UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA, LEÓN

FACULTAD DE CIENCIAS Y TECNOLOGÍA

DEPARTAMENTO DE MATEMÁTICA, ESTADÍSTICA Y ACTUARIALES

LICENCIATURA EN MATEMÁTICA

MONOGRAFÍA PARA OPTAR AL TÍTULO DE LICENCIATURA EN MATEMÁTICA

REDES NEURONALES ARTIFICIALES PARA LA CLASIFICACIÓN DEL RENDIMIENTO ACADÉMICO DE ESTUDIANTES DE LA FACULTAD DE CIENCIAS Y TECNOLOGÍA DE LA UNAN-LEÓN 2011-2018

PRESENTADO POR:

Br. Albert Alonso Martínez Espinal

Br. Jackson Ulises Palma Sandoval

Br. Nestor Adrian Ordoñez Rivera

TUTORA:

PhD. Laureana Adalila Molina Membreño

ASESORES:

M.Sc. William Milton Carvajal Herradora

M.Sc. Lissette del Carmen Quintero Vargas

León, Nicaragua, junio 2021

“A la Libertad por la Universidad”

DEDICATORIA

Dedicamos nuestro trabajo en primer lugar a Dios por ser nuestra fuerza, a nuestros familiares y amigos por ser un soporte tanto en lo emocional y económico durante toda la carrera, también a todos los maestros que fueron parte de nuestra formación como profesionales.

AGRADECIMIENTOS

Queremos expresar nuestra gratitud a Dios por guiarnos a lo largo de nuestra existencia, ser el apoyo y fortaleza en aquellos momentos de dificultad y de debilidad.

A mis padres, familiares y amigos quienes fueron el motor y nuestra mayor inspiración, para culminar está etapa en nuestras vidas.

Agradezco a mi tutora de tesis PhD. Laureana Adalila Molina Membreño, así mismo a mis asesores M. Sc Lissette Quintero Vargas y M. Sc William Milton Carvajal quienes con su experiencia, conocimiento y motivación me orientaron en la investigación.

Muchas gracias a todos.

RESUMEN

En el siguiente trabajo investigativo se presentan los fundamentos teóricos de Redes Neuronales Artificiales, su funcionamiento y algunas de sus aplicaciones. Se describe una metodología con la cual se pudo llevar a cabo el proceso de creación de un código, que permite la clasificación multiclase de rendimientos académicos de estudiantes de la facultad de ciencias y tecnología de la UNAN-LEÓN 2011-2018, iniciando con la elaboración de la base de datos en el software SPSS, continuando con el desarrollo del código mediante librerías de Python tales como Pandas, Theano, TensorFlow, Keras y Matplotlib que sirven tanto para la manipulación del conjunto de datos como para la elaboración y optimización de la red, los resultados obtenidos evalúan la efectividad de nuestro modelo tanto en la fase de entrenamiento como de prueba.

Palabras clave: **Redes Neuronales Artificiales, clasificación multiclase, Keras, TensorFlow**

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

LISTA DE SÍMBOLOS

Acrónimos

ADALINE	ADAptative LINear Elements
ANN	Artificial Neural Network
API	Application Program Interface
ART	Adaptive Resonance Theory
BSD	Berkeley Software Distribution
CNTK	Cognitive Toolkit
CPU	Central Processing Unit
DIMM	Dual In-line Memory Module
DSS	Decision Support System
GPU	Graphics Processing Unit
LISA	Local Integrated Software Architecture
MLP	Multi-Layer Perceptron
RAM	Random Access Memory
RBF	Radial Basis Function
SSD	Solid State Drive
TB	Turbo Bost

CAPÍTULO 1

ASPECTOS INTRODUCTORIOS

1.1. Antecedentes

1.1.1. Historia de las Redes Neuronales Artificiales

Alan Turing en 1936 fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas (Un Cálculo Lógico de la Inminente Idea de la Actividad Nerviosa - Boletín de Matemática Biofísica 5: 115-133). Ellos modelaron una red neuronal simple mediante circuitos eléctricos. (Matich, 2001)

Donald Hebb en 1949 fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Aun hoy, este es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal. Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También intentó encontrar semejanzas entre el aprendizaje y la actividad nerviosa. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales. (Matich, 2001)

A partir de 1986, el panorama fue alentador con respecto a las investigaciones y el desarrollo de las redes neuronales. En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen (sobre todo en el área de control) y las empresas que lanzan al mercado productos nuevos, tanto hardware como software Matich (2001). A continuación se presentan algunos autores con sus proyectos de investigación que fueron tomados en cuenta para la realización de esta monografía:

María G. Longoni, Eduardo A. Porcel, María V. López y Gladys N. Dapozo de la Universidad Nacional del Nordeste (Longoni y cols., 2010), trabajaron en el artículo titulado *Modelos de Redes Neuronales Perceptrón Multicapa y de Base Radial para la predicción del rendimiento académico de alumnos universitarios*. Concluyeron que la técnica de RNA ha mostrado en general una buena capacidad clasificatoria, mediante los modelos MLP y RBF orientados a la predicción del rendimiento académico de los alumnos ingresantes a la FACENA-UNNE en función de sus características socio-educativas.

Eduardo A. Porcel, María V. López y Gladys N. Dapozo de la Universidad Nacional del Nordeste (Porcel y cols., 2011), trabajaron en el artículo titulado *Predicción del rendimiento académico de alumnos de primer año de universidad mediante redes neuronales*. Concluyeron que el empleo de la técnica de redes neuronales ha permitido obtener porcentajes elevados de predicción, superiores al obtenido con la regresión logística binaria aplicada al mismo conjunto de datos en estudios anteriores. Además, pudo observarse que ambas técnicas coinciden en la detección de variables significativas en la estimación del rendimiento académico.

María V. López, Maria G. Ramirez Arballo, Eduardo A. Porcel, Liliana E. Mata y Silvia E. Barreto de la Universidad Nacional Nordeste (López y cols., 2012), trabajaron en el artículo titulado *Redes Neuronales para predecir el rendimiento académico de los alumnos ingresantes a la carrera de Bioquímica de la FACENA-UNNE en función de sus conocimientos matemáticos previos*. Concluyeron que los modelos de tipo Perceptrón Multicapa (PM) y Función Base Radial (FBR) desarrollados constituyen un instrumento idóneo, como lo demuestran los altos porcentajes de clasificación correcta obtenidos.

Nick Z. Zacharis de Technological Educational Institute of Piraeus (Zacharis, 2016), trabajó en el artículo titulado *Predicting Student Academic Performance in Blended Learning using Artificial Neural Networks.* Concluyó que las redes neuronales superan todos los demás clasificadores, con respecto a la precisión de la predicción. Además existe una fuerte evidencia de que el modelo propuesto se puede utilizar de forma eficaz para predecir el rendimiento de los estudiantes en el curso y ayudar al instructor a diseñar intervenciones oportunas que aumenten las posibilidades de éxito.

Mubarak Albarka Umar de Changchun University of Science and Technology (Umar, 2019), trabajó en el artículo titulado *Student Academic Performance Prediction using Artificial Neural Networks: A Case Study*. Concluyó que el modelo de redes neuronales se puede integrar en el proceso de admisión del instituto como sistema de apoyo a la toma de decisiones (DSS) para facilitar y simplificar el proceso de admisión y asegurar la admisión de más calificados candidatos, basados en ciertos atributos del rendimiento escolar de su secundaria, que puede graduarse con éxito a tiempo.

1.2. Introducción

El rendimiento académico de los estudiantes universitarios se ve influenciado por varios factores, que están ligados principalmente por características socioeducativas, los cuales afectan de manera importante el desempeño de los mismos, ya que son determinantes en la preparación del alumno durante toda su trayectoria académica. (López y cols., 2012)

En este estudio se analizó el rendimiento académico de los alumnos de la Facultad de Ciencias y Tecnología, como una estrategia para la identificación temprana de elementos de riesgo, además se brindará las alternativas de acción oportunas que permitan implementar medidas correctivas en el proceso educativo. También se aplicarán técnicas matemáticas que favorecen la comprensión de la variabilidad relacionada con las razones de deserción y los indicadores de desempeño del educando, esto permite medir y cuantificar estas variables, creando la posibilidad de obtener predicciones a través de las Redes Neuronales Artificiales que estimen el rendimiento futuro de los estudiantes universitarios aprovechando el software Python y sus distintas librerías que sirven para desarrollar este tipo de modelo.

Contenido de los capítulos

El presente trabajo monográfico está dividido en 6 capítulos:

El primer capítulo describe los aspectos generales: introducción y contenido de los capítulos, antecedentes, planteamiento del problema, justificación y los objetivos de la investigación (general y específicos).

El segundo capítulo contiene el marco teórico que incluye: fundamentos de las Redes Neuronales Artificiales, retropropagación y funciones de activación.

En el tercer capítulo se describe la metodología, tipo de investigación, etapas del proyecto: Recolección de la información, selección de herramientas a usar, desarrollo del modelo matemático, detección y clasificación, por último la evaluación y valoración.

En el cuarto capítulo se presentan los resultados obtenidos en lenguaje Python y el software SPSS.

Las conclusiones y recomendaciones para trabajos futuros se presentan en el quinto capítulo y los anexos en el sexto capítulo. Al final se presenta la bibliografía consultada.

1.3. Objetivos

Objetivo general

Aplicar clasificación multiclase con Redes Neuronales Artificiales para datos académicos de los estudiantes que ingresaron a las carreras de Licenciatura en Matemática, Ingeniería en Estadística, Ingeniería en Telemática, Licenciatura en Biología, Licenciatura en Química y Licenciatura en Ciencias Actuariales en el plan de estudio 2011-2018.

Objetivos específicos

- Crear un clasificador en el lenguaje de programación Python utilizando las librerías Pandas, Theano, TensorFlow, Keras y Mathplotlib.
- Comparar resultados generados mediante el modelo programado en Python con los obtenidos en el software SPSS.

CAPÍTULO 2

MARCO TEÓRICO

2.1. Neurona Biológica y Neurona Artificial

2.1.1. Neurona Biológica

Según Jiménez (2012) la neurona biológica, célula constituyente del cerebro, fue descubierta en 1836 y su estructura de varias entradas / una salida les supuso a sus descubridores, Camillo Golgi y Santiago Ramón y Cajal, el Premio Nobel de Medicina en 1906. En el cerebro humano hay aproximadamente diez mil millones de neuronas (10^{10} células nerviosas). Cada neurona tiene un cuerpo celular (soma), que tiene un núcleo celular. A partir del cuerpo de la neurona se ramifican una cantidad de fibras llamadas dendritas y una única fibra larga llamada axón. El axón mide en general 100 veces más que el diámetro del cuerpo de la célula (aproximadamente 1 cm). Los axones y dendritas transmiten las señales entre neuronas: el axón permite enviar señales a otras neuronas mientras que las dendritas permiten la recepción de dichas señales. Una neurona se conecta con un número variable de neuronas (entre 100 y 100000), formando un conjunto de conexiones sinápticas. Se estima que hay unas 10^{14} sinapsis en el cerebro de un adulto.

Las señales se propagan entre neuronas mediante una reacción electroquímica: a nivel de la sinapsis, la neurona que genera la señal (que puede ser excitadora o inhibitoria) emite unos neurotransmisores que activan o desactivan los receptores de la neurona que recibe la señal, cuando se supera un umbral. La excitación se propaga rápidamente entre neuronas gracias a su gran conectividad. En la Figura 2.1 se puede apreciar la morfología de una neurona biológica. La regla de aprendizaje de Hebb indica que ocurre un cambio metabólico en la sinapsis cuando la entrada de una neurona está siendo activada repetidamente de manera persistente, reduciendo la resistencia de la sinapsis. Los huecos presentes en las sinapsis y que se encargan de atenuar las señales de entrada de una neurona, cambian de tamaño en respuesta al “aprendizaje”.

La red neuronal del cerebro forma un sistema masivo de procesamiento paralelo de información, en contraste con las computadoras en las cuales un único procesador ejecuta linealmente una serie de instrucciones. Por otra parte, la velocidad de procesamiento representa una diferencia importante. Mientras que el cerebro opera a unos 100 Hz

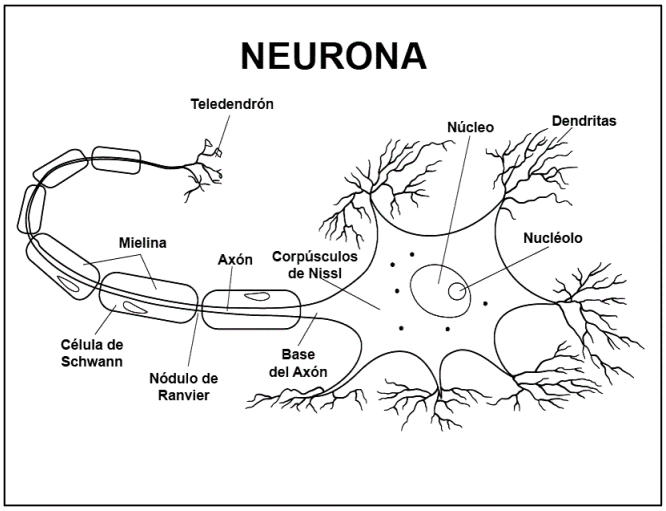


Figura 2.1: Esquema de una neurona biológica, adaptado de Esquema de la neurona, 2021, Esquema.net (https://esquema.net/neurona/).

