El Algoritmo De Retropropagación En Redes Neuronales Multicapa

Almudena Fraga Fatás

Trabajo de Fin de Grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: José Tomás Alcalá Nalvaiz
4 de Julio de 2023

Resumen

En este trabajo de fin de grado se explora el campo de las redes neuronales artificiales y su aplicación a través del algoritmo de retropropagación. Las redes neuronales artificiales permiten simular de forma aproximada el comportamiento de las redes neuronales biológicas, estudiando así la clasificación de datos y la toma de decisiones. Éstas han revolucionado el estudio en inteligencia artificial al crear sistemas capaces de aprender y mejorar simplemente mediante la experiencia. Suponen un gran salto en el mundo de la digitalización, siendo capaces de tomar decisiones de forma similar a los seres humanos.

Concretamente, este trabajo se centra en el algoritmo de retropropagación en redes neuronales multicapa; se puede observar a lo largo del mismo que es una técnica efectiva a la hora de entrenar redes neuronales. Mediante la retropropagación se ajustan los pesos y sesgos, lo que permite minimizar el error y así mejorar la precisión de la red neuronal. Este trabajo está estructurado en 3 capítulos:

En el primero, se hace una presentación general a las redes neuronales y las distintas dinámicas de aprendizaje que existen dependiendo de diversos factores. Haciendo especial hincapié en la estructura de las redes neuronales multicapa, explicando de forma breve varios tipos de funciones de activación utilizadas en este campo.

El segundo se centra en el algoritmo de retropropagación, mostrando detalladamente la importancia del gradiente descendente y cómo se aplica a la hora de tratar de minimizar la función del error. Se estudian los distintos casos que existen a la hora de obtener los resultados y las derivadas en cada capa. Todos estos casos son representados mediante los llamados *B-diagramas*, representaciones básicas en las cuales se divide la neurona en dos, representando en la parte derecha la función de activación y en la izquierda su derivada.

Finalmente, se encuentra el capítulo de aplicación del algoritmo. Mediante tres archivos de código en el lenguaje de programación *R*, se aplica el algoritmo de retropropagación a varios conjuntos de datos, visualizando así su eficacia y precisión mediante gráficas del error a lo largo de las iteraciones.

En conclusión, la intención principal de este proyecto es que con su lectura cualquier persona tenga una idea general de lo que son las redes neuronales artificiales, de su estructura y funcionamiento, así como de su aplicación a distintos conjuntos de datos.

Summary

In this final degree project, it is explored the field of artificial neural networks and their application through the backpropagation algorithm. Artificial neural networks allow simulating in an approximate way the behavior of biological neural networks, thus studying data classification and decision-making. They have revolutionized the study of artificial intelligence by creating systems capable of learning and improving simply through experience. They represent a great leap forward in the world of digitization, being able to make decisions in a similar way than humans.

Specifically, this work focuses on the backpropagation algorithm in multilayer neural networks, it can be seen throughout this work that this is an effective technique when training neural networks. Through backpropagation, weights and biases are adjusted, which allows minimizing the error and thus, improving the accuracy of the artificial neural network. This work is structured in 3 chapters.

In the first one, a general presentation is made on neural networks and the different adaptation dynamics that exist depending on various factors. It also places special emphasis on the structure of multilayer neural networks, explaining briefly several types of activation functions used in this field.

The second chapter focuses on the backpropagation algorithm, showing in detail the importance of the gradient descent and how it is applied when seeking to minimize the error function. The different cases that exist when obtaining the results and the derivatives in each layer are studied. All these cases are represented by means of the so-called *B-diagrams*, basic representations in which the neuron is divided in two, representing the activation function on the right side and its derivative on the left side.

Finally, it is presented the chapter of application of the algorithm. Using 3 code files in the *R* programming language, the backpropagation algorithm is applied to several data sets, thus visualizing its efficiency and accuracy by means of error plots along the iterations.

In conclusion, the main purpose of this project is that, by reading it, anyone will have a general idea of what artificial neural networks are, their structure and operation, as well as their application to different data sets.

Índice general

Capítulo 1

Introducción a las Redes Neuronales Artificiales

Para comenzar este estudio se van a posicionar en el tiempo las distintas etapas que destacan en el estudio de las redes neuronales artificiales. El inicio del estudio de las redes neuronales se remonta a 1936, cuando Alan Turing estudió el cerebro relacionándolo con el mundo de la computación. Sin embargo, no fue hasta 1943 cuando Warren McCulloch y Walter Pitts modelaron una red neuronal simple mediante circuitos eléctricos. En 1949, Donald Hebb fue el primero en explicar los procesos de aprendizaje, la idea era que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. Sus trabajos son considerados las bases de la teoría de redes neuronales. Algunos años más tarde, en 1959, Frank Rosenblatt comenzó el desarrollo del Perceptrón. Este modelo era capaz de identificar patrones y, tras haber aprendido una serie de éstos, reconocer otros similares. Sin embargo, este modelo era incapaz de trabajar con funciones no separables linealmente. En 1960, Bernard Widroff y Marcian Hoff desarrollaron el modelo Adaline ("ADAptative LINear Elements"). Fue la primera red neuronal que se aplicó a un problema real. No obstante, en 1969, Marvin Minsky y Seymour Papert demostraron que el Perceptrón no era capaz de resolver problemas con funciones no lineales. Esto desembocó en la idea de que el Perceptrón era muy débil computacionalmente, provocando la casi muerte de las redes neuronales. En 1974, Paul Werbos desarrolló la idea del algoritmo de aprendizaje de propagación hacia atrás o retropropagación (en inglés backpropagation) y en 1985 fue redescubierto por David Rumelhart y G. Hinton. A partir de 1986 resurgió el estudio de las redes neuronales y a día de hoy son numerosos los estudios relacionados con este campo.

En este capítulo se introduce el concepto de redes neuronales artificiales, en inglés "Artificial Neural Networks" (ANN), y se presenta una visión general de sus tipos y estructura. Se centra en las redes neuronales multicapa, introduciendo algunos conceptos como las redes neuronales feedforward, las dinámicas de aprendizaje y las funciones de activación. En general, nos da un conocimiento inicial para entender el resto del proyecto.

El contenido de este capítulo se ha inspirado en diversas fuentes. La introducción al contexto histórico ha sido obtenida de *Redes Neuronales: Conceptos Básicos y Aplicaciones* [8]. Algunas de las ideas y figuras aplicadas en la Sección 1.1 se basan en *Introduction to Soft Computing* [7] y *Neural Networks. A Systematic Introduction* [4], más concretamente en el Capítulo 7, mientras que la Sección 1.2 sigue la estructura de *Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification* [5] y también de *Introduction to Soft Computing* [7]. Por último, la Sección 1.3 incorpora ideas de *An Overview Of Artificial Neural Networks for Mathematicians* [2] pero con cambios en la notación y la presentación de los conceptos. Estas fuentes han desempeñado un papel crucial a la hora de proporcionar una base sólida y coherente para la redacción de este capítulo.

1.1. Dinámicas de Aprendizaje

Esta sección tiene como objetivo presentar las distintas dinámicas de aprendizaje que puede tener un modelo general de red neuronal. Más tarde se dará una explicación más concreta de las dinámicas de aprendizaje usadas en el caso del algoritmo de retropropagación.

El proceso de aprendizaje puede dividirse en dos aspectos: Incremento y presencia de Profesor.

1.1.1. Según el Incremento

Hay dos formas de proporcionar datos a una red neuronal:

- La red recibe los datos gradualmente y se ajusta de forma progresiva para aumentar la precisión de las respuestas.
- O, por el contrario, la red recibe todos los datos a la vez.

1.1.2. Según el Profesor

Antes de presentar las dos dinámicas de aprendizaje que existen dependiendo del Profesor, van a ser presentados una serie de conceptos de gran relevancia a lo largo del proyecto.

El principal objetivo de una red neuronal artificial es aproximar una función h. La red da como resultado una función F y el objetivo es que sea lo más próxima posible a la función h, de la cual se sabe algún resultado. Estos resultados, junto con sus entradas correspondientes, son conocidos como *Conjunto de Entrenamiento*. Para facilitar la notación se define como el conjunto de pares siguiente:

$$TR = \{(X_1^0, T_1), \cdots, (X_N^0, T_N)\}$$

donde X_i^0 representan los datos de entrada de cada elemento del conjunto de entrenamiento, T_i los de salida y N el número total de pares de datos. La relación entre ambos es la siguiente: $h(X_r^0) = T_r, r = 1, \ldots, N$. Tanto X_i^0 como T_i son vectores $\forall i = 1, \cdots, N$, ya que una red neuronal puede presentar más de una componente de entrada y dar como resultado más de una componente de salida. Estos datos hacen posible el aprendizaje de la red neuronal.

En algunos problemas de redes neuronales se emplea el *Error Cuadrático Medio*, también llamado *Función del Error*, representado por la siguiente ecuación:

$$E := \frac{1}{2N} \sum_{r=1}^{N} ||T_r - F(X_r^0)||^2 \tag{1.1}$$

En cuanto a la función de red F, esta puede tomar distintos valores dependiendo del objetivo del problema a resolver. Por un lado, puede tratarse de un problema de etiquetas, para el cual las salidas pueden ser un número finito de valores; Este tipo de funciones de red se usan para los problemas de clasificación de datos. Por otro lado, la función puede tomar un valor real entre 0 y 1, es decir, se centra en el cálculo de probabilidades. Esto se puede dar aunque las salidas reales sean binarias. Por último, puede tomar un valor real cualquiera.

Una vez conocidos estos términos, el aprendizaje se puede dividir en dos casos según el Profesor: el supervisado y el no supervisado.

Aprendizaje Supervisado

El modelo dispone de datos de entrada y salida, es decir, ya se conocen algunos resultados de la función que queremos aproximar. La función del error (1.1) toma un papel crucial en esta tarea, ya que calcula la distancia entre las salidas dadas por la red neuronal artificial y las salidas reales. Este tipo de aprendizaje es eficaz y puede encontrar soluciones a problemas lineales y no lineales.

Aprendizaje No Supervisado

En el aprendizaje no supervisado, sólo se le proporciona al modelo datos de entrada, por lo que se necesita la capacidad de aprender sin proporcionar señal de error. Al no tener información previa, se permite al algoritmo tomar patrones que no se habían considerado previamente.

Tras explicar los distintos tipos de modelos generales de redes neuronales existentes dependiendo de sus dinámicas de aprendizaje, se va a dar una breve descripción de un modelo básico de red neuronal artificial, haciendo especial hincapié en su estructura y componentes.

1.2. Breve descripción de las redes neuronales

Una Red Neuronal Artificial (también conocida por sus siglas en inglés como ANN) es un modelo matemático que se inspira en la estructura biológica de las neuronas y sus interconexiones. Se utiliza para resolver una amplia gama de problemas, como clasificar o agrupar datos. Para ello, cada ANN puede clasificarse según su modelo de aprendizaje. En esta sección es presentado el modelo más básico de red neuronal, cuya estructura se basa en una única neurona. Asumimos que la señal de entrada está formada por p características o variables, x_i, $i = 1, \ldots, p$.

1. Cada componente x_i de la señal de entrada tiene un peso relacionado w_i. Además, hay un parámetro adicional llamado sesgo (b), es un parámetro de ajuste que influye en la salida y capacidad de aprendizaje de la neurona. Para mantener la coherencia en las fórmulas presentadas más adelante, podemos considerarlo como otra señal de entrada x_0 con valor 1 cuyo peso es $w_0 = b$.

2. La señal de entrada y su respectivo vector de pesos entran en la sinapsis, donde se activa la operación sináptica. El resultado de esta operación es el producto interno $z = \sum_{i=0}^{p} w_i x_i$, el cual es aplicado a las entradas y los pesos de la neurona.

3. El producto interno es evaluado por la función de activación f (Subsección 1.2.1) y su resultado se convierte en la señal de salida de la neurona, denotada por o.

4. En los modelos generales de redes neuronales, esta señal de salida o se convierte en la señal de entrada para la siguiente neurona y el proceso continúa hasta que la última neurona calcula su función de activación, y se obtiene el resultado final.

Todos estos parámetros forman la estructura de una red neuronal artificial básica, se puede visualizar en la Figura 1.1. A lo largo del proyecto se explicarán casos más complejos, como por ejemplo los casos en los cuales las neuronas se encuentran separadas por capas y hay varias neuronas en cada capa (redes neuronales multicapa (MNN)).

