UNIVERSIDAD JUÁREZ AUTÓNOMA DE TABASCO

DIVISIÓN ACADÉMICA DE CIENCIAS BÁSICAS

ESTUDIO DE REDES NEURONALES CON APLICACIONES AL ÁREA DE LA SALUD

T E S I S

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS MATEMÁTICAS

PRESENTA:

LIC. SAÚL DAVID CANDELERO JIMÉNEZ

BAJO LA DIRECCIÓN DE:

DRA. ADDY MARGARITA BOLÍVAR CIMÉ
DR. AROLDO PÉREZ PÉREZ

CUNDUACÁN, TABASCO A 30 DE JULIO DE 2026

Declaración de Autoría y Originalidad

En la ciudad de Villahermosa, el día 03 del mes de julio del año 2026, el que suscribe **Saúl David Candelero Jiménez**, alumno del Programa de la Maestría en Ciencias Matemáticas, con número de matricula 232A21002, adscrito a la División Académica de Ciencias Básicas, de la Universidad Juárez Autónoma de Tabasco, como autor de la Tesis presentada para la obtención del grado de Maestría en Ciencias y titulada **"Estudio de redes neuronales con aplicaciones al área de la salud"** dirigida por **Dra. Addy Margarita Bolívar Cimé** y **Dr. Aroldo Pérez Pérez**.

DECLARO QUE:

La Tesis es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, de acuerdo con el ordenamiento jurídico vigente, en particular, la LEY FEDERAL DEL DERECHO DE AUTOR (Decreto por el que se reforman y adicionan diversas disposiciones de la Ley Federal del Derecho de Autor del 01 de Julio de 2020 regularizando y aclarando y armonizando las disposiciones legales vigentes sobre la materia), en particular, las disposiciones referidas al derecho de cita. Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad o contenido de la Tesis presentada de conformidad con el ordenamiento jurídico vigente

Villahermosa, Tabasco a 03 de julio de 2026.

Lic. Saúl David Candelero Jiménez

DIRECCIÓN

24 de junio de 2026

LIC. SAÚL DAVID CANDELERO JIMÉNEZ
EGRESADO MAESTRÍA EN CIENCIAS MATEMÁTICAS
PRESENTE

Por medio del presente y de la manera más atenta, me dirijo a Usted para hacer de su conocimiento que se le **AUTORIZA** la impresión del trabajo de Tesis titulado "**ESTUDIO DE REDES NEURONALES CON APLICACIONES AL ÁREA DE LA SALUD***"* bajo la Dirección de Dra. Addy Margarita Bolívar Cimé y Dr. Aroldo Pérez Pérez.

La Comisión revisora conformada por el **Dr. Edilberto Nájera Rangel, Dr. Fidel Ulín Montejo, Dra. Addy Margarita Bolívar Cimé, Dr. Jorge López López y Dr. Jair Remigio Juárez** aprobó el documento en virtud de reunir los requisitos para el EXAMEN PROFESIONAL y obtener el grado de **Maestro en Ciencias Matemáticas** bajo la modalidad de titulación por Tesis.

Sin más por el momento, reciba un cordial saludo.

ATENTAMENTE

DIVISIÓN ACADÉMICA DE CIENCIAS BÁSICAS

DRA. HERMICENDA PÉREZ VIDAL
DIRECTORA

C.C.P.- Archivo.

*Dir´Dra.HPV/JP´Dra.LAR/jkal***

Carta de Cesión de Derechos

Villahermosa, Tabasco a 03 de julio 2026

Por medio de la presente manifestamos haber colaborado como AUTOR y/o AUTORES en la producción, creación y/o realización de la obra denominada **“Estudio de redes neuronales con aplicaciones al área de la salud”**.

Con fundamento en el artículo 83 de la Ley Federal del Derecho de Autor y toda vez que, la creación y/o realización de la obra antes mencionada se realizó bajo la comisión de la Universidad Juárez Autónoma de Tabasco; entendemos y aceptamos el alcance del artículo en mención, de que tenemos el derecho al reconocimiento como autores de la obra, y la Universidad Juárez Autónoma de Tabasco mantendrá en un 100% la titularidad de los derechos patrimoniales por un período de 20 años sobre la obra en la que colaboramos, por lo anterior, cedemos el derecho patrimonial exclusivo en favor de la Universidad.

COLABORADORES

Egresado
Lic. Saúl David Candelero Jiménez

Directora
Dra. Addy Margarita Bolívar Cimé

Codirector
Dr. Aroldo Pérez Pérez

TESTIGOS

Dr. Jair Remigio Juárez

Dr. Jorge López López

A Dios, quien me ha dado sabiduría,
fortaleza y prudencia para seguir adelante.

A mis padres, que siempre
me han apoyado incondicionalmente.
Por ellos he podido llegar tan lejos.

A Anahí, mi amada esposa.
Sin ella no sé donde estaría hoy.

A Cindy, Sarai, Iojany, Andry, Ronaldo, Isaac y Didier,
Gracias por su amistad.

A todos los que me acompañaron,
a los profesores que me tomaron cariño,
por todo, muchas gracias.

Índice general

Índice de figuras

Lista de Tablas

Resumen

A lo largo de este trabajo se estudia la teoría de redes neuronales como modelos de predicción, iniciando con la interpretación biológica hasta llegar a las bases matemáticas que sustentan el uso de éstas. Posteriormente, se pasa a la construcción de las redes neuronales usando programación en python y se explica la elección del mejor modelo a partir del empleo de métodos de validación cruzada. Todos estos pasos se usan en conjunto para la construcción de modelos de redes neuronales con aplicación a bases de datos médicos, de los cuales se obtienen ciertas conclusiones importantes.

Palabras clave: - Red neuronal, aprendizaje profundo, gradiente descendente, validación cruzada, aproximación universal, clasificación.

Abstract

This work explores neural network theory as a predictive model, beginning with its biological interpretation and progressing to the mathematical foundations that underpin its use. We then examine the construction of neural networks using Python programming and explain how the best model is selected through cross-validation methods. All these steps are combined to build neural network models applicable to medical databases, from which we can draw certain important conclusions.

Keywords: - Neural networks, deep learning, gradient descent, cross-validation, universal approximation, classification.

Introducción

En la actualidad, debido en parte a la velocidad y capacidad de los ordenadores, las redes neuronales, también conocidas como redes neuronales artificiales, se han convertido en un objeto de estudio importante. La idea de recrear el trabajo realizado por las neuronas humanas, con el fin de imitar el aprendizaje y reconocimiento humano, ha encontrado aplicación en múltiples áreas, como por ejemplo, en el procesamiento de imágenes [3], de voz, audio y señales biomédicas [27], optimización [34], aplicaciones a la ingeniería, medicina y finanzas [14], entre otras. Como se menciona en [16], el procesamiento de información del cerebro humano es muy distinto al procesamiento secuencial de un ordenador convencional, es decir, las redes neuronales buscan replicar el sistema altamente complejo, no lineal y paralelo con el que nuestro cerebro trabaja; de modo que se busca que las principales características de una red neuronal sean:

(1) Aprendizaje de forma natural, adquirir conocimiento en base a la experiencia, el cual debe ser almacenado en el peso relativo de las conexiones interneuronales, tal como en el cerebro.

(2) Adaptabilidad, que sean capaces de cambiar su dinámica sin importar el medio.

(3) Tolerancia a fallos, que sigan teniendo un buen comportamiento aún cuando presentaran algún daño.

(4) Altamente no lineales, que posean la capacidad de recibir la información de otros fenómenos no lineales.

Uno de los lenguajes más usados para el desarrollo de redes neuronales y distintos tipos de learning es Python, proporcionando velocidad y facilidad en paqueterías especializadas que pueden ser oportunas para el desarrollo de las aplicaciones médicas.

El objetivo de esta tesis es estudiar el desarrollo de redes neuronales usando el lenguaje de programación Python, comprender el contexto en el que se relacionan con el cerebro humano, así como proporcionar ejemplos de aplicaciones al área de la salud.

Marco teórico

Las redes neuronales como modelos computacionales surgen de los trabajos de diversos autores que buscaban representar matemáticamente el procesamiento de información en sistemas biológicos [5], sin embargo, su origen se puede ubicar en 1943, año en que Warren McCulloch, psiquiatra y neuroanatomista, junto al prodigioso matemático Walter Pitts dieron el primer modelo simplificado de la neurona [19], [22]. En adelante, varios trabajos teóricos continuaron extendiendo los resultados obtenidos por McCulloch y Pitts, pero fue hasta 1958 en que Frank Rosenblatt dió un paso adelante con su trabajo sobre el perceptron, una clase de "modelo del cerebro", el cual sería un nuevo acercamiento al reconocimiento de patrones, lo cual fue un paso bastante importante hacia las redes neuronales artificiales, pues introdujo la idea de aprendizaje autónomo mediante patrones frecuentes [32], [29].

Pese al gran recibimiento de esta teoría, el campo de investigación se vería colapsado con la publicación, en 1969, del libro "Perceptrons" de Minsky y Seyomur Papert [23], quienes demostraron matemáticamente que existen limitantes fundamentales sobre lo que los perceptrones de una sola capa pueden calcular y expresaron además dudas en si las versiones multicapa podrían ser mejores. Lo anterior condujo a un declive en la financiación continua para la investigación en esta área durante las siguientes dos décadas. Sin embargo, en la década de los 80's se realizaron importantes contribuciones a la teoría y el diseño de redes neuronales, como por ejemplo, el principio de auto-organización conocido como teoría de la resonancia adaptativa de Goossberg (1980), que de hecho, el origen de este principio se remonta al trabajo pionero de muchos otros investigadores [13]. Debido a esto, hubo un resurgimiento del interés en las redes neuronales, que tomó especial impulso en la década de los 90's [13], en que Vapnik y colaboradores inventaron una clase computacionalmente poderosa de redes de aprendizaje supervisado "Support vector machines", para resolver problemas de reconocimiento de patrones, regresión y problemas de estimación de densidad. A partir de esto, año con año se realizan una amplia cantidad de trabajos usando redes neuronales artificiales. En la medicina, por ejemplo, se utilizan como herramienta para la detección y elección del tratamiento adecuado de la apendicitis en niños, como modelo de clasificación para la enfermedad del alzheimer y el deterioro cognitivo leve mediante el procesamiento de imágenes de resonancia magnética [14], entre otras aplicaciones.

Justificación

El diagnóstico temprano y la prevención de enfermedades es un problema de suma importancia, pero a pesar de contar con el asesoramiento médico oportuno, algunas enfermedades requieren de cierta toma de decisiones sobre posibles tratamientos en base a los tipos y/o niveles de avance de la enfermedad ya que para un médico puede ser difícil el tomar una decisión ante tanta información; lo anterior hace de las redes neuronales una herramienta óptima para la predicción confiable sobre el tratamiento o detección de enfermedades basándose en la experiencia proporcionada para su entrenamiento.

Pregunta de investigación

¿Qué tan eficientes son las redes neuronales para predecir enfermedades comparado a otros modelos de aprendizaje automático?

Hipótesis

Existen diversas aplicaciones en el área de la salud donde la efectividad de estos métodos es relevante.

Objetivo general

El objetivo de esta tesis es estudiar el desarrollo de redes neuronales usando el lenguaje de programación Python, comprender el contexto en el que se relacionan con el cerebro humano, así como proporcionar ejemplos de aplicaciones al área de la salud.

Objetivos específicos

I. Estudiar la teoría detrás de las redes neuronales.

II. Mostrar el uso de las redes neuronales artificiales como análogo del proceso de aprendizaje humano.

III. Construir algunos sistemas de redes neuronales usando el lenguaje de programación Python.

IV. Aplicar las redes neuronales a datos del área de la medicina.

Parte I

Metodología

Capítulo 1: El problema de clasificación

1.1. El sistema nervioso y las neuronas

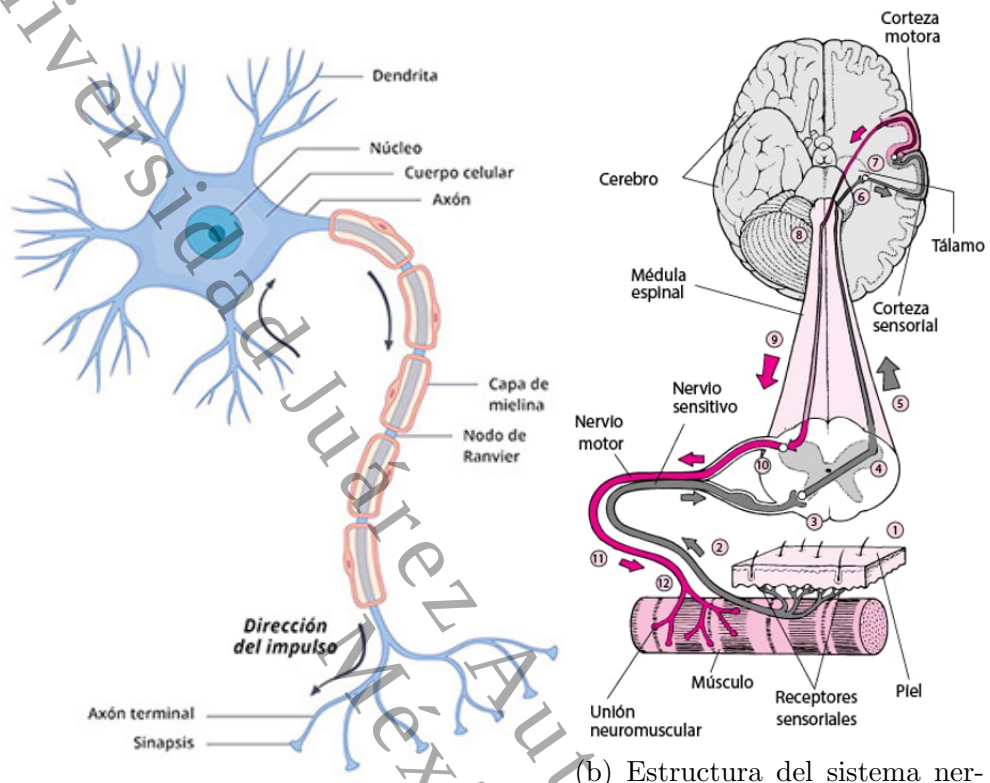
De acuerdo con [28], los centros nerviosos están constituidos principalmente por cuerpos celulares y fibras. Los estudios de Remak, en 1833, aportaron conocimiento sobre la célula nerviosa; mientras que, tres años después, Purkinje describió las prolongaciones de las células, de las cuales, Wagner en 1851 demostró que solo una de las prolongaciones constituía las fibras, hecho que sería confirmado cinco años más tarde por Deiters.

En general, la colección de elementos anatómicos que rigen el funcionamiento del organismo es el sistema nervioso. Este se conforma por dos partes, no totalmente independientes entre si, el sistema nervioso de la vida vegetativa y el sistema nervioso de la vida de relación. El sistema nervioso vegetativo es el que gobierna el funcionamiento de los órganos interiores, por otro lado, el sistema nervioso de la vida de relación, en el cual se halla nuestro objeto de estudio, se encuentra formado por el sistema nervioso cerebroespinal y el sistema nervioso periférico, el primero es una porción central dividida en dos segmentos principales: el encéfalo que es el segmento superior y está dentro de la cavidad craneana, y el segmento inferior que es la médula espinal ubicado dentro del conducto vertebral. De este sistema cerebroespinal, se desprenden cordones nerviosos que conectan con el sistema nervioso periférico, los cuales son también llamados nervios craneales y raquídeos, que se desprenden del encéfalo y la médula, respectivamente. Al conjunto de todos los cordones nerviosos, o simplemente "nervios", se le llama sistema nervioso periférico.

En el funcionamiento del sistema nervioso intervienen ciertos elementos, una es la "señal de entrada", la cual es captada por la neurona sensitiva que recoge las impresiones del exterior o interior del cuerpo, y a su vez, las transmite en forma de influjo nervioso (impulso eléctrico muy débil) a otra neurona o célula efectora (neurona motora), la cual envía dicho influjo por medio de sus prolongaciones al órgano encargado de efectuar la acción. Cabe destacar que, con frecuencia, entre la neurona sensitiva y la neurona efectora, hay neuronas intercaladas en el trayecto del influjo, lo cual hace que el funcionamiento del sistema nervioso sea altamente complejo.

La célula nerviosa está caracterizada por un cuerpo celular que consta de un núcleo y numerosas prolongaciones que forman un conjunto indivisible llamado neurona. Las prolongaciones protoplasmáticas destacan por desprenderse del cuerpo celular por una base más o menos amplia y después ramificarse de forma abundante, como si se tratara de las ramas de un árbol, razón por la que se les dio el nombre de dendritas; otro tipo de prolongación, son las prolongaciones cilindroaxil, que salen de un pequeño cono del lado opuesto a las dendritas, aunque a veces también salen de una prolongación protoplásmica, tales prolongaciones, también llamadas cilindroeje o axón, son prolongaciones de diámetro uniforme y que continúan desde su origen hasta el elemento a

quien está destinado, ver figura 1.1a. Se conoce como sinapsis a la conexión que se presenta entre las dendritas de una neurona con el axón de otra, la cual permite el paso del influjo nervioso entre neuronas, transmitiéndose a través de sus prolongaciones cilindroaxiles.



(a) Estructura de la neurona [15].

(b) Estructura del sistema nervioso central y periférico [30].

Figura 1.1: Sistema nervioso y estructura de la neurona.

1.2. El modelo de la neurona

La fascinación por el funcionamiento del cerebro, en cuanto a realizar tareas altamente complejas en una cantidad de tiempo muy corta, ha llevado a la creación de diversos modelos que buscan replicar el trabajo realizado por las neuronas, pues cada una de éstas, son unidades autónomas que transmiten, procesan y almacenan la información de tal forma que pueden responder a impulsos internos y externos de forma satisfactoria.

Ya se hizo mención de las partes de la neurona, sin embargo, cabe destacar su funcionamiento. Basicamente ocurre lo siguiente: la neurona es estimulada a través de sus entradas y cuando se alcanza cierto umbral, la neurona se activa, pasando una señal hacia el axón, el cual se divide en miles de terminaciones y dependiendo del tamaño de la señal hacen sinapsis con las dendritas de otras neuronas. Se estima que el cerebro está compuesto por alrededor de 10^{11} neuronas que trabajan en paralelo, ya que el procesamiento que realizan las neuronas y la memoria captada por la sinapsis se distribuyen juntos a través de la red. De esta manera, la cantidad de información procesada

y almacenada depende de los niveles del umbral de activación y también del peso dado por cada neurona a cada una de sus entradas.

La eficacia de las neuronas biológicas a la hora de ejecutar tareas altamente complejas se debe, principalmente, a la multiconexión que existe entre ellas, es decir, cada neurona se encuentra conectada con otras de tal forma que puede recibir estímulos de un evento tal cual ocurre, o bien recibir cientos de señales eléctricas con la información aprendida. Cuando llega al cuerpo de la neurona, esta información afecta su comportamiento y también puede afectar a neuronas vecinas o un músculo, el proceso de transportar la información entre neuronas es llevado a cabo por la sinapsis, la cual es un espacio que es ocupado por químicos llamados neurotransmisores. Estos neurotransmisores tienen la función de bloquear o permitir el paso de señales que provengan de otras neuronas. Una vez que es recibida la información, las señales son acumuladas en el cuerpo de la neurona y determina qué hacer. Si el total de señales eléctricas recibidas por la neurona es suficientemente grande, se puede superar el potencial de acción, lo que lleva a que la neurona se active o, por el contrario, permanezca inactiva. Cuando una neurona es activada, es capaz de enviar impulsos eléctricos a las neuronas con las cuales está en contacto y este nuevo impulso eléctrico puede a su vez funcionar como una nueva entrada en las neuronas vecinas o como un estímulo para algún músculo.

Históricamente, fue hasta 1943 en que Warren McCulloch, psiquiatra y neuroanatomista, junto al prodigioso matemático Walter Pitts dieron el primer modelo simplificado de la neurona (véase figura 1.2). Dicho modelo basa su funcionamiento en lo anteriormente descrito y cuenta con los siguientes elementos:

- **Entradas**: Las señales externas que entran a la neurona, denotadas por x_i, donde $i = 1, \ldots, N$ es el índice de la señal entrante y N es el número de señales entrantes, que puede pensarse como el número de dendritas de la neurona. En analogía con el modelo biológico, podemos pensar a las entradas como los impulsos eléctricos que entran (dendritas) y salen (axón) de la neurona por la sinapsis, así en el modelo las x_i's representan las señales recibidas por la neurona que provienen ya sea de los nervios o bien de otras neuronas vecinas y al obtenerse una respuesta, esta se convierte en una nueva x_j saliente de la neurona y que a su vez es una entrada para sus vecinas.
- **Los pesos**: Los valores w_{ij} describen la fuerza de cada conexión, donde $i = 1, 2, \ldots, N$ es el índice de la señal entrante, la cual es recibida por una neurona vecina j y existe una cierta fuerza de conexión entre las neuronas, es decir, la sinapsis entre ambas neuronas.
- **Suma de puntos**: Son funciones simples que recolectan todas las señales de entrada y la ponderan con su respectivo peso, es decir, son funciones de la forma

$$net_j = \sum_{i=1}^{N} w_{ij} x_i + \theta_j,$$

donde θ_j es un sesgo. Tiene como objetivo determinar si la neurona j debe ser activada o permanecer inactiva (en el sentido biológico, ésta está implícita en el

cuerpo de la neurona y es la parte encargada de determinar si se supera o no el potencial de acción).

- **Función de activación**: Esta función transforma la suma de puntos net_j, $f(net_j)$ en la salida de la neurona. En el sistema biológico, es la parte del cuerpo de la neurona encargada de enviar el nuevo impulso eléctrico a las neuronas vecinas o como estímulo a algún músculo.
- **Capa oculta**: En conjunto, lo elementos anteriores describen lo que es una neurona, a un grupo de neuronas en un mismo nivel (esto es, están conectados a los mismos datos de entrada) se le conoce como *capa oculta*.

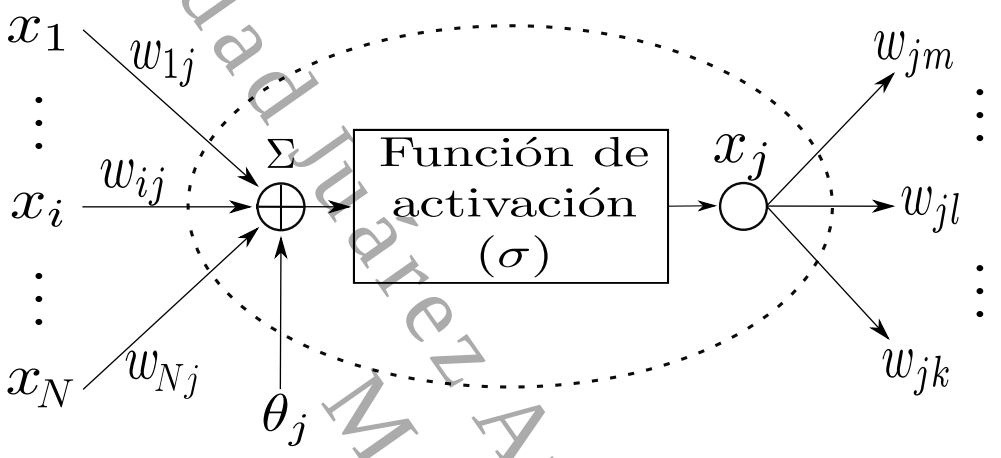


Figura 1.2: Modelo elemental de la neurona.

1.3. Las redes neuronales artificiales

Una red neuronal unicapa (considérese la figura 1.2 sin los w_{jk}'s de la derecha) tiene un nivel de utilidad bajo debido a su poca capacidad de procesar información, esto porque el poder de las redes neuronales radica en la conexión interna de muchas de éstas, tal como pasa en nuestro cerebro. Por otro lado una red neuronal multicapa consiste de varias unidades elementales de procesamiento, llamadas neuronas, las cuales están relacionadas a través de conexiones, son dirigidas y con cada conexión hay un peso asociado. Lo anterior motiva el estudio de las redes neuronales multicapa y vuelve un objetivo, la búsqueda de diferentes arquitecturas que consigan conectar neuronas entre sí. Hablaremos en este apartado acerca de las arquitecturas para conectar neuronas, pero antes necesitamos definir *red neuronal artificial* y *deep learning*.

Para las redes neuronales, Montesinos [24] nos presenta dos definiciones:

Definición 1.3.1 (Francisco-Caicedo & López-Sotelo [12])**.** *Una* ***red neuronal artificial*** *es un sistema compuesto de varios elementos de procesamiento simples, los cuales operan en paralelo y cuya función queda determinada por la estructura de la red y los pesos de las conexiones, donde el procesamiento es llevado a cabo en cada uno de los nodos o elementos informáticos que tienen una baja capacidad de procesamiento.*

Definición 1.3.2 (Kohonen [18]). *Una **red neuronal artificial** es una estructura que consta de elementos simples interconectadas de muchas formas con una organización jerárquica, que intenta replicar el proceso de interacción de los sistemas biológicos nerviosos con objetos del mundo exterior.*

Notemos que ambas definiciones son similares, pues coinciden en que una red neuronal artificial consta de elementos simples que se encuentran interconectados bajo cierta jerarquía o peso; en este caso donde hay interconexiones, estamos pensando en que hay más capas, por lo cual es necesario hablar también del *deep learning* que vendrá a ser un modelo de entrenamiento de las redes.

En este sentido entendemos al *deep learning* como una generalización de redes neuronales artificiales donde se usa más de una capa oculta, lo cual implica que se usan más neuronas para implementar el modelo.

Definición 1.3.3 (Montesinos López, O. A., Montesinos López, A. & Crossa, J. [24]). *Una red neuronal con múltiples capas ocultas recibe el nombre de **Deep Neural Network** (red neuronal profunda), y la práctica de entrenamiento de este tipo de redes se llama deep learning, la cual es una rama del statistical machine learning, donde se usa una topología multicapa para representar las relaciones entre las variables de entrada (variables independientes) y la variable de respuesta (resultado).*

1.4. Redes neuronales y el problema de clasificación

En los inicios de las redes neuronales, alrededor del año 1960, su precursor, el perceptrón, fue desarrollado con la intención de aplicarlo a problemas de reconocimiento de patrones. De hecho, como se menciona en [10], las primeras aplicaciones industriales no triviales de las redes neuronales, durante los inicios de la década de los 80's, estaban relacionadas con el reconocimiento de patrones o señales.

Antes de continuar con los clasificadores por redes neuronales, queremos dejar en claro lo que es un clasificador y las características básicas de los problemas de clasificación. Un clasificador es un algoritmo que asigna automáticamente una clase (o categoría) a un patrón dado. Para entender las características del problema de clasificación, consideremos el ejemplo de aplicación de clasificación automática dado por Dreyfus [10], en el cual, un brazo robótico debe capturar a solicitud, ya sea un capacitor o un circuito integrado, primero debe distinguir entre éstos, en base a una imagen a blanco y negro proporcionada por una cámara de video. A grandes rasgos, los objetos se pueden distinguir mediante su reflectividad y su área, esto porque los circuitos integrados tienden a ser más grandes y oscuros en comparación a los capacitores que suelen ser más pequeños y brillantes. De lo anterior resulta relevante considerar el área A y la reflectividad R como características para la discriminación de los objetos, esto es, asignar un objeto dado a la clase "capacitor" o bien, a la clase "circuito integrado". Supongamos que tenemos recolectada una muestra de capacitores y circuitos integrados en las que tenemos mediciones sobre sus áreas y reflectividades, éstas pueden representarse como puntos sobre el plano, cuyas coordenadas corresponden a su área y reflectividad, tal como se muestra en la figura 1.3.

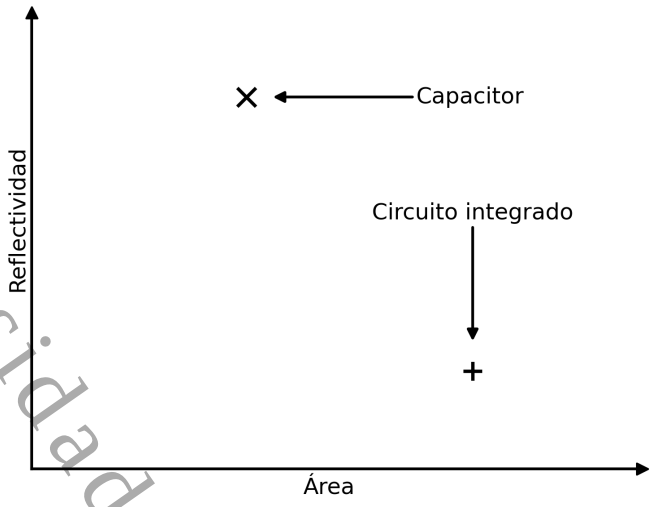


Figura 1.3: Las muestras son representadas por puntos en el plano área-reflectividad.

El ejemplo anterior permite ver los elementos que integran un problema de clasificación:

- Un conjunto de n patrones.
- d variables (o características) que caracterizan a los patrones y juegan un papel relevante en la tarea de asignación de la clase. Al conjunto de características de un patrón dado se le llama *representación* del patrón.
- Un conjunto de clases C que corresponde a las asignaciones de los patrones (a veces resulta conveniente agregar una clase de rechazo, en la cual estarán aquellos patrones que no se puedan asignar a las clases en C).

Tenemos así que resolver un problema de clasificación involucra encontrar una asignación de clases para los elementos del conjunto de patrones.

Algo a tener en cuenta es que las redes neuronales no son un clasificador infalible, es decir, no son adecuados para resolver todos los problemas de clasificación. Una justificación detallada de esto puede consultarse en la sección 1.3.2 del libro de Dreyfus [10], referencia en la que se ejemplifican de forma clara algunos casos en los que el uso de las redes neuronales no es conveniente y dan otros métodos alternativos.

Supongamos pues, que tenemos un problema de clasificación en el cual, el uso de métodos de clasificación probabilista, como las redes neuronales, son apropiados. Los métodos de clasificación probabilista están basados en que tanto las características como las clases funcionan como variables aleatorias. En ese contexto, si seleccionamos de forma aleatoria uno de nuestros datos a clasificar, la clase a la que corresponde resulta de la realización de una variable aleatoria discreta, mientras que las características pueden considerarse como realizaciones de variables aleatorias continuas. De esta manera, en el ejemplo mencionado anteriormente, la "clase " para un capacitor puede ser tomada como el valor 0 y la clase para los circuitos integrados puede ser tomada como el valor 1, siendo así la variable aleatoria discreta "clase" la que toma valores 0 o 1. A su vez la reflectividad R en el área A puede considerarse como una variable aleatoria continua.

Con todos estos elementos podemos resumir el problema de clasificación como sigue:

dado un dato nuevo cuya clase es desconocida, si su reflectividad es r y su área es a, ¿cuál es la probabilidad de que la variable aleatoria discreta "clase" tome el valor 0? Esta probabilidad es conocida como la *probabilidad a posteriori* de la clase "capacitor" dada la reflectividad y el área medida, denotada por

$$P(clase = 0|\{r, a\}).$$

En la siguiente sección, empezamos a abordar el problema de clasificación partiendo desde esto último.

A partir de aquí y hasta terminar el capítulo estaremos analizando la teoría detrás del problema de clasificación binaria, para ello nos basaremos en el libro de Devroye [9].

1.5. El error de Bayes

Para empezar la sección debemos definir lo que es el problema de Bayes, para ello consideremos un par de variables aleatorias X, Y que toman valores en $\mathbb{R}^d$ y $\{0, 1\}$, respectivamente. Este par de variables aleatorias lo podemos describir de múltiples formas, una de ellas sería definirlas por el par (μ, η), donde μ es la medida de probabilidad para X y η es la regresión de Y sobre X. De forma más precisa, para un conjunto Borel medible $A \subset \mathbb{R}^d$,

$$\mu(A) = P(X \in A)$$

y para cualquier $x \in \mathbb{R}^d$,

$$\eta(x) = P(Y = 1|X = x) = E(I_{\{Y=1\}}|X = x) = E(Y|X = x),$$

donde I_A denota a la función indicadora sobre el conjunto A.

Así, $\eta(x)$ es la probabilidad condicional de que $Y = 1$ dado que $X = x$. Para que se logre ver que en efecto, esto es suficiente para describir la distribución de nuestro par aleatorio, observemos que para cualquier $C \subset \mathbb{R}^d \times \{0, 1\}$ se tiene

$$C = (C \cap \mathbb{R}^d \times \{0\}) \cup (C \cap \mathbb{R}^d \times \{1\}) := (C_0 \times \{0\}) \cup (C_1 \times \{1\}),$$

y

$$\begin{aligned} P((X, Y) \in C) &= P(X \in C_0, Y = 0) + P(X \in C_1, Y = 1) \\ &= \int_{C_0} (1 - \eta(x))\mu(dx) + \int_{C_1} \eta(x)\mu(dx). \end{aligned}$$

Como lo anterior es válido para cualquier conjunto Borel medible C, la distribución de nuestro par aleatorio (X, Y) queda determinada por (μ, η). En ocasiones la función η recibe el nombre de *probabilidad a posteriori*.

Cualquier función $g : \mathbb{R}^d \to \{0, 1\}$ genera un *clasificador* o una *función de decisión*. La probabilidad del error de decisión de la función g es definida como $L(g) = P(g(X) \neq Y)$. Una función cuyo error es de particular interés, es la *función de decisión de Bayes*

$$B(x) = \begin{cases} 1 & \text{si } \eta(x) > \frac{1}{2} \\ 0 & \text{en otro caso.} \end{cases}$$

Esta función de decisión minimiza la probabilidad de error.

Teorema 1.5.1 (Minimalidad del error de la función de Bayes, [9])**.** *Para cualquier función* $g : \mathbb{R}^d \longrightarrow \{0, 1\}$,

$$P(B(X) \neq Y) \leq P(g(X) \neq Y).$$

Demostración. Dado $X = x$, la probabilidad del error condicional de cualquier función de decisión g puede ser expresada como

$$\begin{aligned} P(g(X) \neq Y|X = x) &= 1 - P(Y = g(X)|X = x) \\ &= 1 - [P(Y = 1,\ g(X) = 1|X = x) + P(Y = 0,\ g(X) = 0|X = x)] \\ &= 1 - \left[I_{\{g(x)=1\}} P(Y = 1|X = x) + I_{\{g(x)=0\}} P(Y = 0|X = x)\right] \\ &= 1 - \left[I_{\{g(x)=1\}} \eta(x) + I_{\{g(x)=0\}} (1 - \eta(x))\right]. \end{aligned}$$

De este modo, para cada $x \in \mathbb{R}^d$,

$$\begin{aligned} &P(g(X) \neq Y|X = x) - P(B(X) \neq Y|X = x) \\ &= \eta(x) \left[I_{\{B(x)=1\}} - I_{\{g(x)=1\}}\right] + (1 - \eta(x)) \left[I_{\{B(x)=0\}} - I_{\{g(x)=0\}}\right] \\ &= \eta(x) \left[I_{\{B(x)=1\}} - I_{\{g(x)=1\}}\right] + (1 - \eta(x)) \left[(1 - I_{\{B(x)=1\}}) - (1 - I_{\{g(x)=1\}})\right] \\ &= \eta(x) \left[I_{\{B(x)=1\}} - I_{\{g(x)=1\}}\right] + (\eta(x) - 1) \left[I_{\{B(x)=1\}} - I_{\{g(x)=1\}}\right] \\ &= (2\eta(x) - 1) \left[I_{\{B(x)=1\}} - I_{\{g(x)=1\}}\right] \qquad (1.1) \\ &\geq 0, \end{aligned}$$

donde la no negatividad es consecuencia de la definición de B. De este modo, el resultado se sigue integrando ambos lados de (1.1) con respecto a $\mu(dx)$. □

A $L_B = P(B(X) \neq Y)$ se le conoce como la *probabilidad del error de Bayes*, el *error de Bayes* o el *riesgo de Bayes*. De la demostración del teorema 1.5.1 se observa que

$$L(g) = 1 - E\left[I_{\{g(X)=1\}} \eta(X) + I_{\{g(X)=0\}} (1 - \eta(X))\right]$$

y en particular,

$$L_B = 1 - E\left[I_{\{\eta(X) > \frac{1}{2}\}} \eta(X) + I_{\{\eta(X) \leq \frac{1}{2}\}} (1 - \eta(X))\right].$$

Finalmente, observemos que la probabilidad a posteriori minimiza el error cuadrático, es decir, cuando Y es predicha por $f(X)$ para alguna función $f : \mathbb{R}^d \to \mathbb{R}$, se tiene que

$$E([\eta(X) - Y]^2) \leq E([f(X) - Y]^2).$$

En efecto, obsérvese que por la definición de η, para cada $x \in \mathbb{R}^d$,

$$\begin{aligned}
&E([f(X) - Y]^2 | X = x) \\
&= E([f(X) - \eta(X) + \eta(X) - Y]^2 | X = x) \\
&= (f(x) - \eta(x))^2 + 2(f(x) - \eta(x))E(\eta(X) - Y | X = x) \\
&+ E([\eta(X) - Y]^2 | X = x) \\
&= (f(x) - \eta(x))^2 + E([\eta(X) - Y]^2 | X = x).
\end{aligned}$$

De lo cual, el resultado deseado se sigue después de integrar.

A menudo es conveniente considerar otras formas para el error de Bayes:

$$\begin{aligned}
L_B &= \inf_{g:\mathbb{R}^d \to \{0,1\}} P(g(X) \neq Y) \\
&= E(\text{mín}(\eta(X), 1 - \eta(X))) \\
&= \frac{1}{2} - \frac{1}{2} E(|2\eta(X) - 1|).
\end{aligned} \tag{1.2}$$

En efecto, por el teorema 1.5.1 se tiene la igualdad

$$L_B = \inf_{g:\mathbb{R}^d \to \{0,1\}} P(g(X) \neq Y).$$

Por otro lado vimos que

$$L_B = 1 - E\left[I_{\{\eta(X) > \frac{1}{2}\}} \eta(X) + I_{\{\eta(X) \leq \frac{1}{2}\}} (1 - \eta(X)) \right],$$

y de esta igualdad se sigue que

$$\begin{aligned}
L_B =& E\left[1 - \left(1 - I_{\{\eta(X) \leq \frac{1}{2}\}} \right) \eta(X) - I_{\{\eta(X) \leq \frac{1}{2}\}} (1 - \eta(X)) \right] \\
=& E\left[1 - \eta(X) - I_{\{\eta(X) \leq \frac{1}{2}\}} (1 - 2\eta(X)) \right] \\
=& E\left[1 - \eta(X) - ([1 - \eta(X)] - \eta(X))^+ \right],
\end{aligned}$$

donde g^+ denota la parte positiva de g. De este modo las últimas dos igualdades en (1.2) se siguen de observar que $\text{mín}(f, h) = f - (f - h)^+$ y $\text{mín}(f, h) = \frac{f+h}{2} - \frac{|f-h|}{2}$.