(100 operaciones por segundo), una computadora convencional procesa varias centenas de millones de operaciones por segundo. A pesar del “hardware” lento con el que está construido, el cerebro posee una serie de capacidades notables:

- Su rendimiento tiende a degradarse lentamente bajo daño parcial. En contraste, la mayoría de los programas y sistemas son frágiles: si se daña alguna parte arbitrariamente, es muy probable que el sistema entero deje de funcionar.
- Puede aprender (reorganizarse) a partir de la experiencia.
- Es tolerante a fallos. La recuperación de los daños es posible si unidades sanas logran reemplazar a las unidades dañadas, aprendiendo y asumiendo funciones de éstas.
- Ejecuta una cantidad masiva de cálculos en paralelo de forma extremadamente eficiente. Por ejemplo, la percepción visual compleja, que equivale a 10 etapas de procesamiento, ocurre en menos de 100 milisegundos.
- Sustenta a la inteligencia y la conciencia. La ciencia todavía no ha logrado determinar cómo ocurre esto exactamente.

2.1.2. Neurona Artificial

La neurona artificial según Jiménez (2012) trata de capturar la esencia de los sistemas neuronales biológicos e imitar su comportamiento. Una neurona artificial se puede describir de la siguiente manera:

- Recibe una serie de "entradas" (sean datos originales o "salidas" de otras neuronas). Cada entrada llega a través de una conexión que tiene una cierta "fuerza", o "peso", que equivale a la eficiencia sináptica de una neurona biológica. Cada neurona tiene también un determinado valor de umbral. La suma ponderada de las entradas mas el valor de umbral componen la "activación" de la neurona (también conocida como Potencial Post-Sináptico (PPS) de la neurona).
- La señal de activación pasa a través de una función de activación, o función de transferencia, para producir la "salida" de la neurona. Esta función limita el rango de valores que puede tomar la variable de salida de la neurona.

Por lo tanto, el nivel de actividad de cada neurona es función de las entradas que recibe, y el resultado se envía como una señal a través de sus conexiones con otras neuronas. Cada neurona sólo puede dar un valor de salida al mismo tiempo. Una neurona artificial puede estar implementada por medio de componentes hardware o ser simulada por medio de software en un ordenador.

2.2. Redes Neuronales Artificiales

Definición 2.1 "Las Redes Neuronales Artificiales (ANN) son sistemas conexionistas formados por numerosas unidades de proceso (neuronas) altamente conectadas entre sí, que adaptan su estructura mediante técnicas de aprendizaje para resolver problemas de aproximación de funciones y clasificación de patrones. Procesan la información contenida en un conjunto de datos que les son suministrados, bien para obtener relaciones entre los mismos y la función objetivo que se pretende aproximar, o bien clasificando estos datos en diferentes categorías". (Matich, 2001)

"Las ANN, gracias a su habilidad para extraer significado de datos complicados o imprecisos, pueden ser usadas para extraer patrones y determinar tendencias que son muy complejas como para poder ser detectadas tanto por humanos como por otras técnicas informáticas. Puede considerarse a una red neuronal entrenada como un "experto" en la categoría de información a la que ha sido asignada para analizar. Este experto puede ser utilizado entonces para proveer proyecciones dadas nuevas situaciones". (Jiménez, 2012)

2.2.1. Ventajas que ofrecen las Redes Neuronales Artificiales

Las Redes Neuronales Artificiales según Matich (2001) ofrecen las siguientes ventajas:

- **Aprendizaje Adaptativo.** Capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.
- **Auto-organización.** Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- **Tolerancia a fallos.** La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.
- **Operación en tiempo real.** Los cómputos neuronales pueden ser realizados en paralelo; para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.
- **Fácil inserción dentro de la tecnología existente.** Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes.

2.2.2. Arquitectura de las Redes Neuronales Artificiales

En el siguiente apartado, se tratará de el estudio de las redes neuronales artificiales, así como múltiples arquitecturas, desde las más simples hasta las más complejas.

Perceptrón simple

Un perceptrón simple o neuronas según Rosenblatt (1958) es el elemento matemático básico en una red neuronal artificial, su funcionamiento está inspirado en la simulación de una neurona biológica.

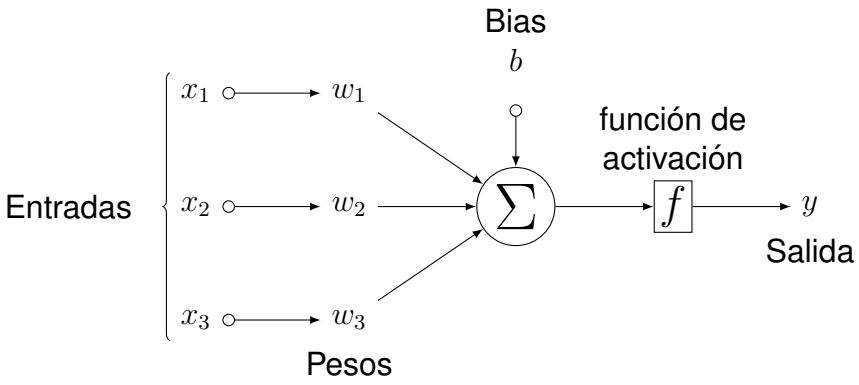


Figura 2.2: Perceptrón simple, adaptado de Inteligencia artificial con aplicaciones a la ingeniería (p. 204), por P. Ponce, 2010, Alfaomega Grupo Editor, S.A. de C.V., México.

En la figura se presentan todos los elementos que componen una neurona.

- Señales de entrada (x_i): son los datos de entrada a la neurona con los que se alimentan. Estos datos son los que utilizará la red para realizar el fenómeno de aprendizaje.
- Pesos sinápticos (w_{jk}) : a cada conexión de la red neuronal se le asigna un peso sináptico, este constituye el principal recurso de memoria a largo plazo, usualmente el aprendizaje se realiza con la actualización de tales pesos.
- Unión sumadora $(\sum)$: se define como la suma ponderada de la multiplicación de las entradas x_i y los pesos w_{jk}.
- Estado de activación: cada neurona se caracteriza en cualquier instante por un valor numérico denominado estado o nivel de activación $a_j(t)$.
- Función de activación (φ): determina el nuevo estado de activación de la neurona teniendo en cuenta la entrada total calculada y el anterior estado de activación, por lo general todas las unidades de la red utilizan la misma función activación.
  En la mayoría de los casos la función identidad, por lo que el estado anterior no se tiene en cuenta, principalmente las tres funciones de activación más usadas son: la función escalonada (ReLu), la función lineal o la función sigmoidal.
- Bias (b_j) : es un tipo de constante que funciona si los valores de entrada netos superan un cierto umbral, la neurona devolverá una señal de activación positiva, y si no lo supera devolverá una señal negativa. Esta constante se define inicialmente, no viene dada en el conjunto de datos.
- Función de salida (y_i) : transforma el estado de activación actual en una señal de salida y_i, dicha señal puede ser de la última neurona o enviarse a otras neuronas.

Perceptrón multicapa.

El perceptrón multicapa (multilayer perceptron) o más adelante red neuronal artificial surgió en 1986 con el algoritmo de la retropropagación (backpropagation).

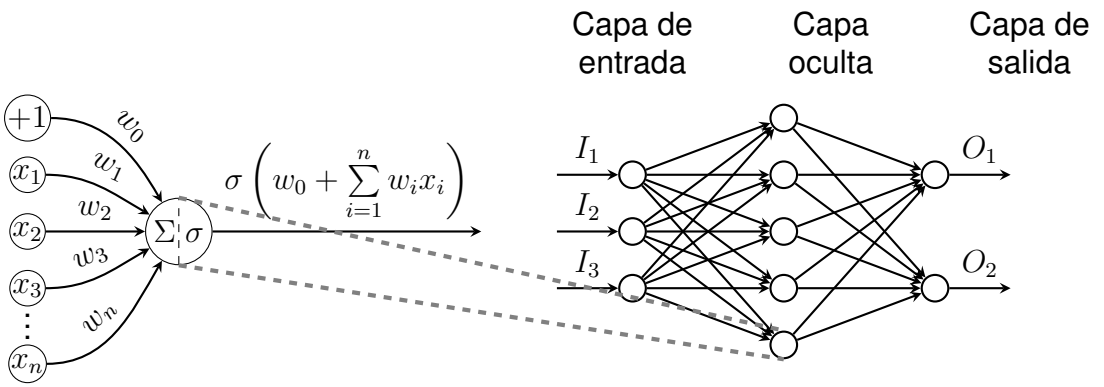


Figura 2.3: Representación esquemática de un perceptrón, adaptado de Sistema de recocimiento de captcha alphanuméricos usando redes neuronales (p. 27), por L. Quintero, 2020.

Principalmente, una red neuronal es la concatenación de un número determinado de perceptrones o neuronas organizadas en tres tipos de capas:

- Capas de entrada (input layer): Es el conjunto de neuronas que funcionan como entrada a la red, es decir, no procesan información, solo se introducen datos.
- Capas ocultas (hidden layer): Es el conjunto de neuronas donde se reciben datos de la capa $i-1$ y se procesan las salidas hacia la capa $i+1$.
- Capas de salida (output layer): Es el conjunto de neuronas cuyos valores de salida corresponde a la salida general de toda la red.

2.2.3. Backpropagation

El algoritmo de la retropropagación (backpropagation) según Nielsen (2015) se introdujo originalmente en la década de 1970, pero su importancia no se apreció por completo hasta un famoso artículo de 1986 de David Rumelhart, Geoffrey Hinton y Ronald Williams. Ese artículo describe varias redes neuronales en las que la propagación hacia atrás funciona mucho más rápido que los enfoques anteriores de aprendizaje, lo que hace posible el uso de redes neuronales para resolver problemas que antes eran insolubles. Hoy en día, el algoritmo de la retropropagación (Backpropagation) es el caballo de batalla del aprendizaje en redes neuronales.

En el corazón de la propagación hacia atrás hay una expresión para la derivada parcial $\frac{\partial C}{\partial w}$ de la función de costo C con respecto a cualquier peso w (o al sesgo b) en la red. La expresión nos dice qué tan rápido cambia el costo cuando cambiamos los pesos y los sesgos. Y aunque la expresión es algo compleja, también tiene una belleza, y cada elemento tiene una interpretación natural e intuitiva, entonces, la propagación hacia atrás no es solo un algoritmo rápido para el aprendizaje, en realidad, nos brinda información detallada sobre cómo cambiar los pesos y los sesgos modifica el comportamiento general de la red.

2.2.4. Calentamiento: un enfoque rápido basado en matrices para calcular la salida de una Red Neuronal Artificial

Comenzaremos con una notación que nos permita referirnos a pesos en la red de una manera inequívoca, usaremos w^l_{jk} para denotar el peso de la conexión de la k-ésima neurona en la $(l-1)$-ésima capa a la j-ésima neurona en la l-ésima capa; por ejemplo, el diagrama a continuación muestra el peso en una conexión desde la cuarta neurona en la segunda capa a la segunda neurona en la tercera capa de una red:

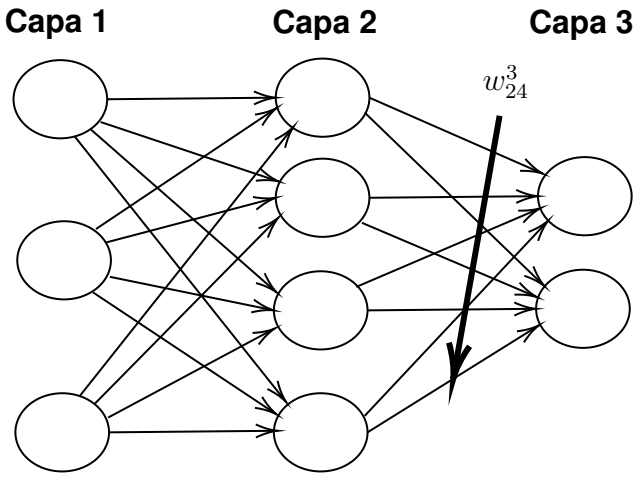


Figura 2.4: Representación de los pesos en una Red Neuronal Artificial, adaptado de Neural Networks and Deep Learning (p. 40), por M. Nielsen, 2015.

Usamos una notación similar para los sesgos y activaciones de la red, explícitamente, usamos b^l_j para el sesgo de la j-ésima neurona en la l-ésima capa, y usamos a^l_j para la activación de la j-ésima neurona en la l-ésima capa. El siguiente diagrama muestra ejemplos de estas notaciones en uso:

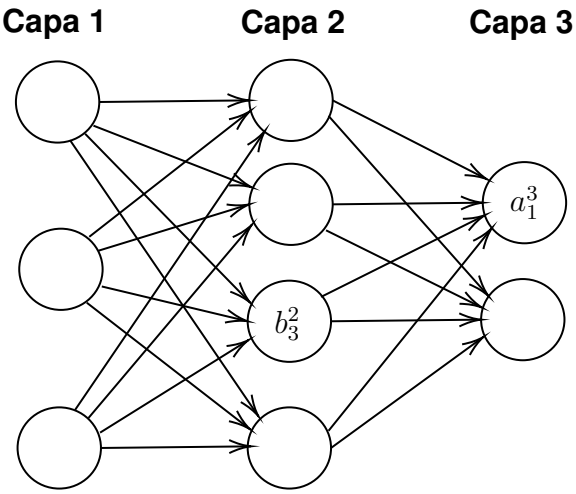


Figura 2.5: Representación del sesgo en una Red Neuronal, adaptado de Neural Networks and Deep Learning (p. 40), por M. Nielsen, 2015.