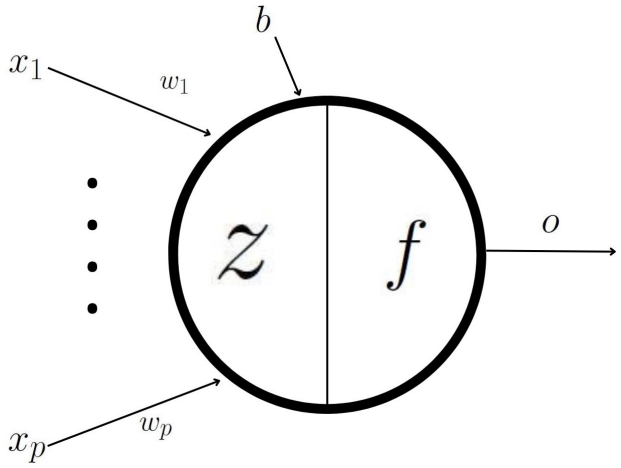
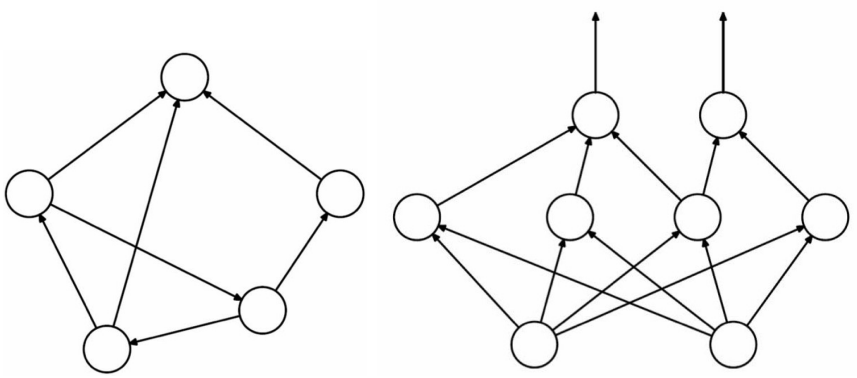


Figura 1.1: Representación Esquemática de una Neurona

A la hora de clasificar las redes neuronales, la primera diferencia que puede haber es en la organización de sus conexiones, nos encontramos ante dos tipos:

- Redes cíclicas (o recurrentes): La salida de una neurona se convierte posteriormente en la entrada de sí misma.
- Redes acíclicas (o feedforward): Por el contrario, este tipo de red no contiene ningún bucle.

Mediante la Figura 1.2 se pueden observar las diferencias entre ambos tipos.



(a) Red Neuronal Cíclica (b) Red Neuronal Acíclica

Figura 1.2: Representación de las redes neuronales dependiendo de su organización. Figura tomada de *Introduction to Soft Computing* [7].

Por otro lado, al igual que en la Subsección 1.1.2, las redes neuronales artificiales se pueden clasificar en función de si presentan aprendizaje supervisado o no supervisado.

Aprendizaje Supervisado

El aprendizaje supervisado en el caso de redes neuronales artificiales consta de dos fases:

- Feedforward: Calcula la salida generada por la red neuronal en cada iteración.
- Backpropagation: Propaga el error desde la capa de salida hasta la inicial, calculando el gradiente del error con respecto a los pesos en cada capa. Esto proporciona a la red una forma óptima de actualizar los pesos y mejorar la aproximación.

Este método va a ser explicado con mayor detalle en el Capítulo 2, el algoritmo de retropropagación requiere este tipo de aprendizaje ya que trabaja basándose en la disminución del valor de la función del error (1.1).

Aprendizaje No Supervisado

Hay varios tipos de dinámicas dentro del aprendizaje no supervisado en redes neuronales. Uno de los más conocidos son los mapas autoorganizados (también conocidos por sus siglas en inglés como SOM). Algunas de sus características son:

- El feedforward está representado por una sola capa computacional que consiste en neuronas dispuestas en filas y columnas.
- Las neuronas con información estrechamente relacionada se colocan cerca unas de otras y presentan conexiones sinápticas.

La capa computacional también se conoce como capa competitiva, ya que las neuronas compiten entre sí para activarse. SOM se utiliza en muchas aplicaciones del mundo real, como problemas de categorización y regresión, ya que representa de forma natural el comportamiento neurológico.

Para terminar con esta breve descripción de las componentes de una red neuronal, se presentan las funciones de activación.

1.2.1. Funciones de Activación

Las funciones de activación desempeñan un papel crucial a la hora de determinar si el impulso de la señal se propaga a la neurona siguiente o no. Además, introducen la no linealidad en la red neuronal artificial. Existen varios tipos de funciones de activación, las principales son las siguientes:

- Step-function:

$$f(z) = \begin{cases} 1, \text{ if } z \geq 0 \\ 0, \text{ if } z < 0 \end{cases} \tag{1.2}$$

  Si el producto interno supera 0, la neurona transmite la señal a la siguiente. En caso contrario, no se transmite ninguna señal. También se puede expresar de la siguiente forma:

$$f(z) = \frac{sgn(z)+1}{2} \tag{1.3}$$

  La step-function no tiene derivada en $x = 0$ y su derivada en cualquier otro lugar es 0. Esta característica no la hace ser adecuada en combinación con el algoritmo de Backpropagation, como veremos en el siguiente capítulo.

- Función Sigmoide:

$$f(z) = \frac{1}{1+e^{-z}} \tag{1.4}$$

  La Función Sigmoide es una generalización de la step-function y es una de las funciones de activación más populares para la retropropagación. Su dominio está formado por todos los números reales y devuelve un valor en el intervalo $(0,1)$. Presenta derivada en todo su dominio:

$$f'(z) = f(z)(1-f(z)) \tag{1.5}$$

- Tangente Hiperbólica:

$$f(z) = \frac{1-e^{-z}}{1+e^{-z}} \tag{1.6}$$

Es análoga a la función sigmoide pero, en este caso, el incremento es continuo de -1 a 1. Su salida se centra en torno a cero. Suele utilizarse en las capas ocultas porque la media de las salidas de la capa oculta es 0 o muy próxima a 0. Ayuda a centrar los datos y facilita el aprendizaje de la siguiente capa. Su derivada es:

$$f'(z) = 1 - (f(z))^2 \tag{1.7}$$

- Función ReLU (rectified linear unit):

$$f(z) = z^+ = max(0, z) \tag{1.8}$$

Su derivada es:

$$f'(z) = \begin{cases} 1, \text{ si } z > 0 \\ 0, \text{ si } z < 0 \end{cases} \tag{1.9}$$

Como sólo se activa un cierto número de neuronas, es computacionalmente más eficiente que otras funciones de activación. Un inconveniente de este método es que algunos pesos nunca cambian durante el proceso de retropropagación, ya que todos los valores de entrada negativos se convierten en 0. Una aproximación suave de esta función es la siguiente:

$$f(z) = ln(1 + e^z) \tag{1.10}$$

Su derivada es:

$$f'(z) = \frac{e^z}{1 + e^z} \tag{1.11}$$

Una vez presentadas las nociones básicas de las redes neuronales artificiales junto con sus divisiones según sus dinámicas de aprendizaje, organización de sus conexiones y funciones de activación, el capítulo concluye con un caso que produce un incremento en la dificultad de estudio de las ANN. El caso de las Redes Neuronales Multicapa (también conocido por sus siglas en inglés como MNN).

1.3. Redes Neuronales Multicapa

El propósito de esta sección es redefinir algunos conceptos presentados anteriormente con el objetivo de adaptarlos al caso de redes neuronales multicapa (MNN). En otras palabras, los casos en los cuales las neuronas se presentan ordenadas por capas. En este tipo de redes neuronales se da un gran aumento en el número de parámetros que forman parte de la misma, por lo tanto, algunos de ellos (como los pesos, vectores de entrada, ...) sufren algunos cambios para adaptar la notación a este incremento. El estudio está centrado en en este modelo neuronal a la hora de definir el Algoritmo de Retropropagación en el siguiente Capítulo 2.

1.3.1. Redes Neuronales Feedforward

Las redes neuronales con propagación hacia delante (también conocidas como Feedforward) son un tipo de ANN en el cual los valores de la capa n se usan para obtener los valores de la capa $n+1$. Para empezar, se presentan dos definiciones obtenidas de *An Overview Of Artificial Neural Networks for Mathematicians* [6].

Definición 1. La **profundidad** es el número de capas ocultas que hay en una red neuronal. Está denotado mediante la letra l. Cuando consideramos también la capa de entrada (capa 0) y la de salida (capa $l+1$), la denotamos por L y entonces $L = l + 2$.

Definición 2. Se denomina **anchura** al vector que contiene el número total de nodos que hay en cada capa. Se denota de la siguiente forma:

$$P = (k_0, \cdots, k_{l+1})$$

En el caso de la capa de entrada k_0, ésta coincide con p. En el caso de la capa de salida k_{l+1}, ésta puede ir desde 1, en el caso de modelos predictivos de una variable numérica, hasta los casos de clasificación múltiple, con g categorías posibles. En este último caso $k_{l+1} = g$.

A continuación, los elementos presentados a lo largo de la Sección 1.2 van a ser generalizados para el caso de redes neuronales multicapa.

Vector de Entrada

Al ser MNN se precisa de un vector de entrada para cada capa, se diferencian mediante el superíndice. Es decir, se denota x_i^n a la componente i del vector de entrada de la capa n.

$$X^n = \begin{pmatrix} x_0^n \\ x_1^n \\ \vdots \\ x_{k_{n-1}}^n \end{pmatrix}, \quad n = 1, \ldots, l+1 \;\; ; \quad X^0 = \begin{pmatrix} x_1^0 \\ \vdots \\ x_{k_0}^0 \end{pmatrix}$$

En cuanto al término x_0^n, este va a ser 1 en todas las capas (salvo la capa de entrada (capa 0)), ya que es el que está relacionado con el sesgo (al igual que en la Sección 1.2).

Pesos y Sesgos

Cuando se habla de pesos, se hace referencia a una representación de la "fuerza de conexión" que existe entre dos neuronas. Para empezar, es introducida la siguiente notación: $w_{a,b}^n$ es el peso relacionado con la conexión del nodo a en la capa $n-1$ y el nodo b en la capa n. Al igual que en la Sección 1.2, el sesgo de cada capa puede ser interpretado como una señal de entrada x_0^n con valor 1 y cuyo peso es $w_{0,i}^n = b_i^n$.

A continuación, pesos y sesgos van a ser presentados juntos en forma matricial:

Definición 3. Pesos en forma Matricial: Sea L el número total de capas de una red neuronal. Los pesos pueden ser representados como una matriz que contiene por columnas todas las conexiones entre un nodo en la capa $n-1$ y cada nodo de la capa n. Se exige lo siguiente: $w_{i,j}^n = 0$ para los casos en los que las dos neuronas correspondientes no presenten ninguna conexión.

$$W_n = \begin{pmatrix} w_{0,1}^n & w_{1,1}^n & \cdots & w_{k_{n-1},1}^n \\ \vdots & \vdots & \ddots & \vdots \\ w_{0,k_n}^n & w_{1,k_n}^n & \cdots & w_{k_{n-1},k_n}^n \end{pmatrix}, \quad n = 1, \ldots, l+1. \tag{1.12}$$

Se puede observar que la primera columna de la matriz hace referencia a los sesgos presentes en la capa n.

Vector de Salida

A modo de introducción, se va a dar una definición para el vector que contiene los productos internos de cada neurona en una capa n. Para facilitar la comprensión de los distintos términos se va a separar la definición de las operaciones correspondientes a la capa de entrada (capa 0) y a la capa de salida (capa $l+1$).

