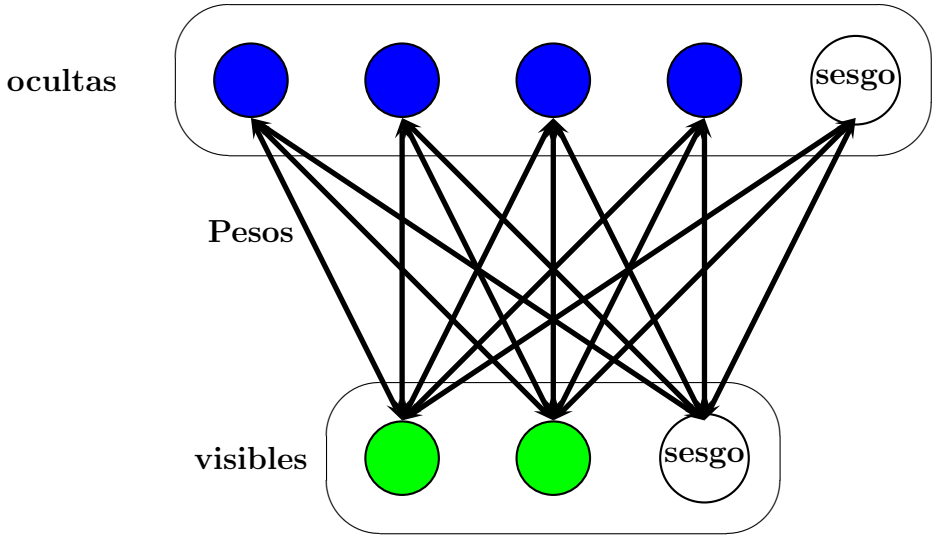

Figura 2.4: Máquina de Boltzmann restringida

Formalmente, una MBR es un modelo generativo basado en una función de energía $E(v, h)$ sobre todas las neuronas de la red, y por la cual se asigna una probabilidad a cada posible par de vectores visibles y ocultos:

$$p(v, h) = \frac{1}{Z} e^{-E(v,h)}$$

donde Z es la función de partición dada por la suma sobre todos los posibles pares de vectores v y h, v es una neurona visible y h es una neurona oculta

$$Z = \sum_{v} \sum_{h} e^{-E(v,h)},$$

la probabilidad asignada por la red a un vector visible v, se obtiene calculando la distribución marginal del vector oculto h.

$$p(v) = \sum_{h} p(v, h) = \frac{1}{Z} \sum_{h} e^{-E(v,h)}$$

y la función de energía es:

$$E(v,h) = -\sum_{i=1}^{N_v}\sum_{j=1}^{N_h} W_{ij}v_i h_j - \sum_{i=1}^{N_v} a_i v_i - \sum_{j=1}^{N_h} b_j h_j$$

Donde W_{ij} denota el peso entre la i-ésima neurona visible (v_i) y la j-ésima neurona oculta (h_j). b y a son el sesgo de las neuronas ocultas y visibles respectivamente. N_v y N_h representan el número de neuronas visibles y ocultas respectivamente. La red asigna un valor de probabilidad con la función de energía para cada estado en las neuronas visibles y ocultas.

La probabilidad de que la red clasifique a un vector de unidades visibles $p(v)$ se puede aumentar mediante el ajuste de los pesos y sesgos para reducir la energía de ese vector y así aumentar la energía de los otros. La derivada de la probabilidad logarítmica de un vector de entrenamiento con respecto al peso es calculada de la siguiente forma:

$$-\frac{\partial log p(v)}{\partial w_{ij}} =< v_i h_j >_{data} - < v_i h_j >_{model} \tag{2.1}$$

donde $< . >_{data}$ es la distribución esperada de los datos y $< . >_{model}$ es la distribución esperada del modelo.

Esto lleva a una sencilla regla de aprendizaje:

$$\delta w_{ij} = \alpha(< v_i h_j >_{data} - < v_i h_j >_{model}) \tag{2.2}$$

donde α es la tasa de aprendizaje, de igual manera la regla de aprendizaje para los sesgos es:

$$\delta a_i = \alpha(< v_i >_{data} - < v_i >_{model})$$

$$\delta b_j = \alpha(< h_j >_{data} - < h_j >_{model})$$

Debido a que no hay conexiones directas entre neuronas ocultas, entonces las neuronas ocultas son independientes de las neuronas visibles. Entonces, dado un ejemplo de entrenamiento seleccionado aleatoriamente, el estado binario h_j para cada neurona oculta j es puesto en 1 si su probabilidad es:

$$p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij}) \tag{2.3}$$

donde σ es la función logística sigmoide. Entonces, $< v_i h_j >_{data}$ puede ser calculada

fácilmente.

De igual manera, debido a que no existe una conexión directa entre las neuronas visibles:

$$p(v_i = 1|h) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{2.4}$$

Sin embargo, calcular $< v_i h_j >_{model}$ es muy difícil porque no puede ser calculado analíticamene (consumiría tiempo computacional exponencial). Un método que puede simplificar esta tarea es usar el muestreo de Gibbs (ver figura 2.5). Este asigna un vector de entrenamiento a las unidades visibles y actualiza los estados de las neuronas ocultas (ver figura 2.6).

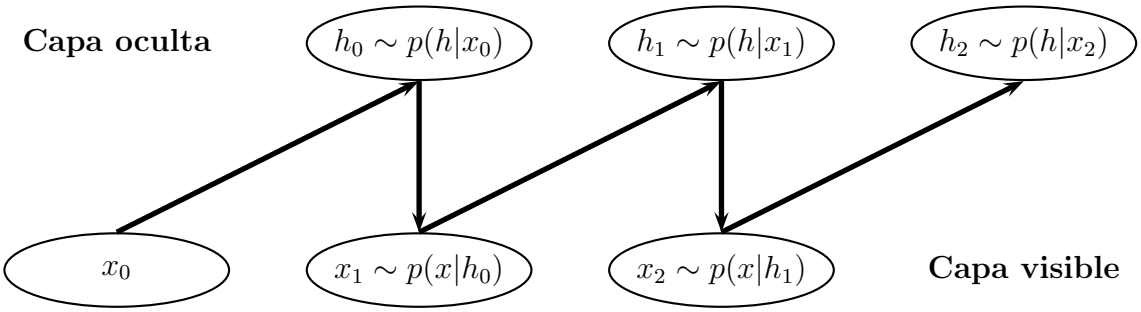

Figura 2.5: Muestreo de Gibbs

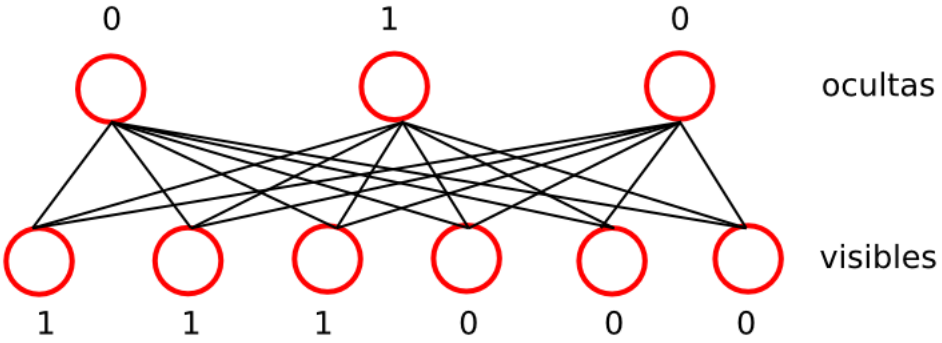

Figura 2.6: Asignación de valores mediante el muestreo de Gibbs

Debido a que este muestreo es lento, es necesario modificar el procedimiento. Esta modificación está inspirada en el método Constrastive Divergence (CD, [Bengio, 2009]), el cual se basa en una aproximación del logaritmo de la verosimilitud del gradiente de los parámetros del modelo, a través de una cadena de Markov. Ésta comienza con el último ejemplo visto y en donde la transición es un paso del algoritmo de Gibbs. Debido a

que la convergencia se da en muchos (infinitos) pasos que consisten en calcular $p(h|v)$ y después la reconstrucción $p(v|h)$, al entrenar una MBR se debe aplicar una versión más rápida que CD denominada CD-k, en donde k es el número de pasos de muestreo de Gibbs.

El algoritmo de entrenamiento para las MBR fue propuesto por Hinton [Hinton, 2002] (ver algoritmo en la sección A.5), el cual se basa en la iteración del algoritmo de muestreo de Gibbs [LeCun et al., 2006].

Debido a las restricciones del algoritmo se puede inferir en paralelo y de manera sencilla los valores h_j dado el vector de entrada v, es decir, hallar $p(h|v)$, ya que los h_j son condicionalmente independientes dado v; y de manera análoga, es fácil reconstruir v a partir de h, calculando $p(v|h)$.

Aunque ya se puede entrenar a la MRB, aún no es suficiente para obtener representaciones significativas debido a las limitaciones de estas redes en cuanto a lo que pueden representar. Para poder evitar tales limitaciones se pueden "apilar" varias MBR formando con esto una red de creencia profunda (Deep Belief Network). Esta red es un modelo generativo de varias capas, donde cada capa contiene un conjunto de neuronas ya sea con valores binarios o continuos y están conectadas con todas la neuronas de las capas adyacentes (superior o inferior), pero no entre ellas. Al final de la pila se agrega una capa que transformará la información de la última capa oculta con las neuronas de salida (ver figura 2.7).

El principio de entrenamiento no supervisado ávido por capa puede ser aplicado a las MBR (ver algoritmo en la sección A.7) apiladas y se realiza de la siguiente forma:

1. Entrenar la primer capa como una MBR que modele la entrada original $x = h(0)$ como su capa visible.

2. Usar esta primer capa para obtener una representación que se usará como datos de entrada para la segunda capa. Dado que una MBR no contiene en su estructura una salida inherente, se tienen dos opciones para la asignación de esta representación. El conjunto de datos elegido es generalmente escogido como las activaciones promedio $p(h^{(1)} = 1|h^{(0)})$ o las muestras de $p(h^{(1)}|h^{(0)})$.

3. Entrenar la segunda capa como una MBR independiente, tomando los datos transformados (muestras o activaciones) como ejemplos de entrenamiento (para la capa visible de esa MBR).

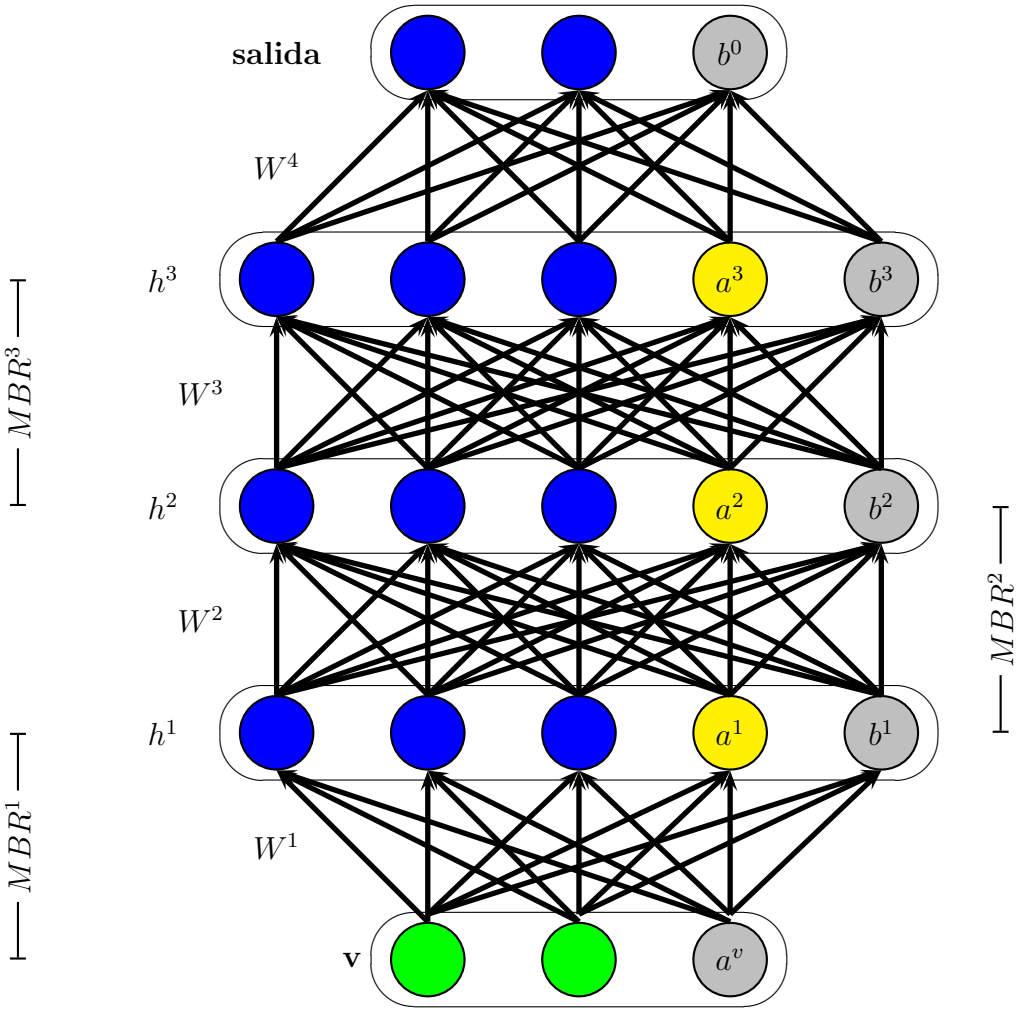

Figura 2.7: Máquina de Boltzmann restringida apilada

4. Repetir pasos 2 y 3 de acuerdo al número deseado de capas, propagando hacia arriba, ya sea las muestras o las activaciones.

5. Realizar un procedimiento de ajuste fino a todos los parámetros de esta arquitectura profunda respecto a una función de aproximación del logaritmo de la verosimilitud de la DBN, o con respecto a un criterio de aprendizaje supervisado. Para la realización de esto último, es necesario añadir un algoritmo extra en la arquitectura de la red que utilice la representación aprendida para llevar a cabo predicciones supervisadas.

Las entradas para las máquinas de Boltzmann restringidas pueden ser binarias (Bernoulli-Bernoulli) o reales (Gaussian-Bernoulli) [Cho et al., 2013]:

Para entradas binarias que se modelan mediante distribuciones Bernoulli-Bernoulli la

energía se calcula de la siguiente forma:

$$E(v,h,\theta) = -\sum_{i=1}^{I}\sum_{j=1}^{J} w_{ij}v_i h_j - \sum_{i=1}^{I} b_i v_i - \sum_{j=1}^{J} a_j h_j$$

Para entradas reales modeladas con distribuciones Gaussian-Bernoulli la energía es calculada como:

$$E(v,h,\theta) = -\sum_{i=1}^{I}\sum_{j=1}^{J} w_{ij}v_i h_j - \frac{1}{2}\sum_{i=1}^{I}(v_i - b_i)^2 - \sum_{j=1}^{J} a_j h_j$$

Debido a la estructura específica de la MBR, las unidades ocultas y visibles son condicionalmente independientes dada una u otra, por lo tanto las probabilidades condicionales para entradas binarias son:

$$P(h_j = 1|v;\theta) = \sigma(a_j + \sum_{i=1}^{I} W_{ij}v_i) \tag{2.5}$$

$$P(v_i = 1|h;\theta) = \sigma(b_i + \sum_{j=1}^{J} W_{ij}h_j) \tag{2.6}$$

y para entradas reales son:

$$P(h_j = 1|v;\theta) = \sigma(a_j + \sum_{i=1}^{I} W_{ij}v_i)$$

$$P(v_i = 1|h;\theta) = \mathcal{N}(b_i + \sum_{j=1}^{J} W_{ij}h_j, 1)$$

Para este último caso la media es $b_i + \sum_{j=1}^{J} W_{ij}h_j$ y la varianza es unitaria.

2.4.2. Auto-codificador

Un auto-codificador es una red neuronal que aprende a producir las salidas de ésta exactamente como la información que recibe en las entradas. Es decir, las capas de entrada y salida siempre deben tener el mismo número de neuronas [Baldi, 2012]. La parte importante se desarrolla en la capa oculta al proponer un auto-codificador que contenga menos neuronas en esta capa que en las capas de entrada y salida. Dado que se supone que esta red produce a la salida el mismo resultado que recibe a la entrada, y la infor-

mación tiene que pasar por la capa oculta, entonces la red se verá obligada a encontrar una representación intermedia de la información en su capa oculta usando menos neuronas. Por tanto, al suministrar un ejemplo de entrada, la capa oculta generará una versión comprimida de la información, pero además dicha versión comprimida se puede volver a descomprimir para recuperar la versión original en la salida de la red.

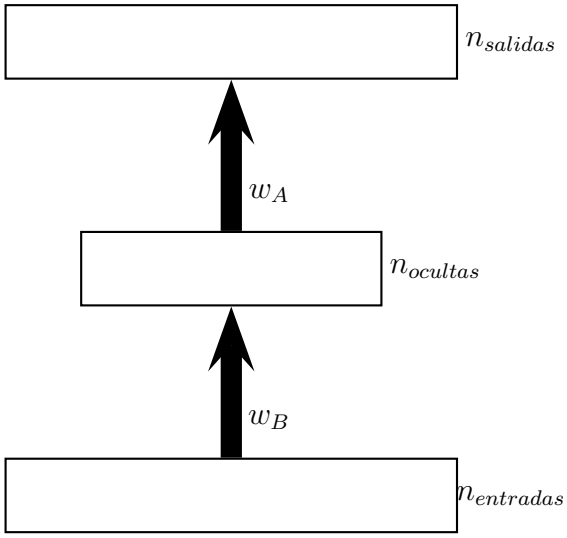

Figura 2.8: Arquitectura de un auto-codificador

Un auto-codificador típico se muestra en la Fig. 2.8, donde $n_{entradas}$ indica el número de neuronas visibles, $n_{ocultas}$ el número de neuronas ocultas y $n_{salidas}$ el número de neuronas de salida del auto-codificador. w_A y w_B son los pesos asociados.

Una vez entrenada, se puede dividir la red en dos, una primera red que utiliza la capa oculta como capa de salida, y una segunda red que utiliza esa capa oculta como capa de entrada. La primera red sería un compresor, y la segunda un descompresor. Por lo anterior, este tipo de redes se denominan auto-codificadores, los cuales son capaces de descubrir por sí mismos una forma alternativa de codificar la información en su capa oculta, sin necesitar a un supervisor que les muestre ejemplos de cómo codificar dicha información.

Un auto-codificador toma como entrada un vector x y se hace un mapeo (utilizando un codificador) a una representación y por medio de una matriz de pesos sinápticos y un sesgo:

$$y = \sigma(Wx + b)$$

Después y es mapeada de vuelta (utilizando un decodificador) a una reconstrucción z de la misma forma y tamaño que x a través de una transformación similar:

$$z = \sigma(W^T y + b')$$

Se puede decir que z es una predicción de x dado el código obtenido en y.

Los parámetros W, b, b' son optimizados de tal forma que el error de reconstrucción promedio sea minimizado. Este error puede ser medido de muchas formas, una de ellas es el error cuadrático $L(x, y) = ||x - y||^2$, otro ejemplo sería la entropía cruzada si la entrada es vista como vectores con probabilidades binarias $L_H(x, y) = -\sum_{k=1}^{d}[x_k * log(z_k) + (1 - x_k)log(1 - y_k)]$ donde d es la dimensión del vector de entrada.

Básicamente lo que se busca es que la y obtenida sea una representación distribuida que capture las coordenadas sobre los factores principales de variación en los datos.

Cuando hay más neuronas ocultas que neuronas visibles, entonces, es necesario utilizar el método de gradiente descendente para obtener representaciones útiles (ver algoritmo en la sección A.10).

Una de las principales desventajas de los auto-codificadores es que funcionan bien para los ejemplos de entrenamiento, pero no para entradas arbitrarias (test).

Auto-codificadores ruidosos

Para evitar la desventaja de que el auto-codificador funcione bien sólo para los ejemplos de entrenamiento, es necesario entrenar al auto-codificador de tal forma que reconstruya la entrada desde una versión deformada de ella. Entonces el auto-codificador ruidoso realiza dos cosas: trata de codificar la entrada e intenta deshacer el efecto de corrupción en los datos. Para mayores detalles del algoritmo correspondiente ver sección A.8.

Auto-codificadores ruidosos apilados

Los autocodificadores ruidosos pueden ser apilados para formar una red de arquitectura profunda simplemente tomando la representación de la capa oculta del auto-codificador ruidoso ubicado en la capa intermedia como entrada de la capa actual (ver figura 2.9).