Con estas notaciones, la activación a_j^l de la neurona j-ésima en la capa l-ésima está relacionada con las activaciones en la capa $(l-1)$-ésima por la ecuación

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right), \tag{1}$$

donde la suma está sobre todas las neuronas k en la $(l-1)$ ésima capa, para reescribir esta expresión en forma de matriz, definimos una matriz de peso w^l para cada capa, l. Las entradas de la matriz de ponderaciones w^l son simplemente las ponderaciones que se conectan a la l-ésima capa de neuronas, es decir, la entrada en la j-ésima fila y la k-ésima columna es w_{jk}^l, de manera similar, para cada capa l definimos un vector de sesgo, b^l. Veamos cómo funciona esto: los componentes del vector de sesgo son solo los valores b_j^l, un componente para cada neurona en la l-ésima capa, y finalmente, definimos un vector de activación cuyos componentes son las activaciones a_j^l.

Por último lo que necesitamos para reescribir (1) en forma de matriz es vectorizar una función σ a cada elemento en un vector v. Usamos la notación $\sigma(v)$ para denotar este tipo de aplicación por elementos de una función, es decir, los componentes de $\sigma(v)$ son solo $\sigma(v)_j = \sigma(v_j)$.

Como ejemplo, si tenemos la función $f(x) = x^2$ entonces la forma vectorizada de f tiene el efecto

$$f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} f(2) \\ f(3) \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \end{bmatrix}, \tag{2}$$

es decir, el f vectorizado simplemente eleva al cuadrado cada elemento del vector, con estas notaciones en mente, la ecuación (1) se puede reescribir en la hermosa y compacta forma vectorizada

$$a^l = \sigma(w^l a^{l-1} + b^l). \tag{3}$$

Esta expresión nos da una forma mucho más global de pensar sobre cómo las activaciones en una capa se relacionan con las activaciones en la capa anterior: simplemente aplicamos la matriz de peso a las activaciones, luego agregamos el vector de sesgo y finalmente aplicamos la función σ. Esa visión global es a menudo más fácil y concisa (¡e involucra menos índices!) que la visión neurona por neurona que hemos adoptado hasta ahora; la expresión también es útil en la práctica, porque la mayoría de las bibliotecas de matrices proporcionan formas rápidas de implementar la multiplicación de matrices, la suma de vectores y la vectorización.

Cuando usamos la ecuación (3) para calcular a^l, calculamos la cantidad intermedia $z^l = w^l a^{l-1} + b^l$ a lo largo del camino, esta cantidad resulta ser lo suficientemente útil como para que valga la pena nombrarla: llamamos z^l a la entrada ponderada de las neuronas en la capa l, haremos un uso considerable de la entrada ponderada z^l, más adelante en esta sección la ecuación (3) a veces se escribe en términos de la entrada ponderada como $a^l = \sigma(z^l)$. También vale la pena señalar que z^l tiene componentes $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$, es decir, z^l_j es solo la entrada ponderada a la función de activación para la neurona j en la capa l.

2.2.5. Las dos suposiciones que necesitamos sobre la función de costo

El objetivo de la retropropagación es calcular las derivadas parciales $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$ de la función de costo C con respecto a cualquier peso w o sesgo b en la red. Para que la propagación hacia atrás funcione, necesitamos hacer dos suposiciones principales sobre la forma de la función de costo. Sin embargo, antes de establecer esas suposiciones, es útil tener en mente un ejemplo de función de costo. Usaremos la función de costo cuadrático, el cual tiene la siguiente forma.

$$C = \frac{1}{2n} \sum_x \left\| y(x) - a^l(x) \right\|^2, \tag{4}$$

donde: n es el número total de ejemplos de entrenamiento; La suma es sobre ejemplos de entrenamiento individuales, x; $y = y(x)$ es la salida deseada correspondiente; l denota el número de capas en la red; y $a^l = a^l(x)$ es el vector de salida de activaciones de la red cuando se ingresa x.

¿Qué suposiciones debemos hacer sobre nuestra función de costo, C, para que se pueda aplicar la propagación hacia atrás? La primera suposición que necesitamos es

que la función de costo se puede escribir como un promedio de $C = \frac{1}{n}\sum_x C_x$ sobre las funciones de costo C_x para ejemplos de entrenamiento individuales, x. Este es el caso de la función de costo cuadrático, donde el costo de un solo ejemplo de entrenamiento es $C_x = \frac{1}{2}|y - a^l|^2$.

La razón por la que necesitamos esta suposición es porque la propagación hacia atrás realmente nos permite hacer es calcular las derivadas parciales $\frac{\partial C_x}{\partial w}$ y $\frac{\partial C_x}{\partial b}$ para un solo ejemplo de entrenamiento. A continuación, recuperamos $\frac{\partial C}{\partial w}$ y $\frac{\partial C}{\partial b}$ promediando los ejemplos de entrenamiento. De hecho, supondremos que el ejemplo de entrenamiento x se ha corregido, y eliminaremos el subíndice x, escribiendo el costo C_x como C.

La segunda suposición que hacemos sobre el costo es que se puede escribir en función de las salidas de la red neuronal:

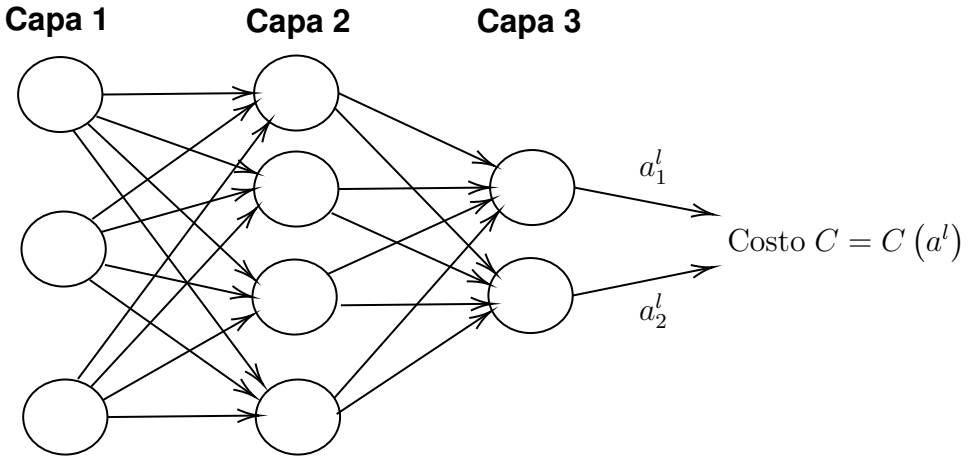


Figura 2.6: Representación de costo en una Red Neuronal, adaptado de Neural Networks and Deep Learning (p. 42), por M. Nielsen, 2015.

Por ejemplo, la función de costo cuadrático satisface este requisito, ya que el costo cuadrático para un solo ejemplo de entrenamiento x puede escribirse como

$$C = \frac{1}{2}\left\|y - a^l\right\|^2 = \frac{1}{2}\sum_j \left(y_j - a_j^l\right)^2, \tag{5}$$

y por tanto es una función de las activaciones de salida. Por supuesto, esta función de costo también depende de la salida deseada y, y quizás se pregunte por qué no consideramos el costo también como una función de y, sin embargo, recuerde que el ejemplo de entrenamiento de entrada x es fijo, por lo que la salida y también es un parámetro fijo.

2.2.6. El producto Hadamard, $s \odot t$

El algoritmo de la retropropagación se basa en operaciones algebraicas lineales comunes, como la suma de vectores, la multiplicación de un vector por una matriz etc, pero una de las operaciones se utiliza con menos frecuencia. En particular, suponga que s y t son dos vectores de la misma dimensión. Luego usamos $s \odot t$ para denotar el producto elemento a nivel de los dos vectores. Por tanto, los componentes de $s \odot t$ son solo $(s \odot t)_j = s_j t_j$. Como ejemplo,

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}. \tag{6}$$

Este tipo de multiplicación por elementos a veces se denomina *producto Hadamard* o *producto Schur*. Nos referiremos a él como el producto Hadamard, las buenas bibliotecas matriciales generalmente proporcionan implementaciones rápidas del producto Hadamard, y eso es útil cuando se implementa la propagación hacia atrás.

2.2.7. Las cuatro ecuaciones fundamentales detrás de la retropropagación

La retropropagación se trata de comprender cómo cambiar los pesos y los sesgos en una red, con respecto a la función de costo. En última instancia, esto significa calcular las derivadas parciales $\frac{\partial C}{\partial w^l_{jk}}$ y $\frac{\partial C}{\partial b^l_j}$, pero para calcularlos, primero introducimos una cantidad intermedia δ^l_j que llamamos *error* en la neurona j-ésima en la capa l-ésima. La retropropación nos dará un procedimiento para calcular el error δ^l_j, y luego relacionará δ^l_j con $\frac{\partial C}{\partial w^l_{jk}}$ y $\frac{\partial C}{\partial b^l_j}$, definimos el error δ^l_j de la neurona j en la capa l por

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}. \tag{7}$$

De acuerdo con nuestras convenciones habituales, usamos δ^l para denotar el vector de errores asociados con la capa l. La retropropagación nos dará una forma de calcular δ^l para cada capa, y luego relacionar esos errores con las cantidades de interés real $\frac{\partial C}{\partial w^l_{jk}}$ y $\frac{\partial C}{\partial b^l_j}$.

Así que nos quedaremos con $\delta^l_j = \frac{\partial C}{\partial z^l_j}$ como nuestra medida de error.

Una ecuación para el error[1] en la capa de salida δ^l

[1]En problemas de clasificación como MNIST, el término 'error' a veces se utiliza para referirse a la tasa de fallos de clasificación. Por ejemplo, si la red neuronal clasifica correctamente el 96.0 por ciento de los dígitos, entonces el error es 4.0 por ciento, esto tiene un significado bastante diferente de nuestros vectores δ.

Los componentes de δ^l están dados por:

$$\delta^l_j = \frac{\partial C}{\partial a^l_j}\sigma'(z^l_j) \tag{BP1}$$

el primer término a la derecha, $\frac{\partial C}{\partial a^l_j}$, solo mide qué tan rápido cambia el costo en función de la j-ésima activación de salida, el segundo término a la derecha, $\sigma'(z^l_j)$, mide qué tan rápido cambia la función de activación σ en z^l_j.

En particular, calculamos z^l_j mientras calculamos el comportamiento de la red, y es solo una pequeña sobrecarga adicional calcular $\sigma'(z^l_j)$, la forma exacta de $\frac{\partial C}{\partial a^l_j}$ dependerá, por supuesto, de la forma de la función de costo, sin embargo, siempre que se conozca la función de costo, no debería haber problemas para calcular $\frac{\partial C}{\partial a^l_j}$. Por ejemplo, si usamos la función de costo cuadrático, $C = \frac{1}{2}\sum_j (y_j - a^l_j)^2$, y entonces $\frac{\partial C}{\partial a^l_j} = (a^l_j - y_j)$.

La ecuación (BP1) es una expresión de componentes para δ^l, pero no la forma basada en matrices que queremos para la retropropagación. Sin embargo, es fácil reescribir la ecuación en forma matricial, como

$$\delta^l = \nabla_a C \odot \sigma'(z^l) \tag{BP1a}$$

Aquí, $\nabla_a C$ se define como un vector cuyos componentes son las derivadas parciales $\frac{\partial C}{\partial a^l_j}$. Puede pensar en $\nabla_a C$ como la expresión de la tasa de cambio de C con respecto a las activaciones de salida. Es fácil ver que las ecuaciones (BP1a) y (BP1) son equivalentes, y por esa razón de ahora en adelante usaremos (BP1) indistintamente para referirse a ambas ecuaciones. Como ejemplo, en el caso del costo cuadrático tenemos $\nabla_a C = (a^l - y)$, por lo que la forma matricial de (BP1) se convierte en

$$\delta^l = (a^l - y) \odot \sigma'(z^l). \tag{8}$$

Una ecuación para el error δ^l en términos del error en la siguiente capa δ^{l+1}

En particular

$$\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'(z^l), \tag{BP2}$$

donde $\left(w^{l+1}\right)^T$ es la transposición de la matriz de pesos w^{l+1} para la (l + 1)-ésima

capa, supongamos que conocemos el error δ^{l+1} en la (l + 1)-ésima capa, cuando aplicamos la transposición a la matriz de pesos, $(w^{l+1})^T$, podemos pensar intuitivamente en esto como mover el error *hacia atrás* a través de la red, dándonos algún tipo de medida del error en la salida de la l-ésima capa. Luego tomamos el producto de Hadamard $\odot\ \sigma'(z^l)$, esto mueve el error hacia atrás a través de la función de activación en la capa l, dándonos el error δ^l en la entrada ponderada de la capa l.

Combinando (BP2) con (BP1) podemos calcular el error δ^l para cualquier capa de la red. Comenzamos usando (BP1) para calcular δ^l, luego aplicamos la ecuación (BP2) para calcular δ^{l-1}, luego la ecuación (BP2) nuevamente para calcular δ^{l-2}, y así sucesivamente, todo el camino de regreso a través de la red.

Una ecuación para la tasa de cambio del costo con respecto a cualquier sesgo en la red

En particular

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{BP3}$$

es decir, el error δ_j^l es *exactamente igual* a la tasa de cambio $\frac{\partial C}{\partial b_j^l}$, por (BP1) y (BP2) podemos reescribir (BP3) en forma abreviada como

$$\frac{\partial C}{\partial b} = \delta \tag{9}$$

donde se entiende que δ se evalúa en la misma neurona que el sesgo b.