Definición 4. Producto Interno en forma matricial:

$$Z^n = \begin{pmatrix} z_1^n \\ \vdots \\ z_{k_n}^n \end{pmatrix} = W_n X^n = \begin{pmatrix} w_{0,1}^n & w_{1,1}^n & \cdots & w_{k_{n-1},1}^n \\ \vdots & \vdots & \ddots & \vdots \\ w_{0,k_n}^n & w_{1,k_n}^n & \cdots & w_{k_{n-1},k_n}^n \end{pmatrix} \begin{pmatrix} x_0^n \\ \vdots \\ x_{k_{n-1}}^n \end{pmatrix} \quad n = 1, \ldots, l+1. \tag{1.13}$$

Con el objetivo de obtener el vector de salida, se evalúa cada componente de este vector en la función de activación y se obtiene:

$$O^n = f(Z^n) = \begin{pmatrix} f(w^n_{0,1}x^n_0 + \cdots + w^n_{k_{n-1},1}x^n_{k_{n-1}}) \\ \vdots \\ f(w^n_{0,k_n}x^n_0 + \cdots + w^n_{k_{n-1},k_n}x^n_{k_{n-1}}) \end{pmatrix} = f(W_nX^n), \;\; 1 \leq n \leq l+1 \tag{1.14}$$

Como ya ha sido mencionado en anteriores secciones, este vector se convierte en el vector de entrada de la capa siguiente. Sin embargo, se puede observar que al evaluar cada componente de Z^n en la función de activación este vector está compuesto por k_n componentes, mientras que el vector de entrada de la siguiente capa está compuesto por estas k_n componentes más el valor 1 asociado a los diversos términos de sesgo. Por lo tanto, es necesario realizar un reajuste a la hora de obtener el vector de salida en las capas ocultas. Este elemento extra $o^n_0 = 1, \;\; 1 \leq n \leq l$ corresponde a la entrada relacionada con el sesgo. Así pues, se puede presentar la siguiente relación:

$$X^{n+1} = \begin{pmatrix} 1 \\ O^n \end{pmatrix} \;\; n = 1, \ldots, l \tag{1.15}$$

Para el caso $n = l+1$ se elimina este elemento extra, ya que esta capa da el resultado final de la red neuronal. Es decir, su resultado no se transforma en el vector de entrada de la siguiente capa.

$$O^{l+1} = f(Z^{l+1}) = \begin{pmatrix} f(w^{l+1}_{0,1}x^{l+1}_0 + \cdots + w^{l+1}_{k_{n-1},1}x^{l+1}_{k_{n-1}}) \\ \vdots \\ f(w^{l+1}_{0,k_n}x^{l+1}_0 + \cdots + w^{l+1}_{k_{n-1},k_n}x^{l+1}_{k_{n-1}}) \end{pmatrix} \tag{1.16}$$

Tras haber definido todas estas componentes, es fácil observar que una red neuronal artificial puede verse como una especie de composición de funciones de activación aplicadas de forma iterada, es decir:

$$X^{n+1} = \begin{pmatrix} 1 \\ O^n \end{pmatrix} = \begin{pmatrix} 1 \\ f(Z^n) \end{pmatrix} = \mathbf{f}(W_nX^n), \;\; n = 1, \cdots, l \tag{1.17}$$

donde se asume que la primera componente de la imagen de $\mathbf{f}$ es siempre 1. Y por lo tanto:

$$F(X^0) = O^{l+1} = f(W_{l+1}\mathbf{f}(W_l\mathbf{f}(\cdots W_2\mathbf{f}(W_1X^1)))) \tag{1.18}$$

Es necesaria una aclaración referente a la capa de entrada. Esta capa no presenta producto interno ni funciones de activación ya que simplemente presenta el vector de entrada a la red neuronal. Como consecuencia de esto, se obtienen las siguientes igualdades: Para empezar, $O^0 = X^0$, y al definir la componente de salida de la capa de entrada (capa 0), se puede dar una fórmula análoga a 1.15.

$$X^1 = \begin{pmatrix} 1 \\ O^0 \end{pmatrix} \tag{1.19}$$

Tras esta introducción a los conceptos fundamentales de las redes neuronales artificiales (ANN) y su estructura, en el Capítulo 2 se va presentar el Algoritmo de Retropropagación. Con la comprensión de este algoritmo los lectores van a ser capaces de ajustar los pesos y sesgos de una red neuronal, minimizar sus errores y, en consecuencia, mejorar la precisión de sus resultados. En este nuevo capítulo se va a continuar aplicando la misma notación que en el Capítulo 1.

Capítulo 2

Algoritmo de Retropropagación

Este capítulo tiene como objetivo principal la explicación del Algoritmo de Retropropagación (también conocido como Algoritmo de Backpropagation) mediante la indagación en conceptos clave y en los pasos que constituyen el mismo.

A modo de introducción, se da una explicación del objetivo del algoritmo y una breve descripción del mismo. A continuación, son explicados el Gradiente Descendente y la Tasa de Aprendizaje. Estos dos factores son los que permiten al algoritmo actualizar los pesos y disminuir el valor de la función del error, mejorando así la precisión de la red. Posteriormente, es presentado el algoritmo. Se han adjuntado algunas figuras con el propósito de facilitar el entendimiento y la visualización del mismo. Para finalizar, se expone una lista de ventajas y desventajas a tener en cuenta a la hora de usar este método.

La mayoría de este capítulo sigue *Neural Networks. A Systematic Introduction* [2] y *An Overview Of Artificial Neural Networks for Mathematicians* [6]. Por otro lado, la Sección 2.4 ha sido tomada de *What is Backpropagation Neural Network And Its Working* [9]. Además, aunque la Figura 2.1 fue creada usando código LaTeX, resultó ser una forma poco efectiva para fórmulas más largas. Como consecuencia, el resto de figuras fueron creadas a partir de la aplicación *Canva* [10]. Estas fuentes han proporcionado una base sólida para el desarrollo de este capítulo.

2.1. Objetivo y Breve Descrpción

A pesar de definir el conjunto TR en el capítulo anterior (Subsección 1.1.2), para presentar el algoritmo se va a suponer que el conjunto de entrenamiento está compuesto únicamente por una pareja de datos (X^0, T). El motivo por el cual podemos hacer esta simplificación es que en el caso general el gradiente del error será la suma de los gradientes obtenidos mediante cada elemento del conjunto de entrenamiento. Esto va a simplificar notablemente la notación, haciendo más fácil la comprensión del algoritmo. Además, se añade la siguiente notación relacionada con la salida del conjunto de entrenamiento: $T = (t_1, \cdots, t_{k_{l+1}})$ donde t_i es la componente i del vector salida correspondiente al conjunto de entrenamiento.

El problema del aprendizaje consiste en determinar la combinación óptima de pesos que da la mejor aproximación a la función h. Más concretamente, el algoritmo de retropropagación es usado para encontrar un mínimo local de la función del error (1.1). Para conseguir esto, se usa el método del Gradiente Descendente, que será explicado con mayor detenimiento en la Sección 2.2. Es necesario calcular las derivadas parciales de la función del error respecto a cada peso. Por esta razón, se precisa que la función de activación sea derivable. En consecuencia, funciones como la Step-Function (1.2) no son útiles para este método. El algoritmo de retropropagación calcula estos valores empezando por la última capa y propagándolos hacia atrás hasta la capa de entrada.

Una vez introducido este método, la siguiente sección se centra en el problema del cálculo del gradiente.

2.2. Gradiente Descendente

Para empezar el capítulo, se van a definir dos de los conceptos más cruciales: *Gradiente Descendente* y *Tasa de Aprendizaje*. Ambos muy importantes a la hora de determinar el método y la velocidad del aprendizaje.

Como se ha comentado previamente, el algoritmo va a ser descrito para el caso de un único par de datos en el conjunto de entrenamiento, por lo tanto el error con el que se va a trabajar es el siguiente:

$$E := \frac{1}{2}||T - F(X^0)||^2 \tag{2.1}$$

Para lograr disminuir el resultado de esta función a lo largo de las iteraciones, se usa el gradiente con respecto a los pesos, ya que son las variables de las que depende el error. Se usa el gradiente debido a que éste indica la dirección del ascenso más pronunciado, por lo tanto, $-\nabla E$ indica la dirección del descenso más pronunciado. La definición formal del gradiente es la siguiente:

Definición 5. El **gradiente** de una función diferenciable $g : \mathbb{R}^n \to \mathbb{R}$ en el punto $P = (p_1, \cdots, p_n) \in \mathbb{R}^n$ es un vector en $\mathbb{R}^n$ de la siguiente forma:

$$\nabla g(x) := \left(\frac{\partial g}{\partial p_1}(x), \cdots, \frac{\partial g}{\partial p_n}(x) \right). \tag{2.2}$$

El Gradiente Descendente es un algoritmo de optimización iterativo de primer orden que permite encontrar mínimos locales en una función diferenciable. Para ello se usa la siguiente expresión:

$$P(s+1) := P(s) - \lambda \nabla g(P(s)) \tag{2.3}$$

donde λ representa la Tasa de Aprendizaje explicada con más detalle a continuación.

Ahora, trasladando esto al objetivo de actualizar pesos de una red neuronal artificial, los nuevos pesos en cada iteración se obtienen mediante el siguiente esquema iterativo elemento a elemento:

$$w^n_{i,j} := w^n_{i,j} - \lambda \nabla E(w^n_{i,j}) \tag{2.4}$$

donde $\lambda > 0$ representa la Tasa de Aprendizaje.

Tasa de Aprendizaje

La *Tasa de Aprendizaje* es un parámetro que determina la cantidad de cambio que sufren los pesos durante el proceso de aprendizaje. En otras palabras, la velocidad que lleva el mismo. Su rango normalmente se encuentra entre 0 y 1. Una constante de aprendizaje elevada permite un aprendizaje más rápido, pero también puede resultar en oscilaciones en los pesos. Por el contrario, una constante de aprendizaje pequeña ofrece resultados más eficientes, pero conlleva tiempos de aprendizaje elevados. Para obtener más información sobre formas precisas de estimar una buena tasa de aprendizaje consultar *Estimating an Optimal Learning Rate For a Deep Neural Network* [8].

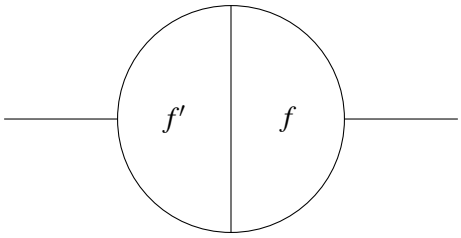


Figura 2.1: B-diagrama

2.2.1. Cómo Calcular el Gradiente Descendente

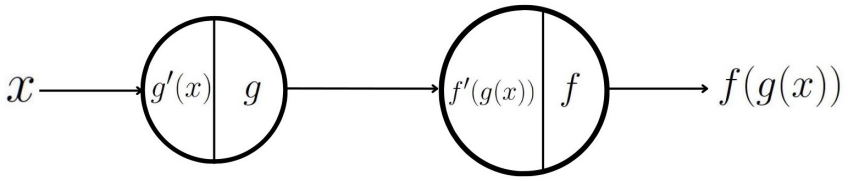
El objetivo principal de esta sección es explicar una forma de calcular las derivadas de una función asociada a una red neuronal artificial con respecto a su valor de entrada de forma eficiente. Para facilitar la comprensión, cada nodo es representado en dos partes: La parte de la derecha contiene el resultado de la función de activación, mientras que la izquierda contiene su derivada. Esta representación se llama *B-diagrama* y ha sido tomada de *Neural Networks. A Systematic Introduction* [2]. En la Figura 2.1 se puede ver un ejemplo de la misma.

A la hora de calcular el gradiente se pueden encontrar con distintos casos dependiendo de las conexiones existentes con las neuronas de cada capa. Estos casos van a ser presentados uno por uno junto con imágenes para lograr una mayor comprensión de los mismos. Su prueba resulta trivial ya que simplemente aplicando la regla de la cadena al resultado del Feedforward se obtiene el mismo resultado que en el paso de Backpropagation.

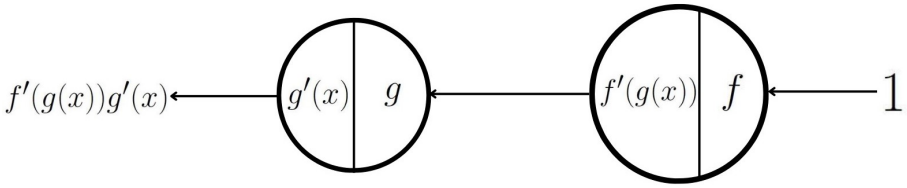
1. El primero es el caso en el que un nodo de la capa $(n+1)$ recibe señal únicamente de un nodo de la capa n y este a su vez recibe señal de un único nodo en la capa $n-1$. Durante el proceso de Feedforward, el resultado es la composición de ambas funciones de activación. Durante la retropropagación, la entrada por el final de la red es 1, y las entradas de cada nodo son multipicadas por el valor de la parte izquierda de cada neurona. Consultar la Figura 2.2 para mejor comprensión.