El preentrenamiento no supervisado de los auto-codificadores ruidosos apilados es hecho

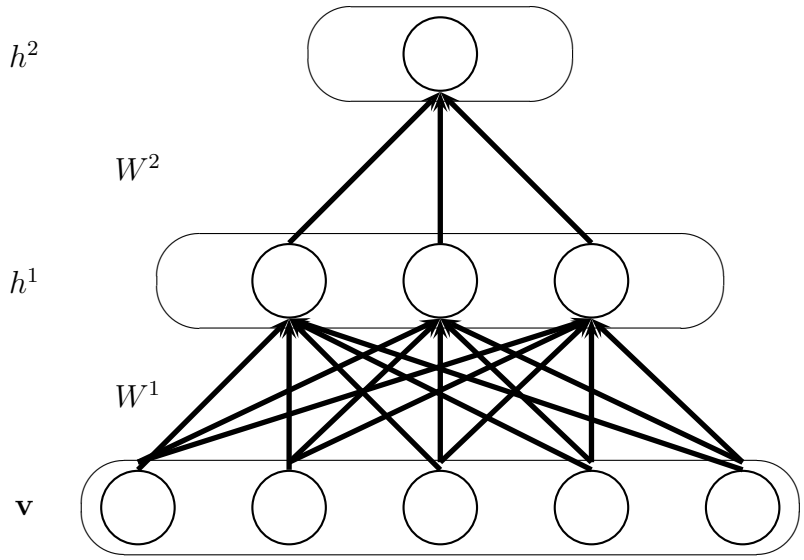

Figura 2.9: Auto-codificador apilado

utilizando una capa a la vez. Cada capa es entrenada como un auto-codificador ruidoso minimizando el error de reconstrucción de su entrada (la cual es el código de salida de la capa anterior). Una vez que las primeras k capas son entrenadas, se puede realizar el entrenamiento de la capa $k+1$ debido a que hasta ese momento se hace posible el cálculo del código o representación oculta de la capa previa. Para mayor información del algoritmo ver sección A.9.

Una vez entrenado el auto-codificador ruidoso apilado, se puede utilizar un entrenamiento supervisado que minimice el error de predicción. Para lograr esto es necesario agregar una capa de regresión logística a la última capa de la red, la cual contiene la información necesaria de las clases. Para mayor información del algoritmo ver sección A.11.

2.4.3. Redes neuronales convolucionales

Los modelos convolucionales corresponden principalmente a una variación del perceptrón multicapa con más de una capa oculta, aunque con ciertas restricciones en la unión entre nodos. Esta red se compone de capas de convolución que efectúan una operación basada en un filtro lineal de los datos que reciben de entrada, y de capas de submuestreo que permiten aumentar el poder de generalización del modelo [LeCun and Bengio, 1998] (ver figura 2.10).

Toda neurona se define como $n(l, m, j)$, en donde l es la capa, m es el mapa y j es el

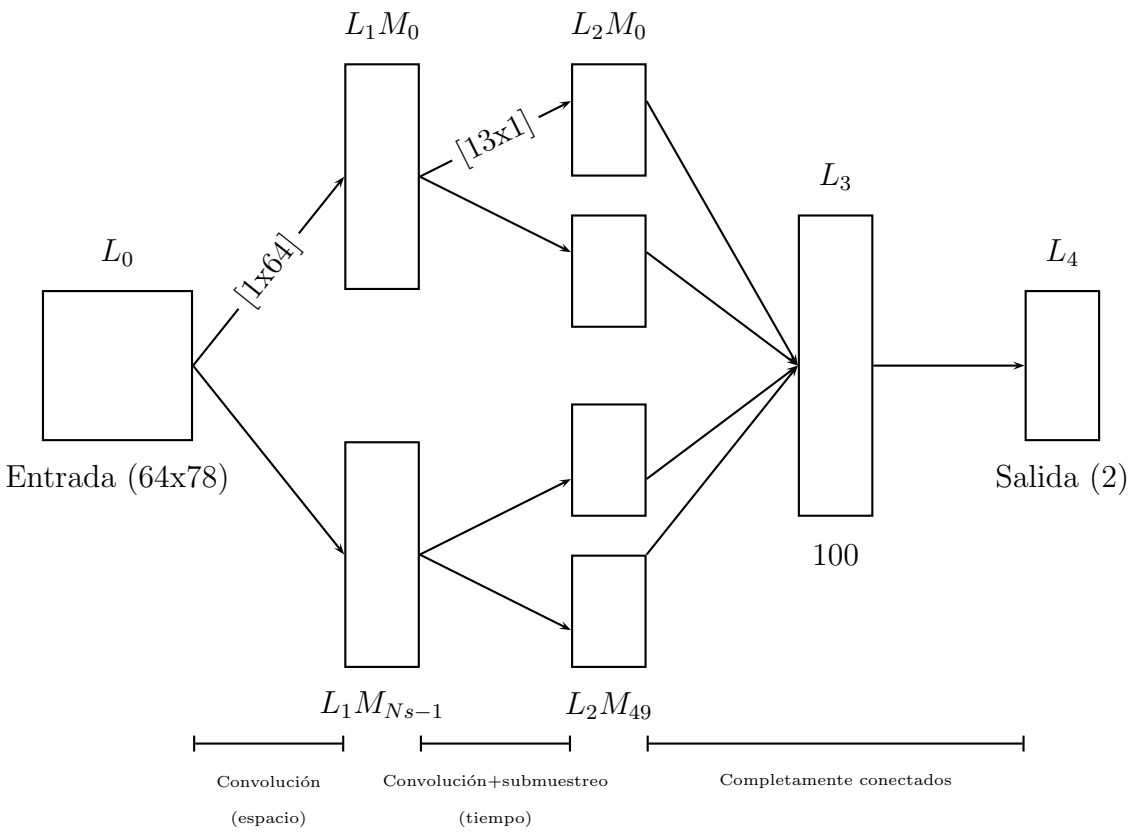

Figura 2.10: Arquitectura de una red neuronal convolucional

número de neurona en el mapa. El valor de cada neurona está dado por:

$$x_m^l(j) = f(\sigma_m^l(j))$$

donde f es la función de transferencia que depende de la capa y puede ser diferente en cada una de ellas, por ejemplo:

$$f(\sigma) = 1{,}7tanh(\frac{2}{3}\sigma)$$

$$f(\sigma) = \frac{1}{1 + exp^{-\sigma}}$$

El valor de σ representa el producto escalar entre las entradas y el peso de las conexiones entre las neuronas involucradas, y se define para todas las capas.

Este tipo de redes normalmente se usa para el reconocimiento de objetos y letras escritas a mano, aunque también puede funcionar, dependiendo de su topología, como extractor de características.

Esta arquitectura permite explotar tres propiedades del modelo:

- Capacidad de extraer características de los datos de entrada: cada neurona obtiene sus entradas desde un campo receptivo en la capa anterior, esto permite extraer características locales.

- Mapa de características: cada capa de la red está compuesta de múltiples mapas de características, donde cada mapa es un plano de neuronas las cuales tienen la restricción de compartir los mismos pesos. Esto permite agregar ventajas de invarianza a la posición de cada rasgo y una reducción del número de parámetros del modelo.

- Submuestreo: A cada capa de convolución le sigue una capa que realiza un promedio de una sub-región de la entrada desde la capa de convolución y realiza una multiplicación con los pesos de la capa para finalmente pasar por una función de activación sigmoidal. Esto tiene el efecto de reducir la sensibilidad del mapa de características desde donde provienen los datos ante efectos de ruido y distorsión.

Con estos modelos se realizaron los primeros ejemplos de arquitecturas profundas que tuvieron éxito al lograr una buena generalización de entradas visuales. Estos también son los mejores métodos para el reconocimiento de objetos [Jarrett et al., 2009].

Las MBR convolucionales (CRBM) son similares a las MBR pero los pesos entre las neuronas ocultas y visibles son compartidos entre todas las localidades de una imagen. Una CRBM consiste de dos capas: Una capa de entrada V y una capa de salida H. La capa de entrada consiste de una matriz de $N_V \times N_V$ neuronas binarias. La capa oculta consiste de k grupos, donde cada grupo es una $N_H \times N_H$ matriz de neuronas binarias. Lo que significa que habrá $N_H^2 k$ neuronas ocultas. Cada uno de los k grupos es asociado con $N_W \times N_W$ filtros, en donde W son los filtros asociados a cada grupo, los pesos de los filtros son compartidos a través de todas las neuronas ocultas en el grupo. En resumen, cada grupo oculto tiene un sesgo b_k y todas las neuronas visibles comparten un solo sesgo c [Lee et al., 2009, Huang et al., 2012].

Aquí la función de energía se define como:

$$E(v,h) = -\sum_{k=1}^{K}\sum_{i,j=1}^{N_H}\sum_{r,s=1}^{N_W} h_{i,j}^k W_{r,s}^k v_{i+r-1,j+s-1} - \sum_{k=1}^{K} b_k \sum_{i,j=1}^{N_H} h_{i,j}^k - c\sum_{i,j=1}^{N_V} v_{i,j}$$

$$E(v,h) = -\sum_{k=1}^{K} h^k \cdot ((W^k)^T * v) - \sum_{k=1}^{K} b_k \sum_{i,j=1}^{N_H} h_{i,j}^k - c \sum_{i,j=1}^{N_V} v_{i,j}$$

donde $*$ es el operador de convolución.

Como en la MBR, se puede realizar el muestreo de Gibbs usando las siguientes distribuciones de probabilidad:

$$P(h_{i,j}^k = 1|v) = \sigma(((W^k)^T * v)_{i,j} + b_k) \tag{2.7}$$

$$P(v_{i,j} = 1|h) = \sigma((\sum_k W^k * h^k)_{i,j} + c) \tag{2.8}$$

Para mayores detalles del algoritmo ver sección A.12 .

Para aprender representaciones de alto nivel, se apilan CRBMs en una arquitectura multicapa similar a un DBNs. Esta arquitectura está basada en un operador denominado $max - pooling$.

En general, los detectores de características de alto nivel, necesitan información de las regiones de entrada cada vez más grandes. Existen representaciones invariantes a la traslación, que obtienen las redes convolucionales, las cuales involucran dos tipos de capas: Capas de detección y capas de agrupamiento. La capa de detección se calcula mediante la convolución de un detector de características con la capa anterior. La capa de agrupamiento se encarga de reducir la representación de la capa de detección por un factor constante. Más específicamente, cada neurona en la capa de agrupamiento calcula la máxima activación de las neuronas en una pequeña región de la capa de detección. Entonces, reduciendo la representación con $max - pooling$, se permite que las representaciones de las capas superiores sean invariantes a pequeñas traslaciones, reduciendo la carga computacional.

Las capas de detección y agrupamiento tienen k grupos de neuronas y cada grupo de la capa de agrupamiento tiene $N_p \times N_p$ neuronas binarias. Para cada $k \in 1, 2, \cdots, K$ la capa de agrupamiento P^k reduce la representación de la capa de detección H^k por un factor de C a lo largo de cada dimensión, donde C es un entero pequeño (2 o 3). Es decir, la capa de detección H^k es particionada en bloques de tamaño $C \times C$, y cada bloque α está conectado a exactamente una neurona binaria P_α^k en la capa de agrupamiento, donde $N_P = \frac{N_H}{C}$, se define $B_\alpha = \{(i,j) : h_{ij} \text{ pertenece al bloque } \alpha\}$ (ver figura 2.11).

Las neuronas de deteción en el bloque B_α y la neurona P_α están conectadas en una sola potencia que hace cumplir las siguientes restricciones: a lo más una de las neuronas de

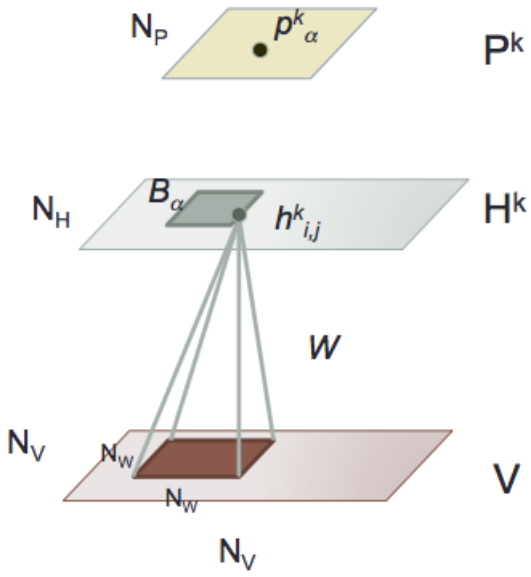

Figura 2.11: MBR Convolucional

detección puede estar prendida y la unidad de agrupamiento está prendida si y sólo si una unidad de deteccción está prendida. Equivalentemente, se puede considerar a estas C^2+1 neuronas como una sola variable aleatoria la cual puede tomar uno de C^2+1 posibles valores: $+1$ para el caso donde todos los nodos en el bloque están apagados.

Similar a la función de energía de una MBR, la función de energía de una MBR convolucional se define por:

$$E(v,h) = -\sum_{k}\sum_{i,j} h^k_{i,j}((W^k)^T * v)_{i,j} - c\sum_{i,j} v_{i,j}$$

sujeto a

$$\sum_{(i,j)\in B_\alpha} h^k_{i,j} \leq 1, \forall k, \alpha$$

Para realizar el muestreo de la capa de detección H y la capa de agrupamiento P dada la capa visible V, k grupos reciben la siguiente señal de abajo hacia arriba de la capa V:

$$I(h_{ij}k) = b_k + ((W^k)^T * v)_{ij}$$

Se muestrea cada bloque independientemente como una función multinomial de sus entradas. Supóngase h^k_{ij} como una neurona oculta contenida en el bloque α, el aumento de

la energía causado por encender la neurona h_{ij}^k es $-I(h_{ij}^k)$, y la probabilidad condicional está dada por:

$$P(h_{i,j}^k = 1|v) = \frac{e^{I(h_{ij}k)}}{1 + \sum_{(i',j' \in B_\alpha)} e^{I(h_{i',j'}^k)}}$$

$$P(p_\alpha^k = 0|h) = \frac{1}{1 + \sum_{(i',j' \in B_\alpha)} e^{I(h_{i',j'}^k)}}$$

Red de creencia profunda convolucional (CDBN)

Finalmente, se puede definir la red de creencia profunda convolucional, análogamente a las DBNs, la CDBN consiste de varias CRBMs apiladas una sobre otra. La red define una función de energía sumando las funciones de energía de todos los pares de capas. El entrenamiento usado es el mismo que se sigue para una DBN, en donde cada una de las capas es entrenada, sus pesos son congelados y sus activaciones son usadas como entrada en la siguiente capa.

2.5. Algunas representaciones convencionales de secuencia de aminoácidos

Las proteínas realizan una serie de funciones que regulan actividades celulares y fisiológicas en los organismos vivos. Las propiedades funcionales de las proteínas dependen de su estructura tridimensional. Por esto, la estructura nativa de una proteína se puede determinar experimentalmente utilizando cristalografía de rayos X, espectrocopía de resonancia magnética nuclear (RMN), ó microscopía electrónica, entre otras. Sin embargo, descifrar la estructura tridimensional de una proteína a partir de su secuencia de aminoácidos es un objetivo en la Biología Molecular y Computacional [Gromiha, 2010].

Las secuencias de proteínas se forman por combinaciones de 20 tipos de compuestos químicos diferentes, los cuales se conocen como aminoácidos y sirven como bloques para construir proteínas. Un ejemplo de secuencia es la siguiente:

$$LSIMAG \ldots AYSSITH$$

Existen 20 aminoácidos naturales como se puede ver en el cuadro 2.1, los cuales se muestran categorizados de acuerdo a residuos hidrofóbicos e hidrofílicos [Gromiha, 2010].

Cuadro 2.1: Aminoácidos nativos

Hidrofóbicos	Hidrogeno	Glicina	G
	Alifáticos	Alanina	A
		Valina	V
		Leucina	L
		Isoleucina	I
	Aromáticos	Fenialanina	F
		Tirosina	Y
		Triptófano	W
	con Azufre	Cisteína	C
		Metionina	M
Hidrofílicos	con Carga negativa	Ácido Aspártico	D
		Ácido Glutámico	E
	con Carga positiva	Histidina	H
		Lisina	K
		Arginina	R
	Polares	Asparginina	N
		Glutamina	Q
		Prolina	P
		Serina	S
		Trionina	T

Un problema relevante para la Biología Computacional y Bioinformática es la clasificación de proteínas. La clasificación en familias o clases y éstas en tipos y subtipos puede contribuir al avance en el diseño de fármacos y en una mejor comprensión de los procesos moleculares implicados en la señalización del receptor, tanto en condiciones normales como patológicas [Cruz-Barbosa et al., 2015].

Actualmente, hay una fuerte necesidad de métodos eficaces y fiables para el análisis de datos de secuencias de proteínas. Los métodos existentes se basan principalmente en la alineación y comparación de secuencias basadas en similitud. Considerando el análisis sobre los patrones y perfiles comunes, se puede tomar en cuenta que de manera implícita la estructura y función de las proteínas están determinadas mayormente por las propiedades físico-químicas de los aminoácidos presentes en su secuencia.

Las secuencias muy pequeñas o similares pueden alinearse manualmente, sin embargo, los problemas más comunes e interesantes deben alinear secuencias muy grandes y

distintas entre ellas, por lo tanto es difícil aplicar dicha forma de alineamiento. Una forma de lograr la alineación de secuencias grandes es utilizando programación dinámica [Smith and Waterman, 1981], sin embargo, el tiempo de cálculo y la memoria requerida aumentan exponencialmente conforme al tamaño. Para esto, resulta más conveniente usar enfoques heurísticos [Altschul et al., 1997], ya que reducen el tiempo para encontrar buenas alineaciones, aunque no necesariamente sean las óptimas.

Los métodos existentes para la clasicación de proteínas utilizan diferentes características de éstas para lograrla. El éxito para obtener una adecuada clasificación depende de la representación de las secuencias. Esto se debe a que las diferentes representaciones pueden excluir y ocultar los diferentes factores explicativos de la variación detrás de los datos.

Las representaciones son necesarias porque ayudan a mejorar el desempeño de las tareas de clasificación o agrupamiento. En el caso de las secuencias de aminoácidos, estas representaciones se llevan a cabo mediante la transformación de la secuencia original de tal forma que esta pueda ser explotada de una manera más efectiva [Bengio, 2009].

La correcta transformación de los datos (representación) hace que sea más fácil la extracción de información útil que será utilizada posteriormente por clasificadores o predictores [Bengio et al., 2013].

2.5.1. Composición de aminoácidos

La composición de aminoácidoses es el modelo más simple para representar una proteína [Cruz-Barbosa et al., 2015, ur Rehman and Khan, 2011, König et al., 2013].

Dada una secuencia de aminoácidos

$$P = [R_1 R_2 \cdots R_L]$$

en donde R_i representa el i-ésimo residuo de la proteína P, la composición de aminoácidos se obtiene de la siguiente forma:

$$P' = [f_1, f_2, \cdots, f_{20}]^T$$

en donde f_i representa la frecuencia de ocurrencia de cada uno de los 20 aminoácidos naturales (ver cuadro 2.1).

Una de las ventajas de esta transformación es que es muy sencilla de implementar y fácil de comprender.

Por otro lado, el principal problema que presenta, es que se pierde la información del orden de la secuencia, y además se pueden obtener resultados iguales o similares para secuencias distintas. Para consultar el pseudocódigo de ésta ver la sección A.1.

2.5.2. Pseudo-Composición de aminoácidos

La Pseudo-Composición de Amino Ácidos (PseAAC) se ha usado en el estudio de diversos problemas y sistemas relacionados con proteínas, tal como: la predicción de la localización subcelular de la proteína [ur Rehman and Khan, 2011].

Para evitar la pérdida de la información del orden de la secuencia, la pseudo-composición agrega factores adicionales que incorporan información de dicho orden a través de diferentes modos.

En este método la transformación puede ser formulada de la siguiente manera:

$$PseAAC = [P_1, P_2, \cdots, P_{20}, \cdots, P_\Lambda]^T$$

donde $\Lambda < L$ (L es la longitud de la secuencia) y además $\Lambda = 20 + n * \lambda$ (λ es el número de niveles usados en PseAAC, $\lambda = 0, 1, \cdots, m$, m es el número máximo de niveles y n es el número de propiedades fisico-químicas usadas). Los primeros 20 elementos $(P_1, P_2, \cdots, P_{20})$ son la frecuencia de ocurrencia los 20 aminoácidos naturales. El resto de los elementos $P_{21}, P_{22}, \cdots, P_\Lambda$ son los factores de correlación del primer al λ-nivel a lo largo de la cadena.

Estos últimos elementos se basan en propiedades fisicoquímicas como hidrofobicidad, hidrofilicidad, masa, etc. En el caso de la hidrofobicidad existen algunas escalas, dentro de las más importantes se encuentran KDH [Kyte and Doolittle, 1982], MH [Mandell et al., 1997], FH [Fauchere and Pliska, 1983], aunque FH es la más discriminativa.