Una ecuación para la tasa de cambio del costo con respecto a cualquier peso en la red

En particular

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

esto nos dice cómo calcular las derivadas parciales $\frac{\partial C}{\partial w^l_{jk}}$ en términos de las cantidades δ^l y a^{l-1}. Sin embargo, la ecuación se puede reescribir en una notación de índice menos pesado como

$$\frac{\partial C}{\partial w} = a_{\text{in}} \delta_{\text{out}}, \tag{10}$$

donde se entiende que a_{in} es la activación de la entrada de la neurona al peso w, y δ_{out} es el error de la salida de la neurona del peso w.

Las ecuaciones de la retropropagación

$$\delta^l = \nabla_a C \odot \sigma'\left(z^l\right) \tag{$BP1$}$$

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l) \tag{$BP2$}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{$BP3$}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{$BP4$}$$

El nombre de retropropagación resulta de la forma en que el error es propagado hacia atrás a través de la red neuronal, en otras palabras el error se propaga hacia atrás desde la capa de salida. Esto permite que los pesos sobre las conexiones de las neuronas ubicadas en las capas ocultas cambien durante el entrenamiento.

El cambio de los pesos en las conexiones de las neuronas además de influir sobre la entrada global, influye en la activación y por consiguiente en la salida de una neurona. Por lo tanto, es de utilidad considerar las variaciones de la función de activación al modificarse el valor de los pesos, esto se llama sensibilidad de la función de activación, de acuerdo al cambio en los pesos. (Matich, 2001)

2.2.8. Derivación de las reglas básicas de la retropropagación

La retropropagación, en esencia, consiste simplemente en aplicar repetidamente la regla de la cadena a través de todos los caminos posibles en nuestra red. Sin embargo, hay un número exponencial de rutas dirigidas desde la entrada hasta la salida. El poder real de la retropropagación surge en forma de un algoritmo de programación dinámica, donde reutilizamos resultados intermedios para calcular el gradiente.

Para aprender cómo funcionan las matemáticas en la retropropagación, comenzaremos con una red simple de una ruta y luego pasaremos a una red con varias unidades por capa. Finalmente, derivaremos el algoritmo de retropropagación general.

En la explicación de la derivación de las reglas básicas del algoritmo de retropropagación, es conveniente utilizar una notación que nos permita referirnos a los pesos en la red de una manera simple de comprender. Usaremos $w_{i\to j}$, donde i se refiere a la neurona de entrada, j a la neurona de salida y $\to$ a la conexión de ambas neuronas, por lo tanto $w_{i\to j}$ denota el peso de la conexión de la i-ésima neurona a la j-ésima neurona.

Para que podamos actualizar las ponderaciones de forma incremental se utiliza el descenso de gradiente :

$$w_{i\to j} = w_{i\to j} - \eta \frac{\partial C}{\partial w_{i\to j}}$$

Para una sola unidad en una red general, podemos tener varios casos: la unidad puede tener una sola entrada y una salida (caso 1), la unidad puede tener varias entradas (caso 2), o la unidad puede tener varias salidas (caso 3). Técnicamente hay un cuarto caso: una unidad puede tener varias entradas y salidas. Pero como veremos, el caso de entrada múltiple y el caso de salida múltiple son independientes, y simplemente podemos combinar las reglas que aprendemos para el caso 2 y el caso 3 para este caso.

Caso 1: Entrada única y salida única

Supongamos que tenemos la siguiente red simple de una ruta:

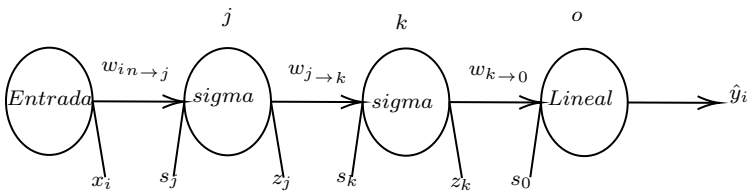


Figura 2.7: Única entrada y única salida.

Explícitamente los valores de entrada de la red (2.7) vienen dados por:

$$
\begin{aligned}
z_j &= w_1 \cdot x_i \\
a_j &= \sigma(in_j) = \sigma(w_1 \cdot x_i) \\
z_k &= w_2 \cdot z_j \\
a_k &= \sigma(in_k) = \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
z_o &= w_3 \cdot z_k \\
\hat{y}_i &= in_o = w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) \\
C &= \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(w_3 \cdot \sigma(w_2 \cdot \sigma(w_1 \cdot x_i)) - y_i)^2
\end{aligned}
$$

Encontrando las derivadas, y viendo si se puede deducir un patrón que podríamos explotar en un algoritmo iterativo. En primer lugar, vamos a encontrar la derivada para $w_{k\to o}$ (recordando que $\hat{y} = w_{k\to o} z_k$ ya que nuestra salida es una unidad lineal).

$$
\begin{aligned}
\frac{\partial C}{\partial w_{k\to o}} &= \frac{\partial}{\partial w_{k\to o}} \frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= \frac{\partial}{\partial w_{k\to o}} \frac{1}{2}(w_{k\to o} \cdot z_k - y_i)^2 \\
&= (w_{k\to o} \cdot z_k - y_i) \frac{\partial}{\partial w_{k\to o}} (w_{k\to o} \cdot z_k - y_i) \\
&= (\hat{y}_i - y_i)(z_k)
\end{aligned}
$$

Encontrando la actualización de los pesos para $w_{i\to k}$:

$$
\begin{aligned}
\frac{\partial C}{\partial w_{j\to k}} &= \frac{\partial}{\partial w_{j\to k}} \frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= (\hat{y}_i - y_i) \left(\frac{\partial}{\partial w_{j\to k}} (w_{k\to o} \cdot \sigma(w_{j\to k} \cdot z_j) - y_i) \right) \\
&= (\hat{y}_i - y_i)(w_{k\to o}) \left(\frac{\partial}{\partial w_{j\to k}} \sigma(w_{j\to k} \cdot z_j) \right) \\
&= (\hat{y}_i - y_i)(w_{k\to o}) \left(\sigma(s_k)(1 - \sigma(s_k)) \frac{\partial}{\partial w_{j\to k}} (w_{j\to k} \cdot z_j) \right) \\
&= (\hat{y}_i - y_i)(w_{k\to o}) \left(\sigma(s_k)(1 - \sigma(s_k)) \right) (z_j).
\end{aligned}
$$

Una vez más, encontrar la actualización de peso para $w_{i\to j}$ el cálculo se vuelve sencillo.

$$\begin{aligned}
\frac{\partial C}{\partial w_{i\to j}} &= \frac{\partial}{\partial w_{i\to j}}\frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= (\hat{y}_i - y_i)\left(\frac{\partial}{\partial w_{i\to j}}(\hat{y}_i - y_i)\right) \\
&= (\hat{y}_i - y_i)(w_{k\to o})\left(\frac{\partial}{\partial w_{i\to j}} \cdot \sigma(w_{j\to k} \cdot \sigma(w_{i\to j} \cdot x_i))\right) \\
&= (\hat{y}_i - y_i)(w_{k\to o})(\sigma(s_k)(1-\sigma(s_k)))(w_{j\to k})\left(\frac{\partial}{\partial w_{i\to j}}\sigma(w_{i\to j} \cdot x_i)\right) \\
&= (\hat{y}_i - y_i)(w_{k\to o})(\sigma(s_k)(1-\sigma(s_k)))(w_{j\to k})(\sigma(s_j)(1-\sigma(s_j)))(x_i)
\end{aligned}$$

Ahora se puede observar un patrón el cual se puede codificar con la retropropagación. Específicamente, vemos la derivada del error de la red, el derivado ponderado de la unidad K y el derivando con respecto a la unidad j. Por tanto en resumen para la red de la figura 2.7 tenemos:

$$\begin{aligned}
\Delta w_{i\to j} &= -\eta\left[(\hat{y}_i - y_i)(w_{k\to o})(\sigma(s_k)(1-\sigma(s_k)))(w_{j\to k})(\sigma(s_j)(1-\sigma(s_j)))(x_i)\right] \\
\Delta w_{j\to k} &= -\eta\left[(\hat{y}_i - y_i)(w_{k\to o})\left(\sigma(s_k)(1-\sigma(s_k))(z_j)\right]\right. \\
\Delta w_{k\to o} &= -\eta\left[(\hat{y}_i - y_i)(z_k)\right]
\end{aligned}$$

Caso 2: Múltiples entradas.

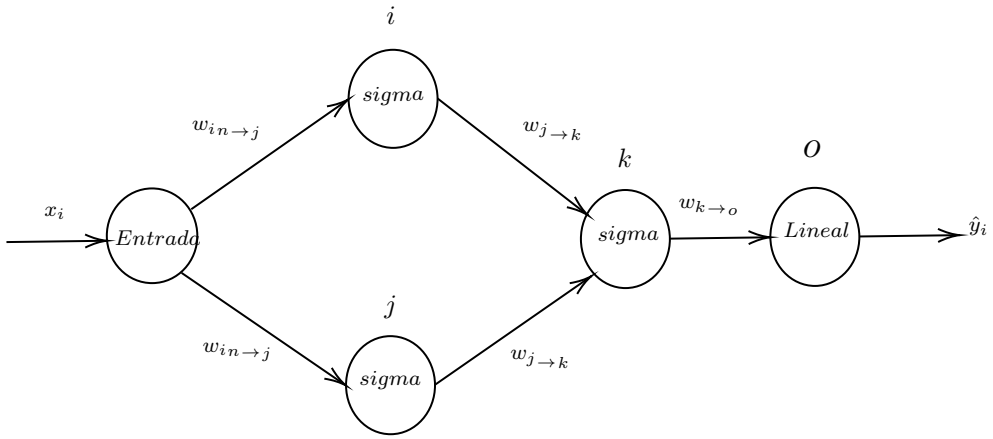


Figura 2.8: Manejo de múltiples entradas.

La regla de actualización de los pesos $w_{i\to k}$ viene dada por la derivada de $w_{i\to k}$

$$\begin{aligned}
\frac{\partial C}{w_{i\to k}} &= \frac{\partial}{w_{i\to k}}\frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= (\hat{y}_i - y_i)\left(\frac{\partial}{w_{i\to k}} z_k w_{k\to o}\right) \\
&= (\hat{y}_i - y_i)(w_{k\to o})\left(\frac{\partial}{w_{i\to k}}\sigma\left(s_k\right)\right) \\
&= (\hat{y}_i - y_i)(\sigma(s_k)(1-\sigma(s_k))w_{k\to o})\left(\frac{\partial}{w_{i\to k}}\left(z_i w_{i\to k} + z_j w_{j\to k}\right)\right) \\
&= (\hat{y}_i - y_i)(\sigma(s_k)(1-\sigma(s_k))w_{k\to o})z_i
\end{aligned}$$

Aquí vemos que la actualización para $w_{i\to k}$ no depende de $w_{i\to k}$ derivado, que conduce a nuestra primera regla, así que podemos actualizar pesos en la misma capa de forma aislada. Hay un orden natural en las actualizaciones que sólo dependen de los valores de otros pesos en la misma capa.

Caso 3: Manejo de múltiples salidas

Ahora vamos a examinar el caso donde una unidad oculta tiene más de una salida.

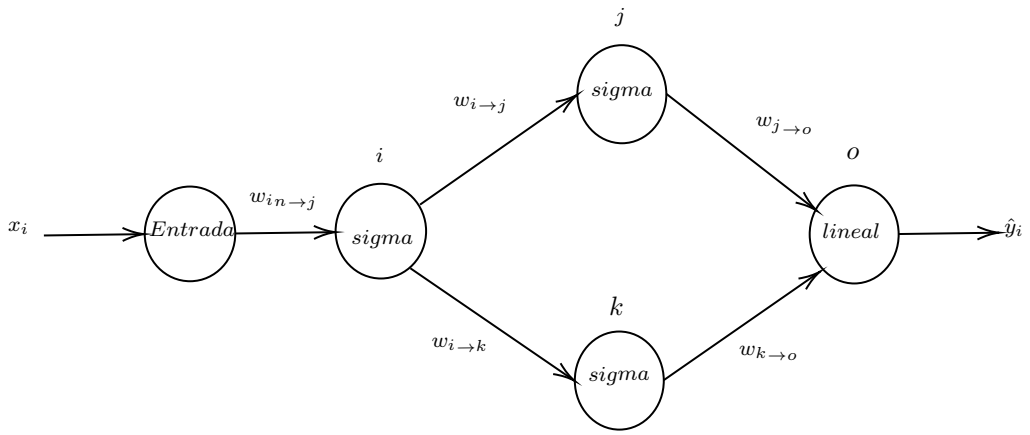


Figura 2.9: Manejo de múltiples salidas.