En casos especiales, otra expresión útil que puede obtenerse depende de conocer la densidad de X. En efecto, supongamos que X tiene densidad f, entonces

$$\begin{aligned}
L_B &= E(\text{mín}(\eta(X), 1 - \eta(X))) \\
&= \int \text{mín}(\eta(x), 1 - \eta(x)) dP_X \\
&= \int \text{mín}(\eta(x), 1 - \eta(x)) f(x) dx \\
&= \int \text{mín}((1 - p) f_0(x), p f_1(x)) dx,
\end{aligned}$$

donde $p = P(Y = 1)$, $f_i(x)$ es la densidad de X dado que $Y = i$ y la última igualdad se tiene debido a que

$$\begin{aligned}\eta(x)f(x) &= P(X = x)P(Y = 1|X = x)\\ &= P(Y = 1, X = x)\\ &= P(Y = 1)P(X = x|Y = 1)\\ &= pf_1(x),\end{aligned}$$

y de forma similar $f(x)(1 - \eta(x)) = (1 - p)f_0(x)$. Comúnmente p y $1 - p$ son llamadas las *probabilidades de clase* y f_0, f_1 son las *densidades condicionales de la clase*. Si f_0 y f_1 satisfacen $\int f_0 f_1 = 0$, entonces se tiene que $L_B = 0$. Para ejemplificar, supongamos que $p = \frac{1}{2}$. Entonces

$$\begin{aligned}L_B &= \frac{1}{2}\int \text{mín}(f_0(x), f_1(x))dx\\ &= \frac{1}{4}\int [f_1(x) + f_0(x) + |f_1(x) - f_0(x)|]dx\\ &= \frac{1}{2} - \frac{1}{4}\int |f_1(x) - f_0(x)|dx. \qquad (1.3)\end{aligned}$$

De este modo, el error de Bayes está directamente relacionado con la distancia entre clases de densidad en el espacio normado L^1.

A continuación se presenta un ejemplo simple acerca de lo mencionado anteriormente.

Ejemplo 1.5.2. *Consideremos el caso de predicción sobre el desempeño de un estudiante durante cierto curso (aprobar/reprobar) cuando una serie importante de factores se le presentan. Primero definamos $Y = 1$ como aprobar y a $Y = 0$ como reprobar. Aquí X es el número de horas de estudio por semana. Aunque ciertamente no es un predictor infalible sobre el desempeño del estudiante, pues para ello necesitariamos más datos acerca del estudiante, por ejemplo la rapidez mental, salud y hábitos sociales. La función de regresión $\eta(x) = P(Y = 1|X = x)$ es posiblemente monótona creciente en x. Supongamos que la función de regresión es $\eta(x) = \frac{x}{x+c}$ con $c > 0$, podemos entonces resolver nuestro problema. Debido a que la función de decisión de Bayes es*

$$B(x) = \begin{cases} 1 & \text{si } \eta(x) > \frac{1}{2} \text{ (es decir, } x > c)\\ 0 & \text{en otro caso,}\end{cases}$$

el correspondiente error de Bayes es

$$\begin{aligned}L_B = L(B) &= E(\text{mín}(\eta(X), 1 - \eta(X)))\\ &= E\left(\text{mín}\left(\frac{X}{X + c}, 1 - \frac{X}{X + c}\right)\right)\\ &= E\left(\frac{\text{mín}(c, X)}{c + X}\right).\end{aligned}$$

Aunque podemos deducir la función de decisión de Bayes solamente de η, para el error de Bayes, L_B, no se puede decir lo mismo, pues desconocemos en principio de cuentas

la distribución de X. Si por un lado, $X = c$ con probabilidad 1 (como es el caso de una escuela militar donde todos los estudiantes son obligados a estudiar c horas por semana), entonces $L_B = \frac{1}{2}$. Si consideramos que tenemos una población bien distribuida, supongamos por ejemplo X uniforme sobre $[0, 4c]$, entonces la situación mejora:

$$L_B = \frac{1}{4c}\int_0^{4c} \frac{\text{mín}(c, x)}{c + x} dx = \frac{1}{4}\log\left(\frac{5e}{4}\right) \approx 0.305785.$$

1.6. Decisiones Plug-in

La mejor predicción de Y dada la observación X es proporcionada por la regla de decisión de Bayes

$$B(x) = \begin{cases} 1 & \text{si } \eta(x) > \frac{1}{2} \\ 0 & \text{en otro caso} \end{cases} = \begin{cases} 1 & \text{si } \eta(x) > 1 - \eta(x) \\ 0 & \text{en otro caso.} \end{cases}$$

Aunque con frecuencia η es desconocida, si se tiene acceso a funciones no negativas $\tilde{\eta}(x)$, $1 - \tilde{\eta}(x)$ que aproximan $\eta(x)$ y $1 - \eta(x)$, respectivamente, resulta natural usar la función de decisión plug-in

$$g(x) = \begin{cases} 1 & \text{si } \tilde{\eta}(x) > \frac{1}{2} \\ 0 & \text{en otro caso} \end{cases}$$

para aproximar la función de decisión de Bayes.

El siguiente teorema es bien conocido y establece que si $\tilde{\eta}(x)$ es cercana a la probabilidad a posteriori real (en el sentido de la distancia en el espacio L^1), entonces la probabilidad del error de decisión de g es cercano al error de Bayes.

Teorema 1.6.1. *Para la probabilidad del error de la función de decisión plug-in g definida anteriormente, se tiene que*

$$P(g(X) \neq Y) - L_B = 2\int_{\mathbb{R}^d} \left|\eta(x) - \frac{1}{2}\right| I_{\{g(x) \neq B(x)\}} \mu(dx) \tag{1.4}$$

y

$$P(g(X) \neq Y) - L_B \leq 2\int_{\mathbb{R}^d} |\eta(x) - \tilde{\eta}(x)| \mu(dx) = 2E(|\eta(X) - \tilde{\eta}(X)|).$$

Demostración. Si para algún $x \in \mathbb{R}^d$, $g(x) = B(x)$, entonces claramente la diferencia entre las probabilidades condicionales del error de g y B es cero:

$$P(g(X) \neq Y | X = x) - P(B(X) \neq Y | X = x) = 0.$$

Por otro lado, si $g(x) \neq B(x)$, entonces la ecuación (1.1) del teorema 1.5.1 se puede reescribir como

$$P(g(X) \neq Y | X = x) - P(B(X) \neq Y | X = x) = |2\eta(x) - 1| I_{\{g(x) \neq B(x)\}}.$$

De este modo, dado que $g(x) \neq B(x)$ implica que $\left|\eta(x) - \frac{1}{2}\right| \leq |\eta(x) - \tilde{\eta}(x)|$, entonces

$$\begin{aligned} P(g(X) \neq Y) - L_B &= 2 \int_{\mathbb{R}^d} \left|\eta(x) - \frac{1}{2}\right| I_{\{g(x) \neq B(x)\}} \mu(dx) \\ &\leq 2 \int_{\mathbb{R}^d} |\eta(x) - \tilde{\eta}(x)| \mu(dx). \end{aligned}$$

□

Observación 1.6.2. *Por la demostración del teorema anterior, se tiene que la igualdad* (1.4) *se cumple para cualquier regla de decisión g.*

Cuando el clasificador $g(x)$ se puede expresar en la forma

$$g(x) = \begin{cases} 0 & \text{si } \tilde{\eta}_1(x) \leq \tilde{\eta}_0(x) \\ 1 & \text{en otro caso,} \end{cases}$$

donde $\tilde{\eta}_1(x)$, $\tilde{\eta}_0(x)$ son aproximaciones de $\eta(x)$ y $1 - \eta(x)$, respectivamente, entonces si $\tilde{\eta}_1(x) + \tilde{\eta}_0(x)$ no es necesariamente igual a 1, la situación mencionada en el teorema anterior cambia. Sin embargo, una desigualdad análoga a la que se menciona resulta cierta.

Teorema 1.6.3. *Dada la función g definida anteriormente, la probabilidad de error de decisión y el error de Bayes satisfacen la siguiente desigualdad*

$$P(g(X) \neq Y) - L_B \leq \int_{\mathbb{R}^d} |[1 - \eta(x)] - \tilde{\eta}_0(x)| \mu(dx) + \int_{\mathbb{R}^d} |\eta(x) - \tilde{\eta}_1(x)| \mu(dx).$$

Demostración. Dado que la igualdad (1.4) del teorema 1.6.1 se cumple para cualquier función de decisión, es suficiente demostrar que para cualquier x tal que $g(x) \neq B(x)$ se satisface

$$|2\eta(x) - 1| \leq |1 - \eta(x) - \tilde{\eta}_0(x)| + |\eta(x) - \tilde{\eta}_1(x)|.$$

En efecto, supongamos que $B(x) = 1$ y $g(x) = 0$, entonces

$$\eta(x) > \frac{1}{2} \text{ y } \tilde{\eta}_0(x) \geq \tilde{\eta}_1(x),$$

y así $|2\eta(x) - 1| = 2\eta(x) - 1$ y $\tilde{\eta}_0(x) - \tilde{\eta}_1(x) \geq 0$. Por lo tanto al sumar ambas ecuaciones obtenemos que

$$\begin{aligned} 2\eta(x) - 1 &\leq 2\eta(x) - 1 + \tilde{\eta}_0(x) - \tilde{\eta}_1(x) \\ &\leq |1 - \eta(x) - \tilde{\eta}_0(x)| + |\eta(x) - \tilde{\eta}_1(x)|. \end{aligned}$$

La demostración para $B(x) = 0$ y $g(x) = 1$ es análoga. □

1.7. Discriminación univariada y divisiones de Stoller

Para comenzar esta sección, debemos explicar la idea detrás de lo que buscamos: dividiremos el espacio por un hiperplano asignándole una clase diferente a cada semiespacio. Esta regla de decisión resulta fácil de interpretar, lo anterior debido a que cada decisión está basada en el signo de $W^T x + b$, donde $x = [x_1, x_2, ..., x_d]^T$, $W = [w_1, w_2, ..., w_d]^T$ son vectores en $\mathbb{R}^d$ y b es un escalar, los w_i's son *pesos* y b es un *sesgo*. De este modo, el vector W determina la importancia relativa de los componentes, haciendo que la decisión sea fácilmente implementada.

A manera de ejemplo, consideremos X univariada. La regla de decisión más simple es la regla de discriminación lineal

$$g(x) = \begin{cases} y' & \text{si } x \leq x' \\ 1 - y' & \text{en otro caso,} \end{cases}$$

donde x' es un punto de división y $y' \in \{0, 1\}$ es una clase. En general x' y y' son funciones medibles del conjunto de datos $D_n = \{(X_1, Y_1), \dots (X_n, Y_n)\}$. Dentro de esta clase de reglas simples, existe una mejor regla posible que puede ser determinada si conocemos la distribución. Asumamos por ejemplo que (X, Y) es descrito en la forma estándar: sea $P(Y = 1) = p$; dado $Y = 1$, X tiene una función de distribución $F_1(x) = P(X \leq x | Y = 1)$, y dado $Y = 0$, X tiene una función de distribución $F_0(x) = P(X \leq x | Y = 0)$, donde F_0 y F_1 son las *clases condicionales de las funciones de distribución*. Entonces teóricamente, una regla óptima queda determinada por el punto de división x^* y la clase y^* tales que

$$(x^*, y^*) = \underset{(x', y')}{\operatorname{argmin}} P(g(X) \neq Y)$$

(el mínimo es siempre alcanzable si consideramos que $x' \in \overline{\mathbb{R}}$). Sea L la correspondiente probabilidad de error mínima y note que

$$\begin{aligned} L = \inf_{(x', y')} \big\{ & I_{\{y'=0\}} [p F_1(x') + (1 - p)(1 - F_0(x'))] \\ & + I_{\{y'=1\}} [p(1 - F_1(x')) + (1 - p) F_0(x')] \big\} . \end{aligned}$$

La justificación de esto es como sigue: sea $g(x)$ como se describió anteriormente, entonces $g(x)$ corresponde a alguna de las dos funciones siguientes, dependiendo de si $y' = 0$ o $y' = 1$,

$$g_0(x) = \begin{cases} 0 & \text{si } x \leq x' \\ 1 & \text{en otro caso,} \end{cases} \quad \text{o} \quad g_1(x) = \begin{cases} 1 & \text{si } x \leq x' \\ 0 & \text{en otro caso.} \end{cases}$$

Podemos calcular la probabilidad del error para $g_0(x)$. En efecto,

$$\begin{aligned} P(g_0(X) \neq Y) &= P(g_0(X) = 1, Y = 0) + P(g_0(X) = 0, Y = 1) \\ &= P(Y = 0)P(g_0(X) = 1|Y = 0) + P(Y = 1)P(g_0(X) = 0|Y = 1) \\ &= P(Y = 0)P(X > x'|Y = 0) + P(Y = 1)P(X \leq x'|Y = 1) \\ &= (1 - p)(1 - F_0(x')) + pF_1(x'). \end{aligned}$$

De modo similar podemos mostrar que $P(g_1(X) \neq Y) = p(1 - F_1(x')) + (1 - p)F_0(x')$. Entonces

$$L(g) = I_{\{y'=0\}}P(g_0(X) \neq Y) + I_{\{y'=1\}}P(g_1(X) \neq Y),$$

para cada $g(x)$. Obsérvese que cada $g(x)$ queda determinada por un único par (x', y'). De este modo, si buscamos la correspondiente probabilidad de error mínima de este tipo de clasificadores,

$$\begin{aligned} L &= \inf_g L(g) \\ &= \inf_{(x',y')} \left\{ I_{\{y'=0\}}[pF_1(x') + (1 - p)(1 - F_0(x'))] \right. \\ &\quad \left. + I_{\{y'=1\}}[p(1 - F_1(x')) + (1 - p)F_0(x')] \right\}, \end{aligned}$$

que es como habíamos dicho.

Un divisor definido por (x^*, y^*) es llamado un *divisor teórico de Stoller*.

Lema 1.7.1. *$L \leq \frac{1}{2}$ con la igualdad si y sólo si $L_B = \frac{1}{2}$.*

Demostración. Tomemos $(x', y') = (-\infty, 0)$. Entonces la probabilidad del error es $1 - p = P(Y = 0)$. Si ahora tomamos $(x', y') = (-\infty, 1)$. Entonces la probabilidad de error es p. Claramente

$$L_B \leq L \leq \text{mín}(p, 1 - p) \leq \frac{1}{2}.$$

Esto prueba la primera parte del lema. Para la segunda parte, si $L = \frac{1}{2}$, entonces $p = \frac{1}{2}$, y para cualquier x, debido a que

$$\begin{aligned} L &= \inf_g L(g) \\ &= \inf_{(x',y')} \left\{ I_{\{y'=0\}}[pF_1(x') + (1 - p)(1 - F_0(x'))] \right. \\ &\quad \left. + I_{\{y'=1\}}[p(1 - F_1(x')) + (1 - p)F_0(x')] \right\}, \end{aligned}$$

se sigue que $pF_1(x) + (1 - p)(1 - F_0(x)) \geq \frac{1}{2}$ y $p(1 - F_1(x)) + (1 - p)F_0(x) \geq \frac{1}{2}$. Usando que $p = \frac{1}{2}$, la primera desigualdad implica $\frac{1}{2}F_1(x) - \frac{1}{2}F_0(x) \geq 0$, mientras que la segunda implica $\frac{1}{2}F_1(x) - \frac{1}{2}F_0(x) \leq 0$. Así que $L = \frac{1}{2}$ significa que para cada x, $F_1(x) = F_0(x)$ y por lo tanto $L_B = \frac{1}{2}$. □

Lema 1.7.2.

$$L = \frac{1}{2} - \sup_x \left| pF_1(x) - (1 - p)F_0(x) - p + \frac{1}{2} \right|.$$

En particular, si $p = \frac{1}{2}$, entonces

$$L = \frac{1}{2} - \frac{1}{2} \sup_x |F_1(x) - F_0(x)|.$$

Demostración. Sea $\rho(x) = pF_1(x) - (1-p)F_0(x)$. Como por definición L se obtiene a partir del ínfimo sobre las (x', y'), entonces es equivalente tomar el ínfimo sobre los $(x', 0)$ y $(x', 1)$, es decir, para cada x' tomar el mínimo sobre las y' y después el ínfimo sobre las x', esto es

$$L = \inf_x \min\{\rho(x) + 1 - p, p - \rho(x)\}.$$

Recordando que $\min\{a, b\} = \frac{1}{2}(a + b - |a - b|)$ y $-\sup(A) = \inf(-A)$, se sigue que

$$\begin{aligned} L &= \inf_x \left(\frac{1}{2} - \left| \rho(x) - p + \frac{1}{2} \right| \right) \\ &= \frac{1}{2} - \sup_x \left| \rho(x) - p + \frac{1}{2} \right|. \end{aligned}$$

□

Este último resultado nos muestra la calidad del divisor teórico de Stoller con la distancia de Kolmogorov-Smirnov ($\sup_x |F_1(x) - F_0(x)|$) entre la clase de funciones de distribución condicionales. Para ver también algunas limitaciones acerca de los divisores teóricos de Stoller, consideremos dos clases con medias $E[X|Y = 0] = m_0$, $E[X|Y = 1] = m_1$, y varianzas $Var[X|Y = 0] = \sigma_0^2$ y $Var[X|Y = 1] = \sigma_1^2$, entonces se cumple la siguiente desigualdad.

Teorema 1.7.3.

$$L_B \leq L \leq \frac{1}{1 + \frac{(m_0 - m_1)^2}{(\sigma_0 * \sigma_1)^2}}.$$

Demostración. Sin pérdida de generalidad podemos asumir que $m_0 \leq m_1$. Claramente, L es más pequeño que la probabilidad del error para la regla que decide 0 cuando $x \leq m_0 + \Delta_0$, y 1 en otro caso, donde $m_1 - m_0 = \Delta_0 + \Delta_1$ y $\Delta_0,\ \Delta_1 > 0$.

La probabilidad de error de la última regla es

$$\begin{aligned} & pF_1(m_0 + \Delta_0) + (1-p)(1 - F_0(m_0 + \Delta_0)) \\ =& pP(X \geq m_1 - \Delta_1 | Y = 1) + (1-p)P(X > m_0 + \Delta_0 | Y = 0) \\ \leq& p\frac{\sigma_1^2}{\sigma_1^2 + \Delta_1^2} + (1-p)\frac{\sigma_0^2}{\sigma_0^2 + \Delta_0^2} \qquad (1.5) \\ =& \frac{p}{1 + \frac{\Delta_1^2}{\sigma_1^2}} + \frac{1-p}{1 + \frac{\Delta_0^2}{\sigma_0^2}}, \end{aligned}$$

donde la desigualdad en (1.5) se sigue de la desigualdad de Chebyshev-Cantelli (teorema A.17 del apéndice de [9]). Así, tomando $\Delta_0 = |m_1 - m_0|\frac{\sigma_0}{\sigma_0 + \sigma_1}$ y $\Delta_1 = \frac{\sigma_1}{\sigma_0}\Delta_0$ se tiene lo deseado. □

Veremos ahora lo que se hace cuando un divisor debe estar basado en una muestra de datos. Stoller sugirió tomar (x', y') de manera que el error empírico sea mínimo. Esto es

$$(x', y') = \operatorname*{argmin}_{(x,y) \in \mathbb{R} \times \{0,1\}} \frac{1}{n} \sum_{i=1}^{n} \left(I_{\{X_i \leq x,\, Y_i \neq y\}} + I_{\{X_i > x,\, Y_i \neq 1-y\}} \right)$$

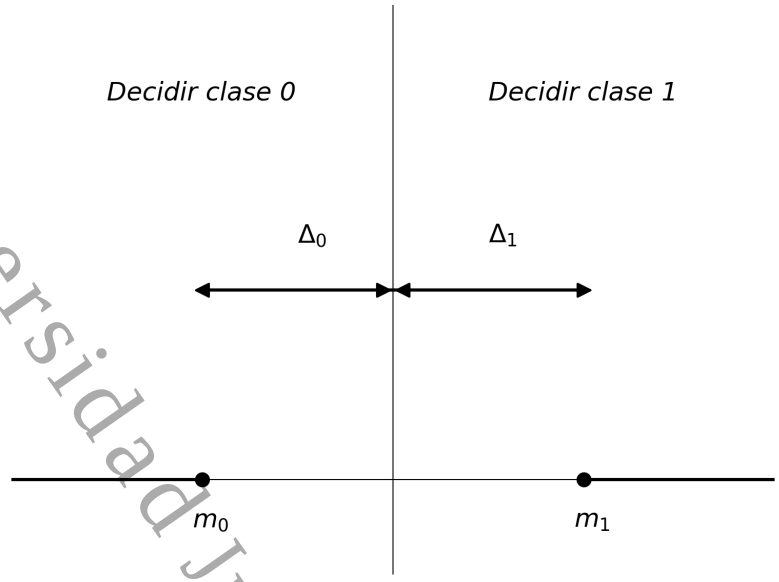


Figura 1.4: Representación gráfica de la división proporcionada por el límite mencionado en el teorema 1.7.3 [9].

(x' y y' son variables aleatorias, pero pese a lo usual, convenimos en dejar la notación en minúsculas por ahora). Llamaremos a esto la *regla de Stoller* y nos referiremos al divisor como un *divisor empírico de Stoller*. Definiendo

$$C(x,y) = \{(-\infty, x] \times \{1-y\}\} \cup \{(x,\infty) \times \{y\}\}.$$

Entonces

$$(x', y') = \underset{(x,y)}{\operatorname{argmin}}\, v_n(C(x,y)),$$

donde v_n es la *medida empírica* para el conjunto de datos $D_n = \{(X_1, Y_1), ..., (X_n, Y_n)\}$, que se define para cada conjunto $A \subset \mathbb{R} \times \{0,1\}$ medible, como

$$v_n(A) = \frac{1}{n} \sum_{i=1}^{n} I_{\{(X_i,Y_i)\in A\}}.$$

No es difícil ver que v_n es una medida. Analicemos la veracidad de la obtención del divisor empírico de Stoller a partir de la medida v_n y los conjuntos $C(x,y)$. En efecto, debido a que $C(x,y)$ es una unión disjunta de dos conjuntos, la indicadora sobre la unión se puede ver como la suma de las indicadoras sobre cada conjunto. Así, para cada $i = 1, 2, ..., n$

$$I_{\{(X_i,Y_i)\in C(x,y)\}} = I_{\{(X_i,Y_i)\in A_1\}} + I_{\{(X_i,Y_i)\in A_2\}},$$

donde $A_1 = \{(-\infty, x] \times \{1-y\}\}$ y $A_2 = \{(x,\infty) \times \{y\}\}$. Considerando la definición

de v_n se obtiene que

$$\begin{aligned} v_n(C(x,y)) &= \frac{1}{n}\sum_{i=1}^{n} I_{\{(X_i,Y_i)\in C(x,y)\}} \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(I_{\{(X_i,Y_i)\in A_1\}} + I_{\{(X_i,Y_i)\in A_2\}}\right) \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(I_{\{X_i\leq x,\, Y_i=1-y\}} + I_{\{X_i>x,\, Y_i=y\}}\right) \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(I_{\{X_i\leq x,\, Y_i\neq y\}} + I_{\{X_i>x,\, Y_i\neq 1-y\}}\right). \end{aligned}$$

De lo cual, la veracidad de la obtención del divisor empírico de Stoller a partir de la medida v_n y los conjuntos $C(x,y)$ es clara.

Denotando a la medida de $(X,Y)\in\mathbb{R}\times\{0,1\}$ por v, se puede ver que

$$E(v_n(C(x,y))) = v(C(x,y)) = P(X\leq x, Y\neq y) + P(X>x, Y\neq 1-y).$$

En efecto, dado que $v_n(C(x,y))$ es una suma de indicadoras y (X_i,Y_i), $i=1,\ldots,n$ es una muestra aleatoria de (X,Y), entonces

$$\begin{aligned} E(v_n(C(x,y))) &= \frac{1}{n}\sum_{i=1}^{n}\left[E(I_{\{X_i\leq x,\, Y_i\neq y\}}) + E(I_{\{X_i>x,\, Y_i\neq 1-y\}})\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[P(X_i\leq x,\ Y_i\neq y) + P(X_i>x,\ Y_i\neq 1-y)\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[P(X\leq x,\ Y\neq y) + P(X>x,\ Y\neq 1-y)\right] \\ &= P(X\leq x,\ Y\neq y) + P(X>x,\ Y\neq 1-y). \end{aligned}$$

Sea $L_n = P(g_n(X)\neq Y|D_n)$ la probabilidad del error de la regla de clasificación g_n con la elección (x',y') dependiente de los datos dada anteriormente, condicionada a los datos. Entonces

$$\begin{aligned} L_n &= v(C(x',y')) \\ &= v(C(x',y')) - v_n(C(x',y')) + v_n(C(x',y')) \\ &\leq \sup_{(x,y)}\left[v(C(x,y)) - v_n(C(x,y))\right] + v_n(C(x^*,y^*)) \\ &\leq 2\sup_{(x,y)}\left|v(C(x,y)) - v_n(C(x,y))\right| + v(C(x^*,y^*)) \\ &= 2\sup_{(x,y)}\left|v(C(x,y)) - v_n(C(x,y))\right| + L, \end{aligned} \tag{1.6}$$

donde (x^*,y^*) minimiza a $v(C(x,y))$ sobre todo (x,y).

Del siguiente teorema se observa que el supremo anterior es pequeño aún para n moderadamente grande y, por lo tanto, la regla de Stoller se acerca a la mejor división independientemente de la distribución de (X,Y).

Teorema 1.7.4. *Para la regla de Stoller, y* $\epsilon > 0$ *se tiene*

$$P(L_n - L \geq \epsilon) \leq 4e^{-n\epsilon^2/2}$$

y

$$E(L_n - L) \leq \sqrt{\frac{2\log(4e)}{n}}.$$

Demostración. Por la desigualdad (1.6) se tiene

$$\begin{aligned} P(L_n - L \geq \epsilon) &\leq P\left(\sup_{(x,y)} [v(C(x,y)) - v_n(C(x,y))] \geq \frac{\epsilon}{2}\right) \\ &\leq P\left(\sup_{x} [v(C(x,0)) - v_n(C(x,0))] \geq \frac{\epsilon}{2}\right) \\ &\quad + P\left(\sup_{x} [v(C(x,1)) - v_n(C(x,1))] \geq \frac{\epsilon}{2}\right) \\ &\leq 4e^{-2n(\epsilon/2)^2}. \end{aligned}$$

□

Nótese que la probabilidad de error de la regla de Stoller es uniformemente cercana a L sobre todas las posibles distribuciones.

Dada una secuencia de datos de entrenamiento $D_n = \{(X_1, Y_1), ..., (X_n, Y_n)\}$, la mejor expectativa sobre una función de clasificación es alcanzar la probabilidad del error de Bayes L_B. Por lo general, no se espera obtener una función que alcance exactamente el error de Bayes, pero sí es posible construir una sucesión de clasificadores $\{g_n\}$, esto es, una regla de clasificación tal que la probabilidad de error

$$L_n = L(g_n) = P(g_n(X, D_n) \neq Y | D_n)$$

se aproxime arbitrariamente a L_B con una alta probabilidad (es decir, para "muchos" D_n). La idea anterior se puede expresar formalmente en la definición de consistencia.

Definición 1.7.5 (Consistencia débil y fuerte)**.** *Una regla de clasificación es consistente para una distribución determinada de* (X, Y) *si*

$$EL_n = P(g_n(X, D_n) \neq Y) \to L_B, \quad \textit{cuando } n \to \infty,$$

y es fuertemente consistente si

$$\lim_{n\to\infty} L_n = L_B \quad \textit{casi seguramente.}$$

Observación 1.7.6. *La consistencia es definida como la convergencia del valor esperado de* L_n *a* L_B*. Dado que* L_n *es una variable aleatoria acotada por* L_B *y 1, esta convergencia es equivalente a la convergencia en probabilidad. En consecuencia, dado que la convergencia casi segura siempre implica convergencia en probabilidad, la consistencia fuerte implica consistencia.*

Una regla consistente garantiza que al incrementarse la cantidad de datos, la probabilidad de que la probabilidad de error se encuentre en un rango muy pequeño del valor óptimo alcanzable se acerque a 1. Intuitivamente, la regla puede eventualmente aprender la decisión óptima de una gran cantidad de datos de entrenamiento con probabilidad alta. La consistencia fuerte, por otro lado, significa que por usar mas datos, la probabilidad de error se acerca al óptimo para cada secuencia de datos de entrenamiento, excepto quizás para una secuencia de datos que tienen probabilidad 0.

Una regla de decisión puede ser consistente solo para una cierta clase de distribuciones de (X, Y). Por lo tanto es deseable tener una regla que sea consistente para una gran clase de distribuciones. Dado que en muchas situaciones no tenemos información previa sobre la distribución, es fundamental tener una regla con buen comportamiento para todas las distribuciones. Lo anterior motiva la definición de consistencia universal que se formula como sigue.

Definición 1.7.7 (Consistencia universal)**.** *Una secuencia de reglas de decisión es llamada universalmente (fuertemente) consistente si es (fuertemente) consistente para cualquier distribución del par* (X, Y).

1.8. Discriminación lineal

En esta sección comenzaremos a tratar el perceptrón de Rosenblatt. Para ello hay que generalizar lo tratado en la sección anterior. El perceptrón está basado en una división de $\mathbb{R}^d$ en 2 partes por un hiperplano. La regla de discriminación con vector de pesos $W = [w_1, w_2, ..., w_d]^T$ y sesgo b está dada por

$$g(x) = \begin{cases} 1 & \text{si } W^T x + b > 0 \\ 0 & \text{en otro caso,} \end{cases}$$

donde $x = [x_1, x_2, ..., x_d]^T$.

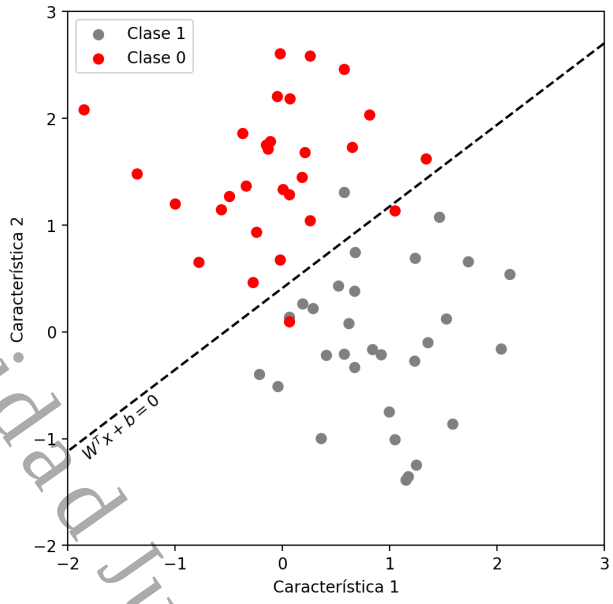


Figura 1.5: Ejemplo de discriminación lineal.

Su probabilidad de error será denotada por $L(W, b)$. Consideremos nuevamente a

$$L = \inf_{W \in \mathbb{R}^d,\, b \in \mathbb{R}} L(W, b)$$

como la mejor probabilidad de error posible dentro de esta clase. Denotamos a las funciones de distribución de $W^T X$ condicionadas a las clases, por $F_{0,W}$ y $F_{1,W}$, dependiendo de si $Y = 0$ o $Y = 1$ y donde $X = [X_1, X_2, \ldots, X_d]^T$ es un vector aleatorio. Para $L(W, b)$, aplicando el lema 1.7.2 a $F_{0,W}$ y $F_{1,W}$ se obtiene

$$L = \frac{1}{2} - \sup_W \sup_x \left| pF_{1,W}(x) - (1-p)F_{0,W}(x) - p + \frac{1}{2} \right|.$$

Cuando $p = \frac{1}{2}$, la ecuación se reduce a

$$L = \frac{1}{2} - \frac{1}{2} \sup_W \sup_x |F_{1,W}(x) - F_{0,W}(x)|.$$

Por lo tanto, $L = \frac{1}{2}$ si y sólo si $p = \frac{1}{2}$ y para todo W, $F_{1,W} \equiv F_{0,W}$.

Lema 1.8.1 (Cramér & Wold [9])**.** *Sean X y Y variables aleatorias con valores en $\mathbb{R}^d$. Las variables son idénticamente distribuidas si y sólo si $W^T X$ y $W^T Y$ tienen la misma distribución para cada vector $W \in \mathbb{R}^d$.*

Demostración. Sabemos que dos variables aleatorias tienen la misma distribución si y sólo si tienen la misma función característica. Así, la función característica de

$X = [X_1, X_2, ..., X_d]^T$ es

$$\begin{aligned}\psi_X(W) &= \psi_X(w_1, w_2, ...w_d)\\ &= E\left[e^{i(w_1X_1+...+w_dX_d)}\right]\\ &= E\left[e^{i(w_1Y_1+...+w_dY_d)}\right] \quad \text{(por hipótesis)}\\ &= \psi_Y(w_1, w_2, ..., w_d)\\ &= \psi_Y(W).\end{aligned}$$

□

De este modo, al igual que en la versión univariada, se tiene el siguiente resultado.

Teorema 1.8.2. *$L \leq \frac{1}{2}$ con la igualdad si y sólo si $L_B = \frac{1}{2}$.*

Así, como en la versión univariada, cuando $L_B < \frac{1}{2}$, una regla de clasificación con $L < \frac{1}{2}$, basada en la separación del espacio con un hiperplano, puede ser posible. Como se menciona en [9], existen también ejemplos para los cuales ninguna separación por hiperplanos mejora a una regla en la que $g(x) \equiv y$ para algún y y todo x, aunque $L_B = 0$.

Una forma general del teorema 1.7.3 se muestra a continuación.

Teorema 1.8.3. *Sean X_0 y X_1 variables aleatorias con distribuciones iguales a X dado $Y = 0$ y a X dado $Y = 1$, respectivamente. Sean $m_0 = E(X_0)$ y $m_1 = E(X_1)$. Defínanse también las matrices de covarianza $\Sigma_1 = E\left[(X_1 - m_1)(X_1 - m_1)^T\right]$ y $\Sigma_0 = E\left[(X_0 - m_0)(X_0 - m_0)^T\right]$. Entonces*

$$L_B \leq L \leq \inf_{W\in\mathbb{R}^d} \frac{1}{1 + \frac{(W^T(m_0-m_1))^2}{\left((W^T\Sigma_0W)^{\frac{1}{2}}+(W^T\Sigma_1W)^{\frac{1}{2}}\right)^2}}.$$

Demostración. Dado cualquier $W \in \mathbb{R}^d$ aplíquese el teorema 1.7.3 a W^TX_0 y W^TX_1. En efecto, la prueba se sigue al notar que

$$\begin{aligned}E(W^TX_0) &= W^TE(X_0) = W^Tm_0,\\ E(W^TX_1) &= W^Tm_1,\end{aligned}$$

y que

$$\begin{aligned}Var(W^TX_0) &= E\left[W^T(X_0 - m_0)(X_0 - m_0)^TW\right] = W^T\Sigma_0W,\\ Var(W^TX_1) &= W^T\Sigma_1W.\end{aligned}$$

□

Capítulo 2: Redes neuronales

En el capítulo anterior se analizó la teoría de clasificación binaria mediante discriminantes lineales. En este capítulo entramos de lleno a la teoría de redes neuronales, empezando por la construcción y definición de los perceptrones multicapa para luego llegar a los teoremas de convergencia, también expondremos los procesos que deben llevarse acabo tanto para el entrenamiento como para la validación del modelo construido. Para llevar a cabo este proceso requerimos del teorema de aproximación universal, tema importantes tratado en [9]. Para la sección de redes multiclase nuestra principal referencia es [10]. Por otro lado, en la sección del entrenamiento de la red, consideraremos la idea de la función de costo, elemento que juega un papel importante en la construcción de las redes y que nos permite buscar (al menos) numéricamente los parámetros que conforman nuestra red. Para esta sección nos basamos en [5]. Para las secciones siguientes, acerca del gradiente descendente, el algoritmo backpropagation, las funciones de costo y activación, utilizamos principalmente las referencias [24], [5] y [13]. Finalmente para las últimas secciones sobre los métodos de validación, tratamiento de datos y ejemplos computacionales usamos [24] y [5].

2.1. Perceptrones multicapa y el teorema de aproximación universal

Para el desarrollo de esta sección, nos apoyamos principalmente de [9].

En general, el *discriminante lineal*, también llamado *perceptrón*, es una función de decisión de la forma

$$\phi(x) = \begin{cases} 0 & \text{si } \psi(x) \leq \frac{1}{2} \\ 1 & \text{en otro caso,} \end{cases} \tag{2.1}$$

basado en una combinación lineal $\psi(x)$ de las entradas,

$$\psi(x) = b + \sum_{i=1}^{d} w_i x_i = b + W^T x, \tag{2.2}$$

donde los w_i's son pesos, b es un sesgo, $x = [x_1, x_2, ..., x_d]^T$ y $W = [w_1, w_2, ..., w_d]^T$. Al discriminante lineal (2.1)-(2.2) se le llama *red neuronal sin capas ocultas*.

En una *red neuronal (hacia adelante) con una capa oculta* se tiene

$$\psi(x) = b_0 + \sum_{i=1}^{k} w_i \sigma(\psi_i(x)), \tag{2.3}$$

donde los w_i's son como antes y cada ψ_i es de la forma dada en (2.2), es decir,

$$\psi_i(x) = b_i + \sum_{j=1}^{d} a_{ij} x_j,$$

para algunas constantes b_i y a_{ij}. La función σ es llamada *sigmoide*, la cual es una función no decreciente con $\sigma(x) \to -1$ cuando $x \downarrow -\infty$ y $\sigma(x) \to 1$ cuando $x \uparrow \infty$.

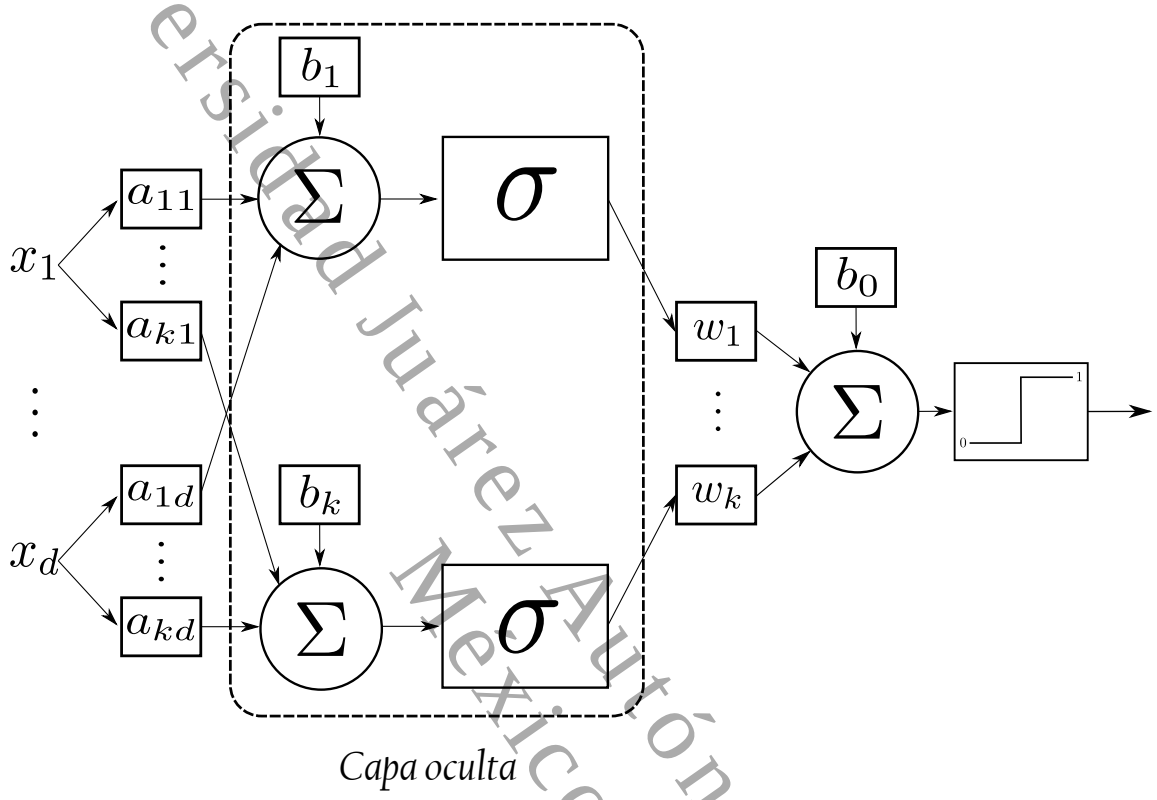


Figura 2.1: Red neuronal con una capa oculta.

