UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

“ESTRUCTURA DE REDES NEURONALES (MLP) Y SU APLICACIÓN COMO APROXIMADOR UNIVERSAL”

MONOGRAFÍA DE TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE MATEMÁTICO
PROYECTO CURRICULAR DE MATEMÁTICAS

Lexly Vanessa Sosa Jerez y Laura Camila Zamora Alvarado
Dirigido por: Luis Alejandro Másmela Caita

Bogotá DC
Julio de 2022

Resumen

Uno de los principales modelos de aprendizaje supervisado del Machine Learning son las redes neuronales artificiales, su nombre y estructura están inspirados en las redes neuronales del cerebro humano, e imitan la forma en la que las neuronas biológicas interactúan entre sí.

El presente trabajo está orientado al estudio del sustento teórico de las redes neuronales artificiales (MPL - Multilayer Perceptron), frente a su capacidad de clasificación. Para este propósito se definen formalmente los conceptos que permiten la comprensión matemática de las mismas, además de la implementación de dichas nociones en la elaboración de un código que muestra, a través de ejemplos particulares, la capacidad de clasificación de las redes neuronales con base a la teoría aquí desarrollada. Adicionalmente se evidencia la capacidad de las redes neuronales para aproximar funciones, a través de un ejemplo aplicado al código desarrollado, este resultado es posible gracias al **Teorema de Aproximación Universal de Funciones**, el cual construye un puente entre las redes neuronales y la teoría de aproximación que no solo es aplicable a funciones, sino que , tal y como un método numérico, también permite aproximarse a la solución de sistemas de ecuaciones diferenciales parciales y ordinarias.

Palabras clave: Red Neuronal Artificial · Perceptrón simple · Teorema de Convergencia del Perceptrón · Descenso del Gradiente · Algoritmo · Perceptrón Multicapa · Aproximación de funciones continuas · Teorema de Aproximación Universal.

Clasificación AMS: 92B20, 68T01, 68T15

Agradecimientos:

Lexly Vanessa Sosa Jerez:
Agradezco en primera instancia a mis padres por toda la ayuda incondicional y el apoyo moral a lo largo de este proceso académico, a mis compañeros que hicieron más ameno mi paso por la universidad. A mi compañera y amiga Camila Zamora por su apoyo moral durante el desarrollo de este trabajo.

Laura Camila Zamora Alvarado:
Agradezco en primera instancia mi familia, y a todas las personas involucradas en mi proceso de formación quienes me apoyaron con su paciencia y compromiso a lo largo de la carrera. particularmente agradezco a mi compañera Vanessa Sosa por su apoyo, y su talante como matemática y amiga.

Conjuntamente queremos agradecer a la Universidad Distrital y la planta docente del Proyecto Curricular de Matemáticas por todos los conocimientos brindados, especialmente a los docentes: Carolina Mejía, Andrés Giraldo, Martín Barajas, por su disposición y paciencia en las explicaciones requeridas y por marcar nuestro proceso académico de forma positiva. También agradecemos al egresado Daniel Meshir por su colaboración y asesoría en cuanto lenguajes de programación, para el desarrollo de los códigos implementados en el trabajo. Por último a nuestro director Alejandro Másmela por confiar inicialmente en nosotras y nuestro trabajo.

1. Introducción

Las redes neuronales solucionan muchos problemas del análisis de datos, como reconocimiento de imágenes, reconocimiento de voz y el procesamiento del lenguaje natural, etc. Existen diferentes tipos de redes neuronales, como lo son las redes neuronales recurrentes o las redes neuronales convoluciones, las cuales surgieron de la teoría de redes MLP. [8]

Las redes neuronales artificiales son un modelo de aprendizaje automático que hace posible que las máquinas ejecuten tareas propias haciendo una analogía al cerebro humano; estas fueron introducidas por Warren McCulloch y Walter Pitts en “Un cálculo lógico de las ideas inmanentes a la actividad nerviosa"(1943) [11] que fue reconocido como el primer trabajo realizado en inteligencia artificial.

El orden escogido para el trabajo busca inicialmente ahondar en el surgimiento y arquitectura de las redes neurales, haciendo énfasis en el perceptrón simple, exhibiendo el algoritmo de aprendizaje de éste, formalizando los conceptos inherentes a su estructura, así como demostrando el teorema de convergencia del perceptrón, el cual es base en la teoría de redes neuronales artificiales.

La segunda sección está enfocada en desarrollar minuciosamente las matemáticas que están presentes en el funcionamiento de las redes neuronales MLP, exponiendo temas primordiales para el entendimiento del trasfondo matemático de las mismas, con conceptos como: funciones de activación, funciones de costo, el algoritmo de descenso del gradiente, el entrenamiento de la red constituido por la pre-alimentación y retro-propagación, y el aprendizaje de la misma.

Finalmente, la tercera parte de este trabajo está enfocada en presentar la sección: Redes Neuronales como aproximador Universal de Funciones, donde se expone el poder y la utilidad de las redes neuronales para aproximar funciones continuas de una o múltiples variables, esta sección está basada en el capítulo 4 del libro neural networks and deep learning (2015) [8], adicionalmente se evidencia que la utilidad de la teoría redes para aproximación, permite encontrar soluciones de ecuaciones diferenciales parciales y ordinarias, basado en el articulo: Artificial Neural Networks for Solving Ordinary and Partial Differential Equations (1998) [7].

2. ¿Qué son las Redes neuronales?

2.1. Surgimiento

El nacimiento de las redes neuronales artificiales, está íntimamente relacionado con la morfología de las neuronas biológicas y sus interconexiones. La estructura de una neurona biológica está compuesta principalmente por cuatro partes: dendritas, soma, axón y terminales axónicos; cada una de éstas, cumple una función en la transferencia y el procesamiento de la información, dado a partir de impulsos eléctricos.

Las dendritas actúan como receptores de información. El soma o cuerpo neuronal, que contiene al núcleo, es el encargado de enviar la información obtenida y procesada mediante un impulso eléctrico hacia el axón, mientras que las ramificaciones en el axón, o terminales axónicos, se encargan de transferir la información de una neurona a otra en un proceso llamado sinapsis. Este proceso solo se realiza en una dirección, es decir: la transferencia de información dada por el impulso eléctrico, va desde los terminales axónicos en una primera neurona hacia las dendritas de una segunda neurona, y solo se da en este sentido.

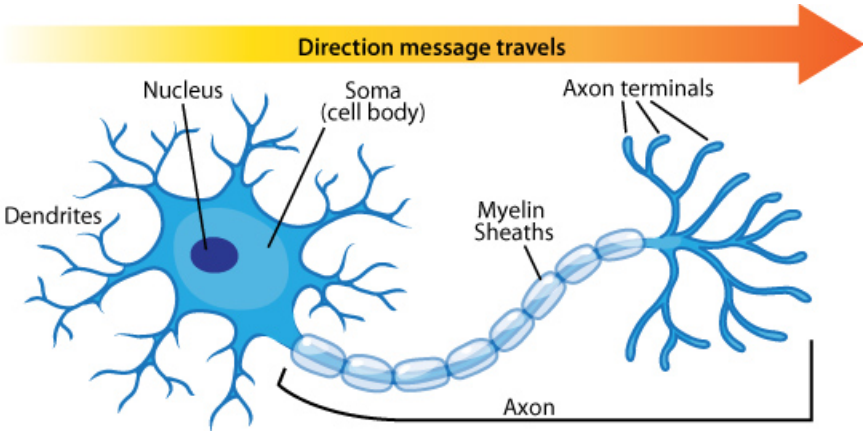


Figura 1: Neurona biológica - Tomado de : *ASU School of Life Sciences* [1]

Las redes neuronales artificiales surgen bajo la intención de representar de forma matemática una estructura capaz de procesar información de manera análoga a la interconexión entre neuronas biológicas. En este sentido, se considera la estructura de una red neuronal, como un grafo simple dirigido ponderado, con una única dirección. La unidad básica de procesamiento en una red neuronal artificial, es conocida como neurona o **Perceptrón simple**.

2.2. Arquitectura

2.2.1. Perceptrón Simple

El perceptrón simple, también cuenta con cuatro partes importantes en su estructura, estás son: entradas (intput) con conexiones y pesos (aristas ponderadas), nodo de procesamiento o suma, función de activación y salidas (output), el nodo de procesamiento, establecerá una regresión lineal que involucra la suma ponderada de los pesos en cada arista de las entradas y un término de sesgo o término independiente, el perceptrón simple funciona como un discriminador lineal, que a partir de un umbral establecido arroja una salida binaria .

Matemáticamente el **Perceptrón simple** es representado a través de la siguiente ecuación:

$$\hat{\mathbf{y}}(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b). \tag{1}$$

La arquitectura que modela dicha ecuación está dada, mediante el siguiente dígrafo simple: donde $\mathbf{x}$

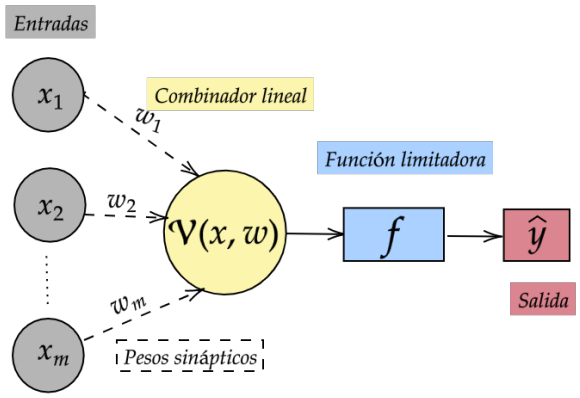


Figura 2: Modelo de perceptrón simple (arquitectura)

denota el vector de entradas, $\mathbf{w}$ denota el vector de pesos asociados a cada arista, $\boldsymbol{b}$ denota el sesgo, o intercepto de la regresión, $\mathcal{V}(\mathbf{x}, \mathbf{w})$ denota el nodo de procesamiento o combinador lineal, y $\boldsymbol{f}$ denota la función de activación o función limitadora, siendo esta una transformación no lineal de la regresión obtenida en el nodo de procesamiento.

El perceptrón simple es eficiente en el aprendizaje y solución de problemas linealmente separables, como las compuertas lógicas AND ($\wedge$) y OR ($\vee$). Sin embargo, tiene ciertas limitaciones en la resolución de problemas que no son de este tipo, un ejemplo típico de esto es su incapacidad de clasificar las salidas de una compuerta lógica del tipo XOR (ó exclusiva), ya que el nodo de procesamiento solo nos permite separar la información por medio de una única recta de regresión.