Sobre la base de los casos anteriores, el único tipo ”nuevo”de actualización de peso es la derivada de $w_{in\to j}$ la diferencia en el caso de salida múltiple es que la unidad i tiene más de un sucesor inmediato, por lo que debemos sumar el error acumulado a lo largo de todos los caminos que están arraigados en la unidad i. Vamos a derivar explícitamente la actualización de peso para $w_{in\to i}$ (para hacer un seguimiento de lo que está pasando, definimos $\sigma_i(\cdot)$ como la función de activación de la unidad i.

$$
\begin{aligned}
\frac{\partial C}{w_{in\to i}} =& \frac{\partial}{w_{in\to i}} \frac{1}{2}(\hat{y}_i - y_i)^2 \\
&= (\hat{y}_i - y_i) \left(\frac{\partial}{w_{in\to i}} (z_j w_{j\to o} + z_k w_{k\to o}) \right) \\
&= (\hat{y}_i - y_i) \left(\frac{\partial}{w_{in\to i}} (\sigma_j(s_j) w_{j\to o} + \sigma_k(s_k) w_{k\to o}) \right) \\
&= (\hat{y}_i - y_i) \left(w_{j\to o}\sigma'_j(s_j) \frac{\partial}{w_{in\to i}} s_j + w_{k\to o}\sigma'_k(s_k) \frac{\partial}{w_{in\to i}} s_k \right) \\
&= (\hat{y}_i - y_i) \left(w_{j\to o}\sigma'_j(s_j) \frac{\partial}{w_{in\to i}} z_i w_{i\to j} + w_{k\to o}\sigma'_k(s_k) \frac{\partial}{w_{in\to i}} z_i w_{i\to k} \right) \\
&= (\hat{y}_i - y_i) \left(w_{j\to o}\sigma'_j(s_j) \frac{\partial}{w_{in\to i}} \sigma_i(s_i) w_{i\to j} + w_{k\to o}\sigma'_k(s_k) \frac{\partial}{w_{in\to i}} \sigma_i(s_i) w_{i\to k} \right) \\
&= (\hat{y}_i - y_i) \left(w_{j\to o}\sigma'_j(s_j) w_{i\to j}\sigma'_i(s_i) \frac{\partial}{w_{in\to i}} s_i + w_{k\to o}\sigma'_k(s_k) w_{i\to k}\sigma'_i(s_i) \frac{\partial}{w_{in\to i}} s_i \right) \\
&= (\hat{y}_i - y_i) \left(w_{j\to o}\sigma'_j(s_j) w_{i\to j}\sigma'_i(s_i) + w_{k\to o}\sigma'_k(s_k) w_{i\to k}\sigma'_i(s_i) \right) x_i
\end{aligned}
$$

Señales de error

La señal de error es simplemente el error acumulado en cada unidad. Por ahora, vamos a considerar la contribución de una sola instancia de entrenamiento (por lo que usaremos $\hat{y}$ en lugar de $\hat{y}_i$). Definimos la señal de error recursiva en la unidad j como

$$\delta_j = \frac{\partial C}{\partial s_j}. \tag{2.1}$$

En términos generales, es una medida de cuánto varía el error de red con la entrada a la unidad j el uso de la señal de error tiene algunas propiedades agradables a saber, podemos reescribir la retropropagación en una forma más compacta. Para ver esto, vamos a expandir δ_j

$$
\begin{aligned}
\delta_j &= \frac{\partial C}{\partial s_j} \\
&= \frac{\partial}{\partial s_j} \frac{1}{2}(\hat{y} - y)^2 \\
&= (\hat{y} - y) \frac{\partial \hat{y}}{\partial s_j}
\end{aligned}
$$

consideremos el caso en el que j es una neurona de salida, esto significa que $\hat{y} = f_j(s_j)$ (si unidad j es la función de activación entonces $f_j(\cdot)$), así que $\frac{\partial \hat{y}}{\partial s_j}$ es simple-

mente $f'_j(s_j)$, dándonos $\delta_j = (\hat{y} - y)f'_j(s_j)$, en caso contrario, si la unidad j es una neurona oculta que conduce a otra capa de neuronas $k \in \text{outs}(j)$ podemos expandir $\frac{\partial \hat{y}}{\partial s_j}$, además, utilizando la regla de cadena obtenemos:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial s_j} &= \frac{\partial \hat{y}}{\partial z_j}\frac{\partial z_j}{\partial s_j} \\ &= \frac{\partial \hat{y}}{\partial z_j} f'_j(s_j)\end{aligned}$$

ahora la $\frac{\partial \hat{y}}{\partial z_j}$ indica las múltiples unidades que dependen z_j específicamente, todas las unidades $k \in \text{outs}(j)$ en la sección sobre varias salidas que un peso que conduce a una unidad con múltiples salidas tiene un efecto en esas unidades de salida, pero para cada unidad k, tenemos $s_k = z_j w_{j\to k}$, con cada s_k no dependiendo de ningún otro s_k, por lo tanto, podemos usar la regla de cadena de nuevo y sumar sobre los nodos de salida $k \in \text{outs}(j)$:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial s_j} &= f'_j(s_j) \sum_{k\in\text{outs}(j)} \frac{\partial \hat{y}}{\partial s_k}\frac{\partial s_k}{\partial z_j} \\ &= f'_j(s_j) \sum_{k\in\text{outs}(j)} \frac{\partial \hat{y}}{\partial s_k} w_{j\to k}\end{aligned}$$

conectando esta ecuación con la función $\delta_j = (\hat{y} - y)\frac{\partial \hat{y}}{\partial s_j}$, obtenemos

$$\delta_j = (\hat{y} - y)f'_j(s_j) \sum_{k\in\text{outs}(j)} \frac{\partial \hat{y}}{\partial s_k} w_{j\to k} \tag{2.2}$$

basándonos en nuestra definición de la señal de error, sabemos que $\delta_k = (\hat{y} - y)\,\frac{\partial \hat{y}}{\partial s_k}$, así que si introducimos a $(\hat{y} - y)$ en la suma, obtenemos que la relación recursiva

$$\delta_j = f'_j(s_j) \sum_{k\in\text{outs}(j)} \delta_k w_{j\to k} \tag{2.3}$$

ahora tenemos una representación compacta del error de la retropropagación, lo último que hay que hacer es juntar todo con un algoritmo general.

Forma general de la retropropagación

Recordando la red simple del primer caso 2.2 tenemos:

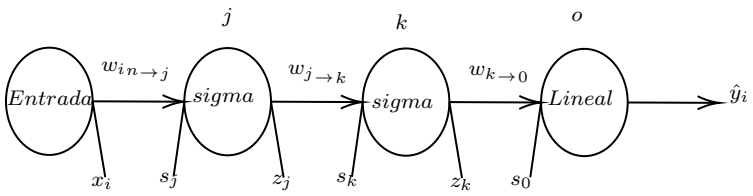


Figura 2.10: única entrada y única salida.

Podemos usar la definición de δ_i para derivar los valores de todas las señales de error en la red:

$$\delta_o = (\hat{y} - y) \text{ (La derivada de una función lineal)}$$
$$\delta_k = \delta_o w_{k\to o}\sigma(s_k)(1-\sigma(s_k))$$
$$\delta_j = \delta_k w_{j\to k}\sigma(s_j)(1-\sigma(s_j))$$

Recordar que las actualizaciones explícitas del peso para esta red eran de la forma:

$$\begin{aligned}
\Delta w_{i\to j} &= -\eta\left[(\hat{y}_i - y_i)(w_{k\to o})(\sigma(s_k)(1-\sigma(s_k)))(w_{j\to k})(\sigma(s_j)(1-\sigma(s_j)))(x_i)\right] \\
\Delta w_{j\to k} &= -\eta\left[(\hat{y}_i - y_i)(w_{k\to o})\,(\sigma(s_k)(1-\sigma(s_k))(z_j)\right] \\
\Delta w_{k\to o} &= -\eta\left[(\hat{y}_i - y_i)(z_k)\right]
\end{aligned}$$

sustituyendo cada una de las señales de error obtenemos:

$$\begin{aligned}
\Delta w_{k\to o} &= -\eta\delta_o z_k \\
\Delta w_{j\to k} &= -\eta\delta_k z_j \\
\Delta w_{i\to j} &= -\eta\delta_j x_i
\end{aligned}$$

luego observando la red más complicada de la sección sobre el manejo de múltiples salidas:

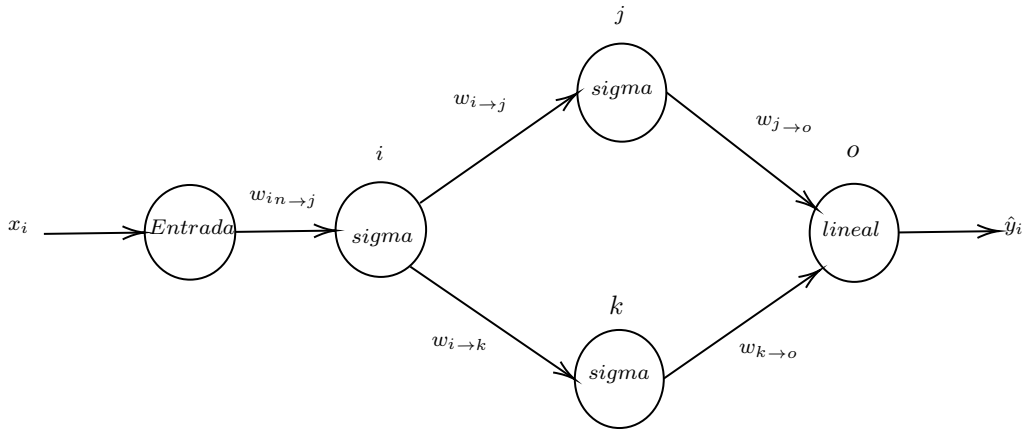


Figura 2.11: Manejo de múltiples salidas.

Podemos volver a derivar todas las señales de error:

$$\begin{aligned}
\delta_o &= (\hat{y} - y) \\
\delta_k &= \delta_o w_{k\to o}\sigma(s_k)(1-\sigma(s_k)) \\
\delta_j &= \delta_o w_{j\to o}\sigma(s_j)(1-\sigma(s_j)) \\
\delta_i &= \sigma(s_i)(1-\sigma(s_i)) \sum_{k\in \text{outs}(i)} \delta_k w_{i\to k}
\end{aligned}$$

aunque no derivamos todas estas actualizaciones de peso a mano, mediante el uso de las señales de error, las actualizaciones de peso se convierten (y se puede comprobar esto a mano, si lo desea):

$$\begin{aligned}
\Delta w_{k\to o} &= -\eta\delta_o z_k \\
\Delta w_{j\to o} &= -\eta\delta_o z_j \\
\Delta w_{i\to k} &= -\eta\delta_k z_i \\
\Delta w_{i\to j} &= -\eta\delta_j z_i \\
\Delta w_{in\to i} &= -\eta\delta_i x_i
\end{aligned}$$

De forma general las derivadas de las actualizaciones de los pesos, viene dada por $\nabla w_{i\to j} = -\eta\delta_j z_i$.

Lo último a tener en cuenta es el caso en el que usamos un minilote de instancias para calcular el descenso, porque tratamos a cada uno y_i como independientes, sumamos todas las instancias de entrenamiento para calcular la actualización completa para un peso (normalmente escalamos por el tamaño del minilote N para que los pasos no sean sensibles a la magnitud de N). Para cada instancia de entrenamiento independiente y_i, añadimos un superíndice (y_i) a los valores que cambian para cada ejemplo de entrenamiento:

$$\nabla w_{i\to j} = -\frac{\eta}{N}\sum_{y_i} \delta_j^{(y_i)} z_i^{(y_i)}$$

por lo tanto, la forma general del algoritmo de la retropropagación para actualizar las ponderaciones consta de los siguientes pasos:

1) Alimente las instancias de entrenamiento a través de la red y registre cada $s_j^{(y_i)}$ y $z_j^{(y_i)}$.

2) Calcule las señales de error $\delta_j^{(y_i)}$ para todas la unidades j y cada ejemplo de entrenamiento y_i. Si j es una neurona de salida, entonces $\delta_j^{(y_i)} = f_j'(s_j^{(y_i)})(\hat{y}_i - y_i)$. Si j no es una neurona de salida, entonces $\delta_j^{(y_i)} = f_j'(s_j^{(y_i)}) \sum_{k\in \text{outs}(j)} \delta_k^{(y_i)} w_{j\to k}$.

3) Actualizar las ponderaciones con la regla $\nabla w_{i\to j} = -\frac{\eta}{N}\sum_{y_i} \delta_j^{(y_i)} z_i^{(y_i)}$. (Dolhansky, 2014)

2.3. Funciones de activación

2.3.1. Función costo

Definición 2.2 (Función costo) "La función costo trata de determinar el error entre el valor estimado y el valor real, con el fin de optimizar los parámetros de la red neuronal"(Calvo, 2018).

Estimadores de error de la función costo

Error medio cuadrático – RMSE

El Error Cuadrático Medio según Heras (2020) es el criterio de evaluación más usado para problemas de regresión. Se usa sobre todo cuando usamos aprendizaje automático supervisado. Para cada dato histórico podremos indicar el resultado correcto. Vamos a ver como se calcula.

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2$$

Características del MSE:

- Calculamos el error al cuadrado, en lugar del error simple, para que el error siempre sea positivo.
- Es una función derivable

Una forma de generalizar las funciones discriminantes lineales es aplicar a la suma una función no lineal σ, llamada función de activación. En el caso de la clasificación en dos clases queda

$$y = \sigma(w_{ij}x_i + b_j)$$

en general se usa una función monótona. Por lo tanto la expresión anterior se puede seguir considerando una función discriminante lineal, dado que las fronteras de decisión siguen siendo lineales. Para esta unidad de procesamiento, la función de transferencia es el resultado de componer la función discriminante lineal con la función de activación. (Calvo, 2018)

2.3.2. Función Sigmoide

La función sigmoide transforma los valores introducidos a una escala $(0, 1)$, donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0 (Calvo, 2018).