En el perceptrón con una capa oculta, decimos que hay k neuronas ocultas. La salida de la i-ésima neurona oculta es $u_i = \sigma(\psi_i(x))$. De este modo (2.3) se reescribe como

$$\psi(x) = b_0 + \sum_{i=1}^{k} w_i u_i,$$

que conserva la forma de (2.2), ver la figura 2.1 para la representación gráfica. Si continuamos con este proceso podemos crear una *red neuronal multicapa hacia adelante*. Por ejemplo, *un perceptrón con dos capas ocultas* usa

$$\psi(x) = b_0 + \sum_{i=1}^{l} w_i z_i,$$

donde

$$z_i = \sigma\left(d_{i0} + \sum_{j=1}^{k} d_{ij} u_j\right), \quad 1 \leq i \leq l,$$

y

$$u_j = \sigma\left(b_j + \sum_{i=1}^{d} a_{ji}x_i\right), \quad 1 \leq j \leq k,$$

donde las d_{ij}'s, b_j's y a_{ji}'s son constantes. La primera capa oculta tiene k neuronas ocultas, mientras que la segunda capa oculta tiene l neuronas ocultas, ver la figura 2.2 para la representación gráfica.

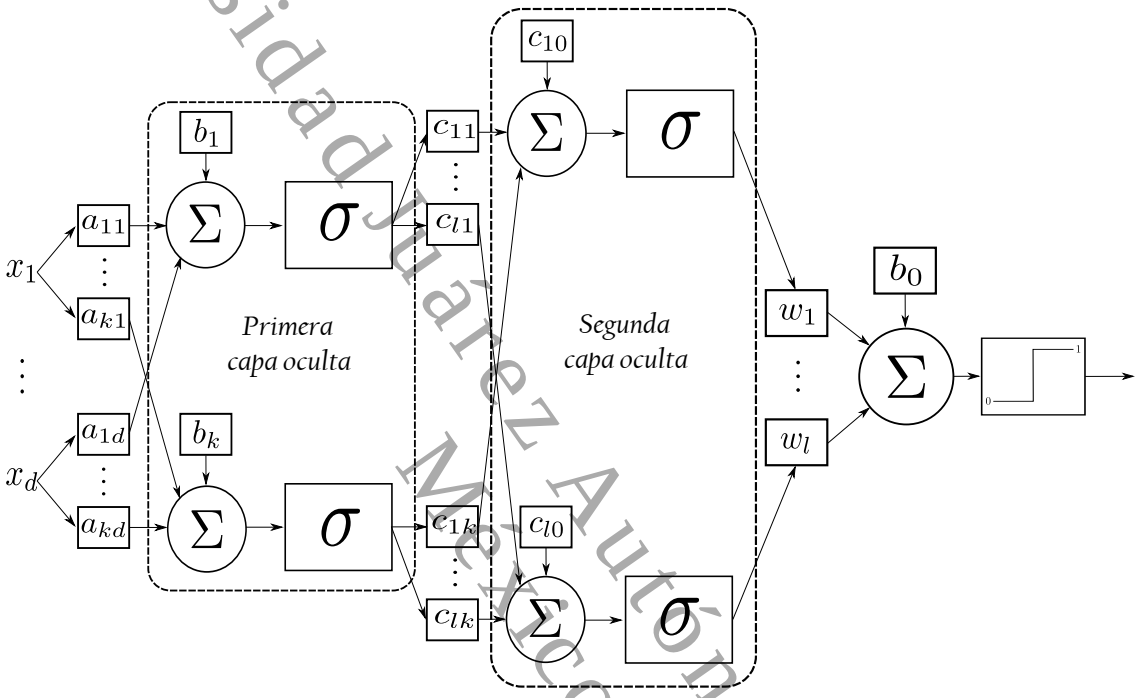


Figura 2.2: Red neuronal con 2 capas ocultas.

El paso del perceptrón a una red neuronal con una capa oculta no es trivial. Es conocido que los discriminantes lineales no pueden conducir a la consistencia universal. Por otra parte, las redes neuronales con una capa oculta si producen discriminantes universalmente consistentes, siempre y cuando permitamos que k, el número de neuronas ocultas, crezca ilimitadamente con n, el número de datos disponibles, lo cual hace de la supuesta consistencia universal algo discutible. Lo último se debe a que el interés por las redes neuronales artificiales radica en la posibilidad de que sean implementadas computacionalmente, como el hardware debe ser fijado de antemano, no podemos dejar que k se convierta en una función de n.

Consideremos primero la clase $C^{(k)}$ de clasificadores del tipo (2.1) que contiene todos los clasificadores de redes neuronales con la función sigmoide

$$\sigma(x) = \text{sign}(x) = \begin{cases} -1 & \text{si } x \leq 0 \\ 1 & \text{si } x > 0, \end{cases} \tag{2.4}$$

y con k neuronas ocultas en dos capas ocultas. El conjunto de entrenamiento D_n es usado para seleccionar un clasificador de $C^{(k)}$. Para un buen funcionamiento de la regla

de selección, es necesario que la mejor regla en $C^{(k)}$ tenga probabilidad de error cercana a L_B, es decir, que

$$\inf_{\phi \in C^{(k)}} L(\phi) - L_B$$

sea pequeño. Llamamos a esta cantidad el *error de aproximación*. Naturalmente, para un número k de neuronas fijo, el error de aproximación es positivo para la mayoría de distribuciones. Sin embargo, conforme k va creciendo, lo esperado es que este valor sea pequeño. De aquí, la cuestión es si esto último es cierto para cada distribución de (X, Y). Se puede mostrar (véase [9]) que la clase de los clasificadores de redes neuronales de 2 capas ocultas con m neuronas en la primera capa y 2^m en la segunda capa contiene todos los clasificadores de arreglos con m hiperplanos. Por lo tanto, para $k = m + 2^m$, la clase de todos los clasificadores de arreglos con m hiperplanos es una subclase de $C^{(k)}$. De este hecho [9] establece el siguiente resultado de aproximación.

Teorema 2.1.1. *Si $C^{(k)}$ es la clase de todos los clasificadores de redes neuronales con la función sigmoide* (2.4) *y k neuronas en 2 capas ocultas, entonces*

$$\lim_{k \to \infty} \inf_{\phi \in C^{(k)}} L(\phi) - L_B = 0$$

para toda distribución de (X, Y).

Mostraremos que la misma propiedad se cumple si $C^{(k)}$ es la clase de las redes neuronales con una capa oculta con k neuronas ocultas, en particular lo haremos con la función sigmoide (2.4).

Nuevamente, consideremos a $C^{(k)}$ como la clase de los clasificadores del tipo (2.1) con ψ como en (2.3).

Por el teorema 1.6.1, tenemos que

$$L(\phi) - L_B \leq 2E(|\psi(X) - \eta(X)|),$$

donde $\eta(x) = P(Y = 1|X = x)$. De este modo $\inf_{\phi \in C^{(k)}} L(\phi) - L_B \to 0$ cuando $k \to \infty$ si

$$E(|\psi_k(X) - \eta(X)|) \to 0$$

para alguna secuencia $\{\psi_k\}$ con $\phi_k \in C^{(k)}$ dada por $\phi_k(x) = I_{\{\psi_k(x) > \frac{1}{2}\}}$. Para la consistencia universal, necesitamos solo probar que la familia de ψ's puede aproximar cualquier η en el sentido de $L^1(\mu)$. En otras palabras, si la clase de funciones ψ es densa en $L^1(\mu)$ para cada μ, entonces el error de aproximación $\inf_{\phi \in C^{(k)}} L(\phi) - L_B \to 0$.

Del hecho de que la convergencia en L^∞ implica la convergencia en L^1 cuando la medida es finita y de que toda función continua sobre un cerrado y acotado es acotada, se tiene que otra condición suficiente para esto, es que la clase $\mathcal{F}$ de funciones ψ sea densa con la norma L^∞ en el espacio de funciones continuas $C[a, b]^d$ sobre $[a, b]^d$, donde $[a, b]^d$ denota el hipercubo de $\mathbb{R}^d$ definido por los vértices opuestos a y b, para a y b en $\mathbb{R}^d$.

Lema 2.1.2. *Supongamos que una sucesión $\{\mathcal{F}_k\}_{k=1}^{\infty}$ de clases de funciones se hace densa con la norma de L^∞ en el espacio de funciones continuas $C[a, b]^d$ (donde $[a, b]^d$ es el hipercubo en $\mathbb{R}^d$ definido por a, b). En otras palabras, supongamos que para cada $a, b \in \mathbb{R}^d$ y cada función acotada g,*

$$\lim_{k\to\infty} \inf_{f\in\mathcal{F}_k} \sup_{x\in[a,b]^d} |f(x) - g(x)| = 0.$$

Entonces para cualquier distribución de (X, Y),

$$\lim_{k\to\infty} \inf_{\phi\in C^{(k)}} L(\phi) - L_B = 0,$$

donde $C^{(k)}$ *es la clase de clasificadores* $\phi(x) = I_{\{\psi(x)>1/2\}}$ *con* $\psi \in \mathcal{F}_k$.

Demostración. Sea $\epsilon > 0$ fijo. Podemos encontrar a, b tales que $\mu\left([a,b]^d\right) \geq 1 - \frac{\epsilon}{3}$, donde μ es la medida de probabilidad de X. Sea $\hat{\eta}$ una función continua que se anula fuera de $[a,b]^d$ y tal que

$$E(|\eta(X) - \hat{\eta}(X)|) \leq \frac{\epsilon}{6},$$

lo cual se sigue del teorema de Lusin (para ver una prueba de este teorema, recomendamos revisar el teorema 7.10 de [11]).

Por hipótesis, para un k lo suficientemente grande, podemos encontrar $f \in \mathcal{F}_k$ tal que

$$\sup_{x\in[a,b]^d} |f(x) - \hat{\eta}(x)| \leq \frac{\epsilon}{6}.$$

Sea $\phi(x) = I_{\{f(x)>\frac{1}{2}\}}$, tenemos por el teorema 1.6.1 que

$$\begin{aligned} L(\phi) - L_B &\leq 2E(|f(X) - \eta(X)|I_{\{X\in[a,b]^d\}}) + \frac{\epsilon}{3} \\ &\leq 2E(|f(X) - \hat{\eta}(X)|I_{\{X\in[a,b]^d\}}) + 2E(|\eta(X) - \hat{\eta}(X)|) + \frac{\epsilon}{3} \\ &\leq 2 \sup_{x\in[a,b]^d} |f(x) - \hat{\eta}(x)| + 2E(|\eta(X) - \hat{\eta}(X)|) + \frac{\epsilon}{3} \\ &\leq \epsilon. \end{aligned}$$

□

Demostraremos ahora que la clase de funciones $C^{(k)}$, es decir, la clase de todos los clasificadores de redes neuronales con k neuronas y 1 capa oculta, es densa en el espacio de funciones continuas $C[a,b]^d$.

Teorema 2.1.3 (Teorema de aproximación universal)**.** *Para cada función continua* $f : [a,b]^d \to \mathbb{R}$ *y para cada* $\epsilon > 0$, *existe una red neuronal con una capa oculta y una función* $\psi(x)$ *como en* (2.3) *tal que*

$$\sup_{x\in[a,b]^d} |f(x) - \psi(x)| < \epsilon.$$

Demostración. Probaremos el teorema para la función sigmoide (2.4). La demostración para una función sigmoide, σ, arbitraria, se sigue de tomar t suficientemente grande y ver que $\sigma(tx)$ se aproxima a la función sigmoide (2.4).

Fijemos $\epsilon > 0$. Tomemos la aproximación de $f(x)$ por series de Fourier. El teorema de Stone-Weierstrass nos garantiza la existencia de un entero positivo M suficientemente grande, así como coeficientes reales $\alpha_1, ..., \alpha_M, \beta_1, ..., \beta_M$ distintos de cero y enteros m_{ij} para $i = 1, ..., M,\ j = 1, ..., d$, tales que

$$\sup_{x\in[a,b]^d} \left| \sum_{i=1}^{M} \left(\alpha_i \cos\left(\frac{\pi}{a} m_i^T x\right) + \beta_i \operatorname{sen}\left(\frac{\pi}{a} m_i^T x\right)\right) - f(x) \right| < \frac{\epsilon}{2}, \tag{2.5}$$

donde $m_i = (m_{i1}, ..., m_{id})$, $i = 1, ..., M$. Claramente, cada función continua sobre la recta real que es cero fuera de algún intervalo acotado puede ser aproximada, arbitrariamente, de manera uniforme sobre el intervalo por medio de redes neuronales, esto es, por funciones de la forma

$$\sum_{i=1}^{k} c_i \sigma(a_i x + b_i) + c_0.$$

Solo observe que la función indicadora de un intervalo $[b, c]$ puede ser escrita como $\frac{\sigma(x-b)+\sigma(-x+c)}{2}$. Esto implica que funciones acotadas como lo son sen y cos pueden ser arbitrariamente aproximadas por redes neuronales. En particular, existen redes neuronales $u_i(x)$, $v_i(x)$ con $i = 1, ..., M$, (es decir, mapeos de $\mathbb{R}^d$ a $\mathbb{R}$) tales que

$$\sup_{x\in[a,d]^d} \left| u_i(x) - \cos\left(\frac{\pi}{a} m_i^T x\right)\right| < \frac{\epsilon}{4M|\alpha_i|}$$

y

$$\sup_{x\in[a,d]^d} \left| v_i(x) - \operatorname{sen}\left(\frac{\pi}{a} m_i^T x\right)\right| < \frac{\epsilon}{4M|\beta_i|}.$$

Por lo tanto, aplicando la desigualdad del triángulo se obtiene

$$\sup_{x\in[a,b]^d} \left| \sum_{i=1}^{M} \left(\alpha_i \cos\left(\frac{\pi}{a} m_i^T x\right) + \beta_i \operatorname{sen}\left(\frac{\pi}{a} m_i^T x\right)\right) - \sum_{i=1}^{M} (\alpha_i u_i(x) + \beta_i v_i(x)) \right| < \frac{\epsilon}{2}. \tag{2.6}$$

Dado que las u_i's y las v_i's son redes neuronales, su combinación lineal

$$\psi(x) = \sum_{i=1}^{M} (\alpha_i u_i(x) + \beta_i v_i(x))$$

también es una red neuronal.

Finalmente, de las ecuaciones (2.5) y (2.6) se sigue, por la desigualdad del triángulo, que

$$\sup_{x\in[a,b]^d} |f(x) - \psi(x)| < \epsilon.$$

□

Este teorema establece que cualquier función continua de complejidad arbitraria definida en $\mathbb{R}^d$ puede ser aproximada sobre un conjunto cerrado y acotado por una red neuronal (hacia adelante) de una capa oculta y suficientes neuronas. Sin embargo la

convergencia puede ser arbitrariamente lenta para algunas f, pero si se restringe la clase de funciones, es posible obtener límites superiores para la tasa de convergencia, para esto, véase la página 520 de [9].

El siguiente corolario del teorema anterior es obtenido como consecuencia del lema 2.1.2.

Corolario 2.1.4. *Sea $C^{(k)}$ la clase que contiene todos los clasificadores de redes neuronales de una capa oculta con k neuronas ocultas y una función sigmoide arbitraria. Entonces para cualquier distribución de (X, Y),*

$$\lim_{k\to\infty} \inf_{\phi\in C^{(k)}} L(\phi) - L_B = 0.$$

La convergencia anterior también se vale si el rango de los parámetros a_{ij}, b_i, w_i es restringido a un intervalo $[-\beta_k, \beta_k]$, donde $\lim_{k\to\infty} \beta_k = 0$.

Observación 2.1.5. *Claramente el teorema de aproximación universal es un teorema de existencia, es decir, se nos garantiza aproximación de una función continua a través de una red neuronal de una capa oculta con un alto número de neuronas, aunque finito, pero el teorema en ningún momento nos habla de cómo obtener los valores apropiados que determinan a la red neuronal. Por lo tanto la importancia de este teorema es puramente teórica.*

2.2. Redes multiclase

Para esta sección consideramos la referencia [10]. Cuando el número de clases involucradas en el problema de clasificación es $s > 2$, se pueden implementar 2 estrategias:

- **Estrategia global**: Seguir una idea muy parecida al caso binario, en donde se estimaban las probabilidades a posteriori de cada clase, es decir, encontrar la solución global mediante la estimación simultánea de $P(Y = k|X = x)$ para cada $k = 1, 2, \ldots, s$.
- **Clasificación por pares**: Dividir el problema en subproblemas de dos clases, construir un conjunto de clasificadores por pares que resuelvan los subproblemas y combinar los resultados de este conjunto en una única probabilidad a posteriori por clase.

2.2.1. Estrategia global

El enfoque de estrategia global resulta ser uno de los enfoques más populares, pese a que para tareas de clasificación difíciles no es el más eficiente. Para entenderlo consideremos un problema en el que tenemos s clases, la red neuronal hacia adelante se diseña con s salidas de modo que el resultado se codifica en un vector de s entradas en las que solo una entrada es distinta de cero, es decir, el evento *"el patrón está en la clase i"* es señalado por el vector cuya i-ésima entrada es la única distinta de cero. De forma similar al caso binario, se puede probar que el valor esperado de las componentes del vector anterior, son las probabilidades a posteriori de las clases.

Ahora, es importante destacar dos importantes diferencias entre las redes neuronales para regresión y las redes neuronales para clasificación:

- Las funciones de activación de las capas de salida en una red neuronal para regresión son usualmente lineales, mientras que para el caso de clasificación tenemos funciones de activación no lineales como lo son las sigmoides, cuyas salidas tienen interpretaciones probabilistas, por lo que deben estar entre 0 y 1.
- Para el caso de clasificación, en el entrenamiento empleamos como función de costo la entropía cruzada (*Cross-entropy*) en lugar de mínimos cuadrados (véase la sección 2.3.1).

Una vez realizado el entrenamiento, es posible corroborar con total seguridad que la suma de las salidas sea igual a 1. El uso de la función *Softmax* permite que la condición anterior se cumpla de forma automática. Por otra parte, veremos que éste no representa un problema en la clasificación por pares.

2.2.2. Clasificación por pares

Para casos donde el problema de clasificación es complicado, a menudo suele ser más frecuente tomar la decisión de dividir un problema de s clases en $\frac{s(s-1)}{2}$ problemas de clasificación binaria. Las razones son las siguientes:

- Cuando se realiza la clasificación por pares, quien diseña el modelo puede hacer uso de diversos resultados teóricos y algoritmos relacionados con la separación lineal de clases.
- Las redes que resultan después de esta metodología son mucho más compactas, rápidas de entrenar y fáciles de analizar; también la interpretación probabilista es sencilla pues la red cuenta con una única salida.
- Debido a que las características relevantes para separar entre la clase A y la clase B no son necesariamente las mismas que separan a las clases A y C; por esta misma razón cada clasificador solo tiene las entradas relevantes para su propia tarea, a diferencia de la metodología global en la que un perceptrón multicapa debe tener las características relevantes de todas las clases de manera simultánea.

Al realizar los $\frac{s(s-1)}{2}$ clasificadores por pares, entonces se calculan probabilidades a posteriori (una por cada clasificador, pues recordemos que en el caso binario la probabilidad a posteriori de la otra clase se calcula en términos de la obtenida), las cuales, posiblemente se han obtenido usando separadores lineales (perceptrones sin capas), una vez obtenida esta información, la probabilidad a posteriori de la $i-$ésima clase para un vector de características x se calcula como se enuncia en la siguiente proposición.

Proposición 2.2.1. *Sea P una medida de probabilidad asociada a un problema de clasificación de s clases ajenas por pares. Entonces para cada $i = 1, 2, \ldots, s$ y cada $x \in X$*

$$P(Y = i|X = x) = \frac{1}{\sum_{j=1,\, j\neq i}^{s} \frac{1}{P_{ij}} - (s-2)},$$

donde P_{ij} es la probabilidad a posteriori de la clase i o la clase j, es decir, la probabilidad a posteriori obtenida al realizar la clasificación binaria entre las clases i y j ($P_{ij} = P(Y = i|Y \in \{i, j\}, X = x)$).

Demostración. Definiendo P_{ij} como se enuncia, se sigue de la definición de probabilidad condicional que

$$\begin{aligned} P_{ij} &= \frac{P(Y \in \{i,j\}, X = x, Y = i)}{P(Y \in \{i,j\}, X = x)} \\ &= \frac{P(Y = i, Y \in \{i,j\}|X = x)P(X = x)}{P(Y \in \{i,j\}, X = x)} \\ &= \frac{P(Y = i|X = x)}{P(Y \in \{i,j\}|X = x)} \\ &= \frac{P(Y = i|X = x)}{P(Y = i|X = x) + P(Y = j|X = x)}. \end{aligned}$$

Por comodidad, escribiremos $p_k = P(Y = k|X = x)$.

Queremos probar que $p_i = \frac{1}{\sum_{j=1,\ j\neq i}^{s} \frac{1}{P_{ij}} - (s-2)}$. Para ello notemos que

$$\sum_{j=1, j\neq i}^{s} \frac{1}{P_{ij}} = \frac{(s-2)p_i + \sum_{j=1}^{s} p_j}{p_i}$$

y como P es una medida de probabilidad, entonces $\sum_{j=1}^{s} p_j = 1$. Así, al separar y despejar obtenemos

$$\sum_{j=1, j\neq i}^{s} \frac{1}{P_{ij}} - (s-2) = \frac{1}{p_i}.$$

De aquí, el resultado deseado se sigue considerando los recíprocos multiplicativos. □

De la proposición anterior vemos que cualquier probabilidad asociada a un problema de clasificación multiclase se puede estimar en base a las probabilidades de los problemas binarios.

2.3. Entrenamiento de la red

En las secciones anteriores hemos estudiado la teoría de clasificadores binarios y llegamos a la justificación del uso de redes neuronales mediante la prueba del teorema de aproximación universal, sin embargo, aun tenemos algunas preguntas de interés:

(1) ¿Cuántas neuronas necesita la red?

(2) Dado un conjunto de datos de entrenamiento D_n, ¿cuáles son los parámetros de la red que mejor ajustan al problema?

Anteriormente hablamos sobre un vector de *pesos*, el cual determinaba la importancia de cada componente y en base a ello, se determina una red capaz de tomar una decisión adecuada acerca de la clase a la que un nuevo dato pertenece. Fue Rosenblatt (1962) quien descubrió el maravilloso potencial de cada regla de decisión lineal y las nombró *perceptrones*. Cambiando uno o mas pesos conforme llegan nuevos datos, esto permite una adaptación fácil y rápida de los pesos a nuevas situaciones. De este modo, un primer acercamiento de entrenamiento o aprendizaje siguiendo el modelo del cerebro humano estaba convirtiéndose en una realidad.

En este sentido, necesitamos una forma de estimar numéricamente los pesos y con ellos minimizar el error relativo a nuestro conjunto de entrenamiento. En otras palabras, buscamos una forma de medir nuestro error y que a su vez nos dé una idea de como podemos estimar los pesos que componen nuestra red, para esto es necesario hablar de *funciones de error* (también llamadas funciones de costo, funciones objetivo o funciones de pérdida).

En el siguiente apartado explicaremos la obtención de dichas funciones de costo. La elección de ésta queda sujeto al problema que estemos abordando (regresión o clasificación).

Observación 2.3.1. *El enfoque que tratamos aquí es un enfoque de máxima verosimilitud, para un enfoque bayesiano veáse sección 5.7 de [5].*

2.3.1. Funciones de costo

El desarrollo de esta subsección está basado principalmente en [5].

Hasta ahora hemos construido a nuestras redes como funciones de decisión multiparamétricas de un vector de entrada $x \in \mathbb{R}^d$, las cuales devuelven un escalar (o vector en el caso multicategórico) y. Una simple analogía del como determinamos los parámetros de nuestra red (pesos) es posible de hacer al comparar la búsqueda de los parámetros en el ajuste de curvas (véase sección 1.1 en [5]) y por lo tanto minimizar una función de error de suma de cuadrados.

En efecto, dado un conjunto de entrenamiento $D_n = \{(x_m, y_m) : m = 1, \ldots, n\}$ donde $x_m \in \mathbb{R}^d$ es el vector de características y y_m es su correspondiente vector objetivo, se busca minimizar la función de costo

$$\mathbf{E}(W) = \frac{1}{2} \sum_{m=1}^{n} \left\| \hat{y}(x_m, W) - y_m \right\|^2, \tag{2.7}$$

donde el valor $\hat{y}(x_m, W)$ hace referencia al valor predicho por la red de parámetros W para el dato m-ésimo.

De este modo, el objetivo de una función de costo en el aprendizaje automático estadístico es cuantificar lo cercano de nuestras predicciones producidas por la red neuronal (o el modelo de deep learning que estemos empleando) de los valores reales. Esto es, la función de costo mide la calidad de las predicciones de la red en base a una distancia entre lo observado y lo predicho.

Sin embargo, a como hemos comenzado en este trabajo con la visión probabilista de los clasificadores, podemos también dar una interpretación en el mismo sentido para el entrenamiento de nuestra red y sus correspondientes salidas. Esto clarificará el porqué de la elección de una función de activación no lineal en la capa de salida, así como la elección de la función de error.

Regresión

Comenzaremos planteando el problema de regresión y nos restringiremos al caso en el que la respuesta es un escalar. Asumamos que nuestra variable de respuesta y tiene una distribución gaussiana con una media dependiente de x, la cual está dada por la salida de la red neuronal, esto es

$$p\left(y\,|x,W\right)=\mathcal{N}\left(y\,|\hat{y}(x,W),\beta^{-1}\right)=\frac{1}{(2\pi\beta^{-1})^{\frac{1}{2}}}\exp\left[-\frac{1}{2\beta^{-1}}(y-\hat{y}(x,W))^2\right], \quad (2.8)$$

donde β corresponde a la precisión del ruido gaussiano.

Para la distribución condicional dada por (2.8), es suficiente tomar en la capa de salida la función de activación identidad, esto como consecuencia del teorema de aproximación universal. Dado un conjunto de n observaciones (independientes e idénticamente distribuidas) $X=\{x_1,x_2,\ldots,x_n\}$, con sus correspondientes valores objetivo $\mathbf{y}=\{y_1,y_2,\ldots,y_n\}$, podemos construir la correspondiente función de verosimilitud

$$L\left(y\,|x,W\right)=\prod_{m=1}^{n}p\left(y_m\,|x_m,W,\beta\right). \quad (2.9)$$

Tomando el negativo del logaritmo, obtenemos la función de error

$$\frac{\beta}{2}\sum_{m=1}^{n}\left[\hat{y}(x_m,W)-y_m\right]^2-\frac{n}{2}\ln(\beta)+\frac{n}{2}\ln(2\pi), \quad (2.10)$$

la cual puede ser utilizada para obtener los parámetros W y β. En la teoría de redes neuronales a menudo se busca minimizar una función de costo en lugar de la maximización de la (log) verosimilitud, por lo que aquí seguiremos esta convención. Consideremos primero el caso de determinar W. Maximizar la función de verosimilitud equivale a minimizar la función de error suma de cuadrados dada por

$$\mathbf{E}(W)=\frac{1}{2}\sum_{m=1}^{n}\left[\hat{y}(x_m,W)-y_m\right]^2,$$

en la cual se han descartado constantes aditivas y multiplicativas. Denotemos por W_{ML} al valor encontrado de W que minimiza $\mathbf{E}(W)$, esto porque será la solución del método de máxima verosimilitud. Cabe mencionar que en la práctica, el uso de funciones de activación en el calculo de $\hat{y}(x_m,W)$ provoca que la función de error no sea convexa y entonces, en la práctica eso hace que máximos locales de la función de verosimilitud sean hallados y por tanto, mínimos locales de la función de error.

Un cálculo rápido, una vez que se ha encontrado W_{ML}, nos permite mostrar que el valor de β, que puede ser encontrado minimizando la log verosimilitud, corresponde a

$$\frac{1}{\beta_{ML}} = \frac{1}{n}\sum_{m=1}^{n}\left[\hat{y}(x_m, W_{ML}) - y_m\right]^2 .$$

El caso en el que nuestras respuestas (variables objetivo) son vectores, asumiendo las condiciones de independencia, idénticamente distribuidas, condicionadas a x y W, así como misma precisión de ruido β, la distribución condicional está dada entonces por

$$\begin{aligned} p\left(y\,|x, W\right) &= \mathcal{N}\left(y\,|\hat{y}(x, W)\,, \beta^{-1}I\right) \\ &= \frac{1}{(2\pi)^{\frac{s}{2}}|\beta^{-1}I|^{\frac{1}{2}}}\exp\left[-\frac{1}{2}(y - \hat{y}(x, W))\beta I(y - \hat{y}(x, W)^T\right], \end{aligned}$$

donde s hace referencia a la dimensión del vector respuesta (el número de variables en nuestros vectores objetivo).

Argumentos similares al del caso en que tenemos valores escalares como respuesta, nos sirven para ver que los pesos que maximizan la verosimilitud son aquellos que minimizan la función de error de mínimos cuadrados (2.7). Para este caso, la precisión del ruido queda determinada por la expresión

$$\frac{1}{\beta_{ML}} = \frac{1}{ns}\sum_{m=1}^{n}\left\|\hat{y}(x_m, W_{ML}) - y_m\right\|^2 . \tag{2.11}$$

Es realmente mucho pedir todas estas restricciones a nuestros datos, sin embargo es posible extender esto a distribuciones condicionales más generales (véase sección 5.6 en [5]). Por otra parte el supuesto de independencia puede abandonarse con consecuencia de tener un problema de optimización más complejo.

Clasificación

Para el caso de clasificación comenzaremos con el más simple que es el de clasificación binaria, consideremos entonces que la variable objetivo y toma el valor $y = 1$ para denotar la clase $\mathcal{C}_1$ y $y = 0$ para la clase $\mathcal{C}_2$. Como ya se ha discutido previamente, en este caso nuestro objetivo es aproximarnos a la regla de Bayes, para ello es necesario estimar la decisión que queda determinada por la *probabilidad a posteriori* (véase sección 1.5). Por ello resulta conveniente considerar una red que tenga una única salida con función de activación sigmoide logística

$$\hat{y} = \sigma^{(l)}(\psi(x, W)) = \frac{1}{1 + \exp(-\psi(x, W))},$$

donde $\psi(x, W)$ es una red neuronal con vector de parámetros W, así garantizamos que $0 \leq y \leq 1$. De este modo podemos interpretar $\hat{y} = \hat{y}(x, W)$ como la probabilidad

condicional $p(\mathcal{C}_1|x)$, con $p(\mathcal{C}_2|x)$ dada por $1 - p(\mathcal{C}_1|x)$. La distribución condicional de la variable objetivo dada una entrada es una distribución Bernoulli de la forma

$$p(y|x, W) = \hat{y}(x, W)^y [1 - \hat{y}(x, W)]^{1-y}.$$

Si consideramos un conjunto de entrenamiento de observaciones independientes, entonces la función de error, la cual está dada por el negativo de la log verosimilitud, recibe el nombre de entropía cruzada (*cross-entropy*) y es la función de error que corresponde a la expresión

$$\mathbf{E}(W) = -\sum_{m=1}^{n} \left[y_m \ln(\hat{y}(x_m, W)) + (1 - y_m) \ln(1 - \hat{y}(x_m, W)) \right].$$

Note que en este caso no hay un análogo a la precisión del ruido β porque los valores objetivos se asumen que están correctamente etiquetados. Sin embargo, el modelo se puede ampliar fácilmente para permitir errores en el etiquetado.

Si tenemos ahora s clasificaciones binarias independientes a realizar, entonces podemos utilizar una red que tenga s salidas en la que cada una tenga una función de activación sigmoide logística. Asociando con cada salida una etiqueta de clase binaria $y_{(r)} \in \{0, 1\}$ donde $r = 1, 2, ..., s$ (esto es $y = [y_{(1)}, \ldots, y_{(s)}]$). Si asumimos que las etiquetas de clase son independientes, dado el vector de entrada, entonces la distribución condicional de nuestro vector objetivo es

$$p(y|x, W) = \prod_{r=1}^{s} \hat{y}_{(r)}(x, W)^{y_{(r)}} [1 - \hat{y}_{(r)}(x, W)]^{1-y_{(r)}}.$$

Tomando de igual forma, el negativo de su correspondiente función de log verosimilitud, obtenemos la función de error

$$\mathbf{E}(W) = -\sum_{m=1}^{n} \sum_{r=1}^{s} \left[y_{mr} \ln(\hat{y}_{(r)}(x_m, W)) + (1 - y_{mr}) \ln(1 - \hat{y}_{(r)}(x_m, W)) \right],$$

donde y_{mr} es la r-ésima entrada del vector y_m (esto es la entrada r-ésima de la etiqueta asociada al m-ésimo dato).

Finalmente, para el caso multiclase, consideremos el problema de clasificación en el que cada entrada está asignada a una de s clases mutuamente excluyentes. Las variables objetivos binarias $y_{(r)} \in \{0, 1\}$ tienen un esquema de codificación 1 a s que indica la clase, y las salidas de la red son interpretadas como $\hat{y}_{(r)}(x, W) = p(y_{(r)} = 1|x)$. La capa de salida tiene por función de activación, la función softmax, la cual está dada por la expresión

$$\hat{y}_{(r)}(x, W) = \frac{\exp(\psi_r(x, W))}{\sum_j \exp(\psi_j(x, W))},$$

donde $\psi_r(x, W)$ es una red neuronal con vector de parámetros W, la cual satisface $0 \leq \hat{y}_{(r)} \leq 1$ y $\sum_r \hat{y}_{(r)} = 1$. Lo anterior conduce a la función de costo

$$\mathbf{E}(W) = -\sum_{m=1}^{n}\sum_{r=1}^{s} y_{mr} \ln(\hat{y}_{(r)}(x_m, W)).$$

2.3.2. Funciones de activación

Tal y como se menciona en Charau [1], las funciones de activación juegan un papel muy importante en el proceso de construcción de un modelo por redes neuronales. En el caso del perceptrón para la clasificación binaria consideramos la función de activación $\sigma(x) = \text{sign}(x)$, sin embargo como se mostró antes, las redes resultan útiles para muy diversos problemas, por lo que resulta necesario tener presente la posibilidad de considerar otras funciones de activación. En efecto, en Montesinos [24] podemos ver un apartado dedicado a una explicación más detallada de las funciones de activación más comunes. A continuación dedicamos un espacio para presentar algunas funciones de activación que tomarán relevancia en este trabajo.

Lineal

Esta función de activación es la ya conocida función $\sigma(x) = ax$, donde la variable dependiente está en proporción directa con la variable independiente. Si nosotros tomamos todas las funciones de activación lineales, sin importar la red que tomemos, como se menciona en Bishop [5], siempre se podrá encontrar una red sin capas ocultas que sea equivalente, esto debido a que la composición de funciones lineales siempre será una función lineal. Debido a lo anterior, esta función de activación no debe ser considerada para problemas no lineales.

ReLU

La función *ReLU* (cuyas siglas son la abreviación de *rectifier linear unit*) es una de las funciones de activación más populares. Esta función es definida como

$$ReLU(x) = \text{máx}\{0, x\}.$$

De este modo, la ReLU activa un nodo si y sólo si la entrada es mayor que 0, cuando es menor, indistintamente de cuan menor es, la salida será 0, así, cuando dicho umbral se alcanza, se tiene una relación lineal con la variable dependiente. No hay que subestimar la simplicidad de la función ReLU, puesto que proporciona una transformación no lineal tal que al usar suficientes de éstos rectificadores lineales se puede aproximar de manera arbitraria funciones no lineales. La función ReLU ha mostrado ser eficaz en muy diversas situaciones, por lo que es, hasta ahora, un notable avance. Debido a que el gradiente de la función es cero o uno, no es fácil controlar el problema de desvanecimiento del gradiente, pese a ello, la función de activación ReLU, en la práctica, ha mostrado mejor comportamiento durante el entrenamiento, que las funciones de

activación sigmoides. Su uso es frecuente, mayormente, en las capas ocultas. Su uso en la capa de salida es, mayormente, para variables de respuesta continuas y positivas.

Sigmoide logístico

La función

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

corresponde a la llamada función de activación *sigmoide*. Esta función se asemeja a una S y posee la propiedad de convertir variables independientes de rango casi infinito en probabilidades (es decir, valores entre 0 y 1) y en la mayoría de las ocasiones, sus salidas corresponden a valores muy cercanos de 0 o 1. Los valores extremos o atípicos pueden ser reducidos sin necesidad de removerlos, tal como pasa con todas las transformaciones logísticas. Para la construcción de modelos de redes neuronales artificiales y deep learning, esta función de activación es de las más comunes cuando buscamos que la respuesta sea binaria o bien una probabilidad. Hay que destacar que esta función de activación, aunque con una armonía entre lineal y no lineal, debido a que los valores fuertemente negativos o positivos tienden a ser muy cercanos a 0 o 1, respectivamente, provocan que la función se "estanque", esto es que el proceso de aprendizaje no mejore debido a los valores de salida de esta función de activación, ya sean grandes o pequeños. En la sección 2.4.2 que trata sobre el *Backpropagation* se puede observar lo que se menciona en estas últimas líneas.

Tanh (Tangente hiperbólica)

La función *tangente hiperbólica* cuya expresión corresponde a

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)},$$

es una función de activación de uso recurrente en redes neuronales. La tangente hiperbólica trabaja bien en algunos casos, al igual que la función sigmoide su imagen corresponde a una forma de S suave, un poco más amplia en altura ya que la función mapea al intervalo $(-1, 1)$, razón por la que cuenta con la ventaja de tener una menor posibilidad de "estancarse", lo cual justifica su aplicación tan usual en las capas ocultas. Una característica más sobre esta función es que los valores fuertemente positivos tendrán un valor positivo cercano a 1, mientras que si son fuertemente negativos, tendrán valores cercanos a -1, lo cual recuerda al comportamiento de la función umbral que habíamos mencionado anteriormente en este trabajo.

