Universidad Autónoma de Yucatán
Facultad de Matemáticas

Clasificación de epítopos usando un método híbrido de aprendizaje automático

T E S I S

presentada por:
Ángel Adrián Bustillos Sosa

en opción al título de:
Ingeniero en Computación

Mérida, Yucatán, México
Mayo 2024

Dedicatoria

Dedico este trabajo principalmente a mi madre y mi padre, a quienes agradezco por siempre procurar por mi bienestar, y por dedicar años de trabajo duro para darme la oportunidad de tener un buen futuro, al que ahora llamo presente.

A mi hermano y mi hermanita, que han estado ahí para darme apoyo en los momentos difíciles, y para disfrutar juntos los momentos de alegría.

A cada uno de los docentes que me ayudaron mostrándome su confianza en cada una de mis etapas de estudiante.

Y a todas aquellas personas que motivaron mi espíritu científico, que fue lo que finalmente me llevó a realizar esta tesis.

Agradecimientos

De manera especial a la Dra. Anabel Martín González por brindarme su confianza, apoyo, y conocimiento, como docente y como mi asesora en la realización de este proyecto.

Al Dr. Carlos Brito Loeza y a la Dra. Gilma Sánchez Burgos por la asistencia durante la elaboración de este proyecto, aportando su valioso conocimiento y experiencia.

Y finalmente, a la Facultad de Matemáticas.

Resumen

Los epítopos son las secciones en las proteínas que activan el sistema inmune del cuerpo. Los más estudiados son los lineales, conformados por un tramo corto de la cadena de aminoácidos que compone a una proteína. La detección de los epítopos de células B es importante para el diseño de vacunas y tratamientos de inmunoterapia más efectivos. Aunque el diseño tradicional de vacunas es un proceso extenso, la identificación de epítopos ofrece un enfoque más robusto y eficiente. Actualmente, la identificación precisa de estos epítopos aún representa un gran reto para los investigadores.

Como producto de esta tesis se desarrollaron clasificadores de cadenas de aminoácidos con el objetivo de identificar epítopos. Los clasificadores están basados en modelos de aprendizaje profundo generados utilizando tres tipos de arquitecturas de redes neuronales: red neuronal completamente conectada (FCN), red neuronal recurrente con celdas LSTM y red neuronal convolucional. Se utilizó un conjunto de datos con más de 200,000 epítopos experimentales que se encuentran en la base de datos IEDB. Se realizó un preprocesamiento diferente a las secuencias de epítopos para cada tipo de arquitectura de red neuronal de los distintos modelos. En el primer modelo se preprocesaron las secuencias de entrada similar al artículo denominado DLBEpitope, para el segundo y el tercer modelo se utilizó la codificación propuesta en otro trabajo denominado ProtVec.

También se propuso un cuarto modelo que combina el uso de las tres arquitecturas (FCN, LSTM y Convolucional) concatenando la salida de la última capa oculta de cada una. Este modelo combinado mostró el mejor rendimiento, seguido por el modelo convolucional.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

Los epítopos, también denominados péptidos antigénicos, son aquellas regiones en las proteínas que generan una activación del sistema inmune (Rodwell et al. 2018). Un péptido es una cadena corta de aminoácidos. Por su estructura, los epítopos se clasifican en dos tipos: lineales y no lineales. Este proyecto se centró en la detección de epítopos lineales, que se conforman por una sección continua de la cadena de aminoácidos que compone a la proteína.

La detección de los epítopos que estimulan la activación de células tipo B tiene su importancia en que pueden ser usados para el diseño de vacunas y tratamientos de inmunoterapia (Dhanda et al. 2016). Debido a que el diseño tradicional de vacunas es un proceso muy extenso, la identificación de epítopos puede ofrecer un enfoque más preciso y eficiente. No obstante, la correcta identificación de estos epítopos sigue siendo un desafío significativo para los investigadores. Es por ello que se han realizado múltiples trabajos de investigación buscando aprovechar el potencial del aprendizaje automático para identificar y clasificar epítopos a partir de secuencias de aminoácidos, con el objetivo de acelerar y reducir la carga de trabajo que conlleva la identificación de epítopos en el laboratorio.

1.1. Objetivos

1.1.1. Objetivo general

El objetivo general de este proyecto es clasificar automáticamente un péptido que promueve la activación de células tipo B como epítopo usando

un método hibrido de aprendizaje automático.

1.1.2. Objetivos específicos

- Obtener una base de datos de péptidos etiquetados como epítopos y no epítopos.
- Encontrar una representación numérica/vectorial robusta para péptidos.
- Implementar modelos de aprendizaje automático para clasificar péptidos como epítopos o no epítopos.
- Evaluar los resultados de concatenar las arquitecturas utilizadas en una única red neuronal artificial.

1.2. Antecedentes

A continuación se presentan trabajos previos realizados en el ámbito de la detección y clasificación de epítopos utilizando aprendizaje automático.

1.2.1. ABCPred

En el estudio realizado por Saha y Raghava (2006), cuyo trabajo nombraron ABCpred, llevaron a cabo pruebas con dos tipos de arquitecturas de redes neuronales diferentes, utilizando un conjunto de datos compuesto por 700 epítopos de células B no redundantes, los cuales fueron obtenidos de la base de datos Bcipep, una base de datos de epítopos de células B. La arquitectura del primer modelo evaluado consistió en una red neuronal completamente conectada (FCN, por sus siglas en inglés), mientras que para el segundo modelo adoptaron una arquitectura de red neuronal recurrente (RNN, por sus siglas en inglés). Los resultados señalaron que la RNN exhibió un rendimiento superior en comparación con la FCN. Esta arquitectura particular de la RNN incluyó una sola capa oculta compuesta por 35 neuronas y una ventana de entrada de 16 aminoácidos de tamaño. La exactitud lograda por este modelo fue de 65.93 %.

1.2.2. LBTope

Singh, Ansari y Raghava (2013) publicaron los resultados de un proyecto al que denominaron LBTope. Para este proyecto se realizaron experimentos con dos técnicas diferentes de aprendizaje automático, la Máquina de Soporte Vectorial (SVM, por sus siglas en inglés) y el algoritmo de K-vecinos más cercanos. Las pruebas también se hicieron utilizando diferentes tipos de caracterización para las cadenas de aminoácidos a clasificar, como su composición dipéptida, su perfil binario, entre otros. En este caso, el conjunto de entrenamiento utilizado sólo contiene péptidos etiquetados como epítopos negativos si han sido validados experimentalmente como tal, pues en los trabajos previos revisados se habían estado usando cadenas de péptidos seleccionadas arbitrariamente sin haber sido validadas realmente como epítopos negativos. Uno de los aportes importantes de este trabajo es la conclusión de que los modelos basados en la composición dipéptida (pares diferentes de aminoácidos dentro del péptido) demostraron un mejor desempeño, por sobre otras representaciones, haciendo mención específicamente a un modelo de máquina de soporte vectorial al cual calificaron de acuerdo con el AUC cuyo valor fue de 0.73.

1.2.3. DLBEpitope

Desarrollado por Liu, Shi y Li (2020), DLBEpitope es un sistema de redes neuronales que consiste en el uso de un ensamble de redes neuronales completamente conectadas como clasificador binario de péptidos (secuencias de proteínas) de longitud fija. Para ello, se realizó un entrenamiento utilizando una base de datos con evidencia experimental de IEDB y modificada, acortando o extendiendo cada una de las secuencias que conforman a los epítopos originales, de tal manera que todas las muestras fuesen de igual longitud. El ensamble de redes se constituyó por un conjunto de 11 redes neuronales FCN, cada una de las cuales recibió un subconjunto de 20,000 muestras negativas junto con alrededor de 20,000 muestras positivas para su entrenamiento. Esta implementación utilizando un ensamble ayuda a solucionar el problema del conjunto de datos desbalanceado, ya que este incluía apenas 20,000 muestras de epítopos positivos comparado a las cerca de 200,000 muestras de epítopos negativos.

Tomando en cuenta que para todas las muestras de la base de datos encontramos solo 20 aminoácidos únicos que las conforman, en DLBEpitope

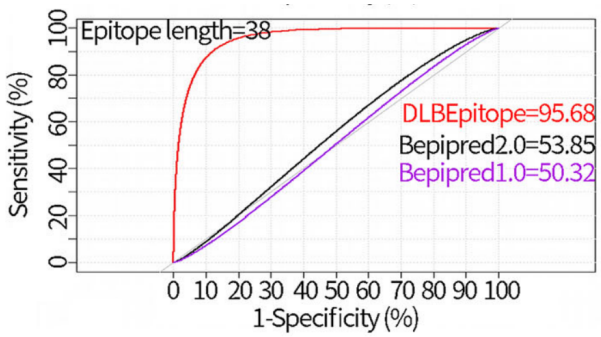


Figura 1.1: Gráfica de la curva ROC y el valor del AUC del mejor modelo de DLBEpitope (Tomado de Liu, Shi y Li 2020)

utilizan como entrada de la red neuronal el histograma de frecuencias de los dipéptidos en cada muestra, esto quiere decir que transforman cada secuencia de aminoácidos en un vector normalizado de 400 elementos, correspondiente a los pares posibles que se pueden formar con los 20 aminoácidos, donde cada elemento está ligado a un único dipéptido y cuyo valor representa el porcentaje de aparición de ese dipéptido en la secuencia.

DLBEpitope logró una precisión del 86.7 % y un área bajo la curva ROC (AUC) de 0.94 en el conjunto de datos de prueba (Figura 1.1), superando a otros métodos existentes en ese momento. Estos valores se obtuvieron de una base datos de péptidos de longitud fija.

1.2.4. Carencias en los antecedentes presentados

Para ABCPred, un punto importante a resaltar es la limitación que impone el uso de FCN, ya que las entradas de modelos con esta arquitectura deben ser de un tamaño fijo. Esta característica generalmente resulta en un problema cuando se trabaja con datos secuenciales, como los epítopos, pues naturalmente estos varían en su tamaño, ya que pueden estar conformados por solo dos aminoácidos o incluso llegar hasta los 50 o más. LBTope, por su parte, es una investigación que utiliza algoritmos de aprendizaje automático sin entrar en el área del aprendizaje profundo.