La importancia de la función sigmoide es que su derivada siempre es positiva y cercana a cero para los valores grandes positivos o negativos; además, toma su valor

máximo cuando $x = 0$. Esto hace que se puedan utilizar reglas de aprendizaje definidas para las funciones escalón, con la ventaja, respecto a esta función, de que la derivada está definida en todo el intervalo.

$$f_k(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

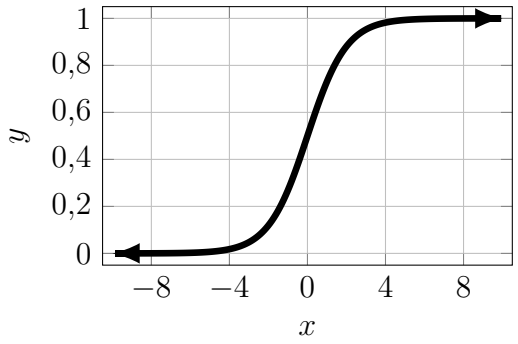


Figura 2.12: Función Sigmoide, adaptado de Función de activación- Redes Neuronales, por D. Calvo, 2015, Diego Calvo (https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/).

Algunas de sus propiedades son:

- $f_k : \mathbb{R} \longrightarrow (0, 1)$
- $x \to -\infty \quad \Leftrightarrow \quad y \to 0$
- $x = 0 \quad \Leftrightarrow \quad y = 0{,}5$
- $x \to \infty \quad \Leftrightarrow \quad y \to 1$
- Dado que para todo $x \in \mathbb{R}$ se tiene que $e^{-x} \neq -1$ entonces la función no se indefine en ningún punto.

Determinemos la derivada de la Función Sigmoide:

$$\begin{aligned}
\frac{d}{d_x} f_k &= \frac{d}{d_x} \frac{1}{1+e^{-x}} \\
&= \frac{d}{d_x} (1+e^{-x})^{-1} \\
&= (1+e^{-x})^{-2} \frac{d}{d_x} (1+e^{-x}) \\
&= -(1+e^{-x})^{-2} \frac{d}{d_x} e^{-x} \\
&= -(1+e^{-x})^{-2} e^{-x} \frac{d}{d_x}(-x) \\
&= \frac{e^{-x}}{(1+e^{-x})^2}
\end{aligned}$$

Concretizando:

$$f'_k = \frac{e^{-x}}{(1+e^{-x})^2} \tag{2.5}$$

Nótese que $e^{-x} \neq -1$ para todo valor de x. por lo tanto es posible calcular la derivada de f_k en cualquier punto x, con lo cual concluimos que efectivamente es diferenciable en el intervalo $(-\infty, \infty)$.

2.3.3. Función Tangente Hiperbólica

La función tangente hiperbólica transforma los valores introducidos a una escala $(-1, 1)$, donde los valores altos tienden de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1 (Calvo, 2018).

$$f_k(x) = \tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{2.6}$$

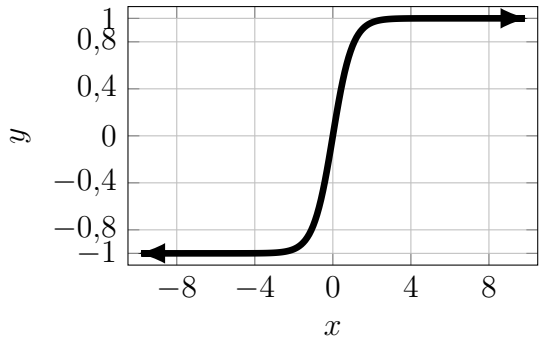


Figura 2.13: Función Tangente Hiperbólica, adaptado de Función de activación- Redes Neuronales, por D. Calvo, 2015, Diego Calvo (https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/).

Algunas de sus propiedades son:

- $f_k : \mathbb{R} \longrightarrow (-1, 1)$
- $x \to -\infty \quad \Leftrightarrow \quad y \to -1$
- $x = 0 \quad \Leftrightarrow \quad y = 0$
- $x \to \infty \quad \Leftrightarrow \quad y \to 1$
- $e^{-2x} \neq -1, \forall x \in \mathbb{R}$

La derivada de la Tangente Hiperbólica está dada por:

$$f'_k = \frac{4e^{2x}}{(1 + e^{2x})^2} \tag{2.7}$$

Tenemos que su denominador no se indefine, por lo tanto la función Tangente Hiperbólica es diferenciable en $(-\infty, \infty)$

2.3.4. Función ReLU

La función ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran (Calvo, 2018).

$$f_k(x) = \text{máx}(0, x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{2.8}$$

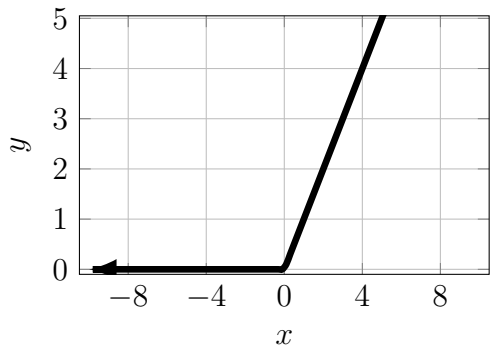


Figura 2.14: Función ReLU, adaptado de Función de activación- Redes Neuronales, por D. Calvo, 2015, Diego Calvo (https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/).

2.3.5. Función de activación Unidad Lineal Rectificada con fuga (Rectified Linear Unit-Leaky)

La función de activación Unidad Lineal Rectificada con fuga transforma los valores introducidos multiplicando los negativos por un coeficiente rectificativo y dejando los positivos según entran (Calvo, 2018).

$$f_k(x) = \text{máx}(0, x) = \begin{cases} x & \text{si } x \geq 0 \\ 0,01x & \text{si } x < 0 \end{cases} \tag{2.9}$$

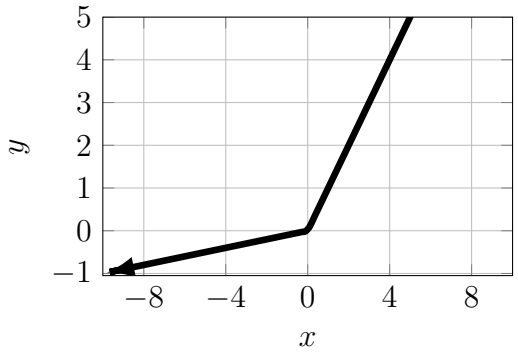


Figura 2.15: Función de activación Unidad Lineal Rectificada con fuga (Rectified Linear Unit-Leaky), adaptado de Función de activación- Redes Neuronales, por D. Calvo, 2015, Diego Calvo (https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/).

2.3.6. Función Sobrealisada (softplus)

La función sobrealisada (softplus) es más reciente que la sigmoide y la tanh, produce salidas en escala de $(0, \infty)$ (Serengil, 2017). Las funciones de activación sobrealisada (sotfplus) se consideran la version suavisada de las transformaciones por rectificación lineal.

$$f_k(x) = \ln(1 + e^x) \tag{2.10}$$

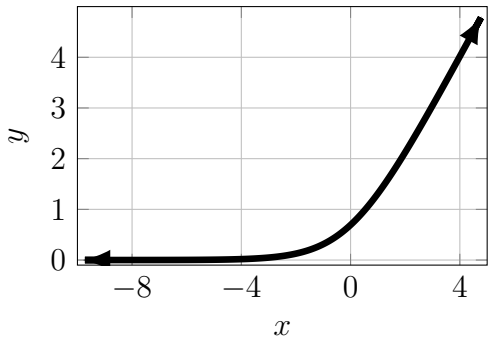


Figura 2.16: Función sobrealisada (softplus), adaptado de Función de activación- Redes Neuronales, por D. Calvo, 2015, Diego Calvo (https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/).

CAPÍTULO 3

DISEÑO METODOLÓGICO

3.1. Etapas del Proyecto

El tipo de investigación en este estudio es proyectiva ya que, consiste en la elaboración de una propuesta, un plan, un programa o un modelo, como solución a un problema o necesidad de tipo práctico, ya sea de un grupo social, o de una institución, o de una región geográfica, en un área particular del conocimiento, a partir de un diagnóstico preciso de las necesidades del momento, los procesos explicativos o generadores involucrados y de las tendencias futuras, es decir, con base en los resultados de un proceso investigativo (Hurtado de Barrera, 1998). En general en la investigación se aplica el método científico, como técnica observacional. Para cumplir con los objetivos propuestos seguiremos el siguiente esquema dividido por etapas:

Etapa I: Recolección de la información

- Fuente de información secundaria: Secretaría de la Facultad de Ciencias y Tecnología en los registros de notas.
- Unidad de estudio: Estudiantes matriculados en las carreras de Licenciatura en Matemática, Ingeniería en Estadística, Ingeniería en Telemática, Licenciatura en Biología, Licenciatura en Química y Licenciatura en Ciencias Actuariales en el plan de estudio 2011-2018. Obteniendo como población total 2348 registros en dicho periodo.
- Variables: La información fue proporcionada en PDF correspondiente a los certificados de notas que contiene las siguientes variables; nombre, carnet, carrera, año de estudio, plan de estudio, componentes curriculares, ciclo, crédito, nota final, letras, promedio (índice académico); algunas de estas variables serán tomadas en cuenta en la etapa de desarrollo del modelo matemático.

Etapa II: Selección de herramientas a usar

En este proyecto se utilizó librerías como Pandas con la que se podrá manipular el conjunto de datos (dataset), además Theano, TensorFlow y Keras las cuales son de relevancia para desarrollar redes neuronales; el algoritmo se desarrolló en el lenguaje Python de alto nivel, multiparadigma e interpretado el cual se concentra en la simpleza y la efectividad de los procedimientos, hace que Python sea un lenguaje elegante y el tipado dinámico hacen que sea un ideal para el desarrollo rápido de toda aplicación. (Python Software Foundation, 2020)

Los modelos matemáticos utilizados en TensorFlow son redes neuronales, que en función de la arquitectura de capas y neuronas que la conforman se podrá modelizar desde un simple modelo de regresión hasta una arquitectura mucho más compleja de aprendizaje automático (machine learning) Tutorials Point (2017); Todo dependerá del nivel de dificultad del problema en cuestión.

Etapa III: Desarrollo del modelo matemático

Se definen los procedimientos para implementar el desarrollo del sistema planteado, los cuales serán los siguientes:

1. Construir el dataset: Se analizarón 2348 registros, que representa la cantidad de estudiantes de primer ingreso de dicho período de las carreras mencionadas, usaremos el programa estadístico SPSS v22 para gestión, depuración y procesamiento de una base de datos, con la información de interés para aplicar el estudio, quedando las siguientes variables, de las cuales a continuación se presenta una pequeña descripción (si así se requiere) y se etiquetará de manera apropiada para facilitar el manejo de estas:

    - Número de carnet: Es el identificador único de cada estudiante, los primeros dos números corresponden al año de ingreso, los cinco siguientes al número de prematrícula y el último a la sede.
    - Suma acumulada de créditos: Se obtiene al sumar los créditos obtenidos por aprobar los componentes curriculares (nota mínima 60), los valores de esta variable están en el intervalo [0,39] para un año académico.
    - Rendimiento académico: Esta variable será la salida de nuestro modelo supervisado y se codificará de la siguiente manera:

      2: Bueno (39 créditos)
      1: Regular (20-38 créditos)
      0: Malo (0-19 créditos)
    - Sexo:

      0: Mujer 1: Hombre

- Año de ingreso:
  1: 2011 2: 2012 3: 2013 4: 2014
  5: 2015 6: 2016 7: 2017 8: 2018
- Carrera:
  1: Ing. en Telemática 2: Ing. en Estadística 3: Lic. en Matemática
  4: Lic. en Ciencias Actuariales 5: Lic. en Química 6: Lic. en Biología

2. Se construyó el modelo basado en redes neuronales con clasificación multiclase para realizar las predicciones.

Etapa IV: Detección y clasificación

En esta etapa, se entrenó el modelo el cual consiste en realizar varias repeticiones de las entradas de datos, para que la red vaya modificando sus parámetros. A estas repeticiones se les llama epoch y también tendremos que fijar el número que haremos de estas; cuantas más hagamos, más durará el entrenamiento total, pero mejor entrenada estará la red. Habrá que encontrar un equilibrio entre ambas partes para la predicción del rendimiento académico (variable objetivo) y así poder evaluar el funcionamiento del modelo al momento de realizar las clasificaciones.

Etapa V: Evaluación y valoración

En esta etapa, se evaluó el desempeño del modelo que realiza las predicciones del rendimiento académico, el porcentaje de aciertos y la confiabilidad. Se realizará predicciones sobre los datos de entrenamiento para ver el ajuste del modelo durante el entrenamiento.

CAPÍTULO 4

RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados que se obtuvieron a través de la clasificación del rendimiento académico utilizando la técnicas explicadas en el capítulo anterior con el objetivo de crear un clasificador en el lenguaje de programación Python.

4.1. Código escrito en el lenguaje de programación Python que realiza una clasificación multiclase

A continuación se muestra el código elaborado en el lenguaje Python.

Para poder importar el conjunto de datos se utilizó la librería Pandas, así mismo se declaró las variables de entrada x y de salida y.

```
import pandas as pd
dataset = pd.read_csv("Base_de_datos.csv")
x= dataset[['Anio_de_ingreso','Suma_acumulada_de_creditos','sexo','Carrera']]
y= dataset['Rendimiento']
```

Se separó el total de datos, asignando $80\,\%$ al conjunto de entrenamiento, el $20\,\%$ restante para el conjunto de prueba y $random\_state = 0$ evita la aleatoriedad.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,
                                                    random_state = 0)
```

StandardScaler realiza la tarea de Estandarización. Por lo general, un conjunto de datos contiene variables que son diferentes en escala. Se aplica el estandarizado al conjunto de variables de entrada, tanto al subconjunto de entrenamiento como de prueba

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Con la librería Tensorflow se creó una red neuronal con una capa de entrada, una oculta y una de salida, en estas últimas se hizo uso de la función de activación relu, la cual resulta efectiva en nuestro problema de clasificación.

```
from tensorflow import keras
from tensorflow.keras import layers
clasificador = keras.Sequential(
    [layers.Dense(4, activation='relu'),
     layers.Dense(1, activation='relu')])
```

Luego de creado el objeto “clasificador” se compila la red, para esto debemos especificar principalmente la función de error y el optimizador siendo en este caso el error cuadrático medio y el gradiente descendente estocástico (El descenso de gradiente estocástico es un algoritmo de aprendizaje iterativo que utiliza un conjunto de datos de entrenamiento para actualizar un modelo) respectivamente.

```
clasificador.compile(loss='mean_squared_error', optimizer='sgd',
metrics=['mse','mae','mape','accuracy'])
```

Se procedió a entrenar la red utilizando los conjuntos de entrenamiento, el tamaño del lote (batch size) es un hiperparámetro del descenso del gradiente que controla la cantidad de muestras de entrenamiento para trabajar antes de que se actualicen los parámetros internos del modelo. El número de épocas (epochs) es un hiperparámetro de descenso de gradiente que controla el número de pases completos a través del conjunto de datos de entrenamiento.

```
Entrenamiento = clasificador.fit(x_train, y_train, batch_size=1, epochs=100)
clasificador.summary()
```

El resultado del entrenamiento es el siguiente:

```
Epoch 1/100
1878/1878 [==============================] - 2s 915us/step - loss: 0.4024 - accuracy: 0.5783
Epoch 2/100
1878/1878 [==============================] - 2s 902us/step - loss: 0.1785 - accuracy: 0.6507
Epoch 3/100
1878/1878 [==============================] - 2s 928us/step - loss: 0.1377 - accuracy: 0.6587
Epoch 4/100
1878/1878 [==============================] - 2s 905us/step - loss: 0.1234 - accuracy: 0.6656
```

...

```
Epoch 96/100
1878/1878 [==============================] - 2s 869us/step - loss: 0.0769 - accuracy: 0.6763
Epoch 97/100
1878/1878 [==============================] - 2s 880us/step - loss: 0.0768 - accuracy: 0.6778
Epoch 98/100
1878/1878 [==============================] - 2s 855us/step - loss: 0.0763 - accuracy: 0.6778
Epoch 99/100
1878/1878 [==============================] - 2s 891us/step - loss: 0.0771 - accuracy: 0.6763
Epoch 100/100
1878/1878 [==============================] - 2s 857us/step - loss: 0.0767 - accuracy: 0.6773
```

Para la época 100 la red llegó a un 67.73 % de exactitud. En una primera aproximación los resultados son aceptables. No obstante, cabe destacar que los resultados no son consistentes entre ejecuciones. En una segunda ronda del programa el entrenamiento arrojó los siguientes resultados:

```
Epoch 1/100
1878/1878 [==============================] - 2s 910us/step - loss: 0.0767 - accuracy: 0.6805
Epoch 2/100
1878/1878 [==============================] - 2s 908us/step - loss: 0.0742 - accuracy: 0.6816
Epoch 3/100
1878/1878 [==============================] - 2s 909us/step - loss: 0.0752 - accuracy: 0.6810

...

Epoch 96/100
1878/1878 [==============================] - 2s 981us/step - loss: 0.0747 - accuracy: 0.6826
Epoch 97/100
1878/1878 [==============================] - 2s 981us/step - loss: 0.0744 - accuracy: 0.6826
Epoch 98/100
1878/1878 [==============================] - 2s 1ms/step - loss: 0.0747 - accuracy: 0.6826
Epoch 99/100
1878/1878 [==============================] - 2s 980us/step - loss: 0.0743 - accuracy: 0.6826
Epoch 100/100
1878/1878 [==============================] - 2s 939us/step - loss: 0.0745 - accuracy: 0.6826
```

El motivo de esto es que la matriz de parámetros es inicializada de manera diferente en cada ejecución. Recordemos que la inicialización de los parámetros w y b determina el punto inicial a partir del cual comenzamos el proceso de descenso por gradiente estocástico. En el primer caso la inicialización fue más favorable y el algoritmo pudo acercarse al mínimo más fácilmente, mientras que en el segundo caso la inicialización no ayudó y ya no pudo mejorar la exactitud por encima del 68.26 %. Esto nos da la pauta de que una de las primeras cosas que podemos considerar para aumentar el rendimiento de la red y lograr resultados más consistentes es elegir una mejor inicialización.

Luego de haber entrenado la red se procedió a predecir con esta el conjunto de pruebas (para las pruebas se tomó el 20 % del total de datos el cual equivale a 470 casos) a la vez que los resultados obtenidos se redondean para poder evaluar los aciertos.

```
y_pred = clasificador.predict(x_test)
y_pred = y_pred.round()
```

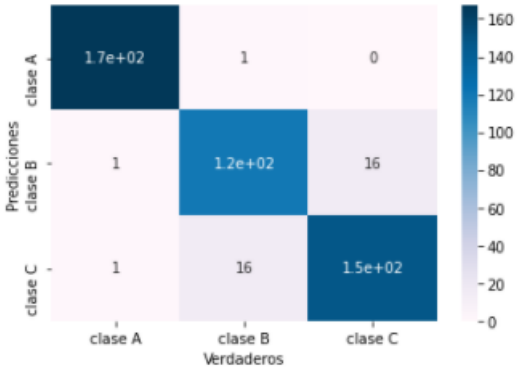
Los aciertos y desaciertos que hubieron al momento de la clasificación los obtuvimos con la matriz de confusión.

En este caso la Red Neuronal Artificial a través de la matriz de confusión muestra los siguientes valores para dos ejecuciones del código respectivamente:

```
from sklearn.metrics import confusion_matrix
mc = confusion_matrix(y_test, y_pred)
print(mc)
```

```
[[167   1   0]
 [  1 120  16]
 [  1  16 148]]
```

(a) Matriz de confusión

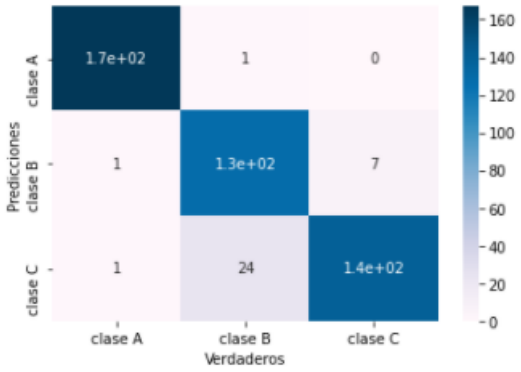


(b) Gráfica de Matriz de confusión

167 casos los clasificó correctamente en rendimiento bueno, 120 en regular y 148 en malo, sin embargo en la segunda ejecución de la red neuronal observamos que:

```
[[167   1   0]
 [  1 129   7]
 [  1  24 140]]
```

(c) Matriz de confusión



(d) Gráfica de Matriz de confusión

167 casos los clasificó correctamente en rendimiento bueno, 129 en regular y 140 en malo.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

La efectividad (Accuracy), Recall (Exhaustividad), muestra la proporción con la cual la red neuronal clasifica cada una de las categorías en la variable de salida.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	168
1	0.89	0.98	0.93	137
2	0.98	0.89	0.93	165
accuracy			0.95	470
macro avg	0.95	0.95	0.95	470
weighted avg	0.96	0.95	0.95	470

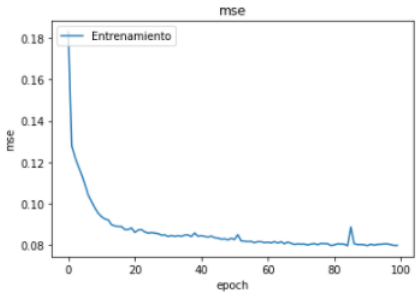
Se observa que el modelo de Redes Neuronales Artificiales tiene una precisión del 99 % al momento de clasificar los casos que poseen un rendimiento bueno, un 89 % para los casos que poseen un rendimiento regular y un 98 % para los casos que poseen un rendimiento malo.

La Exhaustividad (Recall) nos informa sobre la cantidad de casos que el modelo es capaz de identificar,es decir,el modelo identifica en un 99 % los casos con rendimiento bueno, 98 % los casos con rendimiento regular y 89 % para los casos con rendimiento malo respectivamente.

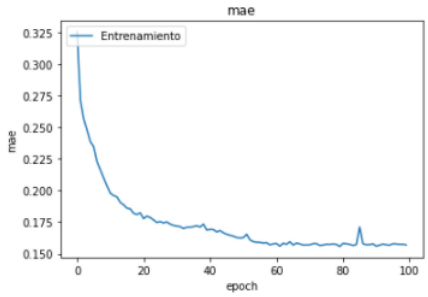
La Exactitud(Accuracy) mide el porcentaje de casos que el modelo clasificó correctamente, en este caso se obtuvo una efectividad del 95 % dicho de otra manera el modelo tiene una efectividad del 95 % en cada caso.

```
import matplotlib.pyplot as plt
plt.plot(Entrenamiento.history['mse'])
plt.title('mse')
plt.ylabel('mse')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(Entrenamiento.history['mae'])
plt.title('mae')
plt.ylabel('mae')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(Entrenamiento.history['mape'])
plt.title('mape')
```

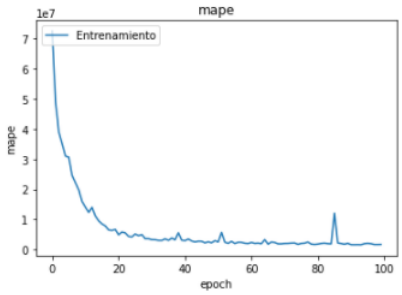
```
plt.ylabel('mape')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(Entrenamiento.history['accuracy'])
plt.title('accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



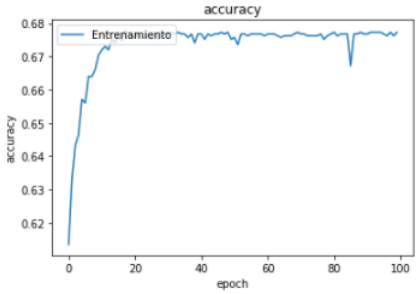
(e) Error cuadrático medio



(f) Error absoluto medio



(g) Error absoluto porcentual promedio



(h) Efectividad

Las gráficas anteriores muestran como varían las distintas métricas de regresión, es evidente que durante el entrenamiento los errores disminuyeron con cada época (epoch), mientras que la efectividad de nuestra red mejoró; lo cual indica que el modelo funciona y es efectivo a la hora de clasificar.

4.2. Resultados obtenidos mediante Software SPSS

En la construcción del modelo, se utilizó una Red Neuronal Artificial con modelo perceptrón multicapa con las siguientes variables de entrada: Carrera, Sexo, Anio_Calendario

Suma_acumulada_de_creditos, se asigno la variable de salida " Rendimiento", luego se le pidió al softaware SPSS realizar el arquetipo de manera automática mostrando la siguiente información.

Información de red

Capa de entrada	Factores	1	Carrera
		2	Sexo
		3	Anio_de_ingreso
	Covariables	1	Suma_acumulada_de_creditos
	Número de unidades[a]		17
	Método de cambio de escala para las covariables		Estandarizados
Capas ocultas	Número de capas ocultas		1
	Número de unidades en la capa oculta 1[a]		6
	Función de activación		Tangente hiperbólica
Capa de salida	Variables dependientes	1	Rendimiento
	Número de unidades		3
	Función de activación		Softmax
	Función de error		Entropía cruzada

La aplicación de las Redes Neuronales Artificiales a través de la matriz de confusión en el programa SPSS muestra los siguientes resultados:

Clasificación

		Pronosticado			
Ejemplo	Observado	R. malo	R. regular	R. bueno	Porcentaje correcto
Entrenamiento	R. malo	577	21	7	95.4%
	R. regular	14	590	10	96.1%
	R. bueno	1	0	689	99.9%
	Porcentaje global	31.0%	32.0%	37.0%	97.2%
Pruebas	R. malo	155	6	1	95.7%
	R. regular	5	103	3	92.8%
	R. bueno	0	1	165	99.4%
	Porcentaje global	36.4%	25.1%	38.5%	96.4%

Variable dependiente: Rendimiento

Para el entrenamiento el programa clasificó correctamente 577 casos en rendimiento malo con una efectividad del 95.4 %, 590 casos en rendimiento regular con una efectividad del 96.1 % y un total de 689 casos en rendimiento bueno con 99.9 % de efectividad, en el caso del entrenamiento el modelo tiene una efectividad global del

97.2 %.

Luego para las pruebas el programa clasificó 155 casos de forma correcta en rendimiento malo con una efectividad del 95.7 %, 103 casos en rendimiento regular con una efectividad del 92.8 % y 165 casos en rendimiento bueno con el 99.4 % de efectividad, para el modelo de prueba el modelo tuvo una efectividad global del 96.4 % .