2. En el segundo caso una neurona de la capa $n+1$ recibe señal de 2 o más neuronas de la capa n. En el paso de Feedforward, el resultado es la suma de los resultados de ambas funciones de activación. Durante la retropropagación, al igual que en el caso anterior, la entrada por el final de la red es 1, como resultado se obtiene la suma de la parte izquierda de ambas neuronas. Consultar la Figura 2.3 para mejor comprensión.

3. El último caso presenta conexiones con pesos. En el paso del Feedforward, los datos de entrada se multiplican por el peso de la conexión, dando como resultado wx. En el caso de la retropropagación, al igual que en los casos anteriores, la entrada por el final de la neurona es 1, se multiplica 1 por el peso de la conexión, dando como resultado w. Este valor representa también la derivada de wx con respecto a x. En consecuencia, las conexiones con pesos se usan de la misma forma en ambos pasos. Consultar la Figura 2.4.

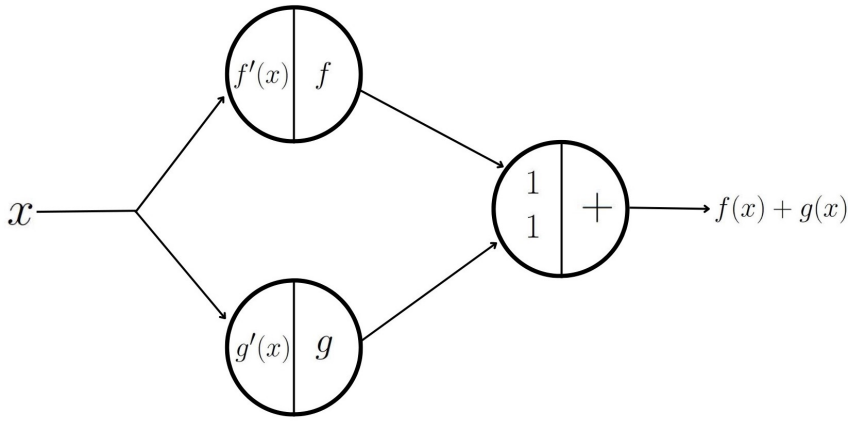


(a) Feed-forward

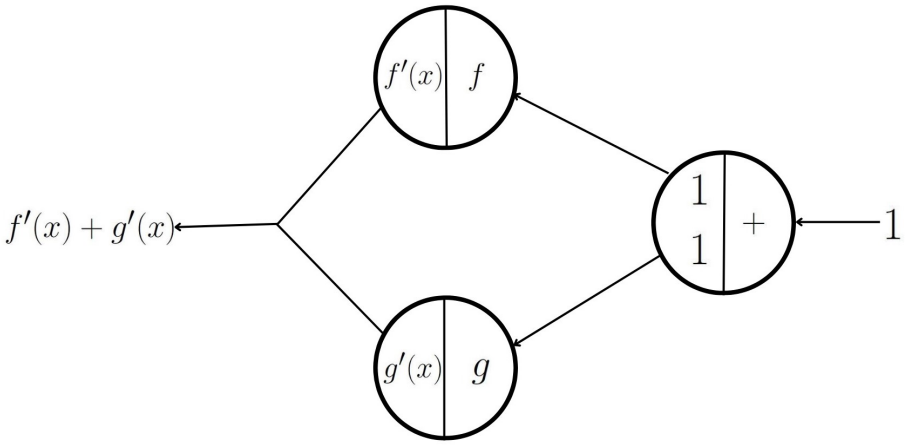


(b) Retropropagación

Figura 2.2: Composición de Funciones

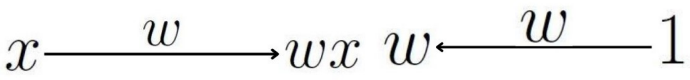


(a) Feed-forward



(b) Retropropagación

Figura 2.3: Suma de Funciones



(a) Feed-forward (b) Retropropagación

Figura 2.4: Enlaces con Pesos

En la siguiente sección, tras obtener los conocimientos necesarios a la hora de calcular el gradiente del error de una forma eficiente, es presentado el Algoritmo de Retropropagación.

2.3. Algoritmo

Tras esta breve explicación de cómo se calcula el gradiente dependiendo de la forma en la que estén conectadas las neuronas, en esta sección se va a presentar finalmente el Algoritmo de Retropropagación. Primero se presenta un algoritmo para los casos de una entrada con una componente, luego se ve el caso en el cual la red presenta 3 capas con varias neuronas en cada una y para finalizar una generalización para el caso de varias capas ocultas. El algoritmo presentado a continuación junto con la proposición asociada a este han sido obtenidos de *Neural Networks. A Systematic Introduction* [2].

Algoritmo 1. Se considera una red con una única entrada real x y una función F asociada a la red. Su derivada $F'(x)$ se calcula en dos fases:

- Propagación hacia delante: x se introduce en la red. Las funciones de activación y sus derivadas se evalúan en cada nodo. Las derivadas se almacenan.
- Retropropagación: La constante 1 se introduce a través de la capa de salida y la red opera en sentido inverso. La información que llega a un nodo se añade y el resultado se multiplica por el valor almacenado en la parte izquierda de la neurona. El resultado recogido en la neurona de entrada es la derivada de la función de la red con respecto a su entrada.

Proposición 1. El Algoritmo 1 obtiene la derivada de la función de red F con respecto a la entrada x correctamente.

(La demostración puede encontrarse en el Anexo A).

Para relacionar esta Proposición con el objetivo del algoritmo, se presenta la Figura 2.5, en la que se observa la red neuronal junto con una última capa la cual calcula el error de cada componente del vector salida. De esta forma, se obtiene el resultado del gradiente del error respecto a la entrada mediante esta Proposición. Como la entrada de cada neurona tiene la forma $o_i^{n-1}w_{i,j}^n \quad \forall i = 0,\cdots,k_{n-1}, \quad \forall j = 1,\cdots,k_n \;\; y \;\; \forall n = 1,\cdots,l+1$, en la próxima Subsección 2.3.1 se explica la forma de obtener el gradiente de la función del error con respecto a los pesos a partir del gradiente con respecto a la entrada de cada neurona.

2.3.1. Caso de Redes Neuronales Multicapa (MNN)

En esta sección, empieza la generalización de este algoritmo para el caso de una capa oculta. Más adelante se presentará el algoritmo para el caso de varias capas ocultas. Es considerada una red neuronal artificial con k_0 elementos de entrada, k_1 neuronas en la capa oculta y k_2 elementos de salida. Se mantiene la notación del Capítulo 1 Para refrescar esta notación, recordamos que $w_{i,j}^n$ hace referencia al peso asociado a la conexión entre la neurona i de la capa $n-1$ y la neurona j de la capa n. Por lo tanto, en este caso los pesos serán de la forma $w_{i,j}^1$ y $w_{i,j}^2$. La estructura general de la red está compuesta por $(k_0+1)\times k_1$ pesos que relacionan la capa de entrada y la oculta, y $(k_1+1)\times k_2$ pesos que relacionan la capa oculta y la de salida.

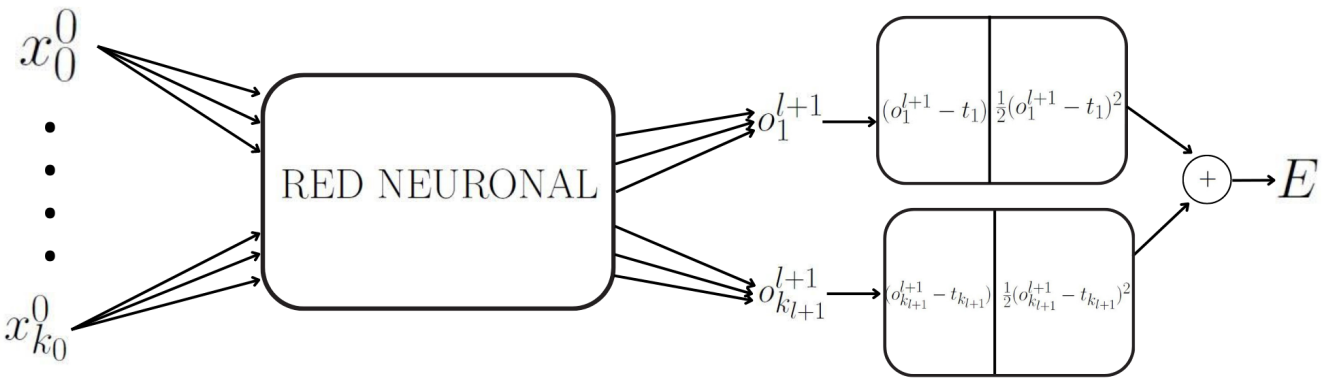


Figura 2.5: Capa del Error

Tras inicializar los pesos aleatoriamente, el algoritmo de retropropagación calcula las correcciones necesarias. Consta de cuatro pasos diferentes que se explican a continuación.

Propagación hacia delante

En primer lugar, como en la Subsección 1.3.1, se calculan las salidas de cada capa, que se convierten en las entradas de la capa siguiente. Este proceso continúa hasta llegar a la capa de salida, en la cual se obtiene el resultado de la función de la red.

Retropropagación desde la capa de salida

En este paso se obtienen los valores $\frac{\partial E}{\partial w_{i,j}^2}\ \forall i=0,\cdots,k_1\ y\ \forall j=1,\cdots,k_2$. Estos valores son necesarios para obtener la mejor aproximación de los pesos de la última capa. Como los valores de entrada a esta capa son de la forma $o_i^1 w_{i,j}^2$, con el Algoritmo 1 se obtiene el valor del gradiente de E con respecto a su valor de entrada, es decir, $\frac{\partial E}{\partial o_i^1 w_{i,j}^2}$. Al considerarse en este paso o_i^1 constante, se puede obtener lo siguiente:

$$\frac{\partial E}{\partial w_{i,j}^2} = o_i^1 \frac{\partial E}{\partial o_i^1 w_{i,j}^2} \tag{2.5}$$

Ahora, usando la siguiente notación: $\delta_j^2 = \frac{\partial E}{\partial o_i^1 w_{i,j}^2}$, se obtiene la variación que sufren los pesos de la segunda capa.

$$\Delta w_{i,j}^2 = -\lambda o_i^1 \delta_j^2 \tag{2.6}$$

Seguidamente, tras obtener una fórmula para esta variación, es necesario saber el valor de δ_j^2. Aplicando el Algoritmo 1, se multiplica la parte izquierda de las neuronas involucradas en la conexión requerida y se obtiene lo siguiente:

$$\delta_j^2 = o'(z_j^2)(o_j^2 - t_j) \tag{2.7}$$

donde $o'(z_j^2)$ es la derivada de la función de activación evaluada en el producto interno de la neurona j de la capa de salida. Finalmente se obtiene el valor deseado:

$$\frac{\partial E}{\partial w_{i,j}^2} = \delta_j^2 o_i^1 = [o'(z_j^2)(o_j^2 - t_j)]o_i^1 \tag{2.8}$$

Retropropagación desde la capa oculta

En este paso se trata de obtener $\frac{\partial E}{\partial w_{i,j}^1}\ \forall i=0,\cdots,k_0\ y\ \forall j=1,\cdots,k_1$. Esto se lograría con un proceso muy parecido al anterior salvo leves cambios en alguna fórmula. Al tratarse de una capa anterior, el

resultado se obtiene mediante la siguiente formula:

$$\delta_j^1 = o'(z_j^1) \sum_{q=1}^{k_2} w_{j,q}^2 \delta_q^2 \tag{2.9}$$

Y finalmente, de forma análoga a la capa anterior, se obtiene:

$$\frac{\partial E}{\partial w_{i,j}^1} = \delta_j^1 o_i^0 \tag{2.10}$$

Se puede concluir que mediante este procedimiento se puede generalizar este método para cualquier número de capas.