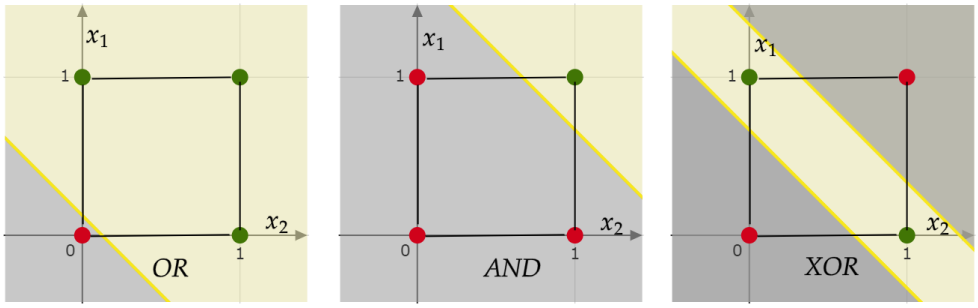


Figura 3: **Compuertas lógicas**

2.2.2. Perceptrón Multicapa (MLP)

Para dar solución al problema de la puerta lógica XOR es suficiente con agregar otra neurona que permite definir una nueva recta de regresión como se muestra en la Figura 3, esto nos permite definir lo que se conoce como **Perceptrón Multicapa o MLP** (por sus iniciales en inglés), también conocido como **Deep neural network**, se trata de una generalización del perceptrón simple, que consta de más de un nivel de neuronas y/o de una o varias capas de neuronas "entre" la capa de entradas y la capa de salidas -denominadas **capas ocultas**- cuyas funciones de activación entre ellas no son necesariamente lineales. Las MLP son consideradas las redes neuronales artificiales por defecto, este tipo de redes neuronales también se representan por medio de un dígrafo simple, el cual pasa las entradas de capa en capa hasta llegar a la capa final.

La red neuronal de la Figura 4 es un ejemplo de una red MLP de dos capas ocultas, en la cual los superíndices denotan la posición en las capas y los subíndices la posición relativa de cada nodo en su respectiva capa. Está compuesta por un vector de entradas $\mathbf{x} \in \mathbb{R}^{d_0}$, tal que $\mathbf{x} = (x_1, ..., x_{d_0})^T$, capas ocultas denotadas por $\mathbf{a}^l$, y un vector de salidas $\hat{\mathbf{y}} \in \mathbb{R}^{d_L}$ con $\hat{\mathbf{y}} = (\hat{y}_1, ..., \hat{y}_{d_L})^T$. Las capas ocultas serán las encargadas de albergar los nodos de procesamiento o neuronas, denotados por $\boldsymbol{a} = f(\mathbf{z})$, con f como funciones de activación de cada capa y $\mathbf{z}$ como un combinador lineal (matricial), los cuales están relacionados entre sí por medio de las aristas ponderadas por $\mathbf{w}$, que denota las matrices de pesos asociados en cada capa, y donde w_{ij}^1 representa el peso asociado a la arista que relaciona la entrada j-ésima con el i-ésimo nodo de procesamiento en la primera capa oculta. Las matrices $\mathbf{w}^l$ tendrán dimensión $(d_l \times d_{l-1})$ (donde la capa de entrada se cuenta como capa cero $l = 0$).

Cabe resaltar que el término $\boldsymbol{b}$, que denota el sesgo en el perceptrón simple, también tiene lugar en la red MLP en cada uno de los nodos de procesamiento; específicamente en el combinador lineal $\mathbf{z}$. Como una convención a partir de ahora el nodo de procesamiento incluirá el termino de sesgo $\boldsymbol{b}$, así $b^1 \in \mathbf{a}^1$ denotará el vector de sesgo en la primera capa oculta.

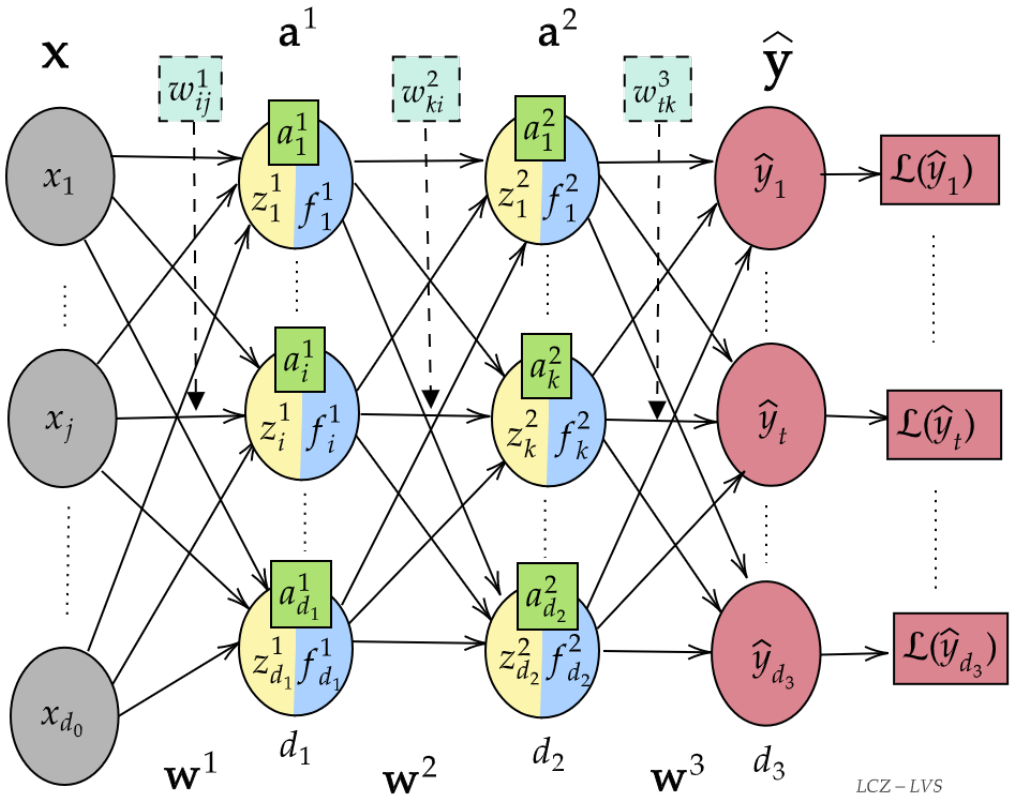


Figura 4: **Red neuronal MLP** (Arquitectura con 2 capas ocultas)

La ecuación matemática que describe a la red de la Figura 4 será:

$$\begin{aligned}\hat{\mathbf{y}} &= f^3(\mathbf{z}^3)\\ &= f^3(\mathbf{w}^3\mathbf{a}^2 + b^3)\\ &= f^3(\mathbf{w}^3(f^2(\mathbf{z}^2)) + b^3)\\ &= f^3(\mathbf{w}^3(f^2(\mathbf{w}^2\mathbf{a}^1 + b^2)) + b^3)\\ &= f^3(\mathbf{w}^3(f^2(\mathbf{w}^2(f^1(\mathbf{z}^1)) + b^2)) + b^3)\\ &= f^3(\mathbf{w}^3(f^2(\mathbf{w}^2(f^1(\mathbf{w}^1\mathbf{x} + b^1)) + b^2) + b^3),\end{aligned} \tag{2}$$

note que f^3 intuitivamente es la función de activación en la capa de salida, esta función habitualmente es tomada como la identidad, sin embargo, existirán modelos de clasificación para los que la predicción será más acertada si ésta es una función de activación limitadora, no lineal.

Por otra parte la última columna que se observa en la Figura 4 será una capa adicional en la que mediante una función de pérdida se evaluará el comportamiento de la red, relacionando la información

obtenida en la capa de salida con los datos esperados en un modelo de aprendizaje supervisado, esto se menciona con más detalle en secciones posteriores.[12]

3. Matemáticas de las Redes Neuronales

3.1. Perceptrón

Inicialmente estableceremos un conjunto de datos a estudiar, denominado $\mathbf{X} \subseteq \mathbb{R}^{m+1}$, el cual se particiona en dos clases linealmente separables $\mathscr{C}_1$ y $\mathscr{C}_2$; a los vectores $\mathbf{x} = (x_1, x_2, ..., x_m, 1)^T$ que pertenecen a $\mathbf{X}$ los llamaremos **entradas**.

Luego implementaremos un conjunto $\mathbf{W} \subseteq \mathbb{R}^{m+1}$, de etiquetas para las aristas del perceptrón, cuyos elementos $\mathbf{w} = (w_1, w_2, ..., w_m, b)^T$ llamaremos **pesos sinápticos**, donde, b es un real fijo, llamado **sesgo**. Para efectos de establecer el algoritmo del Perceptrón, de suma importancia aquí, presentamos cuatro definiciones básicas a continuación, ellas corresponden a clases linealmente separable, combinador lineal, función limitadora y función perceptrón.

Definición 1 (Clases linealmente separables). [12, pág 51] Sean $\mathscr{C}_1$ y $\mathscr{C}_2$ dos clases en un espacio $n-$dimensional. Entonces $\mathscr{C}_1$ y $\mathscr{C}_2$ son clases linealmente separables si existe un vector $\mathbf{w} \in \mathbb{R}^{m+1}$ de pesos sinápticos, de tal manera que se satisfacen las siguientes condiciones:

$$\mathbf{w}^T\mathbf{x}_1 > 0 \text{ para cada vector de entrada } \mathbf{x}_1 \in \mathscr{C}_1.$$
$$\mathbf{w}^T\mathbf{x}_2 \leq 0 \text{ para cada vector de entrada } \mathbf{x}_2 \in \mathscr{C}_2.$$

Definición 2 (Combinador lineal). Sean $\mathbf{x} = (x_1, x_2, ..., x_m, 1)^T$ y $\mathbf{w} = (w_1, w_2, ..., w_m, b)^T$ definimos la función

$$\begin{aligned} \mathcal{V} : \mathbb{R}^{m+1} \times \mathbb{R}^{m+1} &\longrightarrow \mathbb{R} \\ (\mathbf{x}, \mathbf{w}) &\longrightarrow \mathcal{V}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}, \end{aligned}$$

donde $\mathcal{V}(\mathbf{x}, \mathbf{w}) = 0$, denota el hiperplano de separación entre dos regiones de decisión.