Por otro lado, el conjunto de datos de entrenamiento utilizado en DLBEpitope tiene algunas inconsistencias que nos pueden llevar a cuestionar la fiabilidad del sistema presentado. Al utilizar la técnica de extender o recortar las secuencias de aminoácidos con el fin de tener muestras del mismo tamaño,

estas ya no coinciden con el registro que se encuentra en IEDB, por lo que no se puede asegurar que las secuencias alteradas sigan teniendo las propiedades suficientes para seguir funcionando como epítopos, en el caso de las muestras positivas.

1.3. Contribuciones

En este trabajo se presenta la implementación de tres tipos diferentes de arquitecturas de redes neuronales: redes neuronales completamente conectadas (FCN), redes neuronales recurrentes (RNN) con celdas LSTM y redes neuronales convolucionales (CNN), todas entrenadas para la clasificación de péptidos como epítopos y no epítopos. Además, se compara el desempeño de cada una de las arquitecturas, ofreciendo un punto de referencia en los beneficios que cada arquitectura presenta en esta tarea particular.

Capítulo 2

Marco teórico

2.1. Epítopos

El sistema inmunitario del cuerpo humano está compuesto por tres elementos principales: los linfocitos B, los linfocitos T, y el sistema inmune innato. Los linfocitos B y T, también llamados células B y células T, responden de manera adaptativa a las sustancias potencialmente dañinas que entran al organismo, desarrollando una respuesta de defensa específica para cada agente invasor (Nelson y Cox 2017). Al agente invasor se le denomina patógeno, y comúnmente corresponde con alguna bacteria o virus infeccioso. Las células B son las encargadas de sintetizar anticuerpos que circularan en el plasma sanguíneo. Esto conforma la línea frontal del sistema inmune del cuerpo.

De acuerdo con Abbott, Damschroder y Lowe (2013), los epítopos son "las estructuras moleculares dentro de cualquier antígeno objetivo determinado que establecen contactos específicos con el anticuerpo". Por su parte, Saha, Bhasin y Raghava (2005) definen los epítopos de células B como "las regiones antigénicas de las proteínas reconocidas por los sitios de unión de las moléculas de inmunoglobulina". En una forma más general, son aquellas secciones de las proteínas infecciosas capaces de generar una respuesta que activa al sistema inmune que, en consecuencia, genera los anticuerpos necesarios para neutralizarlas.

Los epítopos se clasifican en dos tipos (Rodwell et al. 2018):

1. Lineal o continuo: compuesto por un único tramo continuo de la secuencia de aminoácidos.

2. Conformacionales o discontinuos: los residuos se encuentran en secciones separadas de la secuencia de aminoácidos pero están físicamente cercanos a causa del plegamiento de la proteína.

Las proteínas son moléculas grandes y complejas conformadas por una o más cadenas largas y plegadas de aminoácidos (cada una llamada polipéptido) que siguen una secuencia y orden determinados. Dependiendo de la forma en que se haya plegado, la proteína desempeña una función u otra. Las proteínas son principalmente los componentes estructurales de las células (Zumdahl y DeCoste 2012).

Nombre	Abreviación	Símbolo
Alanina	Ala	A
Arginina	Arg	R
Asparagina	Asn	N
Ácido aspártico	Asp	D
Cisteína	Cys	C
Fenilalanina	Phe	F
Glicina	Gly	G
Ácido glutámico	Glu	E
Glutamina	Gin	Q
Histidina	His	H
Isoleucina	Ile	I
Leucina	Leu	L
Lisina	Lys	K
Metionina	Met	M
Prolina	Pro	P
Serina	Ser	S
Tirosina	Tyr	Y
Treonina	Thr	T
Triptófano	Trp	W
Valina	Val	V

Cuadro 2.1: Lista de los 20 aminoácidos que conforman las proteínas, con abreviación y símbolo.

Los aminoácidos son la unidad base que actúa como estructura fundamental de las proteínas. Aunque en la naturaleza se encuentran más de 300

aminoácidos, las proteínas se sintetizan casi exclusivamente a partir de un conjunto de 20 aminoácidos distintos (Rodwell et al. 2018). Los científicos tienen establecidas abreviaturas de una y tres letras para identificar cada aminoácido (Cuadro 2.1).

Por último, el concepto péptido hace referencia a cadenas cortas de aminoácidos (desde dos hasta 50) vinculados por uniones químicas.

2.1.1. Importancia de la identificación de epítopos

El proceso para diseñar vacunas ha ido evolucionando a lo largo del tiempo. En principio se inyectaba los patógenos enteros vivos en el paciente. Luego se comenzaron a utilizar diferentes técnicas de purificación del antígeno buscando hacer las vacunas más seguras, pero a su vez perdiendo cierto grado de eficacia (Palatnik-de-Sousa, Silva Soares y Rosa 2018). El siguiente paso en la creación de vacunas más seguras y eficaces consiste en el desarrollo de vacunas basadas en epítopos, ya que está demostrado que usando ciertas regiones antigénicas es posible activar el sistema inmunológico. Sin embargo, crear vacunas utilizando esta técnica es un proceso largo y tedioso, ya que la detección de epítopos en laboratorio aún representa un desafío significativo. Es por esto que la identificación de epítopos mediante el uso de aprendizaje automático puede servir para acelerar estos procesos, así como para diseñar tratamientos de inmunoterapia (Dhanda et al. 2016).

2.2. Redes Neuronales Artificiales

2.2.1. Definición y arquitectura

Las redes neuronales artificiales completamente conectadas, o perceptrón multicapa, son estructuras computacionales inspiradas en el funcionamiento de las neuronas biológicas en el cerebro humano. Consisten en capas de neuronas artificiales (o nodos) interconectadas que procesan información y a partir de ella aprenden a hacer predicciones de resultados o clasificar conjuntos de datos. Cada neurona en una capa realiza el cálculo de una suma ponderada de los valores que recibe como entrada para generar un valor de salida que será evaluado en una función no lineal a la que llamamos función de activación. Los valores de salida de las neuronas en una capa serán los valores que recibirán de entrada las neuronas en la siguiente capa.

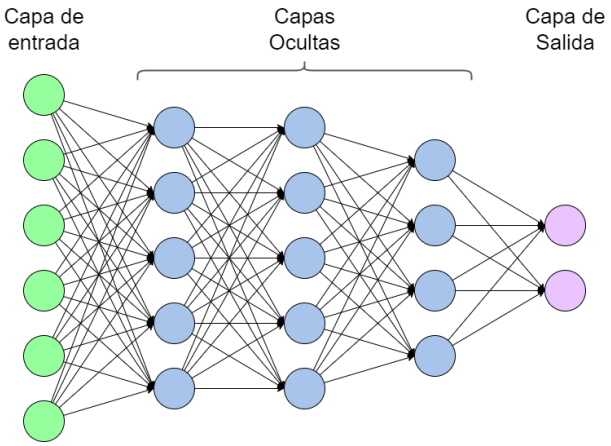


Figura 2.1: Ejemplo de red neuronal profunda prealimentada.

Una red neuronal completamente conectada, que a partir de aquí llamaremos simplemente FCN (por sus siglas en inglés), recibe un conjunto de N datos de entrenamiento, conformado por una gran cantidad de parejas de valor-etiqueta (x, y), pertenecientes a una función desconocida $y = f(x)$ (Goodfellow, Bengio y Courville 2016). El proceso de aprendizaje consiste en encontrar los valores de los parámetros θ (conocidos como pesos) para formar una función $y_o = \hat{f}(x, \theta)$ que haga la mejor aproximación a la función $y = f(x)$. Este proceso de aprendizaje en el cual se proporciona un conjunto de datos ya etiquetados se conoce como aprendizaje supervisado.

De acuerdo con la ruta que sigan los datos que entran y salen en las neuronas en cada capa, podemos clasificar las capas en tres tipos (Figura 2.1):

1. Capa de entrada: Recibe los valores x del conjunto de datos de entrenamiento. Cada neurona de esta capa representa una característica del conjunto de datos.

2. Capas ocultas: Recibe valores de neuronas de la capa de entrada o de otras capas ocultas. En estas se realizan los cálculos para extraer patrones complejos del conjunto de datos de entrada.

3. Capa de salida: Proporciona la predicción de la red neuronal para aproximar el valor y, correspondiente con la respuesta o predicción de la red neuronal. La naturaleza del problema determina el número de neuronas

de salida. Por ejemplo, si nuestro objetivo es clasificar un conjunto de datos en K clases, la red neuronal debe tener K neuronas de salida, cada una representando una clase.

La arquitectura de una red neuronal artificial está definida por su estructura, esto es, la cantidad de capas y neuronas que la conforman. Por su parte, llamamos modelo, a una red neuronal artificial inicializada en un sistema computacional, con los valores de sus pesos definidos.

Las funciones de activación son funciones no lineales que determinan, como su nombre lo indica, el nivel de activación de cada neurona con respecto a los datos de entrada. El uso de estas funciones a la salida de cada capa de la red es lo que le permite aprender patrones complejos de los datos. Ejemplos de funciones de activación más usadas son las siguientes:

1. Función sigmoidal:
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

2. Función ReLU (Nair e Hinton 2010):
$$a(x) = max(0, x) \tag{2.2}$$

3. Función Softmax (Bridle 1989): Es más bien una función de regresión, utilizada en la capa de salida para escenarios de clasificación multiclase:
$$s(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.3}$$

   Donde z es el valor de una neurona de salida y K es el número total de neuronas de salida.

2.2.2. Entrenamiento y Propagación hacia atrás

Durante el entrenamiento de una FCN se pretende aproximar, de manera automática, una función $\hat{f}(x)$ representada por la red a la función $f(x)$ que se infiere de los datos de entrenamiento. Cada muestra x en el conjunto de datos de entrenamiento está acompañada de un valor y denominado etiqueta u objetivo. Los valores x ingresan por la capa de entrada, y la FCN realiza los cálculos, pasando por las capas ocultas, hasta obtener los valores y_o de la

capa de salida, que son los que la red buscará aproximar a los valores y del conjunto de datos.

El error E se usa para evaluar que tan buena es la aproximación entre la función $\hat{f}(x)$ de la red y la función objetivo $f(x)$. Utilizando una función de costo o de pérdida (del inglés *loss function*) es como se obtiene un valor de error. La elección de la función depende del tipo de tarea para la que se entrena la FCN. Las más comunes son:

1. Error cuadrático medio. Se emplea mayormente en tareas de regresión, cuando se busca predecir un valor numérico:

$$E = \frac{1}{n}\sum_{i}^{n}(y^{(i)} - y_o^{(i)})^2 \tag{2.4}$$

2. Entropía cruzada categórica (del inglés *binary cross-entropy*). Se utiliza en redes neuronales cuya tarea se centra en la clasificación/etiquetación de datos:

$$E = -\frac{1}{n}\sum_{i}^{n} y_i \cdot \log y_{o_i} \tag{2.5}$$

   En ambas funciones, n es el número de muestras del conjunto de datos que se están evaluando.

Para un conjunto de entrenamiento con N muestras, el algoritmo de entrenamiento realiza primero un procesamiento de los datos desde la capa de entrada hasta la capa de salida, para luego, calcular el valor de la función de costo E. La segunda etapa consiste en actualizar los pesos θ de cada capa de la red utilizando el algoritmo de retropropagación (del inglés *backward propagation* o más comúnmente llamado *backpropagation*).

El algoritmo de retropropagación utiliza la regla de la cadena del cálculo de derivadas para propagar el error desde la capa final hasta la capa inicial y actualizar cada uno de los pesos con respecto a la muestra actual del conjunto de entrenamiento que fue procesada utilizando lo que se conoce como el algoritmo de descenso de gradiente, que nos permite obtener la razón de cambio (o gradiente) del error con respecto a cada uno de los pesos en la red neuronal.

La magnitud de la actualización de los pesos correspondientes a una neurona v en una capa u de la FCN se calcula con la siguiente fórmula:

$$\Delta W_{[u,v]} = -\eta \frac{\partial L}{\partial W_{[u,v]}} \tag{2.6}$$

donde η es un valor arbitrario que se conoce como tasa de aprendizaje (*learning rate* en inglés). El valor $\Delta W_{[u,v]}$ se suma al valor del peso correspondiente $W_{[u,v]}$ como sigue:

$$\Delta W_{[u,v]}^{(t)} = W_{[u,v]}^{(t-1)} - \Delta W_{[u,v]}^{(t-1)} \tag{2.7}$$

Para acelerar el proceso de entrenamiento de cualquier red neuronal, se divide el conjunto de entrenamiento en subconjuntos o lotes pequeños (*minibatch*), para así, realizar el procesamiento de los datos, el cálculo del error y el algoritmo de retropropagación para todas las muestras de un lote en conjunto, en vez de hacerlo muestra por muestra. Cuando se ha realizado la secuencia completa de pasos de entrenamiento para todas las muestras del conjunto de entrenamiento, se dice que se ha completado una época. Se deben completar tantas épocas como sean necesarias para obtener una mejor aproximación del modelo al comportamiento de los datos.

Al momento de generar un modelo de red neuronal, se busca que sea capaz de generalizar sus predicciones, es decir, que el desempeño que demuestre para nuevos datos sea muy similar al que se obtuvo con los datos de entrenamiento. Cuando el desempeño del modelo al etiquetar las muestras de entrenamiento supera por mucho el desempeño al etiquetar el conjunto de prueba se dice que hubo un "sobreentrenamiento" o "sobreajuste" (se explica con más detalle en la sección 2.2.3). Una técnica ampliamente utilizada para prevenir el sobreajuste es la de *Dropout* (Srivastava et al. 2014), que consiste en omitir o desconectar neuronas de una capa de manera aleatoria durante el entrenamiento. En cada paso del entrenamiento se omitirá el uso de diferentes neuronas, lo que implica también que para ese paso no influirán en la actualización de pesos durante la propagación hacia atrás.

2.2.3. Evaluación del modelo

La evaluación de un modelo de red neuronal se da en dos momentos clave, durante el entrenamiento, y al finalizarlo. Durante el entrenamiento, servirá para saber aproximadamente cuál es el número ideal de épocas necesarias para obtener buenos resultados sin sobreajustar los pesos. Hay que recordar que, matemáticamente, el objetivo de las redes neuronales artificiales es aproximar una función que se infiere de los datos de entrenamiento, sin embargo, ya que estos datos vienen del mundo real es seguro que contendrán cierto nivel de ruido, y el sobreajuste se da cuando la red neuronal ajusta demasiado sus pesos al punto de aprender también el comportamiento del ruido de los

datos. Por lo tanto, se puede decir que un modelo entrenado correctamente debe tener poca sensibilidad a las pequeñas variaciones (o ruido) que existan en los datos de entrada. Una vez que se está satisfecho con el modelo entrenado, viene el segundo momento de evaluación, que se realiza con un nuevo conjunto de datos al que llamamos conjunto de prueba, con características similares al conjunto de entrenamiento, pero que debe cumplir que ninguna de las muestras que incluye haya sido usada durante el entrenamiento, esto significa que las muestras del conjunto de entrenamiento y el de prueba deben ser completamente diferentes.

Si bien, podemos utilizar el valor de la función de error como referencia para medir la evolución del modelo y ver como va mejorando tras al término de cada época, resulta insuficiente como referencia para medir el comportamiento del modelo al etiquetar muestras nuevas. Como se verá en la Sección 2.7, existen varios tipos de funciones de evaluación que podemos utilizar, cuyos valores de salida se encuentran dentro de un rango limitado y están relacionados con la capacidad del modelo para etiquetar los datos de manera correcta, lo que permite hacer comparaciones de diferentes arquitecturas de redes neuronales que pretenden realizar la misma tarea.

2.3. Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales o CNN (abreviación del inglés *Convolutional Neural Networks*), presentadas por Lecun (1995), son arquitecturas de redes neuronales diseñadas para procesar la estructura espacial de los datos. Este tipo de redes son las preferidas en el campo de la visión por computadora, ya que fueron diseñadas inicialmente con el propósito de analizar y extraer patrones en imágenes. Las CNN suelen ser computacionalmente eficientes, porque generalmente requieren menos parámetros que una arquitectura de red completamente conectada, y porque la ejecución de la operación de convolución se puede paralelizar fácilmente en sistemas de cómputo.

En matemáticas, una convolución entre dos funciones, $f, g : R^d \to R$, se define de la siguiente manera:

$$(f * g)(t) = \int f(\tau)g(t - \tau)d\tau \tag{2.8}$$

En cierto modo, se transforman dos funciones f y g en una tercera función, la cual representa el resultado obtenido de la superposición de la función f y

la función g después de haber sido esta última invertida y trasladada. Para el caso discreto, la integral se convierte en una suma (ecuación 2.9).

$$(f * g)(i) = \sum_{a} f(a)g(i-a) \tag{2.9}$$

Aplicándolo al caso particular de matrices o tensores de dos dimensiones, tenemos que (a, b) representa los subíndices de cada elemento en la matriz f, y $(i-a, j-b)$ los subíndices en la matriz g, respectivamente:

$$(f * g)(i, j) = \sum_{a} \sum_{b} f(a, b)g(i-a, j-b) \tag{2.10}$$

En este sentido, f corresponde con la imagen de entrada representada como una matriz de tamaño fijo donde cada uno de sus elementos equivale a un píxel en la imagen. Por su parte, g es una matriz cuadrada conocida como *kernel*, filtro o ventana de convolución, de menor tamaño (usualmente 3×3, 5×5 o algún valor menor que 11), y cuyos valores de sus elementos son los que se obtendrán durante el entrenamiento. El filtro recorre la imagen con el objetivo de extraer las características y patrones más relevantes. El resultado es una matriz, comúnmente de tamaño menor, que corresponde con la salida de la que llamaremos capa de convolución. Cada capa de convolución puede hacer uso de múltiples filtros a la vez para extraer características de diferentes tipos.

A continuación, se presenta un ejemplo de cómo funciona la convolución aplicada a imágenes. En la Figura 2.2 tenemos una imagen de entrada de 3×3 pixeles, y un filtro de tamaño 2×2. De manera visual, la convolución comienza situando el filtro sobre la imagen, con el elemento de la esquina superior izquierda de ambas matrices coincidiendo uno encima del otro. En esta posición, el primer paso es multiplicar cada elemento del filtro con el elemento de la imagen sobre el que esté situado, el segundo paso es realizar la suma de esos valores, y el resultado se colocará en la matriz resultante en su elemento de arriba a la izquierda.

Así, el filtro recorre la imagen, de izquierda a derecha, de arriba a abajo, hasta haber pasado por todos sus píxeles, generando la matriz resultante como la podemos observar en la Figura 2.2. Las imágenes a color suelen estar compuestas de tres matrices sobrepuestas del mismo tamaño, una por cada color, para el rojo, el verde, y el azul. Cada una de estas matrices se conoce como canal, y otro tipo de imágenes pueden contar con un número

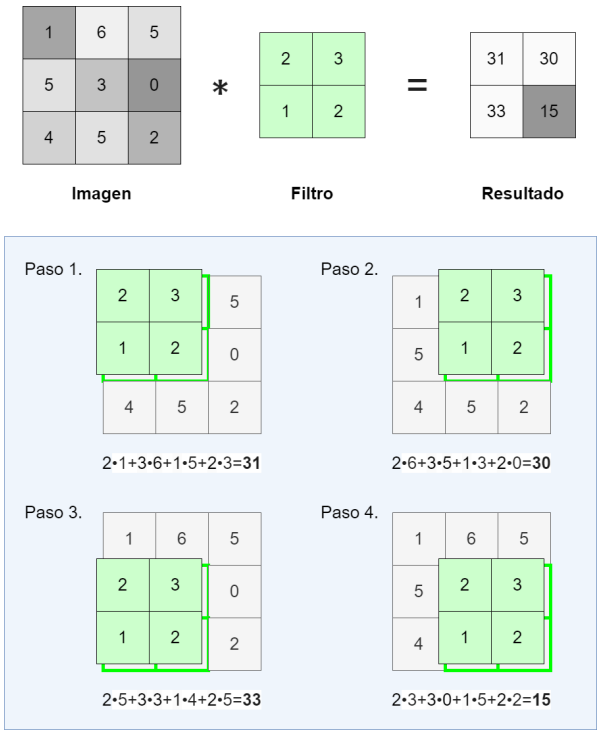


Figura 2.2: Ejemplo de la ejecución y cálculo de una convolución sobre una imagen.

diferente de canales. Cuando se aplica la convolución a una imagen con más de un canal, el filtro debe tener el mismo número de canales que la imagen, de este modo, cada canal del filtro recorrerá el canal respectivo en la imagen.

En el caso de las redes neuronales convolucionales, cada capa de la red consiste en múltiples filtros que se aplican a la imagen para extraer características específicas. Cada filtro corresponde con diferentes aspectos visuales, permitiendo a la red aprender desde detalles simples como bordes, texturas y colores, hasta patrones más complejos. El número de filtros que se use en una capa determina el número de canales que tendrá su salida.

Además del tamaño y número de filtros, hay otras variables que pueden modificarse en una capa de convolución. Una es el tamaño del paso que da el filtro al recorrer la imagen, que en inglés se conoce como *stride*. Por otro lado, al recorrer la imagen con el filtro, como se ha explicado hasta ahora, la matriz resultante es de menor tamaño que la original. Si se quisiera conservar las mismas dimensiones de la imagen original se puede usar un relleno (en inglés conocido como *padding*), que se coloca en los bordes de la matriz.

Por último, hay un tipo de operaciones que funcionan de manera algo similar a la convolución, y que en inglés se llaman *pooling operators* (que puede traducirse como *operadores de agrupación*). Esta operación utiliza una ventana de un tamaño determinado que va recorriendo la imagen, como lo haría un filtro de convolución, y realiza una operación definida sobre los píxeles de la imagen que contiene la ventana en cada paso, que generalmente consiste en calcular el valor máximo o el valor promedio de esos píxeles (Figura 2.3).

2.3.1. Aplicación en secuencias unidimensionales

Las CNN también han sido aplicadas en tareas relacionadas al procesamiento de la estructura de secuencias unidimensionales, como audio, texto, y el análisis de series de datos en el tiempo, tareas donde suelen usarse más las denominadas redes neuronales recurrentes. Esto se logra al aplicar filtros de una dimensión, lo que implica deslizar el filtro a lo largo de la secuencia (sobre una sola dimensión) para extraer características y patrones relevantes.

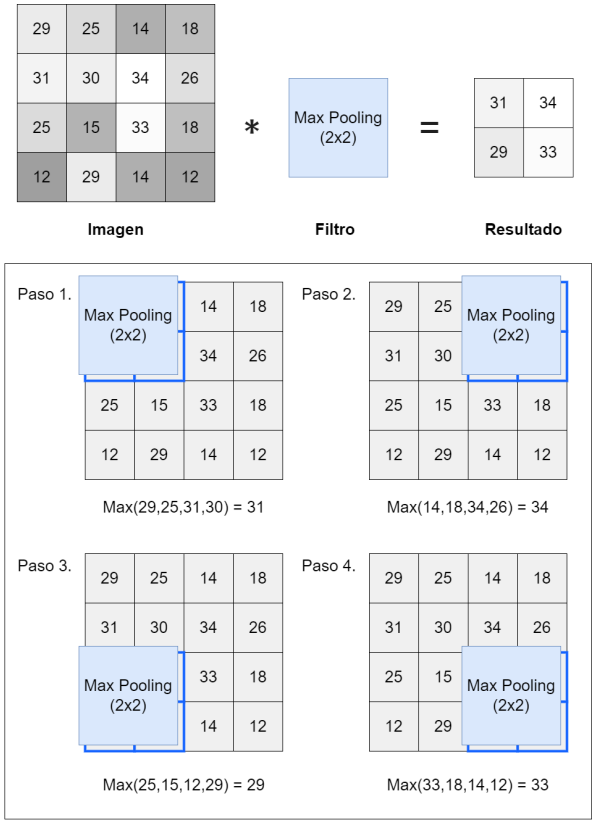


Figura 2.3: Ejemplo de aplicación de la operación de *pooling* de valor máximo sobre una imagen, con un *stride* igual a 2.

2.4. Redes Neuronales Recurrentes (RNN)

Las Redes Neuronales Recurrentes (RNN, por sus siglas en inglés) son sistemas dinámicos diseñados para trabajar con datos secuenciales (Staudemeyer y Morris 2019), donde la relación entre los elementos y su posición en el tiempo es crucial, ya que permite procesar los valores de una secuencia manteniendo un estado interno para cada elemento de la secuencia, el cual tiene influencia en el procesamiento del elemento siguiente. A diferencia de las Redes Neuronales Convolucionales (CNN) que se destacan por su desempeño al analizar la estructura espacial de los datos, las RNN son adecuadas para datos que tienen una dependencia temporal, como lo es el procesamiento de lenguaje natural.

La característica clave de las RNN es su capacidad para mantener una "memoria"de información de estados anteriores para utilizarla en el estado actual. Esto se logra mediante conexiones recurrentes (Figura 2.4). Para una secuencia de longitud T, la RNN analiza cada elemento en orden, y las salidas anteriores de las neuronas en el estado $t-1$ se añaden como entradas en las neuronas para la iteración actual t.

Las RNN necesitan ser entrenadas de manera diferente a las redes feedforward. Para este tipo de redes necesitamos propagar información a través de conexiones recurrentes entre neuronas. Los algoritmos de aprendizaje más comunes para RNN, son retropropagación a través del tiempo (BPTT, *back-propagation through time* en inglés) y aprendizaje recurrente en tiempo real (RTRL, *real-time recurrent learning* en inglés) de Staudemeyer y Morris (2019).

2.4.1. Retropropagación a través del tiempo (BPTT)

Partimos del hecho de que, para un periodo de tiempo finito, una RNN puede ser desplegada en el tiempo, lo que permite visualizarla como una red neuronal feedforward con un comportamiento idéntico (Figura 2.4), pero con la particularidad de que los mismos parámetros se repiten en cada paso temporal. De esta manera, una RNN desplegada puede ser entrenada utilizando el algoritmo de retropropagación que se aplica a redes feedforward. En este caso, los gradientes con respecto a cada parámetro deben sumarse en todos los lugares donde el parámetro aparece en la red desplegada.

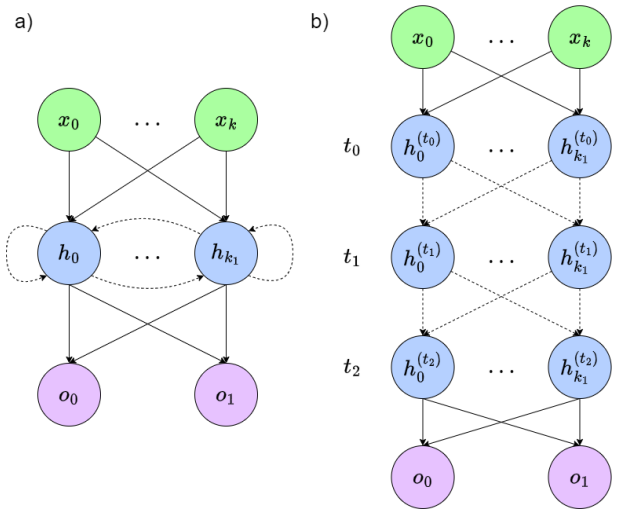


Figura 2.4: a) Arquitectura de una red neuronal recurrente. b) Ejemplo de la red neuronal recurrente desplegada en el tiempo.

2.4.2. Aprendizaje recurrente en tiempo real (RTRL)

El algoritmo de RTRL se describe como un método de aprendizaje *en línea*, ya que toda la información necesaria para calcular el gradiente es recolectada durante el procesamiento de la información de entrada a lo largo de la red (Staudemeyer y Morris 2019). Es por eso que no requiere retropropagación del error, de modo que no requiere un intervalo dedicado únicamente a la actualización de los pesos durante el entrenamiento. No obstante, el algoritmo tiene un costo computacional y de almacenamiento más altos por cada procesamiento, aunque también se debe tener en cuenta que la memoria requerida depende sólo del tamaño de la red y no del tamaño de los datos de entrada.

2.4.3. Desventaja de las RNN

Durante el entrenamiento, las RNN pueden enfrentar el problema del desvanecimiento del gradiente, que ocurre cuando los gradientes se vuelven demasiado pequeños a medida que se propagan hacia atrás en el tiempo, impidiendo el aprendizaje a largo plazo. También es posible que ocurra una situación opuesta, que se conoce como el problema de la explosión del gradiente, cuando los gradientes se vuelven demasiado grandes, lo que puede

llevar a inestabilidad y un error durante el entrenamiento.

2.5. Redes Neuronales LSTM (Long Short-Term Memory)

Las redes neuronales LSTM (Hochreiter y Schmidhuber 1997) son una variante de las RNN diseñadas para superar el problema del desvanecimiento del gradiente. Introduce una estructura de celda más compleja que permite la retención y el olvido selectivo de información (Staudemeyer y Morris 2019).

La celda LSTM es la unidad básica dentro de las redes neuronales del mismo nombre, es equivalente a una capa de una red recurrente, que recibe una entrada x y un estado oculto h, pero que además introducen los conceptos de estado interno de la celda c junto con tres tipos compuertas: de olvido, de actualización, y de salida (Figura 2.5). Su diseño se enfoca en la gestión de memoria a largo plazo, así como la mitigación del problema de desvanecimiento y explosión del gradiente.

A continuación, se describen los conceptos principales de las redes LSTM:

- Compuerta de olvido: utilizando la entrada actual x y el estado oculto h del tiempo anterior, determina si la información del estado de la celda se debe conservar u olvidar.
- Estado de la celda: funciona como una banda que transporta información relevante a través del tiempo, lo que permite manejar memoria a largo plazo.
- Compuerta de actualización: determina qué nueva información debe ser almacenada e integrada en el estado de la celda c para actualizarlo, con respecto a la entrada x y el estado oculto h.
- Compuerta de salida: determina la salida de la celda, con respecto al nuevo estado de la celda c, el estado oculto h, y la entrada x.

2.6. ProtVec

Las muestras de epítopos incluidas en la base de datos están representadas como secuencias de letras, donde cada letra corresponde a un aminoácido específico. Sin embargo, las redes neuronales funcionan realizando operaciones

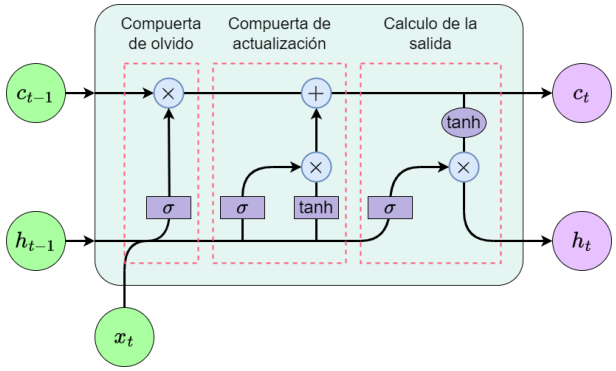


Figura 2.5: Arquitectura de una celda LSTM.

matemáticas, por lo que es necesario encontrar una representación numérica para cada cadena de letras.

Asgari y Mofrad (2015) proveen de un método de representación vectorial para secuencias de proteínas (o cualquier secuencia compuesta por aminoácidos) al que denominaron ProtVec. Han obtenido un espacio vectorial de 100 dimensiones donde mapearon secuencias de tres aminoácidos. Esto significa que para cada tercia ordenada de aminoácidos ProtVec tiene un vector de 100 elementos único que la caracteriza.

El término "ProtVec" se proviene del inglés "protein-vectors" y representa una nueva metodología de representación y extracción de características para secuencias biológicas, específicamente para proteínas. La introducción de ProtVec es parte de una investigación más amplia llamado "biovectors"(BioVec), que abarca representaciones similares para secuencias biológicas en general, como "GeneVec"para secuencias de genes.

Su utilidad radica en su capacidad para representar una secuencia de proteínas mediante un único vector denso de n dimensiones. ProtVec fue utilizado para predecir proteínas desordenadas a partir de proteínas estructuradas. Los resultados muestran una alta precisión, con un 99.8 % de exactitud.

2.7. Métricas de desempeño

Cuando se entrena una red neuronal como clasificador, medir su desempeño va más allá de sólo conocer el porcentaje de aciertos que se obtuvieron. Las métricas de evaluación utilizadas para comparar el desempeño de los modelos de clasificación son: el valor de exactitud, la precisión, la exhaustividad,

el f1-score y el valor del área bajo la curva ROC (AUC).

2.7.1. Precisión (*Precision*)

La precisión mide la proporción de muestras positivas clasificadas correctamente sobre el total de muestras clasificadas como positivas. Se expresa mediante la siguiente fórmula:

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}}$$

2.7.2. Exhaustividad o Sensibilidad (*Recall*)

La exhaustividad evalúa la proporción de muestras positivas que han sido correctamente identificadas respecto al total de muestras positivas en el conjunto de datos. Su fórmula es:

$$\text{Exhaustividad} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}}$$

2.7.3. Exactitud (*Accuracy*)

La exactitud mide la proporción de muestras clasificadas correctamente, ya sean positivas o negativas, sobre el total de muestras. Se calcula mediante la fórmula:

$$\text{Exactitud} = \frac{\text{Verdaderos Positivos} + \text{Verdaderos Negativos}}{\text{Total de muestras}}$$

2.7.4. F1-score

El F1-score es una métrica especialmente útil cuando hay clases desequilibradas en los datos. Combina dos medidas de evaluación de clasificadores: la precisión y la exhaustividad. La fórmula del F1-score se define como:

$$\text{F1-score} = \frac{2 \times \text{Precisión} \times \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

2.7.5. Área bajo la curva ROC (AUC)

La curva ROC (del inglés *Receiver Operating Characteristic*) es una representación gráfica del desempeño de un modelo de clasificación binaria utilizando diferentes valores para el umbral de decisión. Idealmente, un modelo perfecto tendría una curva ROC que se eleva rápidamente hacia el extremo superior izquierdo del gráfico, indicando altas tasas de verdaderos positivos y bajas tasas de falsos positivos. Por lo tanto, un valor de AUC cercano a 1.0 indica un buen rendimiento, mientras que un valor cercano a 0.5 sugiere un rendimiento aleatorio.

Capítulo 3

Metodología

3.1. Base de datos

La base de datos de IEDB (siglas en inglés para *Immune Epitope Database*) es un recurso de acceso gratuito financiado por el Instituto Nacional de Alergias y Enfermedades Infecciosas (NIAID), que recopila datos experimentales sobre epítopos de anticuerpos estudiados en humanos y otras especies animales. En esta se encuentran catalogados datos experimentales sobre epítopos de células B y células T estudiados en humanos, primates no humanos y otras especies animales en el contexto de enfermedades infecciosas, alergias, autoinmunidad y trasplantes (IEDB, iedb.org).

El conjunto de datos de entrenamiento y prueba utilizados en los modelos que se presentan a continuación contienen las mismas secuencias de IEDB que el utilizado en el trabajo de DLBEpitope, con más de 200,000 muestras modificadas. No obstante, para este trabajo se utilizan las secuencias con el tamaño original, tal cual se encuentran registradas en la base de datos IEDB.

Este conjunto de datos generado está dividido en tres subconjuntos: de entrenamiento, validación y prueba. El subconjunto de entrenamiento está compuesto por 18,000 muestras de péptidos clasificados como epítopos positivos (epítopos confirmados) y 197,551 péptidos clasificados como epítopos negativos (no son epítopos). El subconjunto de validación contiene 2,000 muestras de la clase positiva y 2,000 de la clase negativa. Y el de prueba está conformado por 2,012 muestras de la clase positiva y 2,012 muestras de la clase negativa.

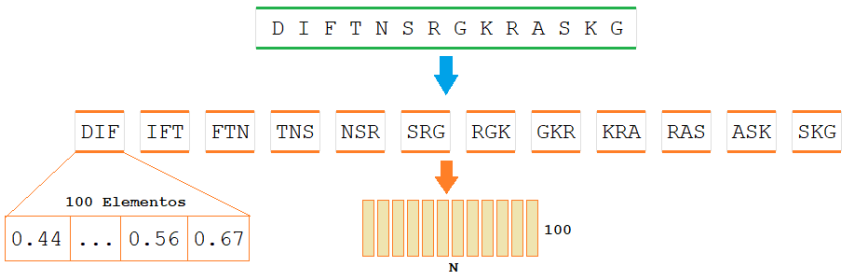


Figura 3.1: Ejemplo del proceso de codificación de una cadena de aminoácidos (en verde) en una lista de vectores de dimensión 100 (en amarillo) utilizando ProtVec.

3.2. Vectorización de las secuencias de aminoácidos

En el caso de las redes neuronales recurrentes y las redes neuronales convolucionales, se requiere una representación secuencial, en la que se mantenga el orden en que se encuentran los aminoácidos en la cadena, que vienen representados por letras. A esto se le llama codificador (*encoder* en inglés), y nos debe permitir convertir la cadena de letras a una representación numérica y poder revertir esa transformación recuperando la cadena de letras original. El codificador que se utiliza en este trabajo es el presentado en ProtVec (ver Sección 2.6), que provee una representación vectorial de 100 dimensiones para cada secuencia de tres aminoácidos que se pueda encontrar en un péptido.

El preprocesamiento de la cadena de aminoácidos consiste en separarla en trigramas. Estos son los grupos superpuestos de tres aminoácidos que se pueden encontrar a lo largo de un péptido. Ejemplo de una conversión se muestra en la Figura 3.1.

3.3. Modelo híbrido

3.3.1. Modelo de red basada en DLBEpitope

Para tener un punto de comparación, se propuso realizar una réplica del modelo DLBEpitope utilizando el mismo procedimiento que describen en su reporte. La medida de desempeño que utilizan para caracterizar su modelo es

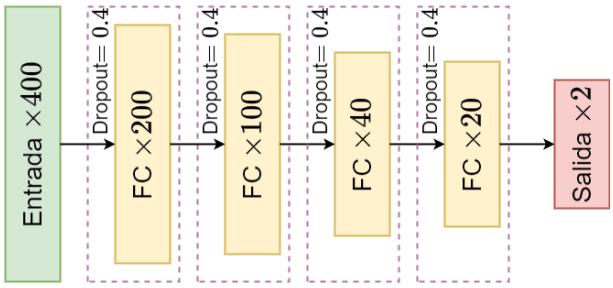


Figura 3.2: Arquitectura de la red neuronal presentada en DLBEpitope.

AUC, por lo que se tomó ese valor como referencia para verificar la similitud de los resultados de la réplica.

La base de datos de entrenamiento que generaron para entrenar y evaluar su modelo fue extraída del banco de datos de IEDB. Consiste en 20,000 muestras de epítopos positivos, y cerca de 200,000 de epítopos negativos. Debido al gran desbalance en el número de muestras positivas con respecto a las negativas, optaron por utilizar un método denominado ensamble, que consiste en generar un número determinado de modelos con una arquitectura idéntica pero entrenados con subconjuntos más balanceados del conjunto de entrenamiento. La predicción final del sistema corresponde con el promedio de las predicciones de todos los modelos que conforman el ensamble.

En el caso de DLBEpitope generaron 11 subconjuntos de entrenamiento, cada uno incluye las 20,000 muestras positivas disponibles, y 20,000 muestras negativas elegidas de manera aleatoria de entre las 200,000 disponibles. Estas muestras han sido modificadas para utilizarse como entrada en la red neuronal recortando o añadiendo a las cadenas de epítopos, con el objetivo de que todas las secuencias tengan un mismo tamaño.

La arquitectura de DLBEpitope, como se ve en la Figura 3.2, consiste en redes neuronales completamente conectadas, con una entrada de 400 elementos, cuatro capas ocultas con 0.4 de dropout: 200 neuronas en la primera capa, 100 neuronas en la segunda, 40 en la tercera y 20 en la cuarta. Finalmente, dos neuronas en la capa de salida fungen como clasificador, una neurona para la clase positiva, y la otra para la negativa. Cada red del ensamble fue entrenada durante 400 épocas, con 8000 muestras por lote, y utilizando una tasa de aprendizaje de 0.001. El ensamble, por lo tanto, consistió en 11 redes con la arquitectura descrita. Cada red individual tiene un total de 315,608 parámetros.

Después de realizar la réplica del sistema de DLBEpitope, se repitió el proceso creando un ensamble de redes neuronales basadas en esa misma arquitectura, pero realizando el entrenamiento con el conjunto de datos propuesto y que contiene las secuencias originales de la IEDB (ver Sección 3.1). Esto es posible realizarlo pues la codificación que se utiliza en DLBEpitope para convertir cada secuencia de aminoácidos en un vector de 400 elementos para alimentar a la red no tiene restricción con respecto a la longitud de la secuencia.

3.3.2. Modelo de red recurrente de tipo LSTM

La idea de construir una red recurrente con celdas de tipo LSTM parte de la amplia evidencia que demuestra un desempeño superior de estas sobre las RNN tradicionales. La arquitectura que se emplea consiste en una primera capa de codificación que utilizando ProtVec transforma la cadena de trigramas de longitud variable a su representación vectorial, dos capas de celdas LSTM bidireccionales, la primera de tamaño 100 y la segunda de 40, una capa densa de 40 neuronas, una capa densa de 20 neuronas y una capa de clasificación binaria con dos neuronas de salida (Figura 3.3). El total de parámetros de esta arquitectura LSTM es de 1,630,968.

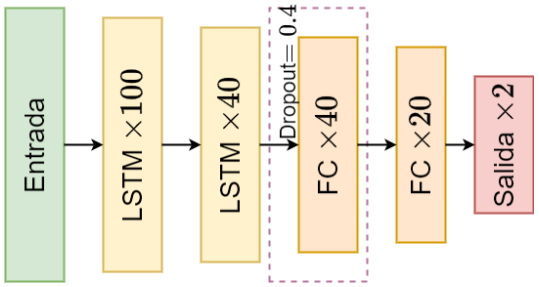


Figura 3.3: Arquitectura propuesta de red neuronal con celdas LSTM.

3.3.3. Modelo de red neuronal convolucional

Se utilizó una versión de red neuronal convolucional basada en la arquitectura de GoogleNet (Szegedy et al. 2015) utilizando convoluciones a lo largo de una dimensión. La cadena de aminoácidos más grande en el conjunto de entrenamiento es de 50 de longitud. Dado que las muestras con las que se

cuenta son de longitud variable se agregó un relleno (padding) de ceros para que todas tuvieran el mismo tamaño. La arquitectura de la red se conforma por una capa de codificación de la cadena, seguida por una capa convolucional de 96 filtros de tamaño 3×3, cuatro módulos de incepción, y un módulo de submuestreo, dos módulos de incepción, una capa de *average pooling*, una capa densa de 20 neuronas, y dos celdas de salida para la clasificación (Figuras 3.4 y 3.5). Esta arquitectura convolucional consta de un total de 1,791,024 parámetros.

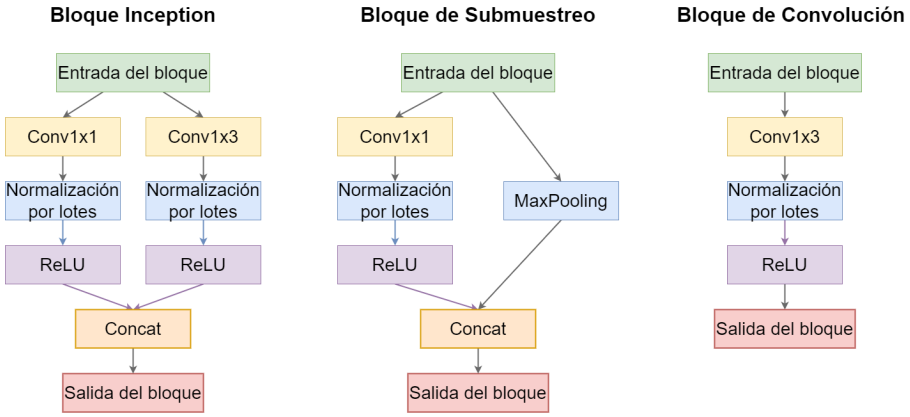


Figura 3.4: Módulos que componen la arquitectura de la red neuronal convolucional propuesta.

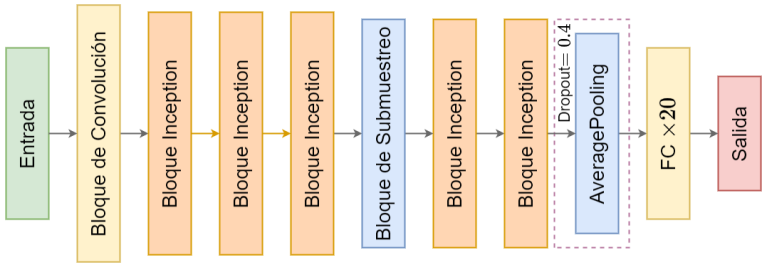


Figura 3.5: Arquitectura de la red neuronal convolucional propuesta basada en la red GoogLeNet.

3.3.4. Creación del modelo híbrido

Finalmente, se propone una arquitectura híbrida que concatena la última capa oculta de las redes FCN, RNN, y Convolucional presentadas (secciones 3.3.1, 3.3.2 y 3.3.3), para luego procesar la información en una última sección

completamente conectada. A este modelo lo llamaremos **modelo híbrido**.

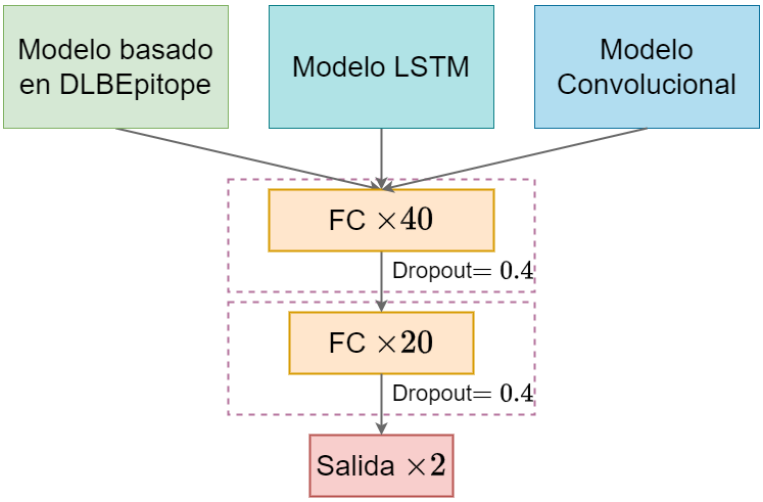


Figura 3.6: Arquitectura del modelo híbrido propuesta que combina la salida de la última capa oculta de una FCN, una red LSTM y una red convolucional.

La razón es que cada una de las tres arquitecturas con las que se ha experimentado extrae características de las cadenas de aminoácidos de una manera diferente. Podemos aprovechar el vector obtenido (vector latente) en la última capa oculta antes de la capa de salida de cada una de estas tres arquitecturas y conectarlo a una sección de capas densas para así generar un clasificador más robusto. Se congelaron los pesos de los clasificadores (ya no se modificaron) y solo se entrenaron esas ultimas capas agregadas esperando que este modelo pueda aprovechar la información más relevante de estos vectores que resultan de cada una de estas tres arquitecturas. Esta arquitectura suma un total de 2,463,396 parámetros de los cuales 3,302 se entrenaron, correspondientes con la última sección completamente conectada.

3.4. Metodología de entrenamiento

Todos los modelos presentados fueron entrenados utilizando la técnica de ensamble. Para cada arquitectura se generaron 11 modelos diferentes que fueron entrenados, validados y evaluados con los conjuntos de datos descritos en la Sección 3.1, previamente preprocesados utilizando el método adecuado para cada arquitectura, y para todos se utilizó el optimizador Adam (Kingma

y Ba 2014) y la función de perdida *binary cross-entropy*. La búsqueda de hiperparámetros se realizó de manera similar para todos los modelos. Primero, se realizó el entrenamiento de una sola red buscando el mejor desempeño de validación, considerando la tasa de aprendizaje, el tamaño de lote y el número de épocas antes de un sobreajuste. Cuando se encontraban los hiperparámetros se procedió a la generación y entrenamiento del ensamble completo, esto por cada arquitectura.

3.5. Metodología de evaluación

Para realizar una comparación confiable de los sistemas generados se emplearon las métricas de evaluación que comúnmente se usan en estos problemas de clasificación binaria. Se calcularon la precisión, la exhaustividad, y el F1-score, para cada ensamble. Además, se incluyó la exactitud, y se agregó la creación de la curva ROC y el cálculo del AUC. Estas evaluaciones se realizaron durante el entrenamiento para llevar un control con el conjunto de epítopos de validación, y posteriormente cuando los ensambles estaban listos, se evaluó cada uno con el conjunto de epítopos de prueba.

Capítulo 4

Resultados

En este capítulo se muestran los resultados del desempeño de cada ensamble de modelos descrito, evaluando cada uno utilizando el conjunto de prueba de 4024 muestras de péptidos de longitud variable (como aparecen en la IEDB), con base en las métricas de precisión, exhaustividad, F1-score, exactitud, y el área bajo la curva ROC (AUC). Los hiperparámetros para cada modelo son los que se muestran en el Cuadro 4.1.

Modelo	**Tasa de aprendizaje**	**Épocas**	**Tamaño de lote**
Modelo basado en DLBEpitope	3.0E-5	250	256
Modelo LSTM	1.0E-4	40	64
Modelo Convolucional	1.0E-5	35	64
Modelo Híbrido	1.0E-5	18	64

Cuadro 4.1: Hiperparámetros utilizados para entrenar los modelos en cada ensamble generado.

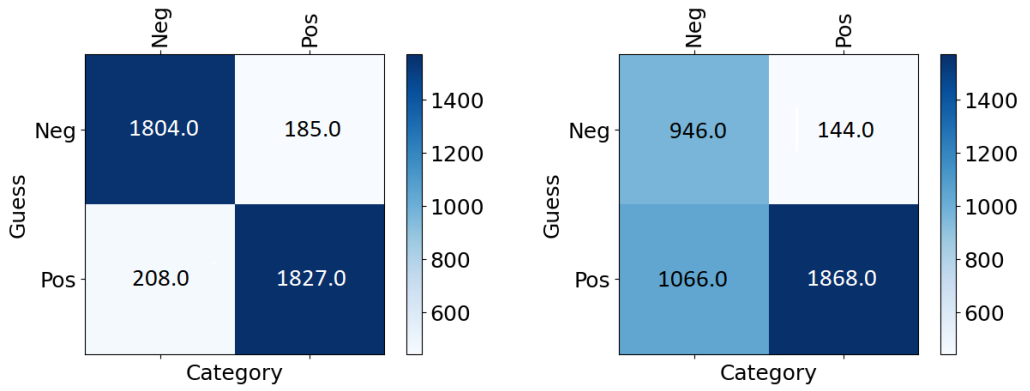
4.1. Resultados de la réplica de DLBEpitope

En esta sección se presenta el desempeño obtenido por la replica del ensamble de DLBEpitope, entrenado siguiendo los criterios que muestran en su artículo. Se hace la comparación del resultado al evaluar sobre el conjunto

de 4024 péptidos modificados, con el resultado al evaluar sobre el conjunto de 4024 péptidos originales equivalentes, tal como aparecen en IEDB (Cuadro 4.2 y Figuras 4.1 y 4.2). Esto prueba el comportamiento del modelo de DLBEpitope al utilizarlo para intentar etiquetar secuencias de longitud variable.

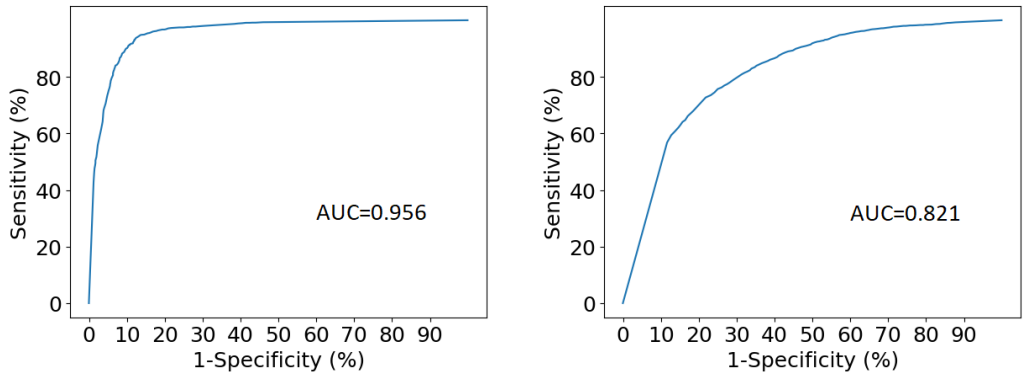
Modelo	**Precisión**	**Exhaustividad**	**F1 Score**	**Exactitud**	**AUC**
Con péptidos modificados	0.897	0.908	0.902	0.902	0.956
Con péptidos originales de IEDB	0.636	0.928	0.755	0.699	0.821

Cuadro 4.2: Resultados de la réplica del ensamble de DLBEpitope, al evaluarlo con a) el conjunto de muestras de péptidos modificados con su método y b) con el conjunto de muestras sin modificar (como aparecen en IEDB).



(a) (b)

Figura 4.1: Matrices de confusión del modelo de DLBEpitope con a) las 4024 secuencias de prueba de su artículo y b) las 4024 secuencias originales de la IEDB.

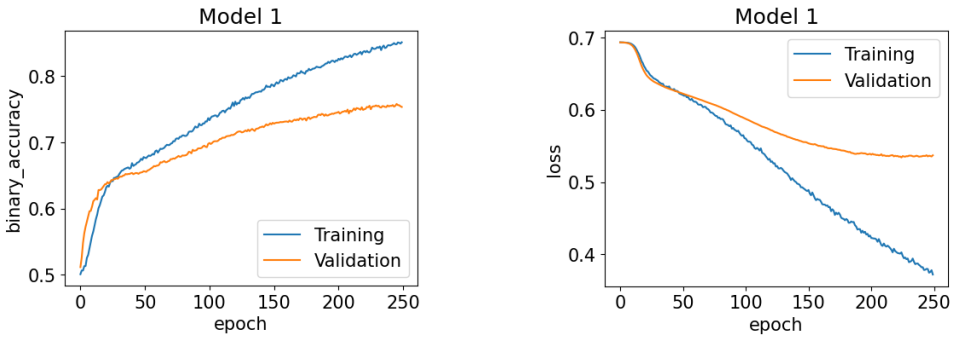


(a) (b)

Figura 4.2: Curvas ROC y su valor de AUC del modelo de DLBEpitope evaluado a) utilizando las 4024 secuencias de prueba de su artículo y b) utilizando las 4024 secuencias originales de la IEDB.

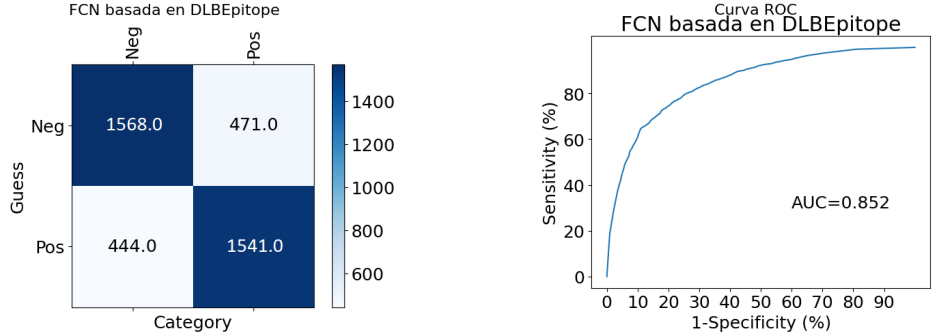
4.2. Resultados de la FCN basada en el método de enamble de DLBEpitope

Los resultados de esta sección corresponden con el desempeño del modelo de FCN utilizando el método ensamble de DLBEpitope consistente en 11 subconjuntos de epítopos originales de la IEDB.



(a) (b)

Figura 4.3: Ejemplos de curvas de (a) la función de evaluación y (b) la función de pérdida para el conjunto de entrenamiento y validación durante el entrenamiento de un modelo FCN.

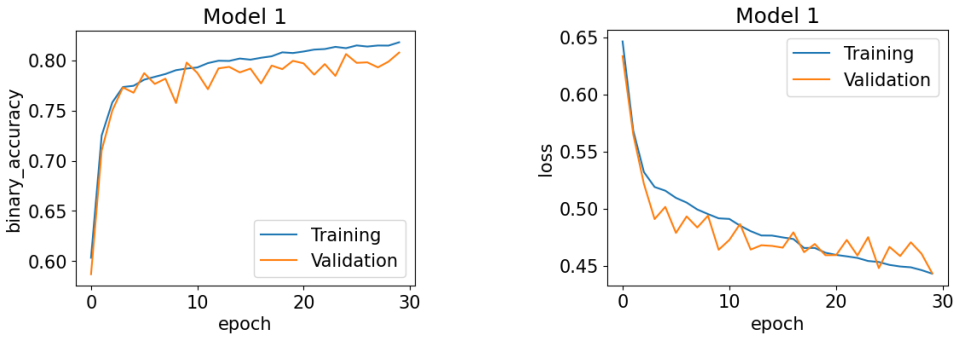


(a) (b)

Figura 4.4: a) Matriz de confusión del modelo FCN con las 4024 muestras de péptidos de prueba. b) Curva ROC y el valor del área bajo la curva.

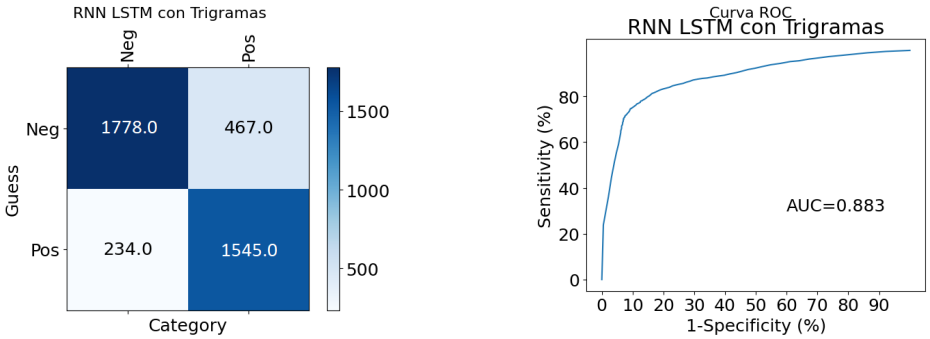
4.3. Resultados del ensamble de RNN con celdas LSTM

Los resultados de esta sección corresponden con el desempeño del modelo de red neuronal recurrente con celdas LSTM, utilizando ProtVec como codificador para las secuencias de aminoácidos.



(a) (b)

Figura 4.5: Ejemplos de curvas de (a) la función de evaluación y (b) la función de pérdida para el entrenamiento y validación de un modelo RNN con celdas LSTM.

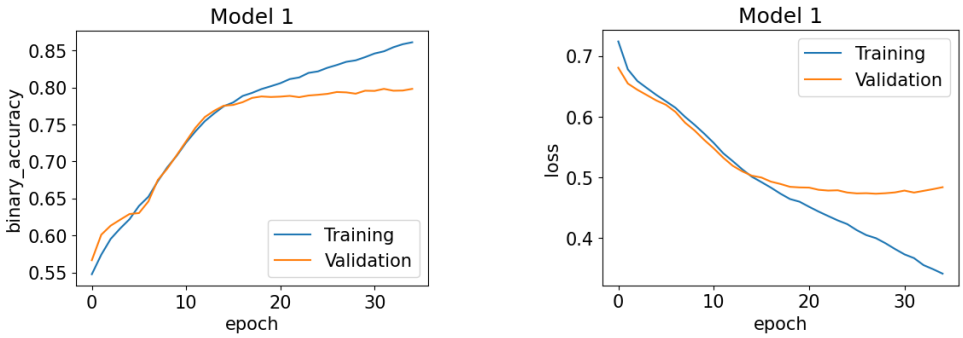


(a) (b)

Figura 4.6: a) Matriz de confusión del ensamble de redes RNN con celdas LSTM con el conjunto de prueba. b) Curva ROC y el valor del AUC.

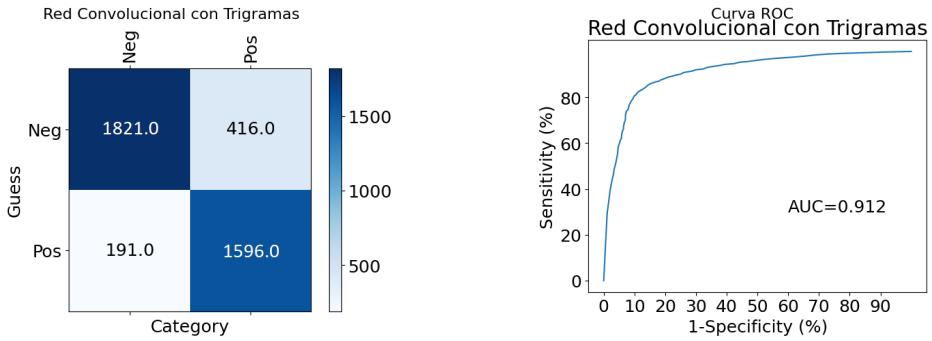
4.4. Resultados del ensamble de redes convolucionales

Los resultados de esta sección corresponden con el desempeño del modelo de red neuronal convolucional basado en el modelo reportado en GoogLeNet, utilizando ProtVec como codificador para las secuencias de aminoácidos.



(a) (b)

Figura 4.7: Ejemplos de curvas de (a) la función de evaluación y (b) la función de pérdida para el entrenamiento y validación de un modelo convolucional.

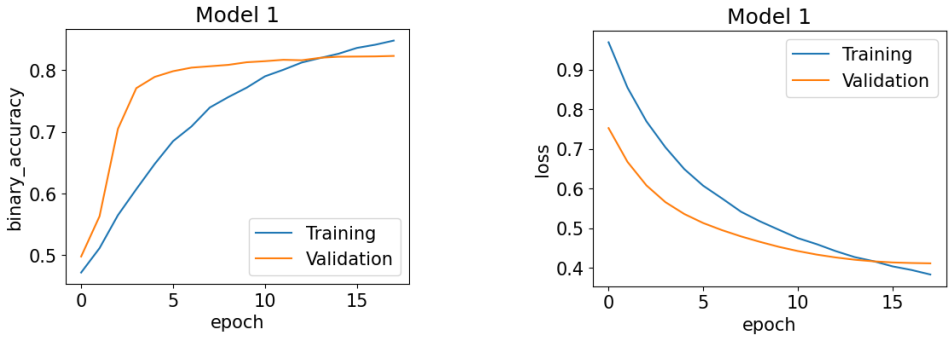


(a) (b)

Figura 4.8: a) Matriz de confusión del ensamble de redes convolucionales empleando el conjunto de péptidos de prueba. b) Curva ROC y el valor del AUC.

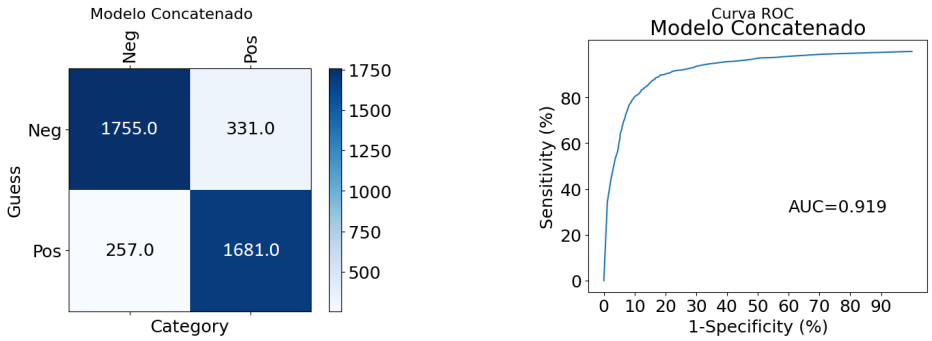
4.5. Resultados del ensamble de modelos concatenados

Los resultados de esta sección corresponden con el desempeño del modelo propuesto de red neuronal concatenado. El modelo recibe cada secuencia en tres formas diferentes, una adecuada para cada arquitectura concatenada.



(a) (b)

Figura 4.9: Ejemplos de curvas de (a) la función de evaluación y (b) la función de pérdida para el entrenamiento y validación de un modelo concatenado.



(a) (b)

Figura 4.10: a) Matriz de confusión del ensamble de modelos concatenados sobre el conjunto de prueba. b) Curva ROC y el valor del AUC.

4.6. Comparación de resultados de los modelos FCN, RNN con celdas LSTM, Red Convolucional, y modelo Concatenado

En esta sección se muestra el cuadro comparativo 4.3 con los resultados obtenidos para cada una de las métricas evaluadas en los cuatro ensambles que se entrenaron. En el cuadro 4.4 se incluye secuencias de ejemplo del

conjunto de prueba y la predicción asignada por los modelos con mejor desempeño: el convolucional y el concatenado.

Modelo	**Precisión**	**Exhaustividad**	**F1 Score**	**Exactitud**	**AUC**
FCN	0.776	0.766	0.771	0.773	0.852
Red LSTM	0.868	0.768	0.815	0.826	0.883
Red Convolucional	**0.893**	0.793	0.840	0.849	0.912
Modelo Concatenado	0.867	**0.835**	**0.851**	**0.854**	**0.919**

Cuadro 4.3: Cuadro comparativo con los resultados de la evaluación utilizando el conjunto de péptidos de prueba en cada uno de los ensambles generados.

Secuencia	**Etiqueta real**	**Predicción Ensamble CNN**	**Predicción Ensamble Híbrido**
SAKAATAPAKAAAAP	Positivo	0.967	0.996
RPCEQHLMQKI	Positivo	0.827	0.965
WGENDTDVFVLNNTR	Positivo	0.133	0.155
AAWGGSGSEAYQGVQQKWDATA	Positivo	0.999	0.997
EKRKAAEATKIAEAE	Positivo	0.997	0.996
GSLAEYDAVSGVPPY	Negativo	0.144	0.097
PNELNTPKGIAYMSV	Negativo	0.270	0.212
GEKDTRSPQPPQPPP	Negativo	0.329	0.133
TFGVPRRQRAID	Negativo	0.967	0.973
LIRRNHFAMAKTIAYDEEARRG	Negativo	0.984	0.980

Cuadro 4.4: Cuadro comparativo de las predicciones del modelo convolucional y el modelo concatenado con algunos ejemplos de las secuencias de epítopos de evaluación.

Capítulo 5

Conclusiones

En general todos los ensambles generados muestran un buen desempeño pues se encuentran por arriba de 0.75 en todas las métricas obtenidas. Sin embargo, para el objetivo final de predicción de epítopos, es preferible tener valores de exactitud lo más cercanos a 1.0.

Con la evidencia cuantitativa de los resultados se puede concluir que de las cuatro arquitecturas la que obtuvo mejor desempeño es la propuesta de modelos concatenados. La idea del modelo concatenado es utilizar la extracción de características de las tres arquitecturas diferentes (FCN, RNN, y Convolucional), ya que cada una lo hace utilizando un enfoque diferente. No obstante, hay que resaltar que la diferencia del ensamble del modelo concatenados no es muy grande con respecto al ensamble del modelo de convolución.

El siguiente paso es verificar en laboratorio la predicción del modelo concatenado utilizando una muestra seleccionada de epítopos etiquetados como positivos.

Hay que considerar que, si bien la base de datos utilizada contiene un gran número de muestras de epítopos etiquetados, también significa que contienen epítopos de diferentes especies de patógenos y huéspedes. Si se considerase continuar el proyecto utilizando los modelos propuestos, se sugiere realizar un segundo entrenamiento con el objetivo de hacer un ajuste fino utilizando una base de datos más reducida y filtrada, por ejemplo, de epítopos que solo se encuentren en pacientes humanos, o que provengan de alguna familia de patógenos determinada.

Bibliografía

Abbott, W. M., M. M. Damschroder y D. C. Lowe (2013). "Current approaches to fine mapping of antigen-antibody interactions". En: *Immunology* 142.4, págs. 526-535. URL: https://doi.org/10.1111/imm.12284.

Asgari, E y M Mofrad (2015). "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics". En: *PLOS ONE* 10.11. DOI: e0141287. URL: https://doi.org/10.1371/journal.pone.0141287.

Bridle, John S. (1989). "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters". En: NIPS'89, págs. 211-217.

Dhanda, Sandeep Kumar et al. (mar. de 2016). "Novel in silico tools for designing peptide-based subunit vaccines and immunotherapeutics". En: *Briefings in Bioinformatics* 18.3, págs. 467-478. ISSN: 1467-5463. DOI: 10.1093/bib/bbw025. eprint: https://academic.oup.com/bib/article-pdf/18/3/467/25408658/bbw025.pdf. URL: https://doi.org/10.1093/bib/bbw025.

Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). *Deep Learning*. The MIT Press.

Hochreiter, Sepp y Jürgen Schmidhuber (1997). "Long Short-Term Memory". En: *Neural Computation* 9.8, págs. 1735-1780. DOI: 10.1162/neco.1997.9.8.1735.

Kingma, Diederik y Jimmy Ba (dic. de 2014). "Adam: A Method for Stochastic Optimization". En: *International Conference on Learning Representations*.

Lecun Y., et al. (1995). "Convolutional networks for images, speech, and time series. The Handbook of Brain Theory and Neural Networks." En: *MIT Press*, pág. 3361.

Liu, Tao, Kaiwen Shi y Wuju Li (abr. de 2020). "Deep Learning Methods Improve Linear B- cell epitope prediction". En: *BioData Mining* 13.1. DOI: 10.1186/s13040-020-00211-0.

Nair, V. y G. E. Hinton (2010). "Rectified linear units improve restricted Boltzmann machines". En: *ICML*.

Nelson, David L y Michael M Cox (2017). *Lehninger Principles of Biochemistry, Fourth Edition*. Ed. por WH Freeman. 7th ed.

Palatnik-de-Sousa, Clarisa, Irene da Silva Soares y Daniela Santoro Rosa (2018). "Editorial: Epitope Discovery and Synthetic Vaccine Design". En: *Front. Immunol* 8.826. DOI: 10.3389/fimmu.2018.00826. URL: https://www.frontiersin.org/journals/immunology/articles/10.3389/fimmu.2018.00826/full.

Rodwell, Victor et al. (2018). *Harper Bioquímica ilustrada*. Ed. por McGrawHill Education. LANGE.

Saha, Sudipto, Manoj Bhasin y Gajendra Raghava (mayo de 2005). "Bcipep: A database of B-cell epitopes". En: *BMC Genomics* 6.1. DOI: 10.1186/1471-2164-6-79.

Saha, Sudipto y Gajendra Raghava (ago. de 2006). "Prediction of continuous b-cell epitopes in an antigen using recurrent neural network". En: *Proteins: Structure, Function, and Bioinformatics* 65.1, págs. 40-48. DOI: 10.1002/prot.21078.

Singh, Harinder, Hifzur Rahman Ansari y Gajendra Raghava (mayo de 2013). "Improved method for Linear B- cell epitope prediction using antigen's primary sequence". En: *PLoS ONE* 8.5. DOI: 10.1371/journal.pone.0062216.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". En: *Journal of Machine Learning Research* 15.56, págs. 1929-1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Staudemeyer, Ralf C. y Eric Rothstein Morris (2019). "Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks". En: *CoRR* abs/1909.09586.

Szegedy, Christian et al. (jun. de 2015). "Going deeper with convolutions". En: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr.2015.7298594.

Zumdahl, Steven S. y Donald J. DeCoste (2012). *Principios de química*. 7.ª ed. Cengage Learning.