4.3. Comparación de los resultados generados por el código escrito en Python respecto a los generados por el software SPSS

- Tipo de red utilizada: Ambos se caracterizan por utilizar perceptrón multicapa.
- Funciones utilizadas: En la red desarrollada en Python se definió ReLU como función de activación en las capas ocultas y en la capa de salida; la función de costo es la del error cuadrático medio. En cambio, en la red generada de manera automática por SPSS la función de activación para la capa oculta fue la tangente hiperbólica, la función softmax para la capa de salida; la función de costo fue la entrópía cruzada.
- Efectividad: La red escrita en el lenguaje de programación Python luego de varias pruebas pudo clasificar con una efectividad del 95 %. Mientras que en SPSS se logró obtener un 96.4 %.

CAPÍTULO 5

CONCLUSIONES

5.1. Conclusiones

- En este trabajo monográfico se realizó un acercamiento a los fundamentos de las Redes Neuronales Artificiales, para describir su estructura, funcionamiento y problemas relacionados a su optimización.
- Se muestra una metodología que permitió clasificar rendimientos académicos, aprovechando la capacidad de las Redes Neuronales Artificiales para resolver problemas de este tipo, además de la facilidad que ofrece el lenguaje Python con librerías tales como: Pandas que permitió manejar la base de datos construida con variables de entrada y de salida para alimentar nuestra red; Theano, Tensorflow y Keras para desarrollar la red que contó con una capa de entrada, una oculta y una de salida, en estas dos ultimas se ha utilizado la función de activación ReLU, dado que es más efectiva en nuestro problema de clasificación obteniendo así mejores resultados y matplotlib que permitió representar gráficamente todo lo elaborado por los pasos anteriores.
- Los resultados obtenidos al implementar Python resaltaron una precisión al clasificar rendimiento bueno del 99 %, 89 % al clasificar rendimiento regular y 98 % al rendimiento malo, los cuales son similares a los que arroja el software SPSS los cuales son: 99.4 % para rendimiento bueno, 92.8 % para rendimiento regular y 95.7 % para rendimiento malo.
- En este trabajo el lenguaje Python y el sotfware SPSS muestran una efectividad para clasificar del 95 % y el 96 % respectivamente.
- A nivel de la gestión de la educación superior, las RNA desarrolladas en este trabajo permiten obtener información necesaria acerca del rendimiento académico de los estudiantes ingresantes a las carreras de Licenciatura en Matemática,

Ingeniería en Estadística, Ingeniería en Telemática, Licenciatura en Biología, Licenciatura en Química y Licenciatura en Ciencias Actuariales en el plan de estudio 2011-2018, contribuyendo de este modo a orientar las políticas y estrategias institucionales para mejorar los preocupantes índices de desgranamiento, abandono y bajo rendimiento.

- Aunque el estudio realizado por nuestro grupo es preliminar, los resultados pueden considerarse como satisfactorios, constando la utilidad en el campo de las matemáticas.

5.2. Recomendaciones

1. Las autoridades pertinentes de la facultad podrían elaborar una base de datos con los atributos necesarios para futuros análisis de este tipo estudios.
2. Comparar diferentes pénsum académico con el propósito de identificar los factores que puedan afectar el bajo rendimiento académico de los estudiantes.
3. Continuar con la ejecución de este tipo de estudios en el resto de facultades de esta alma mater.

REFERENCIAS

Anaconda software distribution. (2020). Anaconda Inc. Descargado de https://docs.anaconda.com/

Calvo, D. (2018, 12). *Función de coste – Redes neuronales.* Descargado de https://www.diegocalvo.es/funcion-de-coste-redes-neuronales/

Dolhansky, B. (2014). *Artificial neural networks: Mathematics of backpropation.* http://www.briandolhansky.com/blog/2013/9/27/artificial-neural-networks-backpropagation-part-4. (Acceso 13-12-2020)

Heras, J. (2020, 10). *Error Cuadrático Medio para Regresión.* Descargado de https://www.iartificial.net/error-cuadratico-medio-para-regresion/

Hurtado de Barrera, J. (1998). Metodología de la investigación holística. *Fundacite–SYPAL. Caracas*, 887.

INTRODUCCIÓN A GOOGLE COLAB PARA DATA SCIENCE. (2020, 11). Descargado de https://www.datahack.es/blog/big-data/google-colab-para-data-science/

Jiménez, F. M. (2012). Redes neuronales y preprocesado de variables para modelos y sensores en bioingeniería. *Ingeniería En Electrónica, Universidad Politécnica de Valencia*.

Longoni, M. G., Porcel, E., López, M. V., y Dapozo, G. N. (2010). Modelos de redes neuronales perceptrón multicapa y de base radial para la predicción del rendimiento académico de alumnos universitarios. En *Xvi congreso argentino de ciencias de la computación.*

López, M. V., Ramírez Arballo, M., Porcel, E., Mata, L. E., y Barreto, S. E. (2012). Redes neuronales para predecir el rendimiento académico de los alumnos ingresantes a la carrera de bioquímica de la facena-unne en función de sus conocimientos matemáticos previos. En *Xviii congreso argentino de ciencias de la computación.*

Matich, D. J. (2001). Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*.

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). Determination press San Francisco, CA.

NVIDIA Corporation. (2010). *NVIDIA CUDA C programming guide.*

NVIDIA, Vingelmann, Péter, Fitzek, Frank H.P. (2020). *Cuda release: 10.2.89.*

Porcel, E. A., López, M. V., y Dapozo, G. N. (2011). Predicción del rendimiento académico de alumnos de primer año de universidad mediante redes neuronales. *Revista de la Escuela de Perfeccionamiento en Investigación Operativa*, *19*(32), 97–111.

Python Software Foundation. (2020, Marzo). *Introducción python.* `http://docs.python.org.ar`. (Acceso 11-08-2020)

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Serengil, S. I. (2017). *Softplus as a neural networks activation function.* `https://sefiks.com/2017/08/11/softplus-as-a-neural-networks-activation-function/`. (Acceso 21-04-2021)

Tutorials Point. (2017, Septiembre). *Tensorflow.* `https://www.tutorialspoint.com/tensorflow/`. (Acceso 10-04-2021)

Umar, M. A. (2019). Student academic performance prediction using artificial neural networks: A case study. *International Journal of Computer Applications*, *975*, 8887.

Unipython. (2019, Noviembre). *Introducción a theano.* Descargado de `https://unipython.com/introduccion-a-theano/`

Zacharis, N. Z. (2016). Predicting student academic performance in blended learning using artificial neural networks. *International Journal of Artificial Intelligence and Applications*, *7*(5), 17–29.

CAPÍTULO 6

ANEXOS

6.1. Instalación de Theano

Theano proporciona extensas instrucciones de instalación para los principales sistemas operativos: Windows, OS X y Linux. Lea la guía de instalación de Theano para su plataforma. Theano asume que usted trabaja en un entorno Python 2 o Python 3 con SciPy. Hay varias formas de realizar la instalación fácil, como usar Anaconda para configurar rápidamente Python y SciPy en su máquina. Con un entorno de trabajo Python y SciPy, es relativamente fácil de instalar Theano usando pip, si estas en window con anaconda ver al prompt de Anaconda. Vaya a "Windows" $-$ $>$ Busque "Anaconda Prompt" y ejecute como administrador, como se muestra a continuación:

Ahora escribe el comando para instalar Theano: `pip install Theano` Si estás en linux:`sudo pip install Theano` Es posible que se anuncien nuevas versiones de Theano y querrá actualizarlas para no obtener cualquier error y mejorar de la eficiencia. Puedes actualizar Theano usando pip de la siguiente manera:

```
sudo pip install --upgrade --no-deps theano
```

*Si estás en window elimina sudo del comando. Si lo desea, puede utilizar la versión más moderna de Theano chequeada directamente desde GitHub. Esto puede ser necesario en el caso de algunas bibliotecas de envolturas que utilizan cambios de API de última generación. Puede instalar Theano directamente desde un directorio de GitHub de la siguiente manera:

```
sudo pip install --upgrade --no-depsgit+git:github.com
/Theano/Theano.git
```

Ahora está listo para ejecutar Theano en su CPU, lo cual es muy importante para el desarrollo de modelos pequeños. Los modelos grandes pueden funcionar lentamente en la CPU. Si tienes una GPU Nvidia, puedes ver cómo congestionar Theano para usar su GPU(Unipython, 2019).

6.2. Instalación de Tensorflow

Si bien TensorFlow tiene fama de ser más fácil de instalar que otras librerías para redes neuronales, este proceso no está exento de problemas y puede resultar difícil para desarrolladores novatos.

Como cualquier programa, TensorFlow usará la CPU de nuestro equipo para funcionar, aunque, si contamos con una tarjeta gráfica (GPU) de la marca Nvidia, también existirá la posibilidad de usarla para acelerar la ejecución y los cálculos.

La guía completa de instalación puede encontrarse en las referencias (NVIDIA Corporation, 2010) y (Tutorials Point, 2017), incluyendo cualquier arquitectura. A continuación, se describirán los pasos que se siguieron para la instalación en nuestro equipo. Primero, debemos comprobar que nuestra GPU soporta este tipo de computación ya que, pese a que para la mayoría de las GPUs modernas es así, puede darse el caso contrario. Esto se hace comprobándolo en la lista de la referencia (NVIDIA, Vingelmann, Péter, Fitzek, Frank H.P., 2020). En esta lista también podremos ver la capacidad de computación de la GPU en cuestión, en nuestro caso 5.0.

Tras esto, comprobaremos que tenemos un sistema operativo que soporta CUDA. CUDA es un modelo de computación y programación inventado por Nvidia que nos permite usar la potencia computacional de la GPU para cualquier tarea que se quiera.

Comprobaremos el sistema operativo en la referencia (Tutorials Point, 2017), así como que tenemos una versión aceptada de Visual Studio instalada. Si alguno de los pasos anteriores fallara, tendremos o que solucionarlos (instalando versiones que soporten lo que queremos hacer, cambiando el hardware, etc.) o simplemente usar TensorFlow con la CPU.

Tras esto, descargaremos CUDA siguiendo el link que nos proporcionan en (NVIDIA Corporation, 2010) y lo instalaremos. Suele tardar bastante, pero no supone ningún problema. Un aspecto a tener en cuenta es que, aunque ponga que soporta Visual Studio 2017 y CUDA 9.0, esto no es así ya que ese soporte se encuentra en desarrollo o, al menos, así era la situación al instalarlo para este proyecto. Por lo tanto, deberemos instalar como máximo las versiones Visual Studio 2015 y CUDA 8.0.

Tras esta instalación verificaremos que los programas se encuentran instalados correctamente, por ejemplo, comprobando que existen los archivos especificados en el punto 2.5.1 de la referencia (NVIDIA Corporation, 2010) o ejecutando esos mismos archivos. Después, debemos instalar cuDNN, una librería para CUDA que nos permite implementar redes neuronales y que será usada por TensorFlow. Para ello, debemos registrarnos como desarrolladores en el Programa para Desarrolladores de Nvidia (Nvidia Developers Program), en la referencia (NVIDIA, Vingelmann, Péter, Fitzek, Frank H.P., 2020). Descargaremos e instalaremos cuDNN y comprobaremos que todo

está correctamente instalado. Finalmente, debemos instalar TensorFlow.

TensorFlow puede ser instalado en varios lenguajes: Python, Java, Go o C. Como se explicó anteriormente, usaremos Python, así que debemos instalarlo si no lo hemos hecho con anterioridad. Puede ser descargado desde la referencia (Python Software Foundation, 2020). Debemos instalar una de las versiones 3.X de Python, preferiblemente la más actual.

TensorFlow puede instalarse de 2 formas, mediante pip, herramienta que proporciona Python para instalar paquetes; o mediante Anaconda, una distribución abierta y libre de Python para ciencia de datos y Machine Learning (*Anaconda Software Distribution*, 2020).

Nosotros usaremos pip, e instalaremos TensorFlow mediante el siguiente comando:

```
pip3 install upgrade tensorflow gpu
```

Tras este paso ya se encuentra todo instalado y sólo nos faltaría realizar un pequeño test en Python para comprobar que, efectivamente, todo funciona correctamente. Para este test podemos utilizar cualquiera de los ejemplos o tutoriales de la referencia (Tutorials Point, 2017).

6.3. Google Colaboratory (Google Colab)

Dentro del mundo del Data Science, existen iniciativas muy interesantes, y una de las que más no puede interesar, además de todos las opciones formativas y herramientas disponibles, son los Google Colab.

Colab es un servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google, con librerías como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV. Todo ello con bajo Python 2.7 y 3.6, que aún no está disponible para R y Scala.

Aunque tiene algunas limitaciones, que pueden consultarse en su página de FAQ, es una herramienta ideal, no solo para practicar y mejorar nuestros conocimientos en técnicas y herramientas de Data Science, sino también para el desarrollo de aplicaciones (pilotos) de machine learning y deep learning, sin tener que invertir en recursos hardware o del Cloud.

Con Colab se pueden crear notebooks o importar los que ya tengamos creados, además de compartirlos y exportarlos cuando queramos. Esta fluidez a la hora de manejar la información también es aplicable a las fuentes de datos que usemos en nuestros proyectos (notebooks), de modo que podremos trabajar con información contenida

en nuestro propio Google Drive, unidad de almacenamiento local, github e incluso en otros sistemas de almacenamiento cloud, como el S3 de Amazon. (*INTRODUCCIÓN A GOOGLE COLAB PARA DATA SCIENCE*, 2020)