Softmax

La función *softmax* es una generalización de la función de activación sigmoide logística, esta función es diseñada específicamente para etiquetas multicategóricas. Generalmente

la encontramos en la capa de salida de una red neuronal entrenada para clasificación en más de dos clases. Esta función devuelve un vector donde las entradas corresponden a la probabilidad de cada categoría, las cuales asumimos, son mutuamente excluyentes. Sin embargo, si tenemos un problema de modelación donde sólo nos interesa la clase con mayor probabilidad, usaríamos una capa de salida softmax con una función argmax() para obtener la clase con dicha probabilidad más alta. Es importante recalcar que si queremos realizar clasificación binaria, no usamos una función softmax sino una función de activación sigmoide. La expresión por entrada de la función softmax es

$$\text{Softmax}(x_r) = \frac{\exp(x_r)}{\sum_{j=1}^{s} \exp(x_j)}, \quad r = 1, 2, \ldots, s.$$

Observemos de la definición por entrada, que la suma es exactamente uno. De este modo, una fuerte predicción debería tener una sola entrada muy cercana a 1 y el resto de ellas cercanas a 0, por otro lado una predicción débil puede presentar valores igualmente probables para más de una categoría. Las funciones de activación sigmoide y softmax devuelven resultados probabilistas de las clases, lo que las vuelve adecuadas para interpretaciones probabilistas.

2.4. Optimización numérica

En esta sección explicaremos lo que hay detrás de la estimación de los pesos en una neurona con propagación hacia adelante y presentaremos las operaciones que ocurren en una red neuronal con una capa oculta, se presenta el proceso de optimización que hay detrás para determinar los pesos. La siguiente explicación está basada en [13].

Consideremos una función de costo, $\mathbf{E}(W)$, que es una función continuamente diferenciable de algún vector de pesos (parámetros) desconocido W. La función $\mathbf{E}(W)$ mapea los elementos W en números reales. Buscamos encontrar una solución óptima W^* tal que se satisfaga la condición

$$\mathbf{E}(W^*) \leq \mathbf{E}(W).$$

Esto es, necesitamos resolver un problema de optimización sin restricciones, en otras palabras, queremos minimizar la función de costo con respecto al vector de pesos W. De este modo la condición necesaria para optimizar es

$$\nabla \mathbf{E}(W^*) = 0,$$

donde ∇ es el operador gradiente

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_d} \right]^T$$

y $\nabla \mathbf{E}(W)$ es el vector gradiente de la función de costo

$$\nabla = \left[\frac{\partial \mathbf{E}}{\partial w_1}, \frac{\partial \mathbf{E}}{\partial w_2}, \ldots, \frac{\partial \mathbf{E}}{\partial w_d} \right]^T.$$

Una clase de algoritmos bien designados para esta tarea se basan en la idea de descenso iterativo localmente, estos a su vez consisten en considerar una solución inicial $W(0)$, la cual se usa para genera una sucesión de vectores de pesos $W(1), W(2), \ldots$, tales que la función de costo $\mathbf{E}(W)$ es reducida en cada iteración del algoritmo, es decir,

$$\mathbf{E}(W(n+1)) < \mathbf{E}(W(n)), \tag{2.12}$$

donde $W(n)$ es el valor antiguo del vector de pesos y $W(n+1)$ es el valor actualizado.

Se espera que el algoritmo eventualmente converja a una solución óptima W^*. Sin embargo, está la posibilidad de que diverja, es decir, sea inestable a menos que precauciones especiales sean tomadas.

2.4.1. Gradiente descendente

El desarrollo de esta subsección está basada principalmente en [13].

El método del *paso descendente*, también conocido como método del *gradiente descendente* (GD) es un método iterativo de primer orden, usado como mecanismo central en aprendizaje estadístico para modelos de entrenamiento, como lo son las redes neuronales.

En el método del gradiente descendente, el ajuste sucesivo que se aplica al vector de pesos W se hace en la dirección opuesta al vector gradiente $\nabla\mathbf{E}(W)$. De acuerdo a lo mencionado, formalmente el algoritmo para obtener la actualización es descrito por

$$W(n+1) = W(n) - \eta\nabla\mathbf{E}(W(n)),$$

donde η es una constante positiva que recibe el nombre de *tamaño de paso* o *parámetro de tasa de aprendizaje* y $\nabla\mathbf{E}(W(n))$ es el vector gradiente evaluado en el punto $W(n)$. Al pasar de la iteración n a la actualización $n+1$, el algoritmo aplica la corrección

$$\begin{aligned} \Delta W(n) &= W(n+1) - W(n) \\ &= -\eta\nabla\mathbf{E}(W(n)). \end{aligned} \tag{2.13}$$

Necesitamos mostrar que en efecto este algoritmo cumple con la condición en (2.12), la cual es deseada. Para mostrar esto, consideremos la expansión en series de Taylor de primer orden alrededor de $W(n)$ para aproximar $W(n+1)$. Entonces

$$\mathbf{E}(W(n+1)) \approx \mathbf{E}(W(n)) + \nabla\mathbf{E}^T(W(n))\Delta W(n).$$

Sustituyendo (2.13) en nuestra anterior aproximación, obtenemos

$$\begin{aligned} \mathbf{E}(W(n+1)) &\approx \mathbf{E}(W(n)) - \eta\nabla\mathbf{E}^T(W(n))\nabla\mathbf{E}(W(n)) \\ &= \mathbf{E}(W(n)) - \eta||\nabla\mathbf{E}(W(n))||^2, \end{aligned}$$

lo anterior muestra que para un tamaño de paso η, la función de costo es decreciente cuando el algoritmo se actualiza desde una iteración anterior. Sin embargo, tengamos

en cuenta que el razonamiento presentado es aproximado en el sentido de que este resultado final solo es cierto para tasas de aprendizaje lo suficientemente pequeñas.

El método del gradiente descendente converge lentamente a la solución óptima W^*. Más aún, el parametro η tiene una profunda influencia en el comportamiento de la convergencia.

- Cuando η es pequeño, la respuesta temporal del algoritmo está *sobreamortiguada*, en este sentido, la trayectoria trazada por $W(n)$ sigue una trayectoria suave en el plano W (véase figura 2.3a).
- Cuando η es grande, la respuesta temporal del algoritmo está *subamortiguada*, en este sentido, la trayectoria trazada por $W(n)$ sigue una trayectoria en zigzag (oscilatoria) en el plano W (véase figura 2.3b).
- Cuando η excede un cierto valor crítico, el algoritmo se vuelve inestable (diverge).

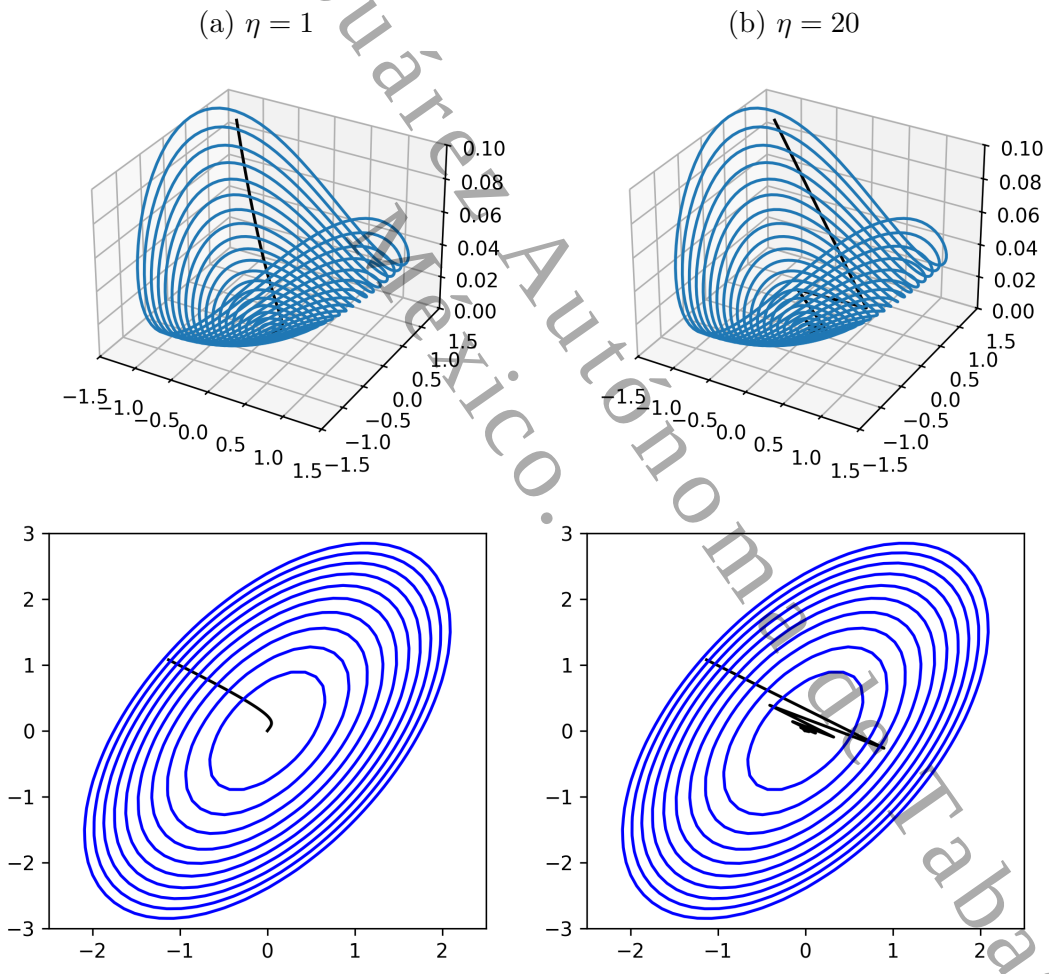


Figura 2.3: Comportamiento del gradiente descendente respecto a cambios en η.

Existen otros métodos usados para la estimación de los parámetros de nuestra red (como el método de Newton) que omitiremos, sin embargo, queremos justificar el uso del gradiente descendente como la base de los algoritmos prácticos para entrenamiento

de redes neuronales. En efecto, cada evaluación del gradiente aporta d elementos de información (dimensión de W), entonces podríamos esperar encontrar el mínimo de la función en $O(d)$ evaluaciones del gradiente. Como se verá mas adelante, con el uso del algoritmo *backpropagation*, cada una de esas evaluaciones requiere solo $O(d)$ pasos y, por lo tanto, ahora el mínimo se puede encontrar en $O(d^2)$ pasos.

2.4.2. El algoritmo Backpropagation

En esta sección tendremos como objetivo presentar una manera eficiente en la que podamos evaluar el gradiente de nuestra función de costo $\mathbf{E}(W)$ para nuestra red neuronal. Para ello nos basamos en la construcción presentada en Montesinos et al. [24] y Bishop [5].

El proceso de entrenamiento de la red consume muchos recursos computacionales y es por ello que durante muchos años las aplicaciones reales de las redes neuronales se vieron limitadas, hasta mediados-finales de la década de los 80's, que fue el momento en que el algoritmo *backpropagation* llegó. Comparado con otras técnicas de aprendizaje automático estadístico, el algoritmo backpropagation es realmente lento, sin embargo fue este quien provocó el resurgimiento del interés en las redes neuronales en los 80's.

Algunas de las ventajas del algoritmo backpropagation son:

(1) Puede aplicarse para la predicción de resultados categóricos o continuos.

(2) En su rubro, es uno de los mejores algoritmos para la captura de patrones complejos.

(3) No es necesario suponer demasiadas cosas sobre una relación profunda en los datos.

Sin embargo este algoritmo no está libre de debilidades, algunas de las cuales son:

(1) El costo computacional va ligado a la forma de la red, entre más compleja sea, mayores recursos computacionales serán necesarios, esto en general es cierto, no solo para redes neuronales.

(2) Es muy susceptible al sobreajuste de los datos de entrenamiento.

(3) Es difícil interpretar los resultados.

La meta del algoritmo backpropagation es encontrar los pesos de una red neuronal multicapa. Sabemos por el teorema de aproximación universal que es posible aproximar cualquier función con cualquier nivel de precisión usando suficientes neuronas ocultas, lo cual hace a las redes neuronales multicapa una poderosa herramienta del aprendizaje automático.

Muchas funciones de costo que son de interés práctico, por ejemplo la definida por la máxima verosimilitud para un conjunto de datos independientes e idénticamente distribuidos, está compuesta por una suma de términos, un sumando por cada dato tomado del conjunto de entrenamiento, es decir

$$\mathbf{E}(W) = \sum_{m=1}^{n} \mathbf{E}_m(W). \tag{2.14}$$

Para ejemplificar mejor las operaciones a realizar en el algoritmo backpropagation haremos uso del caso de regresión lineal/no lineal, por lo que emplearemos la función de costo conocida como mínimos cuadrados.

Volviendo con nuestra función de costo, aquí consideraremos el problema de evaluar $\nabla \mathbf{E}_m(W)$ para cada uno de los términos que conforman la función de costo $\mathbf{E}$. Esto se puede utilizar para la optimización secuencial o pueden acumularse a lo largo del conjunto de entrenamiento en el caso de métodos por bloques.

Consideremos primero el caso de un modelo lineal, razón por la cual las salidas $\hat{y}_{(r)}$ (estamos pensando en el caso multicategórico, por lo que dicha salida viene a ser la predicción de la r-ésima neurona de respuesta) resultan ser combinaciones lineales de las variables de nuestros datos, esto es

$$\hat{y}_{(j)} = \sum_{p=1}^{d+1} w_{jp} x_{mp}, \tag{2.15}$$

donde $x_m = [x_{m1}, \ldots, x_{md}]$. Es importante destacar que hemos omitido el sesgo (bias) debido a que éste se puede considerar añadiendo una dimensión extra a nuestros datos y fijando su valor en 1.

La forma en que calculamos la predicción anterior tiene asociada una función de costo que para el m-ésimo dato, tiene la forma

$$\mathbf{E}_m = \frac{1}{2} \sum_{j=1}^{k} (y_{mj} - \hat{y}_{mj})^2, \tag{2.16}$$

donde $\hat{y}_{mj} = \hat{y}_{(j)}(x_m, W)$. Para cada coeficiente w_{jp}, el gradiente de la función de costo respecto a éste es

$$\Delta w_{jp} = \frac{\partial\ \mathbf{E}_m}{\partial w_{jp}} = (\hat{y}_{mj} - y_{mj}) x_{mp}, \tag{2.17}$$

lo cual puede entenderse como un cálculo "local" que involucra una "señal de error" $(\hat{y}_{mj} - y_{mj})$ que está asociada con el valor de salida al operar con el peso w_{jp} y p-ésima entrada del dato x_m.

Ahora, para una red neuronal con una capa oculta, presentaremos primero con más detalle, cómo "fluye"la información a través de la red. El primer paso es proveer a nuestra capa de entrada con los valores correspondientes (datos), ninguna operación se realiza con estos datos hasta el momento de que la información entra a las unidades ocultas. Entonces es cuando la entrada ponderada en la r-ésima neurona oculta es calculada como

$$z_{mr}^{(h)} = \sum_{p=1}^{d+1} w_{rp}^{(h)} x_{mp}, \tag{2.18}$$

donde d es la dimensión de nuestros datos (y por tanto el número de entradas en cada dato), $w_{rp}^{(h)}$ es el peso de la r-ésima neurona oculta en relación con la p-ésima entrada de nuestros datos, h se refiere a la capa oculta y x_{mp} es el valor de la p-ésima entrada del dato número m. Recordemos que hemos omitido el sesgo (bias) como en el caso anterior. Después que hemos hecho nuestra combinación lineal de los datos en la neurona r de la capa oculta, la función de activación recibe dicho resultado y devuelve la expresión

$$V_{mr}^{(h)} = \sigma^{(h)}(z_{mr}^{(h)}), \tag{2.19}$$

donde $\sigma^{(h)}$ es la función de activación que se aplica a la combinación lineal de cualquier neurona r de la capa oculta. Una vez se han efectuado las evaluaciones por la función de activación en cada neurona de la capa oculta, tomamos estos nuevos valores pasando a la capa de salida y tomamos la combinación lineal para la entrada de la j-ésima neurona de salida como sigue

$$z_{mj}^{(l)} = \sum_{r=1}^{s+1} w_{jr}^{(l)} V_{mr}^{(h)}. \tag{2.20}$$

Obsérvese que el subíndice m de $z_{mj}^{(l)}$ se refiere al m-ésimo dato, por lo cual no hemos perdido "noción"del dato que estamos manipulando en la red, además hay que destacar que, en este caso, el superíndice de la suma hace referencia al número de neuronas más 1 (estamos considerando nuevamente que tenemos una neurona más en la capa oculta con valor fijo 1 para el sesgo, pero ahora de la neurona de respuesta), la l nos indica que es la capa de salida/respuesta.

Una vez que se han calculado las combinaciones lineales para la capa de salida, éstas entran dentro de la función de activación $\sigma^{(l)}$ (no necesariamente debe ser la misma que la empleada en la capa oculta) y se toman entonces las predicciones

$$\hat{y}_{mj} = \sigma^{(l)}(z_{mj}^{(l)}). \tag{2.21}$$

Esta predicción corresponde entonces a la j-ésima neurona de la capa de respuesta en relación al dato número m. De este modo habrán tantas respuestas por dato como categorías se tengan y tantos vectores respuesta como datos se tengan.

Hay que aclarar que como usamos la función de mínimos cuadrados como función de costo para la ilustración del algoritmo, la ecuación (2.16) sigue manteniéndose para cada $m = 1, 2, \ldots, n$.

Con todos estos elementos estamos en posición de atacar nuestro problema y plantear el algoritmo backpropagation, para ello recordemos que nuestro objetivo es encontrar los pesos $(w_{rp}^{(h)}, w_{jr}^{(l)})$ que nos ayuden a minimizar (2.14). Como ya se ha mencionado, la metodología usual es emplear el gradiente descendente, razón por la cual necesitamos

calcular ciertas derivadas, por ejemplo, si queremos aplicar el gradiente para los pesos $w_{jr}^{(l)}$, buscamos calcular

$$\Delta w_{jr}^{(l)} = -\eta \frac{\partial \mathbf{E}}{\partial w_{jr}^{(l)}} = -\eta \sum_{m=1}^{n} \frac{\partial \mathbf{E}_m}{\partial w_{jr}^{(l)}}. \tag{2.22}$$

El algoritmo backpropagation nos permite calcular estas cantidades. En efecto, al sustituir en (2.16) los valores correspondientes por las ecuaciones (2.18)-(2.21), se obtiene

$$\mathbf{E}_m = \frac{1}{2} \sum_{j=1}^{k} \left[y_{mj} - \sigma^{(l)} \left(\sum_{r=1}^{s+1} w_{jr}^{(l)} \sigma^{(h)} \left(\sum_{p=1}^{d+1} w_{rp}^{(h)} x_{mp} \right) \right) \right]^2. \tag{2.23}$$

Esta expresión nos sirve para ver como expandir (2.22), haciendo uso de la regla de la cadena, tenemos

$$\Delta w_{jr}^{(l)} = \eta \sum_{m=1}^{n} \frac{\partial \mathbf{E}_m}{\partial \hat{y}_{mj}} \frac{\partial \hat{y}_{mj}}{\partial z_{mj}^{(l)}} \frac{\partial z_{mj}^{(l)}}{\partial w_{jr}^{(l)}}, \tag{2.24}$$

y de las ecuaciones (2.16), (2.20), (2.21) expresamos las derivadas necesarias

$$\begin{aligned} \frac{\partial \mathbf{E}_m}{\partial \hat{y}_{mj}} &= -(y_{mj} - \hat{y}_{mj}), \\ \frac{\partial \hat{y}_{mj}}{\partial z_{mj}^{(l)}} &= \sigma^{(l)\prime}\left(z_{mj}^{(l)}\right), \\ \frac{\partial z_{mj}^{(l)}}{\partial w_{jr}^{(l)}} &= V_{mr}^{(h)}. \end{aligned} \tag{2.25}$$

Podemos sustituir estas derivadas parciales en (2.24) para obtener el cambio en los pesos asociados con las unidades de salida, $\Delta w_{jr}^{(l)}$, los cuales terminan expresados como

$$\Delta w_{jr}^{(l)} = \eta \sum_{m=1}^{n} (y_{mj} - \hat{y}_{mj}) \sigma^{(l)\prime}\left(z_{mj}^{(l)}\right) V_{mr}^{(h)} = \eta \sum_{m=1}^{n} \delta_{mj} V_{mr}^{(h)}, \tag{2.26}$$

donde $\delta_{mj} = (y_{mj} - \hat{y}_{mj}) \sigma^{(l)\prime}\left(z_{mj}^{(l)}\right)$. Por lo tanto, la fórmula usada para actualizar los pesos para las unidades en la capa de salida es

$$w_{jr}^{(l)(t+1)} = w_{jr}^{(l)(t)} + \Delta w_{jr}^{(l)} = w_{jr}^{(l)(t)} + \eta \sum_{m=1}^{n} \delta_{mj} V_{mr}^{(h)}. \tag{2.27}$$

Esta ecuación refleja que los pesos ajustados son agregados a la estimación actual, $w_{jr}^{(l)(t)}$ para obtener la actualización estimada $w_{jr}^{(l)(t+1)}$.

El siguiente paso consiste en actualizar los pesos que estén asociados con las unidades ocultas. Siguiendo un proceso similar al de (2.26) obtenemos

$$\Delta w_{rp}^{(h)} = -\eta \frac{\partial \mathbf{E}}{\partial w_{rp}^{(h)}} = -\eta \sum_{m=1}^{n} \frac{\partial \mathbf{E}_m}{\partial w_{rp}^{(h)}}. \tag{2.28}$$

Usando la regla de la cadena obtenemos

$$\Delta w_{rp}^{(h)} = -\eta \sum_{m=1}^{n} \sum_{j=1}^{k} \frac{\partial \mathbf{E}_m}{\partial \hat{y}_{mj}} \frac{\partial \hat{y}_{mj}}{\partial z_{mj}^{(l)}} \frac{\partial z_{mj}^{(l)}}{\partial V_{mr}^{(h)}} \frac{\partial V_{mr}^{(h)}}{\partial z_{mr}^{(h)}} \frac{\partial z_{mr}^{(h)}}{\partial w_{rp}^{(h)}}, \tag{2.29}$$

donde $\frac{\partial \mathbf{E}_m}{\partial \hat{y}_{mj}}$ y $\frac{\partial \hat{y}_{mj}}{\partial z_{mj}^{(l)}}$ están determinadas en (2.25), mientras que el resto de derivadas se calculan, nuevamente, a partir de las ecuaciones (2.18), (2.19) y (2.20), dando como resultado las siguientes expresiones

$$\begin{aligned} \frac{\partial z_{mj}^{(l)}}{\partial V_{mr}^{(h)}} &= w_{jr}^{(l)}, \\ \frac{\partial V_{mr}^{(h)}}{\partial z_{mr}^{(h)}} &= \sigma^{(h)\prime}\left(z_{mr}^{(h)}\right), \\ \frac{\partial z_{mr}^{(h)}}{\partial w_{rp}^{(h)}} &= x_{mp}. \end{aligned} \tag{2.30}$$

Obsérvese que el sumando es sobre el número de salidas que genere nuestra neurona, esto es necesario ya que cada neurona en la capa oculta está conectada con todas las neuronas en la capa de respuesta, por esto es necesario considerar que todas las salidas deben verse afectadas si los pesos en la capa oculta cambian.

Sustituyendo nuevamente los cambios para los pesos relacionados con las unidades ocultas $\Delta w_{rp}^{(h)}$, se obtiene

$$\Delta w_{rp}^{(h)} = \eta \sum_{m=1}^{n} \sum_{j=1}^{k} \delta_{mj} w_{jr}^{(l)} \sigma^{(h)\prime}\left(z_{mr}^{(h)}\right) x_{mp} = \eta \sum_{m=1}^{n} \psi_{mr} x_{mp}, \tag{2.31}$$

donde $\psi_{mr} = \sum_{j=1}^{s} \delta_{mj} w_{jr}^{(l)} \sigma^{(h)\prime}\left(z_{mr}^{(h)}\right)$. De este modo, una expresión similar a la ecuación (2.27) se cumple para las unidades ocultas y se expresa como

$$w_{rp}^{(h)(t+1)} = w_{rp}^{(h)(t)} + \Delta w_{rp}^{(h)} = w_{rp}^{(h)(t)} + \eta \sum_{m=1}^{n} \psi_{mr} x_{mp}. \tag{2.32}$$

De forma parecida, esta ecuación refleja como los pesos ajustados son agregados a la estimación actual, $w_{rp}^{(h)(t)}$, para obtener la actualización estimada $w_{rp}^{(h)(t+1)}$.

Aquí hemos presentado, por simplicidad, el algoritmo para un caso de regresión con una capa oculta. Sin embargo, el algoritmo puede llevarse a redes con más capas, así como con otras funciones de costo. En efecto, como se menciona en [2], se puede

Regresión no lineal	
$\mathbf{E}(W)$	$\frac{1}{2}\sum_{m=1}^{n}\left[\hat{y}_m - y_m\right]^2$
$\sigma^{(l)}$	Identidad
$\Delta w_r^{(l)}$	$\eta\sum_{m=1}^{n}(y_m - \hat{y}_m)\sigma^{(h)}\left(z_{mr}^{(h)}\right)$
$\Delta w_{rp}^{(h)}$	$\eta\sum_{m=1}^{n}(y_m - \hat{y}_m)w_r^{(l)}\sigma^{(h)\prime}\left(z_{mr}^{(h)}\right)x_{mp}$
Regresión (con múltiples salidas)	
$\mathbf{E}(W)$	$\frac{1}{2}\sum_{m=1}^{n}\sum_{j=1}^{s}\left[\hat{y}_{mj} - y_{mj}\right]^2$
$\sigma^{(l)}$	Identidad (multivariable)
$\Delta w_{jr}^{(l)}$	$\eta\sum_{m=1}^{n}(y_{mj} - \hat{y}_{mj})\sigma^{(h)}\left(z_{mr}^{(h)}\right)$
$\Delta w_{rp}^{(h)}$	$\eta\sum_{m=1}^{n}\sum_{j=1}^{k}(y_{mj} - \hat{y}_{mj})w_{jr}^{(l)}\sigma^{(h)\prime}\left(z_{mr}^{(h)}\right)x_{mp}$
Clasificación binaria	
$\mathbf{E}(W)$	$-\sum_{m=1}^{n}\left[y_m\ln(\hat{y}_m) + (1-y_m)\ln(1-\hat{y}_m)\right]$
$\sigma^{(l)}$	Sigmoide logística
$\Delta w_j^{(l)}$	$\eta\sum_{m=1}^{n}(y_{mj} - \hat{y}_{mj})\sigma^{(h)}\left(z_{mr}^{(h)}\right)$
$\Delta w_{rp}^{(h)}$	$\eta\sum_{m=1}^{n}(y_m - \hat{y}_m)w_r^{(l)}\sigma^{(h)\prime}\left(z_{mr}^{(h)}\right)x_{mp}$
Clasificación multiclase	
$\mathbf{E}(W)$	$-\sum_{m=1}^{n}\sum_{r=1}^{s}y_{mr}\ln(\hat{y}_{mr})$
$\sigma^{(l)}$	Softmax
$\Delta w_{jr}^{(l)}$	$\eta\sum_{m=1}^{n}(y_{mj} - \hat{y}_{mj})\sigma^{(h)}\left(z_{mr}^{(h)}\right)$
$\Delta w_{rp}^{(h)}$	$\eta\sum_{m=1}^{n}\sum_{j=1}^{k}(y_{mj} - \hat{y}_{mj})w_{jr}^{(l)}\sigma^{(h)\prime}\left(z_{mr}^{(h)}\right)x_{mp}$

Tabla 2.1: Funciones de costo y actualizaciones en el gradiente descendente por tipo de problema.

migrar del caso de regresión al caso de clasificación y las correspondientes ecuaciones de actualización a usar en el gradiente descendente serán idénticas, aunque eso no significa que den lo mismo. En esta misma cita se dan las ecuaciones de actualización para cada modelo manteniendo las ideas sobre función de costo y función de activación en la capa de salida, esto acorde a nuestro tipo de problema tal como mencionamos en la construcción de las funciones de costo, fácilmente se puede llegar a estas ecuaciones realizando los respectivos cambios en las primeras dos expresiones de (2.25). En la tabla 2.1 se presentan las funciones de costo y las actualizaciones en el gradiente descendente por cada tipo de problema de interés.

Por otro lado, el proceso para aplicar *backpropagation* a una red neuronal con más capas es exactamente el mismo, aplicamos primero el proceso hacia adelante (*forward propagation*) y después calculamos por la regla de la cadena las correspondientes derivadas, por ejemplo, para una red con dos capas ocultas, la capa de salida y la segunda capa oculta preservan las estructuras de las ecuaciones ya presentadas y para la primera capa hacemos propagación hacia atrás una vez más.

El siguiente ejemplo es un análogo al presentado por Bishop [5] en la sección 5.3.2.

Ejemplo 2.4.1. *Consideremos una red neuronal de una capa oculta para el caso de*

clasificación binaria (función de costo dada por la entropía cruzada), en las unidades de salida se usa por función de activación $\sigma^{(l)}$ a la sigmoide logística y en las unidades ocultas $\sigma^{(h)}$ a la tangente hiperbólica, esto es,

$$\sigma^{(h)}(x) = \frac{1}{1+e^{-x}},$$
$$\sigma^{(l)}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}.$$

Una característica útil de estas funciones de activación es que se cumplen las siguientes relaciones

$$\sigma^{(h)\prime}(x) = 1 - \left[\sigma^{(h)}(x)\right]^2,$$
$$\sigma^{(l)\prime}(x) = \left[1-\sigma^{(l)}(x)\right]\sigma^{(l)}(x).$$

Al usar la función de entropía cruzada como función de costo, tenemos que las expresiones de $\boldsymbol{E}_m$ son $-[y_m \ln(\hat{y}_m) + (1-y_m)\ln(1-\hat{y}_m)]$ para cada m. Realizando de este modo el "flujo" de la información a través de la red, los análogos de las ecuaciones (2.18)–(2.21) *son*

$$z_{mr}^{(h)} = \sum_{p=1}^{d+1} w_{rp}^{(h)} x_{mp},$$
$$V_{mr}^{(h)} = \sigma^{(h)}(z_{mr}^{(h)}),$$
$$z_{m}^{(l)} = \sum_{r=1}^{s+1} w_{r}^{(l)} V_{mr}^{(h)},$$
$$\hat{y}_m = \sigma^{(l)}(z_m^{(l)}).$$

Recordemos que estamos en el caso de clasificación binaria, por lo que solo tenemos una salida, por lo que se puede omitir el segundo subíndice en los elementos con superíndice (l).

Lo siguiente es calcular las derivadas parciales correspondientes, las que necesitamos explícitamente son

$$\frac{\partial \hat{y}_m}{\partial z_m^{(l)}} = \sigma^{(l)}\left(z_m^{(l)}\right)\left[1-\sigma^{(l)}\left(z_m^{(l)}\right)\right],$$
$$\frac{\partial V_{mr}^{(h)}}{\partial z_{mr}^{(h)}} = 1 - \left[\sigma^{(h)}\left(z_{mr}^{(h)}\right)\right]^2.$$

Observemos que

$$\frac{\partial \boldsymbol{E}_m}{\partial \hat{y}_m} = -\frac{y_m}{\hat{y}_m} + \frac{1-y_m}{1-\hat{y}_m} = \frac{\hat{y}_m - y_m}{\hat{y}_m(1-\hat{y}_m)}.$$

De este modo, se ve fácilmente, que

$$\frac{\partial \boldsymbol{E}_m}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial z_m^{(l)}} = \hat{y}_m - y_m.$$

Para simplificar las cuentas, hacemos

$$\delta_m = (\hat{y}_m - y_m)$$

y

$$\psi_{mr} = \left(1 - \left[\sigma^{(h)}\left(z_{mr}^{(h)}\right)\right]^2\right)\delta_m w_r^{(l)},$$

de este modo las ecuaciones correspondientes quedan como

$$\frac{\partial \boldsymbol{E}_m}{\partial w_{rp}^{(h)}} = \psi_{mr} x_{mp},$$

$$\frac{\partial \boldsymbol{E}_m}{\partial w_r^{(l)}} = \delta_m \sigma^{(h)}(z_{mr}^{(h)}).$$

2.4.3. Representación vectorial del algoritmo Backpropagation

Nos limitaremos aquí a representar en forma vectorial las operaciones del algoritmo Backpropagation para una red neuronal con 2 capas ocultas.

En efecto, supongamos que queremos construir una red neuronal con 2 capas ocultas para clasificación binaria, en la primera capa oculta queremos k neuronas ocultas y en la segunda queremos l neuronas ocultas. Suponemos que tenemos n datos de dimensión d. Para apoyo visual consideremos la figura 2.4.

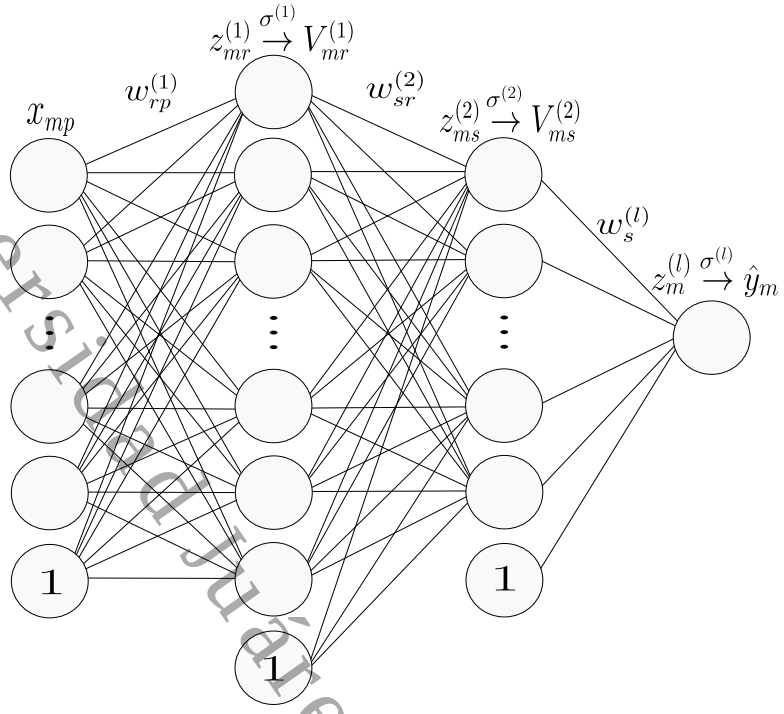


Figura 2.4: Representación del flujo de la información a través de una red neuronal con 2 capas ocultas.

Las ecuaciones correspondientes a la propagación hacia adelante (*Forward-propagation*) son las siguientes

$$\begin{aligned} z_{mr}^{(1)} &= \sum_{p=0}^{d} w_{rp}^{(1)} x_{mp}, \\ V_{mr}^{(1)} &= \sigma^{(1)}(z_{mr}^{(1)}), \\ z_{ms}^{(2)} &= \sum_{r=0}^{k} w_{sr}^{(2)} V_{mr}^{(1)}, \\ V_{ms}^{(2)} &= \sigma^{(2)}(z_{ms}^{(2)}), \\ z_m^{(l)} &= \sum_{s=0}^{l} w_s^{(l)} V_{ms}^{(2)}, \\ \hat{y}_m &= \sigma^{(l)}(z_m^{(l)}). \end{aligned} \tag{2.33}$$

De aquí y en base a (2.14), estamos interesados en obtener

$$\frac{\partial E}{\partial w_s^{(l)}}, \quad \frac{\partial E}{\partial w_{sr}^{(2)}}, \quad \frac{\partial E}{\partial w_{rp}^{(1)}}$$

para el cálculo del gradiente.

Observación 2.4.2. *En el caso de clasificación, asumiendo que la capa de salida tiene una función de activación sigmoide se tiene* $\frac{\partial E}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial z_m^{(l)}} = y_m - \hat{y}_m$.

Las ecuaciones del algoritmo backpropagation en base a (2.33) quedan como:

$$\begin{aligned}
\frac{\partial E}{\partial w_s^{(l)}} &= \sum_{m=1}^{n} \frac{\partial E}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial z_m^{(l)}}\frac{\partial z_m^{(l)}}{\partial w_s^{(l)}} \\
&= \sum_{m=1}^{n} (y_m - \hat{y}_m) V_{ms}^{(2)}, \\
\frac{\partial E}{\partial w_{sr}^{(2)}} &= \sum_{m=1}^{n} \frac{\partial E}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial z_m^{(l)}}\frac{\partial z_m^{(l)}}{\partial V_{ms}^{(2)}}\frac{\partial V_{ms}^{(2)}}{\partial z_{ms}^{(2)}}\frac{\partial z_{ms}^{(2)}}{\partial w_{sr}^{(2)}} \\
&= \sum_{m=1}^{n} (y_m - \hat{y}_m) w_s^{(l)} \sigma^{(2)'}(z_{ms}^{(2)}) V_{mr}^{(1)}, \\
\frac{\partial E}{\partial w_{rp}^{(1)}} &= \sum_{s=1}^{l}\sum_{m=1}^{n} \frac{\partial E}{\partial \hat{y}_m}\frac{\partial \hat{y}_m}{\partial z_m^{(l)}}\frac{\partial z_m^{(l)}}{\partial V_{ms}^{(2)}}\frac{\partial V_{ms}^{(2)}}{\partial z_{ms}^{(2)}}\frac{\partial z_{ms}^{(2)}}{\partial V_{mr}^{(1)}}\frac{\partial V_{mr}^{(1)}}{\partial z_{mr}^{(1)}}\frac{\partial z_{mr}^{(1)}}{\partial w_{rp}^{(1)}} \\
&= \sum_{s=1}^{l}\sum_{m=1}^{n} (y_m - \hat{y}_m) w_s^{(l)} \sigma^{(2)'}(z_{ms}^{(2)}) w_{sr}^{(2)} \sigma^{(1)'}(z_{mr}^{(1)}) x_{mp}.
\end{aligned} \tag{2.34}$$

Apoyados de los superíndices y observando los cambios de la primera ecuación a la segunda y de la segunda a la tercera, se nota el patrón general de como serían las ecuaciones del algoritmo backpropagation si se tuviesen más capas ocultas.

Para continuar, nos apoyaremos de la siguiente notación para mejor comprensión de las cuentas.

- $Y = [Y_1, \ldots, Y_n]$, donde y_i corresponde a la etiqueta del i-ésimo dato.
- $X = [X_1, \ldots, X_n]$, donde $X_i = \begin{bmatrix} x_{i0} \\ x_{i1} \\ \vdots \\ x_{id} \end{bmatrix}$, es decir, X_i es una forma simplificada de referirnos a las características del i-ésimo dato.
- $W_1 = \begin{bmatrix} W_1^{(1)} \\ \vdots \\ W_k^{(1)} \end{bmatrix}$, donde $W_i^{(1)} = \left[w_{i0}^{(1)}, \ldots, w_{id}^{(1)}\right]$, es decir, $W_i^{(1)}$ es una forma simplificada de referirnos a los pesos de la i-ésima neurona de la primera capa oculta con las características de los datos.
- $W_2 = \begin{bmatrix} W_1^{(2)} \\ \vdots \\ W_l^{(1)} \end{bmatrix}$, donde $W_j^{(2)} = \left[w_{j0}^{(2)}, \ldots, w_{jk}^{(2)}\right]$, es decir, $W_j^{(2)}$ es una forma simplificada de referirnos a los pesos de la j-ésima neurona de la segunda capa oculta con las neuronas de la primera capa oculta.
- $W_l = [w_0, \ldots, w_l]$, donde $w_m \in \mathbb{R}$, es decir, W_l representa los pesos que conectan la segunda capa oculta con la neurona de la capa de salida.