Actualización de Pesos

Para finalizar, hay que tener presente que el objetivo de todo este algoritmo es la actualización de sus pesos y sesgos. Éstos, como se expuso previamente, se actualizan en la dirección del descenso más pronunciado obteniendo así las siguientes variaciones:

$$\Delta w_{i,j}^2 = -\lambda o_i^1 \delta_j^2 \quad \forall i = 0, \cdots k_1 \quad y \quad \forall j = 1, \cdots k_2 \tag{2.11}$$

$$\Delta w_{i,j}^1 = -\lambda o_i^0 \delta_j^1 \quad \forall i = 0, \cdots k_0 \quad y \quad \forall j = 1, \cdots k_1 \tag{2.12}$$

Una vez vistos los diferentes casos para el cálculo de las derivadas parciales, se generaliza este algoritmo para el caso de L capas (donde L es la misma que en la Definición 1).

1. Feedforward: Se procede igual que en la Sección 1.3 y se obtiene lo siguiente:

   $$Z^n = W_n X_n \ \ \forall 1 \leq n \leq l$$

   $$O^n = f(Z^n) \ \ \forall 1 \leq n \leq l$$

   donde se reserva la primera componente del vector salida y se le asigna como valor 1 ($o_0^n = 1 \quad \forall n = 1, \cdots, l$), salvo en la capa de salida (capa $l+1$). En cuanto a la capa de entrada (capa 0), funciona de la misma forma que en la Sección 1.3, $X^0 = O^0$.

2. Retropropagación: En este paso se usan las mismas fórmulas que en el apartado anterior pero para el caso de L capas.

   $$\frac{\partial E}{\partial w_{i,j}^n} = o_i^{n-1} \frac{\partial E}{\partial o_i^{n-1} w_{i,j}^n} \ \ \forall 1 \leq n \leq l+1$$

   $$\delta_j^n = \frac{\partial E}{\partial o_i^{n-1} w_{i,j}^n} \ \ \forall 1 \leq n \leq l+1$$

   $$\Delta w_{i,j}^n = -\lambda o_i^{n-1} \delta_j^n \ \ \forall 1 \leq n \leq l+1$$

   Para obtener δ_j^n, al igual que en el apartado anterior, hay que distinguir entre dos casos. El primero sería la derivada de los pesos que conectan la última capa oculta con la capa de salida, y el segundo el resto de derivadas.

   $$\delta_j^n = \begin{cases} o'(z_j^n)(o_j^n - t_j) & si \quad n = l+1 \\ o'(z_j^n) \sum_{q=1}^{k_{n+1}} w_{j,q}^{n+1} \delta_q^{n+1} & si \quad 1 \leq n < l+1 \end{cases}$$

3. Actualización de Pesos: Se generaliza la fórmula de la sección anterior.

   $$\Delta w_{i,j}^n = -\lambda o_i^{n-1} \delta_j^n \ \ \forall 1 \leq n \leq l+1$$

Estos pasos se llevarían a cabo con cada par del conjunto de entrenamiento.

Con este capítulo se obtienen los conocimientos necesarios para comprender cómo actúa el algoritmo de retropropagación en una red neuronal artificial compuesta por un número cualquiera de capas y neuronas. Para finalizar, se exponen una serie de ventajas y desventajas relacionadas con el uso de este algoritmo.

2.4. Ventajas y Desventajas del Algoritmo

Como cualquier algoritmo, el Algoritmo de Retropropagación tiene sus ventajas y sus limitaciones. Es importante ser conocedor de éstas antes de aplicarlo en un conjunto de datos. La lista de ventajas y desventajas ha sido inspirada por *What is Backpropagation Neural Network And Its Working* [9] aplicando alguna modificación.

Ventajas

- Es rápido y fácil tanto de entender como de aplicar.
- Es adaptable a distintas estructuras y eficaz.
- Es un método estandarizado y ofrece resultados eficientes.

Desventajas

- Su rendimiento depende de los datos de entrada.
- Es sensible a los datos ruidosos o imprecisos.
- Requiere el uso de una función de activación continua y derivable.

Durante este capítulo se han expuesto los fundamentos del Algoritmo de Backpropagation, incluyendo una explicación del cálculo del Gradiente Descendente para obtener la disminución del valor de la función del error.

A partir de estos conocimientos, el Capítulo 3 se centra en la aplicación de este algoritmo. Presenta la implementación del código en el lenguaje de programación *R*, explicando sus componentes y funciones. A lo largo de este nuevo capítulo, los lectores adquieren una comprensión global del funcionamiento de una red neuronal artificial, mas concretamente de una red neuronal multicapa (MNN).

Capítulo 3

Aplicación del Algoritmo

Para finalizar este proyecto se expone el Capítulo 3, en el cual se aplica el Algoritmo de Retropropagación a tres conjuntos de datos.

En el primer caso, se prueba la efectividad del algoritmo en un conjunto de datos basado en dos círculos concéntricos. Durante el segundo caso, se aplica el algoritmo a datos circulares no concéntricos y se realiza el mismo método añadiendo una neurona más a cada capa oculta. Por último, el conjunto de datos se genera aleatoriamente de una región y su salida correspondiente se obtiene aplicando a estos datos una red neuronal con unos pesos y sesgos dados. El objetivo de éste último estudio difiere ligeramente de los dos anteriores, ya que surgió una pregunta durante el proyecto: Si aplicamos el algoritmo de retropropagación a un conjunto de datos generado por una red neuronal, utilizando una red neuronal con la misma estructura pero compuesta por pesos y sesgos generados aleatoriamente, ¿será capaz ésta de aproximar los pesos y sesgos con la misma precisión que los resultados obtenidos?

La implementación del código ha sido dividida en tres ficheros los cuales van a ser explicados a continuación. Aplicando este código a distintos conjuntos de datos queda demostrada la habilidad del algoritmo de predecir respuestas y clasificar correctamente la información. Se adjuntan durante el capítulo las gráficas del error a lo largo de las iteraciones y, en los casos de los círculos, las tablas de contingencia que representan el número de predicciones incorrectas y correctas.

La primera implementación del código fue tomada inicialmente de *How to code a Neural Network from scratch in R* [1]. Sin embargo, después de encontrar algunos resultados desconcertantes, todo el código fue revisado y se realizaron algunos cambios en el mismo para un correcto uso de los sesgos.

3.1. Explicación del Código

El código ha sido dividido en tres ficheros: ANNBPv2.R, ANNsimulator.R y entrenarANN.R. Cada fichero realiza una función específica a la hora de obtener resultados. A continuación se da una breve descripción de cada uno, quedando igualmente anexado el código completo (Anexo B).

ANNBPv2.R

En este fichero se incluye la definición de la clase que implementa los elementos pertenecientes a la red neuronal artificial. En él se encuentran las funciones de Feedforward y Retropropagación, las cuales son cruciales para el entrenamiento de la red. También incluye la función de activación y su derivada. En este estudio únicamente se aplica la función sigmoide (1.4).

ANNsimulator.R

Este fichero contiene la simulación de dos conjuntos de datos. Uno de ellos genera valores de entrada aleatorios dentro de una región y su valor de salida se obtiene aplicando el feedforward de una red neuronal artificial al mismo. Puede generar salidas numéricas o binarias. Por otro lado, el segundo modelo obtiene los elementos de entrada de dos círculos, la salida es binaria basada en el círculo al que pertenece la entrada.

entrenarANN.R

Este fichero define la topología de la red neuronal artificial. Aporta la información necesaria respectiva al número de capas, el número de neuronas en cada capa, ... También inicializa la red y lleva a cabo el proceso de entrenamiento, evaluando el decrecimiento de la función del error a lo largo de las iteraciones.

Tras esta breve explicación de cada fichero de código, éste va a ser aplicado a varios conjuntos de datos, probando así la eficacia del algoritmo en distintas situaciones.

3.2. Aplicación del Código

A lo largo de esta sección se va a estudiar el comportamiento y rendimiento del algoritmo en tres conjuntos de datos distintos. Dos de ellos son generados por círculos y dan resultados binarios dependiendo del círculo al que pertenecen mientras que el tercero es generado por una red neuronal artificial. Durante el estudio se van a visualizar los 3 conjuntos, el primer y el tercer conjunto de datos se aplican a redes neuronales con la misma estructura, mientras que para el segundo se producen ligeros cambios en ésta. En resumen, es añadida una neurona más a cada una de las dos capas ocultas. El propósito final es observar la disminución del valor del error a lo largo de las iteraciones en los distintos casos. Sin embargo, a la hora de aplicar el algoritmo al último conjunto se busca también observar si la misma aproximación en los resultados se da en el caso de los pesos y sesgos, es decir, si los pesos y sesgos que dan el resultado final son aproximadamente los mismos que generaron el conjunto de entrenamiento. Se ha establecido un valor de 0,05 para la tasa de aprendizaje.

3.2.1. Conjunto de Datos 1.1

El primer conjunto de datos a estudiar consiste en dos círculos concéntricos, cada uno representado por una salida binaria (0 o 1). Podemos visualizar este conjunto de entrenamiento en la Figura 3.1. Este conjunto de datos ha sido generado mediante el siguiente código:

```
circulo <- function(x, R, centroX=0, centroY=0){
  r = R * sqrt(runif(x))
  theta = runif(x) * 2 * pi
  x = centroX + r * cos(theta)
  y = centroY + r * sin(theta)

  z = data.frame(x = x, y = y)
  return(z)
}
set.seed(123456)
datos1 <- circulo(150,0.35, 0, 0)
datos2 <- circulo(150,1.5)
## Respuesta Binaria
datos1$Y <- 1
```

```
datos2$Y <- 0
datos <- rbind(datos1,datos2)
```

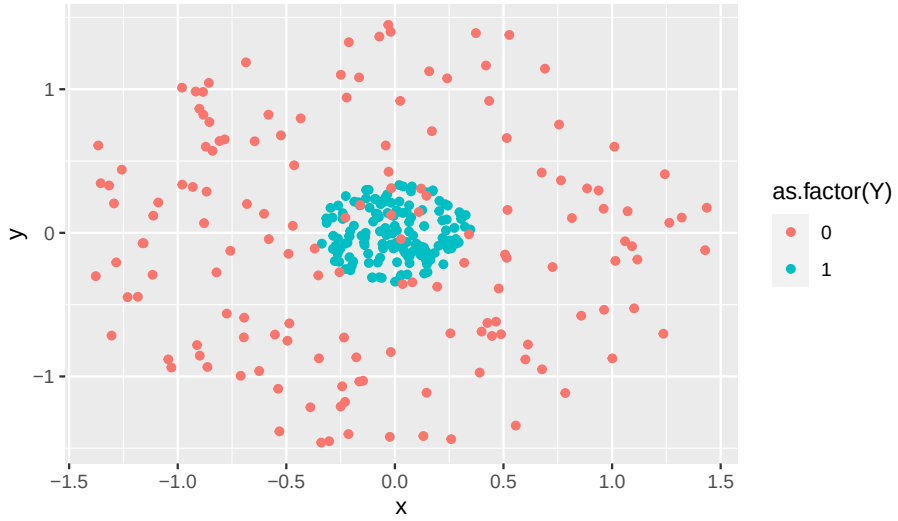


Figura 3.1: Representación Conjunto de Datos 1

Una vez obtenidos los datos del conjunto de entrenamiento, se va a presentar la estructura de la red neuronal artificial aplicada al mismo. Esta va a constar de los siguientes elementos:

- **Conjunto de entrada**: En este caso, consiste en un conjunto de pares de datos, es decir, la entrada a la neurona está organizada por parejas.
- **Capas ocultas**: Esta red va a constar de dos capas ocultas, cada una de las cuales va a presentar 2 neuronas.
- **Conjunto de salida**: La salida consiste en un único valor que podrá ser 0 o 1, dependiendo del círculo inicial al que pertenezca.
- **Función de activación**: En este modelo se ha optado por usar en todas las capas la función sigmoide (función explicada en la Subsección 1.4).