Para esta transformación, es necesario calcular los factores de correlación τ_k de los k-ésimos niveles entre todos los k-ésimos residuos más contiguos [ur Rehman and Khan, 2011, Liu et al., 2012] (ver figura 2.12),

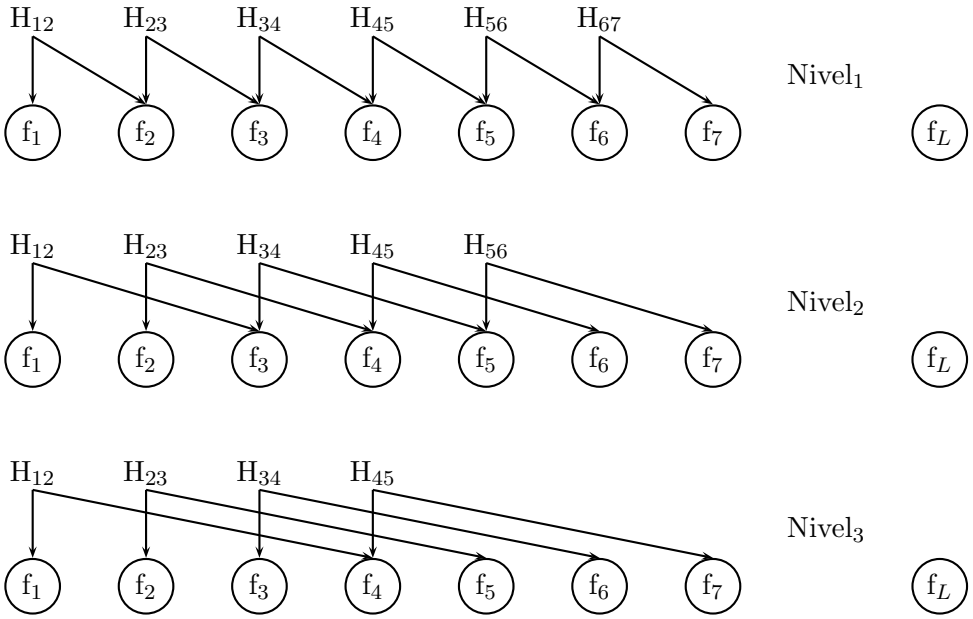

Figura 2.12: Niveles de correlación del orden de la secuencia de proteína

$$\tau_\lambda = \frac{1}{L-\lambda}\sum_{i=1}^{L-\lambda} H_{i,i+\lambda} \tag{2.9}$$

con

$$H_{i,i+k} = \frac{1}{\Gamma}\sum_{q=1}^{\Gamma}[\Phi_q(R_{i+k}) - \Phi_q(R_i)]^2$$

en donde $\Phi_q(R_i)$ es la q-ésima función del aminoácido R_i, y Γ es el número total de funciones consideradas. Entonces, la transformación está dada por:

$$P_u = \begin{cases} \frac{f_u}{\sum_{i=1}^{20} f_i + w\sum_{j=1}^{\lambda}\tau_j} & (1 \le u \le 20) \\ \frac{w\tau_{u-20}}{\sum_{i=1}^{20} f_i + w\sum_{j=1}^{\lambda}\tau_j} & (20+1 \le u \le 20+\lambda) \end{cases} \tag{2.10}$$

Una de las ventajas de esta transformación es que evita la pérdida de la información del orden de la secuencia. En contraste, la interpretación y entendimiento del modelo no es sencillo. Para mayores detalles de la implementación de esta transformación ver la sección A.2.

2.5.3. Wavelet basado en energía multiescala y PseAAC

Un vector de características híbrido es formado combinando energía multiescala y PseAAC (MSE-PseAAC). La transformación wavelet discreta (DWT) es una representación de una señal usando una base ortonormal que consiste de un conjunto infinito de wavelet discretas. Las wavelets son ortogonales y normalizadas para tener una energía unitaria [ur Rehman and Khan, 2011].

Existen muchos métodos para implementar DWT, uno de ellos es el algoritmo Mallat [Mallat, 1989]. La idea básica consiste en representar la wavelet madre como un conjunto de bancos de filtros pasa-baja y pasa-alta. La señal es pasada a través del banco de filtros y decrementada por un factor de 2. La salida del filtro pasa-baja son coeficientes de aproximación. La salida del filtro pasa-alta son coeficientes de detalle de la wavelet (normalmente ruido). En el caso de las señales internas de las proteínas los componentes de baja frecuencia son funcionalmente más importantes. Para llevar a cabo esta transformación primero se realiza lo siguiente:

1. La secuencia es convertida a una forma numérica usando valores hidrofóbicos.
2. Se usa la escala FH para calcular estos valores.
3. Cada uno de los aminoácidos es reemplazado por su correspondiente valor en la escala FH.

El resultado de esta forma numérica es homóloga a una señal digital, por lo tanto es posible aplicar DWT. Los coeficientes de aproximación y detalle son calculados. El nivel de descomposición depende del tamaño de la secuencia, se obtiene calculando Log_2 de la longitud de la secuencia. El vector de características global formado de esta manera se denomina como energía multiescala (MSE).

$$MSE(k) = \left[d_1^k, d_2^k, \cdots, d_m^k, a_m^k\right]$$

d_j^k es la raíz cuadrada de la media de la energía de los coeficientes de detalle, y a_m^k es la raíz cuadrada de la media de la energía de los coeficientes de aproximación.

$$d_j^k = \sqrt{\frac{1}{N_j} \sum_{n=0}^{N_m-1} \left\{u_j^k(n))\right\}^2}$$

$$a_m^k = \sqrt{\frac{1}{N_j}\sum_{n=0}^{N_m-1}\left\{V_m^k(n)\right\}^2}$$

donde N_j es el número de coeficientes de detalle, N_m es el número de coeficientes de aproximación, $u_j^k(n)$ es el n-ésimo coeficiente de detalle en la j-ésima escala y $V_m^k(n)$ es el n-ésimo coeficiente de aproximación en la m-ésima escala, donde la escala significa el nivel de descomposición. Como resultado, la transformación final consiste en concatenar los resultados de PseAAC y MSE :

$$MSE - PseAAC = \left[P_1, P_2, \cdots, P_{20}, \cdots, P_\Lambda, \lambda_1^k, \lambda_2^k, \cdots, \lambda_{m+1}^k\right]$$

donde $P_1, P_2, \cdots, P_\Lambda$ es el vector de características de PseAAC, el resto $\lambda_j^k = d_j^k$ y $\lambda_{m+1}^k = a_m^k$ corresponden al vector de características de MSE.

Al igual que PseAAC, esta transformación evita la pérdida de la información del orden de la secuencia, con la dificultad de su interpretación y entendimiento. Para mayores detalles de la implementación de esta transformación ver la sección A.3.

2.5.4. Auto-covarianza y covarianza cruzada

Esta transformación en un principio toma las secuencias primarias de aminoácidos y las convierte en vectores con valores reales llamados descriptores, los cuales están basados en propiedades fisicoquímicas, seguido por una transformación de los datos en una matriz uniforme.

La transformación ACC [Lapinsh et al., 2002, König et al., 2013, Opiyo and Moriyama, 2007, Liu et al., 2011] contiene dos tipos de variables: auto-covarianza y covarianza cruzada. La **auto covarianza** mide la correlación del mismo descriptor (d) entre dos residuos separados por un intervalo, lg, a lo largo de la secuencia. La **covarianza cruzada** mide la correlación de dos descriptores diferentes entre dos residuos separados por un intervalo a lo largo de la secuencia [Cruz-Barbosa et al., 2015].

La auto covarianza se define como:

$$AC_d(lg) = \frac{\sum_{j=1}^{L-lg}(S_{d,j} - \overline{S_d})(S_{d,j+lg} - \overline{S_d})}{(L - lg)}$$

donde L es la longitud de la secuencia, $S_{d,j}$ es el valor del descriptor d de un aminoácido

en la posición j, $\overline{S_d}$ es el promedio del descriptor d a través de toda la secuencia.

$$\overline{S_d} = \sum_{j=1}^{L} \frac{S_{d,j}}{L}$$

de tal forma que el número de variables de AC se puede calcular como $5 * LG$ para un intervalo máximo LG, esto es, $lg = 1, 2, 3, \cdots, LG$.

La covarianza cruzada se calcula de la siguiente manera:

$$CC_{d_1,d_2}(lg) = \frac{\sum_{j=1}^{L-lg}(S_{d_1,j} - \overline{S_{d_1}})(S_{d_2,j+lg} - \overline{S_{d_2}})}{(L - lg)}$$

en donde d_1, d_2 son dos descriptores diferentes, $\overline{S_{d_i}}$ es el promedio del descriptor d_i a través de toda la secuencia.

Los términos AC y CC son concatenados por cada intervalo (lag) $C(lg) = [AC(lg)\, CC(lg)]$, finalmente la transformación ACC se obtiene concatenando los términos $C(lg)$ para un intervalo máximo, lg_{max}, esto es: $ACC(lg_{max}) = [C(lg_1)C(lg_2), \cdots, C(lg_{max})]$.

Al igual que las transformaciones anteriores, ésta es independiente (libre) del procedimiento de alineamiento, lo cual permite tomar en cuenta toda la información presente en la secuencia. Además, las dependencias de orden entre posiciones de residuos vecinos pueden ser modeladas a través de esta. Para mayores detalles de la implementación de esta transformación ver la sección A.4.

Capítulo 3

Desarrollo del proyecto

En este capítulo se presenta una descripción del los requerimientos de hardware, el entorno de desarrollo utilizado y los módulos necesarios para desarrollar la biblioteca de aprendizaje de representaciones. Además de los conjuntos de datos utilizados para las pruebas en los experimentos, en particular, se presenta un experimento realizado para identificar dígitos manuscritos con la finalidad de conocer la arquitectura profunda que obtiene la mejor representación implícita de los datos.

3.1. Especificaciones de Hardware y Sotfware

Los experimentos del presente proyecto de tesis se realizaron en una computadora de escritorio con un procesador intel I7 a 2.8 GHz y 16 Gb en RAM. La plataforma utilizada fue el sistema operativo Ubuntu 14.02 (64 bits).

El lenguaje de programación utilizado para el desarrollo del software fue C++, específicamente el compilador g++ versión 4.2 del sistema operativo Ubuntu. Este lenguaje permite realizar una biblioteca y aprovechando las capacidades de la programación orientada a objetos de C++, esta biblioteca podría permitir la agregación de nuevas funcionalidades.

3.2. Módulos del proyecto

El proyecto consiste en que dada una entrada de datos (secuencia de aminoácidos), se obtiene inicialmente una representación o transformación con cada una de las arquitecturas profundas (auto-codificador, máquina restringida de Boltzamnn, redes convolucionales).

Posteriormente, cada una de éstas son introducidas a un clasificador para evaluar el rendimiento utilizando dicha transformación y se selecciona la mejor representación que haya capturado las características intrínsecas de los GPCR's de la clase C (ver figura 3.1).

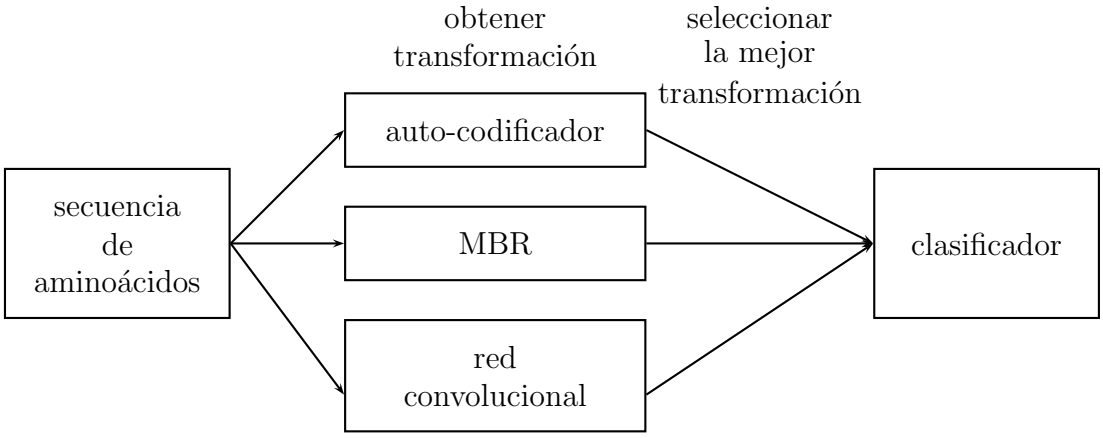

Figura 3.1: Diagrama de bloques del proyecto para obtener transformaciones usando arquitecturas profundas.

Por otro lado y con el fin de comparar la representación obtenida con las arquitecturas profundas, se realizan experimentos usando transformaciones directas de secuencias de aminoácidos (AAC, ACC, PseaAAC, Wavelet-PseAAC, ver sección 2.5) y se evalúa el rendimiento usando un clasificador (ver figura 3.2).

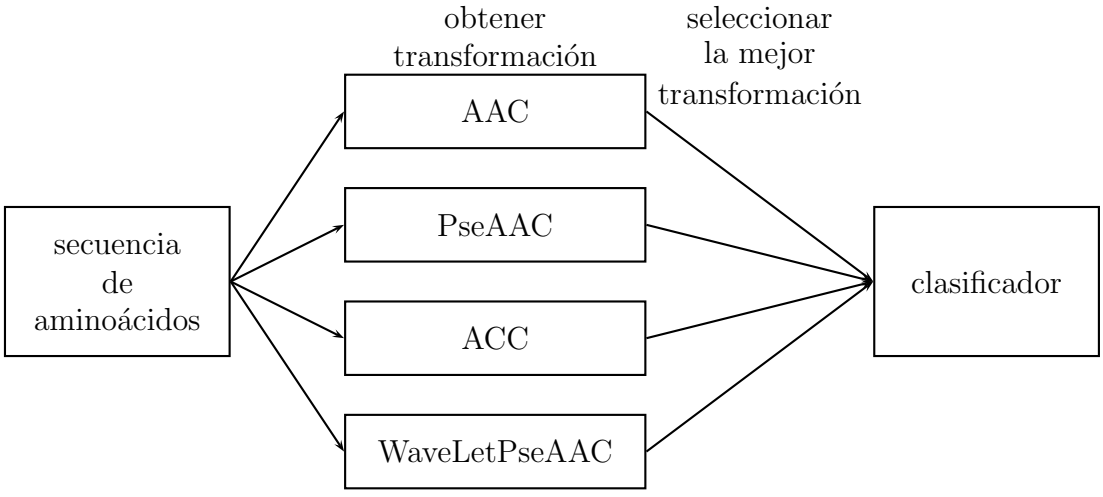

Figura 3.2: Diagrama de bloques para medir rendimiento utilizando transformaciones AAC, ACC, PseAAC, Wavelet-PseAAC.

A continuación se describen los módulos usados para el desarrollo de la biblioteca de

aprendizaje con arquitecturas profundas.

3.2.1. Auto-codificadores

Para el aprendizaje de representaciones utilizando arquitectura profunda de auto-codificadores se crearon las clases que se muestran en la figura 3.3.

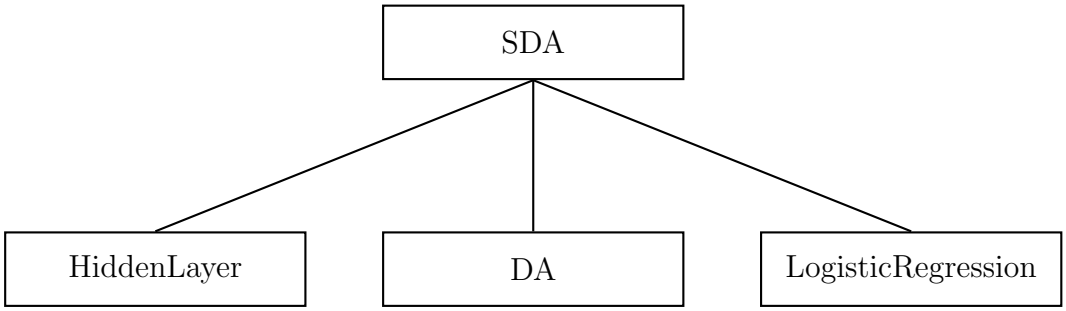

Figura 3.3: Diagrama de clases del auto-codificador

Las descripciones de cada clase se pueden ver en el cuadro 3.1.

Cuadro 3.1: Clases del auto-codificador profundo

Clase	**Descripción**
SDA	Clase que define un autocodificador ruidoso apilado.
DA	Clase que define un autocodificador ruidoso.
HiddenLayer	Clase que que define una capa oculta.
LogisticRegression	Clase que define un perceptrón multicapa que ayuda para la clasificación final.

La definición de cada clase del auto-codificador se puede consultar en la sección B.1.

3.2.2. Máquinas de Boltzmann restringidas

Para la arquitectura profunda con máquinas de Boltzmann restringidas se crearon las clases mostradas en la figura 3.4.

Los detalles de cada clase se pueden ver en el cuadro 3.2.

La definición de cada clase de la DBN se puede consultar en la sección B.2.

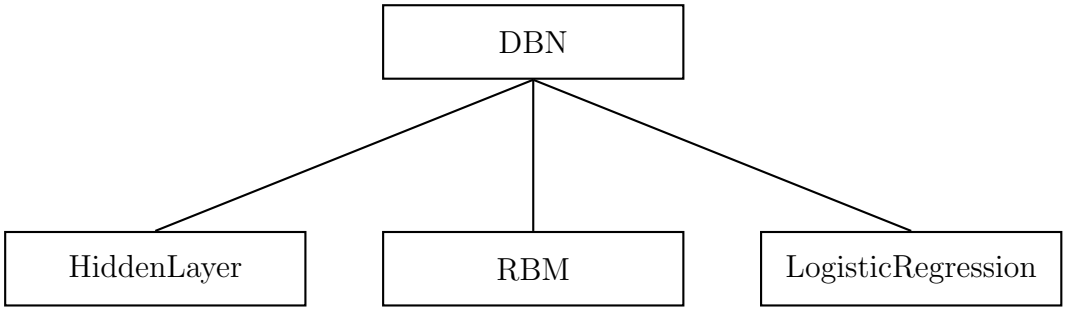

Figura 3.4: Diagrama de clases de la DBN

Cuadro 3.2: Clases de la arquitectura de máquinas de Boltzmann restringidas

Clase	**Descripción**
DBN	Clase que define una red de creencia profunda.
RBM	Clase que define una máquina de Boltzmann restringida.
HiddenLayer	Clase que define una capa oculta.
LogisticRegression	Clase que define un perceptron multicapa que ayuda para la clasificación final.

3.2.3. Redes Convolucionales Profundas

Para la arquitectura de redes convolucionales profundas (CDBN) se crearon las clases que se presentan en la figura 3.5.

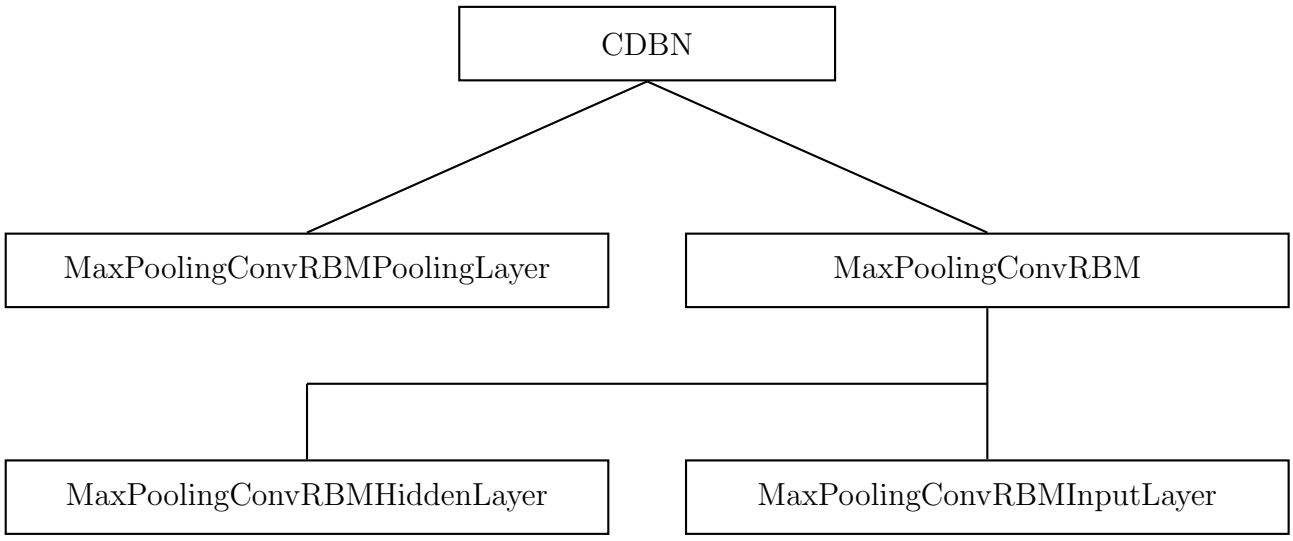

Figura 3.5: Diagrama de clases de la CDBN

Las descripciones de cada clase se pueden ver en el cuadro 3.3.

Cuadro 3.3: Clases de la arquitectura convolucional profunda

Clase	**Descripción**
CDBN	Clase que define una red de creencia profunda convolucional.
MaxPoolingConvRBMInputLayer	Clase que define la capa de entrada.
MaxPoolingConvRBM	Clase que define las capas ocultas de CDBN.
MaxPoolingConvRBMHiddenLayer	Clase que define la capa oculta a entrenar.
MaxPoolingConvRBMPoolingLayer	Clase que define la capa de agrupación que está sobre la capa oculta.

La definición de cada clase de la CDBN se puede consultar en la sección B.3.

3.3. Ejemplo de aprendizaje de representaciones de dígitos

Una vez desarrollados los módulos que componen la biblioteca de aprendizaje de representaciones con arquitecturas profundas, se procede a probarla con un problema de clasificación de dígitos manuscritos. Por lo cual, en esta sección el objetivo es realizar un experimento para encontrar la arquitectura profunda que obtiene la mejor representación de los datos.

Base de datos MNIST

La base de datos MNIST[1] contiene imágenes de dígitos escritos a mano con una resolución de 28 x 28 píxeles, obteniéndose un vector fijo de 784 características por cada ejemplo de entrada. MNIST cuenta con un conjunto de entrenamiento de 60,000 imágenes y un conjunto de puebas de 10,000. La distribución de cada uno de los dígitos para entrenamiento y prueba se presenta en el cuadro 3.4 .

Para calcular el rendimiento de predicción en las tres arquitecturas profundas (autocodificador, máquinas de Boltzmann restringidas y redes convolucionales), se utilizó validación cruzada (ver capítulo 4) con diferentes números de iteraciones, obteniéndose el

[1] http://yann.lecun.com/exdb/mnist/

Cuadro 3.4: Organización de MNIST

Dígito	**Entrenamiento**	**Prueba**
0	5,923	980
1	6,742	1,135
2	5,958	1,032
3	6,131	1,010
4	5,842	982
5	5,421	892
6	5,918	958
7	6,265	1,028
8	5,851	974
9	5,949	1,009

promedio de los rendimientos. Se probó con iteraciones $k = 5$, $k = 10$, $k = 15$, y $k = 20$. El mejor rendimiento se obtuvo con $k = 10$, considerando su funcionamiento en todas las pruebas.

Auto-codificadores

Se llevaron a cabo experimentos usando auto-codificadores con diferente número de capas ocultas y diferente número de neuronas en cada capa oculta. Los mejores resultados se obtuvieron utilizando 2 capas ocultas. Al usar 3 o más capas ocultas, el rendimiento empezaba a disminuir y el tiempo necesario para el pre-entrenamiento y entrenamiento aumentaba considerablemente. Los resultados de exactitud de clasificación promedio de los experimentos con dos capas ocultas y diferente número de neuronas se presentan en el cuadro 3.5. En este cuadro se puede observar que los resultados están cercanos a los reportados en la literatura [Lecun et al., 1998] pero un poco alejado de los mejores resultados obtenidos. Por lo anterior, se puede decir que para el problema de aprendizaje de representaciones de dígitos, las arquitecturas profundas utilizando auto-codificadores no son una buena opción.

Máquinas de Boltzmann restringidas

Los resultados obtenidos en los experimentos realizados con la arquitectura profunda utilizando máquinas de Boltzmann restringidas se pueden ver en el cuadro 3.6. De igual forma que en el cuadro 3.5, se puede observar que las máquinas de Boltzmann restingidas

Cuadro 3.5: Resultados de exactitud del auto-codificador usando dos capas ocultas con distinto número de neuronas ocultas

Configuración	Exactitud de clasificación
2 capas [250,200]	91.98 %
2 capas [500,500]	93.79 %
2 capas [600,500]	93.82 %
2 capas [750,700]	94.34 %

no son una buena opción para este problema. Cabe mencionar que al igual que con los auto-codificadores, se realizaron experimentos con 3, 4 y 5 capas ocultas, pero el rendimiento no mejoró.

Cuadro 3.6: Resultados de exactitud de la máquina de Boltzmann restringida usando dos capas con diferente número de neuronas ocultas

Configuración	Exactitud de clasificación
2 capas [250,200]	92.59 %
2 capas [400,300]	93.53 %
2 capas [600,500]	93.96 %
2 capas [750,700]	93.72 %

Redes convolucionales

En el caso de la arquitectura profunda con redes convolucionales se realizaron experimetos con 2, 3, 4 y 5 capas ocultas, los resultados más relevantes se pueden ver en el cuadro 3.7. Al utilizar 3 capas o más, el rendimiento empieza a disminuir.

Cuadro 3.7: Resultados de exactitud de la arquitectura profunda con redes convolucionales usando distinto número de capas ocultas

Configuración	Exactitud de clasificación
2 capas [400,400]	97.06 %
2 capas [500,500]	98.04 %
2 capas [600,600]	97.17 %

Los resultados de los experimentos anteriores muestran que las arquitecturas convolucionales son las de mejor desempeño para el problema de aprendizaje de representaciones de dígitos. Estos resultados en particular se pueden explicar debido a la naturaleza de la red para extraer información de imágenes. Dichos resultados están muy cercanos a los mejores resultados reportados en la literatura ([Lecun et al., 1998, Lauer et al., 2007]), por lo cual, se puede concluir parcialmente que la biblioteca desarrollada en este proyecto de tesis es adecuada para el aprendizaje de representaciones.

Es importante mencionar y recordar aquí que las arquitecturas profundas aprenden representaciones intrínsecas de los datos y en cada nivel van abstrayendo las características aprendidas en el nivel anterior. En el caso de los dígitos, la arquitectura convolucional puede ir mejorando los filtros [Rifai et al., 2011].

Capítulo 4

Resultados

En este capítulo se muestran los resultados obtenidos al realizar experimentos con representaciones de secuencias de aminoácidos utilizando transformaciones directas (AAC, ACC, PseAAC, Wavelet-PseAAC) para la clasificación de secuencias de aminoácidos en subfamilias de la clase C. También, se presentan los resultados de los experimentos usando las arquitecturas profundas (auto-codificador, máquinas de Boltzmann restringidas y redes convolucionales) para la clasificación de secuencias de aminoácidos con la finalidad de seleccionar la que mejor obtenga las representaciones implícitas de los datos. Finalmente, se presenta una comparativa de rendimiento entre la arquitectura profunda seleccionada y otros clasificadores.

4.1. Conjunto de datos y configuración experimental

Para las pruebas de evaluación de clasificadores utilizando transformaciones directas de secuencias de aminoácidos se utilizó la base de datos pública GPCRDB[1]. En el caso de la evaluación del aprendizaje con arquitecturas profundas para la clasificación de subfamilias de la clase C, se utilizó además de la base de datos GPCRDB, una base de datos que contiene los índices de las propiedades fisicoquímicas de aminoácidos AAIndex[2][Kawashima and Kanehisa, 2000].

Los objetivos de los experimentos son los siguientes. Primero, medir el rendimiento de clasificación de las representaciones explícitas obtenidas con las representaciones de pro-

[1]http://www.gpcr.org/7tm/
[2]http://www.genome.jp/aaindex/

teínas convencionales. Segundo, analizar cual de las 3 arquitecturas profundas obtiene la mejor representación intrínseca de las secuencias de aminoácidos, para lo cual será necesario medir el rendimiento de clasificación de las reprensentaciones obtenidas y seleccionar la mejor. Tercero, una vez seleccionada la arquitectura profunda que obtuvo la mejor representación de los datos, con ayuda de una búsqueda de malla, hallar la mejor configuración de capas ocultas que extraigan una mejor representación de los datos. Cuarto, con una búsqueda de malla, hallar los mejores índices de propiedades fisicoquímicas que extraigan una mejor representación de los datos. Finalmente, comparar el rendimiento obtenido de la arquitectura profunda seleccionada con otros clasificadores.

Base de datos GPCRDB

La base de datos GPCRDB categoriza a las secuencias de receptores acoplados a proteínas G en 5 familias: clase A (Rodopsina), clase B (Secretina), clase C (Metabotrópicos), receptores Vomeronasal y receptores del gusto con un total de $36,418$ secuencias. En esta base de datos existen dos conjuntos de la clase C de acuerdo a la versión 11.3.4 de marzo (2011): las $1,392$ secuencias de aminoácidos sin alinear (ver cuadro 4.1) y las $1,379$ secuencias de aminoácidos alineadas (ver cuadro 4.2). El interés particular por la clase C es debido a [Cruz-Barbosa et al., 2015]:

Complejidad estructural Mientras que todos los GPCR's se caracterizan por compartir un dominio común de siete membranas, responsable de la activación de la proteína β-arrestina, la mayoría de los GPCR's de clase C, incluyen además, un gran dominio extracelular: el dominio denominado "atrapa mosca" en forma de Venus y un dominio rico en cisteína que conecta a ambos.

Alta variabilidad en la longitud de la secuencia La presencia total o parcial de toda la estructura de dominio confiere una alta variabilidad en la longitud de secuencia a esta familia.

Relevancia terapeútica La participación de los GPCR's de clase C en muchos trastornos neurológicos hace que esta clase tenga un objetivo atractivo para el descubrimiento y desarrollo de fármacos.

Para poder utilizar las arquitecturas profundas o cualquier otro método de aprendizaje automático es necesario que las longitudes de las secuencias de aminoácidos sean fijas, por lo cual, generalmente, se recurre a alguna de las siguientes alternativas: 1) se utilizan las

Cuadro 4.1: Secuencias sin alinear de GPCR's de la clase C

Subfamilia	**Cantidad**
Calcium sensing	46
GABA-B	193
Metabotropic glutamate	321
Odorant	91
Phermone	372
Taste	65
Vomeronasal	304

Cuadro 4.2: Secuencias alineadas de GPCR's de la clase C

Subfamilia	**Cantidad**
Calcium sensing	45
GABA-B	186
Metabotropic glutamate	319
Odorant	91
Phermone	370
Taste	65
Vomeronasal	303

secuencias alineadas (ver cuadro 4.2), mediante un procedimiento de alineamiento múltiple, las cuales, como se mencionó anteriormente, son proporcionadas por la base de datos GPCRDB. 2) A las secuencia de GPCR's no alineadas (ver cuadro 4.1) se les aplica una transformación de las mencionadas en la sección 2.5 para poder utilizarlas.

Base de datos de índices de aminoácidos

La base de datos de índices de aminoácidos (AAIndex) [Kawashima and Kanehisa, 2000] contiene índices numéricos que representan propiedades fisicoquímicas y bioquímicas de aminoácidos y pares de aminoácidos. En la versión más reciente hay una sección, AAindex1, que contiene 544 índices importantes para la transformación de proteínas. En [Liu et al., 2012] se modifica AAindex1 eliminando datos incompletos para obtener un conjunto de 531 índices con los cuales se trabajó en los experimentos de este trabajo.

Estratificación de datos

Para tratar que en los k subconjuntos existan siempre datos de todas las clases, es ne-

cesario estratificar los datos. La estratificación consiste en repartir los datos de manera uniforme en cada subconjunto, con esto se asegura que en cada subconjunto de validación, entrenamiento y pruebas, existan elementos de todas las clases.

4.2. Medidas de evaluación de clasificadores

Validación cruzada de k iteraciones

Para evaluar los resultados de rendimiento, es necesario hacer una división de los datos: una parte para el entrenamiento y otra parte para pruebas. Se debe garantizar que la partición hecha entre los datos de entrenamiento y de pruebas sean independientes. Esta técnica es conocida como validación cruzada (cross validation) [Haykin, 1998].

El método consiste en dividir el total del conjunto de ejemplos en dos subconjuntos, se realiza un análisis de un subconjunto llamado conjunto de entrenamiento (training set) y se valida el análisis en el otro subconjunto llamado conjunto de prueba (test set).

El conjunto de entrenamiento se divide en k subconjuntos mutuamente excluyentes de aproximadamente igual tamaño, en cada iteración uno de los k subconjuntos se utiliza como datos de validación y los demás $k-1$ subconjuntos se usan como datos de entrenamiento para formar un modelo. Este proceso se repite k iteraciones, con cada uno de los posibles subconjuntos de datos de validación. El rendimiento de predicción o clasificación del método utilizado se obtiene calculando el promedio de los rendimientos de cada iteración. El rendimiento global se obtiene eligiendo el mejor modelo de los k calculados y al modelo elegido se le aplica el conjunto de pruebas.

El objetivo de la validación cruzada es estimar el nivel de ajuste de un modelo a un cierto conjunto de datos de prueba independiente de los usados para entrenar el modelo.

Matriz de confusión

Otra forma de visualizar el rendimiento de clasificación, es calculando la matriz de confusión, la cual es obtenida cuando se prueban los datos que no intervienen en el entrenamiento. Sea E el conjunto de ejemplos y G el conjunto de clases, se definen 2 funciones $cr, ce : E \to \{1, \cdots, G\}$, como $cr(x)$ y $ce(x)$ que devuelven la clase real y la clase estimada

de x, respectivamente. Entonces, la matriz de confusión C es:

$$C_{i,j} = |\{x \in E : cr(x) = i \ y \ cr(x) = j\}|$$

la ij-ésima entrada de C es el número de casos de la clase real i que han sido asignados a la clase j por el clasificador.

Exactitud

Una vez calculada la matriz de confusión, se puede realizar con esta el cálculo de la exactitud de predicción, que es la habilidad del modelo de predecir correctamente la clase de los nuevos ejemplos. Lo anterior, se obtiene calculando el porcentaje de ejemplos del conjunto de prueba que son correctamente clasificados por el modelo.

$$Exactitud = \frac{\sum_{k=1}^{G} C_{kk}}{\sum_{i,j=1}^{G} C_{ij}}$$

Coeficiente de correlación de Matthews

En el campo de la bioinformática, el coeficiente de correlación de Matthews (MCC), es recomendado como una herramienta óptima para tareas prácticas, ya que presenta un buen equilibrio entre la capacidad discriminatoria, la consistencia y el comportamiento coherente con el número de clases, conjuntos de datos no balanceados y aleatorios [Gorodkin, 2004, Cruz-Barbosa et al., 2015]. Dada la matriz de confusión, el MCC se define como:

$$MCC = \frac{\sum_{k,l,m=1}^{G} C_{kk}C_{ml} - C_{lk}C_{km}}{\sqrt{\sum_{k=1}^{G}\left[(\sum_{l=1}^{G} C_{lk})(\sum_{f,g=1 f\neq k}^{G} C_{gf})\right]}\sqrt{\sum_{k=1}^{G}\left[(\sum_{l=1}^{G} C_{kl})(\sum_{f,g=1 f\neq k}^{G} C_{fg})\right]}}$$

MCC devuelve valores entre [-1 1], 1 significa una correlación completa, clasificación perfecta. 0 significa que no hay correlación, todos los ejemplos fueron clasificados en una sola clase. -1 significa una correlación negativa, clasificación errónea extrema.

Tasa de error equilibrada

Otra métrica es la tasa de error equilibrada (BER Balanced error rate), es la media de tasa de error, da una indicación más sensible del rendimiento de un algoritmo, ya que da igual ponderación a cada una de las clases sin importar el número de ejemplos en cada

clase. Esta resulta del promedio de la proporción de las clasificaciones erróneas en cada clase. Esto es,

$$BER = \frac{1}{G} \sum_i \frac{\sum_j C_{ij} - C_{ii}}{\sum_j C_{ij}}$$

4.3. Evaluación de rendimiento de clasificación utilizando representaciones de proteínas convencionales

En este análisis los datos para las pruebas fueron obtenidos de la base de datos GPCRDB (ver sección 4.1), la cual categoriza a las secuencias de aminoácidos en 5 familias: clase A (Rodopsina), clase B (Secretina), clase C (Metabotrópicos), receptores Vomeronasal y receptores del gusto. Sin embargo, en esta tesis se tomaron en cuenta sólo las secuencias de la clase C debido a su complejidad estructural, la alta variabilidad en la longitud de la secuencia y a su relevancia terapeútica, así se se obtuvo su clasificación en subfamilias (ver cuadro 4.1).

Para medir el rendimiento de exactitud de clasificación utilizando las transformaciones convencionales (ver sección 2.5) se utiliza un percentrón multicapa (MLP) con diferentes configuraciones (número de capas ocultas y número de neuronas en cada capa oculta), el algoritmo de aprendizaje utilizado fue el de retro-propagación. Se utiliza además, la técnica de validación cruzada con 10-iteraciones para evaluar los resultados. Se eligió este clasificador porque usualmente este mismo se ocupa en la parte de entrenamiento supervisado de las arquitecturas profundas, por lo cual, permitirá realizar comparaciones posteriormente.

Para comenzar, se hicieron pruebas de validación cruzada con distinto número de iteraciones, dando a k valores de 5, 10, 15, y 20. El mejor rendimiento se obtuvo con $k = 10$. Los resultados de exactitud de clasificación promedio utilizando distinto número de neuronas en la capa oculta de un MLP son mostrados en el cuadro 4.3.

Para cada una de las transformaciones convencionales, el vector de entrada es diferente, en el caso de la AAC el vector de entrada es de longitud 20 y para la transformación PseAAC el vector de entrada es de longitud 62 usando 2 índices de propiedades fisico-

químicas y 21 niveles como se menciona en la literatura [ur Rehman and Khan, 2011]. En el caso de la transformación Wavelet-PseAAC el vector de entrada es de longitud 74. Finalmente, para la transformación ACC el vector de entrada es de longitud 325 usando 5 índices de propiedades fisicoquímicas y un lg de 13 [Cruz-Barbosa et al., 2015].

Cuadro 4.3: Resultados de exactitud en porcentaje de clasificación promedio con diferentes configuraciones de capas y neuronas ocultas de un MLP.

Transformación/Número de neuronas ocultas	[10]	[15]	[10,10]	[10,15]	[15,15]
AAC	76.62±3.70	76.50±3.99	84.75±3.27	83.75±3.73	81.40±3.48
ACC	77.22±2.96	81.00±3.36	84.80±2.76	84.23±3.80	82.34±2.94
PseAAC	77.31±3.96	80.65±3.57	85.94±2.05	85.04±3.04	83.22±3.47
Wavelet-PseAAC	79.22±4.57	81.72±2.79	84.89±2.50	84.54±3.11	80.85±3.46

En el cuadro 4.3 cada columna contiene una configuración diferente de capas ocultas y número de neuronas por cada capa oculta, por ejemplo [10,15] representa a 2 capas ocultas, la primera con 10 neuronas y la segunda con 15.

En el cuadro 4.3 se puede notar que la transformación PseAAC con dos capas ocultas, cada una con 10 neuronas, es la que obtuvo el mejor rendimiento de exactitud. También, se observa que la transformación AAC (bastante sencilla de obtener comparada con la PseAAC) con la misma configuración tiene un buen rendimiento, a pesar de que la diferencia de rendimiento es pequeña, la desviación estándar indica que la transformación PseAAC es más estable con las diferentes configuraciones que la transformación AAC.

También, se realizaron otros experimentos con 3, 4 y 5 capas ocultas, los cuales no pudieron mejorar el rendimiento de clasificación anterior, y aumentaban el costo computacional por cada capa que se agregaba.

4.4. Evaluación de clasificación de secuencias de aminoácidos utilizando aprendizaje de representaciones

Se realizaron pruebas para medir el rendimiento en la clasificación de subfamilias de secuencias de aminoácidos de la clase C de GPCR's. Para esto se obtuvieron los datos de la

base de datos GPCRDB, en particular, las secuencias de aminoácidos alineadas (ver cuadro 4.2), debido principalmente a que las arquitecturas profundas necesitan un vector de características fijo, y por los objetivos planteados se debe tener el menor pre-procesamiento posible.

Preprocesamiento

Para poder usar las arquitecturas profundas es necesario convertir la secuencia de aminoácidos en un vector fijo de números reales. Para lograr esto, se utilizaron los diferentes índices de propiedades fisicoquímicas de aminoácidos, las cuales fueron obtenidas de la base de datos AAindex [Kawashima and Kanehisa, 2000].

El vector que se usa como entrada de la arquitectura profunda es de longitud 259, sin embargo, por cada índice que se agregue, la longitud aumenta a $259 * n$, en donde n es el número de propiedades.

Un ejemplo del preprocesamiento se puede ver en el cuadro 4.4, utilizando las propiedades fisicoquímicas de: Hidrofobicidad, Hidrofilia y distribuciones de Hidrofobicidad e Hidrofilia. Para obtener el vector de números reales, cada uno de los residuos es sustituido por los tres índices mencionados anteriormente, en caso de haber un hueco (gap), será sustituido por tres ceros.

Cuadro 4.4: Ejemplo de secuencia de aminoácidos convertida a números reales

P	E	F	T	-	W	T	D
0.44528	0.17358	0.40377	0.01887	0	0.70943	0.01887	0.02264
0.32812	1	0.375	0.46875	0	0.17188	0.46875	0.5625
0.13103	0.78621	0.46207	0.83448	0	0.56897	0.83448	0.75862

Particionamiento del conjunto de datos

Para realizar las pruebas, se particionaron los datos en dos conjuntos: entrenamiento y pruebas (ver cuadro 4.5).

El conjunto de pruebas, al igual que el conjunto de entrenamiento, fue estratificado como se muestra en el cuadro 4.6, quedando todas las clases con el mismo porcentaje de

Cuadro 4.5: Partición de los datos para entrenamiento y pruebas

Tipo	Porcentaje	Cantidad
Entrenamiento	80 %	1101
Pruebas	20 %	278

elementos.

Cuadro 4.6: Estratificación de los datos para pruebas

Clase	Cantidad
Calcium sensing (CaSR)	9
GABA-B	38
Metabotropic glutamate (mGluR)	64
Odorant (Od)	19
Phermone (Ph)	74
Taste (Ta)	13
Vomeronasal (VN)	61

Para el entrenamiento, se usó validación cruzada con $k = 10$ y se estratificaron los datos (ver cuadro 4.7).

Cuadro 4.7: Estratificación de los datos para la validación cruzada con $k = 10$

k	Entrenamiento							Validación						
	CaSR	GABA-B	mGluR	Od	Ph	Ta	VN	CaSR	GABA-B	mGluR	Od	Ph	Ta	VN
1	32	133	229	64	266	46	217	4	15	26	8	30	6	25
2	32	133	229	64	266	46	217	4	15	26	8	30	6	25
3	32	133	229	65	266	47	218	4	15	26	7	30	5	24
4	32	133	229	65	266	47	218	4	15	26	7	30	5	24
5	32	133	229	65	266	47	218	4	15	26	7	30	5	24
6	32	133	230	65	266	47	218	4	15	25	7	30	5	24
7	33	133	230	65	267	47	218	3	15	25	7	29	5	24
8	33	133	230	65	267	47	218	3	15	25	7	29	5	24
9	33	134	230	65	267	47	218	3	14	25	7	29	5	24
10	33	134	230	65	267	47	218	3	14	25	7	29	5	24

Se realizaron experimentos con las 3 arquitecturas profundas más comunes: autocodificadores, máquinas de Boltzmann restringidas (MBR) y redes convolucionales, con el objetivo de obtener las representaciones implícitas de los datos por cada una de ellas, lo cual permitirá seleccionar la arquitectura que aprenda la mejor representación. Para esto,

se utilizan arquitecturas básicas de cada uno de los modelos: 2 capas, 700 neuronas en cada capa y un índice de propiedad fisicoquímica (hidrofobicidad).

En el cuadro 4.8 se muestran los resultados de clasificación utilizando las tres arquitecturas anteriormente mencionadas. Aquí, se observa que los auto-codificadores y las redes convolucionales tuvieron rendimientos muy bajos en comparación con las máquinas de Boltzmann restringidas, ya que ninguna de estas dos superó el 80 % de rendimiento.

Los experimentos realizados con 3, 4 y 5 capas requerían un tiempo computacional elevado con respecto a los experimentos con 2 capas, además de que no hubo un aumento en el rendimiento.

Cuadro 4.8: Resultados de exactitud de clasificación promedio de las arquitecturas profundas utilizando auto-codificadores, redes convolucionales y máquinas de Boltzmann restringidas con un índice de propiedad fisicoquímica 370

Arquitectura profunda	Exactitud
Auto-codificador	75.18 %
Redes convolucionales	73.39 %
Máquina de Boltzmann restringida	90.15 %

Es importante notar que el rendimiento en los auto-codificadores es menor debido a que no manejan de manera natural los datos de entrada reales y las máquinas de Boltzmann restringidas modeladas con la distibución Gaussian-Bernoulli si lo hacen[Cho et al., 2013]. Las redes convolucionales trabajan muy bien para el reconocimiento de imágenes, sin embargo, para problemas de otra área su rendimiento disminuye.

Dados los resultados mostrados en el cuadro 4.8, para los experimentos posteriores se utilizó únicamente la arquitectura profunda con máquinas de Boltzmann restringidas (MBR), usando el algoritmo de aprendizaje de retropropagación con momento y el gradiente acelerado por el método de Nesterov [Sutskever, 2013].

Para encontrar la mejor configuración para la arquitectura profunda con MBR, se realizó la estrategia de experimentación denominada diseño factorial [Alpaydin, 2010] (comúnmente llamada búsqueda de malla) usando diferentes configuraciones (factores y niveles): número de capas ocultas (1,2,3,4,5), número de neuronas en cada capa oculta

(300,500,700). Esto con la finalidad de encontrar la configuración más adecuada en cuanto al número de capas ocultas se refiere. Entonces, el procedimiento que se utiliza consiste de los siguentes pasos [König et al., 2013]:

1. Pre-procesamiento del conjunto de datos: estandarización de los datos.
2. Dividir el conjunto de datos en 5 subconjutos estratificados y aplicar validación cruzada con $k = 5$ (5 CV) para los siguientes pasos:
   a) Usar el conjunto de entrenamiento actual para una búsqueda de malla variando los parámetros de número de capas (nc) y número de neuronas por cada capa (nnc) en un rango dado
      1) Por cada combinación de nc y nnc, determinar el rendimiento promedio de clasificación usando una validación cruzada con $k = 5$ interna y actualizar los parámetros con los mejores resultados
      2) Entrenar un modelo usando los parámetros seleccionados (nc y nnc) y el conjunto de entrenamiento actual.
   b) Clasificar el conjunto de prueba actual con el modelo obtenido en el paso anterior usando una medida de clasificación
3. Calcular el valor promedio de la medida de clasificación usada durante el paso 2*b* sobre las 5 iteraciones externas.

En el cuadro 4.9 se muestran los resultados de usar una MBR con sólo una capa oculta y un índice de propiedad fisicoquímica.

Cuadro 4.9: Resultados de exactitud promedio de la arquitectura profunda con MBR usando una capa oculta y un índice de propiedad fisicoquímica

Número de neuronas	Exactitud
300	93.67 %
500	93.41 %
700	93.48 %

Se puede observar en el cuadro 4.9 que la arquitectura profunda obtiene una mejor representación de forma implícita que la usada por las transformaciones convencionales

(85.94 % con la transformación PseAAC) (ver cuadro 4.3).

En el cuadro 4.10 se muestran los resultados de usar una búsqueda de malla para seleccionar el mejor rendimiento empleando dos capas ocultas y un índice de propiedad fisicoquímica.

Cuadro 4.10: Resultados de exactitud promedio de la arquitectura profunda con MBR usando dos capas ocultas y un índice de propiedad fisicoquímica

$Capa_1$ / $Capa_2$	300	500	700
300	93.65 %	93.16 %	93.12 %
500	93.13 %	**94.21 %**	93.02 %
700	93.12 %	93.10 %	93.03 %

Con la búsqueda de malla, se van obteniendo las representaciones implícitas de los datos en cada configuración, sin embargo, para saber cual es la mejor, es necesario utilizar un clasificador que reciba como entrada las representaciones obtenidas. Después de usar el clasificador y medir el rendimiento de cada configuración, se observa que la mejor representación la obtuvo la configuración de 500 neuronas ocultas en cada capa, lo cual se muestra en letra negrita en el cuadro 4.10. Conforme aumenta el número de neuronas en las capas ocultas, el rendimiento se estabiliza y no mejora.

Una vez llevado a cabo este experimento, es necesario realizar una nueva búsqueda de malla, ahora con 3 capas ocultas, con la finalidad de verificar si es posible obtener mejores representaciones implícitas y por lo tanto, mejorar el rendimiento de clasificación (ver cuadro 4.11). Se puede notar que hay algunas configuraciones que tienen mas del 93 % de rendimiento [700-300-300], [700-300-700], lo que significa, que las representaciones implícitas obtenidas no mejoraron el rendimiento de clasificación obtenido con 2 capas ocultas. Además, al requerir 3 capas ocultas, el entrenamiento consume una mayor cantidad de tiempo.

También, se realizaron experimentos con 4 capas ocultas (ver cuadro 4.12) y 5 capas ocultas (ver cuadro 4.13) y diferente número de neuronas en cada capa, con la finalidad de encontrar una nueva configuración que mejore el rendimiento obtenido con el experimento usando 2 capas ocultas. Sin embargo, ninguna de las combinaciones de 4 y 5 capas ocultas mencionadas anteriormente, pudo extraer mejores representaciones intrínsecas de

Cuadro 4.11: Resultados de exactitud promedio de la arquitectura profunda con MBR usando tres capas ocultas y un índice de propiedad fisicoquímica

Capas	300	500	700
300,300	88.12 %	88.05 %	87.92 %
300,500	87.87 %	87.14 %	88.21 %
500,300	87.25 %	87.92 %	87.45 %
500,500	88.03 %	90.67 %	88.24 %
Capas	700	300	500
300,700	92.52 %	92.96 %	90.33 %
700,300	**93.11 %**	92.89 %	93.12 %
700,700	88.36 %	88.92 %	89.13 %
Capas	500	700	300
500,700	92.11 %	92.25 %	92.13 %
700,500	90.07 %	88.77 %	92.92 %

los datos, aunado a que el rendimiento fue disminuyendo a medida que el número de capas ocultas aumentaba.

Cuadro 4.12: Resultados de exactitud promedio de la arquitectura profunda con MBR usando cuatro capas ocultas y un índice de propiedad fisicoquímica

Número de neuronas	Exactitud
300,300,300,300	**92.82 %**
500,500,500,500	92.25 %
700,700,700,700	91.77 %

Cuadro 4.13: Resultados de exactitud promedio de la arquitectura profunda con MBR usando cinco capas ocultas y un índice de propiedad fisicoquímica

Número de neuronas	Exactitud
300,300,300,300,300	91.44 %
500,500,500,500,500	**91.81 %**
700,700,700,700,700	91.23 %

Después de haber realizado experimentos con diferente número de capas ocultas y diferente número de neuronas en cada capa, se determinó que la mejor configuración es la

de 2 capas ocultas con 500 neuronas en cada capa.

Con el siguiente experimento se realizó una nueva búsqueda de malla, con una vecindad de neuronas seleccionadas de tal forma que las configuraciones sean cercanas a las 2 capas con 500 neuronas en cada una de ellas (ver cuadro 4.14).

Cuadro 4.14: Resultados de exactitud promedio de la arquitectura profunda con MBR usando dos capas ocultas y un índice de propiedad fisicoquímica con vecindades cercanas a la mejor configuración obtenida

$Capa_1$ / $Capa_2$	400	450	500	550	600
400	91.56 %	91.12 %	91.84 %	91,03 %	91,56 %
450	93.92 %	88.01 %	89.71 %	91.82 %	90.73 %
500	93.62 %	90.65 %	**94.21 %**	91.92 %	89.63 %
550	89.87 %	89.23 %	89.92 %	91.11 %	92.39 %
600	92.62 %	89.82 %	89.21 %	89.34 %	91.44 %

El resultado de la búsqueda de malla anterior, muestra que las combinaciones de las vecindades cercanas no logran obtener una mejor representación implícita que la obtenida con 500 neuronas en cada capa. Por lo tanto, se continúa buscando un mejor rendimiento a partir de la mejor configuración encontrada hasta el momento.

Se experimenta ahora dejando fijas 2 capas ocultas con 500 neuronas en cada capa y se agrega una tercera con un número de neuronas cercana a las 2 capas previas (ver cuadro 4.15), es decir, con un número de neuronas cercanas a las 500 que hay en las 2 capas existentes.

Sin embargo, nuevamente no se encontró una mejor representación implícita de los datos, por lo tanto, se detiene la búsqueda de una mejor configuración debido a que los rendimientos obtenidos en los experimentos de las nuevas configuraciones han empezado a disminuir. Entonces, la mejor configuración que se selecciona es la que usa dos capas ocultas con 500 neuronas cada una.

Una vez que se realizaron las pruebas con búsqueda de malla para encontrar la mejor configuración, ahora es necesario probar con los 531 índices de propiedades de aminoácidos para encontrar el que mejor ayude a representar las secuencias de aminoácidos con la arquitectura profunda seleccionada.

Cuadro 4.15: Resultados de exactitud promedio de la arquitectura profunda con MBR usando tres capas ocultas y un índice de propiedad fisicoquímica con vecindades cercanas a la mejor configuración obtenida

Capas	Rendimiento
500,500,400	**94.21 %**
500,500,450	93.77 %
500,500,500	91.11 %
500,500,550	93.62 %
500,500,600	94.09 %

Estos experimentos se realizaron con la siguiente estrategia: particionando y estratificando el conjunto de datos en dos subconjuntos, 80 % para entrenamiento y 20 % para pruebas como se puede ver en el cuadro 4.5. En el entrenamiento se usó validación cruzada con $k = 10$ quedando los datos como se muestra en el cuadro 4.7. El rendimiento general se obtiene primero, eligiendo el mejor modelo de los 10 obtenidos en la validación cruzada, y después a este modelo se le aplica el conjunto de datos de pruebas para finalmente obtener dicho rendimiento. Es importante mencionar que esta última experimentación se realizó de esta forma principalmente porque cuando este tipo de modelos son usados en aplicaciones reales (industria), es necesario usar un modelo único para la fase de reconocimiento.

Por lo anterior, los siguientes resultados se presentan usando el mejor modelo obtenido en la validación cruzada. En el cuadro 4.16 se muestran los 10 mejores resultados de exactitud de clasificación con sus índices correspondientes.

De acuerdo a la literatura [König et al., 2013, Cruz-Barbosa et al., 2015], existen 3 subfamilias de la clase C que son difíciles de clasificar: vomeronasal, phermone y odorant. Analizando las matrices de confusión de los 531 experimentos, se puede notar que los primeros índices del cuadro 4.16 obtienen representaciones significativas que logran clasificar con más del 96 % a la subfamilia vomeronasal, más del 88 % la subfamilia phermone y más del 66 % la subfamilia odorant.

La subfamilia odorant es la más complicada de clasificar, suele confundirse con las subfamilias vomeronasal y phermone, en promedio solamente clasifica correctamente al 44 % de las secencias de aminoácidos de esta subfamilia, lo que hace que el rendimiento de clasificación general disminuya.

Cuadro 4.16: Resultados de los 10 mejores rendimientos de exactitud de la arquitectura profunda con MBR usando 1 índice de propiedad fisicoquímica y 2 capas ocultas con 500 neuronas cada una.

Nombre	Índice	Exactitud
Frecuencia de residuo posicional normalizado	411	**94.60** %
AA composición de las proteínas transmembrana individuales	206	94.53 %
Energía libre transferida	358	94.16 %
Mutabilidad relativa	135	93.43 %
Composición de la superficie de los aminoácidos en las proteínas	463	93.43 %
Valor de preferencia relativa en C4	332	93.43 %
Valor de la información para la accesibilidad	10	93.43 %
Parámetro estérico Suavizado Upsilon	79	92.70 %
Energía libre en la región de la zona beta	432	91.97 %
Pesos para la hoja beta en la posicin de la ventana de -5	272	91.60 %

Para mostrar si se puede mejorar el rendimiento obtenido hasta ahora, se realizaron pruebas combinando 2 índices de propiedades fisicoquímicas, haciendo las combinaciones entre los 10 mejores índices individuales obtenidos (ver cuadro 4.17).

Cuadro 4.17: Resultados en % de los mejores rendimientos de exactitud de la arquitectura profunda con MBR usando dos índices de propiedades fisicoquímicas

Índices	411	206	358	135	463	332	10	79	432	272
411	-	93.32	91.02	92.51	90.93	90.17	90.46	91.22	90.78	89.23
206	89.95	-	94,11	95.13	91.75	90.92	90.12	90.34	92.56	91.89
358	92.01	91.39	-	91.49	91.17	92.14	92.21	92.11	91.12	91.35
135	91.61	92.04	90.2	-	91.17	91.12	91.34	90.70	90.81	90.61
463	91.12	92.19	93.54	93.92	-	90.12	91.73	91.82	91.15	91.72
332	90.12	91.12	91.89	91.72	92.04	-	91.62	91.69	91.58	91.60
10	90.10	90.12	90.20	90.52	90.00	89.93	-	89.89	89.91	90.01
79	90.92	90.23	90.29	90.25	89.93	89.91	90.02	-	89.92	89.72
432	90.09	90.12	90.56	90.82	90.11	90.00	89.87	89.69	-	89.52
272	90.28	91.17	90.61	89.12	89.09	90.08	90.20	89.48	90.23	-

Se puede notar que los 3 mejores rendimientos dependen de las propiedades fisicoquímicas 206 y 135. De hecho el mejor rendimiento fue precisamente con esa combinación [206-135]. Este resultado se debe principalmente, a la subfamilia odorant, por separado cada índice tiene un 71 % de rendimiento en esa subfamilia, sin embargo, al combinarlos, el rendimiento aumenta hasta un 76 %, lo cual provoca que se obtenga un rendimiento

general del 95.13 %.

También, se realizaron pruebas combinando 3 índices de propiedades fisicoquímicas, haciendo las combinaciones entre los 5 mejores índices individuales obtenidos (ver cuadro 4.18). Aquí se puede notar la influencia del índice de la propiedad fisicoquímica número 206, ya que el mejor resultado lo obtuvo la combinación de [206-135-358], esta combinación mejora la representación obtenida por los índices separados. Esto es, para el índice 206, el rendimiento de la subclase phermone aumenta de un 89.95 % a un 93.2 %, el rendimiento del índice 135 en la subclase vomeronasal aumenta de un 94.87 % a un 97.94 % y el rendimiento del índice 358 en la subclase phermone aumenta de un 85.20 % a un 93.48 %.

Es importante mencionar que el orden de los índices de propiedades fisicoquímicas no afecta en los rendimientos de clasificación, es decir, el experimento que toma los índices 358, 206 y 411 no varía significativamente al tomar los índices en el orden 206, 358 y 411 ó los índices 411, 358 y 206.

Cuadro 4.18: Resultados de los 10 mejores rendimientos de exactitud con la arquitectura profunda con MBR usando 3 índices de propiedades fisicoquímicas

Índices	Exactitud
206-135-358	94.78 %
206-411-358	94.21 %
358-206-411	94.12 %
206-358-135	93.91 %
206-358-463	93.87 %
206-411-79	93.72 %
358-79-206	93.37 %
206-463-79	93.31 %
358-463-135	93.12 %
358-411-135	93.05 %

Después, se realizaron pruebas combinando 4 índices de propiedades fisicoquímicas, haciendo las combinaciones entre los 6 mejores índices individuales obtenidos (ver cuadro 4.19). En este experimento, se puede obsevar que las representaciones implícitas obtenidas no ayudan a mejorar el rendimiento, al contrario, comienza a disminuir. Esto se debe principalmente a que la subfamilia vomeronasal afecta en la búsqueda de una mejor representación, disminuyendo el rendimiento general. Por lo tanto, ya no se realizan nuevos

experimientos combinando más índices de propiedades fisicoquímicas.

Cuadro 4.19: Resultados de los 7 mejores rendimientos de exactitud con la arquitectura profunda con MBR usando 4 índices de propiedades fisicoquímicas

Índices	Exactitud
206-135-358-411	94.16 %
358-135-206- 79	93.80 %
206-411-358-463	93.43 %
358-135-206-463	93.43 %
206-135-358- 79	92.70 %
358-135-206-411	92.70 %
206-135-358-463	92.34 %

Representación de secuencias mediante ventanas

Otro de los experimentos que se realizó para tratar de encontrar mejores representaciones y por lo tanto, seguir mejorando el rendimiento de clasificación, consistió en crear segmentos de ventanas que se van deslizando en la secuencia de aminoácidos, el cual fue propuesto en [Qi et al., 2014]. Inicialmente, se toman ventanas de tamaño 7, quedando tres aminoácidos del lado izquierdo, tres aminoácidos del lado derecho y el aminoácido central que define la ventana actual. Existen casos especiales en donde no habrá aminoaćidos del lado derecho o izquierdo (ver figura 4.1), entonces simplemente se ignoran las partes vacías.

Figura 4.1: Ventana resultante considerando al primer aminoácido de la secuencia como aminoácido central.

La ventana se va desplazando en intervalos de 1 (ver figura 4.2).

Por cada ventana, existe un vector de 20 elementos que representan a los 20 aminoácidos naturales (ver cuadro 2.1), cada posición del vector contiene el valor del índice de una propiedad fisicoquímica, por lo tanto, habrá como máximo 7 valores distintos de cero en tal vector, Ahora, puesto que la longitud del vector es 20, en total habrá $20 * 259 = 5,180$ características por cada secuencia de aminoácidos. En la figura 4.3 se pude ver el vector obtenido con los valores correspondientes a la ventama analizada. Este vector sólo contiene dos 1s debido a que se utilizó el índice de propiedad fisicoquímica 31, el cual, solamente

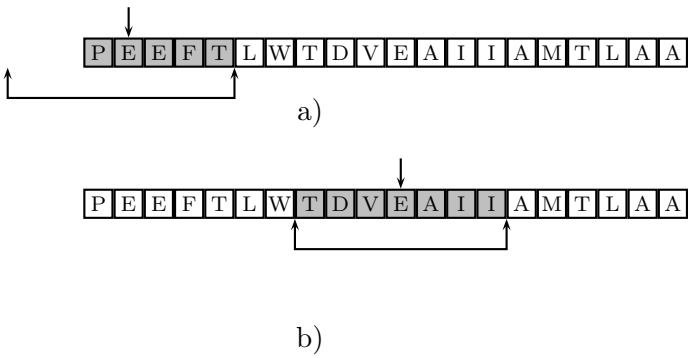

Figura 4.2: Ventanas desplazadas. a) Ventana resultante considerando un desplazamiento. b) Ventana considerando 10 desplazamientos.

tiene valores de 1s y 0s.

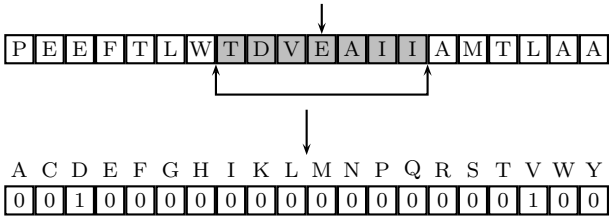

Figura 4.3: Vector resultante de la ventana analizada (área sombreada).

En el siguiente experimento se calculó el rendimiento de exactitud usando ventanas de tamaño 7 con los mejores índices obtenidos en las pruebas anteriores: individuales, pares, ternas, cuadretes (ver cuadro 4.20).

Cuadro 4.20: Resultados de los mejores rendimientos de exactitud usando ventanas de tamaño 7

Índices	Exactitud de clasificación
411	93.31 %
206-135	91.34 %
206-135-358	90.72 %
206-135-358-411	89.21 %

Las representaciones implícitas obtenidas con estas configuraciones no mejoraron el rendimiento, debido principalmente a que el uso de las ventanas hace que aumenten considerablemente el número de variables. Esto es, cuando se usa sólo un índice de propiedad

fisicoquímica se tienen 259 variables, sin representación de ventanas, pero cuando se usa una ventana de tamaño 7 con 1 índice, el número de variables crece a 5180. En el peor de los casos, con 4 propiedades fisicoquímicas el número de variables sin ventanas es de 1036 y si se requiere del uso de ventanas el número de variables crece a 20720, lo cual afecta significativamente el rendimiento de clasificación.

Comparación con otros Clasificadores

Después de haber seleccionado la mejor configuración de capas ocultas, número de neuronas por capa y los índices de propiedades fisicoquímicas con los cuales se obtuvo la mejor representación implicita de los datos, es necesario comparar el rendimiento obtenido de la arquitectura profunda seleccionada con otros clasificadores.

Máquinas de soporte vectorial

En [König et al., 2013] mediante una búsqueda de malla encontraron que los valores de $C = 2$ y $\gamma = 2^{-9}$, ayudan a obtener un buen rendimiento usando máquinas de soporte vectorial (SVM) para clasificar sequencias de aminoácidos de subfamilias de la clase C, llegando a obtener un 93 % de rendimiento.

Aplicando esos mismos valores de C y γ a una máquina de soporte vectorial y el mismo conjunto de datos analizado en este proyecto de tesis, se procede a calcular los rendimientos con las mejores combinaciones de índices de propiedades fisicoquímicas obtenidas anteriormente (ver cuadro 4.21).

Cuadro 4.21: Comparación de los resultados de rendimiento de las medidas de exactitud, MCC y BER usando SVM con C=2 y Gamma=2^{-9} y la arquitectura profunda MBR.

Índices	SVM			Arquitectura profunda		
	Exactitud	MCC	BER	Exactitud	MCC	BER
411	83.61 %	79.64 %	19.13 %	94.60 %	93.45 %	5.83 %
206-135	85.40 %	82.23 %	17.79 %	95.13 %	94.08 %	6.25 %
206-135-358	88.82 %	85.13 %	10.95 %	94.78 %	94.07 %	7.42 %
206-135-358-411	89.78 %	87.44 %	11.01 %	94.16 %	92.70 %	6.71 %

Mientras que las SVM necesitan más información en las entradas para obtener una me-

jor representación de los datos y mejorar el rendimiento de clasificación, la arquitectura profunda puede obtener representaciones significativas implícitas con menos información de entrada. Además, la tasa de error equilibrada (BER) indica que la arquitectura profunda discrimina mejor las subclases de la familia C, a pesar de que el conjunto de datos está desbalanceado.

Árboles de decisión

Los árboles de decisión (J48) fueron también usados para comparar el rendimiento de la arquitectura profunda seleccionada (ver cuadro 4.22). Los experimentos realizados con estos árboles utilizaron los mismos índices de propiedades fisicoquímicas con los que se obtuvieron los rendimientos usando las máquinas de Boltzmann restringidas.

Cuadro 4.22: Resultados de rendimiento usando índices de propiedades fisicoquímicas con árboles de decisión

Índices	Exactitud	MCC	BER
411	81.62 %	76.82 %	19.12 %
206-135	82.12 %	77.93 %	17.03 %
206-135-358	84.26 %	80.13 %	12.25 %
206-135-358-411	80.66 %	76.38 %	19.25 %

Se puede observar que los áboles de decisión no pueden lograr obtener una buena representación de los datos de manera implícita, sin embargo, logran (usando 3 índices) una discriminación de las subfamilias de la clase C similar a la que obtienen las SVM's. Estos árboles tienen la ventaja de ser fáciles de implementar.

k vecinos más cercanos

Otra técnica usada para comparar el rendimiento fueron los k-vecinos más cercanos. Los resultados obtenidos se pueden ver en el cuadro 4.23. Para este experimento se probó con diferente número de vecinos, desde $k = 1$ hasta $k = 10$.

Se puede apreciar que con un $k = 4$ con un índice de propiedad fisicoquímica y con $k = 2$ con tres índices, se obtiene los mejores rendimientos para este clasificador, sin embargo, la diferencia entre los cálculos del BER no es significativa, por lo tanto, utilizando

Cuadro 4.23: Resultados de los mejores porcentajes de rendimientos usando índices de propiedades fisicoquímicas con $k$nn

Índices												
	411			206-135			206-135-358			206-135-358-411		
k	Exac	MCC	BER	Exac	MCC	BER	Exac	MCC	BER	Exac	MCC	BER
1	89.72	87.12	8.23	87.23	84.45	11.10	89.92	87.94	8.35	88.69	86.34	9.08
2	90.23	87.91	8.61	88.69	86.19	11.12	91.35	89.96	7.99	89.78	87.53	9.31
3	91.12	89.16	7.93	89.05	86.64	10.93	90.91	88.91	8.22	89.41	87.11	9.55
4	91.31	89.45	7.93	88.32	85.71	11.36	89.83	87.61	8.88	89.78	87.55	9.26
5	90.12	87.45	9.12	87.96	85.40	11.03	91.31	89.48	7.98	87.59	84.91	10.65
6	89.80	87.45	9.44	86.86	84.04	11.69	90.12	87.88	8.69	87.59	84.91	10.65
7	89.15	86.62	9.92	86.50	83.61	11.93	88.82	86.23	9.63	87.59	84.98	10.09
8	90.00	86.62	9.91	87.23	84.51	11.50	88.70	86.11	9.47	87.59	84.98	10.09
9	89.72	86.81	9.93	86.13	83.23	12.16	88.80	87.45	9.44	87.59	84.98	10.09
10	89.41	85.91	10.38	86.50	83.67	11.91	87.59	84.98	10.09	87.59	84.98	10.09

el principio de la navaja de Occam es mejor utilizar solamente el índice de propieda 411 por requerir menos procesamiento computacional. Estos resultados muestran que k-nn no logra superar el rendimiento de clasificación de la arquitectura profunda MBR.

Es importante mencionar que la implementación de los k vecinos más cercanos es muy sencilla, pero con el incoveniente de que al aumentar el número de ejemplos, el tiempo de ejecución se ve incrementado de manera significativa. Este tipo de clasificadores son útiles cuando el número de muestras es pequeña.

A continuación, en el cuadro 4.24, se presenta el resumen de los mejores porcentajes de rendimiento de cada uno de los cuatro clasificadores analizados.

Cuadro 4.24: Resumen de porcentaje de rendimiento usando cuatro clasificadores

Clasificador	Índices	Exactitud	MCC	BER
MBR	206-135	95.13	94.08	6.25
k-nn (k=4)	411	91.31	89.45	7.93
SVM	206-135-358-411	89.78	87.44	11.01
Árboles de decisión	206-135-358	84.26	80.13	12.25

Se puede notar que a pesar de que el clasificador k-nn es un algoritmo muy sencillo y fácil de implementar, obtiene un rendimiento mejor que las máquinas de soporte vectorial, lo cual indica que para que estas últimas obtengan un mejor rendimiento, se le debe proporcionar entradas con características significativas extraídas previamente de manera

explícita (como en [König et al., 2013]). Además, estos resultados muestran que los índices de propiedades fisicoquímicas 206 y 135 estuvieron presentes en los mejores rendimientos de los tres de los cuatro clasificadores, lo cual se debe a que dichos índices, son los que mejor ayudan a discriminar a las subfamilias de la clase C.

Después de realizar todas las pruebas (encontrar la mejor configuración de la arquitectura profunda, y obtener los mejores índices de propiedades fisicoquímicas), la comparación con otros clasificadores muestra que la representación obtenida de los datos de manera implícita con la arquitectura profunda seleccionada, es mejor que las representaciones explícitas convencionales usadas en la literatura anteriormente.

Capítulo 5

Conclusiones y trabajo a futuro

En este proyecto de tesis se muestra que las arquitecturas profundas logran obtener representaciones implícitas de las secuencias de aminoácidos de las subfamilias de la clase C de GPCR's más significativas (95.13 %) que las representaciones convencionales obtenidas de manera explícita (85.94 %). Para esto, se presentó el desempeño de clasificación usando transformaciones y clasificadores convencionales comparado con tres modelos de arquitectura profunda.

Generalmente, las representaciones explícitas usadas por las transformaciones convencionales, pueden llegar a tener rendimientos muy buenos, sin embargo, obtener dichas representaciones puede ser muy complicado, y se necesita de un experto en el área para lograrlas. Por lo anterior, en este proyecto se buscó extraer representaciones implícitas de los datos analizados. Dichas representaciones han mostrado en los resultados obtenidos que han logrado extraer características significativas de los datos de una mejor manera que las representaciones convencionales, lo cual se refleja en los rendimientos altos de exactitud de clasificación y en las distintas medidas de evaluación aplicadas.

Para obtener las representaciones implícitas de las secuencias de aminoácidos de las subfamilias de la clase C se utilizaron tres modelos de arquitectura profundas. En particular, las máquinas de Boltzmann restringidas manejan mejor los datos con entradas continuas (números reales) y fueron las que obtuvieron mejores representaciones.

En los experimentos se mostró que la búsqueda de malla es muy útil cuando se necesita seleccionar parámetros. En este trabajo se utilizó para encontrar la mejor configuración con respecto a número de capas ocultas, número de neuronas en cada capa y los índices de

propiedades fisicoquímicas necesarios para obtener el mejor rendimiento de la arquitectura profunda seleccionada.

Las representaciones obtenidas por la arquitectura profunda de forma implícita son muy importantes para lograr un alto desempeño de clasificación (95.13 %), lo cual se mostró al comparar diferentes algoritmos de clasificación (91.31 %) con el mismo conjunto de datos. Además, la medida de tasa de error equilibrada, muestra que la arquitectura profunda hace una buena discriminación de subfamilias de la clase C.

De los resultados obtenidos se muestra que las subfamilias de la clase C: odorant, vomeronasal y phermone, afectan el rendimiento de clasificación porque son difíciles de identificar al confundirse entre ellas. Sin embargo, al combinar algunos índices de propiedades fisicoquímicas, se mejora el rendimiento general.

En resumen, con el uso de la búsqueda de malla y arquitecturas profundas, en particular, las máquinas de Boltzmann restringidas se pudo encontrar una configuración que logra obtener una representación significativa de los datos, lo cual ayudó a mejorar el desempeño de clasificación de las secuencias de aminoácidos de las subfamilias de la clase C de GPCR's, lográndose con esto alcanzar el objetivo general y probar que la hipótesis del presente trabajo de tesis es verdadera.

Debido a que la etapa de preentrenamiento y la técnica de búsqueda de malla consumen bastante tiempo, se deja para un futuro implementar estos procedimientos de manera paralela.

También, la importancia de las combinaciones de propiedades fisicoquímicas encontradas en este trabajo y el efecto de la asociación de las secuencias con sus ligandos (subfamilias) correspondientes debe ser validada con expertos en el área (Químicos y/o Bioquímicos).

Bibliografía

[Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, second edition.

[Altschul et al., 1997] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25:3389–3402.

[Arnold et al., 2011] Arnold, L., Rebecchi, S., Chevallier, S., and Paugam-Moisy, H. (2011). An introduction to deep-learning. In *proceedings of the 19th european symposium on artificial neural networks*, pages 477–488.

[Baldi, 2012] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In *proceedings of the international conference on machine learning unsupervised and transfer learning*, pages 37–50.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in machine learning*, 2:1–127.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35:1798–1828.

[Bhasin and Raghava, 2004] Bhasin, M. and Raghava, G. P. S. (2004). GPCRpred: an SVM-based method for prediction of families and subfamilies of G-protein coupled receptors. *Nucleic acids research*, 32:383–389.

[Bhasin and Raghava, 2005] Bhasin, M. and Raghava, G. P. S. (2005). GPCRsclass: a web tool for the classification of amine type of G-protein-coupled receptors. *Nucleic acids research*, 33:143–147.

[Boulanger-Lewandowski et al., 2012] Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *proceedings of the 29th international conference on machine learning*, pages 1159–1166.

[Cho et al., 2013] Cho, K., Raiko, T., and Ilin, A. (2013). Gaussian-Bernoulli deep Boltzmann machine. In *proceedings of the international joint conference on neural networks*, pages 1–7.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *proceedings of the 25th international conference on machine learning*, pages 160–167.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12:2493–2537.

[Cruz-Barbosa et al., 2013] Cruz-Barbosa, R., Vellido, A., and Giraldo, J. (2013). Advances in semi-supervised alignment-free classication of G protein-coupled receptors. In *proceedings of the 4th international work-conference on bioinformatics and biomedical engineering*, pages 759–766.

[Cruz-Barbosa et al., 2015] Cruz-Barbosa, R., Vellido, A., and Giraldo, J. (2015). The influence of alignment-free sequence representations on the semi-supervised classification of class C G protein-coupled receptors. *Medical & biological engineering & computing*, 53:137–149.

[Dahl et al., 2012] Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE transactions on audio, speech, and language processing*, 20:30–42.

[Fauchere and Pliska, 1983] Fauchere, J. L. and Pliska, V. (1983). Hydrophobic parameters-p of amino acid side-chains from the partitioning of N-acetyl aminoacid amide. *European journal of medicinal chemistry*, 18:369–375.

[Gorodkin, 2004] Gorodkin, J. (2004). Comparing two k-category assignments by a k-category correlation coefficient. *Computational biology and chemistry*, 28:367–374.

[Gromiha, 2010] Gromiha, M. (2010). *Protein Bioinformatics: From Sequence to Function*. Elsevier Science, first edition.

[Haykin, 1998] Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, second edition.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14:1771–1800.

[Huang and Yates, 2010] Huang, F. and Yates, A. (2010). Exploring representation-learning approaches to domain adaptation. In *proceedings of the 2010 workshop on domain adaptation for natural language processing*, pages 23–30.

[Huang et al., 2012] Huang, G. B., Lee, H., and Learned-Miller, E. (2012). Learning hierarchical representations for face verification with convolutional deep belief networks. In *proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2518–2525.

[Jarrett et al., 2009] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *proceedings of the IEEE 12th international conference on computer vision*, pages 2146–2153.

[Kawashima and Kanehisa, 2000] Kawashima, S. and Kanehisa, M. (2000). Aaindex: Amino acid index database. *Nucleic acids research*, 28:374.

[König et al., 2013] König, C., Cruz-Barbosa, R., Alquézar, R., and Vellido, A. (2013). SVM-based classification of class C GPCRs from alignment-free physicochemical transformations of their sequences. In *proceedings of the 17th new trends in image analysis and processing*, pages 336–343.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *proceedings of the 26th annual conference on neural information processing systems*, pages 1106–1114.

[Kyte and Doolittle, 1982] Kyte, J. and Doolittle, R. F. (1982). A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology*, 157:105–132.

[Lapinsh et al., 2002] Lapinsh, M., Gutcaits, A., Prusis, P., Post, C., Lundstedt, T., and Wikberg, J. (2002). Classification of G-protein coupled receptors by alignment-independent extraction of principal chemical properties of primary amino acid sequences. *Protein science*, 11:795–805.

[Lauer et al., 2007] Lauer, F., Suen, C. Y., and Bloch, G. (2007). A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40:1816–1824.

[LeCun and Bengio, 1998] LeCun, Y. and Bengio, Y. (1998). *Convolutional Networks for Images, Speech and Time Series*, chapter in The handbook of brain theory and neural networks, pages 255–258. The MIT Press.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

[LeCun et al., 2006] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M. A., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, pages 191–246.

[Lee et al., 2009] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *proceedings of the 26th international conference on machine learning*, pages 609–616.

[Liu et al., 2012] Liu, B., Wang, X., Chen, Q., Dong, Q., and Lan, X. (2012). Using amino acid physicochemical distance transformation for fast protein remote homology detection. *PloS one*, 7:e46633.

[Liu et al., 2011] Liu, X., Zhao, L., and Dong, Q. (2011). Protein remote homology detection based on auto-cross covariance transformation. *Computers in biology and medicine*, 41:640 – 647.

[Mallat, 1989] Mallat, S. G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11:674–693.

[Mandell et al., 1997] Mandell, A. J., Selz, K. A., and Shlesinger, M. F. (1997). Wavelet transformation of protein hydrophobicity sequences suggests their memberships in structural families. *Physica A*, 244:254–262.

[Marr, 1982] Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* Henry Holt and Co., Inc., first edition.

[McDonald, 2009] McDonald, J. H. (2009). *Handbook of Biological Statistics.* Sparky House Publishing, second edition.

[Mohamed et al., 2012] Mohamed, A., Dahl, G. E., and Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Transactions on audio, speech, and language processing*, 20:14–22.

[Mohamed and Hinton, 2010] Mohamed, A. and Hinton, G. E. (2010). Phone recognition using restricted boltzmann machines. In *proceedings of the 35th IEEE international conference on acoustics speech and signal processing*, pages 4354–4357.

[Opiyo and Moriyama, 2007] Opiyo, S. O. and Moriyama, E. N. (2007). Protein family classification with partial least squares. *Journal of proteome research*, 6:846–853.

[Papasaikas et al., 2003] Papasaikas, P. K., Bagos, P., Litou, Z., and Hamodrakas, S. (2003). A novel method for GPCR recognition and family classification from sequence alone using signatures derived from profile hidden markov models. *SAR and QSAR in environmental research*, 14:413–420.

[Qi et al., 2014] Qi, Y., G, S. D., Collobert, R., and Weston, J. (2014). Deep learning for character-based information extraction. In *proceedings of the 36th european conference on information retrieval*, pages 668–674.

[Rifai et al., 2011] Rifai, S., Dauphin, Y., Vincent, P., Bengio, Y., and Muller, X. (2011). The manifold tangent classifier. In *proceedings of the 25th annual conference on neural information processing systems*, pages 2294–2302.

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

[Schmidhuber, 2012] Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3642–3649.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[Seide et al., 2011] Seide, F., Li, G., and Yu, D. (2011). Conversational speech transcription using context-dependent deep neural networks. In *proceedings of the 12th annual conference of the international speech communication association*, pages 437–440.

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147:195–197.

[Smolensky, 1986] Smolensky, P. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, chapter in Information processing in dynamical systems: foundations of harmony theory, pages 194–281. MIT Press.

[Sutskever, 2013] Sutskever, I. (2013). *Training recurrent neural networks*. PhD thesis, University of Toronto, Department of Computer Science. Canada.

[ur Rehman and Khan, 2011] ur Rehman, Z. and Khan, A. (2011). G-protein-coupled receptor prediction using pseudo-amino-acid composition and multiscale energy representation of different physiochemical properties. *Analytical biochemistry*, 412:173–182.

[Vroling et al., 2011] Vroling, B., Marijn, S., Coos, B., Annika, B., Stefan, V., Jan, K., Laerte, O., de V. Jacob, and Gert, V. (2011). GPCRDB: information system for G protein-coupled receptors. *Nucleic acids research*, 39:309–319.

[Weston et al., 2005] Weston, J., Leslie, C., Ie, E., Zhou, D., Elisseeff, A., and Stafford, W. (2005). Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21:3241–3247.

Anexo A

Pseudocódigo de algoritmos utilizados

A.1. Pseudocódigo de la transformación de composición de aminoácidos

Algorithm 1 Pseudocódigo AAC

```
1:  procedure AAC(S)                          ▷ Tranformación AAC de la secuencia S
2:      L ← Longitud de S
3:      i ← 0
4:      while i < 20 do                       ▷ Limpiar el arreglo de frecuencias
5:          f[i] ← 0
6:          i = i + 1
7:      i ← 0
8:      while i < L do                        ▷ Acumula las frecuencias
9:          f[S[i] − 65] = f[S[i] − 65] + 1
10:         i = i + 1
11:     i ← 0
12:     while i < 20 do                       ▷ Calcula las frecuencias
13:         f[i] = f[i] ∗ 100/L
14:         i = i + 1
15:     return [f]                            ▷ Arreglo de frecuencias
```

A.2. Pseudocódigo de la transformación PseAAC

Algorithm 2 Pseudocódigo PseAAC

procedure PseAAC(S) ▷ Tranformación PseAAC de la secuencia S
    Inicializar f con longitud $20 + n * \lambda$
    Calcular τ_j con la formula 2.9
    Convertir cada valor P_u del atributo a su forma estándar usando la formula 2.10
    return $[f]$

A.3. Pseudocódigo de la transformación MSE-PseAAC

Algorithm 3 Pseudocódigo MSE-PseAAC

procedure MSE-PseAAC(S) ▷ Tranformación MSE-PseAAC de la secuencia S
    Covertir la secuencia original de cada residuo en su escala FH
    Calcular $MSE(k) \leftarrow \left[d_1^k, d_2^k, \cdots, d_m^k, a_m^k\right]$
    Calcular PseAAC
    return [PseAAC MSE] ▷ La concatenación de los 2 arreglos

A.4. Pseudocódigo de la transformación ACC

Algorithm 4 Pseudocódigo ACC

1: **procedure** ACC(S) ▷ Tranformación ACC de la secuencia S
2: $L \leftarrow$ *Longitud de* S
3: $j \leftarrow 0$
4: **while** $j < L$ **do** ▷ Limpiar el arreglo de frecuencias
5: $d \leftarrow 0$
6: **while** $d < L$ **do** ▷ Para cada residuo de L
7: Calcular AC(d,lg)

$$AC(d, lg) = \frac{\sum_{j=1}^{L-lg}(S_{d,j} - \overline{S_d})(S_{d,j+lg} - \overline{S_d})}{(L - lg)}$$

8: $d = d + 1$
9: $d \leftarrow 0$
10: **while** $d < L$ **do** ▷ Para cada residuo de L y $d_1 \neq d_2$
11: Calcular CC_{d_1,d_2}(lg)

$$CC_{d_1,d_2}(lg) = \frac{\sum_{j=1}^{L-lg}(S_{d_1,j} - \overline{S_{d_1}})(S_{d_2,j+lg} - \overline{S_{d_2}})}{(L - lg)}$$

12: $d = d + 1$
13: $j = j + 1$
14: Concatenar

$$C(lg) = [AC(lg)\ CC(lg)]$$

15: Calcular

$$ACC(lg_{max}) = [C(lg_1)C(lg_2), \cdots, C(lg_{max})]$$

16: **return** $[ACC(l_{max})]$

A.5. Pseudocódigo de máquina de Boltzmann restringida

Algorithm 5 Pseudocódigo RBM

procedure RBM ▷ Pseudocódigo para implementar una máquina de Boltzmann restringida
    Generar aleatoriamente la matriz W de n_v x n_h
    Generar el vector b_v con ceros
    Generar el vector b_h con ceros
    $p(h|v) = W * visibles + bias_{oculto}$ ecuación 2.5
    $muestra_{ocultas} = binomial(neuronas_{ocultas})$
    $p(v|h) = W^T * muestras_{ocultas} + bias_{visible}$ ecuación 2.6
    $muestras_{visibles} = binomial(neuronas_{visibles})$
    $costo = energia_{libre}(entrada) - energia_{libre}(muestra_{final})$ ecuación 2.1
    $parametro = parametro + tasaAprendizaje * gradiente(costo * parametro)$ ecuación 2.2

A.6. Pseudocódigo de la divergencia constractiva

Algorithm 6 Pseudocódigo divergencia_constractiva

procedure DBN::CONTRASTIVE_DIVERGENCE(entrada lr k) ▷ Pseudocódigo para implementar la divergencia constractiva
    $sample\_h\_given\_v(input, ph\_mean, ph\_sample)$
    $step \leftarrow 1$
    while $step \leq k$ **do**
        if $step = 1$ **then**
            $gibbs\_hvh(ph\_sample, nv\_means, nv\_samples, nh\_means, nh\_samples)$
        else
            $gibbs\_hvh(nh\_samples, nv\_means, nv\_samples, nh\_means, nh\_samples)$
        $step \leftarrow step + 1$
    $i \leftarrow 1$
    while $i \leq nocultas$ **do**
        $j \leftarrow 1$
        while $j \leq nvisibles$ **do**
            $W[i][j] += \frac{lr*(ph\_mean[i]*input[j]-nh\_means[i]*nv\_samples[j])}{N}$
            $j \leftarrow j + 1$
        $hbias[i] += \frac{lr*(ph\_sample[i]-nh\_means[i])}{N}$
        $i \leftarrow i + 1$
    $i \leftarrow 1$
    while $i \leq nocultas$ **do**
        $vbias[i] += \frac{lr*(input[i]-nv\_samples[i])}{N}$
        $i \leftarrow i + 1$

A.7. Pseudocódigo de red de creencia profunda

Algorithm 7 Pseudocódigo DBN

1: **procedure** PREDBN(entrada,lr,k,nepocas) ▷ Pseudocódigo para implementar el pre-entenamiento de una red de creencia profunda
2: $i \leftarrow 1$
3: **while** $i \leq numCapas$ **do**
4: $epoca \leftarrow 1$
5: **while** $epoca \leq nepocas$ **do**
6: $n \leftarrow 1$
7: **while** $n \leq nejemplos$ **do**
8: $entrenamientoX = entrada[n]$
9: $l \leftarrow 1$
10: **while** $l \leq i$ **do**
11: **if** $l = 1$ **then**
12: $capa_{entrada} \leftarrow entrenamientoX$
13: **else**
14: **if** $l = 2$ **then**
15: $tam\_capa_{entrada}Anterior \leftarrow num\_neuronas_{entrada}$
16: **else**
17: $tam\_capa_{entrada}Anterior \leftarrow tam\_capa_{oculta}[l-2]$
18: $capa_{oculta}Anteior \leftarrow capa_{entrada}$
19: $capa_{entrada} \leftarrow sample\_h\_given\_v(capa_{entrada}Anterior, capa_{entrada})$ ▷ ver ecuación 2.3
20: $n \leftarrow n + 1$
21: $l \leftarrow l + 1$ $capa_{rbm} \leftarrow contrastive\_divergence(capa_{entrada}, lr, k)$
22: $epoca \leftarrow epoca + 1$
23: $i \leftarrow i + 1$

A.8. Pseudocódigo de autocodificador ruidoso

Algorithm 8 Pseudocódigo DA

procedure DA(x,lr,corruption_level) ▷ Pseudocódigo para implementar un Autocodificador Ruidoso
  $p = 1 - corruption\_level$
  $get\_corrupted\_input(x, tilde\_x, p)$ ▷ asignar ruido a la entrada
  $get\_hidden\_values(tilde\_x, y)$ ▷ obtener el valor de las neuronas ocultas
  $get_r econstructed\_input(y, z)$ ▷ reconstruir la entrada
  $i \leftarrow 1$
  while $i \leq nvisibles$ **do**
    $L\_vbias[i] = x[i] - z[i]$
    $vbias[i] += \frac{lr*(input[i]-L\_vbias[i])}{N}$
    $i \leftarrow i + 1$
  $i \leftarrow 1$
  while $i \leq nocultas$ **do**
    $L\_hbias[i] = 0$
    $j \leftarrow 1$
    while $j \leq nvisibles$ **do**
      $L\_hbias[i] += W[i][j] * L\_vbias[j]$
      $j \leftarrow j + 1$
    $L\_hbias[i] *= y[i] * (1 - y[i])$
    $hbias[i] += \frac{lr*L\_hbias[i]}{N}$
    $i \leftarrow i + 1$
  $i \leftarrow 1$
  while $i \leq nocultas$ **do**
    $j \leftarrow 1$
    while $j \leq nvisibles$ **do**
      $W[i][j] += \frac{lr*(L\_hbias[i]*tilde\_x[j]+L\_vbias[j]*y[i])}{N}$
      $j \leftarrow j + 1$
    $i \leftarrow i + 1$

A.9. Pseudocódigo de pre-entrenamiento de autocodificador

Algorithm 9 Pseudocódigo preSDA

```
1: procedure preSDA(entrada,lr,corruption_level,nepocas)  ▷ Pseudocódigo para pre entrenar un Autocodificador
2:     i ← 1
3:     while i ≤ numCapas do
4:         epoca ← 1
5:         while epoca ≤ nepocas do
6:             n ← 1
7:             while n ≤ nejemplos do
8:                 entrenamientoX = entrada[n]
9:                 l ← 1
10:                while l ≤ i do
11:                    if l = 1 then
12:                        capa_entrada ← entrenamientoX
13:                    else
14:                        if l = 2 then
15:                            tam_capa_entrada Anterior ← num_neuronas_entrada
16:                        else
17:                            tam_capa_entrada Anterior ← tam_capa_oculta[l − 2]
18:                    capa_oculta Anteior ← capa_entrada
19:                    capa_entrada ← sample_h_given_v(capa_entrada Anterior, capa_entrada)  ▷ ver ecuación 2.3
20:                    n ← n + 1
21:                l ← l + 1
22:                dA_layers[i]− > train(layer_input, lr, corruption_level);
23:            epoca ← epoca + 1
24:        i ← i + 1
```

A.10. Pseudocódigo para calcular gradiente descendente

Algorithm 10 Pseudocódigo Gradiente Descendente

procedure GRADIENTEDES ▷ Pseudocódigo para implementar el gradiente descendente
  while $epoca < epoca_{total}$ y $salir = falso$ **do** ▷
    $epoca \leftarrow epoca + 1$
    while haya *minibatch* **do** ▷ para cada minibatch del conjunto de entrenamiento
      $perdida \leftarrow f(parametros, x_{batch})$
      $gradiente \leftarrow calcularGradiente$
      $parametros \leftarrow parametros - tasaAprendizaje * gradiente$
      if $perdida \leq objetivo$ **then**
        $salir \leftarrow verdadero$
  return *parametros*

Existen dos formas de implementar este algoritmo, el modo incremental y el modo por lotes (minibatch). En el modo incremental se calcula el gradiente y se actualizan los pesos después de que cada ejemplo pasa por la red. En el modo por lotes, sólo hasta que termina una época, es decir, todos los ejemplos, se actualizan los pesos.

A.11. Pseudocódigo de autocodificador ruidoso apilado

Algorithm 11 Pseudocódigo SDA

procedure SDA ▷ Pseudocódigo para implementar un Autocodificador ruidoso apilado
  $i \leftarrow 0$
  while $i < numeroCapas$ **do**
    if $i = 1$ **then**
      $capa_{entrada} \leftarrow entrada$
    else
      $capa_{entrada} \leftarrow$ salida de la capa sigmoide anterior
    $i \leftarrow i + 1$
    construir capa sigmoide con entrada = capa de entrada
    construir capa autocodificador con entrada = capa de entrada W=W de capa sigmoide y bias = bias de capa sigmoide
    capa sigmoide anterior = capa sigmoide

A.12. Pseudocódigo de entrenamiento para una RBM convolucional

Algorithm 12 Pseudocódigo CRBM

procedure CRBM ▷ Pseudocódigo para entrenar una CRBM
  $ejemplo \leftarrow 0$
  while $ejemplo < TotalEjemplos$ **do**
    $V^{(0)} \leftarrow V$ ▷ Poner el ejemplo actual como un mini batch
    Calcular $Q^{(0)} \leftarrow P(H|V^{(0)})$ ecuación 2.7
    muestrear $H^{(0)}$ de $Q^{(0)}$
    $n \leftarrow 0$
    while $n < N_h$ **do**
      Muestrear V^n de $P(V|H^{(n-1)})$ ecuación 2.8
      Calcular $Q^{(n)} \leftarrow P(H|V^n)$ ecuación 2.7
      Muestrear $H^{(n)}$ de $Q^{(n)}$
    $\Delta W^k = \frac{1}{N_H^2}((Q^{(0),k})^T * V^{(0)} - (Q^{(n),k})^T * V^{(n)})$
    $\Delta b^k = \frac{1}{N_H^2} \sum_{ij} (Q_{ij}^{(0),k} - Q_{ij}^{(n),k}) + \Delta b_k$
    $\Delta c = \frac{1}{N_V^2} \sum_{ij} ((V_{ij}^{(0)}) - (V_{ij}^{(n)}))$
    $ejemplo \leftarrow ejemplo + 1$

Anexo B

Definición de clases de la biblioteca propuesta

B.1. Clases para un auto-codificador

Cuadro B.1: Clase SDA

Atributos	N: Número de muestras de entrenamiento n_in: Número de neuronas de la capa de entrada size_hidden_layers: Número de neuronas en cada capa oculta n_out: Número de neuronas en la capa de salida n_layers: Número de capas ocultas sigmoid_layers: Capas ocultas dA_layers: Capas autocodificador log_layer: Capa para clasificar
Métodos	SDA: Construtor de la clase, se encarga de todas las inicializaciones necesarias pretrain: Método que se encarga de pre entrenar capa por capa los autocodificadores ruidosos dA finetune: Método que se encarga de afinar la red con datos etiquetados predict: Método que se encarga de dar la predicción de un ejemplo dado

B.2. Clases para una red de creencia profunda

La definición de la clase HiddenLayer (ver cuadro B.3) y LogisticRegression (ver cuadro B.4) son las mismas que se mencionaron en la arquitectura de autocodificadores.

Cuadro B.2: Clase dA

Atributos	N: Número de muestras de entrenamiento n_visible: Número de neuronas de la capa de entrada n_hidden_layers: Número de neuronas en cada capa oculta W: Pesos que conectan las neuronas visibles con neuronas ocultas hbias: El bias de las neuronas ocultas vbias: El bias de las neuronas visibles
Métodos	dA: Construtor de la clase, se encarga de todas las inicializaciones necesarias get_corrupted_input: Método que ayuda a generar ruido en la entrada get_hidden_values: Método que calcula la probabilidad de salida de una neurona oculta get_reconstructed_input: Método que calcula la probabilidad de salida de una neurona visible train: Método que se encarga de entrenar el modelo con un ejemplo reconstruct: Método que reconstruye el ejemplo de entrada

Cuadro B.3: Clase HiddenLayer

Atributos	N: Número de muestras de entrenamiento n_in: Número de neuronas de la capa de entrada n_out: Número de neuronas de la capa de salida W: Pesos de la red b: Bias de la red
Métodos	HiddenLayer: Construtor de la clase, se encarga de todas las inicializaciones necesarias output: Método que calcula el valor de un determinado nodo en la capa oculta sample_h_given_v: Método que infiere el estado de una neurona oculta dada un neurona visible

B.3. Clases para una red de creencia profunda convolucionada

Cuadro B.4: Clase LogisticRegression

Atributos	N: Número de muestras n_in: Número de neuronas de entrada n_out: Número de neuronas de salida W: Pesos de la red que conectan neuronas de entrada y neuronas de salida b: Bias de las neuronas de salida
Métodos	LogisticRegression: Construtor de la clase, se encarga de todas las inicializaciones necesarias train: Método que entrena el modelos de regresión logística, actualiza los valores de W y b softmax: Método que calcula softmax para un vector de entrada
	predict: Método que realiza una predicción calculando la probabilidad softmax desde la entrada

Cuadro B.5: Clase RBM

Atributos	N: Número de muestras de entrenamiento n_visible: Número de neuronas de la capa de entrada n_hidden: Número de neuronas de la capa de salida W: Pesos de la red hbias: Bias de la red vbias: Bias de la red
Métodos	RBM: Construtor de la clase, se encarga de todas las inicializaciones necesarias contrastive_divergence: Método que realiza la divergencia contrastiva para entrenar la RBM sample_h_given_v: Método que infiere el estado de una neurona oculta dada un neurona visible sample_v_given_h: Método que infiere el estado de una neurona oculta dada un neurona visible propup: Método que infiere el estado de una neurona oculta dada un neurona visible propdown: Método que infiere el estado de una neurona oculta dada un neurona visible gibbs_hvh: Método que realiza el muestreo de gibbs desde un nodo oculto a un nodo visible, después muestrea desde un nodo visible a un nodo oculto. reconstruct: Método que reconstruye la neurona de entrada por la RBM entrenada

Cuadro B.6: Clase DBN

Atributos	N: Número de muestras de entrenamiento n_in: Número de neuronas de entrada n_out: Número de neuronas de salida size_hidden_layers: Número de neuronas en cada capa oculta n_layers: Número de capas sigmoid_layers: Capas ocultas RBM_layers: Capas RBM log_layer: Capa para clasificar
Métodos	DBN: Construtor de la clase, se encarga de todas las inicializaciones necesarias pretrain: Método que realiza el pre entrenamiento usando RBM finetune: Método que realiza el afinamiento de la DBN usando un MLP con retro propagación predict: Método que realiza una predicción calculando la probabilidad softmax desde la entrada

Cuadro B.7: Clase CDBN

Atributos	n_stack_conv_rbm: Número de RBM convolucionadas apiladas inputDimensions: Número de dimensiones en los datos de entrada k: Número de grupos de neuronas en la capa oculta N_V: Longitud de la dimensión de la capa de entrada N_H: Longitud de la dimensión de la capa de oculta N_W: Tamaño del filtro asociado con cada grupo C: Tamaño de una dimensión de la capa de agrupamiento N_P: Tamaño de la capa de agrupamiento inputLayer: Capa de entrada stackedRBMs: Capas ocultas del la CDBN
Métodos	CDBN: Construtor de la clase, se encarga de todas las inicializaciones necesarias train: Método que entrena la CDBN

Cuadro B.8: Clase MaxPoolingConvRBMInputLayer

Atributos	n_stack_conv_rbm: Número de RBM convolucionadas apiladas input: Datos de entrada sample: Reconstrucción de la entrada pr: Probabilidad de activación de la entrada reconstruida c: Bias
Métodos	MaxPoolingConvRBMInputLayer: Construtor de la clase, se encarga de todas las inicializaciones necesarias calculatePr: Método que calcula la probabilidad

Cuadro B.9: Clase MaxPoolingConvRBM

Atributos	rate: Tasa de aprendizaje H: Capa oculta a entrenar P: Capa de agrupamiento que está sobre la capa oculta pr: Probabilidad de activación de la entrada reconstruida c: Bias
Métodos	MaxPoolingConvRBM: Construtor de la clase, se encarga de todas las inicializaciones necesarias train: Método que entrena la capa oculta

Cuadro B.10: Clase MaxPoolingConvRBMHiddenLayer

Atributos	h: Neuronas pr: Probabilidad de activación de la nuerona oculta W: k-ésima matriz de pesos b: k-ésimo bias
Métodos	MaxPoolingConvRBMHiddenLayer: Construtor de la clase, se encarga de todas las inicializaciones necesarias calculatePr: Método que calcula la probabilidad de activación de la nuerona oculta sample: Método que calcula la activación de la muestra para la (i, j)-ésima neurona del k-ésimo grupo

Cuadro B.11: Clase MaxPoolingConvRBMPoolingLayer

Atributos	pr: Probabilidad de activación de las nuerona de la capa actual
Métodos	MaxPoolingConvRBMPoolingLayer: Construtor de la clase, se encarga de todas las inicializaciones necesarias calculatePr: Método que calcula la máxima probabilidad de activación de las nueronas en una pequeña regioón de la capa oculta sample: Método que calcula la activación de la muestra para la (i, j)-ésima neurona del k-ésimo grupo

Anexo C

Manual de usuario de la biblioteca desarollada

El software desarrollado para este trabajo de tesis se implementó en lenguaje C++, específicamente el compilador g++ versión 4.2 del sistema operativo Ubuntu. Los detalles de instalación y utilización se presentan en las siguientes secciones.