Definición 3 (Función limitadora). Sea $\mathscr{A}$ el conjunto de todas las combinaciones lineales $\mathcal{V}(\mathbf{x}, \mathbf{w})$, consideremos $t \in \mathscr{A}$, se define la función limitadora g como sigue:

$$\begin{aligned} g : \mathscr{A} &\longrightarrow \{1, -1\} \\ t &\longrightarrow g(t) = \begin{cases} 1 & si \quad t > 0 \\ -1 & si \quad t \leq 0. \end{cases} \end{aligned}$$

Definición 4 (Función Perceptrón). Dadas $\mathcal{V}(\mathbf{x},\mathbf{w})$ y $g(t)$, se define la aplicación clasificadora, como sigue:

$$\begin{aligned}\mathscr{P} : \mathbb{R}^{m+1} \times \mathbb{R}^{m+1} &\longrightarrow \{-1,1\}\\ (\mathbf{x},\mathbf{w}) &\longrightarrow \mathscr{P}(\mathbf{x},\mathbf{w}) = g(\mathcal{V}(\mathbf{x},\mathbf{w})) = \hat{y},\end{aligned}$$

donde $\hat{y} \in \{-1,1\}$, será la salida de la función perceptrón; además la aplicación perceptrón posee una representación a través de un dígrafo simple, como se muestra en la Figura 2

A través de la función establecida en la **Definición 4**, se puede desarrollar un modelo de aprendizaje supervisado de clasificación binaria, denominado **Perceptrón**; el cual involucra las funciones definidas anteriormente con el objetivo de clasificar correctamente un conjunto de entradas **X**, linealmente separables, en dos clases; aplicando una regla de aprendizaje adaptativa sobre cada uno de los pesos sinápticos (**w**), en una cantidad finita de pasos (n). A dicho proceso se le conoce como **algoritmo de aprendizaje del Perceptrón**.

Definición 5. Considérese las entradas y los pesos sinápticos, respectivamente, en el perceptrón, $\mathbf{x}(n) = (x_1(n), x_2(n), ..., x_m(n), 1)^T$ y $\mathbf{w}(n) = (w_1(n), w_2(n), ..., w_m(n), b)^T$, definimos el combinador lineal del perceptrón, como

$$\mathcal{V}(n) = \mathbf{w}^T(n)\mathbf{x}(n),$$

donde n denota el número de iteraciones en la aplicación del algoritmo.

Vamos a considerar a $\mathscr{H} \subset \mathbf{X}$, como el subsepacio vectorial de entrenamiento, donde $\mathscr{H}_1$ será el subespacio de vectores de entrenamiento $\mathbf{x}_1(1), \mathbf{x}_1(2), ...$ que pertenecen a la clase $\mathscr{C}_1$ y $\mathscr{H}_2$ el espacio de vectores de entrenamiento $\mathbf{x}_2(1), \mathbf{x}_2(2), ...$ que pertenecen a la clase $\mathscr{C}_2$, donde, $\mathscr{H} = \mathscr{H}_1 \cup \mathscr{H}_2$. Con el fin de evitar un sobre entrenamiento, sobre alguna de las dos clases, consideraremos la misma cardinalidad para $\mathscr{H}_1$ y $\mathscr{H}_2$.

Debido a que el perceptrón es un modelo de aprendizaje supervisado se establece $y(k) \in \{-1,1\}$, como la clase a la que cada entrada $x(k)$ de $\mathscr{H}$, pertenece realmente. Note que el valor $y(k) - \hat{y}(k)$, es el *error* que comete el Perceptrón en su clasificación, y de éste se desprende la siguiente definición:

Definición 6 (Función actualización por corrección del error). Se define la regla de actualización de los pesos sinápticos como sigue:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)[y(n) - \hat{y}(n)]x(n).$$

Así:

$$\mathbf{w}(n+1) = \begin{cases}\mathbf{w}(n) + 2\eta(n)x(n) & \text{si} \quad y(n) = 1 \text{ y } \hat{y} = -1,\\ \mathbf{w}(n) & \text{si} \quad y(n) = \hat{y}(n),\\ \mathbf{w}(n) - 2\eta(n)x(n) & \text{si} \quad y(n) = -1 \text{ y } \hat{y} = 1,\end{cases}$$

donde $\eta(n) = \eta > 0$, será una regla de adaptación de incremento fijo llamada *tasa de aprendizaje.*

Teniendo en cuenta la **Definición** 6, podemos construir un algoritmo que ejecute el aprendizaje del perceptrón.

3.1.1. Algoritmo de aprendizaje del Perceptrón

Algoritmo 1 Aprendizaje del Perceptrón

Entrada: $\mathbf{x}, \mathbf{y}$
Salida: $\mathbf{w}(n+1)$

Inicialización: $\mathbf{w}(1)$,
Optimización:
para $n \leftarrow 1,..,k$ **hacer**
  $\mathcal{V}(n) \leftarrow 0$
  para $i \leftarrow 1,...,m+1$ **hacer**
    $\mathcal{V}(n) \leftarrow \mathcal{V}(n) + w_i(n)x_i(n)$
  fin para
  $\hat{\mathbf{y}}(n) \leftarrow f(\mathcal{V}(n))$
  $\mathbf{w}(n+1) \leftarrow \mathbf{w}(n) + \eta[\mathbf{y}(n) - \hat{\mathbf{y}}(n)]\mathbf{x}(n)$
fin para
Retorno $\mathbf{w}(n+1)$

3.1.2. Teorema de convergencia del perceptrón

Teorema A. *Sean $\mathscr{H}_1$ y $\mathscr{H}_2$, subconjuntos de vectores de entrenamiento linealmente separables . Considere las m entradas presentadas al perceptrón, como elementos de estos dos subconjuntos. El perceptrón converge después de n_o iteraciones, en el sentido que:*

$$\boldsymbol{w}(n_0) = \boldsymbol{w}(n_0+1) = \boldsymbol{w}(n_0+2) = \cdots,$$

es un vector solución para $n_0 \leq n_{max}$.

Demostración. Como las clases $\mathscr{C}_1$ y $\mathscr{C}_2$ son linealmente separables, existe un w^* que satisface las condiciones de la **Definición 1**.

Supongamos que en la iteración $k-$ésima, la salida del perceptrón $\hat{y}(k)$ no coincide con la salida deseada $y(k)$; por lo tanto se debe aplicar la regla de actualización sobre los pesos, donde la función de coste $\mathcal{C}$, será:

$$\mathcal{C}(k+1) = \|w(k+1) - w^*\|^2, \tag{3}$$

donde:

$$\begin{aligned}\mathcal{C}(k+1) &= \langle w(k+1)-w^*, w(k+1)-w^*\rangle \\ &= \|w(k+1)\|^2 - 2\langle w(k+1), w^*\rangle + \|w^*\|^2 \\ &= \|w(k)+\eta[y(k)-\hat{y}(k)]x(k)\|^2 - 2\langle w(k)+\eta[y(k)-\hat{y}(k)]x(k), w^*\rangle + \|w^*\|^2 \\ &= \|w(k)\|^2 - 2w^T(k)w^* + \|w^*\|^2 + 2\eta[y(k)-\hat{y}(k)]w^T(k)x(k) - 2\eta[y(k)-\hat{y}(k)]w^{*T}x(k) \\ &\quad + \eta^2[y(k)-\hat{y}(k)]^2\|x(k)\|^2,\end{aligned} \tag{4}$$

en la cual, $[y(k)-\hat{y}(k)]w^T(k)x(k) < 0$ ya que si: $w^T(k)x(k) > 0$, implica que $\hat{y}(k) = 1$; como la salida está errada, se tiene que $y(k) = -1$, así:

$$[y(k)-\hat{y}(k)]w^T(k)x(k) = -2w^T(k)x(k) < 0,$$

del mismo modo, si $w^T(k)x(k) < 0$, implica que $\hat{y}(k) = -1$; como la salida está errada, se tiene que $y(k) = 1$, así:

$$[y(k)-\hat{y}(k)]w^T(k)x(k) = 2w^T(k)x(k) < 0,$$

con lo cual, dicho termino puede ser reemplazado por:

$$-2\left|w^T(k)x(k)\right|,$$

análogamente; el término $[y(k)-\hat{y}(k)]w^{*T}x(k) > 0$, ya que por hipótesis el parámetro w^* satisface las condiciones de la Definición 1, clasificando adecuadamente las entradas del perceptrón, con lo cual, si $w^{*T}x(k) > 0$, esto implica que $y(k) = 1$, y la salida errada será $\hat{y}(k) = -1$, así:

$$[y(k)-\hat{y}(k)]w^{*T}x(k) = 2w^{*T}x(k) > 0,$$

del mismo modo, si $w^{*T}x(k) < 0$, esto implica que $y(k) = -1$, y la salida errada será $\hat{y}(k) = 1$, así:

$$[y(k)-\hat{y}(k)]w^{*T}x(k) = -2w^{*T}x(k) > 0,$$

obteniendo que dicho termino puede ser reemplazado por:

$$2\left|w^{*T}x(k)\right|,$$

por lo tanto, la función de coste $\mathcal{C}$ en la ecuación (4), será:

$$\begin{aligned}\mathcal{C}(k+1) &= \|w(k)-w^*\|^2 - 4\eta\left|w^{*T}x(k)\right| - 4\eta\left|w^T(k)x(k)\right| + 4\eta^2\|x(k)\|^2 \\ &\leq \|w(k)-w^*\|^2 - 4\eta\left|w^{*T}x(k)\right| + 4\eta^2\|x(k)\|^2,\end{aligned} \tag{5}$$

sea:

$$\mathcal{C}(k) = \|w(k) - w^*\|^2,$$

y consideremos:

$$M = \max_{1\leq k\leq m} \left\{\|x(k)\|^2\right\}, \qquad L = \min_{1\leq k\leq m} \left\{|w^{*T} x(k)|\right\}$$

donde, de la ecuación (5), se tiene:

$$\begin{aligned} \mathcal{C}(k+1) &\leq \mathcal{C}(k) - 4\eta L + 4\eta^2 M \\ &\leq \mathcal{C}(k) + 4\eta(\eta M - L), \end{aligned}$$

Si consideramos, $\eta M - L < 0$, se tiene que $\mathcal{C}(k+1) < \mathcal{C}(k)$. Así pues, si elegimos el valor η tal que $0 < \eta < \frac{L}{M}$; obtendremos que $\mathcal{C}(k+1)$ se reduce por lo menos en la cantidad constante $\eta(\eta M - L)$, en cada iteración en que se realiza una corrección; por lo tanto, el número de iteraciones con corrección debe ser finito ($n_0 \leq n_{max}$), ya que de ser infinito, en un momento dado obtendríamos que el valor de $\mathcal{C}(k)$ sería negativo, lo cual es un absurdo. □

Antes de establecer las matemáticas que desarrollan las redes neuronales MLP, se requiere un preámbulo de conceptos que permitirán la comprensión de éstas. Estos son las funciones de activación y el algoritmo de descenso del gradiente.