Una vez que está todo listo, se procede a entrenar la red, disminuyendo así el valor del error. Tras esto, se obtiene la evolución del error a lo largo de las iteraciones (Figura 3.2). Se puede observar que el error decrece hasta valores menores que 0,1. Una vez visto el descenso del error, es fácil observar que el error no disminuye hasta 0. Por lo tanto, sigue habiendo valores que la red no es capaz de clasificar bien. Por ese motivo, se ha desarrollado una tabla de contingencia que nos indica el número de valores que están correcta e incorrectamente clasificados.

```
> xtabs(~as.factor(Y)+ predichos_v2)
            predichos_v2
as.factor(Y) FALSE TRUE
           0   117   33
           1     2  148
```

Se aprecian una cantidad total de 35 datos mal clasificados frente a 265 correctamente clasificados. Además, visualizando la aproximación con un mallado de resultados el cual representa la aplicación de la función de la red F a la región $[-2,2] \times [-2,2]$ tomando puntos con una distancia de 0,025 y comparándolo con los puntos superpuestos al mismo los cuales corresponden al conjunto de entrenamiento, se puede verificar que esta red no es capaz de identificar la presencia de círculos tras estos datos (Figura

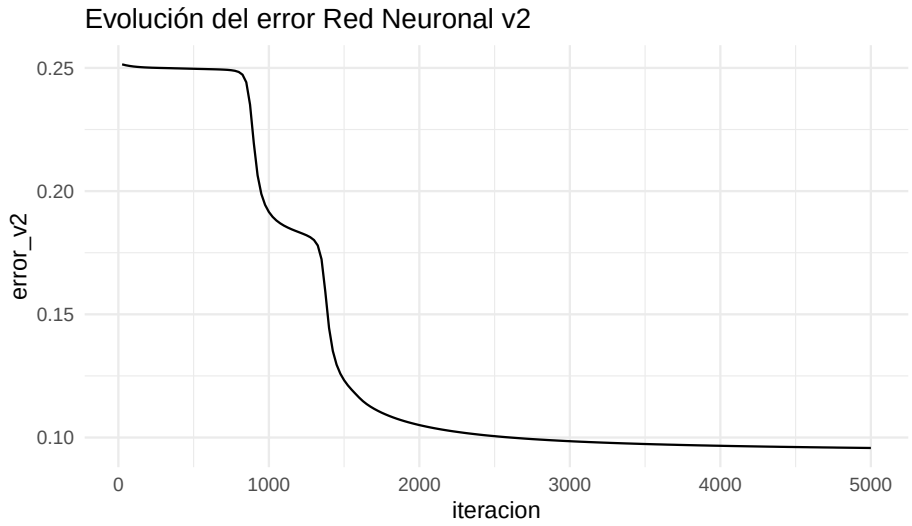


Figura 3.2: Evolución del Error

3.3). Esto se debe a que los círculos presentan una separación casi lineal. Por lo tanto, otros algoritmos creados para este tipo de problemas serían más convenientes. Por otro lado, se está aplicando el algoritmo a una red simple, aplicándolo a una red más extensa se obtendrían mejores resultados.

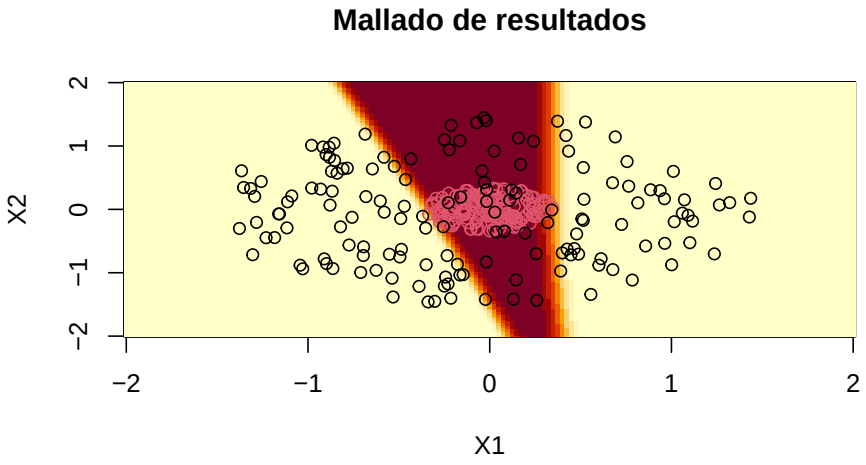


Figura 3.3: Mallado Conjunto de Datos 1

Una vez visto que esta red neuronal es capaz, aunque no sea de forma perfecta, de clasificar los datos se realizan una serie de modificaciones tanto en el conjunto de datos como en la propia estructura de la red neuronal artificial para poner a prueba la eficacia del algoritmo.

3.2.2. Conjunto de Datos 1.2

El objetivo ahora es demostrar que el código se acomoda a diferentes estructuras aplicando simplemente algunos cambios. Los cambios realizados son los siguientes:

- **Cambios en el Conjunto de Datos**: Este nuevo conjunto de datos se ha obtenido también de dos círculos. Sin embargo, en este caso los círculos no van a ser concéntricos. Los cambios realizados en el código son los siguientes:

```
datos1 <- circulo(150,0.75, -0.25, 0)
datos2 <- circulo(150,0.5,0.25,0)
```

El conjunto de datos tendrá la siguiente forma:

Figura 3.4: Representación Conjunto de Datos 2

- **Cambios en la Estructura**: Esta nueva red neuronal artificial mantiene la estructura de una capa de entrada, dos capas ocultas y una capa de salida. Sin embargo, en este nuevo modelo se añade una neurona más a cada capa oculta, resultando en un total de 3 neuronas en cada una de estas capas. (Los cambios en el código son visibles en el Anexo B).

Tras presentar todos los cambios realizados se aplica el algoritmo, este facilita la siguiente gráfica del error con respecto a las iteraciones:

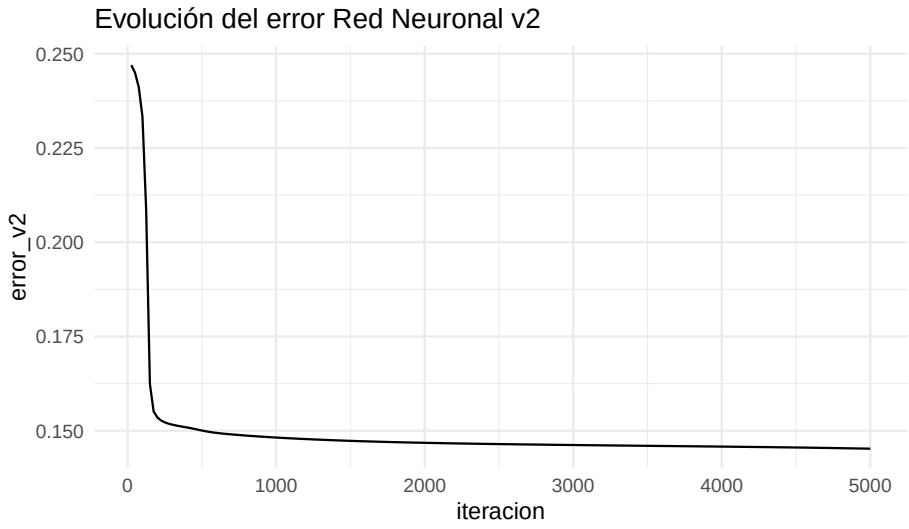


Figura 3.5: Evolución del Error

En ella se detecta la disminución del error hasta valores entre 0,1 y 0,15. Este error es mayor que el del conjunto de datos anterior, esto se debe a la distribución de los círculos. Estos círculos presentan los datos más mezclados que los anteriores y por lo tanto dificultan la labor de clasificación de la red. Al igual que en el caso anterior, se ha desarrollado una tabla de contingencia que muestra los valores tanto correcta como incorrectamente clasificados:

```
    > xtabs(~as.factor(Y)+ predichos_v2)
             predichos_v2
as.factor(Y) FALSE TRUE
           0   141    9
           1    59   91
```

En este caso, un total de 68 datos han sido mal clasificados frente a 232 correctamente clasificados. Asimismo, visualizando la aproximación con un mallado de resultados de la misma forma que en el caso 3.2.1, se puede verificar que esta red no es capaz de identificar la presencia de círculos tras estos datos (Figura 3.6). Luego de haber realizado un estudio del funcionamiento del algoritmo con dos conjuntos

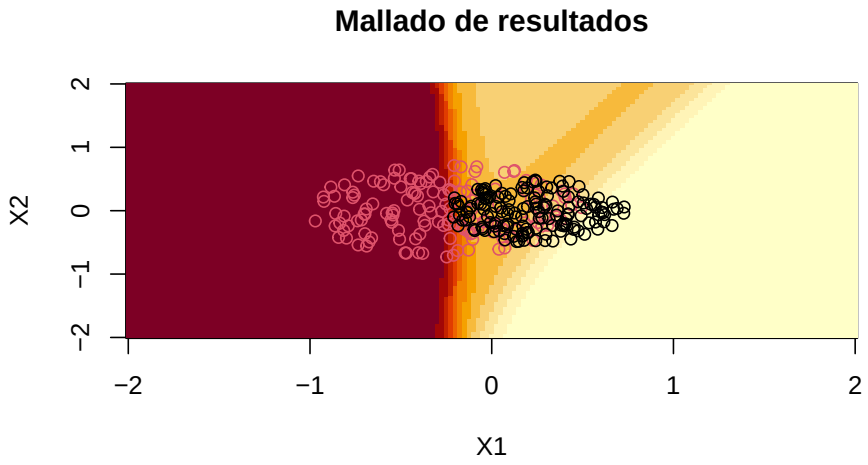


Figura 3.6: Mallado Conjunto de Datos 2

de datos generados por círculos, se va a hacer un análisis del último conjunto de datos. Este conjunto es generado por puntos aleatorios de una región determinada.

3.2.3. Conjunto de Datos 2

El objetivo actual es observar si, sacando un conjunto de entrenamiento de una serie de puntos de una región y obteniendo su salida mediante la aplicación de una red neuronal con ciertos pesos y sesgos, los pesos y sesgos de la red final, tras las iteraciones, se aproximan a los que originaron los datos. También va a ser estudiada la precisión de los resultados que genera la red. En este caso la respuesta va a ser continua, más concretamente va a ser un número en el intervalo $(0, 1)$.

Primero se observa la representación del conjunto de datos:

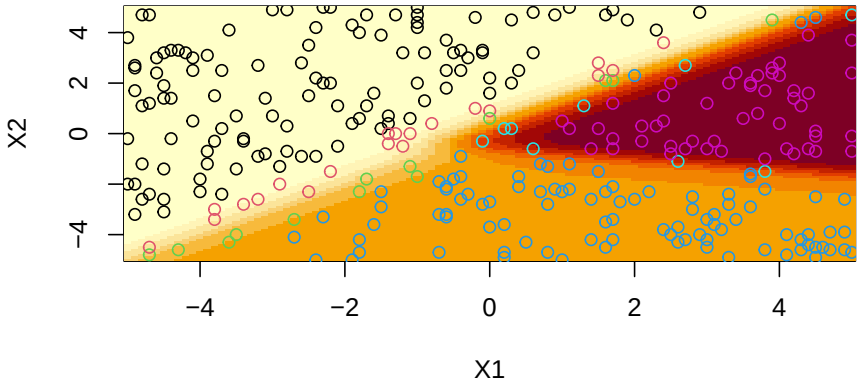


Figura 3.7: Representación del conjunto de entrenamiento

Este conjunto de datos se ha implementado mediante el siguiente código:

```
X1 = seq(-5, 5, by = 0.1)
X2 = seq(-5, 5, by = 0.1)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('x1', 'x2')
y_grid = mired(as.matrix(grid_set))

plot(X1, X2, type="n")
contour(X1,X2, matrix(as.numeric(y_grid), length(X1), length(X2)))
image(X1,X2, matrix(as.numeric(y_grid), length(X1), length(X2)))

## tomamos una muestra de tamaño 300  con puntos al azar del recinto
filas<-sample(1:length(y_grid), size= 300, replace = FALSE)
X<-as.matrix(grid_set[filas,])

Y<- y_grid[filas]
points(X, col=10*Y)
```

Se puede observar que se ha tomado una muestra de 300 elementos de dicha región para hacer el estudio. Para realizar el estudio en este conjunto de datos se ha usado la misma topología que en el primer conjunto de datos (2 neuronas en cada una de las dos capas ocultas, Caso 3.2.1). Tras aplicar el algoritmo se obtiene la gráfica con la evolución del error (Figura 3.8), en ella se observa que el error desciende hasta valores menores que 10^{-3}. Esto es debido a que estos sesgos y pesos se han generado manualmente sin datos ruidosos, por lo tanto, el algoritmo es capaz de hacer aproximaciones muy precisas. Para finalizar el estudio de este conjunto de datos, se comparan los valores de los pesos y sesgos originales con los estimados tras aplicar el algoritmo.

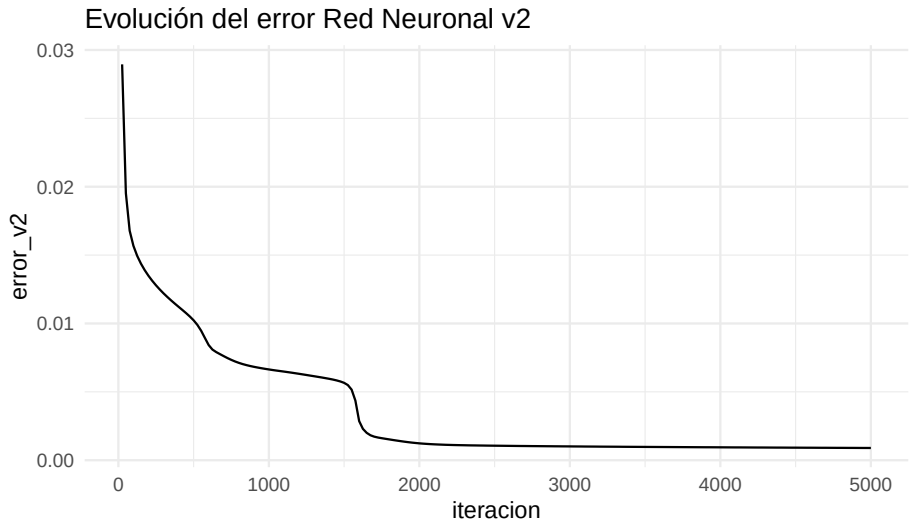


Figura 3.8: Evolución del error

Cuadro 3.1: Comparación Pesos Originales y Pesos Estimados

	Pesos Originales	Pesos Estimados
Capa 1 Neurona 1	-1	-1.873
	-5	-3.949
Capa 1 Neurona 2	2	-3.376
	-2	3.121
Capa 2 Neurona 1	-10	0.722
	-5	-1.038
Capa 2 Neurona 2	1	2.098
	-5	1.379
Capa 3	10	3.551
	-4.5	-4.195

Cuadro 3.2: Comparación Sesgos Originales y Sesgos Estimados

	Sesgos Originales	Sesgos Estimados
Capa 1 Neurona 1	-3	0.395
Capa 1 Neurona 2	2	0.105
Capa 2 Neurona 1	-7	0.406
Capa 2 Neurona 2	5	0.116
Capa 3	3	0.844

A simple vista se puede observar la no concordancia de ambos conjuntos de valores, esto se debe a que un mismo conjunto de datos se puede aproximar mediante topologías totalmente distintas.

3.3. Conclusiones

En este capítulo se ha presentado un estudio sobre la aplicación del Algoritmo de Retropropagación a tres conjuntos de datos diferentes. Dos de ellos generados por círculos y un tercero generado aleatoriamente por una red neuronal artificial.

En el primer conjunto de datos, el cual ha sido formado por círculos concéntricos, se aplica el algoritmo a una red neuronal compuesta por dos capas ocultas con dos neuronas en cada una y cuya función de activación es la sigmoide en todas las capas. Se observó una disminución en el error a lo largo de las iteraciones. Sin embargo, esta disminución se da hasta valores del error de aproximadamente 0,1, es decir, no disminuye hasta 0. Esto implica que la red no ha sido capaz de clasificar todos los datos correctamente. Puede ser debido a la poca complejidad en la estructura de la red o a que el conjunto de datos facilitados presenta una separación casi lineal. Por lo tanto, otros algoritmos centrados en este tipo de problemas habrían obtenido resultados más eficientes.

En el segundo conjunto de datos, el cual ha sido formado por círculos no concéntricos, se realizaron cambios en la estructura de la red. En cada capa oculta se añadió una neurona. Aunque el error disminuye considerablemente a lo largo de las iteraciones, este no se aproxima a 0. Esto se debe a que los datos estaban muy mezclados y la red no fue capaz de clasificarlos correctamente.

En el tercer conjunto de datos, el cual fue generado mediante otra red neuronal con la misma estructura pero pesos y sesgos establecidos, la red neuronal aproxima de forma casi perfecta los resultados. El error disminuye hasta valores aproximadamente de 10^{-3}. Sin embargo, este valor tampoco es 0, una pequeña parte de los datos han sido mal clasificados. En cuanto a la relación entre los pesos iniciales y finales, estos han resultado ser totalmente distintos. Como conclusión se obtiene que se pueden realizar aproximaciones muy precisas con redes neuronales artificiales completamente distintas.

Con este estudio se concluye que el Algoritmo de Retropropagación resulta ser eficiente. Sin embargo, conviene explorar otra técnicas de redes neuronales en función de las especificaciones de cada problema.

Como conclusión de todo el Trabajo de Fin de Grado, se han logrado exponer conceptos fundamentales y aplicaciones prácticas de redes neuronales artificiales y el Algoritmo de Retropropagación. Con estos conocimientos, el lector es capaz de explorar al completo el potencial que presentan las redes neuronales artificiales en su día a día y en sus estudios.

Bibliografía

[1] A. FERNÁNDEZ, *How to code a Neural Network from scratch in R*, fuente consultada en Mayo de 2023, obtenida en: https://anderfernandez.com/en/blog/how-to-code-a-neural-network-from-scratch-in-r/.

[2] L. FERREIRA, *An Overview Of Artificial Neural Networks for Mathematicians*, University of Chicago, 2018, fuente consultada en Febrero de 2023, obtenida en: https://math.uchicago.edu/~may/REU2018/REUPapers/Guilhoto.pdf.

[3] J. MATICH, *Redes Neuronales: Conceptos Básicos y Aplicaciones*, Universidad Tecnológica Nacional, Rosario, 2001, fuente consultada en Mayo de 2023, obtenida en: https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf.

[4] R. ROJAS, *Neural Networks. A Systematic Introduction*, Springer-Verlag, Berlin, 1996, fuente consultada en Febrero de 2023, obtenida en la página web: https://page.mi.fu-berlin.de/rojas/neural/.

[5] R. SATHYA, A. ABRAHAM, *Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification*, (IJARAI) International Journal of Advanced Research in Artificial Intelligence, 2013, fuente consultada en Mayo de 2023, obtenida en: https://www.researchgate.net/publication/273246843_Comparison_of_Supervised_and_Unsupervised_Learning_Algorithms_for_Pattern_Classification.

[6] S. SRIHARI, *Backpropagation*, fuente consultada en Febrero de 2023, obtenida en: https://cedar.buffalo.edu/~srihari/CSE574/Chap5/Chap5.3-BackProp.pdf.

[7] M. ŠTĚPNIČKA, L. ŠTĚPNIČKA, *Introduction to Soft Computing*, University of Ostrava, fuente consultada durante el curso académico 2022/2023.

[8] P. SURMENOK, *Estimating an Optimal Learning Rate For a Deep Neural Network*, fuente consultada en Julio de 2023, obtenida en: https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0.

[9] WATELECTRONICS, *What is Backpropagation Neural Network And Its Working*, fuente consultada en Junio de 2023, obtenida en: https://www.watelectronics.com/back-propagation-neural-network/.

[10] WORD INC, *Canva*, Perth, Australia, 2012, aplicación descargada en Apple Store en Mayo de 2023, https://apps.apple.com/es/app/canva-dise%C3%B1o-foto-y-v%C3%ADdeo/id897446215.

Anexo A

Demostración de la Proposición 1

Esta demostración ha sido obtenida de *Neural Networks. A Systematic Introduction* [4], más concretamente del Capítulo 7.

Demostración. Ya hemos demostrado que el algoritmo funciona para unidades en serie, unidades en paralelo y también para conexiones con pesos. Hagamos la hipótesis de inducción de que el algoritmo funciona para cualquier red feed-forward con n o menos capas. Consideremos ahora el B-diagrama de la Figura A, que contiene $n+1$ capas. El paso feed-forward se ejecuta primero y el resultado de la única unidad de salida es la función de red F evaluada en x. Supongamos que m unidades, cuyas respectivas salidas son $F_1(x), \cdots, F_m(x)$ están conectados a la unidad de salida. Dado que la función primitiva de la unidad de salida es φ, sabemos que

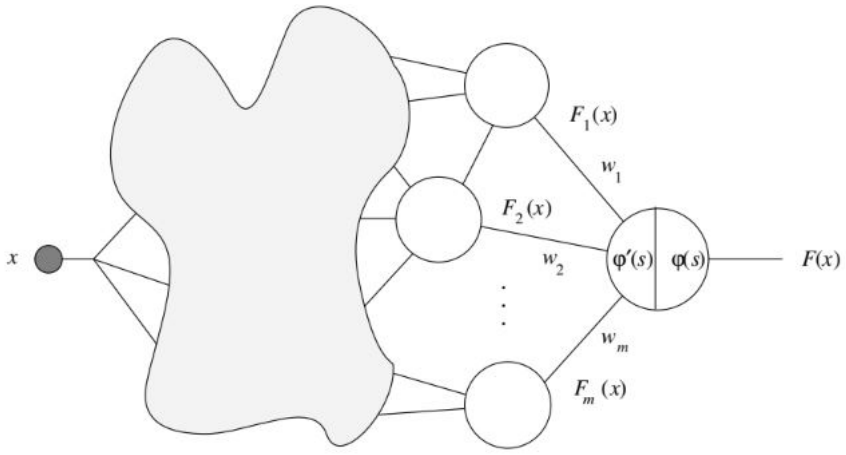


Figura A.1: Retropropagación en la última neurona. Figura tomada de *Neural Networks. A Systematic Introduction*[2]

$$F(x) = \varphi(w_1F_1(x) + w_2F_2(x) + \cdots + w_mF_m(x))$$

La derivada de F en x es

$$F'(x) = \varphi'(s)(w_1F_1'(x) + w_2F_2'(x) + \cdots + w_mF_m'(x)),$$

donde $s = w_1F_1(x) + w_2F_2(x) + \cdots + w_mF_m(x)$. El subgrafo del grafo principal que incluye todos los caminos posibles desde la unidad de entrada hasta la unidad cuya salida es $F_1(x)$ define una subred cuya función de red es F_1 y que consta de n o menos capas. Por el supuesto de inducción podemos calcular la derivada de F_1 en x, introduciendo un 1 en la unidad y recorriendo el subgrafo hacia atrás. Lo mismo se puede hacer con las unidades cuyas salidas son $F_2(x), \cdots, F_m(x)$. Si en lugar de 1 introducimos la constante $\varphi'(s)$ y la multiplicamos por w_1 obtenemos $w_1F_1'(x)\varphi'(s)$ en la unidad de entrada en el

paso de retropropagación. De forma similar obtenemos $w_2F_2'(x)\varphi'(s),\cdots,w_mF_m'(x)\varphi'(s)$ para el resto de unidades. En el paso de retropropagación con toda la red sumamos estos m resultados y obtenemos finalmente:

$$\varphi'(s)(w_1F_1'(x)+w_2F_2'(x)+\cdots+w_mF_m'(x))$$

que es la derivada de F evaluada en x. Obsérvese que la introducción de las constantes $w_1\varphi'(s),\cdots,w_m\varphi'(s)$ en las m unidades conectadas a la unidad de salida puede hacerse introduciendo un 1 en la unidad de salida, multiplicando por el valor almacenado $\varphi'(s)$ y distribuyendo el resultado a las m unidades a través de las conexiones con pesos $w_1,w_2,\cdots,w_m$. De hecho, estamos ejecutando la red hacia atrás, como exige el algoritmo de retropropagación. Esto significa que el algoritmo funciona con redes de $n+1$ capas y con esto concluye la prueba. □

Anexo B

Código de R usado en el Capítulo 3

Estos son los códigos usados en el Capítulo 3 para la aplicación del algoritmo.

ANNBPv2.R

```
#genera una clase con los parametros necesarios en cada capa
neurona <- setRefClass(
  "neurona",
  fields = list(
    fun_act = "list",
    numero_conexiones = "numeric",
    numero_neuronas = "numeric",
    W = "matrix",
    b = "numeric"
  ),
  methods = list(
    initialize = function(fun_act, numero_conexiones, numero_neuronas)
    {
      fun_act <<- fun_act
      numero_conexiones <<- numero_conexiones
      numero_neuronas <<- numero_neuronas
      W <<- matrix(runif(numero_conexiones*numero_neuronas),
                   nrow = numero_conexiones)
      b <<- runif(numero_neuronas)
    }
  )
)
#define la f sigmoide y su derivada
sigmoid = function(x) {
  y = list()
  y[[1]] <- 1 / (1 + exp(-x))
  y[[2]] <- x * (1 - x)
  return(y)
}
#gráfica de la función:
x <- seq(-5, 5, 0.01)
plot(x, sigmoid(x)[[1]], col='blue')

#feedforward
entrenar_v2 <- function(red_v2, X,Y, coste){
```

```
  out = list()
  out[[1]] <- append(list(matrix(0,ncol=2,nrow=1)), list(X))

  for(i in c(1:(length(red)))){
    d<-dim(out[[length(out)]][[2]] %*% red_v2[[i]]$W)
    mb<-matrix(red_v2[[i]]$b, nrow=d[1], ncol=d[2], byrow=TRUE)
    z = list((out[[length(out)]][[2]] %*% red_v2[[i]]$W + mb))
    a = list(red_v2[[i]]$fun_act[[1]](z[[1]])[[1]])
    out[[i+1]] <- append(z,a)
  }
  return(out)
}

#funcion del error
coste <- function(Yp,Yr){
  y <- list()
  y[[1]] <- mean((Yp-Yr)^2)
  y[[2]] <- (Yp-Yr)
  return(y)
}

forward <- entrenar_v2(red_v2, X,Y, coste)
forward[[4]][[2]]

#esto es la backpropagation
backprop <- function(out, red_v2, lr = 0.1){

  delta <- list()

  for (i in rev(1:length(red_v2))){
    z = out[[i+1]][[1]]
    a = out[[i+1]][[2]]

    if(i == length(red_v2)){
      delta[[1]] <- coste(a,Y)[[2]] * red_v2[[i]]$fun_act[[1]](a)[[2]]
    } else{
      delta <- list(delta[[1]] %*% W_temp * red_v2[[i]]$fun_act[[1]](a)[[2]],delta)
    }

    W_temp = t(red_v2[[i]]$W)

    red_v2[[i]]$b <- red_v2[[i]]$b - mean(delta[[1]]) * lr
    red_v2[[i]]$W <- red_v2[[i]]$W - t(out[[i]][[2]]) %*% delta[[1]] * lr
  }
  x = list()
  x[[1]] <- red_v2
  x[[2]] <- out[[length(out)]][[1]]
  return(x)

}
```

```
red_neuronal_v2 <- function(red_v2, X,Y, coste,lr = 0.05){
  ## Front Prop
  out = list()
  out[[1]] <- append(list(matrix(0,ncol=4,nrow=1)), list(X))

  for(i in c(1:(length(red_v2)))){
    d<-dim(out[[length(out)]][[2]] %*% red_v2[[i]]$W)
    mb<-matrix(red_v2[[i]]$b, nrow=d[1], ncol=d[2], byrow=TRUE)
    z = list((out[[length(out)]][[2]] %*% red_v2[[i]]$W + mb))
    a = list(red_v2[[i]]$fun_act[[1]](z[[1]])[[1]])
    out[[i+1]] <- append(z,a)
  }


  ## Backprop & Gradient Descent
  delta <- list()

  for (i in rev(1:length(red_v2))){
    z = out[[i+1]][[1]]
    a = out[[i+1]][[2]]

    if(i == length(red_v2)){
      delta[[1]] <- coste(a,Y)[[2]] * red_v2[[i]]$fun_act[[1]](a)[[2]]
    } else{
      delta <- list(delta[[1]] %*% W_temp * red_v2[[i]]$fun_act[[1]](a)[[2]],delta)
    }

    W_temp = t(red_v2[[i]]$W)

    red_v2[[i]]$b <- red_v2[[i]]$b - mean(delta[[1]]) * lr
    red_v2[[i]]$W <- red_v2[[i]]$W - t(out[[i]][[2]]) %*% delta[[1]] * lr

  }
  return(out[[length(out)]][[2]])

}
```

ANNsimulator.R

```
mifun<-function(x){
if(length(x)== 0) return(print("vector de dimension 0"))
y<-1/(1+exp(-x))
return(y)
}

#PARA CASO DE 2 NEURONAS EN LAS CAPAS OCULTAS

W1<-matrix(0,2,2) # pesos primera capa
W1<-matrix(c(-1,-5,2,-2),2,2)
b1<-c(-3,2)  #% bias primera capa
```

```
z1<-rep(0, 2) # output

W2<-matrix(0,2,2) # pesos primera capa
W2<-matrix(c(-10,-5,1,-5),2,2)
b2<-c(-7,5) #% bias primera capa

z2<-rep(0, 2)

#PARA 3 NEURONAS EN LAS CAPAS OCULTAS

W1<-matrix(0,3,3) # pesos primera capa
W1<-matrix(c(-1,-5,2,-2,4,-3,-1,3,2),3,3)
b1<-c(-3,2,-1)  #% bias primera capa

z1<-rep(0, 3) # output

W2<-matrix(0,3,3) # pesos primera capa
W2<-matrix(c(-10,-5,1,-5,-2,3,1,-2,-2),3,3)
b2<-c(-7,5,-2) #% bias primera capa

z2<-rep(0, 3)

Wout<-rep(0,2)
Wout<-c(10,-4.5)
bout<- 3
zout<-0


mired<-function(x,nW1=W1, nb1=b1,nW2=W2, nb2=b2, nWout=Wout, nbout=bout)
{
  if(nrow(nW1)!=ncol(x)) return(print("error dimensiones"))
  nb1<-as.numeric(b1)
  nb2<-as.numeric(b2)
  z1<-t(apply(x%*%nW1 + matrix(nb1, nrow=nrow(x), ncol=length(nb1), byrow=TRUE),1, FUN=mifun)
  print(dim(z1))

  z2<-t(apply(z1%*%nW2 + matrix(nb2, nrow=nrow(z1),ncol=length(nb2),byrow=TRUE),1,FUN=mifun))
  print(dim(z2))
  zout<-apply(z2%*%nWout+nbout,1, FUN=mifun)
  return(zout)
}

## Primer modelo a simular
## Se simula toda una región y luego se toma una muestra aleatoria de puntos
## de la region.

X1 = seq(-5, 5, by = 0.1)
X2 = seq(-5, 5, by = 0.1)

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('x1', 'x2')
```

```
y_grid = mired(as.matrix(grid_set))

plot(X1, X2, type="n")
contour(X1,X2, matrix(as.numeric(y_grid), length(X1), length(X2)))
image(X1,X2, matrix(as.numeric(y_grid), length(X1), length(X2)))


## tomamos una muestra de tamaño 300  con puntos al azar del recinto

filas<-sample(1:length(y_grid), size= 300, replace = FALSE)
X<-as.matrix(grid_set[filas,])

Y<- y_grid[filas]  ## respuesta continua

points(X, col=10*Y)

## una vez entrenada y con buena convergencia, se toman los parámetros de  red_v2
## y se compara  las curvas de nivel (image) de esta red estimada

y_grid_2<-mired(as.matrix(grid_set),
                nW1=red_v2[[1]]$W,
                nb1=red_v2[[1]]$b,
                nW2=red_v2[[2]]$W,
                nb2=red_v2[[2]]$b,
                nWout=red_v2[[3]]$W,
                nbout=red_v2[[3]]$b
                )

plot(X1, X2, type="n")
image(X1,X2, matrix(as.numeric(y_grid_2), length(X1), length(X2)))
points(X, col=(Y>.5)+1)

#####################################
### Segundo Modelo
## simulación de dos circulos
#####################################

circulo <- function(x, R, centroX=0, centroY=0){
  r = R * sqrt(runif(x))
  theta = runif(x) * 2 * pi
  x = centroX + r * cos(theta)
  y = centroY + r * sin(theta)

  z = data.frame(x = x, y = y)
  return(z)
}

set.seed(123456)
datos1 <- circulo(150,0.35, 0, 0) ## cambiar el centro (0,0) por otros valores si se desea
datos2 <- circulo(150,1.5)
```

```
## Respuesta Binaria

datos1$Y <- 1
datos2$Y <- 0
datos <- rbind(datos1,datos2)

rm(datos1,datos2)

library(ggplot2)
ggplot(datos,aes(x,y, col = as.factor(Y))) + geom_point()


X <- as.matrix(datos[,1:2])
Y <- as.matrix(datos[,3])

##mallado completo alrededor
X1 = seq(-2, 2, by = 0.025)
X2 = seq(-2, 2, by = 0.025)

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('x1', 'x2')
rm(datos)

#Esta parte se usa con la primera simulación, da como resultado
#todos los pesos y bias de la red neuronal inicial y final.
primerospesos<-data.frame(pesos_capa_1=W1,
                          pesos_capa_2=W2,
                          pesos_capa_3=Wout)
nuevospesos<-data.frame(pesos_capa_1=round(red_v2[[1]]$W,3),
                        pesos_capa_2=round(red_v2[[2]]$W,3),
                        pesos_capa_3=round(red_v2[[3]]$W,3))
primerospesos
nuevospesos
primerosbias<-data.frame(bias_capa_1=b1,
                         bias_capa_2=b2,
                         bias_capa_3=bout)
nuevosbias<-data.frame(bias_capa_1=round(red_v2[[1]]$b, 3),
                       bias_capa_2=round(red_v2[[2]]$b, 3),
                       bias_capa_3=round(red_v2[[3]]$b,3))
primerosbias
nuevosbias
```

entrenarANN.R

```
### Definición, entrenamiento y validación de una red

## definición
### poner la topología deseada
capas <- c(2,2,2,1)
#capas<-c(2,3,3,1)
funciones <- list(sigmoid, sigmoid, sigmoid)
```

```
### crear la red con la topología deseada

red_v2<- list()

for (i in 1:(length(capas)-1)){
  set.seed(12345)
  red[[i]] <- neurona$new(funciones[i],capas[i], capas[i+1])
  set.seed(12345)
  red_v2[[i]] <- neurona$new(funciones[i],capas[i], capas[i+1])
}

print(red_v2)

## Entrenamiento con un data_set  fijado

## X es la matriz de datos de entrada (asumimos que tiene dos columnas)
## Y es el vector respuesta (puede ser numérico o de tipo factor)

## X,Y pueden ser del modelo 1 o del modelo 2 (circulos)

resultado_v2 <- red_neuronal_v2(red_v2, X,Y, coste)
dim(resultado_v2)

niteraciones<-5000


for(i in seq(niteraciones)){

  Yt_v2 = red_neuronal_v2(red_v2, X,Y, coste, lr=0.05)


  if(i %% 25 == 0){
    if(i == 25){
      iteracion <- i
      error_v2 <- coste(Yt_v2,Y)[[1]]
    }else{
      iteracion <- c(iteracion,i)
      error_v2 <- c(error_v2,coste(Yt_v2,Y)[[1]])

    }
  }
}


## Evolución del error

library(ggplot2)

grafico_v2 = data.frame(Iteracion = iteracion,Error = error_v2)

ggplot(grafico_v2,aes(iteracion, error_v2)) + geom_line() + theme_minimal() +
```

```
  labs(title = "Evolución del error de la Red Neuronal v2")

## Nivel de acierto sobre el conjunto de entrenamiento

resultado_v2 <- entrenar_v2(red_v2, X,Y, coste)

plot(Y,resultado_v2[[4]][[2]],  col=Y+1)

predichos_v2<-as.factor(resultado_v2[[4]][[2]] > .5)

xtabs(~as.factor(Y)+ predichos_v2)

## función aproximante en un entorno del conjunto de entrenamiento

res_mallado_v2<- entrenar_v2(red_v2, as.matrix(grid_set), coste)

contour(X1,X2, matrix(res_mallado_v2[[4]][[2]], length(X1), length(X2)))
image(X1,X2, matrix(res_mallado_v2[[4]][[2]], length(X1), length(X2)))
points(X, col=Y+1)
title("Mallado de resultados")
```