No perdamos de vista que

- $\dim(Y) = 1 \times n$,
- $\dim(X) = (d+1) \times n$,
- $\dim(W_1) = k \times (d+1)$,
- $\dim(W_2) = l \times (k+1)$,
- $\dim(W_l) = 1 \times (l+1)$.

Con la notación propuesta, tenemos que las ecuaciones en (2.33) se reescriben de forma general como:

$$
\begin{aligned}
&Z_1 = \left(z_{mr}^{(1)}\right) = \left(W_r^{(1)} X_m\right) = W_1 X, \\
&V^1 = \left[V_1^{(1)}, \ldots, V_n^{(1)}\right] \text{ donde } V_m^{(1)} = \begin{bmatrix} V_{m0}^{(1)} \\ \vdots \\ V_{mk}^{(1)} \end{bmatrix}, \; V_{m0}^{(1)} = 1, \; V_{mr}^{(1)} = \sigma^{(1)}(z_{mr}^{(1)}) \\
&Z_2 = \left(z_{ms}^{(2)}\right) = \left(W_s^2 V_m^{(1)}\right) = W_2 V^1 \\
&V^2 = \left[V_1^{(2)}, \ldots, V_n^{(2)}\right] \text{ donde } V_m^{(2)} = \begin{bmatrix} V_{m0}^{(2)} \\ \vdots \\ V_{ml}^{(2)} \end{bmatrix}, \; V_{m0}^{(2)} = 1, \; V_{ms}^{(1)} = \sigma^{(2)}(z_{ms}^{(2)}), \\
&Z_l = \left(z_m^{(l)}\right) = \left(w_l V_m^{(2)}\right) = W_l V^2, \\
&\hat{Y} = \sigma^{(l)}(Z_l),
\end{aligned}
$$

con

- $\dim(Z_1) = k \times n$,
- $\dim(V^1) = (k+1) \times n$,
- $\dim(Z_2) = l \times n$,
- $\dim(V^2) = (l+1) \times n$,
- $\dim(\hat{Y}) = 1 \times n$.

A partir de estas ecuaciones, tenemos que

$$
\begin{aligned}
\frac{\partial E}{\partial W_l} &= \left[\frac{\partial E}{\partial w_0^{(l)}}, \ldots, \frac{\partial E}{\partial w_l^{(l)}}\right] \\
&= \left[\sum_{m=1}^{n} (y_m - \hat{y}_m) V_{m0}^{(2)}, \ldots, \sum_{m=1}^{n} (y_m - \hat{y}_m) V_{ml}^{(2)}, \right] \\
&= (Y - \hat{Y}) V^{2^T}.
\end{aligned}
$$

Para las otras matrices definimos la siguiente operación, que es como un producto de matrices pero entrada por entrada:

$$
\begin{array}{ccc}
* : \mathcal{M}_{m\times n}(\mathbb{C}) \times \mathcal{M}_{m\times n} & \longrightarrow & \mathcal{M}_{m\times n} \\
(a_{ij}) * (b_{ij}) & \to & (a_{ij} b_{ij}).
\end{array}
$$

Continuando con la construcción, tenemos que calcular

$$\frac{\partial E}{\partial W_2} = \begin{bmatrix} \frac{\partial E}{\partial w_{10}^{(2)}} & \cdots & \frac{\partial E}{\partial w_{1k}^{(2)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial w_{l0}^{(2)}} & \cdots & \frac{\partial E}{\partial w_{lk}^{(2)}} \end{bmatrix}$$
$$= \begin{bmatrix} \sum_{m=1}^{n}(y_m - \hat{y}_m)w_1^{(l)}\sigma^{(2)'}(z_{m1}^{(2)})V_{m0}^{(1)} & \cdots & \sum_{m=1}^{n}(y_m - \hat{y}_m)w_1^{(l)}\sigma^{(2)'}(z_{m1}^{(2)})V_{mk}^{(1)} \\ \vdots & \ddots & \vdots \\ \sum_{m=1}^{n}(y_m - \hat{y}_m)w_l^{(l)}\sigma^{(2)'}(z_{ml}^{(2)})V_{m0}^{(1)} & \cdots & \sum_{m=1}^{n}(y_m - \hat{y}_m)w_l^{(l)}\sigma^{(2)'}(z_{ml}^{(2)})V_{mk}^{(1)} \end{bmatrix}.$$

Para determinar esta matriz definamos $\tilde{W}_l = [w_1, \ldots, w_l]$. Así, es fácil observar que

$$\tilde{W}_l^T(Y - \hat{Y}) = \begin{bmatrix} w_1(y_1 - \hat{y}_1) & \ldots & w_1(y_n - \hat{y}_n) \\ \vdots & \ddots & \vdots \\ w_l(y_1 - \hat{y}_1) & \ldots & w_l(y_n - \hat{y}_n) \end{bmatrix}$$

y si consideramos la matriz

$$T_1 = \begin{bmatrix} w_1(y_1 - \hat{y}_1)\sigma^{(2)'}(z_{11}^{(2)}) & \ldots & w_1(y_n - \hat{y}_n)\sigma^{(2)'}(z_{n1}^{(2)}) \\ \vdots & \ddots & \vdots \\ w_l(y_1 - \hat{y}_1)\sigma^{(2)'}(z_{1l}^{(2)}) & \ldots & w_l(y_n - \hat{y}_n)\sigma^{(2)'}(z_{nl}^{(2)}) \end{bmatrix}$$
$$= \tilde{W}_l^T(Y - \hat{Y}) * \sigma^{(2)'}(Z_2),$$

cuya dimensión es $l \times n$, un cálculo directo nos permite ver que

$$T_1 V^{1^T} = \frac{\partial E}{\partial W_2}.$$

Nos resta determinar la matriz

$$\frac{\partial E}{\partial W_1} = \begin{bmatrix} \frac{\partial E}{\partial w_{10}^{(1)}} & \cdots & \frac{\partial E}{\partial w_{1d}^{(1)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial w_{k0}^{(1)}} & \cdots & \frac{\partial E}{\partial w_{kd}^{(1)}} \end{bmatrix}$$
$$= \sum_{s=1}^{l} \begin{bmatrix} \gamma_{10}^{(s)} & \cdots & \gamma_{1d}^{(s)} \\ \vdots & \ddots & \vdots \\ \gamma_{k0}^{(s)} & \cdots & \gamma_{kd}^{(s)} \end{bmatrix},$$

donde $\gamma_{rp}^{(s)} = \sum_{m=1}^{n}(y_m - \hat{y}_m)w_s^{(l)}\sigma^{(2)'}(z_{ms}^{(2)})w_{sr}^{(2)}\sigma^{(1)'}(z_{mr}^{(1)})x_{mp}$.

Consideremos $\tilde{W}^2 = \begin{bmatrix} \tilde{W}_1^{(2)} \\ \vdots \\ \tilde{W}_l^{(1)} \end{bmatrix}$, donde $\tilde{W}_j^{(2)} = \left[w_{j1}^{(2)}, \ldots, w_{jk}^{(2)}\right]$.

Observemos que fijados m y r, tenemos que la cantidad

$$\sum_{s=1}^{l}(y_m - \hat{y}_m)w_s^{(l)}\sigma^{(2)'}(z_{ms}^{(2)})w_{sr}^{(2)}$$

se puede obtener a partir del producto punto de la columna r-ésima de $\tilde{W}^2$ con la columna m-ésima de T_1, esto es

$$\tilde{W}_2^T T_1 = \begin{bmatrix} \sum_{s=1}^{l} w_s(y_1 - \hat{y}_1)\sigma^{(2)'}(z_{1s}^{(2)})w_{s1}^{(2)} & \dots & \sum_{s=1}^{l} w_s(y_n - \hat{y}_n)\sigma^{(2)'}(z_{ns}^{(2)})w_{s1}^{(2)} \\ \vdots & \ddots & \vdots \\ \sum_{s=1}^{l} w_l(y_1 - \hat{y}_1)\sigma^{(2)'}(z_{1s}^{(2)})w_{sk}^{(2)} & \dots & \sum_{s=1}^{l} w_l(y_n - \hat{y}_n)\sigma^{(2)'}(z_{ns}^{(2)})w_{sk}^{(2)} \end{bmatrix}$$

cuya dimensión es $k \times n$.

Consideremos ahora

$$\begin{aligned} T_2 &= \sum_{s=1}^{l} \begin{bmatrix} w_s(y_1 - \hat{y}_1)\sigma^{(2)'}(z_{1s}^{(2)})w_{s1}^{(2)}\sigma^{(1)'}(z_{11}^{(1)}) & \dots & w_s(y_n - \hat{y}_n)\sigma^{(2)'}(z_{ns}^{(2)})w_{s1}^{(2)}\sigma^{(1)'}(z_{n1}^{(1)}) \\ \vdots & \ddots & \vdots \\ w_l(y_1 - \hat{y}_1)\sigma^{(2)'}(z_{1s}^{(2)})w_{sk}^{(2)}\sigma^{(1)'}(z_{1k}^{(1)}) & \dots & w_l(y_n - \hat{y}_n)\sigma^{(2)'}(z_{ns}^{(2)})w_{sk}^{(2)}\sigma^{(1)'}(z_{nk}^{(1)}) \end{bmatrix} \\ &= \tilde{W}_2^T T_1 * \sigma^{(1)'}(Z_1) \\ &= \tilde{W}_2^T(\tilde{W}_l^T(Y - \hat{Y}) * \sigma^{(2)'}(Z_2)) * \sigma^{(1)'}(Z_1). \end{aligned}$$

Continuando con lo obtenido previamente, una simple multiplicación de matrices nos permite ver que

$$T_2 X^T = \frac{\partial E}{\partial W_1}.$$

Por lo tanto, los pasos del gradiente descendente se pueden ver vectorialmente como

$$\begin{aligned} W_l(t+1) &= W_l(t) - \eta\frac{\partial E}{\partial W_l(t)} \\ &= W_l(t) - \eta(Y - \hat{Y}(t))V^2(t)^T, \\ W_2(t+1) &= W_2(t) - \eta\frac{\partial E}{\partial W_2(t)}V^1(t)^T \\ &= W_2(t) - \tilde{W}_l^T(t)\left[Y - \hat{Y}(t)\right] * \sigma^{(2)'}\left(Z_2(t)\right), \\ W_1(t+1) &= W_1(t) - \eta\frac{\partial E}{\partial W_1(t)} \\ &= W_1(t) - \eta\tilde{W}_2^T(t)\left[\tilde{W}_l^T(t)\left[Y - \hat{Y}(t)\right] * \sigma^{(2)'}\left(Z_2(t)\right)\right] * \sigma^{(1)'}\left(Z_1(t)\right)X^T, \end{aligned}$$

donde la evaluación en t representa la actualización al tiempo t.

Observación 2.4.3. *El número de sumandos que interfieren en cada una de las ecuaciones* (2.34) *puede ser obtenido a partir del número de conexiones (como se presentan en la figura 2.4) multiplicado por el número de datos. En efecto, si tuviésemos 3 capas ocultas, con k, l y m neuronas ocultas en cada capa respectiva y n datos para entrenamiento, entonces tendríamos desde cada peso que conecta la 3ra capa con la capa de salida en la ecuación correspondiente, n sumandos, para pesos que conectan la 2da capa con la 3ra tendríamos ahora nm sumandos y para pesos que conectan la 1ra capa con la 2da tendríamos nmk sumandos. Todo esto se deriva del grafo formado por la red neuronal, si cada arista corresponde a un peso, entonces el número de conexiones después del peso (a partir del vértice de llegada, es decir, la neurona correspondiente) forma el número de operaciones en las que se ve involucrado el peso y por tanto, el flujo de información de regreso viene de todas esas conexiones.*

Observación 2.4.4. *Basados en la observación anterior, la necesidad de recortar las matrices originales nace del hecho de que ningún peso de la capa actual influye en el sesgo (bias) de la capa siguiente, pues es el término independiente de cada combinación lineal en la capa que le sigue.*

Observación 2.4.5. *En base a las operaciones presentadas, un patrón es claro para las operaciones vectoriales del algoritmo backpropagation, conforme aumentamos el número de capas, es necesario formar una matriz auxiliar que constará de la multiplicación de dos matrices y luego operada entrada por entrada con una matriz de derivadas para luego multiplicar por otra matriz. Por ejemplo, si tuviéramos ahora 3 capas, tendríamos W_1, W_2, W_3 y W_l como pesos a optimizar, y entonces tendríamos que*

$$\begin{aligned}
\frac{\partial E}{\partial W_l} &= (Y - \hat{Y}){V^3}^T, \\
\frac{\partial E}{\partial W_3} &= T_1 {V^2}^T, \\
\frac{\partial E}{\partial W_2} &= T_2 {V^1}^T, \\
\frac{\partial E}{\partial W_1} &= T_3 X^T,
\end{aligned}$$

donde

$$\begin{aligned}
T_1 &= \tilde{W}_l^T (Y - \hat{Y}) * \sigma^{(3)'}(Z_3), \\
T_2 &= \tilde{W}_3^T T_1 * \sigma^{(2)'}(Z_2), \\
T_3 &= \tilde{W}_2^T T_2 * \sigma^{(1)'}(Z_1)
\end{aligned}$$

y las matrices V^i, Z_i y $\tilde{W}_i$ para $i = 1, 2, 3$ son las extensiones correspondientes a las que presentamos para el caso de dos capas.

2.4.4. Gradiente descendente con mini lotes (Mini-Batch)

Esta subsección está, principalmente, basada en [1].

Hemos hasta ahora, hablado acerca del método del gradiente descendente que nos permite buscar actualizaciones de los pesos, así como del método de backpropagation que nos ayuda a calcular las actualizaciones propuestas por el método del gradiente en un problema determinado, además, ya hemos visto también que la actualización de los parámetros depende directamente de los datos de entrenamiento, así que podemos cuestionarnos acerca de aplicaciones a gran escala, en los que el número de parámetros llega a millones, y es necesario efectuar las evaluaciones simultáneas de todos los datos hacia adelante y hacia atrás de la red para calcular las actualizaciones vía bacpropagation. Obsérvese que la ecuación (2.14) determina a la función de pérdida como suma de los errores puntuales, esto sucede en la mayoría de los procesos de optimización, con esto obtenemos la ecuación (2.22), la cual muestra como la actualización del gradiente completo, respecto a todos los puntos, resume los efectos específicos de cada punto.

Sin embargo, como se menciona en [1], no es práctico realizar el cálculo del gradiente empleando todo el conjunto de datos de una vez. Debe tenerse en cuenta que los requisitos de memoria de todas las predicciones intermedias y finales para cada instancia de entrenamiento deberían mantenerse mediante el descenso del gradiente. Esto último puede ser excesivamente alto en gran parte de los entornos prácticos. Por esta razón y también, debido que al comienzo del proceso de aprendizaje los pesos que conforman los parámetros de la red suelen estar muy alejados de lo óptimo, una pequeña muestra de datos puede utilizarse para crear una buena estimación de la dirección del gradiente. Lo comentado proporciona una base práctica para el éxito del *método del gradiente estocástico*, el cual consiste en actualizaciones específicas usando un dato a la vez en lugar de emplear todos los datos simultáneamente, este proceso suele ser más eficiente debido a que los problemas de aprendizaje automático presentan propiamente un alto nivel de redundancia entre el conocimiento capturado por los diferentes datos del entrenamiento.

Cabe aclarar que a este tipo de descenso de gradiente se le llama *estocástico* debido a que la forma en que se recorren los puntos es aleatoria. Hay que destacar que el efecto a largo plazo de las actualizaciones repetidas es aproximadamente el mismo, aunque cada actualización en el descenso de gradiente estocástico solo puede considerarse como una aproximación probabilista. Cada gradiente local puede calcularse eficientemente, lo que hace que el descenso de gradiente estocástico sea rápido, aunque a costa de la precisión en el cálculo del gradiente, arrojando en ocasiones resultados muy deficientes con ciertas ordenaciones de puntos de entrenamiento.

Por otro lado, el *gradiente descendente estocástico con mini lotes (Mini-Batch)* consiste en usar un conjunto B tomado del conjunto de entrenamiento de tal modo que la actualización de los parámetros se realice en base a esos puntos. En otras palabras, el gradiente descendente estocástico con mini lotes es un punto intermedio entre el gradiente estocástico que usa un solo dato y el gradiente descendente usual que emplea todos los datos a la vez, proporcionando de este modo la mejor relación entre estabilidad, velocidad y requisitos de memoria.

Finalmente, se debe explicar el funcionamiento de los mini lotes en el proceso de entrenamiento de nuestra red. Para implementarlos, primero definimos una *época* [24], el número de épocas es un hiperparámetro que determina la cantidad de ocasiones en las que funcionará el algoritmo de aprendizaje en todo el conjunto de entrenamiento. Una

época significa que cada uno de los datos del conjunto de entrenamiento ha tenido la oportunidad de contribuir en la actualización de los pesos de la red. Así, una época se compone de uno o más lotes, por ejemplo, si tenemos una muestra con 600 datos y deseamos construir lotes de 15 datos y 500 épocas, esto significa que el conjunto de datos se dividirá en 40 lotes, cada uno con 15 muestras. Los pesos de la red se actualizarán tras efectuar las operaciones con cada lote y esto también significa que una época implicará 40 lotes o 40 actualizaciones de 15 muestras para completarse. Como estamos deseando efectuar 500 épocas, el modelo usará todo el conjunto de datos 500 veces. Es decir, se efectuarán un total de $40 \times 500 = 20000$ lotes durante todo el proceso de entrenamiento.

2.5. Tratamiento de los datos y validación del modelo

Para esta sección, nuestras principales referencias fueron [24], [17], [10] y [5].

2.5.1. Escalamiento de los datos

Como se menciona en [24] y [17], es conveniente considerar formas de ajustar los datos de entrada antes del proceso de entrenamiento, ya que es muy usual que las entradas se midan en escalas muy diferentes, lo cual puede provocar que las de magnitudes más pequeñas no sean tomadas en cuenta durante el entrenamiento [10]. El proceso de hacer un tratamiento de los datos previo a un proceso de entrenamiento o análisis se llama escalamiento [17], preprocesamiento [10] o normalización [24] de los datos de entrada. En general, en los algoritmos de aprendizaje automático estadístico y en el análisis de datos es una preocupación común el proceso de escalamiento, pues su realización contribuye a mejorar la estabilidad numérica en el proceso de estimación de algunos algoritmos. A continuación exponemos los procesos de escalamiento que se mencionan en Montesinos [24] y Dreyfus [10].

Variables Booleanas

Cuando se utilizan valores de 0 y 1 para variables Booleanas, éstas pueden ser transformadas en valores de -1 y 1 respectivamente. Lo anterior contribuye a centralizar las variables y en el contexto de redes neuronales, que la contribución de esa característica no sea nula para un cierto conjunto de nuestros datos.

Centrado

Este proceso de escalamiento, como su nombre lo dice, centra los datos, para ello, se calcula la media muestral y se le resta a cada uno de los datos, esto es, si $X_1, \ldots, X_n \in \mathbb{R}^d$

son nuestros datos con $\overline{X}$ su media muestral, entonces nuestros nuevos datos escalados se toman como

$$X_m^* = X_m - \overline{X},\ m = 1, 2 \ldots, n.$$

Observemos que este proceso coloca al vector $\mathbf{0}$ como eje central entre nuestros datos, esto permite que tengamos datos tanto con entradas negativas como positivas (véase la figura 2.5b).

Escalado

Este proceso consiste en dividir cada variable de nuestros datos entre su respectiva desviación estándar (véase la figura 2.5c). De este modo, si $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ son nuestros datos, donde $X_m = [x_{m1}, \ldots, x_{md}]$ para cada $m = 1, 2, \ldots, n$, nuestros nuevos datos $X_m^* = [x_{m1}^*, \ldots, x_{md}^*]$ son aquellos donde

$$x_{mj}^* = \frac{x_{mj}}{\sigma_j},\ j = 1, 2, \ldots, d.$$

Normalización Max

Este proceso de escalamiento consiste en dividir cada variable de la entrada por el valor máximo que alcanza (véase la figura 2.5d), esto es, si $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ son nuestros datos, donde $X_m = [x_{m1}, \ldots, x_{md}]$ para cada $m = 1, 2, \ldots, n$, nuestros nuevos datos $X_m^* = [x_{m1}^*, \ldots, x_{md}^*]$ son aquellos donde

$$x_{mj}^* = \frac{x_{mj}}{\max\limits_s \{x_{sj}\}},\ j = 1, 2, \ldots, d.$$

Normalización Minimax

El método de normalización Minimax consiste en restar a cada característica su respectivo valor mínimo y luego dividir dicho resultado entre la distancia máxima existente entre los valores de esa característica (véase la figura 2.5e). Es decir, si $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ son nuestros datos, donde $X_m = [x_{m1}, \ldots, x_{md}]$ para cada $m = 1, 2, \ldots, n$, nuestros nuevos datos $X_m^* = [x_{m1}^*, \ldots, x_{md}^*]$ son aquellos donde

$$x_{mj}^* = \frac{x_{mj} - \min\limits_s \{x_{sj}\}}{\max\limits_s \{x_{sj}\} - \min\limits_s \{x_{sj}\}},\ j = 1, 2, \ldots, d.$$

Una observación que hace Montesinos [24] acerca de este método, es que hace que las entradas con distribuciones uniformes dominen a las distribuciones con colas largas, esto es que el método no distingue muy bien entre valores atípicos. Para una comprensión más específica y visual puede consultarse [8].

Hay que mencionar que tanto los procesos de escalado, el de normalización max (suponiendo que todas nuestras mediciones son positivas) y el de normalización minimax

mantienen a nuestros datos positivos, en particular, normalización max coloca a todos los datos en el intervalo $[0, 1]$, (véase las figuras 2.5c, 2.5d y 2.5e) lo cual, en palabras de Izenman [17], no es requerido en redes neuronales y una mejor elección es centrar los datos alrededor del $\mathbf{0}$.

Estandarización

Este proceso de normalización consiste en centrar cada característica respecto a su media muestral y dividir con respecto a su desviación estándar (véase la figura 2.5f). Esto es, si $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ son nuestros datos, $\overline{X}$ es la media muestral y s es el vector de varianzas, donde $X_m = [x_{m1}, \ldots, x_{md}]$ para cada $m = 1, 2, \ldots, n$, $\overline{X} = [\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_d]$ y $s = [s_1^2, s_2^2, \ldots, s_d^2]$, nuestros datos transformados $X_m^* = [x_{m1}^*, \ldots, x_{md}^*]$ son aquellos tales que

$$x_{md}^* = \frac{x_{mj} - \overline{X}}{s_j},\ j = 1, 2, \ldots, d.$$

Usaremos este método de escalamiento para las aplicaciones que se presentarán en el capítulo 3.

2.5.2. Métodos de validación

Este apartado se retoma de lo mencionado en [17] y [24].

En los problemas de aprendizaje supervisado, una vez que se ha obtenido un modelo particular, resulta necesario evaluar la precisión con la que se ajusta a los datos. En este caso, usamos el error de predicción como medida de precisión de nuestras predicciones. De forma genérica, el error de predicción se calcula como la media de las diferencias de los errores al cuadrado de la predicción para el caso de regresión, aquí los errores son entre un valor de salida real y su correspondiente valor de salida predicho; mientras que para el caso de clasificación se calcula como la probabilidad de clasificar erróneamente un dato.

En la práctica, puesto que solo contamos con una cantidad limitada de datos, en ambos casos, lo primero es ajustar un modelo usando un conjunto de aprendizaje $\mathcal{L}$. Para el caso de regresión lo siguiente es, usando este modelo ajustado, predecir los valores de salida de $\mathcal{L}$ (los valores de entrada se toman en $\mathcal{L}$) o predecir los valores de salida de un conjunto de prueba $\mathcal{T}$ (los valores de entrada se toman en $\mathcal{T}$). El error de predicción es la media (calculada solo sobre el conjunto correspondiente) de los errores de predicción al cuadrado (en este caso el error es el valor de salida real menos el valor de salida predicho). Cuando la media se hace sobre $\mathcal{L}$, el error de predicción es llamado *error de aprendizaje de regresión*, mientras que si lo realizamos sobre el conjunto $\mathcal{T}$ se le llama *error de prueba de regresión*. Por otra parte, cuando realizamos un modelo de clasificación, el error de predicción es diferente, lo que se hace es predecir las clases de cada dato en $\mathcal{L}$ o $\mathcal{T}$. Si la predicción de la clase ha sido correcta, asignamos un 0 a ese dato, mientras que si ha sido incorrecta, le asignamos un 1, de tal modo que el error de

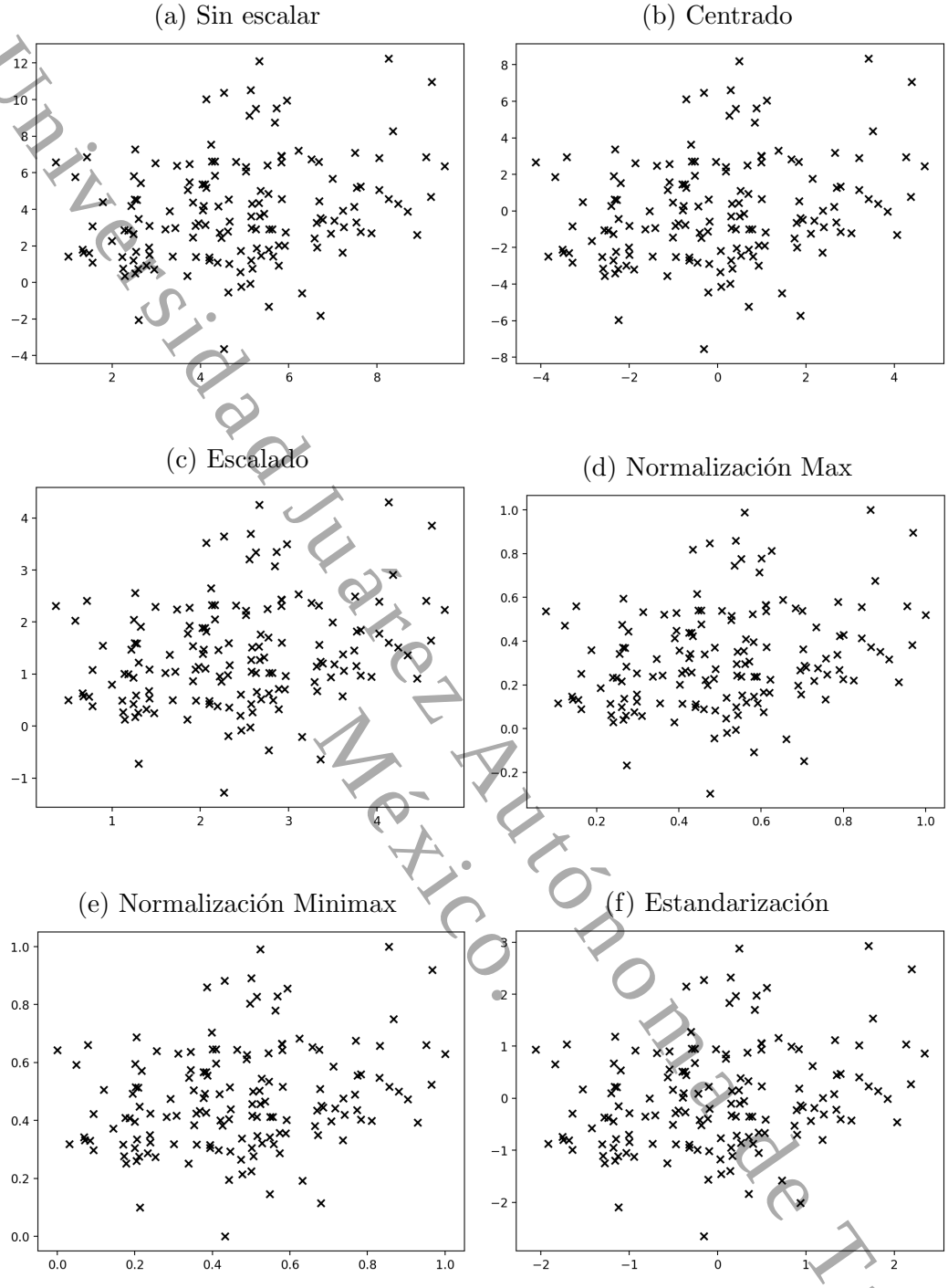


Figura 2.5: Métodos de escalamiento más comunes.

predicción se define como el promedio de todos estos 0's y 1's (esto es, la proporción de datos mal clasificados). Si nuestro promedio se realizó sobre las predicciones de los datos en $\mathcal{L}$, el error de predicción se denomina *error de aprendizaje de clasificación* mientras que si promediamos sobre los datos en $\mathcal{T}$ se le llama *error de prueba de clasificación.*

La estrategia de *validación cruzada* (CV por sus siglas en inglés) se usa para la selección de un modelo o de algún algoritmo. La técnica CV consiste en dividir los datos (al

menos una vez) para así estimar el error de predicción de cada algoritmo. Una vez que se han hecho las divisiones, parte de los datos son usados para el entrenamiento (el conjunto de entrenamiento $\mathcal{L}$) y el resto son empleados para estimar el error de predicción (los datos de prueba $\mathcal{T}$). Posteriormente, el CV elige el modelo con el menor error de predicción estimado. Por esta razón, la utilidad del CV radica en evaluar la predicción de un modelo con datos fuera de la muestra de entrenamiento. La técnica del CV garantiza que los datos utilizados para entrenar el modelo sean independientes de los que se usaron para evaluarlo. El CV evalúa la eficacia con la que el modelo ha adquirido nuevos datos que no han sido utilizados previamente para el entrenamiento.

Los resultados del CV dependen de como se ha hecho la división entre los conjuntos de entrenamiento y de prueba, en el caso particular de este trabajo estamos considerando la validación cruzada *K-Fold* y la *validación cruzada aleatoria*, las cuales presentamos a continuación.

K-Fold

En el proceso de validación cruzada *K-Fold* el conjunto de datos se divide en K subconjuntos disjuntos (grupos) de aproximadamente el mismo tamaño, la división es aleatoria. Uno de los K subconjuntos será tomado como el subconjunto de prueba y servirá para estimar el error de prueba de clasificación/regresión, por otro lado los $K-1$ subconjuntos restantes se usarán para el entrenamiento del modelo. El proceso se repite por cada uno de los K subconjuntos, haciendo así un total de K ajustes utilizando, cada uno, una partición distinta del subconjunto de prueba y los $K-1$ restantes para el entrenamiento. Finalmente se promedian las K estimaciones y este se considera como el rendimiento de la predicción del modelo. Este método, a coste de un mayor gasto computacional, es muy preciso porque combina K mediciones de los ajustes resultantes de los K subconjuntos de entrenamiento y prueba en los que fue dividido el conjunto de datos original. En la práctica la elección del número de divisiones depende del número de datos con los que contamos, siendo 5 o 10 divisiones las elecciones más comunes.

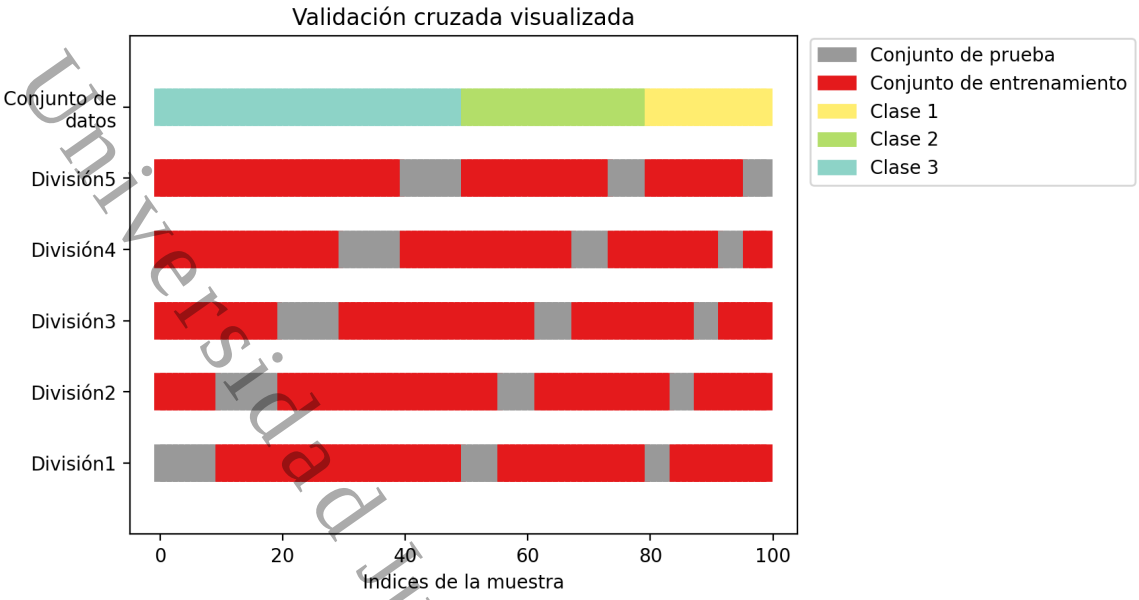


Figura 2.6: Representación de un 5-Fold.

Validación cruzada aleatoria

En este tipo de validación cruzada, el experimentador escoge el número de particiones a realizar (las divisiones independientes del conjunto de datos de entrenamiento y prueba), cuantas más particiones, mejor. La idea consiste en dividir aleatoriamente el conjunto de datos en dos subconjuntos, el subconjunto de datos de entrenamiento y el subconjunto de datos de prueba. El porcentaje de datos que serán usados para entrenamiento y para prueba son definidos previamente por el experimentador. Por ejemplo, el experimentador puede decidir que en cada partición aleatoria, el 80 % de los datos se usen para entrenar el modelo y el 20 % restante sean usados para prueba. La gran diferencia con el K-Fold es el hecho de que las particiones no deben ser mutuamente excluyentes; esto es, en el enfoque de *validación cruzada aleatoria*, una observación puede aparecer en más de una partición. Esto conlleva a que algunas muestras no puedan evaluarse, mientras que otras puedan hacerlo más de una vez, lo que significa que los subconjuntos de prueba y entrenamiento se sobrepongan. Para garantizar la aleatoriedad y garantizar la reproducción del experimento, es recomendable usar una semilla específica en el generador de números aleatorios. Una sugerencia común es usar al menos diez particiones aleatorias para obtener suficiente precisión en la estimación del rendimiento de la predicción.

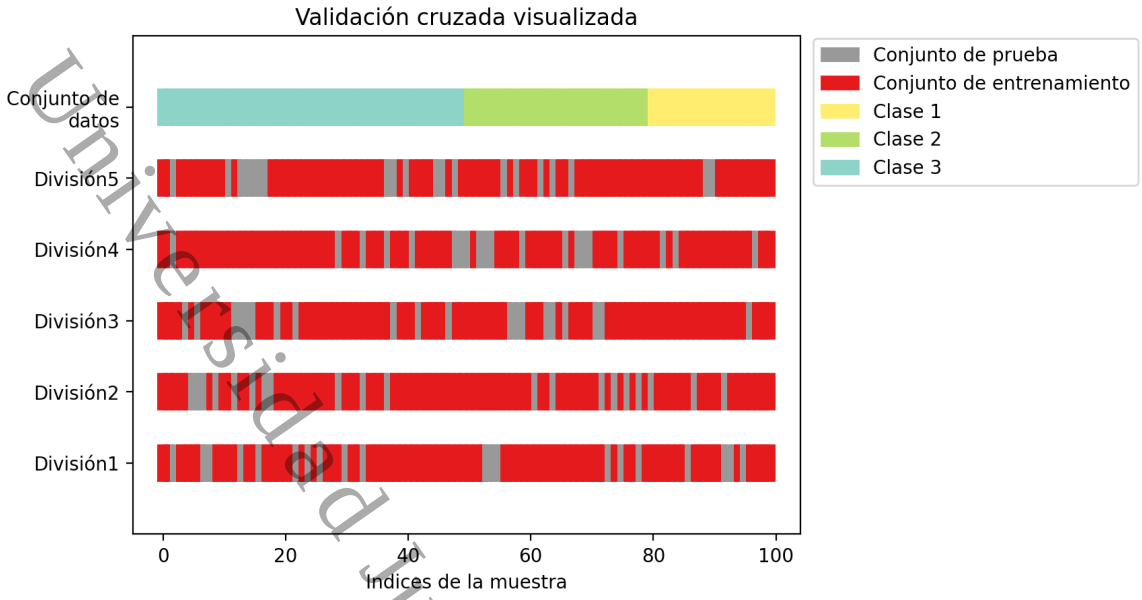


Figura 2.7: Representación de 5 divisiones hechas por una validación cruzada aleatoria.

2.6. Un ejemplo computacional del algoritmo Backpropagation

En el repositorio se encuentra un código elaborado con google colab (basado en python), el cual creamos para entender mejor el uso del algoritmo backpropagation en la construcción de una red neuronal con una capa oculta, nos basamos en las ideas presentadas en la sección 2.4.2 y aprovechamos las mencionadas en las secciones 1.5 y 2.1 para visualizar lo relacionado al teorema de aproximación universal. Este ejercicio viene de una muestra similar presentada en [5].

En efecto, la idea general de este ejemplo es considerar dos conjuntos de datos aleatorios en $\mathbb{R}^2$, cada uno con su respectiva distribución y con una clase preasignada, queremos visualizar la regla de decisión con base en la probabilidad a posteriori de alguna de ellas, en otras palabras construir el clasificador de la regla de Bayes. Decidimos usar una mezcla de gaussianas para la clase 0 y una normal multivariada para la clase 1, se generaron 10000 datos de entrenamiento (5000 para cada clase) y posteriormente generamos otros 10000 (de forma equitativa nuevamente) datos de prueba.

Tal como se comenta en la sección de la regla de Bayes, debido a que conocemos las distribuciones de los datos de cada clase, podemos calcular explícitamente la probabilidad a posteriori. La idea es comparar la región de decisión del clasificador regla de Bayes con la correspondiente a una red neuronal. Para esto, se dibuja la curva que corresponde a las probabilidades a posteriori iguales a $\frac{1}{2}$ para ambas reglas de decisión, y se observa donde caen los datos de prueba en el plano.