3.2. Funciones de Activación

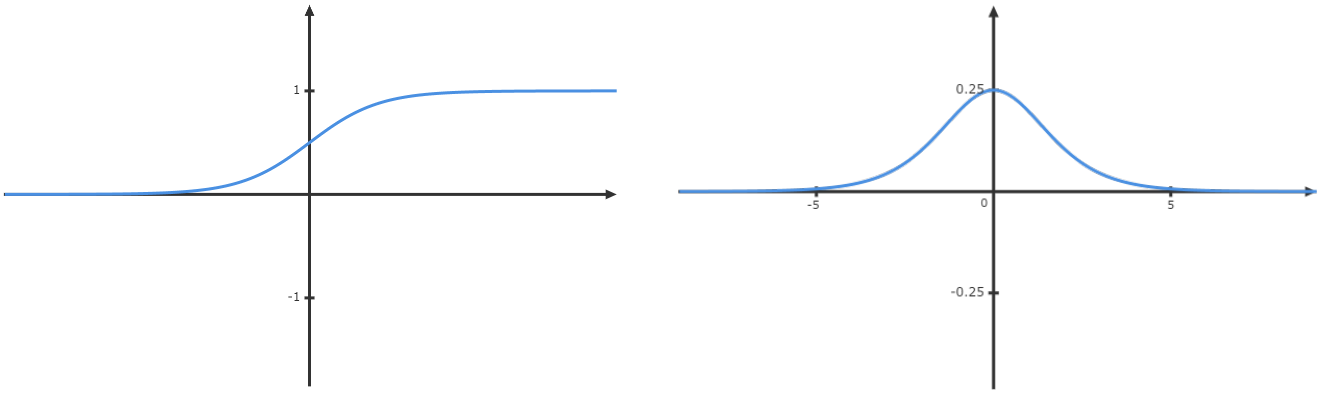
Las funciones de activación tienen un papel primordial en el funcionamiento de la red neuronal. Estas funciones transmiten la información generada por la combinación lineal de los pesos y las entradas, y la envían a otras neuronas. Como ya se ha visto anteriormente, existe una analogía entre las neuronas biológicas, y las redes neuronales artificiales, en este caso, las funciones de activación son análogas a la tasa del potencial de acción disparado en el cerebro.

Hay diferentes tipos de funciones de activación, se definirán algunas de las más utilizadas, y se mostrará en qué casos es más conveniente usar una u otra.

1. **Función sigmoidal**: La función sigmoidal y su derivada están dadas por las siguientes ecuaciones:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}. \tag{6}$$

   Las respectivas gráficas de $\sigma(x)$ y $\sigma'(x)$ se muestran en la Figura 5.



(a) Función (b) Derivada

Figura 5: Función de activación Sigmoidal y su derivada

La función de activación sigmoidea $\sigma(x) : \mathbb{R} \to [0,1]$ es una función suave y diferenciable en todo punto, esta función compacta cualquier valor de entrada en un valor entre 0 y 1. Aunque es una función muy utilizada y antigua, cuenta con inconvenientes, por ejemplo, si consideramos un valor x alejado del origen, tendremos que el gradiente de la función en x es un valor muy pequeño, esto generará un estancamiento en el proceso de backpropagation y este tardará mucho tiempo en converger, si es que lo hace.

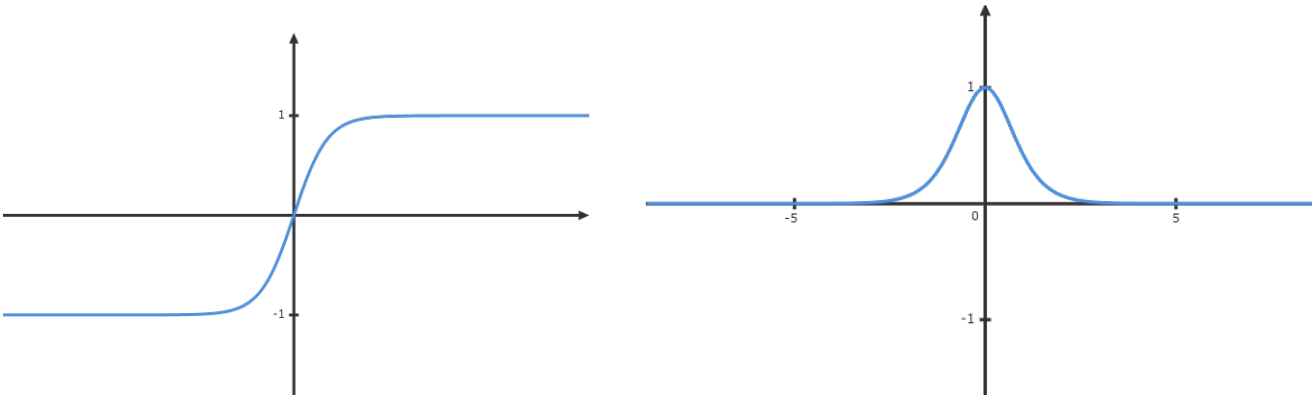
2. **Función Tangente Hiperbólica** La función de activación tangente hiperbólica es una modificación de la función sigmoidal, su definición y la de su derivada, son:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad tanh'(x) = \frac{1 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2}. \tag{7}$$

Cuyas gráficas se observan en la Figura 6. La función de activación Tangente hiperbólica $tanh(x) : \mathbb{R} \to [-1,1]$ es una función suave y diferenciable en todo punto. A diferencia de la función sigmoidea, esta función se aproxima a la función identidad cerca al cero, sin embargo, el gradiente de la función evaluado en valores muy alejados al origen será un valor muy pequeño, por lo que sigue generando un estancamiento en el proceso de retropropagación.

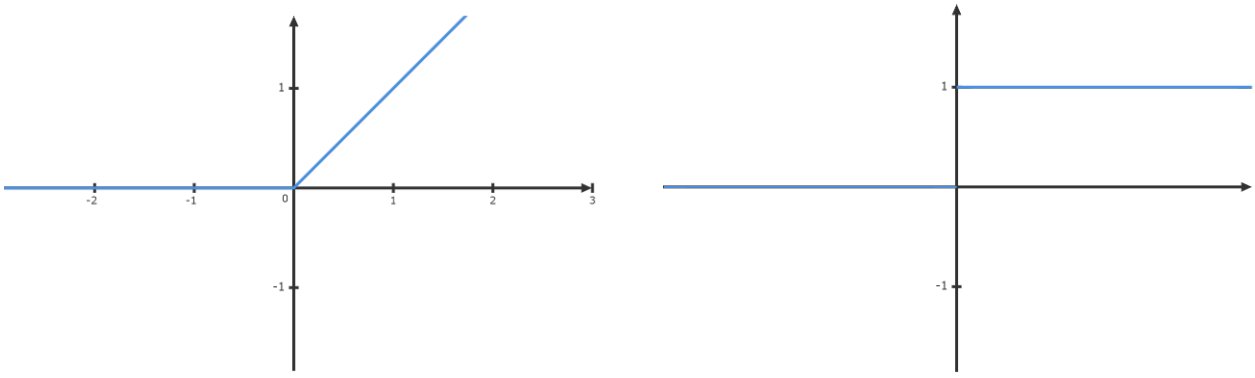
3. **Función Rectificadora** La función de activación rectificadora o unidad lineal rectificadora (ReLU), como se muestra en la Figura 7 es una función de activación computacionalmente muy eficiente, esta función y su derivada se definen de la siguiente manera:

$$F(x) = max(0,1), \quad F'(x) = \begin{cases} 0 & x < 0, \\ 1 & x \geq 0. \end{cases} \tag{8}$$



(a) Función (b) Derivada

Figura 6: Función de activación Tangente Hiperbólica y su derivada



(a) Función (b) Derivada

Figura 7: Función de activación Rectificadora (ReLu) y su derivada

4. **Función Softmax** La función Softmax es usada mayormente en redes neuronales orientadas a resolver problemas de clasificación, esta función nos da como resultado un porcentaje de que tan probable es que los datos ingresados pertenezca a cada una de las clases, es común utilizar esta función de activación en las últimas capas de la red neuronal. La expresión que la define es:

$$S = \frac{e^{a_i^l}}{\sum_{k=1}^{K}\left(e^{a_k^l}\right)}, \text{ para } i = 1,..,K$$

donde a es la salida de las capas ocultas y K es el número de clases en el modelo.

La funciones que se definieron anteriormente son usadas principalmente en redes neuronales con pocas capas ocultas, la función de activación rectificadora (ReLU) es computacionalmente muy eficiente, una de sus características es que es muy utilizada en las redes neuronales profundas, como por ejemplo las redes neuronales convolucionales.

3.3. Funciones de Coste

Las funciones de coste, funciones de pérdida ó funciones objetivo son funciones que nos permiten medir la diferencia entre el resultado obtenido y el valor deseado. Esta función es crucial en el proceso de descenso del gradiente, ya que mediante este proceso buscaremos que la salida de la función de coste sea lo más pequeña posible, así el valor obtenido por la red neuronal será muy cercano al valor deseado.

Para ser usada en la retropropagación, la función de coste debe cumplir con dos propiedades:

1. La función de costo C debe poder escribirse como un promedio:

$$C = \frac{1}{n}\sum_{x}\mathcal{L}_x,$$

sobre las funciones de perdida $\mathcal{L}_x$ para ejemplos de entrenamiento individuales, x.

2. La función de costo C no debe depender de ningún valor de activación además de los valores de salida a^L. Si la función de costo depende de otras capas de activación además de la de salida, la retro propagación no será válida porque la idea de propagar hacia atrás ya no funciona.

Nota: Es importante recalcar que la función de coste y la función de pérdida son diferentes, ya que la función de costo es el promedio de las perdidas de todas las muestras o ejemplos. Sin embargo, es

común ver el uso de la función de costo como sinónimo o con el mismo propósito de la función de pérdida.

1. **Error Cuadrático Medio** El costo cuadrático o también conocido como Error Cuadrático Medio (MSE) es una de las funciones de coste más comunes y simples, se define como

$$\mathcal{C} = \frac{1}{2n} \sum_{k=1}^{n} \left\| y_k(x) - \hat{y}_k(x) \right\|^2, \tag{9}$$

   donde n denota el número total de ejemplos de entrenamiento.

2. **Entropía cruzada** Es más eficiente usar esta función cuando trabajamos problemas de clasificación, donde la salida de nuestro modelo es una probabilidad entre 0 y 1. Definimos la función de costo de entropía cruzada como

$$\mathcal{C} = -\frac{1}{n} \sum_{k=1}^{n} \Big[y_k \; ln \; a_k^L + (1 - y_k) ln(1 - a_k^L) \Big], \tag{10}$$

   donde n es el número total de elementos de datos de entrenamiento.

3.4. Descenso del Gradiente

El gradiente o vector gradiente es una generalización de la derivada en varias variables, su definición formal se muestra a continuación.