C.1. Proceso de instalación

El software para utilizar las arquitecturas profundas se distribuye como un archivo ejecutable **DeepLearning**. Para la ejecución de este archivo, basta con utilizar la consola de Ubuntu y escribir:

```
./DeepLearning
```

C.2. Integración del software

En la figura C.1 se muestra un ejemplo de la integración de la biblioteca desarrollada. La función principal llamada *main* hace un llamado al método *prueba_dbn* (línea 33), este método se encarga del entrenamiento y clasificación de los datos.

Debe haber ejemplos de entrada para el entrenamiento (línea 10) con sus correspondientes salidas (línea 12). Es necesario también crear la red de creencia profunda, especificando sus parámetros (línea 15)

.

Una vez definidos los datos de entrenamiento y la arquitectura de la dbn, se procede al pre-entrenamiento de los datos (línea 17). Después, es necesario refinar la matriz de pesos,

main.c UNREGISTERED

main.c

```
1  void prueba_dbn()
2  { double tasa_aprendizaje_preentrenamiento = 0.1,  tasa_aprendizaje_refinamiento = 0.1;
3      int epocas_preentrenamiento = 1000;
4        int epocas_refinamiento = 500, epocas_entrenamiento = 1;
5        int k = 1,i,j;
6        int ejemplos_entrenamiento = 6, ejemplos_prueba = 2;
7        int neuronas_entrada = 6, neuronas_salida = 2;
8        int tamano_capas_ocultas[] = {3, 3};
9        int n_capas = sizeof(tamano_capas_ocultas) / sizeof(tamano_capas_ocultas[0]);
10   int datos_entrenamiento_X[6][6] = // datos de entrenamiento
11     { {1,1,1,0,0,0},{1,0,1,0,0,0},{1,1,1,0,0,0},{0,0,1,1,1,0},{0,0,1,0,1,0},{0,0,1,1,1,0}};
12       int datos_entrenamiento_Y[6][2] =
13     {{1,0}, {1,0},{1,0},{0,1},{0,1},{0,1}};
14   // Se construye la red de creencia profunda
15     DBN dbn(ejemplos_entrenamiento, neuronas_entrada, tamano_capas_ocultas, neuronas_salida, n_capas);
16       // preentrenamiento
17       dbn.pretrain(datos_entrenamiento_X, tasa_aprendizaje_preentrenamiento, k, epocas_preentrenamiento);
18       // refinamiento
19       dbn.finetune(datos_entrenamiento_X, datos_entrenamiento_Y,tasa_aprendizaje_refinamiento,epocas_refinamiento);
20       int datos_prueba_X[3][6] =
21     { {1,1,0,0,0,0},{0,0,0,1,1,0},{1,1,1,1,1,0}};
22       double datos_prueba_Y[3][2];
23       for( i=0; i<ejemplos_prueba; i++)
24   {
25             dbn.predict(datos_prueba_X[i], datos_prueba_Y[i]);
26               for(j=0; j<neuronas_salida; j++)
27               cout << datos_prueba_Y[i][j] << " ";
28               cout << endl;
29       }
30   }
31   void main()
32   {
33      prueba_dbn();
34   }
```

Figura C.1: Ejemplo de integración de la biblioteca

lo cual se lleva a cabo en la línea 19.

Cuando los pasos anteriores se han realizado, entonces se puede clasificar los ejemplos de prueba (línea 20), usando la función de predicción *predict* (línea 25). Esta función devuelve los valores que logra predecir la red.

Es importante decir que los datos de salida son valores reales, almacenados en un arreglo con dimensión del tamaño del número de neuronas de salida, por lo que el índice de la coordenada del arreglo correspondiente al valor más grande de la salida será el que indique la clase a la que pertenece ese ejemplo.

C.3. Utilización del software

El programa **DeepLearning** requiere para su utilización un archivo de configuración denominado **confi.txt** que se describe a continuación (ver cuadro C.1):

Cuadro C.1: Archivo de configuración para ejecutar arquitecturas profundas

opción	1 para autocodificador 2 para máquina de Boltzmann restingida 3 para red convoluvional
número de ejemplos	
número de capas ocultas	en el caso de red convolucional es el número de filtros
número de neuronas por cada capa oculta	separadas por comas
tamaño de filtro por cada capa oculta	separadas por comas, sólo para red convolucional, omitir este dato para los auto-codificadores y las MBR.
número de clases	
tasa de aprendizaje (valor entre 0 y 1)	
épocas del preentrenamiento	
tasa de aprendizaje para el refinamiento (valor entre 0 y 1)	
épocas del refinamiento	
nivel de corrupción (valor entre 0 y 1)	para auto-codificador, omitir este dato para las MBR y la red convolucional.
número de reducción	para red convoluvional, omitir este dato para los auto-codificadores y las MBR.

Este archivo de texto se debe colocar en la misma ruta en donde se encuentre el ejecutable **DeepLearning**, y se debe de nombrar todo con minúsculas **confi.txt**. El programa abre por default este archivo.

C.3.1. Ejemplo de uso de arquitecturas profundas para el reconocimiento de dígitos manuscritos

Para los siguientes experimentos se utilizó la partición de los datos del: 80 % para el conjunto de entrenamiento y 20 % para el conjunto de pruebas. Se recomienda esta división de los datos para poder obtener el rendimiento general del clasificador con datos que no se han usado en la fase de entrenamiento, en este caso, los datos de prueba.

auto-codificador

Se define el archivo de configuración, en este caso, para el auto-codificador, se realiza una prueba con 4 capas ocultas y con 500 neuronas ocultas en cada capa (ver figura C.2).

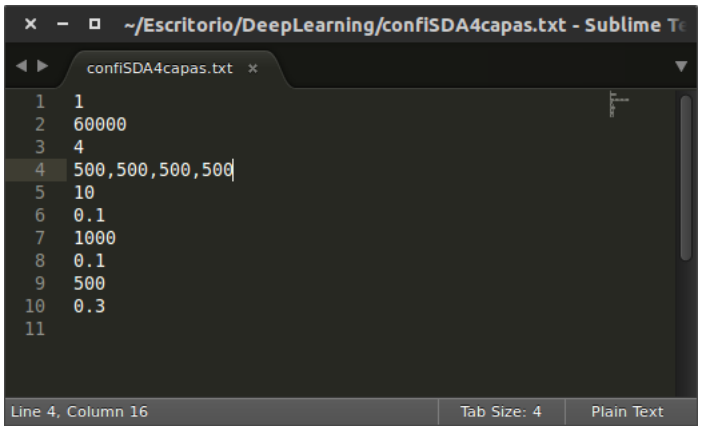

Figura C.2: Archivo de configuración del auto-codificador para reconocer dígitos manuscritos con 4 capas ocultas.

Después de ejecutar ‘‘`DeepLearning`’’ se muestran los resultados de la matriz de confusión, y el porcentaje de exactitud de clasificación del conjunto de test como se muestra en la figura C.3. Cabe mencionar que en el archivo de configuración, uno de los datos de entrada es el número de ejemplos total, por lo cual, es responsabilidad del usuario realizar una partición interna para ejemplos de entrenamiento y de prueba.

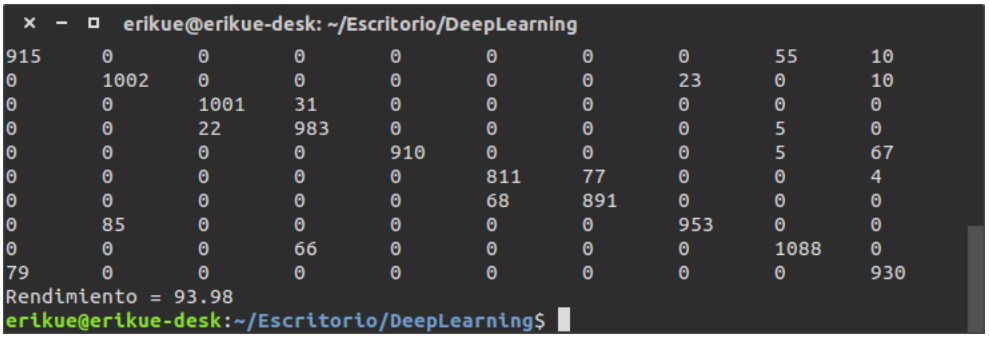

Figura C.3: Resultados de la matriz de confusión y rendimiento de la ejecución del auto-codificador con 4 capas ocultas.

El archivo de configuración para otro ejemplo de un auto-codificador, pero con 2 capas ocultas se muestra en la figura C.4.

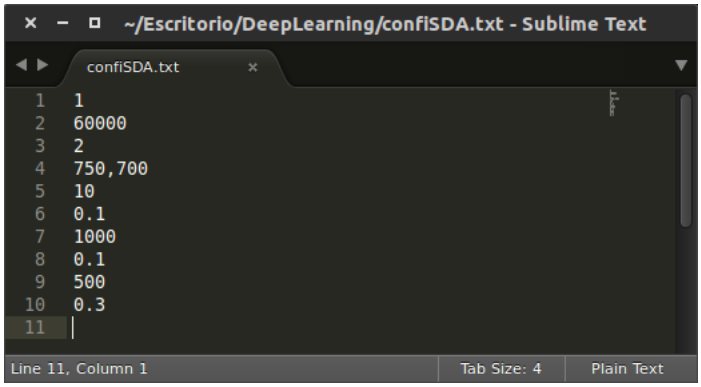

Figura C.4: Archivo de configuración del autocodificador para reconocer dígitos manuscritos con 2 capas ocultas.

Los resultados de la matriz de confusión, y el porcentaje de exactitud de clasificación del conjunto de test, para este ejemplo se muestran en la figura C.5.

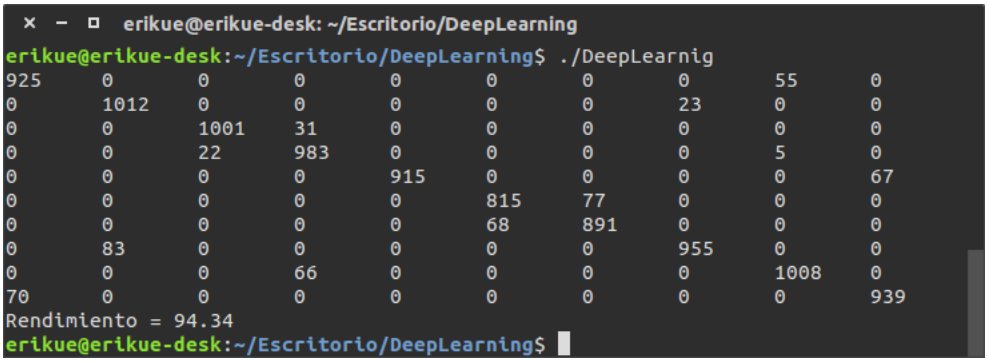

Figura C.5: Resultados de la matriz de confusión y rendimiento de la ejecución del autocodificador con 2 capas ocultas.

máquina de Boltzmann restringida

Ahora, se define el archivo de configuración, en este caso, para una máquina de Boltzmann restringida, se realiza una prueba con 4 capas ocultas y con 500 neuronas ocultas en cada capa (ver figura C.6).

Figura C.6: Archivo de configuración de la máquinas de Boltzmann restringidas para reconocer dígitos manuscritos con 4 capas ocultas.

Los resultados de la matriz de confusión, y el porcentaje de exactitud de clasificación del conjunto de test, para este ejemplo se muestran en la figura C.7.

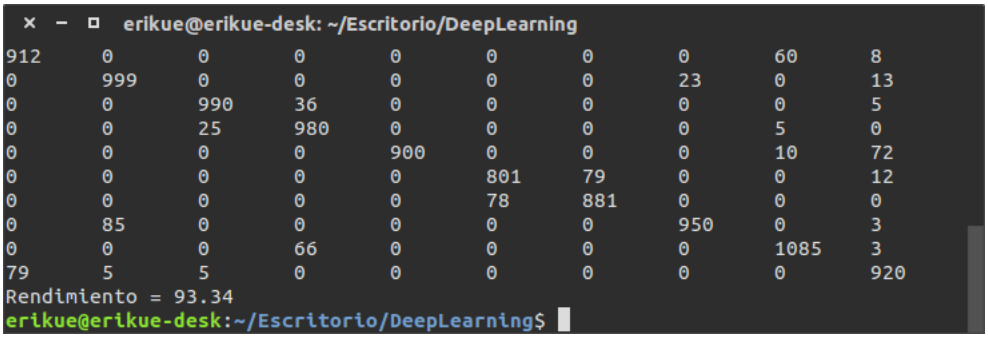

Figura C.7: Resultados de la matriz de confusión y rendimiento de la ejecución de la máquina de Boltzmann restringida con 4 capas ocultas.

Otro ejemplo de configuración de una máquina de Boltzmann restringida, pero con 2 capas ocultas se puede ver en la figura C.8.

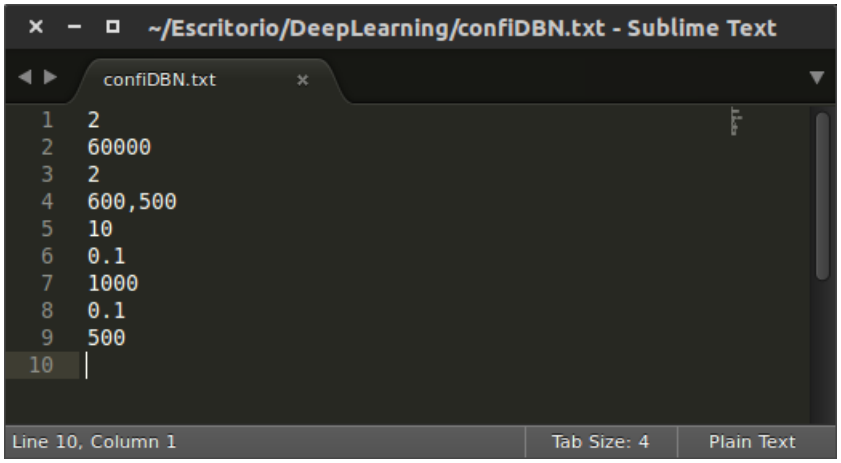

Figura C.8: Archivo de configuración la máquina de Boltzmann restringida para reconocer dígitos manuscritos con 2 capas ocultas.

Los resultados de la matriz de confusión, y el porcentaje de exactitud de clasificación del conjunto de test, para este ejemplo se muestran en la figura C.9.

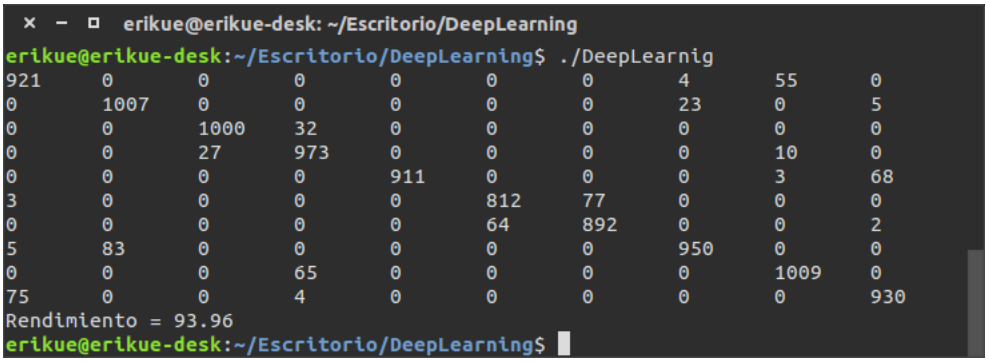

Figura C.9: Resultados de la matriz de confusión y rendimiento de la ejecución de la máquina de Boltzmann restringida con 2 capas ocultas.

red convolucional

Por último, se define el archivo de configuración, en este caso, para una red convolucional, se realiza una prueba con 4 capas ocultas con 500 neuronas ocultas en cada capa y 4 filtros (ver figura C.10).

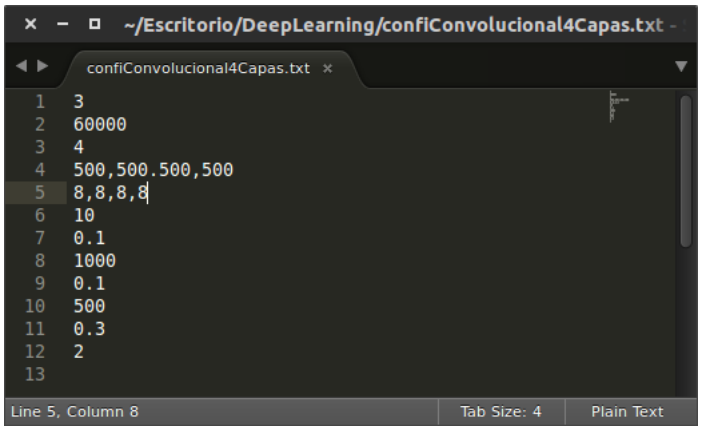

Figura C.10: Archivo de configuración de la red convolucional para reconocer dígitos manuscritos con 4 capas ocultas.

Los resultados de la matriz de confusión, y el porcentaje de exactitud de clasificación del conjunto de test, para este ejemplo se muestran en la figura C.11.

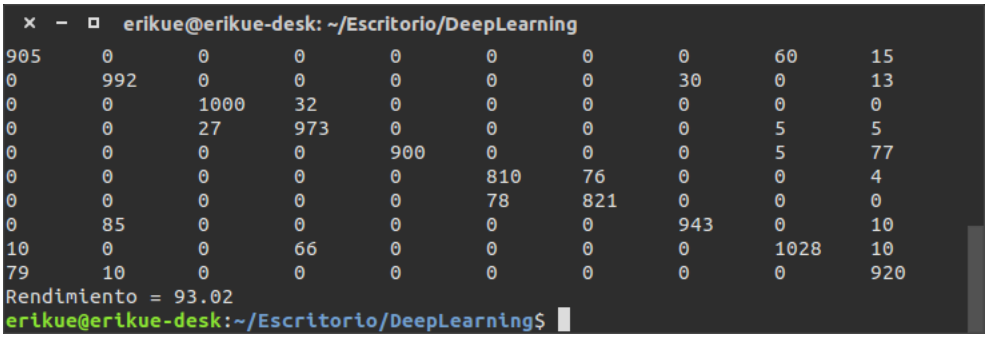

Figura C.11: Resultados de la matriz de confusión y rendimiento de la ejecución de la red convolucional con 4 capas ocultas.

Otro ejemplo de una red convolucional, pero con 2 capas ocultas se puede ver en la figura C.12.

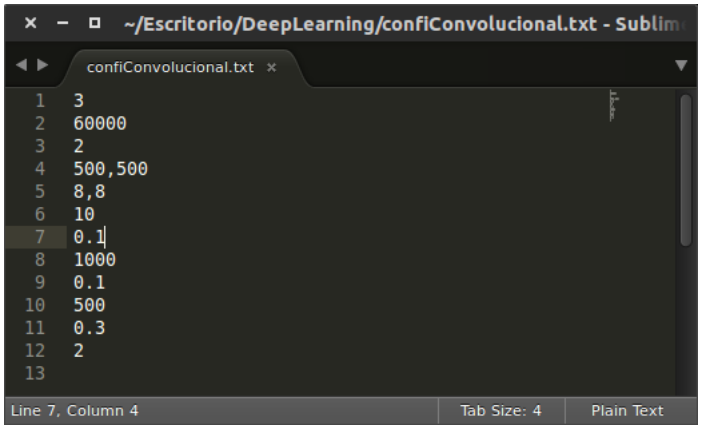

Figura C.12: Archivo de configuración de la red convolucional para reconocer dígitos manuscritos con 2 capas ocultas.

Los resultados de la matriz de confusión, y el porcentaje de exactitud de clasificación del conjunto de test, para este ejemplo se muestran en la figura C.13.

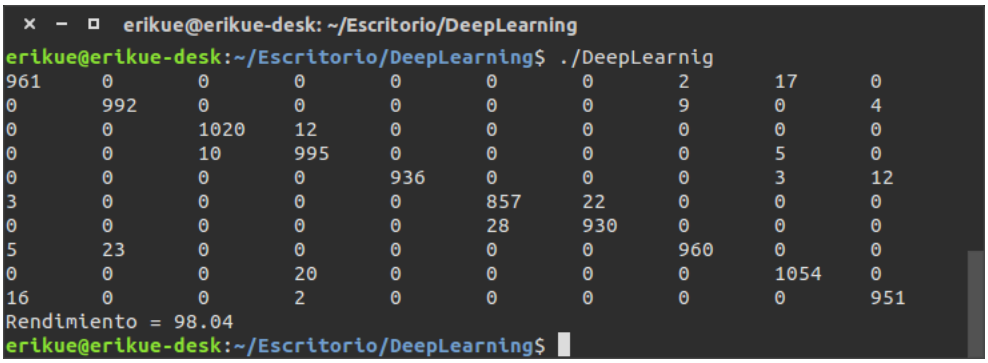

Figura C.13: Resultados de la matriz de confusión y rendimiento de la ejecución de la red convolucional con 2 capas ocultas.