Realizamos dos casos, en el primer caso usamos la misma matriz de covarianza y en el segundo consideramos covarianzas distintas para las dos clases. En ambos casos, la

red neuronal constó de una sola capa oculta y tuvo por funciones de activación a la tangente hiperbólica en la capa oculta y el sigmoide logístico en la capa de salida. La clase 0 se generó en base a una mezcla de 3 gaussianas, así que

$$f_1(x) = \sum_{i=1}^{3} p_i f_{\mathcal{N},i}(x),$$

donde $p = (p_1, p_2, p_3) = (0.4, 0.2, 0.4)$ es el vector de pesos, $f_{\mathcal{N},i}(x)$ es la densidad de una variable aleatoria con distribución $\mathcal{N}_2(\mu_i, \Sigma)$ y se eligieron

$$\mu_1 = (-3, 0),\ \mu_2 = (2.5, 5),\ \mu_3 = (8, -0.5),\ \Sigma = \begin{bmatrix} 8 & 0.5 \\ 0.5 & 3 \end{bmatrix}.$$

Por otro lado, la clase 1 se generó a partir de una distribución gaussiana, por lo que $f_2(x)$ corresponde a la densidad de una normal bivariada, en este caso tomamos dicha normal como $\mathcal{N}_2((0, -6), \Sigma)$.

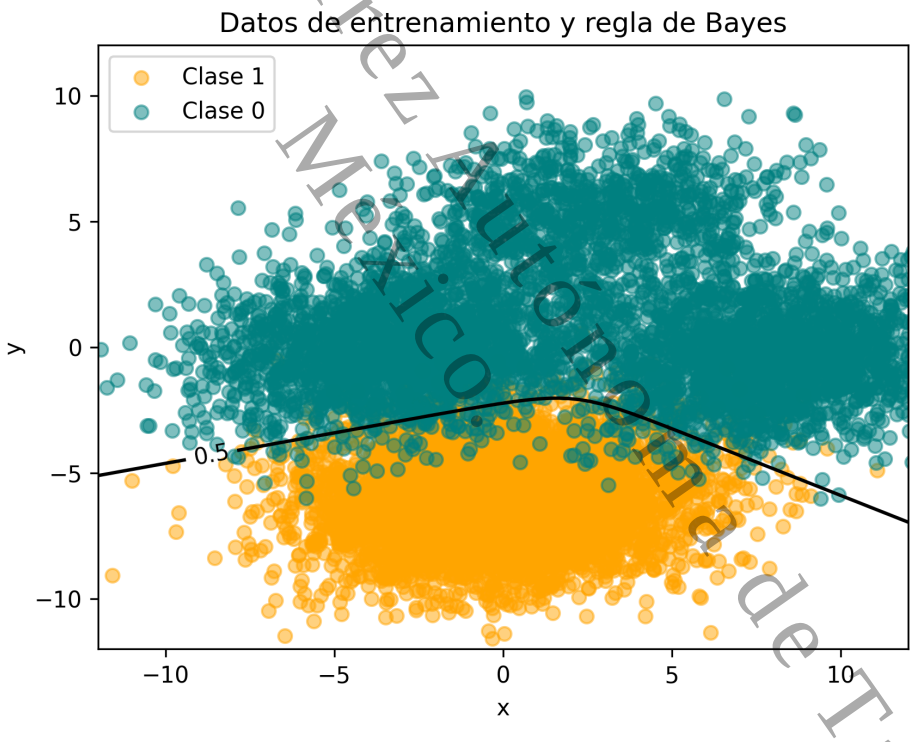


Figura 2.8: Conjuntos de datos y separación por la regla de Bayes: Caso de matrices de covarianza iguales.

Calculamos empíricamente el error de Bayes usando los datos de prueba (recordemos la complejidad de calcular el error de Bayes usando lo obtenido en la sección 1.5 y de esto mismo la necesidad de usar tantos datos). En este caso, el error empírico resultó en 0.029.

Variando el número de neuronas, ajustamos cada modelo y los resultados obtenidos se presentan en la tabla 2.2. La curva negra sólida corresponde a la separación de las clases

por el clasificador regla de Bayes y la curva roja punteada corresponde a la regla de decisión de la red neuronal. Se observa en la figura que conforme el número de neuronas aumenta, la proporción de error de clasificación de la red neuronal se aproxima al error de Bayes empírico.

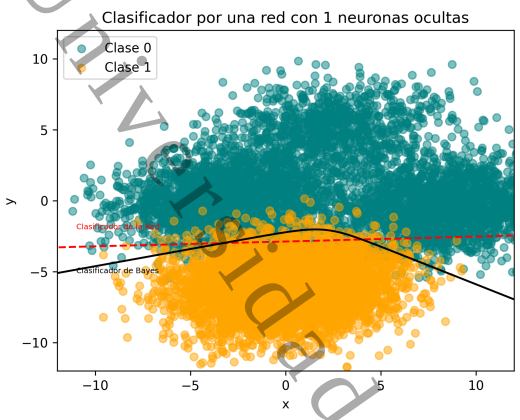
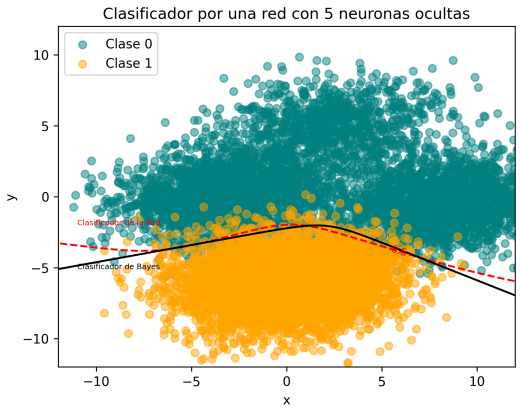
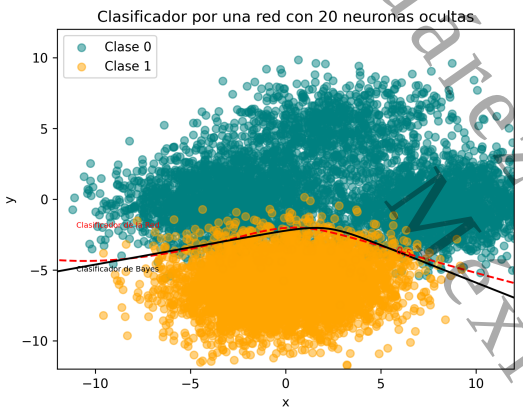
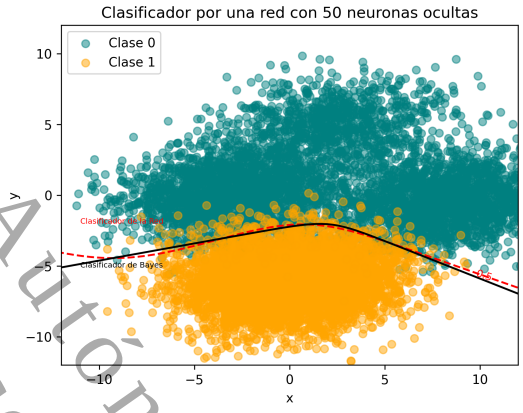
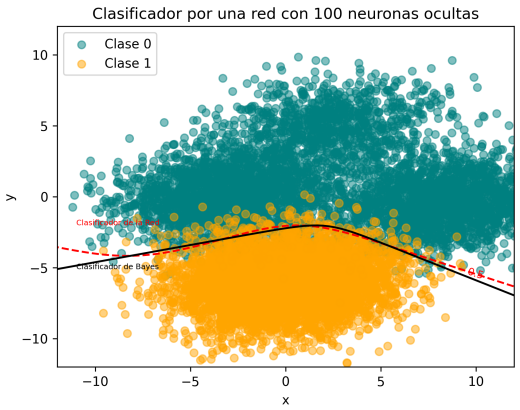
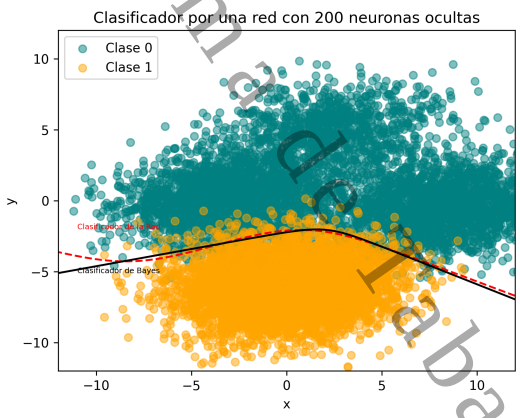
Comparación entre el clasificador de la regla de Bayes y el clasificador generado por una red neuronal	
 Error de clasificación: 0.0492.	 Error de clasificación: 0.031.
 Error de clasificación: 0.0306.	 Error de clasificación: 0.0299.
 Error de clasificación: 0.0299.	 Error de clasificación: 0.0299.

Tabla 2.2: Clasificador de Bayes vs clasificador por RNA para diversas cantidades de neuronas ocultas y dispersión de los datos de prueba (ejemplo 1).

Para el segundo caso donde las matrices de covarianza difieren, consideramos nuevamente una mezcla de gaussianas para la clase 0 y una normal multivariada para la clase

1. Repetimos nuevamente el ejercicio, pero usando ahora los siguientes parámetros

$$\mu_1 = (-3, 0),\ \mu_2 = (2.5, 5),\ \mu_3 = (3, -6),$$
$$\Sigma_1 = \begin{bmatrix} 6 & 0.3 \\ 0.3 & 5 \end{bmatrix},\ \Sigma_2 = \begin{bmatrix} 5 & 0.5 \\ 0.5 & 2 \end{bmatrix},\ \Sigma_3 = \begin{bmatrix} 4 & 0.5 \\ 0.5 & 2 \end{bmatrix},$$
$$p = (0.3, 0.4, 0.3).$$

Para la clase 1 elegimos ahora una normal bivariada con vector de medias $\mu = (7, -1)$ y matriz de covarianza $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.

En la figura 2.9 se presenta la dispersión de los datos de entrenamiento y la curva del clasificador de la regla de Bayes. Para los datos de prueba, el error empírico se estimó en 0.0106.

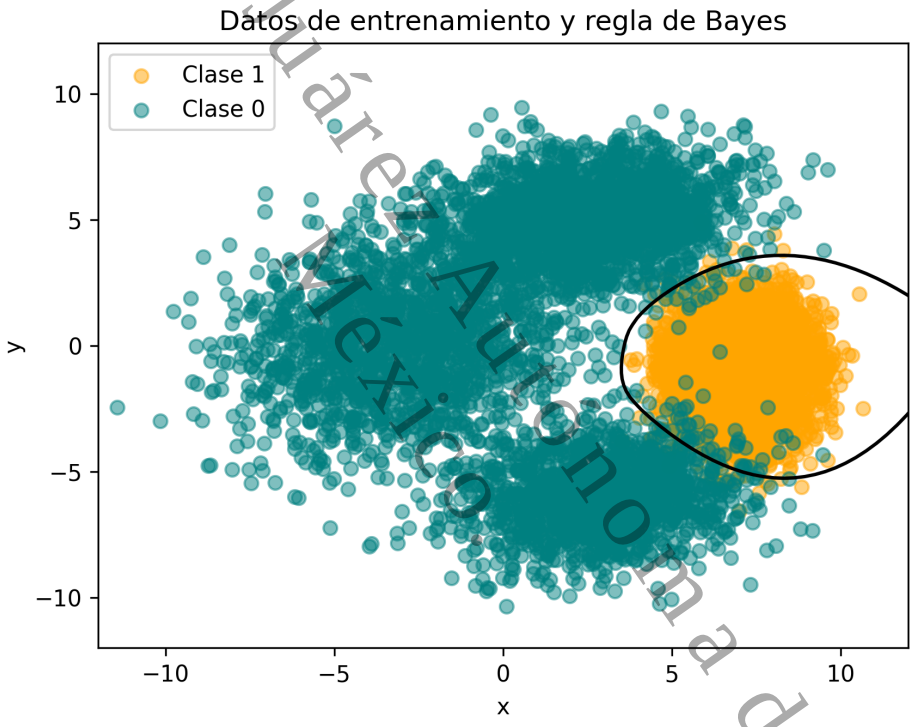


Figura 2.9: Conjuntos de datos y separación por la regla de Bayes: Caso de matrices de covarianza distintas.

Los resultados se presentan en la tabla 2.3.

En esta tabla podemos observar claramente que logramos aproximarnos de forma bastante cercana al error de Bayes aún sin tantas neuronas, esto pese a que no nos acercarnos completamente al clasificador de Bayes. Por lo que este ejemplo puede ayudarnos a observar como es que el teorema de aproximación universal funciona en la práctica.

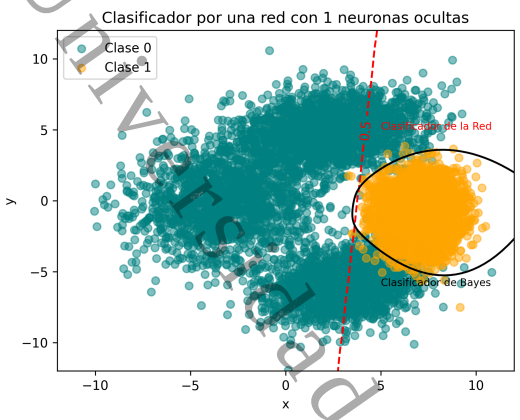
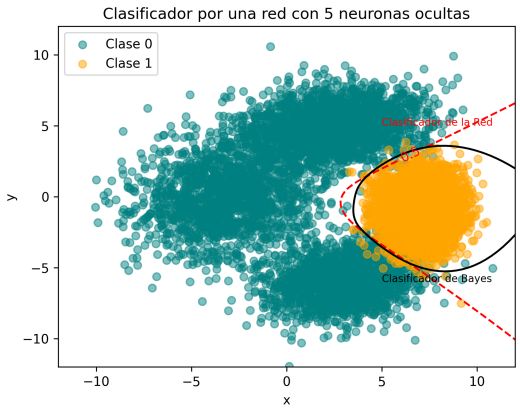
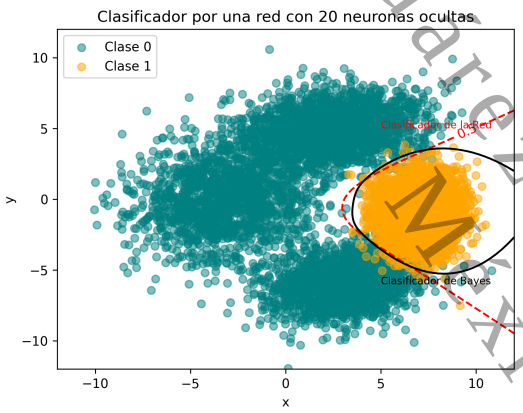
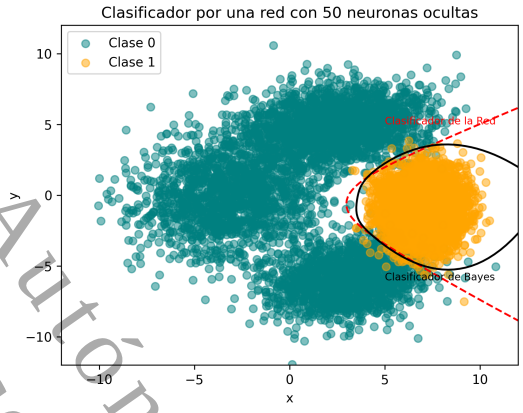
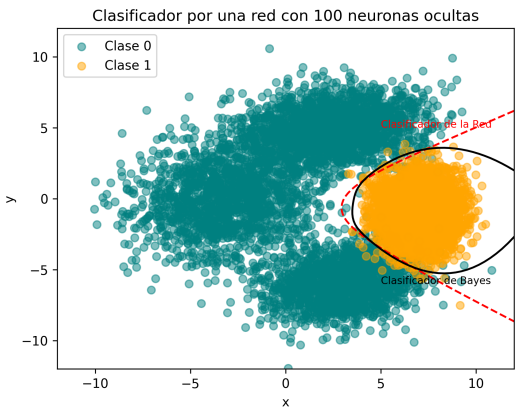
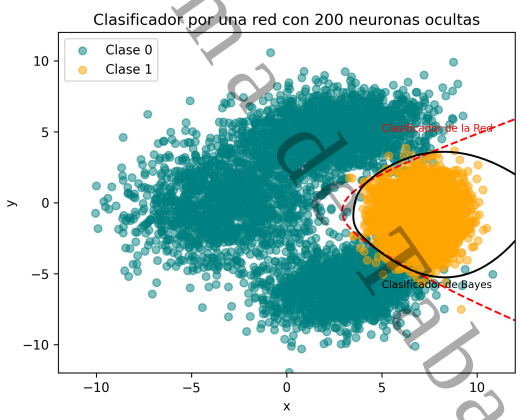
Comparación entre el clasificador de la regla de Bayes y el clasificador generado por una red neuronal	
 Error de clasificación: 0.1156.	 Error de clasificación: 0.0112.
 Error de clasificación: 0.011.	 Error de clasificación: 0.0108.
 Error de clasificación: 0.0107.	 Error de clasificación: 0.0106.

Tabla 2.3: Clasificador de Bayes vs clasificador por RNA para diversas cantidades de neuronas ocultas y dispersión de los datos de prueba (ejemplo 2).

2.7. Paqueterías en Python

Una de las ventajas de la programación en python la componen las diversas librerías especializadas como lo son *numpy*, *pandas*, *matplotlib*, entre otras. En este trabajo hemos considerado las mencionadas anteriormente para realizar el tratamiento de las bases de datos. A continuación damos una breve descripción de algunas librerías extras que también usamos en este trabajo.

2.7.1. Scikit-Learn

La biblioteca de *scikit-learn* con documentación oficial [31], es una paquetería especializada en aprendizaje máquina, haciendo el uso de herramientas simples y eficientes para análisis predictivo de datos. Scikit-Learn cuenta con herramientas para hacer clasificación, regresión y clustering, así como para hacer reducción de dimensión y selección de modelos. En este trabajo hacemos uso de esto último, pues de aquí tomaremos las herramientas de *K-Fold* y *validación cruzada aleatoria* para la validación del modelo, usaremos también algunas de las métricas que esta librería incluye como lo es la *matriz de confusión*.

2.7.2. TensorFlow

TensorFlow es una librería de código abierto desarrollada por Google para la creación de modelos de aprendizaje profundo (*Deep learning*). La documentación oficial de esta librería se puede consultar en [33]. Esta librería puede ser usada en conjunto con CUDA, paquetes creados por NVIDIA para el aprovechamiento del uso de GPU y procesos en paralelo. Esta librería puede trabajar con los objetos generados por numpy y pandas, sin embargo, es conveniente transformarlos a tensores. En parte de este trabajo aprovechamos dicha capacidad para acelerar el proceso, además, la facilidad con la que podemos crear manualmente redes más versátiles, puesto que TensorFlow permite editar con mayor libertad capa por capa sin mucha dificultad, siendo esta una de las razones por la que decidimos hacer uso de esta librería. Cabe destacar que esta paquetería es la principal herramienta detrás de la inteligencia artificial *Gemini* de Google.

Parte II

Resultados

Capítulo 3: Análisis de datos reales

3.1. Bases de datos médicos

En esta sección presentamos las bases de datos médicos que se han recopilado para cumplir con el objetivo principal de esta tesis. Explicaremos brevemente en cada caso la metodología a seguir y daremos una breve exposición de los datos. Los códigos generados para la simulación de cada caso se encuentran en el repositorio de GitHub `https://github.com/SaulDavid2025/Redes_Neuronales-Tesis_Maestria`.

Los códigos fueron generados en formato .ipynb para una mejor visualización. En este trabajo hemos decidido enfocarnos en cuatro ejemplos de aplicación, de los cuales los últimos dos corresponden a un mismo conjunto de datos:

(1) *Detección de cáncer de mama - Winsconsin, Estados Unidos.*

(2) *Predicción de supervivencia en pacientes con insuficiencia cardíaca - Punjab, Pakistán.*

(3) *Predicción de patrones morfológicos con cardiotogramas fetales procesados automáticamente.*

(4) *Predicción de estado fetal con cardiotocografía.*

3.1.1. Datos del cáncer de mama

En [17] se nos presenta un ejemplo del uso de clasificadores para la detección de tumores malignos. Entre los tipos de cáncer, en lo que respecta al género femenino, el de mama es el segundo con mayor número de decesos. Los métodos de detección actuales son tres: mamografía, aspiración con aguja fina (FNA por sus siglas en inglés) con interpretación visual; y biopsia quirúrgica. Es conocido que las biopsias son las más precisas, éstas son invasivas, costosas y requieren de mucho tiempo. En la década de los 90's se desarrolló en la universidad de Winsconsin-Madison un sistema de imágenes computarizadas, el cual tiene por objetivo desarrollar un procedimiento que diagnostique FNA's con alta precisión. El procedimiento consiste en tomar una muestra de líquido del bulto o masa mamaria (la cual ha sido detectada mediante autoexploración o mamografía) haciendo uso de una aguja de pequeño calibre (es decir, FNA). La FNA se coloca sobre un portaobjetos de vidrio y se pigmenta con la finalidad de resaltar los núcleos de las células constituyentes; la imagen de la FNA es transferida a una estación de trabajo usando una videocámara montada en un microscopio; y se determinan los límites exactos de los núcleos.

A partir de la muestra del líquido, diez características del núcleo de cada célula son calculadas (véase la tabla 3.1). Las variables se construyen de tal forma que una mayor probabilidad de malignidad sea representada por valores más altos. En cada imagen, la cual se compone de entre 10 y 40 núcleos, se calculan el valor medio (mv), el valor extremo (es decir, el valor más grande o peor, el tamaño más grande, la forma más

Tabla 3.1: Variables para el estudio del cáncer de mama de Wisconsin.

radius	Radio de un núcleo individual.
texture	Varianza de los niveles de grises dentro del límite del núcleo.
peri	Distancia alrededor del perímetro del núcleo.
area	Área del núcleo.
smooth	Suavidad del contorno de un núcleo medido por la variación local de segmentos radiales.
comp	Una medida de la compacidad de un núcleo celular usando la formula (**peri**)^2**/área**.
scav	Gravedad de las concavidades o hendiduras en un núcleo celular mediante una medición de tamaño que enfatiza las hendiduras pequeñas.
ncav	Número de puntos cóncavos o hendiduras en un núcleo celular.
symt	Simetría de un núcleo celular.
fracd	Dimensión fractal (del contorno) de una célula.

Tabla 3.2: Variables extraídas del estudio de cáncer de mama.

Valor medio		Desviación estándar		Forma más irregular	
(1)	radius.mv	(11)	radius.ev	(21)	radius.sd
(2)	texture.mv	(12)	texture.ev	(22)	texture.sd
(3)	peri.mv	(13)	peri.ev	(23)	peri.sd
(4)	area.mv	(14)	area.ev	(24)	area.sd
(5)	smooth.mv	(15)	smooth.ev	(25)	smooth.sd
(6)	comp.mv	(16)	comp.ev	(26)	comp.sd
(7)	scav.mv	(17)	scav.ev	(27)	scav.sd
(8)	ncav.mv	(18)	ncav.ev	(28)	ncav.sd
(9)	symt.mv	(19)	symt.ev	(29)	symt.sd
(10)	fracd.mv	(20)	fracd.ev	(30)	fracd.sd

irregular) (ev) y la desviación estándar (sd) de cada una de estas características celulares, lo cual nos deja con 30 variables reales. Las 30 variables se enlistan en la tabla 3.2.

El conjunto de datos consta de 569 casos (imágenes), de los cuales 212 están diagnosticados como malignos (confirmados por biopsia) y los 357 restantes benignos (confirmados por biopsia o por subsecuentes examinaciones médicas periódicas).

Los datos pueden ser obtenidos directamente del repositorio *kaggle* en formato .csv desde la liga `https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset`.

Para el análisis de esta base de datos hemos construido un modelo a partir de una red neuronal artificial con una capa oculta y 50 neuronas, para el entrenamiento se permitieron 20000 iteraciones del gradiente descendente con una tasa de aprendizaje de $\eta = 0.01$. Se hicieron análisis comparando distintas funciones de activación en la capa oculta y también analizando los datos con y sin estandarización. Los resultados presentados en las tablas 3.3 y 3.4 constituyen las estadísticas obtenidas de 10 validaciones cruzadas 5-Fold (50 modelos entrenados) y 50 validaciones cruzadas aleatorias (50 modelos, divididos en 80 % para entrenamiento y 20 % para prueba), el total de datos en las tablas se obtiene de multiplicar la cantidad de datos de prueba por la cantidad de iteraciones del método.

Antes de continuar debemos explicar que al usar los algoritmos K-Fold y la validación cruzada aleatoria (RCV) de scikit-learn, hay una pequeña diferencia, el K-Fold divide exactamente los datos en la cantidad de grupos sin necesidad de completar, al ir en orden (véase la imagen 2.6) no necesita añadir más de los mismos datos para obtener la división, solo divide la cantidad total de la forma más "exacta" posible, por ejemplo, supongamos que tenemos 19 elementos y quiero hacer un 5-Fold, 4 de las divisiones tendrían 4 elementos y la restante tendría 3; por otra parte, el RCV usa los mismos datos para completar y hacer la división aleatoria, en el caso de los 19 datos, hubiese completado a 20 y habría hecho la división equitativa de los datos.

Tabla 3.3: Resultados obtenidos con datos estandarizados y 20,000 iteraciones del GD usando redes neuronales.

K-Fold

Función de activación	**Tanh**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,040	80	2,120
Benignos	61	309	3,570
Total Fila	2,101	3,589	5,690

Proporción de error de clasificación: 0.025
Proporción de clasificación correcta: 97.5 %

Función de activación	**ReLU**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,016	104	2,120
Benignos	92	3,478	3,570
Total Fila	2,108	3,582	5,690

Proporción de error de clasificación: 0.034
Proporción de clasificación correcta: 96.6 %

Función de activación	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,063	57	2,120
Benignos	58	3,512	3,570
Total Fila	2,121	3,569	5,690

Proporción de error de clasificación: 0.020
Proporción de clasificación correcta: 98 %

CV aleatorio

Función de activación	**Tanh**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,050	91	2,141
Benignos	62	3,497	3,559
Total Fila	2,112	3,588	5,700

Proporción de error de clasificación: 0.027
Proporción de clasificación correcta: 97.3 %

Función de activación	**ReLU**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,021	120	2,141
Benignos	99	3,460	3,559
Total Fila	2,120	3,580	5,700

Proporción de error de clasificación: 0.038
Proporción de clasificación correcta: 96.2 %

Función de activación	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,065	76	2,141
Benignos	51	3,508	3,559
Total Fila	2,116	3,584	5,700

Proporción de error de clasificación: 0.022
Proporción de clasificación correcta: 97.8 %

Tabla 3.4: Resultados obtenidos con datos sin estandarizar y 20,000 iteraciones del gradiente descendente usando redes neuronales.

K-Fold

Función de activación	**Tanh**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,794	326	2,120
Benignos	129	3,441	3,570
Total Fila	1,923	3,767	5,690

Proporción de error de clasificación: 0.080
Proporción de clasificación correcta: 92 %

Función de activación	**ReLU**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,840	280	2,120
Benignos	175	3,395	3,570
Total Fila	2,015	3,675	5,690

Proporción de error de clasificación: 0.080
Proporción de clasificación correcta: 92 %

Función de activación	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,831	289	2,120
Benignos	191	3,379	3,570
Total Fila	2,022	3,668	5,690

Proporción de error de clasificación: 0.084
Proporción de clasificación correcta: 91.6 %

CV aleatorio

Función de activación	**Tanh**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,799	342	2,141
Benignos	150	3,409	3,559
Total Fila	1,949	3,751	5,700

Proporción de error de clasificación: 0.086
Proporción de clasificación correcta: 91.4 %

Función de activación	**ReLU**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,590	551	2,141
Benignos	165	3,394	3,559
Total Fila	1,755	3,945	5,700

Proporción de error de clasificación: 0.126
Proporción de clasificación correcta: 87.4 %

Función de activación	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,823	318	2,141
Benignos	159	3,400	3,559
Total Fila	1,982	3,718	5,700

Proporción de error de clasificación: 0.084
Proporción de clasificación correcta: 91.6 %

En estas tablas se observa que existe diferencia entre el uso de datos estandarizados y no estandarizados. La estandarización de los datos mejora la precisión de los modelos. En especial la red neuronal que tiene activación sigmoide mostró, con el uso de datos estandarizados, los mejores resultados en las validaciones K-Fold y aleatoria.

Posteriormente hicimos un análisis comparativo entre el tratamiento de estandarización y la no estandarización de los datos. Para observar las diferencias se realizó el proceso de entrenamiento a un menor número de iteraciones del gradiente descendente pero con el mismo número de neuronas y tamaño de paso. Las pruebas se hicieron para 500, 1,000, 5,000 y 10,000 iteraciones. Los resultados de la precisión (porcentaje de clasificación correcta) obtenidos de las mismas validaciones cruzadas (mismos 5-Fold y validaciones cruzadas aleatorias) del caso anterior, se presentan en las tablas 3.5 y 3.6 para los datos estandarizados y no estandarizados, respectivamente.

Tabla 3.5: Clasificación correcta de una red en distintas épocas con datos estandarizados.

K-Fold

Iteraciones del GD	Activación	Porcentaje de clasificación correcta
500	**Tanh**	97.6 %
	ReLU	96.5 %
	Sigmoide	95.7 %
1,000	**Tanh**	97.6 %
	ReLU	96.6 %
	Sigmoide	96.3 %
5,000	**Tanh**	97.6 %
	ReLU	96.4 %
	Sigmoide	97.5 %
10,000	**Tanh**	97.7 %
	ReLU	96.4 %
	Sigmoide	97.8 %

CV aleatorio

Iteraciones del GD	Activación	Porcentaje de clasificación correcta
500	**Tanh**	97.3 %
	ReLU	96.3 %
	Sigmoide	95.2 %
1,000	**Tanh**	97.3 %
	ReLU	96.4 %
	Sigmoide	96 %
5,000	**Tanh**	97.5 %
	ReLU	96.3 %
	Sigmoide	97.4 %
10,000	**Tanh**	97.4 %
	ReLU	96.1 %
	Sigmoide	97.7 %

Tabla 3.6: Clasificación correcta de una red en distintas épocas con datos no estandarizados.

K-Fold

Iteraciones del GD	Activación	Porcentaje de clasificación correcta
500	**Tanh**	87.2 %
	ReLU	79.2 %
	Sigmoide	78 %
1,000	**Tanh**	91.1 %
	ReLU	79 %
	Sigmoide	87.9 %
5,000	**Tanh**	91.8 %
	ReLU	87.1 %
	Sigmoide	90.8 %
10,000	**Tanh**	91.9 %
	ReLU	91.1 %
	Sigmoide	91.1 %

CV aleatorio

Iteraciones del GD	Activación	Porcentaje de clasificación correcta
500	**Tanh**	87.1 %
	ReLU	77.2 %
	Sigmoide	76.9 %
1,000	**Tanh**	90.1 %
	ReLU	76.9 %
	Sigmoide	87.7 %
5,000	**Tanh**	91 %
	ReLU	82.7 %
	Sigmoide	90.9 %
10,000	**Tanh**	91.1 %
	ReLU	85.9 %
	Sigmoide	91.2 %

De estas tablas se aprecia la mejora global que proporciona la estandarización de datos, pues nótese que con un número pequeño de iteraciones se obtienen resultados altamente confiables en comparación a si usáramos datos no estandarizados.

Para comparar la efectividad de las redes neuronales se emplearon metodologías bien conocidas para clasificación. Las herramientas escogidas fueron *máquina de vectores soporte* (SVM) con kernel lineal y no lineal y el método de regresión logística, métodos que ya se encuentran implementados en la paquetería *Scikit-learn* de python [25], [31]. De igual forma, presentamos en las tablas 3.7-3.10 los resultados obtenidos haciendo uso de datos estandarizados así como datos sin estandarizar, evaluando los métodos con las mismas validaciones cruzadas.

Para el caso del SVM no lineal hemos optado por tomar kernel sigmoide, gaussiano y polinomial, para el caso del polinomial se decidió hacer uso de grados 3, 4 y 5, término independiente 1, el mismo término independiente se usó para el kernel sigmoide;

en todos los casos el hiperparámetro de escala fue $\gamma = \frac{1}{\#Datos}$. Cabe mencionar que para el kernel polinomial no obtuvimos convergencia del método iterativo y por tanto no observamos resultados, esto paso con cada uno de los grados con los datos sin estandarizar. Sospechamos que dicho problema radica en el proceso interno del código, como el algoritmo está diseñado para resolver un problema de optimización, al darle la libertad de seguir iterando hasta que se rebase una cierta tolerancia, creemos que esta no se rebasó en un número considerable de iteraciones y por tanto no recibimos una respuesta del método.

Tabla 3.7: Resultados obtenidos con datos estandarizados usando métodos lineales.

K-Fold

Método	**SVM lineal**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,007	113	2,120
Benignos	59	3,511	3,570
Total Fila	2,066	3,624	5,690

Proporción de error de clasificación: 0.030
Proporción de clasificación correcta: 97 %

Método	**Regresión logística**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,025	95	2,120
Benignos	42	3,528	3,570
Total Fila	2,067	3,623	5,690

Proporción de error de clasificación: 0.024
Proporción de clasificación correcta: 97.6 %

CV aleatorio

Método	**SVM lineal**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,028	113	2,141
Benignos	56	3,503	3,559
Total Fila	2,084	3,616	5,700

Proporción de error de clasificación: 0.030
Proporción de clasificación correcta: 97 %

Método	**Regresión logística**		
	Predicciones Benigno	Predicciones maligno	Total
Benignos	2,040	101	2,141
Malignos	37	3,522	3,559
Total Fila	2,077	3,623	5,700

Proporción de error de clasificación: 0.024
Proporción de clasificación correcta: 97.6 %

Tabla 3.8: Resultados obtenidos con datos sin estandarizar usando métodos lineales.

K-Fold

Método	**SVM lineal**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,944	176	2,120
Benignos	91	3,479	3,570
Total Fila	2,035	3,655	5,690

Proporción de error de clasificación: 0.047
Proporción de clasificación correcta: 95.3 %

Método	**Regresión logística**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,947	173	2,120
Benignos	104	3,466	3,570
Total Fila	2,051	3,639	5,690

Proporción de error de clasificación: 0.049
Proporción de clasificación correcta: 95.1 %

CV aleatorio

Método	**SVM lineal**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,961	180	2,141
Benignos	95	3,464	3,559
Total Fila	2,056	3,644	5,700

Proporción de error de clasificación: 0.048
Proporción de clasificación correcta: 95.2 %

Método	**Regresión logística**		
	Predicciones Benigno	Predicciones maligno	Total
Benignos	1,965	176	2,141
Malignos	106	3,453	3,559
Total Fila	2,071	3,629	5,700

Proporción de error de clasificación: 0.049
Proporción de clasificación correcta: 95.1 %

Tabla 3.9: Resultados obtenidos con datos estandarizados usando SVM no lineal.

K-Fold

Kernel	**Polinomial**		
Grado	3		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,025	95	2,120
Benignos	23	3,547	3,570
Total Fila	2,048	3,642	5,690

Proporción de error de clasificación: 0.021
Proporción de clasificación correcta: 97.9 %

Kernel	**Polinomial**		
Grado	4		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,026	94	2,120
Benignos	35	3,535	3,570
Total Fila	2,061	3,629	5,690

Proporción de error de clasificación: 0.023
Proporción de clasificación correcta: 97.7 %

Kernel	**Polinomial**		
Grado	5		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,023	97	2,120
Benignos	63	3,507	3,570
Total Fila	2,086	3,604	5,690

Proporción de error de clasificación: 0.028
Proporción de clasificación correcta: 97.2 %

Kernel	**Gaussiano**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,032	88	2,120
Benignos	54	3,516	3,570
Total Fila	2,086	3,604	5,690

Proporción de error de clasificación: 0.025
Proporción de clasificación correcta: 97.5 %

Kernel	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,830	290	2,120
Benignos	190	3,380	3,570
Total Fila	2,020	3,670	5,690

Proporción de error de clasificación: 0.084
Proporción de clasificación correcta: 91.6 %

CV aleatorio

Kernel	**Polinomial**		
Grado	**3**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,033	108	2,141
Benignos	21	3,538	3,559
Total Fila	2,054	3,646	5,700

Proporción de error de clasificación: 0.023
Proporción de clasificación correcta: 97.7 %

Kernel	**Polinomial**		
Grado	4		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,037	104	2,141
Benignos	33	3,526	3,559
Total Fila	2,070	3,630	5,700

Proporción de error de clasificación: 0.024
Proporción de clasificación correcta: 97.6 %

Kernel	**Polinomial**		
Grado	5		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,033	108	2,141
Benignos	59	3,500	3,559
Total Fila	2,092	3,608	5,700

Proporción de error de clasificación: 0.029
Proporción de clasificación correcta: 97.1 %

Kernel	**Gaussiano**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	2,041	100	2,141
Benignos	48	3,511	3,559
Total Fila	2,089	3,611	5,700

Proporción de error de clasificación: 0.026
Proporción de clasificación correcta: 97.4 %

Kernel	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	1,839	302	2,141
Benignos	209	3,350	3,559
Total Fila	2,048	3,652	5,700

Proporción de error de clasificación: 0.090
Proporción de clasificación correcta: 91 %

Tabla 3.10: Resultados obtenidos con datos sin estandarizar usando SVM no lineal.

K-Fold

Kernel	**Gaussiano**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	0	2,120	2,120
Benignos	0	3,570	3,570
Total Fila	0	5,690	5,690

Proporción de error de clasificación: 0.373
Proporción de clasificación correcta: 62.7 %

Kernel	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	0	2,120	2,120
Benignos	0	3,570	3,570
Total Fila	0	5,690	5,690

Proporción de error de clasificación: 0.373
Proporción de clasificación correcta: 62.7 %

CV aleatorio

Kernel	**Gaussiano**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	0	2,141	2,141
Benignos	0	3,559	3,559
Total Fila	0	5,700	5,700

Proporción de error de clasificación: 0.376
Proporción de clasificación correcta: 62.4 %

Kernel	**Sigmoide**		
	Predicciones maligno	Predicciones benigno	Total
Malignos	0	2,141	2,141
Benignos	0	3,559	3,559
Total Fila	0	5,700	5,700

Proporción de error de clasificación: 0.376
Proporción de clasificación correcta: 62.4 %

Al igual que en el caso de las redes, observamos en esta tabla la mejora proporcionada por los datos estandarizados, dicha mejora es más notoria con los métodos no lineales. La mejora en lo que respecta a los métodos lineales no resultó tan significativa como en la de las redes.

Finalmente, se ajustó un modelo con 2 capas ocultas y con función de activación tangente hiperbólico en ambas capas ocultas, se utilizaron 40 neuronas en la primera capa oculta y 30 neuronas en la segunda, se permitió un entrenamiento del modelo haciendo 40000 iteraciones del gradiente descendente y se uso una tasa de aprendizaje $\eta = 0.001$. Los resultados se muestran en las tablas 3.11 para los datos estandarizados y en las tablas 3.12 para los datos no estandarizados.

Tabla 3.11: Resultados obtenidos con datos estandarizados usando una red neuronal con 2 capas ocultas.

K-Fold

	Predicciones maligno	Predicciones benigno	Total
Malignos	2,030	90	2,120
Benignos	48	3,522	3,570
Total Fila	2,078	3,612	5,690

Proporción de error de clasificación: 0.024
Proporción de clasificación correcta: 97.6 %

CV aleatorio

	Predicciones maligno	Predicciones benigno	Total
Malignos	2,042	99	2,141
Benignos	40	3,519	3,559
Total Fila	2,082	3,618	5,700

Proporción de error de clasificación: 0.024
Proporción de clasificación correcta: 97.6 %

Tabla 3.12: Resultados obtenidos con datos sin estandarizar usando una red neuronal con 2 capas ocultas.