Definición 7 (Vector Gradiente). [9, pág 193] Sea $f : U \subseteq \mathbb{R}^n \to \mathbb{R}$ una función diferenciable definida en el conjunto abierto U de $\mathbb{R}^n$. Se define el (vector) gradiente de la función f en el punto x_0 de U, denotado por $\nabla f(x_0)$ como el vector en $\mathbb{R}^n$ dado por

$$\nabla f(x_0) = \left(\frac{\partial f}{\partial x_1}(x_0), \frac{\partial f}{\partial x_2}(x_0), \cdots, \frac{\partial f}{\partial x_n}(x_0) \right).$$

Además el vector gradiente nos indica la dirección en la cual la función f crece más rápidamente, este resultado se formaliza de la siguiente manera.

Teorema B. *Sea $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ diferenciable en $x_0 \in X$; el gradiente apunta al mayor crecimiento de f.*

Demostración. Ver [14, pág 939] □

El método de descenso del gradiente es fundamental en el entrenamiento de las redes neuronales, por medio de este somos capaces de considerar los valores más óptimos y eficaces, en este caso los pesos **w** de nuestra red neuronal. Este método nos permite estimar cada nuevo parámetro a partir del anterior, teniendo en cuenta la derivada de la función de coste, además este proceso cuenta con ventajas como la sencillez del mismo y la rapidez de convergencia. [2][5]

3.4.1. Algoritmo descenso del gradiente

Consideremos una función de costo $\mathcal{C}$ tal que $\mathcal{C} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, el algoritmo de gradiente descendiente nos permite encontrar un valor w en Ω tal que $\mathcal{C}(w)$ es mínimo (extremo local).

Las actualizaciones de los w, se harán de la siguiente manera:

$$w_{k+1} = w_k - \alpha \nabla \mathcal{C}(w_k),$$

donde α será la tasa de aprendizaje y k es el número de iteraciones. Inicialmente se escoge un valor inicial w_0 (se puede permitir que el programa escoja de forma aleatoria este valor, o bien, puede ser seleccionado por nosotros mismos), en este punto se comenzará el algoritmo. El propósito del algoritmo es variar el valor del peso inicial hasta posicionarlo en el punto mínimo de la función.

Nota: Para ver un ejemplo más detallado de este algoritmo puede visitar el siguiente *blog de anexos*.

Algoritmo 2 Descenso del gradiente

Entrada: α, n
Salida: $w(k)$

1: Inicialización: $w(0),\ i = 0,\ k = 1$
2: **mientras** $i = 0$ **hacer**
3: $\quad w(k) \leftarrow w(k-1) - \alpha \nabla \mathcal{C}(w(k-1))$
4: $\quad$ **si** $||w(k) - w(k-1)|| \leq 1e^{-10}$ **entonces**
5: $\quad\quad i = 1$
6: $\quad$ **fin si**
7: $\quad k = k + 1$
8: **fin mientras**
9: **Retorno** $w(k+1)$

3.5. Redes neuronales MLP

Como ya vimos en la sección de arquitectura de las redes MLP, éstas pueden ser representadas mediante un diágrafo simple, con aristas ponderadas y un conjunto de atributos que las caracterizan, en esta sección formalizaremos la estructura de dicha red, con sus definiciones correspondientes:

Nota: la comprensión de las definiciones y algoritmos que se darán a continuación, se facilitará si al tiempo se visualiza la red que aparece en la **Figura (4)**.

Definición 8 (Redes neuronales artificiales (MLP)). [15, pág 2]

Se define una red neuronal artificial como una tripla $< \mathscr{D}, \{f\}, \mathscr{A} >$, donde:

- $\mathscr{D}$ es un dígrafo contable, localmente finito, con aristas etiquetadas, cuyos vértices corresponden a los nodos de procesamiento (neuronas) y las etiquetas de las aristas denominadas ***pesos***, corresponden a las intensidades de las conexiones sinápticas, denotadas por w_{ij} que indica el ***peso*** de la conexión entre la neurona j-ésima y la i-ésima.
- $\mathscr{A}$ es el conjunto que contiene los elementos de "entrada" de las unidades o nodos de procesamiento, generalmente $\mathscr{A} = \mathbb{R}$.
- $\{f : \mathscr{A} \to \mathscr{A},\}$ es una colección de funciones llamadas de activación.

En el dígrafo $\mathscr{D}$, se definen las capas como las columnas de vértices en $\mathscr{D}$, cada una de éstas columnas puede ser representada matemáticamente a través de un vector, así:

1. **Capa de entrada**: corresponde a la primera columna de vértices de $\mathscr{D}$, cuya representación matemática estará dada por:
$$\mathbf{x} = (x_1, ..., x_{d_0})^T \quad \text{con } \mathbf{x} \in \mathbb{R}^{(d_0 \times 1)}. \tag{11}$$
2. **Capa de salida**: corresponde a la última columna de vértices de $\mathscr{D}$, cuya representación matemática estará dada por:
$$\hat{\mathbf{y}} = (\hat{y}_1, ..., \hat{y}_{d_L})^T \quad \text{con } \hat{\mathbf{y}} \in \mathbb{R}^{(d_L \times 1)}. \tag{12}$$
3. **Capas ocultas** corresponde a las columnas intermedias entre la capa de entrada y la de salida, cuya representación matemática estará dada por:
$$\begin{aligned} \mathbf{a}^l &= (\boldsymbol{a}_1^l, ..., \boldsymbol{a}_{d_l}^l)^T \quad \text{con } \mathbf{a}^l \in \mathbb{R}^{(d_l \times 1)} \\ &= \boldsymbol{f}^l(\mathbf{z}^l) \qquad\quad \text{con } l = 1, ..., L-1. \end{aligned} \tag{13}$$
Note que cada $\mathbf{a}^l$ corresponde a una columna de vértices en $\mathscr{D}$; donde L denotará la totalidad de capas en la red, $\boldsymbol{f}^l$ será una función de activación vectorial y $\mathbf{z}^l$ será el combinador lineal matricial, ambos en la capa l. Con lo cual la cakpa de salida también puede representarse como el vector $\mathbf{a}^L$, y la capa de entrada como el vector $\mathbf{a}^0$

Definición 9 (Neuronas o nodos de procesamiento). Las neuronas de la red MLP, serán los vértices de las capas ocultas en $\mathscr{D}$, es decir las componentes de $\mathbf{a}^l$ las cuales serán denotadas por $\boldsymbol{a}_i^l$, donde:

$$\boldsymbol{a}_i^l = f^l(z_i^l).$$

Definición 10 (Funciones de activación). Se define $\boldsymbol{f}^l$ como una función de activación vectorial, tal que:

$$\boldsymbol{f}^l : \mathbb{R}^{(d_l \times 1)} \longrightarrow \mathbb{R}^{(d_l \times 1)}.$$

Definición 11 (Matriz de pesos en cada capa). Para cada capa l en $\mathscr{D}$, se define $\mathbf{w}^l$ como una matriz de dimensiones $d_l \times d_{l-1}$, donde d_l es la cantidad de neuronas en la capa l, así:

$$\mathbf{w}^l = \begin{bmatrix} w^l_{11} & \dots & w^l_{1j} & \dots & w^l_{1d_{l-1}} \\ \vdots & \dots & \vdots & \dots & \vdots \\ w^l_{i1} & \dots & w^l_{ij} & \dots & w^l_{id_{l-1}} \\ \vdots & \dots & \vdots & \dots & \vdots \\ w^l_{d_l1} & \dots & w^l_{d_lj} & \dots & w^l_{d_ld_{l-1}} \end{bmatrix}. \tag{14}$$

Definición 12 (Sesgo). Se define el sesgo, como el vector:

$$\boldsymbol{b}^l = (b^l_1, \dots, b^l_{d_l})^T \quad \text{con } \boldsymbol{b}^l \in \mathbb{R}^{(d_l \times 1)}, \tag{15}$$

correspondiente a la capa l, cuyas entradas serán el parámetro de sesgo de cada neurona.

Definición 13 (Combinador lineal matricial:). dados $\mathbf{a}^{l-1}$, $\mathbf{w}^l$ $\boldsymbol{b}^l$ se define el combinador lineal como:

$$\begin{aligned} \mathbf{z}^l &= (z^l_1, \dots, z^l_{d_l})^T \quad \text{con } \mathbf{z}^l \in \mathbb{R}^{(d_l \times 1)} \\ &= \mathbf{w}^l \mathbf{a}^{l-1} + \boldsymbol{b}^l, \end{aligned} \tag{16}$$

donde

$$z^l_i = \sum_{j=1}^{d_{l-1}} w^l_{ij} a^{l-1}_j + b_i. \tag{17}$$

Note que la ecuación (17) tiene una fuerte relación con la **Definición** 2 de combinador lineal en el perceptrón, sin embargo, para $\mathbf{z}^l$ hemos añadido el vector de parámetros de sesgo $\boldsymbol{b}^l$.

Definición 14 (Función de pérdida). Se define $\mathcal{L} : \mathbb{R}^n \longrightarrow \mathbb{R}$, tal que

$$\begin{aligned} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) &= \tfrac{1}{2} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 \\ &= \tfrac{1}{2} \|\mathbf{y} - \mathbf{a}^L\|^2 \\ &= \tfrac{1}{2} \sum_{r=1}^{d_L} (y_r - a^L_r)^2, \end{aligned}$$

como la función de perdida de la red MLP

Definición 15 (Conjunto de datos de entrenamiento). sea $\mathbf{X} = (\mathbf{x}(1),....,\mathbf{x}(n))$, donde $\mathbf{x}(k)$, con $k = 1,...,n$, representa el k-ésimo dato, en el conjunto $\mathbf{X}$, siendo éste el vector de entradas de la red neuronal en la $k-$ésima etapa.

Definición 16 (Conjunto de salidas de la red). Sea $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}(1),...,\hat{\mathbf{y}}(n))$ donde $\hat{\mathbf{y}}(k)$, con $k = 1,...,n$, representa el k-ésimo dato, en el conjunto $\hat{\mathbf{Y}}$, siendo éste el vector de salidas de la red neuronal en la $k-$ésima etapa.

Definición 17 (Resultados esperados). sea $\mathbf{Y} = (\mathbf{y}(1),...,\mathbf{y}(n))$ donde $\mathbf{y}(k)$, con $k = 1,...,n$, representa el k-ésimo dato, en el conjunto $\mathbf{Y}$, siendo éste el vector de resultados esperados correspondiente al dato $\mathbf{x}(k)$.

3.6. Entrenamiento y aprendizaje de la red neuronal MLP

El proceso de aprendizaje de una red neuronal, es un modelo de aprendizaje supervisado, y consiste en establecer un algoritmo que a partir de un conjunto de datos de entrenamiento (entradas y resultados esperados), permita entrenar la red por etapas para que esta logre calcular por sí misma los valores de pesos y sesgos más adecuados para clasificar los datos de entrada, en salidas, que tendrán una diferencia mínima respecto de los resultados esperados. Al finalizar el proceso, la red neuronal debe estar capacitada para clasificar cualquier dato, que no fue dado inicialmente en el conjunto de entrenamiento (datos de testeo), otorgando una salida con un error de clasificación mínimo. Dicho entrenamiento se compone de dos importantes procesos, estos son la propagación hacia adelante o *feedforward* y la retropropagación, o *back-propagation.*

3.6.1. Prealimentación o propagación hacía adelante

El proceso de prealimentación, es la base del entrenamiento y aprendizaje de la red, éste considera los siguientes pasos:

1. Elegir un vector de datos $\mathbf{x} \in \mathbf{X}$, como entrada de la red neuronal MPL.
2. Establecer matrices $\mathbf{w}^l$ de pesos y vectores $\boldsymbol{b}^l$ de sesgo, cuyas componentes tendrán entradas aleatorias que pertenecen a un umbral prefijado.
3. "Alimentar" la red neuronal en una única dirección, para esto, debemos empezar por establecer lo que tendremos en la primera capa de procesamiento, y luego generalizarlo:
   - Alimentación primera capa: debemos establecer los productos matriciales que se dan en cada nodo de procesamiento, estos son:

$$\mathbf{z}^1 = (\mathbf{w}^1\mathbf{x}) + \boldsymbol{b}^1 \quad ; \quad \mathbf{a}^1 = f^1(\mathbf{z}^1).$$

- Generalización: teniendo en cuenta cómo se "alimenta" la red en la primera capa, haremos el mismo proceso para cada capa siguiente:

$$\mathbf{z}^l = (\mathbf{w}^l \mathbf{a}^{l-1}) + \boldsymbol{b}^l \quad ; \quad \mathbf{a}^l = f^l(\mathbf{z}^l).$$

nótese que en este paso, lo que en la primera capa era $\mathbf{x}$ en cualquier capa diferente será $\mathbf{a}^{l-1}$ esto se debe a que la red es un dígrafo $\mathscr{D}$, cuya salida de la capa anterior $(l-1)$ se convierte en el vector de entrada para la capa siguiente (l).

3.6.2. Retropropagación

la retropropagación es el proceso empleado en las redes neuronales como algoritmo de aprendizaje y su objetivo es ajustar de manera "eficiente" los pesos de la red. Este proceso consiste en establecer de manera inicial y aleatoria los pesos requeridos en la red para obtener una salida, la cual será comparada por medio de la función de perdida $\mathcal{L}$, con el resultado esperado; de esta manera se podrá calcular el error de aproximación de la red. Lo que se busca es minimizar este error, a través de un proceso de optimización de la función $\mathcal{L}$, de la cual se obtendrán los valores adecuados para los parámetros de la red, dicha optimización se realiza a través de una generalización del algoritmo de descenso del gradiente, haciendo uso de la regla de la cadena, recorriendo la red de atrás hacia adelante. Al realizar este proceso de forma iterativa, la red "aprende" a establecer los pesos y sesgos adecuados para cada neurona con el fin de obtener una salida que se aproxime al resultado esperado.

Para comprender el funcionamiento del algoritmo de retropropagacion, debemos empezar calculando las derivadas con respecto a los parámetros de pesos y sesgos, de la función de coste, en nuestra red pre alimentada. Empecemos calculando la derivada de $\mathcal{L}$, con respecto a uno de los pesos que afectan a la última capa:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_{ij}^L} &= \frac{1}{2} \sum_{r=1}^{d_L} \frac{\partial}{\partial w_{ij}^L} (y_r - a_r^L)^2 \\
&= \sum_{r=1}^{d_L} (a_r^L - y_r) \left(\frac{\partial a_r^L}{\delta w_{ij}^L} \right) \\
&= \sum_{r=1}^{d_L} (a_r^L - y_r) \frac{\partial}{\partial w_{ij}^L} f^L(z_r^L) \\
&= \sum_{r=1}^{d_L} (a_r^L - y_r) \frac{\partial}{\partial w_{ij}^L} f^L \left(\sum_{t=1}^{d_L} w_{rt}^L a_t^{L-1} + b_r^L \right),
\end{aligned} \tag{18}$$

note que la expresión en (18) se anula en todos los valores en los que $r \neq i$ o $t \neq j$, entonces si $r = i$ y $t = j$, se tiene:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^L} = (a_i^L - y_i) f^{(1)L}(z_i^L) a_j^{L-1}. \tag{19}$$

Así, en la ecuación anterior, obtenemos la derivada particular de la función de perdida con respecto a un único peso. Para generalizar esta situación y calcular $\frac{\partial \mathcal{L}}{\partial \mathbf{w}^L}$, debemos cuidar las dimensiones y definir una nueva operación matricial.

Definición 18 (Producto Hadamard). Dadas dos matrices A , B ambas de dimensión $(m \times n)$ el producto de Hadamard: $(A \odot B)$ es una matriz de dimensión $(m \times n)$ tal que:

$$(A \odot B)_{ij} = (A)_{ij}(B)_{ij}.$$

Generalizando la ecuación (19), y haciendo uso de la definición anterior, se tiene:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}^L} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^L} \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{z}^L}{\partial \mathbf{w}^L} \\ &= [(\mathbf{a}^L - y) \odot f^{(1)L}(\mathbf{z}^L)] \, (\mathbf{a}^{L-1})^T, \end{aligned} \tag{20}$$

note que el error en la última capa será: $(\mathbf{a}^L - y)$ y lo denotaremos como $\mathbf{e}^L$, también denotaremos todo el producto de Hadamard $[(\mathbf{a}^L - y) \odot f^{(1)L}(\mathbf{z}^L)]$ como δ^L, así:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^L} = \delta^L (a^{L-1})^T. \tag{21}$$

Si realizamos de forma análoga el proceso desde proceso desde (18), hasta (21), para hallar $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^L}$, obtenemos:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^L} = \delta. \tag{22}$$

Sabemos que $\mathcal{L}$ en $\mathscr{D}$, depende de las matrices de pesos y vectores de sesgo de cada capa, por lo tanto, debemos minimizar la función $\mathcal{L}$ en cada capa l. Si consideramos las derivadas en la capa $L-1$, obtenemos:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}^{L-1}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}^L} \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{z}^L}{\partial \mathbf{a}^{L-1}} \frac{\partial \mathbf{a}^{L-1}}{\partial \mathbf{z}^{L-1}} \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{w}^{L-1}} \\ &= [((\mathbf{w}^L)^T \delta^L) \odot f^{L-1(1)}(\mathbf{z}^{L-1})](\mathbf{a}^{L-2})^T \\ &= \delta^{L-1}(\mathbf{a}^{L-2})^T, \end{aligned} \tag{23}$$

donde, $((\mathbf{w}^L)^T \delta^L) = \mathbf{e}^{L-1}$ y $[((\mathbf{w}^L)^T \delta^L) \odot f^{L-1}(\mathbf{z}^{L-1})] = \delta^{L-1}$.

Análogamente:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{L-1}} = \delta^{L-1}. \tag{24}$$

Generalizando se tiene

$$\delta^l = [((\mathbf{w}^{l+1})^T \delta^{l+1}) \odot f^l(\mathbf{z}^l)], \quad \text{para } l = L-1, \ldots, 1.$$

De donde

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^l} = \delta^l (a^{l-1})^T, \qquad \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \delta^l. \tag{25}$$

Ahora podemos definir el proceso iterativo de backpropagation en los siguientes pasos:

1. Calcular el error $\mathbf{e}^L$ y δ^L en la última capa, como en (20) y (21).
2. Calcular $\mathbf{e}^l$ y δ^l en cada capa l:
$$\begin{cases} \mathbf{e}^l = (\mathbf{w}^{l+1})^T \delta^{l+1}, \\ \delta^l = (f^{l(1)}(\mathbf{z}^l)) \odot \mathbf{e}^l. \end{cases}$$
3. Por último, se procede a realizar la actualización de los pesos y sesgos, para ello antes vamos a considerar la función de coste $\mathcal{C}$ como en (9) aplicada a la perdida $\mathcal{L}$, así:
$$\mathcal{C} = \frac{1}{n} \sum_{k=1}^{n} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})(k).$$
Calculemos su derivadas con respecto a $\mathbf{w}^l$ y $\boldsymbol{b}^l$ empleando las ecuaciones (21), (22) y (25), así:
$$\begin{aligned} \frac{\partial \mathcal{C}}{\partial \mathbf{w}^l} &= \frac{1}{n} \sum_{k=1}^{n} \frac{\partial}{\partial \mathbf{w}^l} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})(k) \\ &= \nabla_{\mathbf{w}^l} \mathcal{C}. \end{aligned} \tag{26}$$
Análogamente para el sesgo;
$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{b}^l} = \frac{1}{n} \sum_{k=1}^{n} \frac{\partial}{\partial \boldsymbol{b}^l} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})(k) = \nabla_{\mathbf{b}^l} \mathcal{C}, \tag{27}$$
note que en (26) y (27), las derivadas con respecto a $\mathbf{w}^L$ y $\boldsymbol{b}^L$, son las mismas que deducimos en (25), con estos últimos parámetros obtenidos en las ecuaciones anteriores, podemos aplicar el

algoritmo de descenso del gradiente, tal como lo vimos en (2), para actualizar nuestras matrices de pesos y sesgos, así:

$$\begin{cases} \mathbf{w}^l(t) := \mathbf{w}^{l-1}(t-1) - \eta \nabla_{\mathbf{w}^l} \mathcal{C}, \\ \mathbf{b}^l(t) := \mathbf{b}^{l-1}(t-1) - \eta \nabla_{\mathbf{b}^l} \mathcal{C}, \end{cases} \tag{28}$$

donde t es el iterador de las épocas que se le asignan a la red.

Aplicando estos algoritmos (Feed-Forward y Backpropagation), podemos construir una red neuronal que a partir de una muestra del conjunto de datos (datos de testeo o entrenamiento), se entrene para conseguir una clasificación acertada de los datos del conjunto. Cabe aclarar que antes de aplicar el conjunto de testeo a la red, se debe realizar un proceso de tratamiento de datos, en el cual se eliminan datos atípicos y se normaliza el conjunto de entrenamiento, para así evitar confusión en la clasificación. Podemos ver un ejemplo de la aplicación de los algoritmos paso a paso en el blog de anexos.

A partir de estos algoritmos, hemos construido una red neuronal clasificadora usando Python, la cual ha sido entrenada a partir de un conjunto de datos que contiene naranjas y manzanas con características los tamaños y los pesos de estas, para así clasificarlas. La aplicación de este ejemplo particular se encuentra en el blog de anexos.