K-Fold

	Predicciones maligno	Predicciones benigno	Total
Malignos	1,806	314	2,120
Benignos	112	3,458	3,570
Total Fila	1,918	3,772	5,690

Proporción de error de clasificación: 0.076
Proporción de clasificación correcta: 92.4 %

CV aleatorio

	Predicciones maligno	Predicciones benigno	Total
Malignos	1,819	322	2,141
Benignos	127	3,432	3,559
Total Fila	1,946	3,754	5,700

Proporción de error de clasificación: 0.079
Proporción de clasificación correcta: 92.1 %

De esta tabla podemos concluir que aunque los resultados son buenos, se puede notar que no fueron muy superiores a los obtenidos con una sola capa oculta, esto nos muestra una consecuencia del teorema de aproximación universal, el cual nos garantiza la existencia de una red neuronal lo suficientemente confiable (aquella que alcanza el error de Bayes).

Finalmente, presentamos en estas últimas tablas, un resumen comparando los porcentajes de clasificación correcta de cada modelo con los datos estandarizados, los cuales estaremos trabajando de ahora en adelante en los siguientes ejemplos.

Tabla 3.13: Comparación de los porcentajes de clasificación correcta en los distintos modelos presentados usando datos estandarizados.

K-Fold

Modelo	Porcentaje de clasificación correcta
Red Neuronal 1 capa-Tanh	97.52 %
Red Neuronal 1 capa-ReLU	96.55 %
Red Neuronal 1 capa-Sigmoide	97.97 %
Red Neuronal 2 capas-Tanh	97.1 %
SVM lineal	96.97 %
Regresión Logística	97.59 %
SVM Polinomial-Grado 3	97.92 %
SVM Polinomial-Grado 4	97.73 %
SVM Polinomial-Grado 5	97.18 %
SVM Gaussiano	97.5 %
SVM Sigmoide	91.56 %

CV aleatorio

Modelo	Porcentaje de clasificación correcta
Red Neuronal 1 capa-Tanh	97.31 %
Red Neuronal 1 capa-ReLU	96.15 %
Red Neuronal 1 capa-Sigmoide	97.77 %
Red Neuronal 2 capas-Tanh	97.1 %
SVM lineal	97.03 %
Regresión Logística	97.57 %
SVM Polinomial-Grado 3	97.73 %
SVM Polinomial-Grado 4	97.59 %
SVM Polinomial-Grado 5	97.07 %
SVM Gaussiano	97.4 %
SVM Sigmoide	91.03 %

Concluimos de esta tabla que las redes neuronales tuvieron los mejores resultados con respecto al uso de datos estandarizados, esto en cuanto a precisión se refiere, destacando la eficiencia de la red neuronal con función de activación sigmoide.

Tabla 3.14: Comparación de los porcentajes de clasificación correcta en los distintos modelos presentados usando datos no estandarizados.

K-Fold

Modelo	Porcentaje de clasificación correcta
Red Neuronal 1 capa-Tanh	92 %
Red Neuronal 1 capa-ReLU	92 %
Red Neuronal 1 capa-Sigmoide	91.4 %
Red Neuronal 2 capas-Tanh	86.22 %
SVM lineal	95.3 %
Regresión Logística	95.11 %
SVM Polinomial-Grado 3	– –
SVM Polinomial-Grado 4	– –
SVM Polinomial-Grado 5	– –
SVM Gaussiano	62.74 %
SVM Sigmoide	62.74 %

CV aleatorio

Modelo	Porcentaje de clasificación correcta
Red Neuronal 1 capa-Tanh	91.36 %
Red Neuronal 1 capa-ReLU	87.43 %
Red Neuronal 1 capa-Sigmoide	91.63 %
Red Neuronal 2 capas-Tanh	85.82 %
SVM lineal	95.17 %
Regresión Logística	95.05 %
SVM Polinomial-Grado 3	– –
SVM Polinomial-Grado 4	– –
SVM Polinomial-Grado 5	– –
SVM Gaussiano	62.43 %
SVM Sigmoide	62.43 %

Con respecto a esta tabla debemos destacar la estabilidad de los métodos lineales, como la regresión logística y el SVM lineal, pues mantuvieron buenos resultados con los datos no estandarizados. Esta tabla nos ayuda a notar el contraste entre el uso de datos estandarizados y datos no estandarizados, además de resumir la información obtenida acerca de la precisión de cada uno de los métodos revisados.

En resumen, podemos observar la efectividad de las redes neuronales, las cuales tuvieron buen desempeño en los casos con estandarización como no estandarización, aunque en este último no fueron los mejores. Vemos además como el uso de datos estandarizados mejora significativamente el desempeño del modelo y reduce el entrenamiento de la red neuronal, pues desde las 500 iteraciones se observaron buenos resultados para las 3 funciones de activación cuando se usaron datos estandarizados. Destacamos además la eficiencia de la red neuronal con activación sigmoide, la cual fue la que mejor resultados mostró para este caso.

3.1.2. Datos de insuficiencia cardíaca

Cada año las enfermedades cardiovasculares cobran la vida de aproximadamente 17 millones de personas en el mundo, manifestándose principalmente como infartos de miocardio e insuficiencia cardíaca. La insuficiencia cardíaca (IC) ocurre cuando el corazón no consigue bombear suficiente sangre para satisfacer las necesidades del organismo. Debido a lo imprescindible que es el corazón humano, predecir la insuficiencia cardíaca es un objetivo prioritario para los médicos y especialistas, pero hasta la fecha, la predicción de esta clase de eventos no suelen alcanzar una alta precisión en la práctica clínica. Sin embargo, como se menciona en [7], los historiales médicos electrónicos de los pacientes (las historias clínicas electrónicas) cuantifican los síntomas, las características corporales y los valores de las pruebas clínicas, los cuales en conjunto pueden ser usados en un análisis bioestadístico con el fin de lograr resaltar patrones y correlaciones que de otra manera serían indetectables por los médicos.

Los métodos de aprendizaje automático, implementados con la información recolectada en los historiales médicos, resultan ser herramientas eficaces tanto para predecir la supervivencia de pacientes con síntomas de insuficiencia cardíaca, así como para detectar las características clínicas más importantes (o factores de riesgo) ligadas al desarrollo de la insuficiencia cardíaca. Para un contexto más profundo de la aplicación del aprendizaje automático en esta clase de enfermedades puede consultarse [7].

En este caso se realiza el análisis de una base de datos que consta de los registros médicos de 299 pacientes con insuficiencia cardíaca, estos han sido recolectados en el Instituto de Cardiología Faisalabad y en el Hospital Allied en Faisalabad (Punjab, Pakistán), durante abril a diciembre de 2015. La base de datos se encuentra dividida entre la información de 105 mujeres y 194 hombres, con edades variando entre 40 y 95 años. Todos los pacientes registrados en la base han padecido disfunción sistólica ventricular izquierda, así como antecedentes de insuficiencia cardíaca que, de acuerdo a la clasificación de la Asociación del Corazón de Nueva York (NYHA por sus siglas en inglés), entran en clase III o IV. Este conjunto de datos consta de 12 características, las cuales reportan información clínica, del cuerpo y del estilo de vida de los pacientes, éstas se describen brevemente en la tabla 3.15.

Tabla 3.15: Variables consideradas para los reportes médicos.

Age	Edad del paciente.
Anaemia	Disminución de glóbulos rojos o hemoglobina.
High blood pressure	Si un paciente padece de hipertensión.
Creatinine phosphokinase (CPK)	Nivel de creatinina fosfocinasa.
Diabetes	Si el paciente tiene diabetes.
Ejection fraction	Porcentaje de sangre que abandona el corazón en cada contracción.
Sex	Hombre o mujer.
Platelets	Plaquetas en la sangre.
Serium creatinine	Nivel de creatinina en la sangre.
Serium sodium	Nivel de sodio en la sangre.
Smoking	Si el paciente fuma.
Time	Periodo de seguimiento.

Estas características se usan para predecir con efectividad el deceso o supervivencia de pacientes que han padecido insuficiencia cardíaca. Los datos fueron tomados del repositorio de machine learning UCI desde la siguiente liga: `https://archive.ics.uci.edu/dataset/519/heart+failure+clinical+records`

Se ajustaron redes neuronales con el fin de realizar un modelo de clasificación adecuado a este problema. De una forma similar al ejemplo anterior, analizamos el comportamiento de distintas funciones de activación, en este trabajo presentamos los mejores 3 modelos obtenidos de forma empírica tras analizar varias simulaciones con distintos números de neuronas e iteraciones del gradiente descendente, así como una prueba más general del modelo de redes neuronales donde no tenemos tanto control de los hiperparámetros y condiciones iniciales, algo más cerca a la realidad.

Elección empírica de hiperparámetros

Los resultados que se presentan se obtuvieron usando datos estandarizados. Hacemos una comparación de nuestros resultados con los presentados en la tabla 4 de la referencia [7], tabla donde registran ciertas métricas obtenidas de distintos métodos de clasificación binaria, dichas métricas son:

- *Sensibilidad o recuperación* (**TP rate**)
- *Especificidad* (**TN rate**)
- *Precisión* (**Accuracy**)
- *Media armónica de la precisión y recuperación* (F_1 **Score**)
- *Coeficiente de correlación de Matthews* (**MCC**)
- *Área bajo la curva Característica Operativa del Receptor* (**ROC AUC**)
- *Área bajo la curva Precisión-Recuperación* (**PR AUC**)

Para la función de activación tangente hiperbólico el mejor resultado observado fue con 50 neuronas, una tasa de aprendizaje de 0.005 y 1950 iteraciones del gradiente descendente (entrenamiento sin batch). Las matrices de confusión después de 100 simulaciones de validación cruzada aleatoria y 20 validaciones 5-Fold (100 modelos) son presentadas en la tabla 3.16.

Cabe mencionar, nuevamente que las tablas que resumen los resultados registran como total de datos a la cantidad de validaciones cruzadas por la cantidad de datos de validación, también recalcamos que la validación K-Fold hace una división exacta, entonces por cada validación, estamos barriendo todos los datos, mientras que la validación cruzada aleatoria añade de los mismos datos para realizar la división exacta en cada validación, es por ello que notamos más datos en el CV aleatorio con respecto al K-Fold.

Tabla 3.16: Matriz de confusión usando una red neuronal con función de activación tangente hiperbólica.

K-Fold

	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,647	413	4,060
Muerte	612	1,308	1,920
Total Fila	4,259	1,721	5,980

Proporción de error de clasificación: 0.171
Proporción de clasificación correcta: 82.9 %

CV aleatorio

	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,634	462	4,096
Muerte	607	1,297	1,904
Total Fila	4,241	1,759	6,000

Proporción de error de clasificación: 0.178
Proporción de clasificación correcta: 82.2 %

Para la función de activación ReLU el mejor resultado observado fue con 140 neuronas, una tasa de aprendizaje de 0.0125 y 1,050 iteraciones del gradiente descendente (entrenamiento sin batch). Las matrices de confusión después de 100 simulaciones de validación cruzada aleatoria y 20 validaciones 5-Fold (100 modelos) son presentadas en la tabla 3.17.

Tabla 3.17: Matriz de confusión usando una red neuronal con función de activación ReLU.

K-Fold

	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,628	432	4,060
Muerte	652	1,268	1,920
Total Fila	4,280	1,700	5,980

Proporción de error de clasificación: 0.181
Proporción de clasificación correcta: 81.9 %

CV aleatorio

	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,622	474	4,096
Muerte	645	1,259	1,904
Total Fila	4,267	1,733	6,000

Proporción de error de clasificación: 0.186
Proporción de clasificación correcta: 81.4 %

Para la función de activación sigmoide el mejor resultado observado fue con 40 neuronas, una tasa de aprendizaje de 0.1 y 500 iteraciones del gradiente descendente (entrenamiento sin batch). Las matrices de confusión después de 100 simulaciones de validación cruzada aleatoria y 20 validaciones 5-Fold (100 modelos) son presentadas en la tabla 3.18.

Tabla 3.18: Matriz de confusión usando una red neuronal con función de activación sigmoide.

K-Fold

	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,666	394	4,060
Muerte	659	1,261	1,920
Total Fila	4,325	1,655	5,980

Proporción de error de clasificación: 0.176
Proporción de clasificación correcta: 82.4 %

CV aleatorio

	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,662	434	4,096
Muerte	656	1,248	1,904
Total Fila	4,318	1,682	6,000

Proporción de error de clasificación: 0.182
Proporción de clasificación correcta: 81.8 %

Con el propósito de comparar la efectividad, presentamos las matrices de confusión de métodos lineales (máquinas de vectores soporte y regresión logística), así como no lineales (máquinas de vectores soporte kernel). Las matrices de confusión para los métodos lineales se presentan en la tabla 3.19.

Tabla 3.19: Matrices de confusión usando métodos lineales.

K-Fold

Método	**SVM lineal**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,606	454	4,060
Muerte	637	1,283	1,920
Total Fila	4,243	1,737	5,980

Proporción de error de clasificación: 0.182
Proporción de clasificación correcta: 81.8 %

Método	**Regresión logística**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,643	417	4,060
Muerte	636	1,284	1,920
Total Fila	4,279	1,701	5,980

Proporción de error de clasificación: 0.176
Proporción de clasificación correcta: 82.4 %

CV aleatorio

Método	**SVM lineal**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,610	486	4,096
Muerte	626	1,278	1,904
Total Fila	4,236	1,764	6,000

Proporción de error de clasificación: 0.185
Proporción de clasificación correcta: 81.5 %

Método	**Regresión logística**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,629	467	4,096
Muerte	621	1,283	1,904
Total Fila	4,250	1,750	6,000

Proporción de error de clasificación: 0.181
Proporción de clasificación correcta: 81.9 %

De igual forma, realizamos comparación con el método no lineal usando diferentes kernels. Los mejores resultados se obtuvieron con estos hiperparámetros: grado 3 para el kernel polinomial así como valor independiente 1; valor independiente 0.001 para el kernel sigmoide, en todos los casos el parámetro de escala fue $\gamma = \frac{1}{\#Datos}$. Las matrices de confusión se presentan en la tabla 3.20.

Tabla 3.20: Resultados obtenidos con datos estandarizados usando SVM no lineal.

K-Fold

Kernel	**Polinomial grado 3**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,471	589	4,060
Muerte	814	1,106	1,920
Total Fila	4,285	1,695	5,980

Proporción de error de clasificación: 0.235
Proporción de clasificación correcta: 76.5 %

Kernel	**Gaussiano**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,618	442	4,060
Muerte	764	1,156	1,920
Total Fila	4,382	1,598	5,980

Proporción de error de clasificación: 0.202
Proporción de clasificación correcta: 79.8 %

Kernel	**Sigmoide**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,628	432	4,060
Muerte	656	1,264	1,920
Total Fila	4,284	1,696	5,980

Proporción de error de clasificación: 0.182
Proporción de clasificación correcta: 81.8 %

CV aleatorio

Kernel	**Polinomial grado 3**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,485	611	4,096
Muerte	820	1,084	1,904
Total Fila	4,305	1,695	6,000

Proporción de error de clasificación: 0.239
Proporción de clasificación correcta: 76.1 %

Kernel	**Gaussiano**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,635	461	4,096
Muerte	752	1,152	1,904
Total Fila	4,387	1,613	6,000

Proporción de error de clasificación: 0.202
Proporción de clasificación correcta: 79.8 %

Kernel	**Sigmoide**		
	Predicción supervivencia	Predicción muerte	Total
Supervivencia	3,633	463	4,096
Muerte	651	1,253	1,904
Total Fila	4,284	1,716	6,000

Proporción de error de clasificación: 0.186
Proporción de clasificación correcta: 81.4 %

En conjunto con las matrices de confusión, el código realizado nos muestra la media de las 100 simulaciones (tanto para los 20 5-Fold como las 100 simulaciones del CV aleatorio) de las métricas anteriormente mencionadas. Por tanto, se presenta a continuación un análogo a la tabla 4 de [7].

Tabla 3.21: Resultados de la predicción de supervivencia sobre todas las características clínicas usando los datos estandarizados (elección empírica).

Media de los 5-Fold

Método	MCC	F_1 Score	Accuracy	TP rate	TN rate	PR AUC	ROC AUC
NN ReLU	+0.5719	0.8692	0.8187	0.8941	0.6581	0.7571	0.8626
NN tanh	**+0.597**	**0.8757**	**0.8286**	0.8981	**0.6802**	0.7657	0.8671
NN sigmoide	+0.584	0.8734	0.8239	**0.9032**	0.6573	**0.7663**	**0.8708**
Regresión log.	+0.5859	0.8728	0.8239	0.8977	0.6674	0.7658	0.8677
SVM lineal	+0.5723	0.8675	0.8175	0.8885	0.6667	0.7596	0.8646
SVM polinomial	+0.4492	0.8305	0.7654	0.8551	0.5789	0.6538	0.7965
SVM sigmoide	+0.5739	0.8689	0.8184	0.8954	0.6582	0.7615	0.8691
SVM gaussiano	+0.5229	0.8561	0.7983	0.8918	0.6032	0.7283	0.8530

Media de los CV aleatorio

Método	MCC	F_1 Score	Accuracy	TP rate	TN rate	PR AUC	ROC AUC
NN ReLU	+0.5585	0.8655	0.8134	0.885	0.6591	0.7435	0.8547
NN tanh	**+0.5784**	**0.871**	**0.8218**	0.8875	**0.6784**	0.7479	0.8585
NN sigmoide	+0.5687	0.8697	0.8183	**0.8947**	0.6548	0.7547	**0.8648**
Regresión log.	+0.5713	0.8691	0.8186	0.8867	0.6715	**0.7552**	0.8624
SVM lineal	+0.5617	0.8659	0.8146	0.8819	0.6676	0.7444	0.8579
SVM polinomial	+0.434	0.8281	0.7615	0.8506	0.569	0.6407	0.7905
SVM sigmoide	+0.5601	0.8663	0.8143	0.8876	0.6558	0.7498	0.8608
SVM gaussiano	+0.5149	0.8563	0.7978	0.8881	0.603	0.7081	0.8444

Podemos observar con facilidad en la tabla la supremacía de las redes sobre casi todas las métricas (salvo PR AUC en el caso de validación cruzada aleatoria) para este caso de búsqueda empírica, además, comparado a los resultados presentados en la tabla 4 de [7], nuestras mediciones han sido ampliamente mejores.

Elección por búsqueda en malla

En [7], en la sección *Aggregate feature rankings and prediction on the top features*, hacen mención acerca de la metodología llevada a cabo para obtener la tabla 4. Cuando realizamos la selección empírica de hiperparámetros, solo seguimos las ideas acerca de la reproducción de los modelos y la división de los datos (aunque ignoramos la parte de considerar los datos de validación), esto puede considerarse una metodología muy restrictiva pues estamos realizando entrenamientos "a prueba y error" buscando cual produce los mejores resultados, sin embargo, en la práctica contamos con una cantidad limitada de datos, lo cual hace que al recibir nuevos datos, no podamos garantizar que nuestro modelo propuesto sea el mejor (ni siquiera nuestra metodología) y que la comparación en la efectividad de los modelos (hablando de una forma general) sea sesgada.

Por lo anterior, decidimos seguir una metodología más general, no imponemos ahora

los hiperparámetros y asumimos no tener los datos de prueba en cada simulación, la elección de los hiperparámetros fue realizada por medio de una *búsqueda en malla* combinado con validación cruzada K-Fold.

La metodología para encontrar hiperparámetros que empleamos para redes neuronales se elaboró en un código aparte y se realizó empleando la paquetería *TensorFlow*, la cual es una de las librerías más comunes para realizar el ajuste y entrenamiento de redes neuronales, en particular trabajamos con la versión 2.10.0 de esta librería, esto nos conllevó a usar la librería *numpy* en una versión menor a la 2.0, razón por la cual decidimos usar la 1.26.4 de ésta.

La idea para llevar a cabo la búsqueda en malla consistió en 3 pasos:

(1) Creamos un espacio de hiperparámetros, en pocas palabras, tomamos unos conjuntos predeterminados y el espacio de hiperparámetros resulta ser el producto cartesiano de éstos.

(2) Una vez dado el orden en el que fijamos los elementos del espacio de hiperparámetros, lo siguiente fue crear un modelo general de redes neuronales en base a esos hiperparámetros, para luego entrenarse.

(3) Por último se crea la función de búsqueda en malla, la cual es simplemente un ciclo donde, determinados los hiperparámetros, se realiza una validación cruzada K-Fold, la cual busca dividir a los datos en entrenamiento y validación, y con esto ajusta el modelo usando los datos de entrenamiento y con los datos de validación, calcula la precisión, nosotros nos fijamos en esta métrica para determinar el mejor modelo y los mejores hiperparámetros. Una simulación (repetición) consta de una división en datos de ajuste y datos de prueba.

Debemos destacar que el método de búsqueda en malla es bastante exhaustivo, explicaremos a que nos referimos con nuestro caso. Para este trabajo ocupamos los siguientes hiperparámetros, cuyos valores buscamos entre las listas de valores que también presentamos :

- **Neuronas de capa oculta:** $[30, 50, 80, 100]$
- **Función de activación:** $[\text{ReLU}, \text{Tanh}, \text{Sigmoide}]$
- **Tasa de aprendizaje:** $[0.1, 0.01, 0.001]$
- **Batch:** $[32, 64, 128]$
- **Épocas:** $[10, 30, 50, 100]$

Es fácil ver que el producto cartesiano de estos valores es un conjunto con 432 elementos, debido a que quisimos replicar lo más cercano que se pudiese el experimento presentado en el articulo [7], teníamos que dividir en 60 % para entrenamiento y 20 % para validación, es decir, primero hacíamos nuestra división 80 % para ajuste y 20 % para prueba (este último no se tocaba hasta el final de la búsqueda en malla). De la división hecha para el ajuste teníamos que realizar una división 75 %-25 % para que las divisiones mencionadas en [7] coincidiesen, dicha división se corresponde con un 4-Fold, es decir, realizábamos por cada elección de hiperparámetros el ajuste de 4 modelos. De este modo, por cada simulación (es decir, por cada división 80 %-20 %) requeríamos

entrenar $432 \times 4 = 1,728$ modelos de redes neuronales solo para escoger el mejor de ellos. La cantidad total de modelos entrenados al final del experimento consistió de 172,800, ya que se realizaron 100 simulaciones (repeticiones).

Debido a que cada simulación involucra una división distinta del conjunto de datos, podemos realizar las 100 simulaciones en diferentes momentos, por lo que para evitar conflictos por causas externas al programa, decidimos ejecutar en bloques de 10, estos bloques son independientes y la única consideración al final del experimento es tomar todos los valores recabados y dividir. En efecto, por cada bloque calculamos los promedios para cada una de las métricas enlistadas anteriormente, las cuales a su vez son calculadas por cada simulación. Como el promedio es una media aritmética (los bloques son del mismo tamaño) y las simulaciones son independientes, al final solo debemos promediar los resultados de cada bloque y esto nos da el promedio de las 100 simulaciones.

Los resultados presentados se enlistan como en la tabla 3.21, solo que ahora comparamos el método de redes neuronales en general con Regresión logística, SVM lineal, SVM polinomial, SVM gaussiano y SVM sigmoide, a los cuales se les realizó una búsqueda en malla usando la función *GridSearchCV* de la paquetería Scikit-Learn.

Las mallas usadas para la búsqueda de los hiperparámetros fueron los espacios formados por el producto cartesiano de las siguientes listas:

(1) SVM lineal y Regresión logística

- **Parámetro de regularización:** $[5, 1, 0.1, 0.01]$

(2) SVM polinomial

- **Término independiente:** $[5, 1, 0.1, 0.01]$
- **Coeficiente del kernel:** $[\frac{1}{\#\text{datos}}, 1, 0.1, 0.01]$
- **Grado:** $[3, 4, 5]$

(3) SVM gaussiano

- **Coeficiente del kernel:** $[\frac{1}{\#\text{datos}}, 1, 0.1, 0.01]$

(4) SVM sigmoide

- **Término independiente:** $[5, 1, 0.1, 0.01]$
- **Coeficiente del kernel:** $[\frac{1}{\#\text{datos}}, 1, 0.1, 0.01]$

Las 100 simulaciones fueron realizadas mediante validaciones cruzadas aleatorias. Los resultados obtenidos se presentan en la tabla 3.22.

Tabla 3.22: Resultados de la predicción de supervivencia sobre todas las características clínicas usando los datos estandarizados (elección por búsqueda en malla).

Media de las 100 simulaciones

Método	MCC	F_1 Score	Accuracy	TP rate	TN rate	PR AUC	ROC AUC
Redes neuronales	+0.5409	0.8624	0.8078	0.8884	0.6329	0.7332	0.8464
SVM lineal	**+0.5771**	**0.8697**	**0.8205**	0.8821	**0.6862**	0.7545	**0.8632**
Regresión log.	+0.5649	0.8684	0.8166	0.8912	0.6565	**0.7565**	**0.8632**
SVM polinomial	+0.5264	0.8585	0.8018	0.8877	0.6157	0.7338	0.8475
SVM sigmoide	+0.538	0.861	0.8056	0.8856	0.63468	0.7351	0.8539
SVM gaussiano	+0.504	0.8551	0.7943	**0.8947**	0.579	0.7144	0.8467

De lo obtenido, podemos destacar el buen trabajo en promedio que realizaron las redes neuronales, en general podemos observar la significativa mejora obtenida con respecto a los datos presentados en la tabla 4 de [7], dichas mejoras se deben a la estandarización de los datos, es decir, la sola estandarización de los datos le otorga mejor estabilidad a un modelo por redes neuronales, dicha diferencia es bastante clara con las métricas presentadas en [7].

Como conclusión, podemos ver que la estandarización de los datos puede mejorar de manera significativa los resultados de un problema de clasificación con redes neuronales, esto en particular nos ayuda a tener mayor confianza en las predicciones hechas por el modelo y permiten aumentar la seguridad de los expertos a la hora de tomar decisiones. Cabe destacar el trabajo realizado por los métodos SVM, los cuales tuvieron en general los mejores resultados.

Tenemos también que reconocer que nuestros datos son pocos, por lo que considerar algunas otras técnicas para muestreo o aumentar nuestra base de datos podrían hacer más precisos nuestros resultados.

3.1.3. Datos de cardiotocografía

Durante el embarazo, un número notable de mujeres experimentan dificultades, lo cual puede ocasionar graves problemas de salud en el feto. Sin embargo, una detección temprana de estos riesgos puede salvar la vida tanto del bebé como de la madre.

Como se menciona en [26], una de las técnicas más usadas para registrar cambios en las contracciones uterinas y la frecuencia cardíaca fetal (FHR por sus siglas en inglés) es la cardiotocografía (CTG). Los datos proporcionados por una cardiotocografía otorgan información valiosa mediante el monitoreo de la frecuencia cardiaca fetal, la presión de las contracciones uterinas maternas, los movimientos fetales en el útero y otros parámetros, dicha información es utilizada para predecir riesgos para el bienestar fetal y para la toma de decisiones clínicas. La evaluación cardiotocográfica es muy importante para identificar el bienestar de los fetos, pues observa si los tejidos reciben oxígeno, es decir monitorea la posibilidad de deficiencia de oxígeno (hipoxia) o acidosis.

El análisis de los datos de CTG permite conocer el estado de salud fetal y prevenir posibles riesgos para el bebé. Este tipo de prueba es sencilla, accesible y se acostumbra realizar después de la semana 28 (7mo mes) de embarazo. Hay que resaltar que la

detección precoz de cualquier anomalía presentada a través de la prueba permite al equipo médico tomar las medidas adecuadas y, posiblemente, prevenir la mortalidad materna y fetal.

Como se menciona en [4], en el siglo XXI, los sistemas de salud avanzados emplean con frecuencia diversos modelos de aprendizaje automático para predecir diversos problemas de salud. En la mayoría de los casos, estos modelos robustos son mucho más fiables. Debido a lo crucial de los resultados clínicos, una errónea decisión, por pequeña que sea, puede acarrear graves problemas de salud. En la prueba cardiotocográfica, un sistema de apoyo de decisión proporciona valiosos datos clínicos en tiempo real en forma de informes visuales, cuya interpretación depende de la experiencia de los obstetras. Si bien estos sistemas han apoyado desde antaño a la obstetricia para predecir el bienestar fetal, tienen menos probabilidades de gestionar situaciones inciertas. Por lo tanto, los médicos o expertos en obstetricia analizan manualmente estos datos sensibles para tomar las decisiones definitivas. Sin embargo, el riesgo de un error involuntario sigue presente, haciendo esta práctica vulnerable y por tanto con posibilidad de diagnosticar falsos negativos. Así mismo, hay varias investigaciones (véase las citas mencionadas en [4]) con distintos enfoques de aprendizaje automático para desarrollar sistemas viables que puedan realizar predicciones de manera eficaz, con menos errores y que mejoren significativamente la precisión, todo esto analizando los datos de CTG.

En cuanto a la base de datos sobre cardiotocogramas, ésta es proporcionada en [6] y consta de 2126 CTG fetales que se procesaron automáticamente, y a los cuales se han medido sus características diagnósticas. Tres obstetras expertos clasificaron los CTG y se les asignó una etiqueta de clasificación consensuada. La clasificación se realizó con respecto a un patrón morfológico $(A, B, C, \ldots)$ como al estado fetal (N, S, P).

Las 21 características consideradas para la clasificación, tanto del patrón morfológico como el estado fetal se muestran en la tabla 3.23.

Tabla 3.23: Variables resultantes de los cardiotocogramas.

LB	Base FHR.
AC	No. de aceleraciones por segundo.
FM	No. de movimientos fetales por segundo.
UC	No. de contracciones del útero por segundo.
DL	No. de desaceleraciones ligeras por segundo.
DS	No. de desaceleraciones severas por segundo.
DP	No. de desaceleraciones prolongadas por segundo.
ASTV	Porcentaje de tiempo con variabilidad anormal a corto plazo.
MSTV	Valor medio de la variabilidad a corto plazo.
ALTV	Porcentaje de tiempo con variabilidad anormal a largo plazo.
MLTV	Valor medio de la variabilidad a largo plazo.
Witdth	Ancho del histograma del FHR.
Min	Mínimo del histograma del FHR.
Max	Máximo del histograma del FHR.
Nmax	No. de picos del histograma.
Nzeros	No. de ceros del histograma.
Mode	Moda del histograma.
Mean	Media del histograma.
Median	Mediana del histograma.
Variance	Varianza del histograma.
Tendency	Tendencia del histograma.

Para una mejor comprensión de las métricas consideradas en el cardiotocograma, puede consultarse [20].

Predicción del patrón morfológico

Para el análisis del patrón morfológico usando la información de esta base de datos hemos construido un modelo a partir de una red neuronal artificial con una capa oculta y 150 neuronas, se permitieron $4,000$ épocas para el entrenamiento con una tasa de aprendizaje de $\eta = 0.001$, y como estamos usando el paso del gradiente estocástico, permitimos un tamaño de lote de 200. Hay que aclarar además que los modelos entrenados se hicieron usando el módulo *NeuralNetworks* que la librería Scikit-Learn ofrece, esto para facilitar el entrenamiento de modelos de redes neuronales aunque con ciertas limitaciones, como restringir el uso de una sola función de activación para todas las capas ocultas.

Se hicieron análisis comparando distintas funciones de activación en la capa oculta usando datos estandarizados. Los resultados presentados en la tabla 3.24 constituyen las estadísticas obtenidas de 10 validaciones cruzadas 5-Fold (50 modelos entrenados).

Tabla 3.24: Resultados obtenidos con datos estandarizados usando una red neuronal con 1 capa oculta.

Función sigmoide

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,048	266	30	0	44	20	73	0	22	337	3,840
B	198	5,298	0	42	61	143	30	0	0	18	5,790
C	374	67	74	0	0	0	15	0	0	0	530
D	0	300	0	484	0	26	0	0	0	0	810
E	236	97	0	0	195	9	0	0	0	183	720
F	67	144	5	20	2	2,891	167	24	0	0	3,320
G	62	0	1	0	5	131	2,239	62	0	20	2,520
H	0	0	0	0	0	8	128	934	0	0	1,070
I	43	0	0	0	0	0	0	4	333	310	690
J	514	23	3	0	12	0	0	0	69	1,349	1,970
Tot. Col.	4,542	6,195	113	546	319	3,228	2,652	1,024	424	2,217	21,260

Proporción de error de clasificación: 0.208
Proporción de clasificación correcta: 79.2 %

Función tangente hiperbólica

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	2,953	254	112	0	57	20	91	0	23	330	3,840
B	203	5,259	13	70	62	150	22	0	1	10	5,790
C	246	41	228	0	0	0	15	0	0	0	530
D	0	179	0	621	0	10	0	0	0	0	810
E	129	41	0	0	432	0	6	0	0	112	720
F	48	174	9	8	9	2,938	113	21	0	0	3,320
G	37	0	7	0	6	138	2,270	43	5	14	2,520
H	0	0	0	0	0	6	99	965	0	0	1,070
I	17	0	0	0	0	0	0	1	400	272	690
J	459	19	4	0	39	0	0	0	86	1,363	1,970
Tot. Col.	4,092	5,967	373	699	605	3,262	2,616	1,030	515	2,101	21,260

Proporción de error de clasificación: 0.180
Proporción de clasificación correcta: 82 %

Función ReLU

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,053	242	112	0	65	21	66	0	20	261	3,840
B	240	5,224	7	76	57	165	10	0	0	11	5,790
C	226	34	256	0	0	0	14	0	0	0	530
D	0	182	0	619	0	9	0	0	0	0	810
E	151	33	0	0	446	0	1	0	0	89	720
F	39	152	19	4	10	2,977	97	22	0	0	3,320
G	44	0	6	0	0	138	2,278	42	0	12	2,520
H	0	0	0	0	0	7	96	967	0	0	1,070
I	20	0	0	0	0	0	0	0	411	259	690
J	413	24	0	0	30	0	7	0	79	1,417	1,970
Tot. Col.	4,186	5,891	400	699	608	3,317	2,569	1,031	510	2,049	21,260

Proporción de error de clasificación: 0.170
Proporción de clasificación correcta: 83 %

Como comparación a la metodología de redes neuronales, implementamos también

el proceso de clasificación usando otras metodologías, tal como se hizo en los casos anteriores.

Se hace el uso de la metodología *uno contra el resto* (OVR) [17, 5] para la implementación de métodos lineales y no lineales, los resultados obtenidos con estos se presentan en las tablas 3.25, 3.26 y 3.27.

Tabla 3.25: Resultados obtenidos con datos estandarizados usando métodos lineales (OVR).

SVM lineal

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,039	224	121	0	61	20	74	0	14	287	3,840
B	263	5,199	21	94	31	145	19	0	4	14	5,790
C	247	42	238	0	0	0	3	0	0	0	530
D	0	188	0	609	0	13	0	0	0	0	810
E	138	59	0	0	452	1	0	0	0	70	720
F	58	159	0	6	0	2,942	146	48	0	0	3,320
G	61	1	11	0	5	138	2,244	48	0	12	2,520
H	0	1	6	0	0	7	83	973	0	0	1,070
I	35	0	0	0	4	0	0	10	419	222	690
J	496	31	2	0	43	0	0	0	124	1,274	1,970
Tot. Col.	4,337	5,904	399	709	596	3,266	2,569	1,040	561	1,879	21,260

Proporción de error de clasificación: 0.182
Proporción de clasificación correcta: 81.8 %

Regresión logística

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	2,955	346	51	0	46	15	89	0	22	316	3,840
B	260	5,292	9	30	58	120	19	0	0	2	5,790
C	317	90	85	0	9	0	29	0	0	0	530
D	0	307	0	452	0	51	0	0	0	0	810
E	183	45	0	0	372	0	16	0	0	104	720
F	77	353	0	5	10	2,589	252	34	0	0	3,320
G	146	0	2	0	2	143	2,161	47	0	19	2,520
H	0	0	10	0	0	15	119	926	0	0	1,070
I	65	3	0	0	0	0	0	7	341	274	690
J	552	69	0	0	24	0	3	0	50	1,272	1,970
Tot. Col.	4,555	6,505	157	487	521	2,933	2,688	1,014	413	1,987	21,260

Proporción de error de clasificación: 0.226
Proporción de clasificación correcta: 77.4 %

Los resultados presentados para los SVM kernel se obtuvieron con estos hiperparámetros: variamos el grado desde 3 hasta 5 para el kernel polinomial, con valor independiente 1, mismo valor independiente para el kernel sigmoide; en todos los casos el parámetro de escala fue $\gamma = \frac{1}{\#Datos}$.

Tabla 3.26: Resultados obtenidos con datos estandarizados usando SVM polinomial.

Polinomial Grado 3

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,194	211	100	0	50	17	49	0	4	215	3,840
B	255	5,252	13	50	12	176	8	3	0	21	5,790
C	237	31	260	0	0	0	2	0	0	0	530
D	0	178	0	617	0	15	0	0	0	0	810
E	194	35	0	0	399	0	4	0	0	88	720
F	34	186	9	8	0	2,974	90	19	0	0	3,320
G	70	2	16	0	3	150	2,220	47	0	12	2,520
H	0	0	0	0	0	6	81	983	0	0	1,070
I	29	0	0	0	9	0	0	2	422	228	690
J	440	21	0	0	32	0	7	0	65	1,405	1,970
Tot. Col.	4,453	5,916	398	675	505	3,338	2,461	1,054	491	1,969	21,260

Proporción de error de clasificación: 0.166
Proporción de clasificación correcta: 83.4 %

Polinomial Grado 4

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,205	211	112	0	65	16	58	0	9	164	3,840
B	265	5,195	9	76	18	192	8	4	3	20	5,790
C	211	31	280	0	0	1	7	0	0	0	530
D	0	132	0	661	0	17	0	0	0	0	810
E	177	45	0	0	403	0	9	0	0	86	720
F	54	213	9	8	0	2,918	101	17	0	0	3,320
G	59	2	22	0	6	184	2,187	48	0	12	2,520
H	0	0	3	0	0	5	102	960	0	0	1,070
I	25	0	0	0	12	0	0	5	515	133	690
J	382	26	0	0	40	1	7	0	63	1,451	1,970
Tot. Col.	4,378	5,855	435	745	544	3,334	2,479	1,034	590	1,866	21,260

Proporción de error de clasificación: 0.164
Proporción de clasificación correcta: 83.6 %

Polinomial Grado 5

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,233	224	110	0	68	15	55	0	14	121	3,840
B	271	5,219	11	75	16	162	6	9	5	16	5,790
C	195	33	286	0	0	0	16	0	0	0	530
D	0	138	0	658	0	14	0	0	0	0	810
E	161	61	0	0	410	0	9	0	0	79	720
F	62	229	12	11	0	2,865	127	14	0	0	3,320
G	61	8	26	0	3	179	2,179	55	0	9	2,520
H	0	0	8	0	0	12	94	956	0	0	1,070
I	21	0	0	0	12	0	0	10	530	117	690
J	314	28	0	0	54	1	7	0	67	1,499	1,970
Tot. Col.	4,318	5,940	453	744	563	3,248	2,493	1,044	616	1,841	21,260

Proporción de error de clasificación: 0.161
Proporción de clasificación correcta: 83.9 %

Tabla 3.27: Resultados obtenidos con datos estandarizados usando SVM gaussiano y sigmoide.