Las redes neuronales artificiales tienen un amplio campo de aplicación en el análisis de información y la clasificación de datos, sin embargo el gran potencial de estas radica en su capacidad de aproximar funciones.

4. Redes Neuronales como Aproximador Universal de Funciones

A lo largo del tiempo se han estudiado formas de aproximar funciones, esto ya que la aproximación tiene muchas utilidades, por ejemplo, describir el movimiento de los objetos como función del tiempo. Uno de los métodos más usados para aproximar funciones es a partir del polinomio de Taylor, este es de gran valor en el estudio de los métodos numéricos, ya que proporciona un medio para predecir el valor de una función en un punto en términos del valor de la función y sus derivadas en otro punto. En particular, el teorema establece que cualquier función suave puede aproximarse por un polinomio [3]. Para nuestra sorpresa, las redes neuronales tienen también el poder de hacer aproximaciones de funciones tan buenas como se deseen.

El teorema de aproximación universal, es el puente entre las redes neuronales y la aproximación de funciones; éste establece que las redes neuronales son capaces de aproximarse, tanto como desee, al valor real de una función, siempre y cuando dicha función cumpla las condiciones del teorema. Inicialmente daremos una explicación simple de como una red neuronal de tan solo una capa oculta, también conocida como *Shallow*, es capaz de hacer una aproximación tan buena como se desee de

cualquier función continua, esta breve introducción al teorema formal se hará para funciones de una sola variable, claramente el teorema nos permite trabajar con funciones de múltiples variables, sin embargo concentraremos nuestros esfuerzos en la explicación para el caso univariado.

Consideremos una red neuronal con una única neurona de entrada, una capa oculta de dos neuronas y una única neurona de salida, como se observa en la Figura 8.

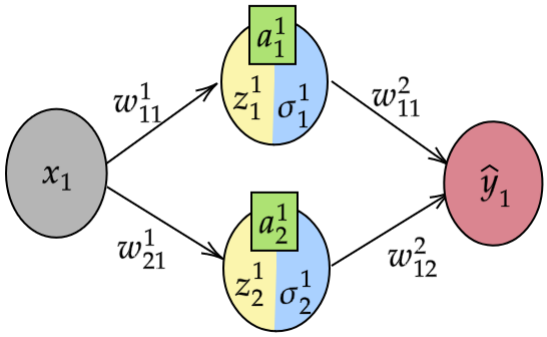


Figura 8: Red Shallow 2 neuronas

La función de activación asociada a las dos neuronas de la capa oculta será la función sigmoidal definida en la ecuación (6), es importante analizar de esta función que si tomamos valores para w lo suficientemente grandes y valores para b lo suficientemente pequeños, logramos transformar la función sigmoidal en una función escalonada como la que se muestra en la Figura 9.

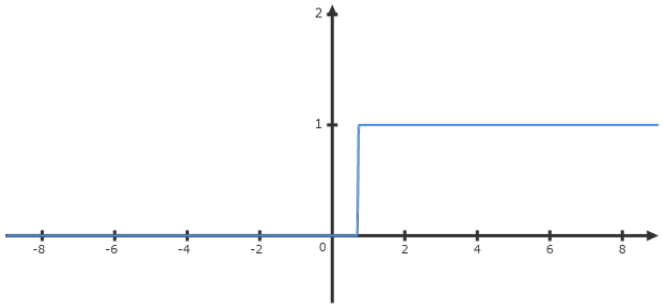


Figura 9: Función sigmoidal con $w = 142$ y $b = -200$

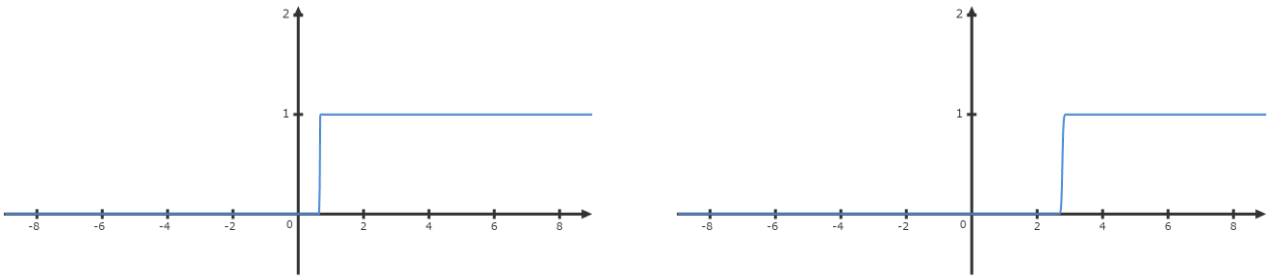
Manipular las funciones de esta manera es útil ya que algebraicamente es más sencillo trabajar con

funciones escalonadas que con funciones sigmoideas o con alguna de las otras funciones de activación expuestas anteriormente.

Ya que contamos con dos neuronas en la capa oculta, obtenemos dos funciones sigmoidales las cuales denotamos σ_1^1 y σ_2^1 para la primera y segunda neurona respectivamente. Por la definición de la función sigmoidea tenemos que:

$$\sigma_1^1(x_1) = \frac{1}{1+e^{-w_{11}^1 x_1 - b_1^1}}, \quad \sigma_2^1(x_1) = \frac{1}{1+e^{-w_{21}^1 x_1 - b_2^1}}. \tag{29}$$

Si manipulamos los pesos w_{11}^1, w_{21}^1 y los sesgos b_1^1, b_2^1 de la misma forma que se manipularon anteriormente, es decir, w_{ij}^1 lo suficientemente grande y b lo suficientemente pequeño, obtendremos dos funciones escalonadas como se observa en la Figura 10.



(a) Función $\sigma_1^1(x_1)$ (b) Función $\sigma_2^1(x_2)$

Figura 10

La combinación lineal que corresponde a la salida $\hat{y}$ está dada por $\hat{y} = w_{11}^2\sigma_1^1(x_1) + w_{12}^2\sigma_2^1(x_1)$. Es decir, que la salida $\hat{y}$ será una protuberancia como la mostrada en la Figura 11.

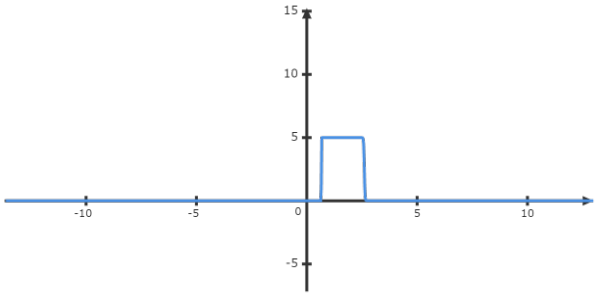


Figura 11: Salida $\hat{y}$

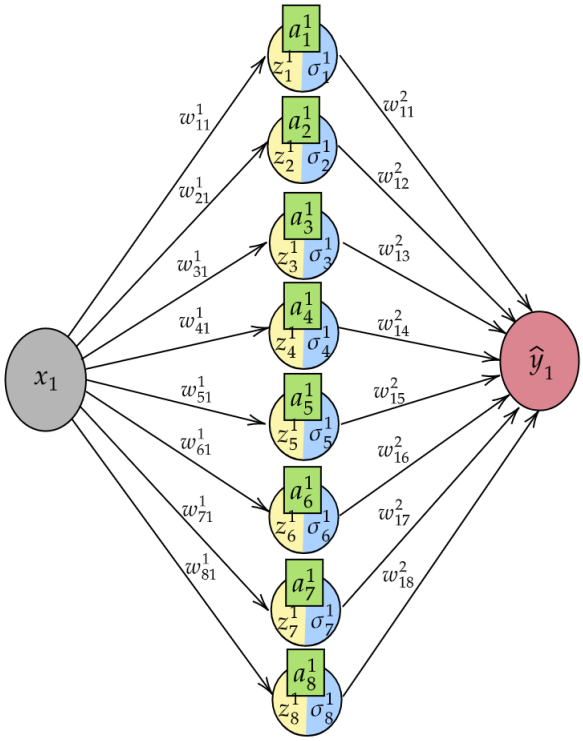


Figura 12: Red aproximadora Shallow 8 neuronas

Para crear más protuberancias o torres simplemente debemos agregar más neuronas en la capa oculta, tal como se muestra en la Figura 12.

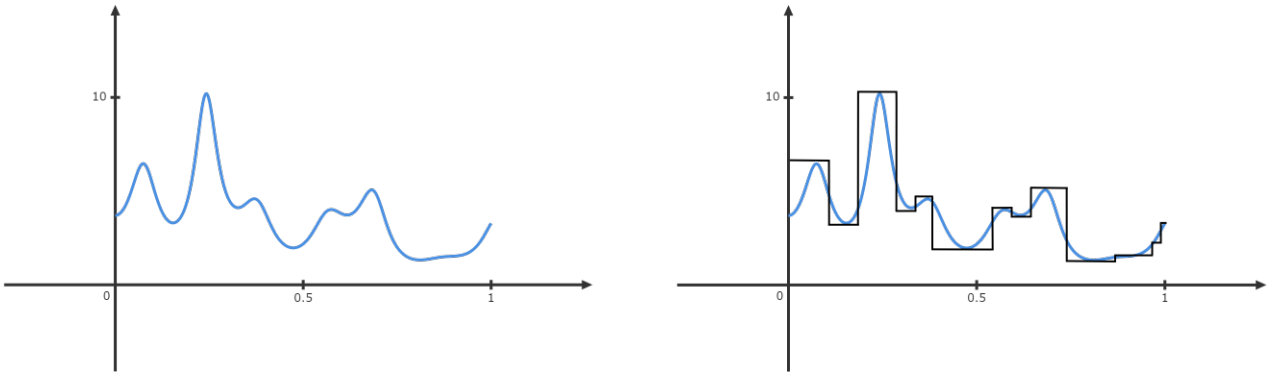
De manera que si se considera la función $f(x) = (0{,}2 + 0{,}44x^2 + 0{,}3x\sin(17x) + 0{,}07\cos(40x))^{-1}$ se podría llegar a una aproximación como la mostrada en la Figura 13.

Para ver más ejemplos del comportamiento de la función sigmoide como protuberancias o torres puede acceder al apartado de *Aproximación de funciones* en el blog de anexos.

4.1. Teorema de Aproximación Universal

Teorema C. *[4, Pág 6] Sea φ cualquier función sigmoidal continua. Entonces las sumas finitas de la forma*

$$G(x) = \sum_{j=1}^{N} \alpha_i \varphi\left(w_j^T x + b_j\right), \tag{30}$$



(a) Función $f(x)$ (b) Función aproximada

Figura 13: Aproximación de funciones mediante redes neuronales tipo Shallow

son densas en $C(I_n)$. *En otras palabras, dada cualquier* $f \in C(I_n)$ *y* $\epsilon > 0$, *hay una suma,* $G(x)$, *de la forma (30), para la cual*

$$|G(x) - f(x)| < \epsilon \quad \textit{para todo } x \in I_n.$$

Los detalles de la demostración, de este teorema, se encuentran en: [4, pág 5]

Ejemplo aproximación de funciones

Para ejemplificar la aplicación del teorema anterior y la teoría de aproximación de funciones a partir de redes neuronales, hemos implementado a nuestro código de red neuronal MLP, un aprendizaje que al finalizar no separa en clases, lo que permite la predicción como aproximador.

Hemos considerado la función $f(x) = \sin(x)$, y nuestro conjunto de datos ha sido construido a partir del intervalo $(-4, 4)$ considerando 80 particiones regulares, cada uno de estos valores corresponderá a un dato de nuestro conjunto de entrenamiento, el cuál será un vector de 80 componentes; el conjunto de resultados esperados será el vector resultante de aplicar la función $\sin(x)$ al vector de entrenamiento (es decir, componente a componente), de esta manera construimos una red aproximadora con las siguientes especificaciones:

Resumen de la red:

Las características de la red que se usó para el ejemplo son las siguientes:

- La red cuenta con 3 capas, y una de ellas es una capa oculta.
- La primera capa cuenta con 800 neuronas y la función de activación usada fue la función sigmoide.

- La segunda capa tiene una neurona y la función de activación utilizada fue la función de identidad.
- El tamaño del paso α fue de $1e^{-2}$ y se usaron 500 épocas.

Obteniendo la siguiente aproximación con un error relativo del 9 %:

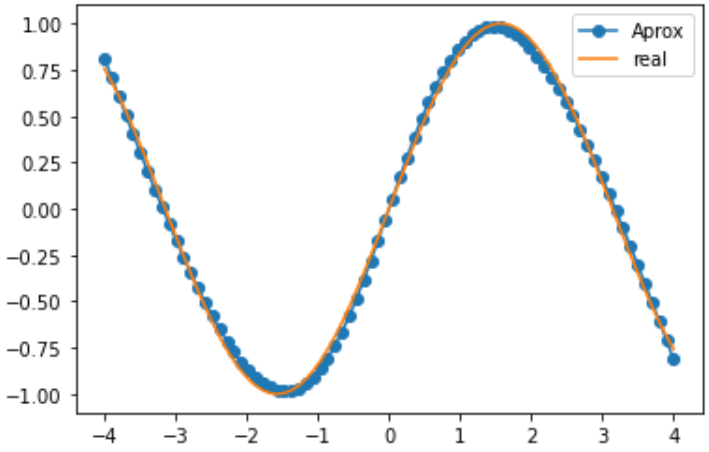


Figura 14: Aproximación de la función sen(x) a partir de una red Shallow

4.2. Redes Neuronales como Método para Aproximar Soluciones de Ecuaciones Diferenciales

Las ecuaciones diferenciales nos han permitido a lo largo del tiempo describir fenómenos o situaciones de la vida real, si un fenómeno se puede expresar mediante una o varias razones de cambio entre las variables implicadas entonces correspondientemente tenemos una o varias ecuaciones diferenciales.[16]

Ahora, dado que solucionar una ecuación diferencial es encontrar una función que satisfaga determinadas condiciones tenemos que, por el poder de las redes neuronales como aproximador universal de funciones continuas, somos capaces de aproximar una solución a una determinada ecuación diferencial.

Para ejemplificar dicha aplicación, consideramos la siguiente ecuación diferencial;

$$\Delta u(x,y) = e^{-x}(x-2+y^3-6y), \quad \text{donde, } \Delta u(x,y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \tag{31}$$

en el cuadrado unitario $I^2 = [0,1]^2$ cuyas condiciones de frontera son:

$$u(0,y) = y^3,\ u(1,y) = (1+y^3)e^{-1},\ u(x,0) = xe^{-x},\ u(x,1) = e^{-x}(x+1).$$

Consideraremos una red neuronal con las siguientes especificaciones:

- La red tendrá 3 capas.
- La capa oculta contará con 5 neuronas y usará como función de activación a la función sigmoide.
- La capa de salida tendrá 1 neurona y la función de activación usada será la función identidad.

Esta red será entrenada de forma habitual, pero cuyos datos de entrenamiento serán considerados a partir de una malla de puntos interiores de I^2 de 30×30 este apartado acerca de la alimentación de la red, requiere de un proceso adicional el cual se puede observan en [7], y el caso particular de la ecuación (31) se puede ver a detalle en el blog de anexos, cuyo resultado obtenido se puede visualizar en la siguiente gráfica:

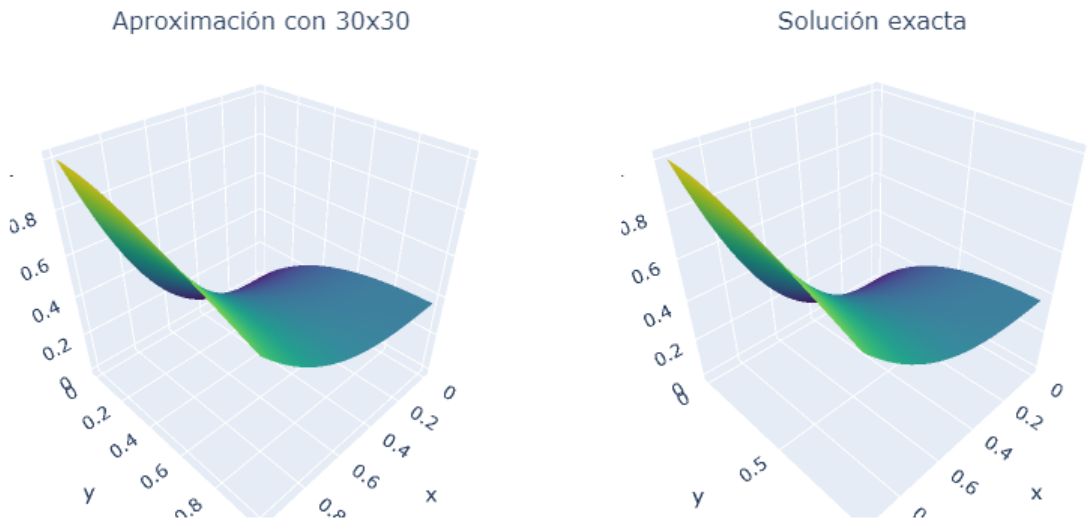


Figura 15: Aproximación a la solución de la E.D.P, a partir de una red neuronal de 2 capas

5. Conclusiones

En el desarrollo de este trabajo, hemos evidenciado cómo las matemáticas son el sustento teórico detrás de los modelos de aprendizaje supervisado, en particular las redes neuronales, como clasificadoras de datos.

Destacamos la necesidad de definiciones formales dentro de la teoría de redes y por ende nos permitimos construirlas con el objetivo de brindar una mayor compresión teórica de cada proceso con el fin de desentrañar el comportamiento de las redes neuronales y develar la idea general de “caja negra” por la que es tratada.

Asimismo, se programó una red neuronal en el lenguaje de programación Python, la red implementada respeta la estructura exhibida en el trabajo, este código nos permitió percibir y manipular de mejor forma todos los datos involucrados en una red neuronal, así como mostrar de forma gráfica los resultados de la red neuronal.

Adicionalmente, evidenciamos cómo esa teoría dió cabida a la teoría de aproximación de funciones aplicando redes neuronales, la cual es realmente relevante con relación a nuestra formación profesional, ya que no sólo se conciben la redes neuronales como clasificadoras de datos, sino que nos permitimos comparar estas mismas redes con métodos numéricos capaces de aproximar funciones continuas y por ende también soluciones de ecuaciones diferenciales ordinarias y parciales.

De este trabajo nos surge el interrogante de si la capacidad de las redes neuronales no se queda solamente bajo las hipótesis de los teoremas mencionados, sino que se expande para dar soluciones a problemas que impliquen condiciones más restrictivas. En este último apartado, cabe mencionar que los sustentos teóricos tras la solución de E.D.O y E.D.P se fundamentan en conceptos propios del análisis funcional y la teoría de la medida la cual se escapa de los propósitos de estudio del trabajo de grado.

Referencias

[1] ASU School of Life Sciences. *https://askabiologist.asu.edu/neuron-anatomy*. 2021.

[2] D. Ballesteros, A. Gaona y L. Pedraza. «Comparación por simplicidad de métodos de aprendizaje en estimación de funciones». Spanish. En: (2010). doi: 10.14483/22484728.279.

[3] S. Chapra y R Canele. *Métodos numéricos para ingenieros*. Mc Graw hill, 2015. isbn: 978-607-15-0499-9.

[4] G. Cybenko. «Approximation by Superpositions of a Sigmoidal Function». English. En: (1989).

[5] G. Hernandez. «Métodos Clásicos de Optimización para Problemas No-Lineales sin Restricciones». Spanish. En: (2006).

[6] F. Higham y J Higham. «Deep Learning: An Introduction for Applied Mathematicians». English. En: 61.4 (2019). doi: 10.1137/18M1165748.

[7] I. Lagaris y A Likas. «Artificial Neural Networks for Solving Ordinary and Partial Differential Equations». English. En: 9.5 (1998).

[8] M. Nielsen. *Neural Networks and deep Learning*. 2015. url: http://neuralnetworksanddeeplearning.com/.

[9] C. Pita. *Cálculo vectorial*. Pretince Hall, 1995. isbn: 9688805297.

[10] Giovanni Prodi. «Soluzioni periodiche di equazioni a derivate parziali di tipo iperbolico non lineari». Italian. En: *Annali di Matematica Pura ed Applicata* 42.1 (1956), págs. 25-49. issn: 0373-3114. doi: 10.1007/BF02411872.

[11] S. Russel y P. Norvig. *Artificial Intelligence A Modern Approach*. Pearson, 2010. isbn: 9780136042594.

[12] H. Simon. *Neural Networks and Learning Machines*. Pearson, 2009. isbn: 9780131471399.

[13] M. Spivak. *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. Advanced book program. W. A. Benjamin, 1965. isbn: 9780805390216.

[14] J. Stewart. *Cálculo de varias variables. Trascendentes tempranas*. Cengage learning, 2012. isbn: 9786074818987.

[15] L. Torres. «Redes neuronales y aproximación de funciones». Spanish. En: Vol. 2 (1995).

[16] J. Ventura y D Elizarraraz. *Ecuaciones diferenciales*. Azcapotzalco, 2004. isbn: 970-31-0230-1.