SVM gaussiano

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,203	252	38	0	24	26	36	0	1	260	3,840
B	289	5,220	9	33	23	174	14	0	0	28	5,790
C	315	40	169	0	0	0	6	0	0	0	530
D	0	258	0	538	0	14	0	0	0	0	810
E	277	58	0	0	231	0	4	0	0	150	720
F	42	203	6	0	0	2,819	212	38	0	0	3,320
G	90	2	16	0	1	158	2,202	44	0	7	2,520
H	0	0	0	0	0	17	71	982	0	0	1,070
I	41	0	0	0	0	0	0	6	344	299	690
J	500	17	0	0	3	0	18	0	22	1,410	1,970
Tot. Col.	4,757	6,050	238	571	282	3,208	2,563	1,070	367	2,154	21,260

Proporción de error de clasificación: 0.195
Proporción de clasificación correcta: 80.5 %

SVM sigmoide

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,194	211	100	0	50	17	49	0	4	215	3,840
B	255	5,252	13	50	12	176	8	3	0	21	5,790
C	237	31	260	0	0	0	2	0	0	0	530
D	0	178	0	617	0	15	0	0	0	0	810
E	193	35	0	0	399	0	4	0	0	0	720
F	34	186	9	8	0	2,974	90	19	0	0	3,320
G	70	2	16	0	3	150	2,220	47	0	12	2,520
H	0	0	0	0	0	6	81	983	0	0	1,070
I	29	0	0	0	9	0	0	2	422	228	690
J	440	21	0	0	32	0	7	0	65	1,405	1,970
Tot. Col.	4,453	5,916	398	675	505	3,338	2,461	1,054	491	1,969	21,260

Proporción de error de clasificación: 0.166
Proporción de clasificación correcta: 83.4 %

Hasta ahora los resultados obtenidos con redes neuronales habían superado los obtenidos con otros métodos como los SVM y la regresión logística. Sin embargo en este caso de datos multiclase, una sola capa oculta no parecía ofrecer mucha mejoría pese a que se estuviese aumentando la cantidad de neuronas ocultas (uno de los mejores resultados obtenidos empíricamente constaba de 150 neuronas), sin embargo, esto cambió al agregar una capa oculta más y pudimos reducir significativamente las neuronas. En efecto, con el fin de también analizar resultados para redes multicapa, decidimos considerar una red neuronal con dos capas ocultas, en la primera capa oculta incluimos 60 neuronas y en la segunda agregamos 50 más, nuevamente utilizamos las 3 funciones de activación mencionadas, nuestra tasa de aprendizaje fue de $\eta = 0.01$, permitimos un tamaño de lote de 200 y realizamos el proceso de entrenamiento durante 2500 épocas. Los resultados obtenidos se presentan en la tabla 3.28.

Tabla 3.28: Resultados obtenidos con datos estandarizados usando una red neuronal con 2 capas ocultas.

Función sigmoide

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,049	206	141	0	61	33	75	3	21	248	3,840
B	212	5,269	10	83	46	160	2	1	3	4	5,790
C	176	35	304	0	0	1	14	0	0	0	530
D	0	113	0	685	0	12	0	0	0	0	810
E	112	34	0	4	482	0	0	0	0	88	720
F	37	152	20	4	6	2,967	113	21	0	0	3,320
G	33	0	7	0	3	159	2,264	42	0	12	2,520
H	0	0	0	0	0	3	89	978	0	0	1,070
I	30	0	0	0	5	0	0	5	433	217	690
J	344	30	0	0	55	0	2	0	106	1,433	1,970
Tot. Col.	3,993	5,842	482	776	658	3,335	2,559	1,050	563	2,217	21,260

Proporción de error de clasificación: 0.160
Proporción de clasificación correcta: 84 %

Función tangente hiperbólica

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,144	263	103	0	70	21	77	0	18	144	3,840
B	197	5,270	8	69	43	166	17	3	5	12	5,790
C	170	22	322	0	2	0	14	0	0	0	530
D	0	101	0	696	2	11	0	0	0	0	810
E	127	34	0	2	497	4	0	0	2	54	720
F	45	167	13	6	5	2,954	106	24	0	0	3,320
G	53	0	6	0	8	152	2,248	38	0	15	2,520
H	0	1	8	0	0	8	75	977	1	0	1,070
I	8	0	0	0	12	0	0	1	546	123	690
J	232	14	0	0	55	0	8	0	85	1,576	1,970
Tot. Col.	3,976	5,872	460	773	694	3,316	2,545	1,043	657	1,924	21,260

Proporción de error de clasificación: 0.143
Proporción de clasificación correcta: 85.7 %

Función ReLU

	Patrón predicho										
Patrón	A	B	C	D	E	F	G	H	I	J	Tot. Fila
A	3,074	272	129	0	62	19	72	0	21	191	3,840
B	194	5,216	17	73	48	208	8	0	4	22	5,790
C	184	26	314	0	0	0	6	0	0	0	530
D	0	124	0	673	3	10	0	0	0	0	810
E	132	40	0	6	470	0	1	0	0	71	720
F	30	166	26	8	10	2,945	113	22	0	0	3,320
G	39	4	12	0	0	164	2,244	43	0	14	2,520
H	0	3	0	0	0	15	92	960	0	0	1,070
I	15	0	0	0	4	0	0	0	555	116	690
J	246	17	0	0	79	0	8	0	87	1,533	1,970
Tot. Col.	3,914	5,868	498	760	676	3,361	2,544	1,025	667	1,947	21,260

Proporción de error de clasificación: 0.154
Proporción de clasificación correcta: 84.6 %

Vemos con estos resultados que las redes neuronales pueden mejorar significativamente

con las capas ocultas sin necesidad de agregar tantas neuronas en ellas. Notemos que los mejores resultados que se vieron con 1 capa correspondían a 150 neuronas ocultas, mientras que con distribuir 110 neuronas en 2 capas (60-50) obtuvimos mejoras significativas.

Tabla 3.29: Métricas obtenidas para redes neuronales - Media de 50 simulaciones.

Red neuronal con activación sigmoide (1 capa)

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7262	0.8838	0.2279	0.7103	0.3701	0.883	0.8655	0.8911	0.5892	0.6423
Accuracy	0.8924	0.9346	0.9767	0.9817	0.9694	0.9639	0.9673	0.9893	0.9789	0.9299
TP rate	0.7937	0.9153	0.1513	0.6001	0.2773	0.8717	0.8889	0.8767	0.4921	0.6855
TN rate	0.9142	0.942	0.9981	0.9969	0.9939	0.9812	0.9779	0.9955	0.9955	0.955

Red neuronal con activación tangente hiperbólica (1 capa)

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7439	0.8942	0.5048	0.8185	0.6404	0.8928	0.8836	0.9175	0.6545	0.6673
Accuracy	0.9047	0.9417	0.9789	0.9874	0.9783	0.9667	0.9719	0.992	0.9809	0.9367
TP rate	0.7693	0.9085	0.4505	0.7682	0.5989	0.8857	0.9011	0.9049	0.5799	0.6924
TN rate	0.9346	0.9542	0.993	0.9961	0.9915	0.9819	0.9815	0.9967	0.9944	0.9617

Red neuronal con activación ReLU (1 capa)

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7604	0.894	0.5428	0.8151	0.662	0.8966	0.895	0.9194	0.6805	0.7031
Accuracy	0.9096	0.942	0.9803	0.9872	0.9794	0.9678	0.9749	0.9921	0.9822	0.9442
TP rate	0.7959	0.9021	0.5001	0.7647	0.6196	0.8967	0.9043	0.9071	0.6011	0.7201
TN rate	0.9349	0.9568	0.993	0.996	0.9921	0.981	0.9844	0.9968	0.9951	0.9672

Red neuronal con activación sigmoide (2 capas)

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7777	0.9057	0.5965	0.8606	0.6934	0.8909	0.8910	0.9198	0.6866	0.7203
Accuracy	0.9183	0.9485	0.9809	0.9898	0.9805	0.9660	0.9740	0.9922	0.9817	0.9479
TP rate	0.7945	0.9102	0.5868	0.8437	0.6713	0.8934	0.8985	0.9146	0.6333	0.7282
TN rate	0.9458	0.9629	0.9914	0.9955	0.9914	0.9794	0.9842	0.9964	0.9936	0.9705

Red neuronal con activación tangente hiperbólica (2 capas)

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.8033	0.9035	0.6433	0.8777	0.6934	0.8902	0.8871	0.9216	0.8057	0.8063
Accuracy	0.9281	0.9472	0.9837	0.9910	0.9802	0.9657	0.9732	0.9925	0.9880	0.9651
TP rate	0.8186	0.9104	0.6193	0.8599	0.6887	0.8905	0.8918	0.9146	0.7934	0.7991
TN rate	0.9522	0.9610	0.9933	0.9962	0.9904	0.9798	0.9841	0.9967	0.9946	0.9819

Red neuronal con activación ReLU (2 capas)

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7926	0.8948	0.5998	0.8504	0.6682	0.8809	0.8859	0.9138	0.8150	0.7803
Accuracy	0.9244	0.9423	0.9811	0.9894	0.9785	0.9627	0.9729	0.9917	0.9883	0.9599
TP rate	0.8007	0.9011	0.5994	0.8265	0.6531	0.8871	0.8902	0.8991	0.8061	0.7775
TN rate	0.9517	0.9578	0.9911	0.9957	0.9899	0.9768	0.9839	0.9967	0.9945	0.9785

Tabla 3.30: Métricas obtenidas para otros métodos - Media de 50 simulaciones.

SVM lineal

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	07427	0.8888	0.5113	0.7982	0.6749	0.8933	0.8816	0.9204	0.6604	0.66
Accuracy	0.9012	0.939	0.9786	0.9858	0.9806	0.9669	0.9717	0.9922	0.9805	0.9388
TP Rate	0.7923	0.8982	0.4693	0.7563	0.6239	0.8866	0.8911	0.9113	0.6062	0.6481
TN Rate	0.9254	0.9544	0.9922	0.9951	0.9929	0.9819	0.9826	0.9966	0.9931	0.9686

Regresión logística

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7031	0.8604	0.2456	0.6941	0.5918	0.8280	0.8301	0.8879	0.6139	0.6409
Accuracy	0.8831	0.9195	0.9756	0.9815	0.9766	0.9494	0.9582	0.989	0.9801	0.9335
TP Rate	0.7704	0.9141	0.1723	0.5671	0.523	0.7814	0.8593	0.8686	0.5022	0.6466
TN Rate	0.9081	0.9215	0.9965	0.9982	0.9927	0.9808	0.9718	0.9956	0.9965	0.9629

SVM polinomial grado 3

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7697	0.8972	0.5596	0.8264	0.6383	0.8923	0.8914	0.923	0.7091	0.7107
Accuracy	0.9103	0.9434	0.9808	0.9881	0.9799	0.9666	0.9745	0.9925	0.9841	0.9468
TP Rate	0.8323	0.9075	0.5134	0.7642	0.5539	0.8958	0.8815	0.9197	0.6153	0.7133
TN Rate	0.9277	0.957	0.9933	0.9971	0.9948	0.9797	0.9871	0.9964	0.9966	0.9707

SVM polinomial grado 4

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7796	0.8919	0.5775	0.8461	0.6269	0.8759	0.8751	0.9102	0.8029	0.7543
Accuracy	0.9149	0.9409	0.9809	0.9890	0.9784	0.9615	0.9706	0.9913	0.9882	0.9560
TP Rate	0.8354	0.8974	0.5515	0.8174	0.5604	0.8787	0.8685	0.8997	0.7517	0.7368
TN Rate	0.9326	0.9573	0.9925	0.9958	0.9931	0.9768	0.9844	0.9963	0.9963	0.9785

SVM polinomial grado 5

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7919	0.8899	0.5746	0.8408	0.6229	0.8713	0.8689	0.9025	0.8094	0.7850
Accuracy	0.9204	0.9392	0.9806	0.9888	0.9782	0.9605	0.9691	0.9904	0.9884	0.9617
TP Rate	0.8423	0.9018	0.553	0.8115	0.5695	0.863	0.8653	0.8953	0.7722	0.761
TN Rate	0.9377	0.9533	0.9919	0.9957	0.9925	0.9786	0.9832	0.9956	0.9958	0.9822

SVM gaussiano

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7445	0.8815	0.4393	0.7763	0.4493	0.8628	0.8658	0.9152	0.6368	0.6812
Accuracy	0.8969	0.9341	0.9797	0.9856	0.9746	0.9581	0.9680	0.9917	0.9826	0.9386
TP Rate	0.8347	0.9020	0.3419	0.6692	0.3211	0.8498	0.8741	0.9184	0.4967	0.7157
TN Rate	0.9107	0.9463	0.9966	0.9983	0.9975	0.9783	0.9807	0.9956	0.9988	0.9614

SVM sigmoide

	Clases									
Métrica	A	B	C	D	E	F	G	H	I	J
F_1 Score	0.7697	0.8972	0.5596	0.8264	0.63843	0.8923	0.8914	0.923	0.7091	0.7107
Accuracy	0.9103	0.9434	0.9808	0.9881	0.9799	0.9666	0.9745	0.9925	0.9841	0.9468
TP Rate	0.8323	0.9075	0.5134	0.7642	0.5539	0.8958	0.8815	0.9197	0.6153	0.7133
TN Rate	0.9277	0.957	0.9933	0.9971	0.9948	0.9797	0.9871	0.9964	0.9966	0.9707

Por último, en las tablas 3.29 y 3.30 añadimos algunas métricas de las presentadas en el ejemplo de insuficiencia cardíaca. Para calcular los verdaderos positivos (TP), los verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN) nos basamos en las explicaciones dadas en [21], donde se nos habla acerca de la idea de cómo calcular estos valores (los cuales son necesarios para determinar las métricas implementadas) para el caso multiclase.

Las últimas tablas nos permiten visualizar de forma local que tan bueno está siendo nuestro modelo, es decir, al tomar medidas sobre una clase previamente fijada, estamos estimando que tan eficiente es el modelo para predecir dicha clase. Esto puede ser de utilidad si ciertos patrones morfológicos resultan de mayor relevancia y por tanto, mayor confiabilidad en predecirlos.

Podemos ver que las redes neuronales con 2 capas ocultas arrojan los mejores resultados en cuanto a precisión (accuracy) y F_1 score en la mayoría de las clases, por otra parte las mediciones obtenidas para la razón de verdaderos positivos (TP Rate) y verdaderos negativos (TN Rate) varían más entre las clases.

De este ejemplo concluimos que aunque el teorema de aproximación universal nos garantiza que podemos obtener resultados cercanas al error de Bayes, en la práctica requerimos a veces más capas, esto debido a la forma de los datos y también a la aproximación numérica. Se requiere así, buscar alternativas que generen un equilibrio entre trabajo computacional y tiempo.

Los resultados obtenidos con redes neuronales en este caso fueron los más destacables. En este ejemplo la mejora fue mucho más significativa en promedio que en los ejemplos anteriores, donde las mejoras eran de menos del 1 %; sin embargo, ahora alcanzamos mejoras de hasta casi 3 %, lo cual en una toma de decisiones crítica juega un papel importante.

Predicción de estado fetal

Por otra parte, con la información del caso de clasificación del estado fetal quisimos hacer un análisis diferente. A continuación realizamos una comparación de tamaños de lote, esto es, probamos con distintos tamaños de lote el entrenamiento de una red, fijamos los hiperparámetros como neuronas ocultas, capas, funciones de activación y los sometimos a la misma cantidad de épocas, nos encargamos de medir la precisión y el tiempo (en segundos) de ejecución. Los resultados obtenidos se presentan en las tablas 3.31-3.33, estos se obtienen después de realizar 20 simulaciones de validación cruzada aleatoria.

La idea de comparar la relación precisión-lote-tiempo es porque la elección del tamaño de lote es un método de regularización en el proceso de redes neuronales, como se mencionó anteriormente, el descenso de gradiente estocástico es una versión del gradiente descendente en la que sólo usamos un dato para actualizar los pesos, pues es probable que la contribución de un solo dato sea necesaria para encaminar en la dirección del gradiente general, sin embargo, hay probabilidad de que no sea así, por ello, decidimos comparar distintos tamaños de lotes ya que el mini lote es un paso intermedio entre el descenso del gradiente y el descenso del gradiente estocástico, pues no es tan lento

(debe tenerse en cuenta que el número de épocas define el número de veces que usamos cada dato en las actualizaciones del gradiente, es decir, entre más pequeño el lote, más número de actualizaciones por época) y tampoco requiere tanto uso de memoria (a medida de que los datos van aumentando en cantidad, los requerimientos de operaciones matriciales para el algoritmo backpropagation se vuelven bastante grandes). Así, el fin de esta comparación es ver qué tanta eficiencia hay a razón del coste computacional.

Tabla 3.31: Resultados obtenidos usando activación sigmoide.

60 neuronas (1 capa) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9091	507.5418
16	0.9009	403.9377
32	0.8909	179.4439
64	0.8892	86.2072
128	0.8865	58.3359
256	0.8829	68.9876

60 neuronas (1 capa) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9102	589.0958
16	0.9123	681.3236
32	0.8909	183.3855
64	0.8892	85.7618
128	0.8865	57.5582
256	0.8829	68.7514

80 neuronas (1 capa) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9017	438.0317
16	0.8973	363.9524
32	0.8929	191.4087
64	0.8902	99.3091
128	0.8886	65.1761
256	0.8816	77.0079

80 neuronas (1 capa) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9018	467.8477
16	0.9008	472.4479
32	0.8929	188.7854
64	0.8902	99.6945
128	0.8886	67.1840
256	0.8816	75.9819

60-40 neuronas (2 capas) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9018	467.8477
16	0.9008	472.4479
32	0.8929	188.7854
64	0.8902	99.6945
128	0.8886	67.1840
256	0.8816	75.9819

60-40 neuronas (2 capas) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9009	443.9641
16	0.8976	442.1566
32	0.8935	304.1436
64	0.8896	180.8482
128	0.8843	189.7946
256	0.8503	165.3432

Tabla 3.32: Resultados obtenidos usando activación tangente hiperbólica.

60 neuronas (1 capa) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9279	419.5929
16	0.9266	340.5251
32	0.9274	231.5162
64	0.9188	144.9726
128	0.9043	86.7888
256	0.8946	52.2734

60 neuronas (1 capa) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9279	417.5351
16	0.9266	345.5307
32	0.9260	284.6504
64	0.9218	194.4075
128	0.9049	89.2810
256	0.8946	52.4986

80 neuronas (1 capa) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9264	420.3386
16	0.9269	349.5904
32	0.9260	251.2481
64	0.9214	158.6260
128	0.9070	92.9163
256	0.8977	55.4263

80 neuronas (1 capa) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9264	420.5928
16	0.9271	369.4354
32	0.9261	323.5397
64	0.9235	208.3722
128	0.9070	91.9985
256	0.8977	55.4053

60-40 neuronas (2 capas) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9275	332.6351
16	0.9275	262.2571
32	0.9291	230.0259
64	0.9271	201.9434
128	0.9190	189.1427
256	0.8997	116.5998

60-40 neuronas (2 capas) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9275	315.1826
16	0.9275	265.4274
32	0.9291	231.4031
64	0.9279	263.9731
128	0.9242	319.3551
256	0.8997	118.8669

Tabla 3.33: Resultados obtenidos usando activación ReLU.

60 neuronas (1 capa) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9261	257.2916
16	0.9281	194.1422
32	0.9279	184.4885
64	0.9235	123.3630
128	0.9124	70.4189
256	0.9030	50.5879

60 neuronas (1 capa) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9261	257.2045
16	0.9281	194.8559
32	0.9281	182.8400
64	0.9259	137.9942
128	0.9124	70.0756
256	0.9030	50.2526

80 neuronas (1 capa) con 1,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9277	263.7815
16	0.9282	198.9136
32	0.9274	200.0676
64	0.9244	132.5083
128	0.9147	82.7581
256	0.9031	60.8812

80 neuronas (1 capa) con 2,000 épocas

Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9277	263.1839
16	0.9282	197.8756
32	0.9274	205.2513
64	0.9250	148.2906
128	0.9149	81.9534
256	0.9031	60.5229

60-40 neuronas (2 capas) con 1,000 épocas

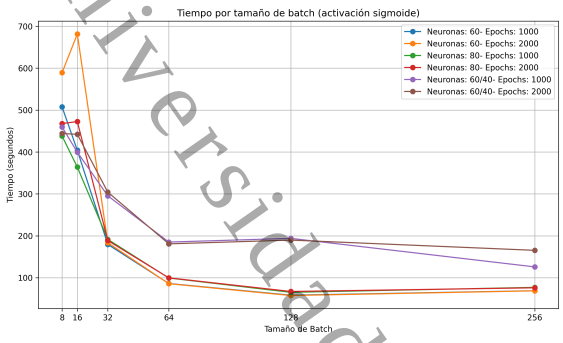
Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9239	225.4186
16	0.9264	188.5401
32	0.9269	150.6167
64	0.9275	174.0730
128	0.9237	169.6510
256	0.9104	115.5469

60-40 neuronas (2 capas) con 2,000 épocas

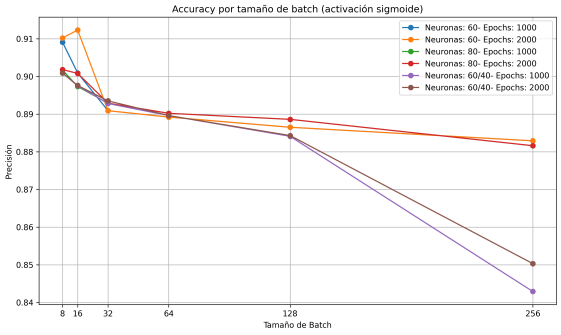
Tamaño de lote	Precisión (Accuracy)	Tiempo
8	0.9239	225.1994
16	0.9264	187.1879
32	0.9269	147.5479
64	0.9281	178.6151
128	0.9262	244.0205
256	0.9113	117.7750

Con el fin de analizar mejor las tablas, las figuras en 3.1 se hacen dos a dos por activación, de tal forma que es más fácil analizar la información.

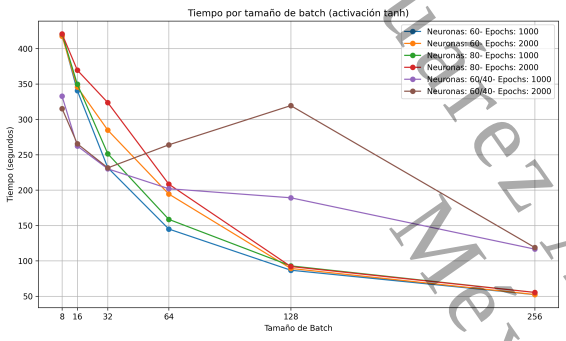
(a) Sigmoide - batch vs tiempo



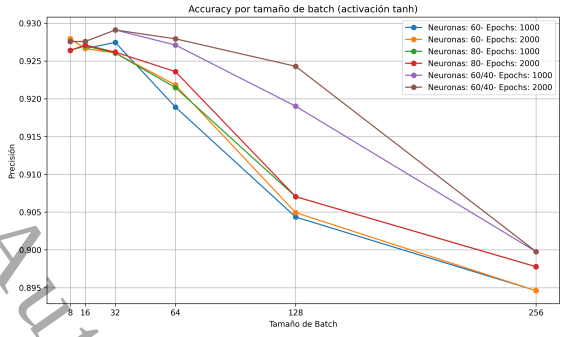
(b) Sigmoide - batch vs precisión



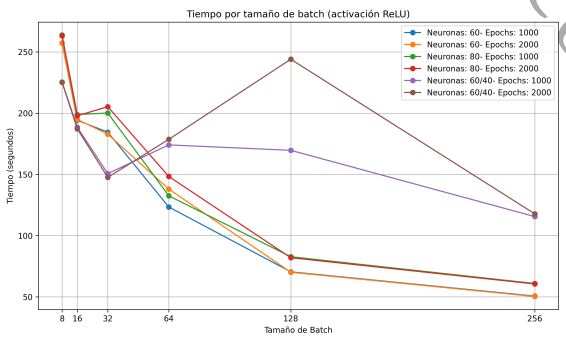
(c) Tangente hiperbólica - batch vs tiempo



(d) Tangente hiperbólica - batch vs precisión



(e) ReLU - batch vs tiempo



(f) ReLU - batch vs precisión

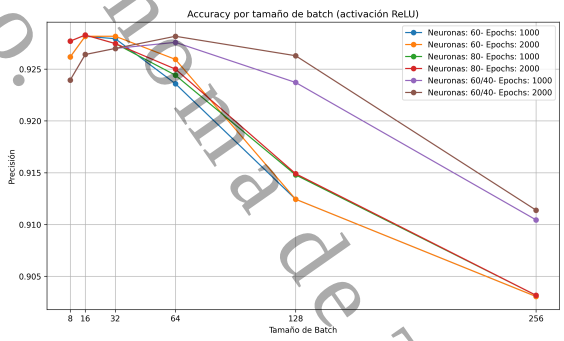


Figura 3.1: Gráficas resultantes de la comparación del tamaño de lote.

Aquí es fácil observar cómo reacciona cada función de activación al uso de batchs, vemos que en el caso de la función sigmoide tenemos una correspondencia directa (a menor tiempo, menor precisión) hasta los lotes de tamaño 32; para tamaños de 64 en adelante vemos como se parece nivelar el tiempo de ejecución (alrededor de 100 segundos en 4 modelos y menos de 200 en los restantes), pero la precisión comienza a decaer gradualmente. Por otra parte, con la función de activación tangente hiperbólica es fácil identificar que las redes con 2 capas que fueron propuestas son beneficiadas por el uso de lotes de tamaño 32, pues presentaron menor tiempo de cómputo usando ese

lote y aún así, fueron más eficaces. Finalmente, para la ReLU, al considerar dos capas ocultas, vemos que el tamaño de lote no parece afectarle tanto en cuanto al tiempo y devuelve una precisión bastante alta.

En la tabla 3.34 presentamos las matrices de confusión de las redes que mejores resultados devolvieron en cuanto a relación de tiempo-precisión. Agregamos también la información del tamaño de lote y las características de la red.

Tabla 3.34: Matrices de confusión de las mejores redes por función de activación.

Sigmoide - Lote: 8, Neuronas: 60, Épocas: 1000

	Estado predicho			
Estado	Normal	Sospechoso	Patológico	Total Fila
Normal	6,369	246	36	6,651
Sospechoso	258	853	72	1,183
Patológico	60	102	524	686
Total Columna	6,687	1,021	632	8,520

Tiempo total de ejecución: 507.54 segundos
Proporción de clasificación correcta: 90.9 %

Tanh - Lote: 32, Neuronas: (60,40), Épocas: 1000

	Estado predicho			
Estado	Normal	Sospechoso	Patológico	Total Fila
Normal	6,433	185	33	6,651
Sospechoso	248	885	50	1,183
Patológico	45	56	585	686
Total Columna	6,726	1,126	668	8,520

Tiempo total de ejecución: 230.02 segundos
Proporción de clasificación correcta: 92.9 %

ReLU - Lote: 16, Neuronas: 60, Épocas: 1000

	Estado predicho			
Estado	Normal	Sospechoso	Patológico	Total Fila
Normal	6,432	186	33	6,651
Sospechoso	243	898	42	1,183
Patológico	46	62	578	686
Total Columna	6,721	1,146	653	8,520

Tiempo total de ejecución: 194.1422 segundos
Proporción de clasificación correcta: 92.8 %

Como conclusión, podemos observar que el tamaño de lote si afecta la efectividad y que dependiendo de los hiperparámetros, ciertos tamaños de lote pueden beneficiar al modelo. Así mismo podemos observar la alta efectividad de las redes neuronales para clasificar estos datos, lo cual es importante pues en una toma de decisiones se ve involucrada una vida y por tanto una responsabilidad que requiere la mayor precisión posible, pero que esta misma sea realista.

Aunado a todo esto, podemos comparar la precisión de nuestro modelo con la del

modelo propuesto en [4] donde registran una efectividad del 96.05 %, teniendo en cuenta que ellos realizaron una reducción de características y un proceso de búsqueda en malla para la elección de sus hiperparámetros. Un posible trabajo futuro sería analizar los procesos que ellos llevaron a cabo y replicar ese mismo proceso adaptándolo a redes neuronales y ver si hay una mejora en los resultados presentados por ellos.

Conclusiones y trabajos futuros

Aunque en este trabajo se presentó una cantidad considerable de información acerca de las redes neuronales artificiales, hay aún muchos más temas por estudiar; de manera que este trabajo representa solo una breve introducción sobre redes neuronales artificiales hacia adelante (*feed-forward*), hemos partido de la interpretación biológica hacia al modelo matemático, el cual se justifica a través del teorema de aproximación universal. Consideramos así, cumplido uno de los fines de este escrito, pues una de las ideas fundamentales era presentar la conexión de las redes neuronales artificiales con el cerebro humano y cómo este último es de inspiración para los modelos actuales.

Tenemos presente que no se han abarcado completamente todas las ideas de las redes neuronales, pues requeriríamos mucho más tiempo para poder hablar sobre métodos de regularización (L_1, L_2, *dropout, early stopping*), transferencia de aprendizaje (*transfer learning*), así como trabajar con otros tipos de redes neuronales (*redes neuronales recurrentes* y *redes neuronales convolucionales*). Pese a todo esto, hemos presentado las ideas básicas con las cuales podemos entender como funciona una red neuronal. Por el lado de las redes convolucionales, debido a que hemos estado trabajando con bases de datos del área de la salud, creemos en el potencial de este tipo de redes como herramientas de gran utilidad para la detección de enfermedades crónicas cuyo diagnóstico requiere un análisis preciso de imágenes clínicas, este tipo de información aún no se comparte tan abiertamente debido a temas de ética en el área de la salud, por lo que el tipo de datos para experimentar está un poco limitado, sin embargo, consideramos que una continuación directa de este trabajo podría ser el estudio de redes convolucionales y su uso para la detección de enfermedades a través de imágenes clínicas, las cuales podrían ser buscadas en algún centro de salud que esté dispuesto a colaborar.

Queremos destacar que esta área del aprendizaje máquina sigue manteniéndose ampliamente activa, en gran parte a su utilidad en los sistemas de inteligencia artificial, razón por la cual, parte de este trabajo se dirige a confirmar dicha utilidad en la actualidad. Al conocer el funcionamiento de las redes, uno se quita el paradigma de complejidad hacia los programas de clasificación de hoy en día, nos resulta más intuitiva la forma en que se estructuran las redes y también cómo están diseñadas, por lo cual concluimos que el objetivo de estudiar el desarrollo de las redes neuronales y su aplicación se ha cumplido satisfactoriamente en este trabajo de tesis.

A lo largo de este escrito, no solo hemos considerado parte de la teoría matemática y computacional, sino también hemos implementado la práctica, con datos de libre acceso, a través de la construcción de modelos de redes neuronales para clasificar datos del área médica. Cada uno de los ejemplos presentados en este trabajo fue llevado a cabo de forma distinta, con el propósito de lograr una mejor comprensión de la metodología de las redes neuronales. Cabe destacar también que dicho enfoque nos permitió entender algunos conceptos estadísticos que son de común utilidad en el área de la salud, el ejemplo más claro de lo descrito son las métricas mencionadas en el ejemplo de los datos de insuficiencia cardíaca, donde dichas métricas nos permiten evaluar la capacidad de predicción de un modelo desde diferentes perspectivas. Es

claro que el objetivo de proporcionar ejemplos para el área de la salud también se ha cumplido satisfactoriamente.

Siguiendo con lo anterior, tenemos que reconocer la importancia que han tomado los métodos de aprendizaje automático en la toma de decisiones para el área médica, esto lo vemos claramente debido a los grandes volúmenes de información que se manejan hoy, así como en la alta demanda de especialistas, esto provoca que la toma de decisiones por unos cuantos expertos sea complicada y el tener estos modelos pueden ayudar a tener una guía de la urgencia a la hora de decidir que caso tomar o prestar más atención, siendo una ayuda en la carga médica además de contribuir en la reducción del estrés y la carga emocional que trabajar con pacientes delicados de salud pueda conllevar.

Finalmente, en lo personal, puedo afirmar que este trabajo ha servido ampliamente para conocer el lenguaje de programación python, acercarse más a la programación en general y también entender el potencial de los ordenadores para construir herramientas que sean útiles en diferentes áreas del conocimiento. Destaco esto, debido a que uno de los objetivos de este proyecto era explotar el potencial de las librerías del lenguaje python para redes neuronales, lo cual también hemos realizado de manera exitosa desde mi parecer.

Referencias

[1] Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer.

[2] Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

[3] Ansari, S. (2020). *Building computer vision applications using artificial neural networks*. Apress.

[4] Bhowmik, P. et al. (2021). «Cardiotocography data analysis to predict fetal health risks with tree-based ensemble learning». En: *Inf. Technol. Comput. Sci* 5, págs. 30-40.

[5] Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

[6] Campos, D. y Bernardes, J. (2000). *Cardiotocography [Dataset]*. UCI Machine Learning Repository. DOI: 10.24432/C51S4N.

[7] Chicco, D. y Jurman, G. (2020). «Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone». En: *BMC medical informatics and decision making* 20.1, págs. 1-16.

[8] Codecademy. (s.f.). *Normalization*. URL: https://www.codecademy.com/article/normalization.

[9] Devroye, L., Györfi, L. y Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. Springer.

[10] Dreyfus, G. (2005). *Neural networks: methodology and applications*. Springer Science & Business Media.

[11] Folland, G. B. (1999). *Real analysis: modern techniques and their applications*. John Wiley & Sons.

[12] Francisco-Caicedo, E. F. y López-Sotelo, J. A. (2009). *Una aproximación práctica a las redes neuronales artificiales*. Cali: Universidad del Valle.

[13] Haykin, S. (2009). *Neural networks and learning machines*. 3.ª ed. Pearson Education India.

[14] Iliadis, L. S., Papadopoulos, H. y Jayne, C. (Eds.). (2013). *Engineering applications of neural networks: 14th International Conference, EANN 2013, Halkidiki, Greece, September 2013, Proceedings, Part I*. Springer.

[15] Instituto Nacional de la Salud infantil y Desarrollo Humano Eunice Kennedy Shriver. (2019). *¿Cuáles son las partes del sistema nervioso?* URL: https://espanol.nichd.nih.gov/salud/temas/neuro/informacion/partes (consultado el 14 de nov. de 2023).

[16] Izaurieta, F. y Saavedra, C. (2000). *Redes neuronales artificiales*. Departamento de Física, Universidad de Concepción Chile.

[17] Izenman, A. J. (2008). *Modern multivariate statistical techniques*. New York: Springer.

[18] Kohonen, T. (2000). *Self-organizing maps*. Springer, Berlin.

[19] Krawczak, M. (2013). *Multilayer Neural Networks*. Springer.

[20] Libretexts. (2024, 30 de octubre). *Basic Terms of Fetal Heart Rate and Contraction Patterns*. URL: https://med.libretexts.org/Bookshelves/Nursing/Maternal-Newborn_Nursing_(OpenStax)/16:_Electronic_Fetal_and_Uterine_Contraction_Monitoring/16.02:_Basic_Terms_of_Fetal_Heart_Rate_and_Contraction_Patterns (consultado el 15 de mar. de 2025).

[21] Loukas, S. (2025, 20 de enero). *Multi-class classification: extracting performance metrics from the confusion matrix*. URL: https://towardsdatascience.com/multi-class-classification-extracting-performance-metrics-from-the-confusion-matrix-b379b427a872/ (consultado el 12 de ago. de 2025).

[22] McCulloch, W. S. y Pitts, W. (1943). «A logical calculus of the ideas immanent in nervous activity». En: *Bulletin of Mathematical Biophysics* 5, págs. 115-133.

[23] Minsky, M. L. y Papert, S. A. (1969). *Perceptrons: An Introduction to Computational Geometry*. Expanded edition 1990.

[24] Montesinos-López, O. A., Montesinos-López, A. y Crossa, J. (2022). *Multivariate statistical machine learning methods for genomic prediction*. Springer.

[25] Pedregosa, F. et al. (2011). «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12, págs. 2825-2830.

[26] Permanasari, A. E. y Nurlayli, A. (2017). «Decision tree to analyze the cardiotocogram data for fetal distress determination». En: *International Conference on Sustainable Information Engineering and Technology (SIET)*.

[27] Prasad, B. y Prasanna, S. R. M. (Eds.). (2007). *Speech, audio, image and biomedical signal processing using neural networks*. Springer.

[28] Quiroz, F. (1991). *Anatomía humana*. México: Editorial Porrúa.

[29] Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan.

[30] Rubin, M. (s.f.). *Introducción al sistema nervioso periférico - Enfermedades cerebrales, medulares y nerviosas*. Manual MSD versión para público general. URL: https://www.msdmanuals.com/es/hogar/enfermedades-cerebrales-medulares-y-nerviosas/trastornos-del-nervio-perif%C3%A9rico-y-trastornos-relacionados/introducci%C3%B3n-al-sistema-nervioso-perif%C3%A9rico (consultado el 16 de nov. de 2023).

[31] Scikit-learn. (s.f.). *Scikit-learn: machine learning in Python — scikit-learn 1.6.1 documentation*. URL: https://scikit-learn.org/stable/index.html.

[32] Taylor, B. J. (Ed.). (2006). *Methods and procedures for the verification and validation of artificial neural networks*. Springer Science & Business Media.

[33] TensorFlow. (s.f.). *TensorFlow Official Website*. URL: https://www.tensorflow.org/ (consultado el 21 de abr. de 2025).

[34] Zhang, X.-S. (2013). *Neural networks in optimization*. Springer Science & Business Media.

Alojamiento de la Tesis en el Repositorio Institucional	
Título de la Tesis:	Estudio de redes neuronales con aplicaciones al área de la salud
Autor de la Tesis:	Saúl David Candelero Jiménez
ORCID:	0009-0001-2531-4896
Resumen de la Tesis:	**Resumen:** A lo largo de este trabajo se estudia la teoría de redes neuronales como modelos de predicción, iniciando con la interpretación biológica hasta llegar a las bases matemáticas que sustentan el uso de éstas. Posteriormente, se pasa a la construcción de las redes neuronales usando programación en python y se explica la elección del mejor modelo a partir del empleo de métodos de validación cruzada. Todos estos pasos se usan en conjunto para la construcción de modelos de redes neuronales con aplicación a bases de datos médicos, de los cuales se obtienen ciertas conclusiones importantes. **Abstract:** This work explores neural network theory as a predictive model, beginning with its biological interpretation and progressing to the mathematical foundations that underpin its use. We then examine the construction of neural networks using Python programming and explain how the best model is selected through cross-validation methods. All these steps are combined to build neural network models applicable to medical databases, from which we can draw certain important conclusions.
Palabras clave de la Tesis:	**Palabras clave:** Red neuronal, aprendizaje profundo, gradiente descendente, validación cruzada, aproximación universal, clasificación. **Keywords:** Neural networks, deep learning, gradient descent, cross-validation, universal approximation, classification.

Referencias citadas: