COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN MODELACIÓN MATEMÁTICA

TESIS: APLICACIÓN DE LAS REDES NEURONALES AL RECONOCIMIENTO DE PATRONES

PARA OBTENER EL TÍTULO DE

LICENCIADO EN MODELACIÓN MATEMÁTICA

PRESENTA:

PEDRO CARRERA ARAGÓN

DIRECTOR DE TESIS:

M. en C. MIGUEL ÁNGEL MENDOZA REYES

Ciudad de México, enero de 2024

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Dedicatoria

Con todo mi cariño y el amor que les tengo:
A mis padres Pedro y Bernarda.
A mis hermanos.

Agradecimiento

A mi director de tesis, el M. en C. Miguel Ángel Mendoza Reyes, por haber aceptado dirigir la tesis, por su paciencia y el aporte que me ayudo a contribuir este trabajo.
A la Universidad Autónoma de la Ciudad de México por darme la oportunidad de continuar con mis estudios y formar parte a esta casa de estudios.
A mis profesores por sus enseñanzas.
A mis compañeros y amistades de la universidad.

Introducción

Entender cómo funciona el aprendizaje dentro y fuera de nuestro entorno ha sido una característica que el ser humano ha ido fomentando desde que empezó a tener el uso de la razón, conocer cómo es que nuestra forma de vida progresa con el transcurso del tiempo, adaptándose así a diferentes contextos y situaciones que se enfrenta para su supervivencia. El ser humano ha sido la única especie que posee la oportunidad de evolucionar y aprender nuevos conocimientos, aprender de sus errores y a partir de eso dar una solución. Sabemos exactamente que el ser humano nunca ha estado conforme cuando alcanza un objetivo, siempre trata de superar sus propios límites, cada descubrimiento, cada interés que se le atraviesa para poder comprender cómo funciona el objeto que se está estudiando, siempre tratando de obtener la mejor solución para adaptarse a un nuevo contexto que él mismo está construyendo. Así es, nos estamos refiriendo al concepto de "conocimiento", una definición abstracta que es difícil de explicar, así, si tratáramos de dar una definición en particular podríamos decir que el ser humano obtiene conocimiento a través de la experiencia y de los errores que comete, como aprendizaje para mejorar el resultado final.

Todo el conocimiento que tenemos hoy en día está al alcance de nuestras manos gracias a las grandes contribuciones de personajes que han dedicado su vida entera al conocimiento, desde filósofos hasta científicos, entonces, ¿dónde se encuentra este

mecanismo que nos ayuda a entender el entorno del universo que vivimos?, ¿es posible estudiar este objeto para comprender su estructura y cómo funciona?

A partir del conocimiento, nos lleva a deducir que la herramienta más importante que posee el ser humano y que pueda realizar todo lo que se ha mencionado anteriormente es el cerebro, una máquina compleja que puede llegar a realizar múltiples tareas, además, ha sido uno de lo más grandes retos al ser estudiada en el ámbito científico y filosófico.

Definitivamente, tratar de realizar, imaginar y plasmar un concepto abstracto no es algo sencillo, pero teniendo las bases necesarias es posible que se pueda simular, es por eso que una de las herramientas que ha tenido popularidad en los últimos años ha sido la *Inteligencia Artificial (IA)*.

En el capítulo 1 se estudiará el mecanismo que tiene una neurona biológica del cerebro, conocer su estructura y sus componentes básicos para entender el proceso cuando una neurona recibe, procesa y envía información a otras neuronas. En el capítulo 2 se enfocará el conocimiento previo y necesario de las matemáticas para comprender toda la relación matemática que se implementa en el campo de las redes neuronales. En el capítulo 3 se expone una amplia información del campo de la ciencia de la computación. La *IA* ha tenido muchas ramas de investigación, pero será una en particular que se enfocará en este proyecto, conocido como las redes neuronales artificiales. El capítulo 4 se complementa con el capítulo 3, explicando la construcción del modelo mátematico de las redes neuronales, tomando las bases necesarias para crear la ecuación desde una neurona artificial de una capa hasta la neurona artificial para l capas, y en el capítulo 5 se presenta un proyecto realizado para una aplicación de reconocimiento de patrones.

El objetivo es diseñar una red neuronal artificial para entrenar un conjunto de dígitos escrito a mano para poner a prueba e imprimir resultados para la predicción del número, diseñada en un ambiente de programacion en MATLAB.

Índice general

Índice de figuras

Capítulo 1

La complejidad del cerebro

El cerebro humano ha sido considerado una de las máquinas más complejas al ser estudiada en el campo de la ciencia, una herramienta que posee la habilidad de realizar múltiples tareas y que tenga un sentido para la vida del ser humano es realmente increíble. Podemos distinguir colores, reconocer el rostro de alguna persona, almacenar información, realizar operaciones complejas hasta llegar incluso ser inconscientes de lo que hacemos y distintas tareas que podríamos seguir describiendo, todo esto es gracias a una pequeña masa que realiza tal funcionamiento.

Desde los últimos siglos el estudio del pensamiento y la epistemología del cerebro humano ha pasado por numerosas aportaciones y teorías que han tratado de explicar el cerebro. Pero si bien lo que llevó a una revolución a este estudio, fue una investigación realizada por el neurobiólogo Santiago Ramón y Cajal [1], que da lugar al inicio del entendimiento del sistema nervioso, proponiendo varias postulaciones sobresalientes. Dicha publicación despertaría el interés y la tendencia de explicar la complejidad del cerebro.

1.1. La influencia con la ciencia

Fue a finales del siglo XIX cuando el científico investigador español *Santiago Ramón y Cajal* llevara a cabo una investigación a la estructura del sistema nervioso, demostrando que la unidad funcional del cerebro es la neurona. Su teoría postulaba que las células nerviosas (neuronas) tenían presencia de algunas prolongaciones del cuerpo celular, estas prolongaciones eran pequeñas ramificaciones que hacían conectar a otras neuronas para transmitir información, planteó la existencia de las dendritas y el axón, las dendritas son pequeñas ramificaciones que conecta al núcleo, y el axón es una prolongación que envía la información a otras neuronas. Esta investigación lo convertiría a ser en uno de los pioneros en el campo de la neurociencia y ganador de un premio Nobel compartiéndolo con el investigador Camilo Golgi.[1]

La neurociencia estudia el funcionamiento del sistema nervioso, un campo de la ciencia que desafía el misterio que tiene el cerebro, tratando de explicar conceptos abstractos como lo es el pensamiento, la consciencia, los sentimientos, conceptos que hasta hoy en día siguen siendo un gran desafío.

Entonces, hasta este punto actual, ¿Qué es lo que conocemos del cerebro? Conocemos la estructura física del cerebro, que está constituido por miles de millones de neuronas, a la vez cada una de ellas puede estar conectada entre 10 mil a 20 mil neuronas, interactuando, enviando, y compartiendo información, mediante un proceso conocido como impulso nervioso, por lo que es importante mencionar este proceso.[15, 4, 3, 6, 8]

1.2. Red neuronal

El cerebro está compuesto por 10^{11} neuronas, una cantidad bastante grande y difícil de imaginar, en la que existen redes formadas por miles de neuronas que están interconectadas unas con otras, enviando y compartiendo información, esta **función**

es lo que lo hace diferente a las otras partes u órganos del cuerpo, concluyendo que es el elemento fundamental para el sistema de comunicación del sistema nervioso [9], entonces, ¿Qué es lo que lo hace especial a esta estructura al realizar tal funcionamiento?. Es interesante responder esta pregunta, ya que esto nos lleva a estudiar en algo particular y que ésta información sentaría la base a una de las áreas del campo de la ciencia de la computación.

1.2.1. La estructura y función de una neurona biológica

La estructura de una neurona tiene varios componentes y características biológicas, pero en este trabajo solo nos enfocaremos en las componentes principales, ya que es importante conocer cómo funciona la dinámica de este sistema y que además este conocimiento nos ayudará a fundamentar en los próximos capítulos.

La neurona biológica se conforma por tres componentes básicos que son; el **soma** o el cuerpo de la célula nerviosa, las **dendritas** y el **axón**. El soma contiene el núcleo de la célula donde existe todo el material genético, en ella ocurre el proceso bioquímico, las dendritas son pequeñas ramificaciones que se encargan de recibir la información y enviársela al núcleo, por último el axón es la prolongación conectada al núcleo donde conduce el impulso nervioso hacia otra neurona, esta explicación podemos visualizar en la figura 1.1.

1.2.2. La sinapsis y el impulso nervioso

La función principal de la neurona es poder enviar, procesar y transmitir información a otras neuronas, la **sinapsis** es el proceso cuando una neurona trata de comunicarse con otra a través del axón y las dendritas, pero estas no se llegan a tocar físicamente, dejando un pequeño espacio, llamado espacio intersináptico, donde la información es transmitida en pulsos químicos-

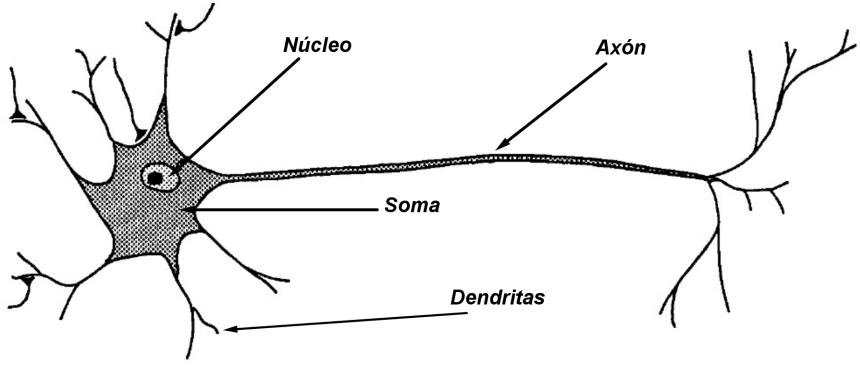


Figura 1.1: Estructura de una neurona biológica

eléctricos, proceso en donde se liberan sustancias químicas llamados neurotransmisores que se difunden por medio del espacio intersináptico, en este "contacto" de neuronas ocurre lo que se le conoce los impulsos nerviosos, es decir, el **impulso nervioso** está determinado a responder cuánta es la actividad de sinapsis que recibe, poniéndolo en dos tipos de estados, que el impulso genere un estado de **excitación** o **inhibición**. Hablamos que un proceso de sinapsis son de excitación si reciben una gran cantidad de sinapsis para que este dispare a un impulso generando un potencial de acción, y en el caso si los impulsos no son lo suficientemente resistente, entonces la neurona no emitirá un impulso y este se mantendrá en estado inhibitorio.

1.2.3. Potencial de acción

La característica fundamental que tiene un impulso nervioso es generar una salida de estimulación eléctrica que recorre todo a su paso a la membrana celular, esto se debe a que ocurre principalmente a una distribución diferencial de iones que tiene la célula nerviosa, constituidas mayoritariamente entre el cloro y el sodio. El proceso para poder visualizar este mecanismo funciona cuando una célula recibe una estimulación, inicia con un flujo

de aproximadamente de -70 mV (miliVolts), proceso que se le conoce como **potencial de reposo**, instantáneamente crece de forma ascendente y se dispara a un cierto umbral, luego decrece momentáneamente hasta llegar en la forma inicial, este último proceso se le conoce como periodo refractario, un ejemplo práctico podemos visualizar en la siguiente figura.

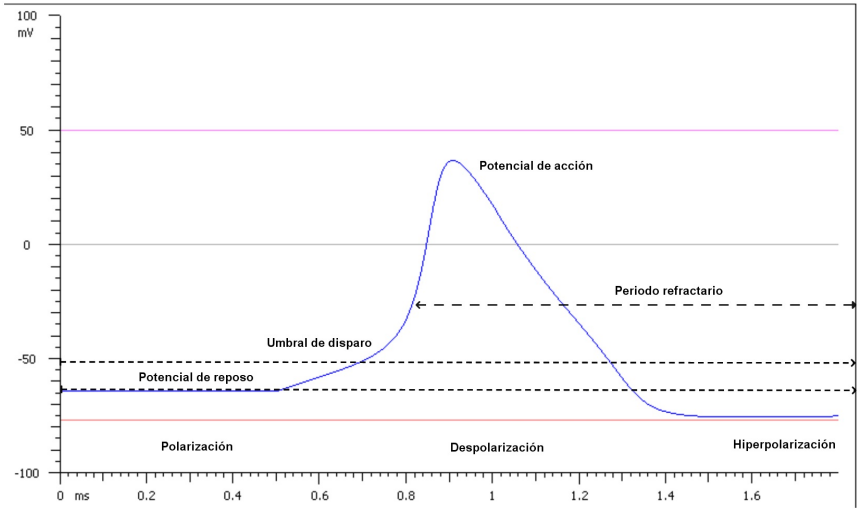


Figura 1.2: Gráfica de un potencial de acción

1.3. Modelos de neuronas

1.3.1. Modelo de Hodgkin y Huxley(HH)

Realizar una simulación al comportamiento del sistema dinámico de una neurona genera nuevos retos a descubrir y una de ellas es modelar un sistema de ecuaciones diferenciales que realice tal comportamiento.

Una de las primeras investigaciones a la modelación matemática en el estudio de las células nerviosas fueron hechos por Alan

Lloyd Hodgkin y Andrew Huxley en 1952, dos biofísicos que estudiaron experimentalmente la dinámica de voltaje transmembranal en el axón gigante de una neurona de calamar. Proponiendo un modelo de un sistema de cuatro ecuaciones diferenciales no lineal, el estudio demostró que el comportamiento de potencial de la membrana se debe a la función de las corrientes iónicas como el **sodio** y el **potasio**, además de que este tipo de ecuaciones tienen que ser estudiados recurriendo al análisis de método cuantitativo y simulaciones computacionales, asentando un sólido fundamento biofísico.

1.3.2. Modelo de FitzHugh y Nagumo

Desarrollar un modelo que simplifique el sistema de HH nos lleva a imaginar qué elementos o características se integrarían, y este hecho fue desarrollado por el investigador Richard FitzHugh. Influenciado por el trabajo de la ecuación de Balthazar Van de Pol, su modelo teórico consistía en dos ecuaciones diferenciales de primer orden, una lineal y una cúbica, además, este modelo propondría una mejor compresión al fenómeno que se estaba estudiando, desde la visualización del comportamiento de órbita en el espacio de fases y esquemas teóricos.

Desde otra parte del mundo, el investigador japones Jin-ichi Nagumo trabajó en un proyecto similar en la simplificación del modelo de HH desde un circuito neuronal, deduciendo un sistema de dos ecuaciones diferenciales semejantes a las de Van de Pol. Motivo por el cual el modelo es atribuido como FitzHugh-Nagumo, ya que uniendo estas dos investigaciones complementan una mejor compresión del sistema dinámico de la neurona, fundamentando al campo de la fisiología de la célula nerviosa.

1.4. La neurona mecánica

Al haber estudiado algunos tipos de modelos es importante mencionar que existen modelos geométricos que pueden representar la dinámica de una neurona biológica, tales proceso como: el potencial de reposo, el umbral de disparo y el periodo refractario, uno de estos modelos es la neurona mecánica.

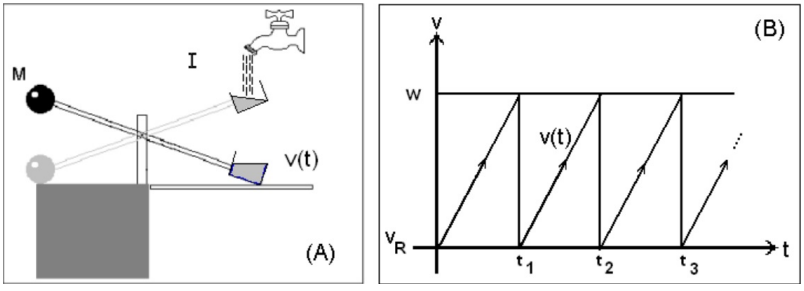


Figura 1.3: Neurona mecánica, disparos de la neurona

La neurona mecánica consiste en que, en un extremo de una balanza se coloca un peso fijo **M**, del otro lado se coloca un recipiente en el cual se está vertiendo agua a un ritmo constante. Cuando el peso del agua es mayor que el peso fijo en el extremo contrario, el brazo de la palanca que contiene al recipiente con el agua desciende hasta el piso, el agua se vacía en un instante, luego, el brazo con el peso vuelve a bajar y todo el proceso se vuelve a repetir.[12],[2]

De acuerdo a la analogía de la neurona mecánica, las variaciones del potencial de membrana de la célula queda simulado por las variaciones de cantidad de agua en el recipiente de sube y baja, los potenciales de acción quedan representados por las descargas del recipiente.

Ahora, ¿Qué pasaría si la neurona mecánica se le aplicara una fuerza forzada $A(t) = -\alpha sen(2\pi t)$? El comportamiento que tendría sería más interesante de estudiar, en este caso la manera simple y conveniente sería proponer un elevador debajo del sube

y baja, cuya altura esté expresada en función del tiempo por una función periódica $h(t)$, como lo indica la figura, para hacer que ahora las descargas del recipiente ocurran a diferentes niveles, que posiblemente varían periódicamente.

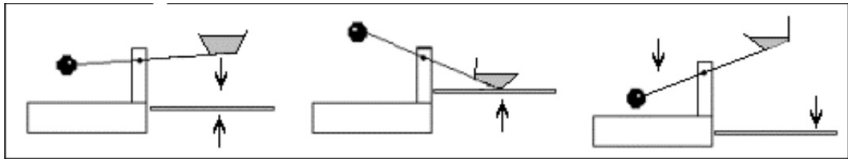


Figura 1.4: Sube y baja del ascensor

Suponiendo que el agua cae a una razón constante m, la gráfica del volumen del agua en el recipiente contra el tiempo podría representarse de la siguiente manera:

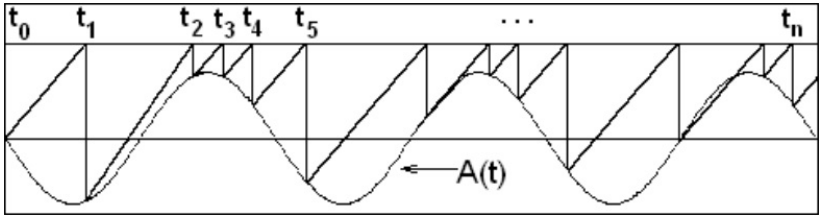


Figura 1.5: Forzamiento periódico

Se puede deducir en una forma sencilla que la sucesión de tiempos de disparo $\{t_0, t_1, t_{2,\ldots}, t_n\}$ que genera este sistema dinámico a partir de un tiempo inicial t_0 es

$$t_{n+1} = t_n + a + bsen(2\pi t_n),$$

donde a y b son parámetros que dependen de $A(t)$ y M .

Mediante unos sencillos cálculos es posible dar una función de disparos, tal que a partir de un tiempo t_n podemos calcular el tiempo t_{n+1} donde ocurre la descarga o disparo del sistema

$$\begin{aligned}
(x_1, y_1) &= (t_n, Asen(2\pi t_n)) \\
(x_2, y_2) &= (t_{n+1}, W) \\
m &= \frac{W + Asen(2\pi t_n)}{t_{n+1} - t_n} \\
t_{n+1} - t_n &= \frac{w}{m} + \frac{A}{m} sen(2\pi t_n) \\
t_{n+1} &= t_n + \frac{w}{m} + \frac{A}{m} sen(2\pi t_n) \\
a &= \frac{W}{m}, b = \frac{A}{m}
\end{aligned}$$

$$t_{n+1} = t_n + a + bsen(2\pi t_n)$$

Es decir, $t_{n+1} = f(t_n) = t_n + a + bsen(2\pi t_n)$, sin necesidad de efectuar el proceso de carga a partir de $(t_n, 0)$ hasta llegar al punto $(t_{n+1}, 1)$. A la función de disparos $t_{n+1} = t_n + a + bsen(2\pi t_n)$ se le conoce como la función de Arnold.

Capítulo 2

Preliminares matemáticos

En este capítulo se abordarán conceptos y notación matemática preliminares que se estarán utilizando durante el transcurso de este trabajo.[10, 11, 8, 16],[21]

2.1. Espacios vectorial y matricial

Definición 1 *Definimos un vector de dimensión n como un conjunto de elementos de números reales, el conjunto de todos los vectores de dimensión se representa como* $\mathbb{R}^n$ *y se representa de la siguiente manera:*

$$v = (v_1, v_2, \ldots, v_n)^T \tag{2.1}$$

Teorema 2 *Si* $u = (u_1, u_2, \ldots, u_n)^T, v = (v_1, v_2, \ldots, v_n)^T$ *y* $w = (w_1, w_2, \ldots, w_n)^T$ *son vectores en* $\mathbb{R}^n, \alpha$ *y* β *son escalares, entonces;*

i) $u + v = v + u$

ii) $u + (v + w) = (u + v) + w$

iii) $u + 0 = 0 + u = u$

iv) $u + (-u) = 0$

v) $\alpha\ (\beta\ u) = (\alpha\beta)\ u$

vi) $\alpha(u + v) = \alpha u + \alpha\ v$

vii) $(\alpha + \beta)\ u = \alpha u + \beta\ v$

viii) $1u = u$

Definición 3 *Una matriz de A de $m \times n$ es un arreglo bidimensional de mn números ordenados en m renglones y n columnas.*

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \dots & \dots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \tag{2.2}$$

Teorema 4 *Sean A, B y C tres matrices de $m \times n$ y sean α y β dos escalares. Entonces:*

i) $A + 0 = A$

ii) $0A = 0$

iii) $A + B = B + A$

iv) $(A + B) + C = A + (B + C)$

v) $\alpha(A + B) = \alpha A + \alpha B$

vi) $1A = A$

vii) $(\alpha + \beta)A = \alpha A + \beta A.$

2.1.1. Productos de vectores y matrices

Definición 5 *Sean* $a = (a_1, a_2, \ldots, a_n)^T$ *y* $b = (b_1, b_2, \ldots, b_n)^T$ *dos* $n-$*vectores. Entonces el producto punto de* a *y* b *, representado por* $a \cdot b$ *esta dado por*

$$a \cdot b = a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_n \cdot b_n \tag{2.3}$$

Definición 6 *Si* u *y* v *son vectores en* $\mathbb{R}^n$ *y* α *es un escalar cualesquiera, entonces;*

i) $\mathrm{u} \cdot \mathrm{v} = \mathrm{v} \cdot \mathrm{u}$

ii) $(\mathrm{u} + \mathrm{v}) \cdot \mathrm{w} = \mathrm{u} \cdot \mathrm{w} + \mathrm{v} \cdot \mathrm{w}$

iii) $(\alpha\ \mathrm{u}) \cdot \mathrm{v} = \alpha\ (\mathrm{u} \cdot \mathrm{v})$

iv) $\mathrm{v} \cdot \mathrm{v} \geq 0$. Además, $\mathrm{v} \cdot \mathrm{v} = 0$ sí y solo si $\mathrm{v} = 0$

Definición 7 *Si* $A = [a_{ij}]$ *es una matriz de* $m \times n$ *y* $B = [b_{ij}]$ *es una matriz de* $n \times p$ *, entonces el producto de* AB *es una matriz de* $m \times p$

$$AB = [c_{ij}]$$

donde

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}$$

2.2. Diferenciación

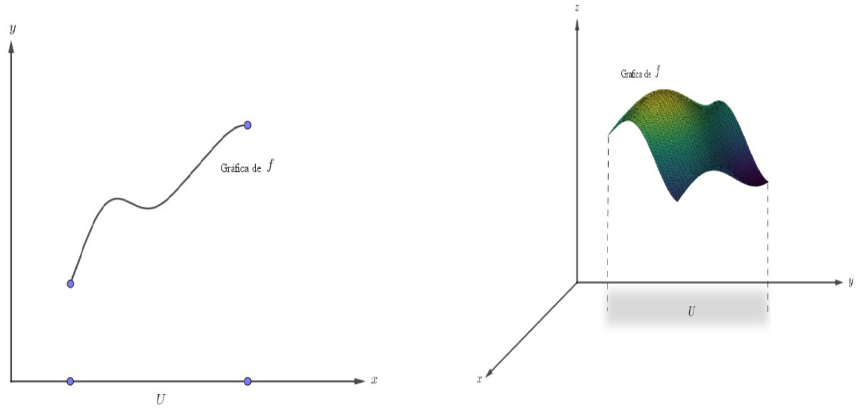
Definición 8 Gráfica de una función

Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}$. *Definimos la gráfica de* f *como el subconjunto de* $\mathbb{R}^{n+1}$ *formado por los puntos*

$$(x_1, \cdots, x_n, f(x_1, \cdots, x_n))$$

de $\mathbb{R}^{n+1}$ *en los que* $(x_1, \cdots, x_n)$ *es un punto de* U. *Simbólicamente*

$$f = \{(x_1, \cdots, x_n, f(x_1, \cdots, x_n)) \in \mathbb{R}^{n+1} \mid (x_1, \cdots, x_n) \in U\}$$



(a) Gráfica de una función de una variable.

(b) Gráfica de una función de dos variables.

Figura 2.1: Gráficas de una función de una variable y de dos variables.

Definición 9 Curvas y superficies de nivel

Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}$ y sea $c \in \mathbb{R}$. Entonces, el conjunto de nivel valor c se define como el conjunto de los puntos $x \in U$ en los cuales $f(x) = c$. Si $n = 2$ hablaremos de curva de nivel (de valor c); y si $n = 3$, hablaremos de superficie de nivel. Con símbolos, el conjunto de nivel valor c se escribe:

$$\{x \in U \mid f(x) = c \subset \mathbb{R}^n\}$$

Nótese que el conjunto de nivel siempre está en el dominio de la función

Definición 10 Derivadas parciales

Sea $U \subset \mathbb{R}^n$ un conjunto abierto y supóngase que $f : U \subset \mathbb{R}^n \to \mathbb{R}^n$ es una función con valores reales. Entonces $\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n}$ las derivadas parciales de f con respecto de la primera, segunda,.., $n-$ésima variable, son las funciones con valores reales de

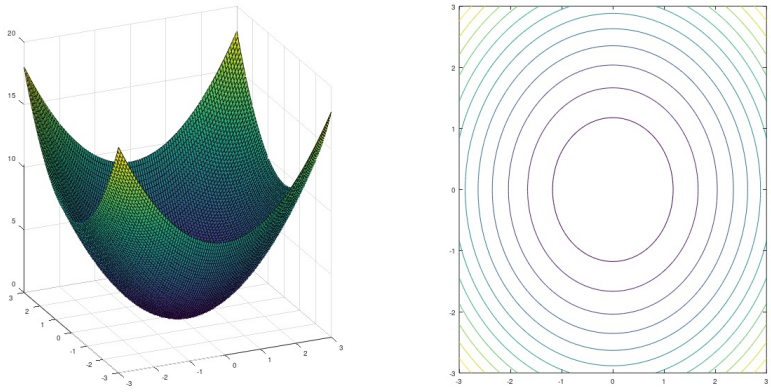


Figura 2.2: Gráficas de una curva y una superficie de nivel.

n variables que, en el punto $(x_{1,\dots}, x_n)$ *se define como;*

$$\begin{aligned} \frac{\partial f}{\partial x_j}(x_1, \cdots, x_n) &= \lim_{h\to 0} \frac{f(x_1, x_2, \cdots, x_j+h, \cdots, x_n) - f(x_1, \cdots, x_n)}{h} \\ &= \lim_{h\to 0} \frac{f(x+he_j)-f(x)}{h} \end{aligned}$$

si los límites existen, donde $1 \leq j \leq n$ *y* e_j *es el vector* $j-$*ésimo de la base canónica definido por* $e_j = (0, \cdots, 1, \cdots, 0)$ *con el* 1 *en la posición* $j-$*ésima. El dominio de la función* $\partial f/\partial x_j$ *es el conjunto de* $x \in \mathbb{R}^n$ *para los que el límite existe.*

En otras palabras, $\partial f/\partial x_j$ es la derivada de f con respecto de la variable x_j, con las otras variables fijas. Si $f : \mathbb{R}^3 \to \mathbb{R}$, a menudo utilizaremos la notación $\partial f/\partial x$, $\partial f/\partial y$, $\partial f/\partial z$, en vez de $\partial f/\partial x_1$, $\partial f/\partial x_2$, $\partial f/\partial x_3$. Si $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$, entonces podemos escribir

$$f(x_1, ..., x_n) = (f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n))$$

de forma que podemos hablar de las derivadas parciales de cada componente; por ejemplo $\partial f_m/\partial x_n$ es la derivada parcial de la componente $m-$*ésima* respecto de x_n, la variable $n-$*ésima*.

Definición 11 Diferenciabilidad: El caso general

Diferenciable, n variables, m funciones. Sea U un conjunto abierto en $\mathbb{R}^n$ y sea $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$ una función dada. Decimos que f es diferenciable en $x_0 \in U$ si las derivadas parciales de f existen en x_0 y, además,

$$\lim_{x \to x_0} \frac{\|f(x) - f(x_0) - T(x - x_0)\|}{\|x - x_0\|}$$

donde $T = Df(x_0)$ es la matriz de $m \times n$ con elementos $\frac{\partial f_i}{\partial x_j}$ evaluadas en x_0 y $T(x - x_0)$ es el producto de T por $x - x_0$ (visto como matriz columna). Llamamos a T a la derivada de f en x_0.

2.3. Propiedades de la derivada

Teorema 12 Sumas, productos y cocientes

i) **Regla de multiplicación por una constante.**

Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$ una función diferenciable en x_0 y sea c un número real. Entonces $h(x) = cf(x)$ es diferenciable en x_0

$$Dh(x_0) = cDf(x_0)$$

ii) **Regla de la suma**

Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$ y $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$ diferenciables en x_0. Entonces $h(x) = f(x) + g(x)$ es diferenciable en x_0 y

$$Dh(x_0) = Df(x_0) + Dg(x_0)$$

iii) **Regla del producto**

Sea $f : U \subset \mathbb{R}^n \to \mathbb{R}^m$ y sean $g : U \subset \mathbb{R}^n \to \mathbb{R}^m$ diferenciables en x_0 y sea $h(x) = f(x) \cdot g(x)$. Entonces $h : U \subset \mathbb{R}^n \to \mathbb{R}^m$ es diferenciable en x_0 y

$$Dh(x_0) = g(x_0)Df(x_0) + f(x_0)Dg(x_0)$$

iv) **Regla del cociente**

De la misma hipótesis que en la regla iii), sea $h(x) = f(x)/g(x)$ y supóngase que g nunca se anula en U. Entonces h es diferenciable en x_0 y

$$Dh(x_0) = \frac{g(x_0)Df(x_0) - f(x_0)Dg(x_0)}{[g(x_0)]^2}$$

Teorema 13 ***La regla de la cadena***

Sean $U \subset \mathbb{R}^n$ y $V \subset \mathbb{R}^m$ dos conjuntos abiertos. Sean $g : U \subset \mathbb{R}^n \to \mathbb{R}^m$ y $f : V \subset \mathbb{R}^m \to \mathbb{R}^p$ funciones dadas tales que g manda a U en V, de forma que $f \circ g$ está definida. Supóngase que g es diferenciable en x_0 y que f es diferenciable en $y_0 = g(x_0)$. Entonces $f \circ g$ es diferenciable en x_0 y

$$D(f \circ g)(x_0) = Df(y_0)Dg(x_0)$$

Definición 14 *Si $f : U \subset \mathbb{R}^3 \to \mathbb{R}$ es diferenciable, el gradiente de $f(x, y, z)$ es el vector del espacio $\mathbb{R}^3$ dado por*

$$\bigtriangledown f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) \tag{2.4}$$

Este vector también se denota por $\bigtriangledown f = (f_x, f_y, f_z)$. Por lo tanto, en este caso, $\bigtriangledown f$ es exactamente la matriz de la derivada Df escrita como vector.

2.4. Optimización: el descenso del gradiente

El descenso del gradiente es un algoritmo de optimización iterativo de primer orden para encontrar un mínimo local de una función diferenciable, en él se toman pasos proporcionarles al negativo del gradiente de la función en el punto actual con el objetivo de minimizar la función.[13]

Consideremos las funciones multivariables e imaginemos una superficie descrita por la función de $f(x)$ con un punto aleatorio x_0. El descenso del gradiente aplica que $f(x_0)$ disminuye más rápido si la iteración se mueve desde x_0 en la dirección del gradiente negativo $-(\bigtriangledown f(x_0))$. Tenemos la siguiente ecuación:

$$x_1 = x_0 - \eta \nabla f(x_0) \tag{2.5}$$

donde $\eta > 0$ es un parámetro pequeño definido como la **tasa de aprendizaje**. La tasa de aprendizaje η determina el tamaño de los pasos que damos para alcanzar un mínimo local. En otras palabras, seguimos la dirección de la pendiente de la superficie creada por la función objetivo cuesta abajo para llegar al valle.

Si tenemos un punto inicial x_0 y nos movemos a una distancia η en la dirección del gradiente negativo, podemos definir una fórmula de esquema iterativo para calcular el paso x_{n+1} a partir de x_n, siguiendo que:

$$x_{n+1} = x_n - \eta \nabla f(x_n) \tag{2.6}$$

A partir de una condición inicial x_0 los valores descienden poco a poco hasta encontrar un mínimo local. Este proceso puede tomar miles de iteraciones, por lo que normalmente se implementa el desceso de gradiente con apoyo de la programación numérica. El método del descenso de gradiente puede converger de forma rápida o de una forma lenta para encontrar el mínimo de una función.

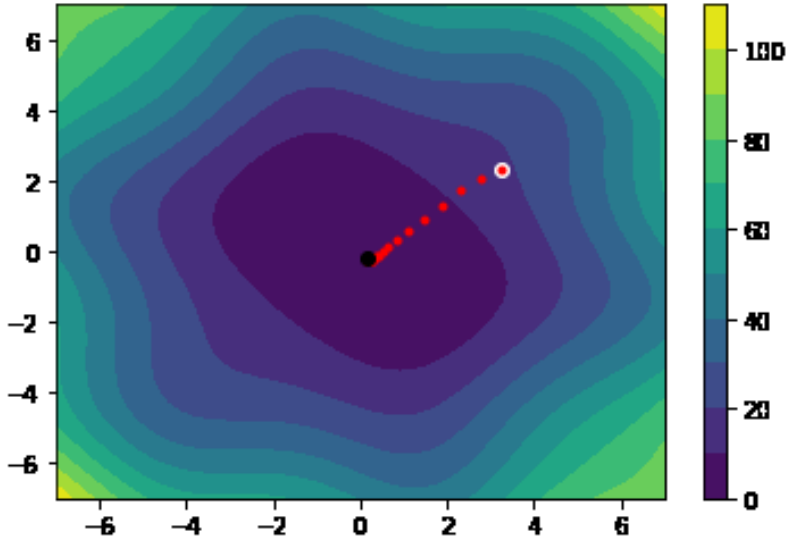


Figura 2.3: Descenso de gradiente por la función $f(x) = x^2 + y^2$

2.5. La tasa de aprendizaje

Elegir un buen tamaño de paso η es importante en el descenso de gradiente. Si el tamaño del paso es demasiado pequeño, el descenso del gradiente puede ser lento. Si el tamaño del paso se elige demasiado grande, el descenso de gradiente puede sobrepasarse, no estrecharse o incluso diverger. La limitación que puede tener el método del descenso de gradiente es que sólo encuentre mínimos locales, este comportamiento únicamente estudia el área de la región para encontrar un mínimo siempre y cuando la tasa de aprendizaje sea la adecuada en las iteraciones.

Hacemos énfasis que es de suma importancia optimizar el tamaño de paso η, para que la tasa de aprendizaje sea óptima y eficaz al hacer cada iteración. Una última limitación es que el descenso de gradiente funciona cuando nuestra función objetivo es diferenciable en todas partes. Por esta razón es que se prefiere utilizar la función de costo cuadrática promedio.

2.6. La función de costo

La función de costo permite medir la diferencia existente entre los datos reales y y los datos obtenidos tras realizar la *regresión lineal*. Existen diferentes formas de definir matemáticamente esta pérdida, pero la más usada es a través del error cuadrático medio, definido de la siguiente manera:

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.7}$$

donde N es el número total de datos que se tienen originalmente, a partir de los datos originales $\hat{y}_i$ y los obtenidos a partir de la regresión y_i, donde la diferencia $(y_i - \hat{y}_i)$ se eleva al cuadrado para que todas las diferencias que están en la suma sean no negativas y no se cancelen entre sí.

A partir de la figura 2.4 podemos hacer unos cálculos para obtener una función de error cuadrático medio. Sea un conjunto de puntos $D = \{(x_1, d_1), ..., (x_n, d_n)\}$, ubicamos los puntos en el plano cartesiano para ajustar los puntos en una recta, tenemos;

$$\begin{aligned} f(x_i) &\approx d_i \\ f(x_i) &= d_i + \hat{e}_i \end{aligned} \tag{2.8}$$

donde a $\hat{e}_i$ lo definimos como el error de la aproximación, y es de interés controlar estos errores. Hacerlos tan pequeños para que la recta se pueda ajustar lo mejor posible a los puntos.

Consideremos lo siguiente

$$\begin{aligned} \hat{e}_i &= f(x_i) - d_i \\ &= |f(x_i) - d_i| \\ &= |d_i - f(x_i)| \end{aligned} \tag{2.9}$$

en esta definición de error se utiliza la función del valor absoluto, la cual no es diferenciable, por esta razón se prefiere utilizar

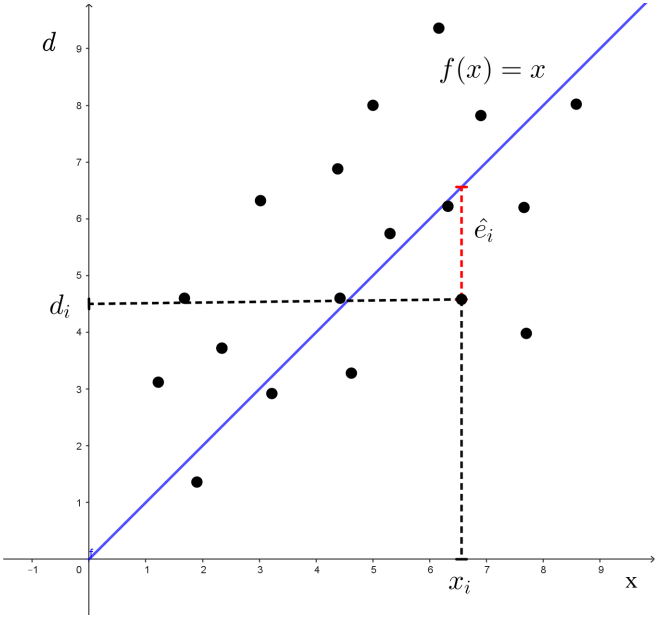


Figura 2.4: Ejemplo del cálculo del error cuadrático medio para un par de puntos

funciones cuadráticas, las cuales son diferenciables, para definir el error.

Definición 15 *La función de costo o función del error cuadrático medio es:*

$$\begin{aligned} J &= \frac{1}{2}(d-y)^2 \\ &= \frac{1}{2}(\hat{e})^2 \end{aligned} \tag{2.10}$$

Capítulo 3

Redes neuronales

La ciencia de la computación ha cobrado mayor importancia en la tecnología y en la vida del ser humano, pero todo esto ha sido el fruto o consecuencia de muchos años de investigación y de científicos que han hecho aportaciones fundamentales para construir la base de la computación.

Así, una de las áreas que se ha desarrollado actualmente es la inteligencia artificial. En realidad, pretender dar una definición de lo que es la inteligencia artificial nos lleva a una tarea complicada, por lo mismo de que es un concepto que depende de la propia definición de la **inteligencia,** que al día de hoy sigue siendo un enigma, donde varios autores han dado su propia definición y si nosotros tratáramos de dar una definición, en particular con una idea en común, podríamos decir que la *IA* trata de dotar a un ente artificial con la capacidad de realizar trabajos que requieran inteligencia, como lo haría un ser humano. Además, en la *IA* podemos encontrar diferentes ramas que han podido replicar máquinas inteligentes, sin embargo, si hay algo que en verdad nos define como la especie superior de este mundo, es la capacidad de aprender, y la rama que estudia este comportamiento es el Aprendizaje Automático o en la forma más conocida en inglés el *Machine Learning*.

El Machine Learning es una rama de la *IA* que estudia cómo

dotar a las máquinas para que tengan la capacidad de "aprender" a partir de un conjunto de experiencias, dentro de esta rama han surgido varias técnicas que tienen como propósito cubrir diferentes tipos de aplicaciones tales son como: los árboles de decisión, modelos de regresión, técnicas de agrupamiento o clusterización de datos entre otros. Pero si una de estas técnicas que han surgido le ha dado popularidad al campo de Machine Learning en la última década son las redes neuronales.

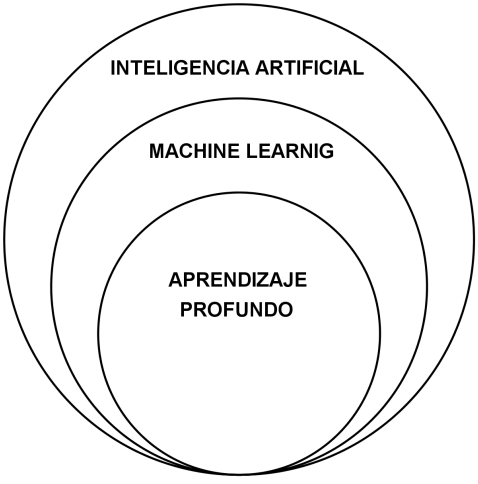


Figura 3.1: Diagrama de la inteligencia artificial y su contenido.

3.1. Antecedentes

Algunos pioneros al campo de la ciencia de la computación lograron que cobrara importancia al estudio de las redes neuronales a inicios del siglo XX, la evolución que tuvo que trascender para mejorar técnicas y las facilidades para poder simular una inteligencia artificial.

Alan Turing, un matemático e informático inglés que sentaría el inicio del comportamiento del cerebro a la relación de la

computación al proponer la metodología de cómo una máquina funcionaría para aprender, proporcionó un aprendizaje fundamental a los conceptos de algoritmo, pero no fue hasta el año de 1936, cuando publicó un trabajo original, donde postulaba que una máquina era capaz de resolver cualquier problema matemático a partir de un proceso de algoritmo, conocido como la Máquina de Turing, un objeto que sería el primer modelo teórico para las computadoras, donde constituiría la base del funcionamiento de las computadoras digitales. En 1950 publicó un artículo *"Computing Machinery Intelligence"* [17], donde planteaba una prueba, la Prueba de Turing, en la que él trataba de explicar cómo una maquina era capaz de pensar. La prueba trataba de que una persona evaluara las conversaciones entre un humano real y una máquina, donde el evaluador podría distinguir las respuestas de cada uno de ellos y deducir si la conversación correspondía a la de una máquina.

Warren McCulloch y Walter Pitts, dos icónicos personajes que propondrían el primer modelo de una neurona artificial. En 1943 explican en su artículo *"A logical Calculus of ideas Immanent in Nervous Activit"* [14], explican cómo simular el comportamiento complejo del cerebro humano en las interacciones de las células nerviosas, llevándolo al sistema computacional. El modelo está basado en que las neuronas operan mediante impulsos binarios, es decir, la neurona opera como valores de entrada binaria, y devuelve una salida binaria, este trabajo demuestra un riguroso análisis matemático, sentando una gran contribución para las futuras investigaciones.

Donald Hebb fue un psicólogo, presentó por primera vez la regla de aprendizaje en su libro *"Organization of Behavior"* [5] en 1949. Él expone que el cerebro está cambiando continuamente por las distintas tareas funcionales que el cerebro realiza, presentando una regla de aprendizaje donde menciona que durante el proceso de una sinapsis entre dos neuronas puede incrementar la activación o puede ocurrir algún cambio metabólico en una o en ambas células, dicha aportación es lo que hoy en día se le conoce

como el Aprendizaje de Hebb o la Regla de Hebb, además de que se considera como un *aprendizaje no supervisado*.

Frank Rosenblatt fue un psicólogo estadounidense, después de la publicación de McCulloch un nuevo panorama existiría por la problemática del reconocimiento de patrones y esta solución fue introducido por Rosenblatt en su investigación al proponer una nueva generalización de modelo de una red de neurona llamada el perceptrón, un novedoso método de *aprendizaje supervisado* donde la innovación esencial fue la introducción de pesos numéricos, en el que el aprendizaje se lleva a cabo ajustando los pesos de las conexiones entre los niveles de entrada y salida. Demuestra que el aprendizaje del perceptrón converge hacia un estado finito, mejor conocido como el teorema de convergencia del perceptrón, es decir, si el problema que se está trabajando tiene solución, el modelo garantiza encontrar la existencia de esta.

En 1960 **Bernad Widrow** y **Marcian Hoff** desarrollaron la primera serie de sistemas adaptativos especializados para el procesamiento de señales, crearon el modelo ADALINE (*Adaptative Linear Element*) y MADALINE (*Multiple Adaptative Linear Element*), una red neuronal aplicada para el procesamiento adaptativo de sistemas de control y sistemas adaptivos de antenas, esta red neuronal artificial tiene una gran similitud al perceptrón, ajusta los pesos entre los niveles de entrada y salida en función del error entre el valor esperado de salida y el obtenido, estos autores probaron matemáticamente que el error de la salida deseada y obtenida puede ser minizado hasta el límite usando la aplicación mediante la *regla Delta*.

3.2. ¿Qué es una red neuronal?

Las redes neuronales artificiales son un modelo computacional inspirado en el funcionamiento de las neuronas del cerebro humano, un sistema que es capaz de adquirir conocimiento

a través de la experiencia. Consiste en un conjunto de unidades que están altamente interconectadas, intercambiando datos e información para obtener una respuesta de salida.

3.2.1. Arquitectura de una red neuronal

Una red neuronal está formada por un conjunto estructurado de neuronas, conocidas como capas, estas neuronas tiene conexiones repetidas que reciben en la siguiente capa de valores de entrada, luego las envía a otras neuronas para continuar el proceso de datos, mediante una suma ponderada con el objetivo de tener un valor de salida. A cada una de las conexiones de entrada se les asocia un valor de peso que servirá para ajustar con qué intensidad afecta la neurona.

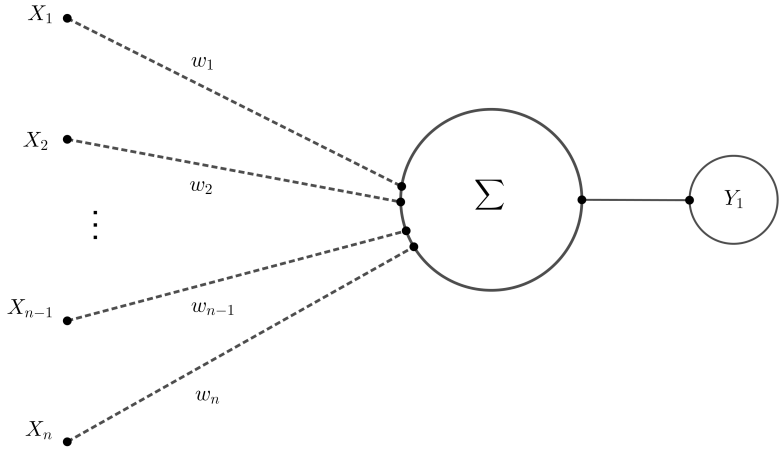


Figura 3.2: Estructura de una neurona artificial

En el capítulo 1 nos damos cuenta que hay una estrecha analogía con las neuronas biológicas, tenemos un modelo computacional que representa la estructura de una neurona, que en este caso sería la artificial. Los valores de entradas que recibe la neurona vendrían representando el proceso sináptico, el proce-

so donde toda la información es reunida y procesada representaría el núcleo de la neurona y por último tenemos una salida que estaría representando el axón donde conduce la información procesada para poder enviar a otras neuronas.

Una vez hecho el esquema gráfico, es importante relacionarlos como términos matemáticos, empezar a ponerle nombres a los valores que la neurona recibe y parametrizarlo, esto con el propósito de que el modelo sea sofisticado y eficaz; aquellos valores de entrada que recibe la neurona lo definimos como la variable x_j donde el subíndice j puede tomar valores desde $j = \{1, 2, \cdots, n\}$, estas entradas de valores puede ser representadas mediante un vector:

$$X = \{x_1, x_2, \cdots, x_n\}^T$$

Sin embargo, a partir de la morfología de la neurona y el conocimiento que hemos estudiado, sabemos que las conexiones sinápticas poseen distintos valores, diferentes características, de alguna manera esta restricción nos hace crear o introducir un nuevo parámetro para que mida las diferencias de los pesos sinápticos, y ese parámetro lo vamos a nombrar con w_j donde el subíndice j puede tomar valores de $j = \{1, 2, \cdots, n\}$ y podríamos representar en forma de vector de la siguiente manera:

$$W = (w_1, w_2, \cdots, w_n)^T$$

Además, es importante recordar que a partir de las señales de entradas sinápticas que recibe, la neurona puede entrar en dos estados de activación, produciendo un estado de salida de excitación o inhibición, traduciendo esto en término matemático, decimos que las entradas x_j pueden tomar dos estados definidos como:

$$x_j = \begin{cases} 1 \text{ estado exitatorio} \\ 0 \text{ estado inhibitorio} \end{cases}$$

En el proceso neuronal donde reúnen todas las entradas sinápticas mencionamos que realiza una suma ponderada, pero si lo

visualizáramos en el ámbito matemático no es más que una función que describe que:

$$f(x) = \sum_{j=1}^{n} x_j w_i$$

Deduciendo que la salida tenemos:

$$\begin{aligned} y &= f(X, W) \\ &= f(x_1 \cdot w_{1,\,\dots\,}, x_n \cdot w_n) \end{aligned}$$

donde y es la salida, f es la función, X y W serían las variables de entrada y el peso correspondientemente.

Lo sorprendente de las redes neuronales son capaces de aprender de forma jerarquizada, es decir, la información se aprende por niveles, proceso en donde las primeras capas aprenden de una introducción y en las capas posteriores se usa este aprendizaje para optimizar el resultado que se espera. Lo fundamental está en que las conexiones de cada neurona que comparte y envía información genera como una experiencia de aprendizaje previa para tener un mejor resultado.

3.3. Función de activación

La función de activación, definida como $\varphi()$, evalúa la información generada dada por un conjunto de entradas a través de una combinación lineal de sus pesos y establecen un límite para la salida en el intervalo de $[0, 1]$ o alternativamente en $[-1, 1]$, es decir, la neurona se activará o no, dependiendo de los valores de entrada y de la ponderación que se haga, un último recurso para que la salida de la neurona sea competente. A continuación se explicarán algunas de las funciones de activación más comunes que se han estado utilizando.

- *Función de activación lineal*

  Tenemos la función de activación lineal, la función más básica que podemos implementar. En términos matemáticos podemos observar la función identidad, lo que hace el resultado de la neurona es que devuelve el valor y no cambia en lo absoluto y está definida como:

$$\varphi(x) = x$$

- *Función de activación rampa*

  Esta función da como salida valores en el intervalo $[-1, 1]$, y se aplica la función lineal a ese intervalo. La salida de la neurona se activa solo cuando el estado de activación respete esta función.

$$\varphi(x) = \begin{cases} -1 & \text{si} \quad f(x) < -1 \\ x & \text{si} \quad -1 \leq f(x) \leq 1 \\ 1 & \text{si} \quad x > 1 \end{cases}$$

- *Función de activación sigmoide*

  Se define como una función estrictamente creciente, posee como imagen un rango continuo en el intervalo $[0, 1]$, el valor de la salida se encuentra comprendido entre la zona alta o baja de la función sigmoidal.

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

- *Función de activación arcotangente*

  Esta función tiene la misma propiedad que la función sigmoidal, solo que en este caso el intervalo de valor de salida puede comprender de $[-\pi/2, \pi/2]$.

$$\varphi(x) = \arctan(x)$$

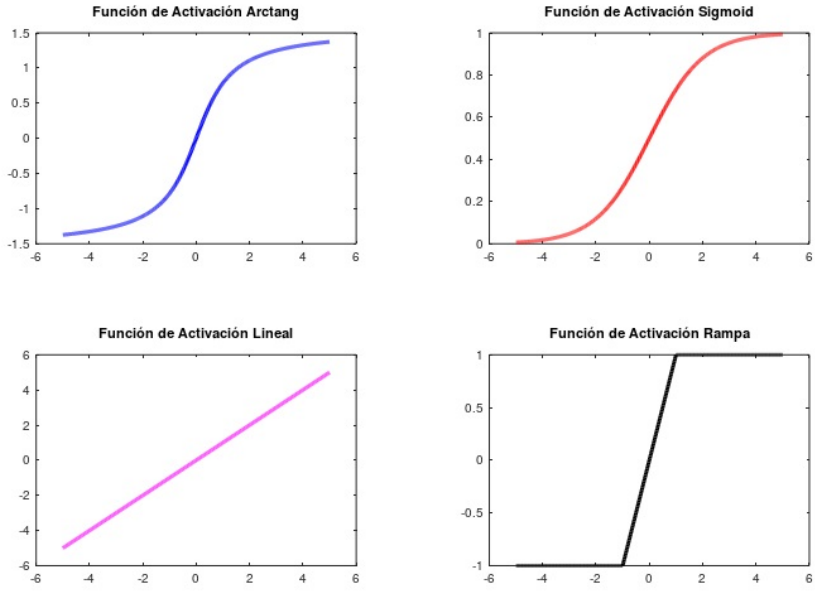


Figura 3.3: Gráficas de las funciones de activación

3.4. Paradigmas de aprendizajes

Una red neuronal tendrá la capacidad de realizar el proceso de aprendizaje al ingresar los valores de entrada y que nos devuelva una respuesta de salida favorable, el cual será capaz de almacenar conocimiento, a este proceso se le conoce como entrenamiento o regla de aprendizaje, es decir, este mecanismo de aprendizaje realiza todo un proceso al modificar los pesos de las variables de entrada con el fin de poder ajustar que valores son los adecuados para obtener un resultado de salida esperado.

Esto nos lleva a tal punto que todos los algoritmos y técnicas que se han realizado en este campo han sido entrenados y clasificados en tres grandes grupos: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

3.4.1. Aprendizaje supervisado

El aprendizaje supervisado es un método que busca descubrir alguna relación existente entre los datos de entrada y los datos de salida, dicho de otra forma, el aprendizaje surge a través de enseñar a la red cuál es el resultado que se espera obtener al proporcionarle varios datos, además, este método se realiza mediante un entrenamiento controlado por un agente externo que se encarga de supervisar la respuesta de salida a partir de los datos de entrada. En particular controla la salida de la red y si fuese necesario, modificar los pesos de las entradas con el objetivo de que la salida se aproxime al resultado que queremos obtener. Al denominarle "supervisado", viene del hecho de que en la salida de la respuesta que obtengamos estamos participando en la supervisión de su conocimiento.

3.4.2. Aprendizaje no supervisado

El caso del aprendizaje no supervisado es un método muy interesante, aquí no hay influencia externa que determine si es necesario ajustar los valores de pesos de entrada, no recibe un conocimiento previo para poder predecir qué resultado es lo que se espera. El proceso de entrenamiento que realiza trata de encontrar patrones, correlaciones de características importantes para trabajar con ellas, ya sea aplicando técnicas de agrupamiento o clusterización que se puedan establecer entre los datos de entrada, todo esto dependiendo de la estructura y del algoritmo de la red que se esté trabajando.

3.4.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo o aprendizaje reforzado es otro mecanismo mejorado que se implementa para dar una solución mucho más óptima, en este paradigma el proceso es que el programa o el algoritmo, llamémosle agente, trata de entender lo que

esta sucediendo a su entorno, aprender cada estado que vaya trabajando y a partir de la secuencia de acciones que el agente vaya decidiendo será recompensado en caso se haya hecho la tarea correcta o en caso contrario una advertencia de no repetir el mismo error.

3.5. Tipos de redes

3.5.1. Modelo neuronal de McCulloch-Pitts

El primer modelo matemático de una neurona artificial que se pueda considerar fue en el año de 1943, presentado por Warren McCulloch y Walter Pitts.

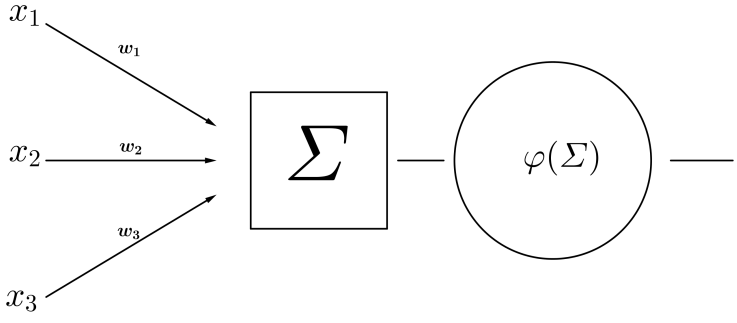


Figura 3.4: Modelo McCulloch-Pitts

El modelo está diseñado para aceptar variables de entrada, realiza la suma ponderada correspondiente aplicando una función de activación y da la salida como un resultado, este modelo se considera como una neurona estándar, ya que su principal característica es trabajar con valores binarios (valores de 0 y 1).

3.5.2. El modelo del perceptrón

El modelo del perceptrón fue creado por Frank Rosenblatt en 1962, inspirado en las primeras etapas de procesamiento a través de los sistemas sensoriales (de la visión).

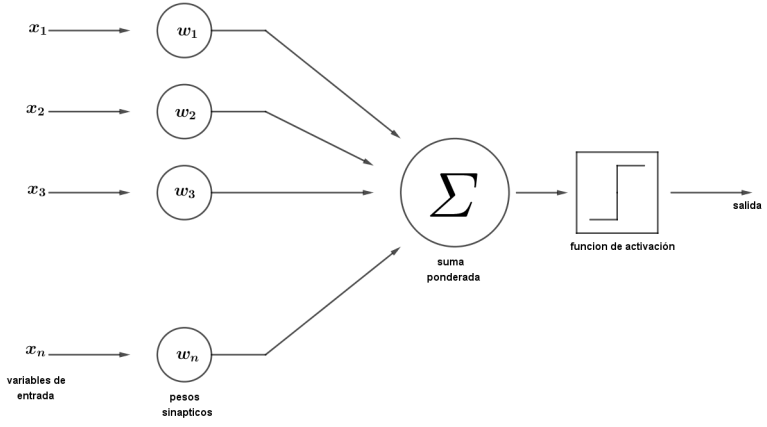


Figura 3.5: Modelo del perceptrón

La arquitectura del modelo es algo simple, se trata de una estructura monocapa, es decir, tenemos una capa de entrada que son las señales de entrada y tenemos la capa de salida que puede ser una o varias señales de salida.

El modelo propone una nueva generalización de una neurona artificial, proyectando un sistema capaz de realizar tareas de clasificación. La importancia del modelo radica en que la red puede ser entrenada, esto a partir de un conjunto de ejemplos que se le dé a la red y de forma automática fuera capaz de clasificar las clases.

3.5.3. El modelo ADALINE

Conocido como Adaptative Linear Neuron (ADALINE), es un modelo muy similar al percetrón, un mecanismo físico creado

en su inicio en un dispositivo electrónico capaz de ser entrenado para la clasificación de patrones.

$$y = \sum_{i=1}^{n} w_i x_i + b$$

El aprendizaje calcula la diferencia entre el valor esperado y el valor de salida, a esta regla de aprendizaje se le conoce como la *regla delta*. El objetivo es minimizar el error cometido de la diferencia de los valores, evaluar globalmente el error para todos los patrones, y existe un criterio para realizar tal proceso conocido como el error cuadrático medio, matemáticamente, el aprendizaje a partir de la salida de una función lineal permite la minimización de una función continua de costo o pérdida.

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

El proceso intentará minimizar este valor para todos los elementos del conjunto de patrones de aprendizaje, la manera de minimizar este error es recurrir a un proceso iterativo en el que se van presentando los patrones, uno a uno, y modificando los parámetros de la red o peso de las conexiones, mediante la regla del *descenso del gradiente*.

3.5.4. El modelo del perceptrón multicapa

El perceptron multicapa surgió a partir de las limitaciones que tenía el perceptron simple, para este modelo la arquitectura está formada por múltiples capas, denominadas de la siguiente manera: la capa de entrada, las capas ocultas y la capa de salida.

La capa de entrada son aquellas señales de neuronas de entrada, las capas ocultas son aquellas neuronas que reciben de la capa anterior al haber hecho la suma ponderada, o el trabajo correspondiente, repitiendo el mismo proceso si la red tiene

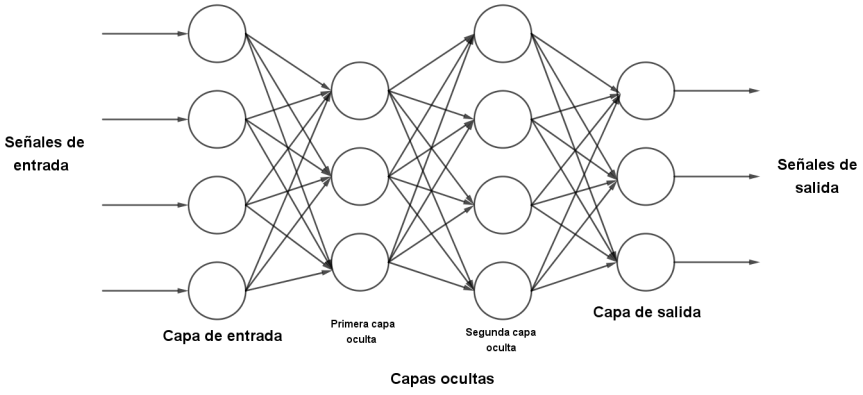


Figura 3.6: Modelo perceptrón multicapa

varias capas ocultas, y por último la capa de salida, en la que puede haber uno o más resultados de salida.

De acuerdo a este modelo, el proceso rige a un procedimiento que siempre va hacia adelante, en donde cada neurona de entrada está interconectada con cada una de las neuronas de la primera capa oculta, formando así un sistema de red amplia, esto con la finalidad de que el modelo sea aún más sofisticado que a los modelos anteriores, esta red puede ser entrenada a través de un conjunto de ejemplos, comprobando experimentalmente que es más eficaz en la clasificación de clases o el mapeo de imágenes, siempre y cuando proporcionando los valores correctos para un resultado más factible. Al usar este modelo de neurona artificial estamos introduciéndonos más a fondo al campo de la inteligencia artificial, que es el aprendizaje profundo.

Capítulo 4

Modelo matemático: redes neuronales artificiales

4.1. Modelo matemático: una neurona sencilla

Como se ha mencionado anteriormente, el primer modelo matemático general de una neurona artificial fue desarrollado por McCulloch-Pitts en 1943 para estudiar el procesamiento de señales del cerebro:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i\right) = \varphi(u)$$

Una representación gráfica puede ser la figura 4.1 .

A partir de aquí se trabajará con una red de neuronas sencilla, para que podemos empezar con un modelo matemático básico hasta analizar una red de neuronas con varias capas. Llamaremos nodos a los distintos parámetros que corresponda para extender y generalizar el modelo que se quiera trabajar.

La siguiente ecuación representa un modelo general de una neurona artificial sencilla, donde la salida y se produce a través de la función de activación $\varphi(z)$, la b_i es denominado como el

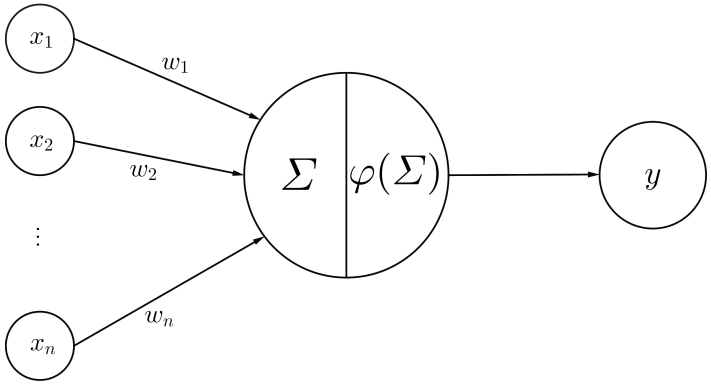


Figura 4.1: Modelo de neurona artificial desarrollado por McCulloch-Pitts

"sesgo", que normalmente se necesita en caso de ponderaciones o entradas de activación en cero.

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i\right) = \varphi(z)$$

En la figura 4.2 podemos visualizar un esquema gráfico para cada uno de los valores de entrada, para cada peso correspondiente, un nodo para las ponderaciones, la función de activación y el valor de la salida.

4.2. Modelo matemático: una capa oculta

Determinemos que los nodos x_i serán los valores de entrada de la primera capa, w_{ij} representarán los pesos correspondientes de x_i, denominaremos preactivación z_i a la suma ponderada de los valores de x_i y w_{ij}, y la función de activación como $\varphi()$.

Para cada nodo x_i en la primera capa oculta, calculamos la

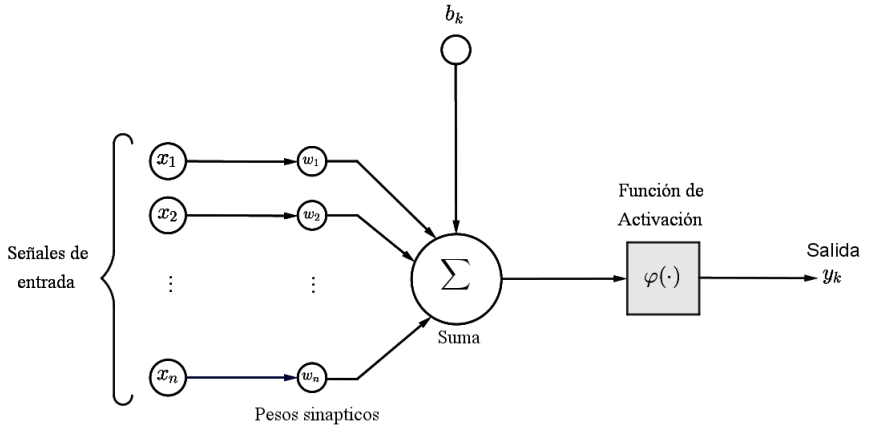


Figura 4.2: Representación gráfica de una red neuronal con una sola capa de neuronas. Los valores x_i son las entradas, los w_j son los pesos, la neurona efectúa una suma ponderada $\sum_{i=1}^{n} w_i x_i$ y da como salida $y = \varphi\left(\sum_{i=1}^{n} w_i x_i\right)$

suma ponderada z_i^1 de las coordenadas x_i:

$$z_i^{(1)} = \sum_{j=1}^{N} w_{ij}^{(1)} x_j$$

El valor $z_i^{(1)}$ es el argumento para la función de activación φ_i de cada nodo i, la variable N representa todas las entradas posibles a un nodo i dado en la primera capa. Definimos la salida $y_i^{(1)}$ de todas las neuronas en la capa 1 como:

$$y_i^{(1)} = \varphi(z_i^{(1)}) = \varphi\left(\sum_{j=1}^{N} w_{ij}^{(1)} x_j\right), \tag{4.1}$$

donde asumimos que todos los nodos en la misma capa tienen funciones de activación idénticas, de ahí la notación φ. En el

caso general, para hacer los cálculos de todas las salidas y_i, identificaremos estas funciones con un superíndice l para la última capa

$$y_i^{(l)} = \varphi(z_i^{(l)}) = \varphi\left(\sum_{j=1}^{Nl-1} w_{ij}^{(l)} y_j^{l-1}\right) \tag{4.2}$$

donde N_l es el número de nodos en la capa l. Cuando se calcula la salida de todos los nodos en la primera capa oculta, se pueden calcular los valores de la capa posterior y así sucesivamente hasta que se obtiene la salida.

A modo de que el lector entienda estas ecuaciones se puede apoyar en la figura 4.3, donde tenemos una red de neurona artificial para una capa oculta.

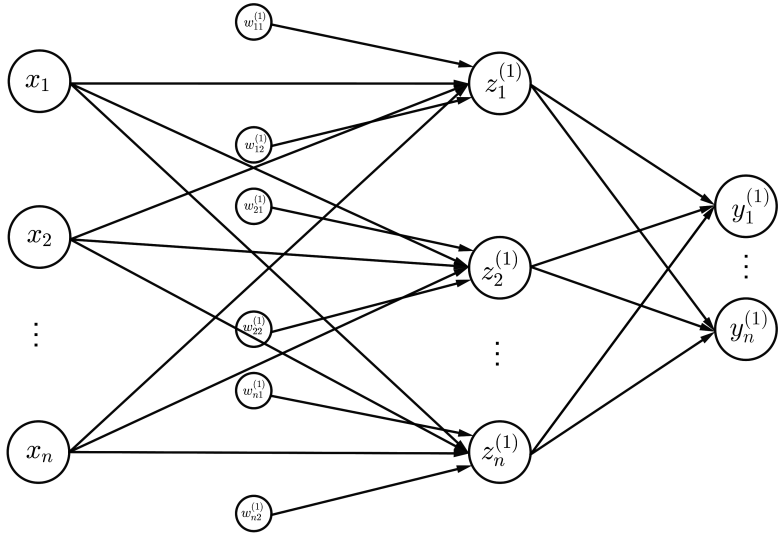


Figura 4.3: Representación gráfica de una red neuronal compuesta de una capa de entrada con nodos $\{x_1, \ldots, x_n\}$, una capa oculta con nodos $\{z_1, \ldots, z_n\}$ y una capa de salida de $\{y_1, \ldots, y_n\}$.

Ejemplo 16 *Ejemplo práctico de una neurona artificial*

Un ejemplo sencillo de cómo procesan los datos de entrada.

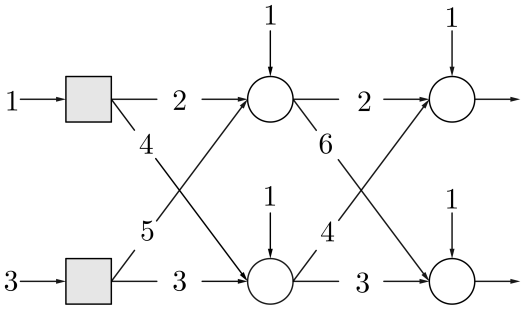


Figura 4.4: Ejemplo de una red neuronal sencilla con valores.

Por conveniencia usaremos la función de activación $\varphi(x) = x$ lineal, esta función permite que los nodos envíen la suma ponderada por sí misma.

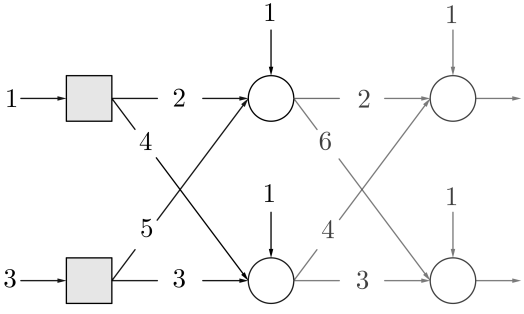


Figura 4.5: Red neuronal para hacer los cálculos en la primera capa

Ahora, calculemos la salida de los nodos de la capa oculta.

- Suma ponderada: $v = (2 \times 1) + (5 \times 3) + 1 = 2 + 15 + 1 = 18$

- Salida $y = \varphi(v) = 18$

Salida del segundo nodo de la capa oculta:

- Suma ponderada: $v = (4 \times 1) + (3 \times 3) + 1 = 4 + 9 + 1 = 14$
- Salida $y = \varphi(v) = 14$

Si la llevamos a su forma matricial:

$$\begin{aligned}
\begin{bmatrix} 2\times 1 & 5\times 3 & +1 \\ 4\times 1 & 3\times 3 & +1 \end{bmatrix} &= \begin{bmatrix} 2 & 5 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 2+15 \\ 4+9 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 18 \\ 14 \end{bmatrix}
\end{aligned}$$

De manera similar, hacemos el mismo proceso para calcular los nodos de la capa de salida

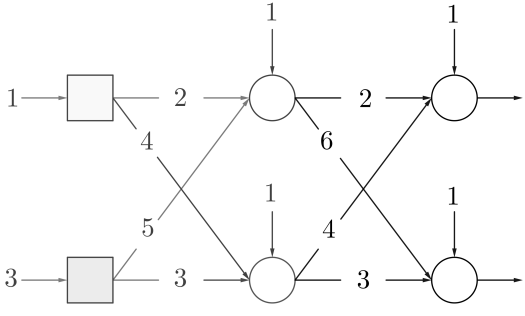


Figura 4.6: Red neuronal para hacer los cálculos para la capa de salida.

Ahora, calculemos la salida del primer nodo de la capa de salida.

- Suma ponderada: $v = (2 \times 18) + (4 \times 14) + 1 = 36 + 56 + 1 = 93$
- salida $y = \varphi(v) = 93$

Salida del segundo nodo de la capa de salida:

- Suma ponderada: $v = (6 \times 18) + (3 \times 14) + 1 = 108 + 42 + 1 = 151$
- salida $y = \varphi(v) = 151$

Los valores de entrada de la red produce una salida tal:

$$\begin{aligned}
\begin{bmatrix} 2 & 4 \\ 6 & 3 \end{bmatrix} \begin{bmatrix} 18 \\ 14 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} &= \begin{bmatrix} 2 \times 18 + 4 \times 14 \\ 6 \times 18 + 3 \times 14 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 36 + 56 \\ 108 + 42 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 92 \\ 150 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&\quad \begin{bmatrix} 93 \\ 151 \end{bmatrix}
\end{aligned}$$

4.3. Modelo matemático: dos capas ocultas

Al trabajar una red para dos capas ocultas se puede considerar que entramos al concepto del perceptrón multicapa. El perceptrón multicapa es una red de neurona artificial formada por múltiples capas. Para poder deducir el modelo matemático para dos capas ocultas, nos apoyaremos del gráfico 4.7, que representa una forma más convencional de trabajar.

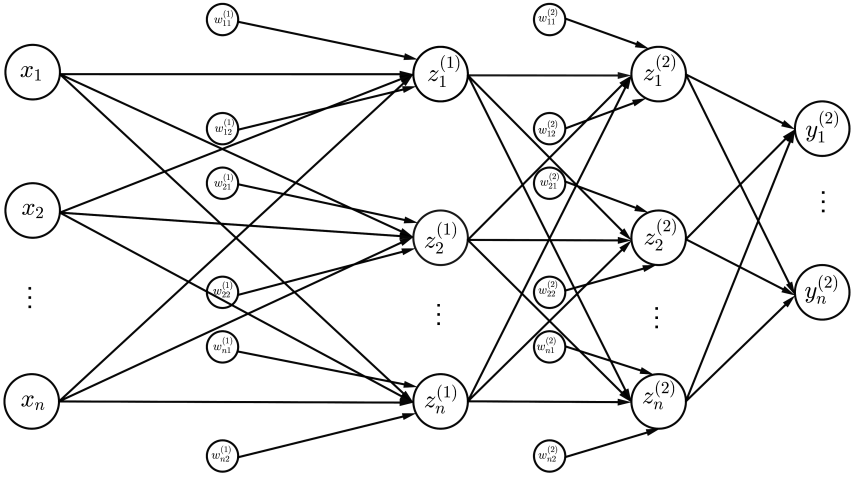


Figura 4.7: Representación gráfica de un perceptrón con una capa de entrada $\{x_1, x_2, \ \dots \ , \ x_n\}$, dos capas ocultas $\{z_1^{(1)}, z_2^{(1)}, \ \dots \ , \ z_n^{(1)}\}$ y $\{z_1^{(2)}, z_2^{(2)}, \ \dots \ , z_n^{(2)}\}$ y una capa de salida $\{y_1^{(2)}, y_2^{(2)}, \ \dots \ , \ y_n^{(2)}\}$. Los pesos $w_{ik}^{(k)}$ son los que ponderan las conexiones de la capa $k-1$ a la capa k.

Si observamos detalladamente el esquema, es indispensable usar esta notación y distinguir los nodos que corresponden para cada capa, usamos la notación del *super-índice* para ubicar en qué momento inicia y termina una capa, donde ahora $w_{ij}^{(2)}, z_i^{(2)}$, y $y_k^{(2)}$ corresponden valores de la segunda capa.

Para la salida de la neurona para $y_i^{(2)}$ tenemos:

$$y_i^{(2)} = \varphi^{(2)} \left(\sum_{j=1}^{N} w_{ij}^{(2)} y_j^{(1)} \right), \tag{4.3}$$

entonces, la salida $y_i^{(2)}$ de la red neuronal se calcula mediante

$$y_i^{(2)} = \varphi^{(2)} \left[\sum_{j=1}^{N} w_{ij}^{(2)} \varphi^{(1)} \left(\sum_{k=1}^{M} w_{jk}^{(1)} x_k \right) \right] \tag{4.4}$$

4.4. Modelo matemático: tres capas ocultas

De forma análoga, para una red con tres capas ocultas, se muestra 4.8, para explicar cómo se comporta la red y encontrar la ecuación que le corresponde.

De modo que para la salida de la neurona para $y_i^{(3)}$ tenemos:

$$y_i^{(3)} = \varphi^{(3)} \left(\sum_{j=1}^{N} w_{ij}^{3} y_j^{(2)} \right) \tag{4.5}$$

$$y_i^{(3)} = \varphi^{(3)} \left[\sum_{j} w_{ij}^{3} \varphi^{(2)} \left(\sum_{k} w_{jk}^{(2)} \varphi^{(1)} \left(\sum_{m} w_{km}^{(1)} x_m \right) \right) \right] \tag{4.6}$$

4.5. Modelo matemático: para l capas

Al seguir trabajando con varias capas, sin importar el número de valores de entradas y cuántas neuronas se consideren en las capas ocultas, es posible generalizar esta expresión a un modelo de perceptrón multicapa con l capas ocultas como se muestra en

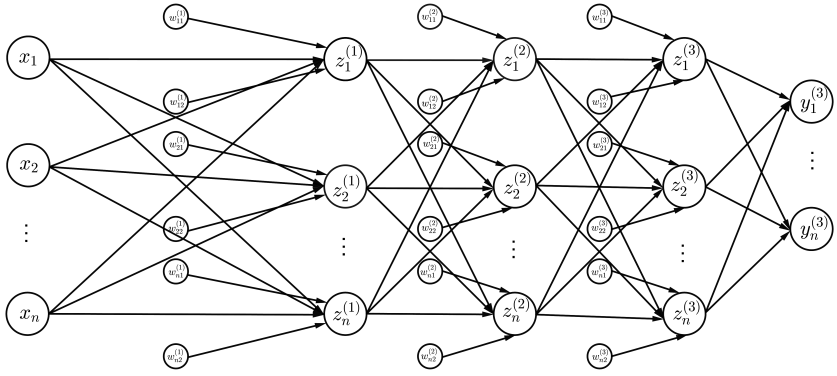


Figura 4.8: Representación gráfica de un perceptrón con una capa de entrada $\{x_1, x_2, \ \dots \ , \ x_n\}$, tres capas ocultas $\{z_1^{(1)}, z_2^{(1)}, \ \dots \ , \ z_n^{(1)}\}$, $\{z_1^{(2)}, z_2^{(2)}, \ \dots \ , z_n^{(2)}\}$ y $\{z_1^{(3)}, z_2^{(3)}, \ \dots \ , z_n^{(3)}\}$ y una capa de salida $\{y_1^{(3)}, y_2^{(3)}, \ \dots \ , \ y_n^{(3)}\}$. Los pesos $w_{ik}^{(k)}$ son los que ponderan las conexiones de la capa $k-1$ a la capa k.

la figura 4.9. La forma funcional completa es la ecuación final para la salida de la neurona para $y_i^{(l+1)}$. Tenemos:

$$y_i^{(l+1)} = \varphi^{(l+1)}\left[\sum_{j=1}^{N_l} w_{ij}^{(l)}\varphi^{(l)}\left(\sum_{k=1}^{N_{l-1}} w_{jk}^{(l-1)}\left(\dots \right.\right.\right.$$
$$\left.\left.\left. \cdots \varphi^{(l)}\left(\sum_{n=1}^{N_0} w_{mn}^{(1)} x_n\right)\cdots\right)\right)\right] \tag{4.7}$$

4.6. Modelo matemático: propagación hacia atrás

En el Capítulo 2, en la sección de la función de costo, explicábamos cómo en un conjunto de puntos se podía ajustar a

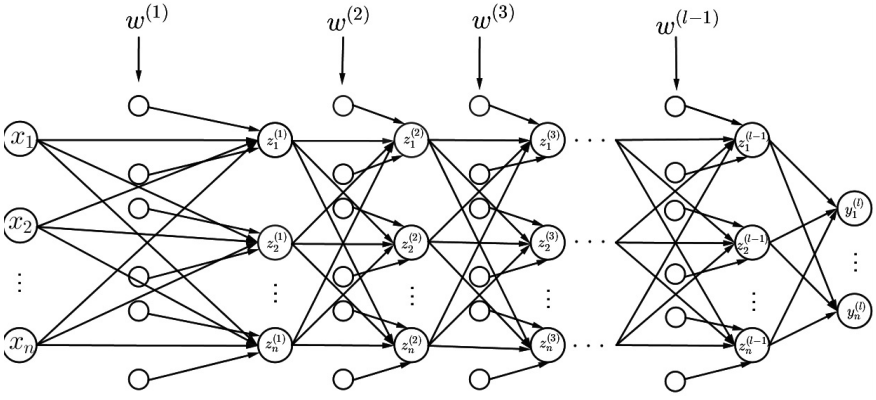


Figura 4.9: Representación gráfica de un perceptrón con una capa de entrada $\{x_1, x_2, \ldots, x_n\}$, l capas ocultas $\{z_1^{(1)}, z_2^{(1)}, \ldots, z_n^{(1)}\}$, $\{z_1^{(2)}, z_2^{(2)}, \ldots, z_n^{(2)}\}$, $\ldots$, $\{z_1^{(l-1)}, z_2^{(l-1)}, \ldots, z_n^{(l-1)}\}$ y una capa de salida $\{y_1^{(l+1)}, y_2^{(l+1)}, \ldots, y_n^{(l+1)}\}$. Los pesos $w_{ik}^{(k)}$ son los que ponderan las conexiones de la capa $k-1$ a la capa k.

una recta de regresión lineal, la cual podemos expresar de la siguiente forma:

$$y = f(x) = wx \tag{4.8}$$

Esta ecuación la podemos ver como un modelo neuronal, si lo exponemos gráficamente (ver figura 4.10), tenemos una entrada de $x \in \mathbb{R}$ y una unidad de salida y, además de una conexión de pesos w, y la función que se está utilizando para la salida de esta simple red es la función de activación lineal, es decir, podemos definir como la salida de $y = a_1^{(1)}$. Tendríamos una red neuronal con una unidad de entrada y una unidad de salida.

La función de propagación para esta red está definida como:

$$y = f(z_1^{(1)}) = f(wx) = wx \tag{4.9}$$

Podemos deducir que el problema de regresión lineal se convierte en un modelo de red neuronal, donde en ésta se puede

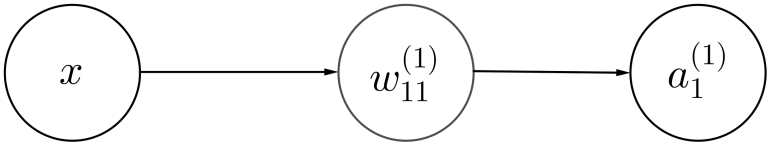


Figura 4.10: Un valor de entrada x que se conecta con $w_{11}^{(1)}$ como el peso correspondiente y $a_1^{(1)}$ es el resultado de la función de activación de x y $w_{11}^{(1)}$.

controlar el error de salida, mediante la función de costo:

$$J = \frac{1}{2}(d - a_1^{(1)})^2$$

Para minimizar estos errores tenemos que aplicar técnicas de derivación para encontrar su punto mínimo.

4.7. Algoritmo del descenso de gradiente

Para actualizar el valor del parámetro w, utilizaremos el algoritmo del descenso del gradiente. Nos apoyaremos en el gráfico para hacer los cálculos correspondientes.

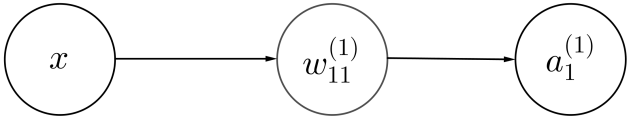


Figura 4.11: Un valor de entrada x que se conecta con $w_{11}^{(1)}$ como el peso correspondiente y $a_1^{(1)}$ es el resultado de la función de activación de x y $w_{11}^{(1)}$.

La función de costo J depende del parámetro w. La podemos

definir de la siguiente manera:

$$J(w) = \frac{1}{2}\left(d - a_1^{(1)}\right)^2$$

La figura 4.12 nos permite ver las dependencias de la función de costo.

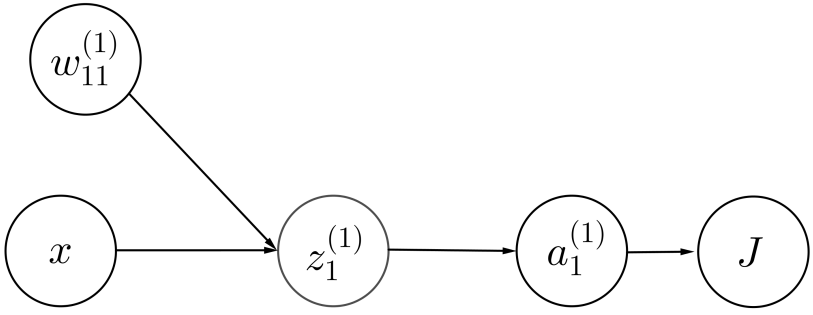


Figura 4.12: Dependencia de la función J del modelo neuronal.

Tenemos x como nodo de entrada, se conecta con una preactivación $z_1^{(1)}$, y a esta preactivación se pondera con el peso $w_{11}^{(1)}$, es decir, la preactivación lo que hace es efectuar el producto de x con $w_{11}^{(1)}$, pasa por una función de activación $a_1^{(1)}$ y finalmente se conecta con la función de costo $J(w)$.

Necesitamos calcular la derivada $J(w)$ de la función de costo aplicando la regla de la cadena. En la figura 4.13 podemos visualizar otro esquema. Para calcular la derivada de $J(w)$, necesitamos recorrer de adelante hacia atrás la trayectoria como lo muestra el dibujo. Para calcular la derivada de $\partial J/\partial w_{11}^{(1)}$ necesitamos calcular la derivada de $\partial J/\partial a_1^{(1)}$, y como la activación $a_1^{(1)}$ depende de la preactivación $z_1^{(1)}$, entonces calculamos la derivada parcial $\partial a_1^{(1)}/\partial z_1^{(1)}$, y de la preactivación $z_1^{(1)}$ vemos que depende nuevamente del parámetro $w_{11}^{(1)}$ para así calcular la derivada parcial $\partial z_1^{(1)}/\partial w_{11}^{(1)}$. Finalmente para calcular la derivada de $J(w)$ multiplicamos de adelante hacia atrás las derivadas parciales.

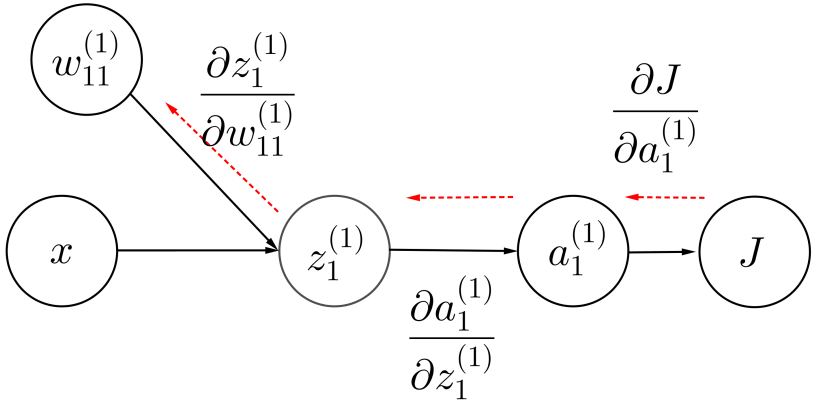


Figura 4.13: Modelo de neurona para calcular la derivada parcial de $J(w)$

$$\frac{\partial J}{\partial w_{11}^{(1)}} = \frac{\partial J}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} \tag{4.10}$$

$$\begin{aligned} \frac{\partial J}{\partial a_1^{(1)}} &= \frac{\partial J}{\partial a_1^{(1)}} \left(\frac{1}{2}(d - a_1^{(1)})^2 \right) = \frac{1}{2} \frac{\partial}{\partial a_1^{(1)}} \left((d - a_1^{(1)})^2 \right) \\ &= (d - a_1^{(1)}) \cdot \frac{d}{d(a_1^{(1)})}(d - a_1^{(1)}) \\ &= -(d - a_1^{(1)}) \end{aligned}$$

$$\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} = \frac{\partial z_1^{(1)}}{\partial z_1^{(1)}} = 1$$

$$\begin{aligned} \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} &= \frac{\partial}{\partial w_{11}^{(1)}}(w_{11}^{(1)} x) = x \frac{d}{d(w_{11}^{(1)})}(w_{11}^{(1)}) \\ \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} &= x \end{aligned}$$

Sustituyendo los resultados de las derivadas parciales tenemos

$$\begin{aligned} \frac{\partial J}{\partial w_{11}^{(1)}} &= -(d - a_1^{(1)})(1)(x) \\ &= -(d - a_1^{(1)})(x) \end{aligned} \tag{4.11}$$

Definamos el error $\hat{e} = d - a_1^{(1)}$ y tendremos la derivada de nuestra función de costo como:

$$\frac{\partial J}{\partial w_{11}^{(1)}} = \hat{e}_1^{(1)} x \tag{4.12}$$

4.8. Regla de actualización de pesos

Consideremos nuestra función de costo

$$J(w_{11}^{(1)}) = \frac{1}{2}(d - a_1^{(1)})^2$$

donde $a_1^{(1)} = w_{11}^{(1)} x$. Entonces podemos definir

$$J(w_{11}^{(1)}) = \frac{1}{2}(d - w_{11}^{(1)} x)^2$$

La función de costo será cero cuando:

$$d - w_{11}^{(1)} x = 0, \implies w_{11}^{(1)} = \frac{d}{x} \tag{4.13}$$

Si igualamos la derivada parcial igual a cero, concluimos que el mínimo de esta función ocurre en $w_{11}^{(1)} = d/x$. Como estamos tratando de un modelo aproximado de la regresión, generalmente no vamos a alcanzar de manera exacta este mínimo, por lo tanto, para solucionar este problema se requiere un esquema númerico en donde éste trate de aproximarse al punto óptimo. ¿Cómo realizamos este proceso numérico?

Se propone tener un punto inicial con un peso w^* arbitrario y a medida de que este vaya iterando, el peso w^* se irá acercando al punto mínimo.

Definición 17 *Regla de actualización de pesos*

$$w_{11}^{(1)} = w, \implies w^{(t+1)} = w^{(t)} - \eta \bigtriangledown J(w)$$

$$\begin{aligned} \bigtriangledown J(w) &= \frac{\partial J}{\partial w} = -(d - a_1^{(1)})x \\ w^{(t+1)} &= w^{(t)} + \eta(d - a_1^{(1)})x \\ w^{(t+1)} &= w^{(t)} + \eta(\hat{e}_1^{(1)})x \end{aligned} \tag{4.14}$$

La tasa de aprendizaje η determina cuánto cambia el peso. Si este valor es demasiado alto, la salida deambula por la solución y no logra converger. Por el contrario, si es demasiado bajo, el cálculo llega a la solución lentamente, donde la tasa de aprendizaje η puede tomar valores $0 < \eta \leq 1$.

Sea $D = \{(x_1, d_1), ..., (x_n, d_n)\}$ un conjunto de puntos donde $(x, d) \in D$. El algoritmo del descenso de gradiente para la actualización de pesos es el siguiente:

1. Tomamos un punto (x_i, d_i)

2. Propagamos la red a la función de activacion $(a_1^{(1)})^{(i)}$

3. Calculamos el error de la salida

$$(\hat{e}_i^{(1)})^{(i)} = d_i - (a_1^{(1)})^{(i)}$$

4. Calculamos la actualización de peso $\Delta w^{(t)}$

$$\Delta w^{(t)} = \eta(\hat{e}_i^{(1)})^{(i)} x_i$$

5. Actualizamos el nuevo peso w

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)}$$

4.9. Propagación hacia atrás

Sentemos los parámetros que vamos a usar con la notación correspondiente, apoyándonos de las figuras de cada red de neuronas a partir de un ejemplo para después obtener la ecuación general de la red de retropropagación.

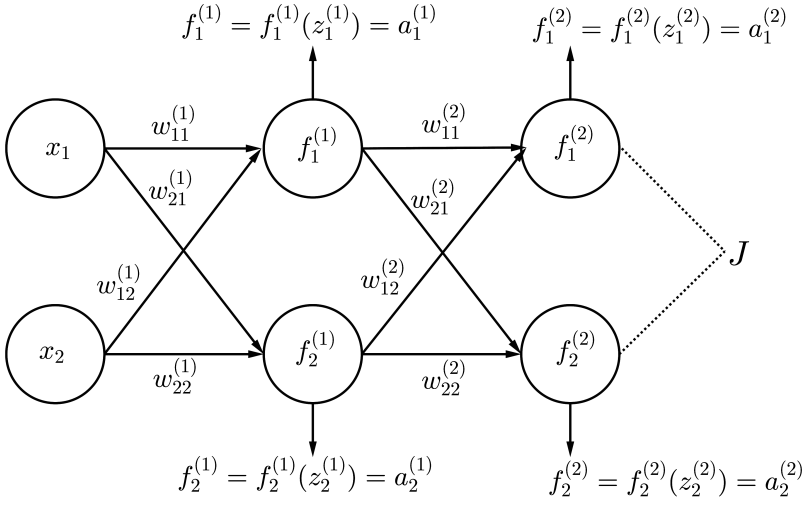


Figura 4.14: Gráfico de un modelo neuronal que depende de $J(W^{(1)}, W^{(2)})$.

Apoyándonos de la figura 4.14, definamos los parámetros como x_{ij} los valores de entrada, w_{ij} los pesos, $z_i^{(n)}$ como la preactivación, $a_i^{(n)}$ la función de activación. La función de costo que corresponde a la figura 4.14, depende de todos los parámetros de matriz de peso $W^{(1)}$ y $W^{(2)}$.

$$J(W^{(2)}, W^{(1)}) = \frac{1}{2}\sum_{k=1}^{2}(d_k - a_k^{(2)})^2$$

Esta es una función que depende de muchos parámetros, entonces calcular las derivadas parciales con respecto a cada uno

de estos parámetros es un procedimiento extenso, pero con los gráficos que hemos estado trabajando han sido muy útiles para aterrizar y escribir las ecuaciones que hemos calculado, además proporciona al lector accesibilidad al comportamiento y a las trayectorias que sigue cada ecuación.

Realizamos los cálculos para la propagación de la figura 4.14.

$$\begin{aligned} z_1^{(1)} &= w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 \rightarrow f_1^{(1)}(z_1^{(1)}) = a_1^{(1)} \\ z_2^{(1)} &= w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 \rightarrow f_2^{(1)}(z_2^{(1)}) = a_2^{(1)} \\ z_1^{(2)} &= w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} \rightarrow f_1^{(2)}(z_1^{(2)}) = a_1^{(2)} \\ z_2^{(2)} &= w_{21}^{(2)} a_1^{(1)} + w_{22}^{(2)} a_2^{(1)} \rightarrow f_2^{(2)}(z_2^{(2)}) = a_2^{(2)} \end{aligned}$$

Calculamos la función de costo extendida, donde vemos la dependencia de J con respecto a las activaciones de $a_1^{(2)}$ y $a_2^{(2)}$

$$J = \frac{1}{2}(d_1 - a_1^{(2)})^2 + \frac{1}{2}(d_2 - a_2^{(2)})^2 \quad (4.15)$$

$$J(W^{(2)}, W^{(1)}) = \frac{1}{2}\sum_{k=1}^{2}(d_k - a_k^{(2)})^2 \quad (4.16)$$

En la figura 4.15 podemos visualizar una red mucho más compacta, en donde se visualice el proceso para calcular las ecuaciones necesarias. Definamos como *trayectoria* aquel recorrido para un parámetro que se tenga que calcular, depediendo el nodo que le corresponda.

En figura 4.16 podemos observar todos los parámetros y la dependencia que tiene cada uno de los nodos, así como también la trayectoria que se tenga que recorrer para calcular la derivadas parciales correspondientes para cada peso w.

4.10. Actualización de pesos $\mathbf{W}^{(2)}$

La región que está sombreada de la figura 4.17 , será la primera trayectoria que se va a trabajar, en donde en cada dire-

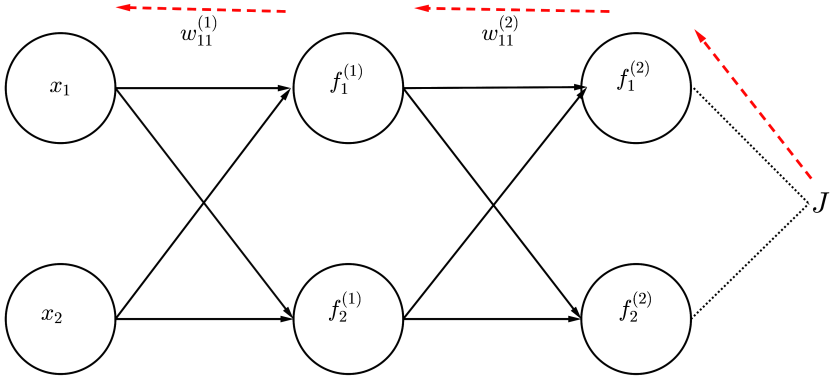


Figura 4.15: Modelo de red simplificado con trayectoria hacia atrás.

cción de la flecha se agregan los valores η y las derivadas parciales correspondientes. Para esta parte necesitamos calcular la derivada parcial de $\partial J/\partial w_{11}^{(2)}$, multiplicando las derivadas parciales de adelante hacia atrás.

$$\frac{\partial J}{\partial w_{11}^{(2)}} = \frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

Calculamos las derivadas parciales correspondientes:

$$\begin{aligned} \frac{\partial J}{\partial a_1^{(2)}} &= \frac{\partial}{\partial a_1^{(2)}} \left[\frac{1}{2} \left(d_1 - a_1^{(2)} \right)^2 + \frac{1}{2} \left(d_2 - a_2^{(2)} \right)^2 \right] \\ &= \frac{1}{2} \frac{\partial}{\partial a_1^{(2)}} \left(d_1 - a_1^{(2)} \right)^2 = 2(\frac{1}{2})(d_1 - a_1^{(2)}) \\ &= -\left(d_1 - a_1^{(2)} \right) \end{aligned}$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = \frac{\partial}{\partial z_1^{(2)}} f(z_1^{(2)}) = f'(z_1^{(2)})$$

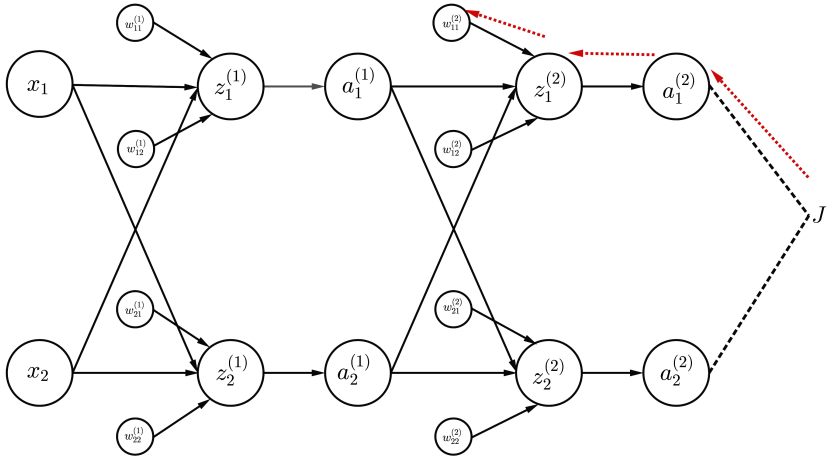


Figura 4.16: Modelo de red extendida con todos los parámetros correspondientes desde los valores de entrada hasta la salida.

$$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = \frac{\partial}{\partial w_{11}^{(2)}}(w_{11}^{(2)} \cdot a_1^{(1)} + w_{12}^{(2)} \cdot a_2^{(1)}) = \frac{\partial}{\partial w_{11}^{(2)}}(w_{11}^{(2)} \cdot a_1^{(1)})$$

$$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = a_1^{(1)}$$

El resultado de la derivada parcial de $\partial J/\partial w_{11}^{(2)}$ es:

$$\frac{\partial J}{\partial w_{11}^{(2)}} = -\left(d_1 - a_1^{(2)}\right) \cdot f'(z_1^{(2)}) \cdot a_1^{(1)}$$

podemos mejorar esta ecuación haciendo algunos ajustes, definamos algunas ecuaciones;

$$\hat{e}_1^{(2)} = d_1 - a_1^{(2)}, \quad \delta_1^{(2)} = f'(z_1^{(2)}) \cdot \hat{e}_1^{(2)}$$

Así, de forma más compacta el resultado de la derivada $\partial J/\partial w_{11}^{(2)}$ es:

$$\frac{\partial J}{\partial w_{11}^{(2)}} = -\delta_1^{(2)} \cdot a_1^{(1)} \tag{4.17}$$

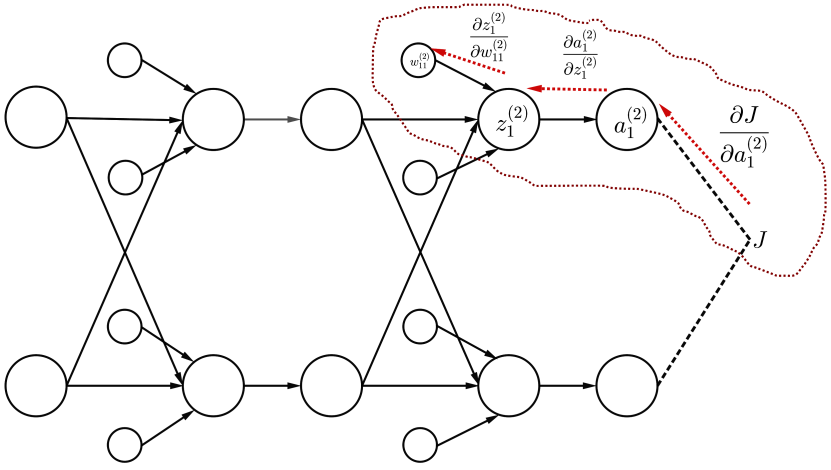


Figura 4.17: Modelo de red extendida con una trayectoria hacia el nodo de $w_{11}^{(2)}$

Para actualizar los pesos de cada iteración hacemos lo siguiente

$$\begin{aligned} w_{11}^{(2)} &\leftarrow w_{11}^{(2)} - \eta \frac{\partial J}{\partial w_{11}^{(2)}} \\ w_{11}^{(2)} &\leftarrow w_{11}^{(2)} + \eta \delta_1^{(2)} a_1^{(1)} \end{aligned} \tag{4.18}$$

En la figura 4.18 de manera análoga vamos a seguir la trayectoria para el nodo $w_{12}^{(2)}$, para calcular sus derivadas correspondientes:

$$\frac{\partial J}{\partial w_{12}^{(2)}} = \frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}}$$

$$\begin{aligned} \frac{\partial J}{\partial a_1^{(2)}} &= \frac{\partial}{\partial a_1^{(2)}} \left[\frac{1}{2} \left(d_1 - a_1^{(2)} \right)^2 + \frac{1}{2} \left(d_2 - a_2^{(2)} \right)^2 \right] \\ &= \frac{1}{2} \frac{\partial}{\partial a_1^{(2)}} \left(d_1 - a_1^{(2)} \right)^2 = 2(\frac{1}{2})(d_1 - a_1^{(2)}) \\ &= -\left(d_1 - a_1^{(2)} \right) \end{aligned}$$

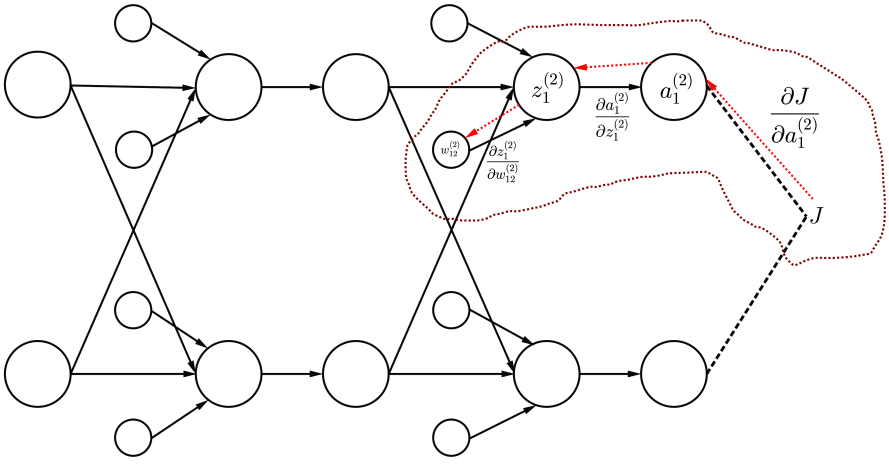


Figura 4.18: Modelo de red extendida con una trayectoria hacia el nodo de $w_{12}^{(2)}$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = \frac{\partial}{\partial z_1^{(2)}} f(z_1^{(2)}) = f'(z_1^{(2)})$$

$$\begin{aligned} \frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}} &= \frac{\partial}{\partial w_{12}^{(2)}}(w_{11}^{(2)} \cdot a_1^{(1)} + w_{12}^{(2)} \cdot a_2^{(1)}) = \frac{\partial}{\partial w_{12}^{(2)}}(w_{12}^{(2)} \cdot a_2^{(1)}) \\ \frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}} &= a_2^{(1)} \end{aligned}$$

$$\frac{\partial J}{\partial w_{12}^{(2)}} = -\left(d_1 - a_1^{(2)}\right) \cdot f'(z_1^{(2)}) \cdot a_2^{(1)}$$

$$\hat{e}_1^{(2)} = d_1 - a_1^{(2)}, \quad \delta_1^{(2)} = f'(z_1^{(2)}) \cdot \hat{e}_1^{(2)}$$

El resultado de la derivada parcial de $\partial J / \partial w_{12}^{(2)}$, tenemos:

$$\frac{\partial J}{\partial w_{12}^{(2)}} = -\delta_1^{(2)} \cdot a_2^{(1)} \tag{4.19}$$

Para actualizar los pesos de cada iteración, tenemos lo siguiente

$$\begin{aligned} w_{12}^{(2)} &\leftarrow w_{12}^{(2)} - \eta \frac{\partial J}{\partial w_{12}^{(2)}} \\ w_{12}^{(2)} &\leftarrow w_{12}^{(2)} + \eta \delta_1^{(2)} a_2^{(1)} \end{aligned} \tag{4.20}$$

Ahora, en la figura 4.19 seguimos la trayectoria para el nodo $w_{21}^{(2)}$ y continuar el mismo procedemiento como en los nodos anteriores.

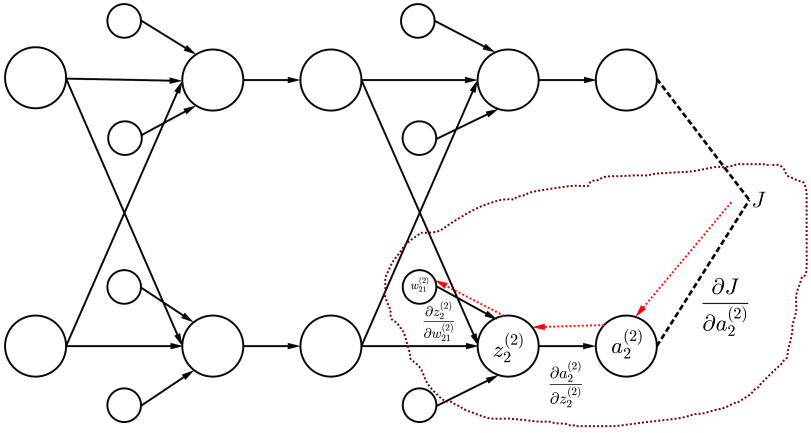


Figura 4.19: Modelo de red extendida con una trayectoria hacia el nodo de $w_{21}^{(2)}$

$$\frac{\partial J}{\partial w_{21}^{(2)}} = \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial w_{21}^{(2)}}$$

$$\begin{aligned} \frac{\partial J}{\partial a_2^{(2)}} &= \frac{\partial}{\partial a_2^{(2)}} \left[\frac{1}{2} \left(d_1 - a_1^{(2)} \right)^2 + \frac{1}{2} \left(d_2 - a_2^{(2)} \right)^2 \right] \\ &= \frac{1}{2} \frac{\partial}{\partial a_2^{(2)}} \left(d_2 - a_2^{(2)} \right)^2 = 2(\frac{1}{2})(d_2 - a_2^{(2)}) \\ &= -\left(d_2 - a_2^{(2)} \right) \end{aligned}$$

$$\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = \frac{\partial}{\partial z_2^{(2)}} f(z_2^{(2)}) = f'(z_2^{(2)})$$

$$\begin{aligned} \frac{\partial z_2^{(2)}}{\partial w_{21}^{(2)}} &= \frac{\partial}{\partial w_{21}^{(2)}}(w_{21}^{(2)} \cdot a_1^{(1)} + w_{22}^{(2)} \cdot a_2^{(1)}) \\ &= \frac{\partial}{\partial w_{21}^{(2)}}(w_{21}^{(2)} \cdot a_1^{(1)}) \\ &= a_1^{(1)} \end{aligned}$$

$$\frac{\partial J}{\partial w_{21}^{(2)}} = -\left(d_2 - a_2^{(2)}\right) \cdot f'(z_2^{(2)}) \cdot a_1^{(1)}$$

$$\hat{e}_2^{(2)} = d_2 - a_2^{(2)}, \quad \delta_2^{(2)} = f'(z_2^{(2)}) \cdot \hat{e}_2^{(2)}$$

Así, como resultado de la derivada parcial de $\partial J/\partial w_{21}^{(2)}$, tenemos:

$$\frac{\partial J}{\partial w_{21}^{(2)}} = -\delta_2^{(2)} \cdot a_1^{(1)} \tag{4.21}$$

Actualizamos los pesos de cada itereación para $w_{21}^{(2)}$:

$$\begin{aligned} w_{21}^{(2)} &\leftarrow w_{21}^{(2)} - \eta \frac{\partial J}{\partial w_{21}^{(2)}} \\ w_{21}^{(2)} &\leftarrow w_{21}^{(2)} + \eta \delta_2^{(2)} a_1^{(1)} \end{aligned} \tag{4.22}$$

Tenemos la figura 4.20, seguimos la trayectoria para el nodo $w_{22}^{(2)}$ y calcular sus derivadas correspondientes:

$$\frac{\partial J}{\partial w_{22}^{(2)}} = \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial w_{22}^{(2)}}$$

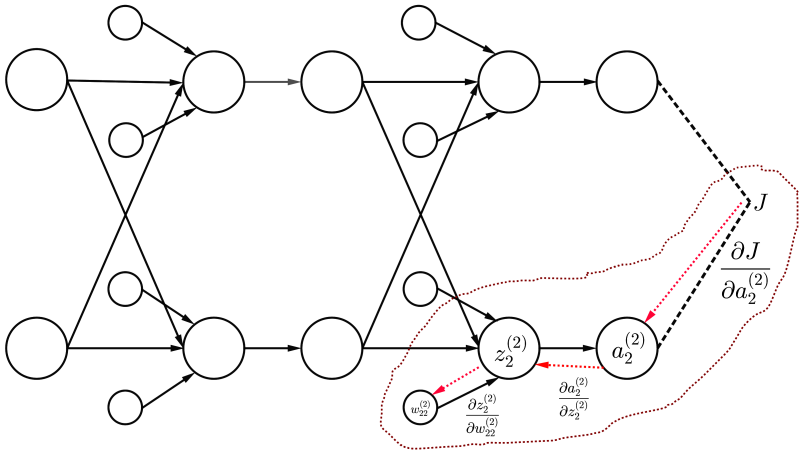


Figura 4.20: Modelo de red extendida con una trayectoria hacia el nodo de $w_{22}^{(2)}$

$$\begin{aligned}\frac{\partial J}{\partial a_2^{(2)}} &= \frac{\partial}{\partial a_2^{(2)}}\left[\frac{1}{2}\left(d_1 - a_1^{(2)}\right)^2 + \frac{1}{2}\left(d_2 - a_2^{(2)}\right)^2\right] \\ &= \frac{1}{2}\frac{\partial}{\partial a_2^{(2)}}\left(d_2 - a_2^{(2)}\right)^2 = 2(\frac{1}{2})(d_2 - a_2^{(2)}) \\ &= -\left(d_2 - a_2^{(2)}\right)\end{aligned}$$

$$\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = \frac{\partial}{\partial z_2^{(2)}} f(z_2^{(2)}) = f'(z_2^{(2)})$$

$$\begin{aligned}\frac{\partial z_2^{(2)}}{\partial w_{22}^{(2)}} &= \frac{\partial}{\partial w_{22}^{(2)}}(w_{21}^{(2)} \cdot a_1^{(1)} + w_{22}^{(2)} \cdot a_2^{(1)}) \\ &= \frac{\partial}{\partial w_{22}^{(2)}}(w_{22}^{(2)} \cdot a_2^{(1)}) \\ &= a_2^{(1)}\end{aligned}$$

$$\frac{\partial J}{\partial w_{22}^{(2)}} = -\left(d_2 - a_2^{(2)}\right) \cdot f'(z_2^{(2)}) \cdot a_2^{(1)}$$

$$\hat{e}_2^{(2)} = d_2 - a_2^{(2)}, \quad \delta_2^{(2)} = f'(z_2^{(2)}) \cdot \hat{e}_2^{(2)}$$

La derivada parcial de $\partial J/\partial w_{22}^{(2)}$ tenemos:

$$\frac{\partial J}{\partial w_{22}^{(2)}} = -\delta_2^{(2)} \cdot a_2^{(1)} \tag{4.23}$$

Para actualizar los pesos de cada iteración de $w_{22}^{(2)}$, tenemos lo siguiente

$$\begin{aligned} w_{22}^{(2)} &\leftarrow w_{22}^{(2)} - \eta \frac{\partial J}{\partial w_{22}^{(2)}} \\ w_{22}^{(2)} &\leftarrow w_{22}^{(2)} + \eta \delta_2^{(2)} a_2^{(1)} \end{aligned} \tag{4.24}$$

Hemos podido actualizar los pesos asociados a la matriz de $W^{(1)}$, en donde se reducen simplemente en cuatro ecuaciones.

$$\begin{aligned} &1)\ w_{11}^{(2)} \leftarrow w_{11}^{(2)} + \eta \delta_1^{(2)} a_1^{(1)}, \quad 3)\ w_{22}^{(2)} \leftarrow w_{22}^{(2)} + \eta \delta_2^{(2)} a_1^{(1)} \\ &2)\ w_{12}^{(2)} \leftarrow w_{12}^{(2)} + \eta \delta_1^{(2)} a_2^{(1)}, \quad 4)\ w_{22}^{(2)} \leftarrow w_{22}^{(2)} + \eta \delta_2^{(2)} a_2^{(1)} \end{aligned} \tag{4.25}$$

Al observar estas ecuaciones nos damos cuenta que tienen similitudes y que nos ayudará mas adelante para deducir la ecuación final de la red de retropropagación.

4.11. Actualización de pesos $\mathbf{W}^{(1)}$

En la figura 4.21 tenemos una nueva trayectoria, al observar como hace el recorrido, los cálculos se hacen cada vez mas extensos, seguimos la trayectoria para el nodo $w_{11}^{(1)}$.

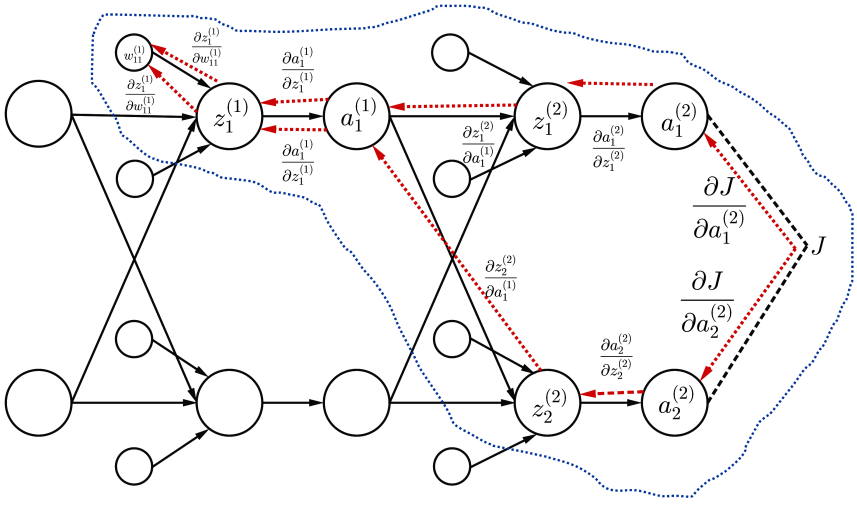


Figura 4.21: Modelo de red extendida con una trayectoria hacia el nodo de $w_{11}^{(1)}$

$$\frac{\partial J}{\partial w_{11}^{(1)}} = \frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} + \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}}$$

simplificando esta ecuación, tenemos lo siguiente

$$\frac{\partial J}{\partial w_{11}^{(1)}} = \left[\frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} + \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} \right] \cdot \left(\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} \right)$$

Analizando las derivadas parciales que se han generado, hay algunas que se han resuelto a partir de los nodos de $W^{(1)}$, esto

ayuda resolver las derivadas parciales restantes:

$$\frac{\partial J}{\partial a_1^{(2)}} = -\left(d_1 - a_1^{(2)}\right), \quad \frac{\partial J}{\partial a_2^{(2)}} = -\left(d_2 - a_2^{(2)}\right)$$
$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = f'\left(z_1^{(2)}\right), \qquad \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = f'\left(z_2^{(2)}\right)$$

calculamos la derivadas parciales restantes

$$\begin{aligned}
\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} &= \frac{\partial}{\partial a_1^{(1)}}\left(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)}\right) = w_{11}^{(2)} \\
\frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} &= \frac{\partial}{\partial a_1^{(1)}}\left(w_{21}^{(2)} a_1^{(1)} + w_{22}^{(2)} a_2^{(1)}\right) = w_{21}^{(2)} \\
\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} &= \frac{\partial}{\partial z_1^{(1)}} f(z_1^{(1)}) = f'(z_1^{(1)}) \\
\frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} &= \frac{\partial}{\partial w_{11}^{(1)}}\left(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2\right) = x_1
\end{aligned}$$

Si sustituimos todos los resultados de la derivada parcial de $\partial J / \partial w_{11}^{(1)}$:

$$\frac{\partial J}{\partial w_{11}^{(1)}} = \left[-\left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right) \cdot w_{11}^{(2)} \ldots \right.$$
$$\left. -\left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right) \cdot w_{21}^{(2)}\right] \cdot \left(f'\left(z_1^{(2)}\right) \cdot x_1\right)$$

Definimos

$$\delta_1^{(2)} = \left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right), \quad \delta_2^{(2)} = \left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right)$$

y tenemos

$$\frac{\partial J}{\partial w_{11}^{(1)}} = -\left[\delta_1^{(2)} \cdot w_{11}^{(2)} + \delta_2^{(2)} \cdot w_{21}^{(2)}\right] \cdot \left(f'\left(z_1^{(2)}\right) \cdot x_1\right)$$

Definimos nuevamente

$$\hat{e}_1^{(1)} = \delta_1^{(1)} \cdot w_{11}^{(2)} + \delta_1^{(2)} \cdot w_{21}^{(2)}, \quad \delta_1^{(1)} = \hat{e}_1^{(1)} \cdot f'\left(z_1^{(2)}\right)$$

Al simplificar $\partial J/\partial w_{11}^{(1)}$ tenemos:

$$\frac{\partial J}{\partial w_{11}^{(1)}} = -\delta_1^{(1)} \cdot x_1 \tag{4.26}$$

Para actualizar los pesos para cada iteración efectuamos lo siguiente:

$$\begin{aligned} w_{11}^{(1)} &\leftarrow w_{11}^{(1)} - \eta \frac{\partial J}{\partial w_{11}^{(1)}} \\ w_{11}^{(1)} &\leftarrow w_{11}^{(1)} + \eta \delta_1^{(1)} \cdot x_1 \end{aligned} \tag{4.27}$$

En la figura 4.22 de forma análoga tenemos una nueva trayectoria, seguimos la trayectoria para el nodo $w_{12}^{(1)}$ y calculamos sus derivadas parciales:

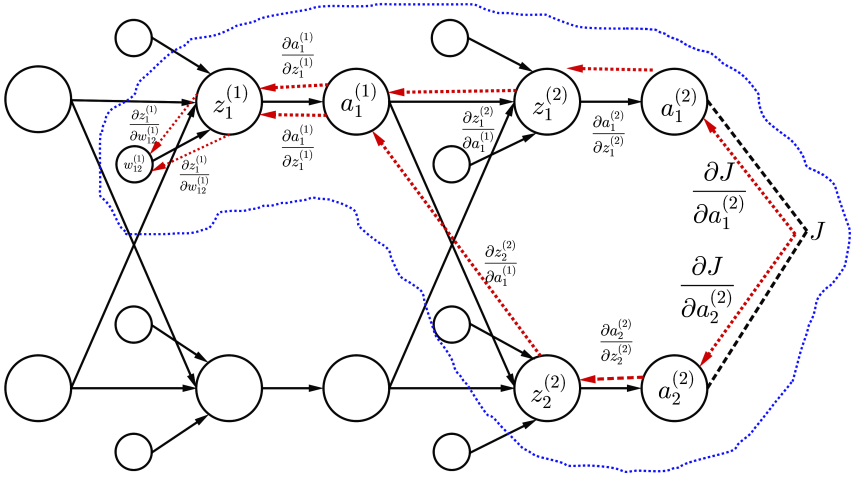


Figura 4.22: Modelo de red extendida con una trayectoria hacia el nodo de $w_{12}^{(1)}$

Ahora, hagamos lo cálculos para $w_{12}^{(1)}$

$$\begin{aligned}\frac{\partial J}{\partial w_{12}^{(1)}} &= \frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{12}^{(1)}} \\ &+ \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{12}^{(1)}}\end{aligned}$$

Simplificando la ecuación tenemos;

$$\begin{aligned}\frac{\partial J}{\partial w_{12}^{(1)}} &= \left[\frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} + \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}}\right] \\ &\cdot \left(\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \cdot \frac{\partial z_1^{(1)}}{\partial w_{12}^{(1)}}\right)\end{aligned}$$

Calculamos las derivadas parciales de $w_{12}^{(1)}$:

$$\begin{aligned}\tfrac{\partial J}{\partial a_1^{(2)}} &= -\left(d_1 - a_1^{(2)}\right), \quad \tfrac{\partial J}{\partial a_2^{(2)}} = -\left(d_2 - a_1^{(2)}\right) \\ \tfrac{\partial a_1^{(2)}}{\partial z_1^{(2)}} &= f'\left(z_1^{(2)}\right), \qquad \tfrac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = f'\left(z_2^{(2)}\right)\end{aligned}$$

siguiendo la misma idea al calcular las derivadas parciales restantes en la trayectoria en el nodo $w_{11}^{(1)}$, pero ahora para el nodo $w_{12}^{(1)}$ tenemos;

$$\begin{aligned}\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} &= \frac{\partial}{\partial a_1^{(1)}}\left(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)}\right) = w_{11}^{(2)} \\ \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} &= \frac{\partial}{\partial a_1^{(1)}}\left(w_{21}^{(2)} a_1^{(1)} + w_{22}^{(2)} a_2^{(1)}\right) = w_{21}^{(2)} \\ \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} &= \frac{\partial}{\partial z_1^{(1)}} f(z_1^{(1)}) = f'(z_1^{(1)}) \\ \frac{\partial z_1^{(1)}}{\partial w_{12}^{(1)}} &= \frac{\partial}{\partial w_{12}^{(1)}}\left(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2\right) = x_2\end{aligned}$$

Si sustituimos todos los resultados de la derivada parcial de $\partial J/\partial w_{12}^{(1)}$

$$\frac{\partial J}{\partial w_{11}^{(1)}} = \left[-\left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right) \cdot w_{11}^{(2)} \ldots \right.$$
$$\left. -\left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right) \cdot w_{21}^{(2)} \right] \cdot \left(f'\left(z_1^{(1)}\right) \cdot x_2\right)]$$

Definimos

$$\delta_1^{(2)} = \left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right), \quad \delta_2^{(2)} = \left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right)$$

y tenemos

$$\frac{\partial J}{\partial w_{12}^{(1)}} = -\left[\delta_1^{(2)} \cdot w_{11}^{(2)} + \delta_2^{(2)} \cdot w_{21}^{(2)}\right] \cdot \left(f'\left(z_1^{(1)}\right) \cdot x_2\right)$$

Definimos nuevamente

$$\hat{e}_2^{(1)} = \delta_1^{(1)} \cdot w_{11}^{(2)} + \delta_1^{(2)} \cdot w_{21}^{(2)}, \quad \delta_1^{(1)} = \hat{e}_2^{(1)} \cdot f'\left(z_1^{(1)}\right)$$

Al simplificar tenemos la derivada $\partial J/\partial w_{12}^{(1)}$, tenemos:

$$\frac{\partial J}{\partial w_{12}^{(1)}} = -\delta_1^{(1)} \cdot x_2$$

Para actualizar los pesos de $w_{12}^{(1)}$ cada iteración tenemos lo siguiente:

$$\begin{aligned} w_{12}^{(1)} &\leftarrow w_{12}^{(1)} - \eta \frac{\partial J}{\partial w_{12}^{(1)}} \\ w_{12}^{(1)} &\leftarrow w_{12}^{(1)} + \eta \delta_1^{(1)} \cdot x_2 \end{aligned} \tag{4.28}$$

Para la figura 4.23 tenemos la trayectoria para el nodo $w_{21}^{(1)}$ para calcular sus derivadas parciales:

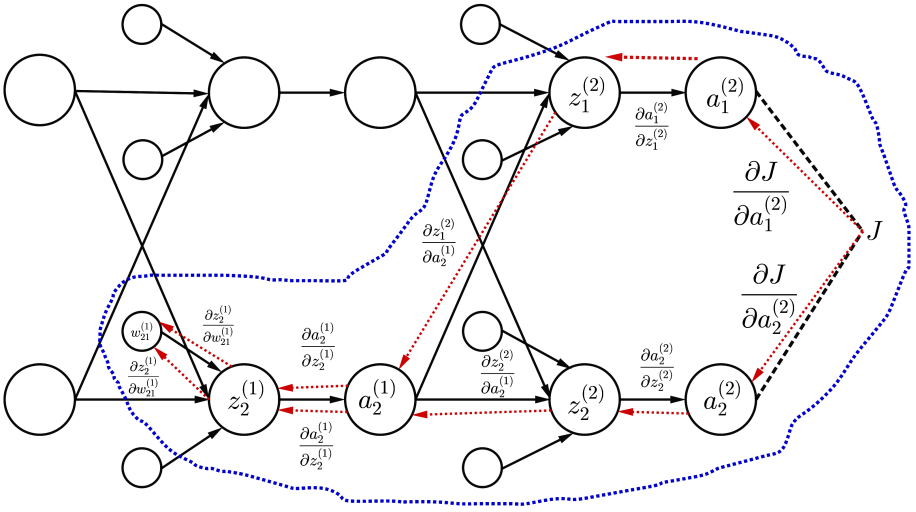


Figura 4.23: Modelo de red extendida con una trayectoria hacia el nodo de $w_{21}^{(1)}$

Ahora, hagamos los cálculos para $w_{21}^{(1)}$

$$\begin{aligned}\frac{\partial J}{\partial w_{21}^{(1)}} &= \frac{\partial J}{\partial a_1^{(2)}}\cdot\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\cdot\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}}\cdot\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}\cdot\frac{\partial z_2^{(1)}}{\partial w_{21}^{(1)}}\cdots \\ &+\frac{\partial J}{\partial a_2^{(2)}}\cdot\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\cdot\frac{\partial z_2^{(2)}}{\partial a_2^{(1)}}\cdot\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}}\cdot\frac{\partial z_2^{(1)}}{\partial w_{21}^{(1)}}\end{aligned}$$

simplificando la ecuación

$$\begin{aligned}\frac{\partial J}{\partial w_{21}^{(1)}} &= \left[\frac{\partial J}{\partial a_1^{(2)}}\cdot\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}}\cdot\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}}+\frac{\partial J}{\partial a_2^{(2)}}\cdot\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}}\cdot\frac{\partial z_2^{(2)}}{\partial a_1^{(1)}}\right] \\ &\cdot\left(\frac{\partial a_2^{(1)}}{\partial z_1^{(1)}}\cdot\frac{\partial z_2^{(1)}}{\partial w_{21}^{(1)}}\right)\end{aligned}$$

Analizando las derivadas parciales que se han generado de los nodos anteriores, hay algunas que se han resuelto, esto ayuda a

encontrar las derivadas parciales restantes:

$$\frac{\partial J}{\partial a_1^{(2)}} = -\left(d_1 - a_1^{(2)}\right), \quad \frac{\partial J}{\partial a_2^{(2)}} = -\left(d_2 - a_2^{(2)}\right)$$
$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = f'\left(z_1^{(2)}\right), \qquad \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = f'\left(z_2^{(2)}\right)$$

Si calculamos las derivadas parciales restantes

$$\begin{aligned}
\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} &= \frac{\partial}{\partial a_2^{(1)}}\left(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)}\right) = w_{12}^{(2)} \\
\frac{\partial z_2^{(2)}}{\partial a_2^{(1)}} &= \frac{\partial}{\partial a_2^{(1)}}\left(w_{21}^{(2)} a_1^{(1)} + w_{22}^{(2)} a_2^{(1)}\right) = w_{22}^{(2)} \\
\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} &= \frac{\partial}{\partial z_2^{(1)}} f(z_2^{(1)}) = f'(z_2^{(1)}) \\
\frac{\partial z_2^{(1)}}{\partial w_{21}^{(1)}} &= \frac{\partial}{\partial w_{21}^{(1)}}\left(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2\right) = x_1
\end{aligned}$$

Si sustituimos todos los resultados de la derivada parcial de $\partial J / \partial w_{21}^{(1)}$

$$\frac{\partial J}{\partial w_{21}^{(1)}} = \Big[-\left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right) \cdot w_{12}^{(2)} \ldots$$
$$-\left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right) \cdot w_{22}^{(2)}\Big] \cdot \left(f'\left(z_2^{(1)}\right) \cdot x_1\right)$$

Definimos

$$\delta_1^{(2)} = \left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right), \quad \delta_2^{(2)} = \left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right)$$

y tenemos

$$\frac{\partial J}{\partial w_{21}^{(1)}} = -\left[\delta_1^{(2)} \cdot w_{12}^{(2)} + \delta_2^{(2)} \cdot w_{22}^{(2)}\right] \cdot \left(f'\left(z_2^{(1)}\right) \cdot x_1\right)$$

Definimos nuevamente

$$\hat{e}_2^{(1)} = \delta_1^{(2)} \cdot w_{12}^{(2)} + \delta_2^{(2)} \cdot w_{22}^{(2)}, \quad \delta_2^{(1)} = \hat{e}_2^{(1)} \cdot f'\left(z_2^{(1)}\right)$$

Al simplificar tenemos $\partial J/\partial w_{21}^{(1)}$ tenemos:

$$\frac{\partial J}{\partial w_{21}^{(1)}} = -\delta_2^{(1)} \cdot x_1$$

Actualizamos los pesos de $w_{21}^{(1)}$ tenemos:

$$\begin{aligned} w_{21}^{(1)} &\leftarrow w_{21}^{(1)} - \eta \frac{\partial J}{\partial w_{21}^{(1)}} \\ w_{21}^{(1)} &\leftarrow w_{21}^{(1)} + \eta \delta_2^{(1)} \cdot x_1 \end{aligned} \tag{4.29}$$

Para 4.24 tenemos la última trayectoria, seguimos la trayectoria para el nodo $w_{22}^{(1)}$ y resolver sus derivadas parciales:

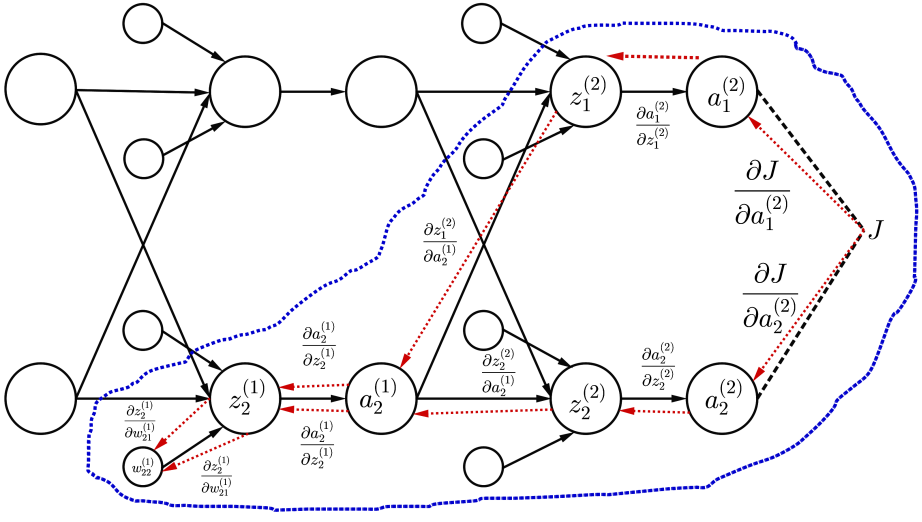


Figura 4.24: Modelo de red extendida con una trayectoria hacia el nodo de $w_{22}^{(1)}$

Ahora, hagamos los cálculos para $w_{22}^{(1)}$:

$$\begin{aligned}\frac{\partial J}{\partial w_{22}^{(1)}} &= \frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} \cdot \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} \cdot \frac{\partial z_2^{(1)}}{\partial w_{22}^{(1)}} \cdots \\ &+ \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial a_2^{(1)}} \cdot \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} \cdot \frac{\partial z_2^{(1)}}{\partial w_{22}^{(1)}}\end{aligned}$$

Simplificando la ecuación

$$\begin{aligned}\frac{\partial J}{\partial w_{22}^{(1)}} &= \left[\frac{\partial J}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} + \frac{\partial J}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} \cdot \frac{\partial z_2^{(2)}}{\partial a_2^{(1)}}\right] \\ &\cdot \left(\frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} \cdot \frac{\partial z_2^{(1)}}{\partial w_{22}^{(1)}}\right)\end{aligned}$$

Analizando las derivadas parciales que se han generado, calculamos las derivadas de restantes para la trayectoria de $\partial w_{22}^{(1)}$:

$$\begin{aligned}\tfrac{\partial J}{\partial a_1^{(2)}} &= -\left(d_1 - a_1^{(2)}\right), \quad \tfrac{\partial J}{\partial a_2^{(2)}} = -\left(d_2 - a_2^{(2)}\right) \\ \tfrac{\partial a_1^{(2)}}{\partial z_1^{(2)}} &= f'\left(z_1^{(2)}\right), \qquad \tfrac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = f'\left(z_2^{(2)}\right)\end{aligned}$$

calculamos las derivadas parciales que hacen falta

$$\begin{aligned}\frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} &= \frac{\partial}{\partial a_2^{(1)}}\left(w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)}\right) = w_{12}^{(2)} \\ \frac{\partial z_2^{(2)}}{\partial a_2^{(1)}} &= \frac{\partial}{\partial a_2^{(1)}}\left(w_{21}^{(2)} a_1^{(1)} + w_{22}^{(2)} a_2^{(1)}\right) = w_{22}^{(2)} \\ \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} &= \frac{\partial}{\partial z_2^{(1)}} f(z_2^{(1)}) = f'(z_2^{(1)}) \\ \frac{\partial z_2^{(1)}}{\partial w_{22}^{(1)}} &= \frac{\partial}{\partial w_{22}^{(1)}}\left(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2\right) = x_2\end{aligned}$$

Si sustituimos todos los resultados de la derivada parcial de $\partial J/\partial w_{22}^{(1)}$:

$$\begin{aligned}\frac{\partial J}{\partial w_{22}^{(1)}} = \Big[&- \left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right) \cdot w_{12}^{(2)} ... \\ &- \left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right) \cdot w_{22}^{(2)}\Big] \cdot \left(f'\left(z_2^{(1)}\right) \cdot x_2\right)\end{aligned}$$

Definimos

$$\delta_1^{(2)} = \left(d_1 - a_1^{(2)}\right) \cdot f'\left(z_1^{(2)}\right), \quad \delta_2^{(2)} = \left(d_2 - a_2^{(2)}\right) \cdot f'\left(z_2^{(2)}\right)$$

y tenemos

$$\frac{\partial J}{\partial w_{22}^{(1)}} = -\left[\delta_1^{(2)} \cdot w_{12}^{(2)} + \delta_2^{(2)} \cdot w_{22}^{(2)}\right] \cdot \left(f'\left(z_2^{(1)}\right) \cdot x_2\right)$$

Definimos nuevamente

$$\hat{e}_2^{(1)} = \delta_1^{(2)} \cdot w_{12}^{(2)} + \delta_2^{(2)} \cdot w_{22}^{(2)}, \quad \delta_2^{(1)} = \hat{e}_2^{(1)} \cdot f'\left(z_2^{(1)}\right)$$

Al simplificar tenemos que la derivada parcial de $J/\partial w_{22}^{(1)}$ es:

$$\frac{\partial J}{\partial w_{22}^{(1)}} = -\delta_2^{(1)} \cdot x_2$$

Y finalizamos actualizando los pesos para $w_{22}^{(1)}$:

$$\begin{aligned} w_{22}^{(1)} &\leftarrow w_{22}^{(1)} - \eta \frac{\partial J}{\partial w_{22}^{(1)}} \\ w_{22}^{(1)} &\leftarrow w_{22}^{(1)} + \eta \delta_2^{(1)} \cdot x_2 \end{aligned} \tag{4.30}$$

4.12. Deduciendo el modelo

Al haber hecho todo los cálculos necesarios, resumimos nuestras ecuaciones de la siguiente manera:

Capa de entrada	Capa de salida
$w_{11}^{(1)} \leftarrow w_{11}^{(1)} + \eta\delta_1^{(1)}x_1$	$w_{11}^{(2)} \leftarrow w_{11}^{(2)} + \eta\delta_1^{(2)}a_1^{(1)}$
$w_{12}^{(1)} \leftarrow w_{12}^{(1)} + \eta\delta_1^{(1)}x_2$	$w_{12}^{(2)} \leftarrow w_{12}^{(2)} + \eta\delta_1^{(2)}a_2^{(1)}$
$w_{21}^{(1)} \leftarrow w_{21}^{(1)} + \eta\delta_2^{(1)}x_1$	$w_{21}^{(2)} \leftarrow w_{21}^{(2)} + \eta\delta_2^{(2)}a_1^{(1)}$
$w_{22}^{(1)} \leftarrow w_{22}^{(1)} + \eta\delta_2^{(1)}x_2$	$w_{22}^{(2)} \leftarrow w_{22}^{(2)} + \eta\delta_2^{(2)}a_2^{(1)}$

Estas 8 ecuaciones nos servirán en el siguiente paso para deducir el proceso de propagación hacia atrás. Apoyándonos en la siguiente grafica 4.25.

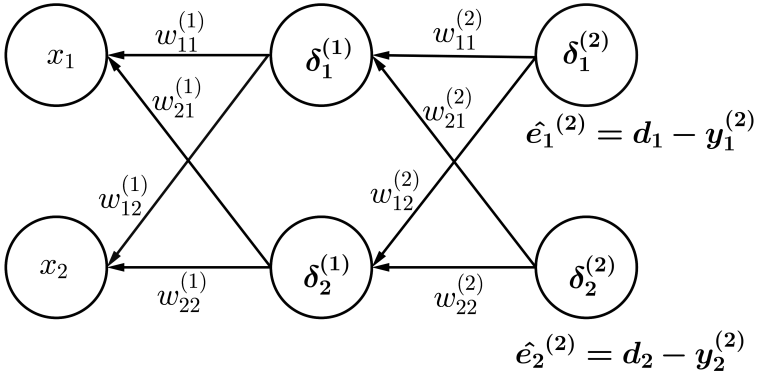


Figura 4.25: Propagación hacia atrás: actualización de pesos

Calculamos los errores de la salida asociada a las unidades de salida

$$\hat{e}_k^{(2)}, \quad \hat{e}^{(2)} = \begin{bmatrix} \hat{e}_1^{(2)} \\ \hat{e}_2^{(2)} \end{bmatrix} = \begin{bmatrix} d_1 - a_1^{(2)} \\ d_1 - a_2^{(2)} \end{bmatrix}$$

Calculamos δ asociados a las unidades

$$\delta_k^{(2)}, \quad \delta^{(2)} = \begin{bmatrix} \delta_1^{(2)} \\ \delta_2^{(2)} \end{bmatrix} = \begin{bmatrix} f'(z_1^{(2)})\hat{e}_1^{(2)} \\ f'(z_2^{(2)})\hat{e}_2^{(2)} \end{bmatrix}$$

Definición 18 *Sean A y B matrices $m \times n$ con entradas en C. El producto de Hadamard de A y B se define por $[A \circ B]_{ij} = [A]_{ij}\ [B]_{ij}$ para todos $1 \leq i \leq m$, $1 \leq j \leq n$.*

Usando el *producto de Hadamard*

$$\begin{aligned} \delta^{(2)} &= \begin{bmatrix} \delta_1^{(2)} \\ \delta_2^{(2)} \end{bmatrix} = \begin{bmatrix} f'(z_1^{(2)})\hat{e}_1^{(2)} \\ f'(z_2^{(2)})\hat{e}_2^{(2)} \end{bmatrix} \\ &= \begin{bmatrix} f'(z_1^{(2)}) \\ f'(z_2^{(2)}) \end{bmatrix} \circ \begin{bmatrix} \hat{e}_1^{(2)} \\ \hat{e}_2^{(2)} \end{bmatrix} \end{aligned}$$

$$\delta^{(2)} = f'(z^{(2)}) \circ \hat{e}^{(2)}$$

Necesitamos propagar $\delta^{(2)}$ hacia atrás para calcular la capa oculta

$$\begin{aligned} \hat{e}^{(1)} &= \begin{bmatrix} \hat{e}_1^{(2)} \\ \hat{e}_2^{(2)} \end{bmatrix} = \begin{bmatrix} \delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{21}^{(2)} \\ \delta_1^{(2)} w_{12}^{(2)} + \delta_2^{(2)} w_{22}^{(2)} \end{bmatrix} \\ &= \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} \\ w_{12}^{(2)} & w_{21}^{(2)} \end{bmatrix} \circ \begin{bmatrix} \delta_1^{(2)} \\ \delta_2^{(2)} \end{bmatrix} \end{aligned}$$

$$\hat{e}^{(1)} = (w^{(2)})^T \delta^{(2)}$$

Realizamos el mismo procedimiento que el anterior

$$\begin{aligned} \delta^{(1)} &= \begin{bmatrix} \delta_1^{(1)} \\ \delta_2^{(1)} \end{bmatrix} = \begin{bmatrix} f'(z_1^{(1)})\hat{e}_1^{(1)} \\ f'(z_2^{(1)})\hat{e}_2^{(1)} \end{bmatrix} \\ &= \begin{bmatrix} f'(z_1^{(1)}) \\ f'(z_2^{(1)}) \end{bmatrix} \circ \begin{bmatrix} \hat{e}_1^{(1)} \\ \hat{e}_2^{(1)} \end{bmatrix} \end{aligned}$$

$$\delta^{(1)} = f'(z^{(1)}) \circ \hat{e}^{(1)}$$

Calculamos de manera conjunta la actualización de pesos de la capa de salida

$$\begin{array}{cc} \text{Capa de entrada} & \text{Capa de salida} \\ w_{11}^{(2)} \leftarrow w_{11}^{(2)} + \eta\delta_1^{(2)}a_1^{(1)} & w_{12}^{(2)} \leftarrow w_{12}^{(2)} + \eta\delta_1^{(2)}a_2^{(1)} \\ w_{21}^{(2)} \leftarrow w_{21}^{(2)} + \eta\delta_2^{(2)}a_1^{(1)} & w_{22}^{(2)} \leftarrow w_{22}^{(2)} + \eta\delta_2^{(2)}a_2^{(1)} \end{array}$$

$$\begin{aligned} \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix} &\leftarrow \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix} + \eta \begin{bmatrix} \delta_1^{(2)}a_1^{(1)} & \delta_1^{(2)}a_2^{(1)} \\ \delta_2^{(2)}a_1^{(1)} & \delta_2^{(2)}a_2^{(1)} \end{bmatrix} \\ W^{(2)} &\leftarrow W^{(2)} - \eta\frac{\partial J}{\partial W^{(2)}} \\ \text{donde } \frac{\partial J}{\partial W^{(2)}} &= \begin{bmatrix} \frac{\partial J}{\partial w_{11}^{(2)}} & \frac{\partial J}{\partial w_{12}^{(2)}} \\ \frac{\partial J}{\partial w_{21}^{(2)}} & \frac{\partial J}{\partial w_{22}^{(2)}} \end{bmatrix} \end{aligned}$$

haciendo unos cálculos de matriz obtenemos que;

$$w^{(2)} \leftarrow w^{(2)} + \eta(\delta^{(2)}a^{(1)})$$

y con la actualización de pesos tendriamos que

$$\begin{aligned} \triangle W^{(2)} &= \eta(\delta^{(2)}a^{(1)}) \\ W^{(2)} &\leftarrow W^{(2)} + \triangle W^{(2)} \end{aligned}$$

Calculemos la actualización de pesos para la capa de entrada.

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \delta^{(2)} = \begin{bmatrix} \delta_1^{(1)} \\ \delta_2^{(1)} \end{bmatrix}$$

$$\begin{aligned} W^{(1)} &\leftarrow W^{(1)} - \eta\frac{\partial J}{\partial W^{(1)}} \\ W^{(2)} &\leftarrow W^{(2)} + \eta(\delta^{(1)}x) \end{aligned}$$

$$\begin{aligned} \triangle W^{(1)} &= \eta(\delta^{(1)} x^{(1)}) \\ W^{(1)} &\leftarrow W^{(1)} + \triangle W^{(1)} \end{aligned}$$

Resumiendo la actualización de pesos de $W^{(1)}$ y $W^{(2)}$ tenemos;

$$\begin{array}{ll} \triangle W^{(1)} = \eta(\delta^{(1)} x^{(1)}) & \triangle W^{(2)} = \eta(\delta^{(2)} a^{(1)}) \\ W^{(1)} \leftarrow W^{(1)} + \triangle W^{(1)} & W^{(2)} \leftarrow W^{(2)} + \triangle W^{(2)} \end{array} \tag{4.31}$$

A partir de este análisis, la red de neuronas que trabajamos anteriormente para una capa de entrada, una capa oculta y una capa de salida, nos servirá de apoyo para deducir y encontrar un algoritmo de retropropagación con ayuda de las ecuaciones que obtuvimos.

4.13. Algoritmo de retropropagación

El siguiente paso es generalizar el algoritmo de propagación hacia atrás sobre una red que contenga l capas. Si observamos en la figura 4.26, tenemos una red con una capa de entrada, un número de l capas ocultas y una capa de salida, asignamos respectivamente los parámetros para cada capa y los nodos correspondientes, para la unidad de salida calculamos el primer error $\hat{e}$ de la salida y luego calculamos el δ asociada a la capa l, esto se hará de igual forma para las demás capas ocultas realizando el método de las operaciones de derecha a izquierda, así para terminar de actualizar de manera conjunta todos los pesos w para cada una de las capas ocultas.

Algoritmo de propagación hacia atrás para l capas.

1. Para cada $k \in \{1, 2, .., l\}$ inicialice w^k con valores aleatorios entre $[-1, 1]$.

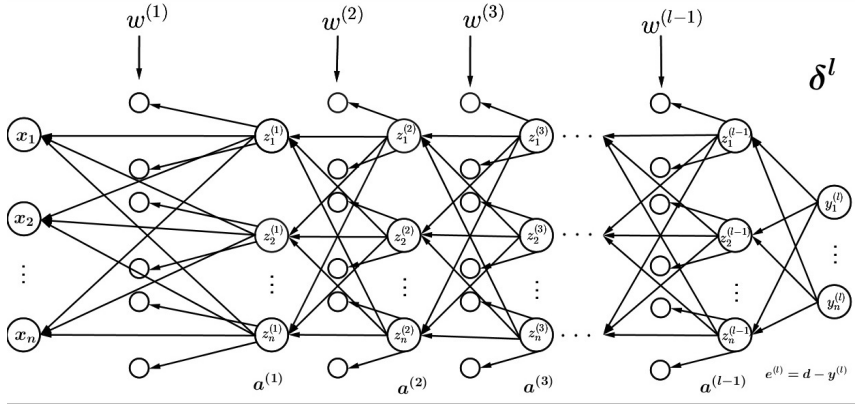


Figura 4.26: Modelo de red de propagación hacia atrás con l capas.

2. Tome un dato de entrada x y propaguelo hacia adelante

$$\begin{aligned} z^{(1)} &= wx \\ a^{(1)} &= f(z^{(1)}) \end{aligned}$$

3. Para cada $k \in \{2, .., l\}$

$$\begin{aligned} z^{(k)} &= w^k a^{(k-1)} \\ a^{(k)} &= f(z^{(k)}) \end{aligned}$$

4. Calcular el error $\hat{e}$ y el delta δ de la salida

$$\begin{aligned} e^{(l)} &= d - a^{(l)} \\ \delta^{(l)} &= f'(z^{(l)})\hat{e}^{(l)} \end{aligned}$$

5. Para cada $k \in \{l-1, l-2, .., 2, 1\}$

$$\begin{aligned} \hat{e}^{(k)} &= w^{(k+1)}\delta^{(k+1)} \\ \delta^{(k)} &= f'(z^{(k)})\hat{e}^{(k)} \end{aligned}$$

6. Realizamos la propagación hacia atrás actualizando los pesos w.

$$\begin{aligned} \text{para } k &= 1 \\ \triangle w^{(1)} &= \eta(\delta^{(1)} x^{(1)}) \\ w^{(1)} &\leftarrow w^{(1)} + \triangle w^{(1)} \end{aligned}$$

$$\begin{aligned} \text{para } k &= \{2, 3, ..l\} \\ \triangle w^{(k)} &= \eta(\delta^{(k)} a^{(k-1)}) \\ w^{(k)} &\leftarrow w^{(k)} + \triangle w^{(k)} \end{aligned}$$

Al calcular la actualización de pesos w concluimos que el algoritmo de propagación funcionará al realizar un solo proceso iterativo, entonces, para que una red sea entrenada y devuelva valores óptimos, se necesita repetir e iterar la veces necesarias para poder ajustar mejor los valores y tener una red mejor entrenada.

Capítulo 5

Proyecto MNIST: reconocimiento de patrones

El reconocimiento de patrones es la disciplina científica cuyo objetivo es la clasificación de objetos en una serie de categorías o clases. Dependiendo de la aplicación, estos objetos pueden ser imágenes o formas de onda de señal, o cualquier tipo de medición que deba clasificarse. El reconocimiento de patrones es una parte integral de la mayoría de los sistemas de inteligencia artificial creados para la toma de decisiones. [18]

5.1. ¿Qué es la base de datos MNIST?

Como una aplicación de lo explicado anteriormente en el presente trabajo, se expondrá un proyecto de reconocimiento de patrones. En este caso, la aplicación es el reconocimiento de dígitos, hechos mediante escritura a mano. Dicha aplicación es un proyecto muy popular, conocido como la Base de datos de MNIST. La base de datos de MNIST (Modified National Institute of Standards and Technology database) es una gran base

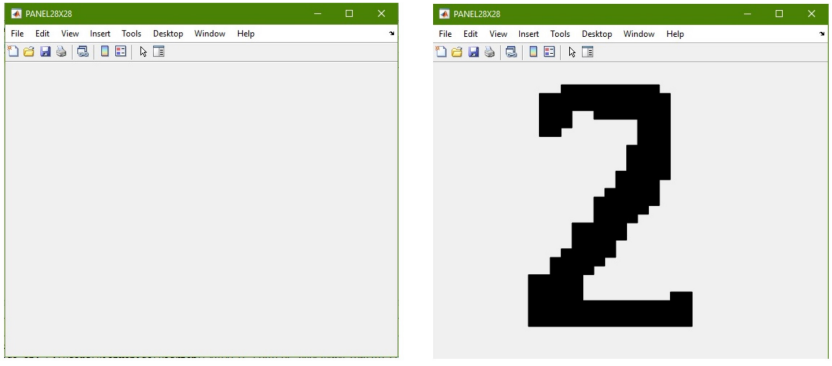
de datos de dígitos escritos a mano, que se usa comúnmente para entrenar sistemas de procesamiento de imágenes. La base de datos de MNIST contiene 60,000 imágenes de entrenamiento y 10,000 imágenes de prueba, donde estas imágenes fueron estandarizadas en un tamaño de 28×28 pixeles. Es posible descargar esta base de datos a tráves de algunos sitios de internet, por ejemplo, (*http://yann.lecun.com/exdb/mnist/*).

Figura 5.1: Algunos dígitos hechos a mano, guardados en una imagen en blanco y negro de 28 × 28 pixeles de la base de datos MNIST.

Este proyecto es realizado con la programación en Matlab, un software que contiene la capacidad de realizar proyectos avanzados, efectuar eficientemente operaciones matemáticas matriciales y que además tiene una amplia paquetería y librería para manejar y entrenar redes de neuronas artificial. El objetivo del proyecto es entrenar una red con el propósito de que el usuario pueda manejar e interactuar con los programas, dibujar los dígitos en una paleta o panel de figura, crear nuestra propia de base datos como prueba y hacer que la red ya entrenada pueda darnos una predicción de los dígitos proporcionados por el usuario.

5.2. Panel de figura

Como primer punto tenemos un panel de figura para realizar y dibujar un dígito en ella.



a) Panel de figura sin dibujar

b) Panel de dibuja dibujado

Figura 5.2: Panel de figura para dibujar un dígito

Guardamos y exportamos la imagen en formato JPG con la dimensión de 28×28 píxeles.

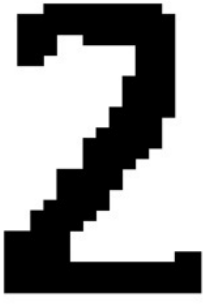


Figura 5.3: Imagen dibujada y exportada con una dimensión de 28×28 pixeles.

5.3. Crear base de datos

Crear nuestra propia base de datos es otro de los objetivos a implementar en este proyecto, para eso se necesitó crear varias imágenes de diferentes dígitos con la dimensión de 28×28 pixeles, guardalos en un directorio o una carpeta con todas la imágenes disponibles.

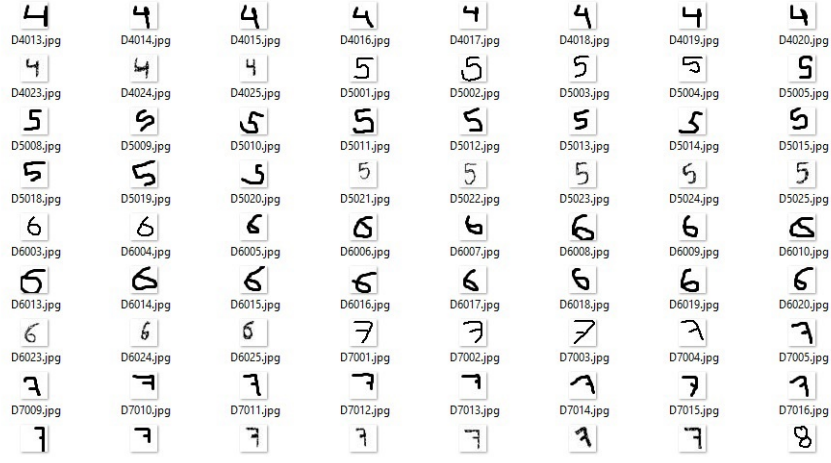


Figura 5.4: Se realizaron varias imágenes de dígitos para crear una base de datos

Necesitamos exportar estas imágenes a un solo archivo o formato, y la solución fue crear un código en donde se lea cada una de la imágenes y los guarda en un archivo de Excel con extensión .csv, en donde al tener una imagen de 28×28 pixeles se tendría que reacomodar a un vector de [1×784], porque la cantidad de pixeles que equivale una imagen es justamente de 784 pixeles en total. La función del código es contar la cantidad de imágenes (k) que hay en el directorio, leer cada una de ellas y exportar sus valores en pixeles a un archivo que lo denominamos como *dataMnistPrueba.csv,* guardará todo los valores de cada imagen, así construyendo una matriz de valores de [k×784].

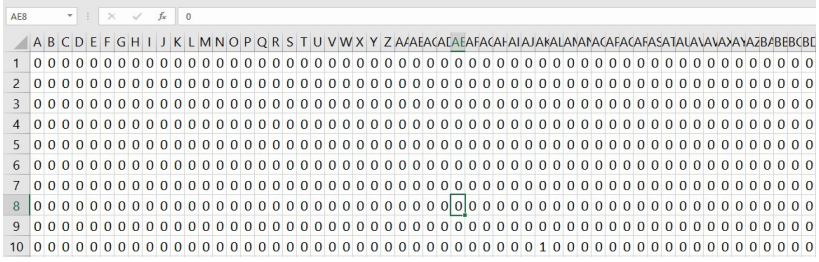


Figura 5.5: Base de datos guardada en *dataMnistPrueba.csv*

Además, se necesitará crear otro archivo en formato .csv en donde cada una de las imágenes que fueron creadas tendrá una etiqueta correspondiente al número dibujado, por lo que en el archivo *dataMnistPrueba.csv* se le agregará otra columna de dimensión [1,k] para escribir las etiquetas, y se guarda en otro archivo con nombre *dataMnistPrueba_ Etiqueta.csv*, esto con la idea de que en los siguientes códigos sea más accesible leer sus valores, para comprobar el entrenamiento y poner a prueba nuestra red artificial.

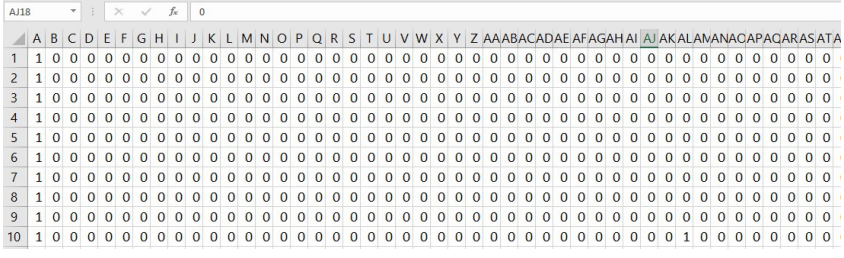


Figura 5.6: Base de datos con etiqueta correspondiente, guardada en *dataMnistPuebaEtiqueta.csv*

Es decir, en la figura 5.6 tenemos un ejemplo del archivo *dataMnistPrueba_ Etiqueta.csv,* las imágenes que se habían generado fueron exportadas y guardada en forma de filas de tamaño de [1×784], pero en este caso se agregó otra columna extra para asociar una etiqueta correspondiente a la primera fila, en la figu-

ra 5.6 observamos que la primera celda contiene el número 1, esto nos quiere decir que, la primera fila del archivo está guardado todos los valores de un número dibujado, y esté se le etiqueta el número que le corresponda.

Una vez hechos todos los pasos anteriores, la segunda etapa corresponde en leer todos los valores a un programa, poner a prueba nuestra base de datos para que pueda tomar la información y nos indique que número corresponde. La base de datos de MNIST será de apoyo para reforzar y comprobar el código cuando la red esté entrenada.

5.4. Estructura de una red entrenada

El siguiente paso es descargar la base de datos de MNIST y guardar en un directorio fijo. Si observamos en la figura 5.7 hay una pestaña definida como *load_ images.m,* este programa manda a llamar la base de datos y a su vez están siendo aplicados a una función de *loadMNISTImages()* y *loadLabels()*, las pestañas de *loadMNISTImages.m y loadLabels.m* son funciones que realizan para que la base de datos se puedan descomprimir y decodificar los datos.

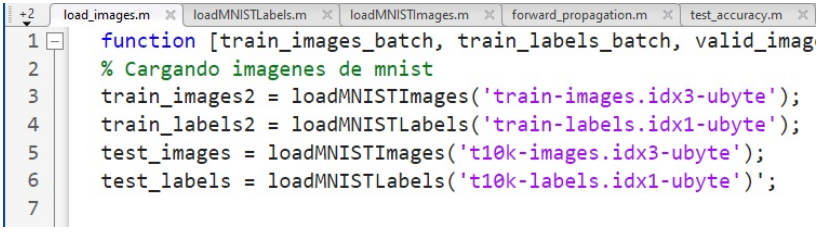


Figura 5.7: Exportar la base de datos MNIST

Es importante mencionar que la base de datos se guardarán en un formato que MATLAB tiene a su disposición, nos referimos al formato .mat. Este formato tiene la capacidad de procesar partes de conjunto de datos muy grandes, ya que esto favorece el

rendimiento del código. Para guardar este tipo de datos se puede realizar como se muestra en la figura 5.8, *donde test_images* y *test_labels* son dos conjuntos de datos de prueba de imágenes y etiqueta, lo guadará en un parámetro definido como *test_set,* una vez que se ejecute este programa generará un archivo *test_set.mat.*

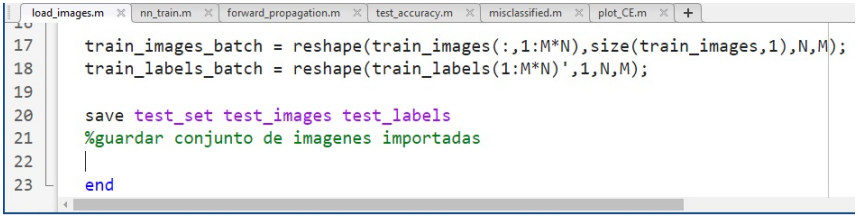


Figura 5.8: Función para guardar conjuntos de datos en un archivo *test_set.mat*

Para la figura 5.9 nos muestra el programa *nn_train.m*, este programa realiza la parte fundamental para entrenar la red. La estructura del programa inicia con las primeras lineas del código definiendo los hiperparámetros, estos son parámetros que nos permiten ajustar y controlar el procesamiento de entrenamiento de la red, observamos el *batchsize(mini-lotes),* este parámetro su función es designar un subconjunto del conjunto de datos para lograr un equilibrio al aplicar el descenso del gradiente permitiendo la estabilidad del algoritmo. Tenemos el *learning_rate (taza de aprendizaje),* como anteriormente se habia definido, es un parámetro que nos permite ajustar la convergencia del algoritmo, *numhid (numero de capas ocultas),* asigna el número de capas ocultas que queremos trabajar, el parámetro *epochs (épocas)* es el número de proceso iterativo que se tiene que realizar para entrenar la red para ajustar los valores de pesos, y los parámetros *momentum, lambda, dropout* son escalares que nos ayuda a la regularización de valores de salida cuando se calcula para cada capa oculta. Cuando la red está siendo entrenada y no muestra resultados favorables, modificar los hiperparámetros

es una buena opción, ya que esto permite modificar los pesos y controlar el error.

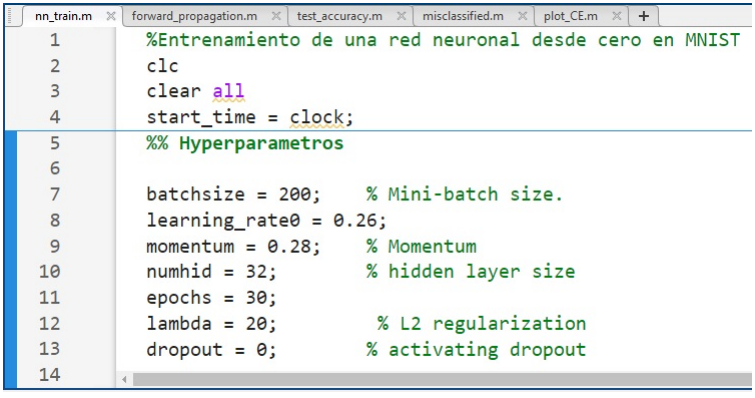


Figura 5.9: Programa para entrenar la red.

El programa inicializa los pesos de la red para iniciar los cálculos, procede a realizar el método de propagación hacia adelante para obtener el valor de la salida, iniciamos el proceso iterativo para un valor época=1 hasta la cantidad de épocas que el usuario haya ingresado e inicia el proceso para entrenar la red, calculando el error de la capa de salida y realizando los cálculos correspondientes al algoritmo de retropropagación, donde la principal función es la actualización del conjunto de pesos para iterar nuevamente.

En las pestañas de *forward_propagation.m, test_acurracy.m y plot_CE.m* son funciones que fueron definidas independientemente para realizar otras tareas, y que al momento de ejecutar *nn_train.m* los mandará a llamar para complementar al entrenamiento del modelo, *forward_propagation.m* es un programa que está definido a recibir los valores y pesos de la red y este calcula la función de activación (*sigmoide*) de todas las capas ocultas, en donde, para la última capa l, la función que se utiliza es una función conocida como el *softmax.* La función softmax mantiene la suma de los valores de salida en uno y también limi-

ta las salidas individuales para que estén dentro de los valores de 0-1[22]. El programa *test_ acurracy.m* es una función que clasifica y lee las imágenes correctamente a partir de *test_ set.mat.*

El objetivo del proyecto es proporcionar nuestra propia base de datos, una vez que la red ha sido entrenada por la base de datos de MNIST llega el momento de poner a prueba ahora nuestro conjunto de datos, de la figura 5.10 vamos mandar a llamar los archivos *dataMnistPrueba.csv* y *dataMnistPrueba_ Etiqueta.csv* para exportar los datos y guardarlos en el formato de *example.mat.*

Base_de_datos_MAT.m | forward_propagation.m | test_accuracy.m | nn_train.m | misclassified.m | plot_CE.m | +

```
1   dataMnist = csvread('dataMnistPrueba.csv');
2
3   dataMnistT=(dataMnist)';
4   prueba = dataMnistT(:,1);
5
6   dataMnist_EtiquetaM = csvread('dataMnistPrueba_Etiqueta.csv');
7   Etiqueta = dataMnist_EtiquetaM(:,1);
8
9   test_images = dataMnistT;
10  test_labels = (Etiqueta)';
11
12  save example.mat test_images test_labels ;
```

Figura 5.10: Código del programa para generar el archivo example.mat a partir de la base de datos creada.

De todos los programas que anteriormente se han explicado, necesitamos un programa principal en donde mande a llamar todas las funciones de manera simplificada, a este programa lo definimos como *misclasified.m* si observamos en la figura 5.11 en la línea 12 y 13, observamos que los formatos de prueba son *load test_ set* y *load example.mat.*

Al ejecutar el programa *misclasified.m* imprime un panel de figura como se muestra en 5.12, del lado izquierdo observamos que imprime el dígito que fue dibujado, y en el lado derecho muestra las probabilidades del número que corresponda.

nn_train.m | forward_propagation.m | misclassified.m | test_accuracy.m | plot_CE.m | +

```
clc
clear all
model_name = 'model.mat';
try load(model_name)
catch
    load model_9896
    disp('model.mat not found, model_9896.mat evaluated instead.')
end

%load test_set.mat
load example.mat
```

Figura 5.11: Programa principal para ejecutar las bases de datos de MNIST y la base de datos de ejemplos propuesta para la prueba.

Si observamos detallamente esta figura podemos visualizar que el dígito que corresponde es al numero 6, pero en la gráfica nos muestra con un porcentaje más alto nos dice que es el número 0, esto nos lleva a deducir que el programa funciona y la red si fue entrenada, pero esto nos lleva a pensar que debemos cambiar los valores de los hiperparámetros para volver a entrenar la red y en la próxima ejecución tengamos una respuesta favorable.

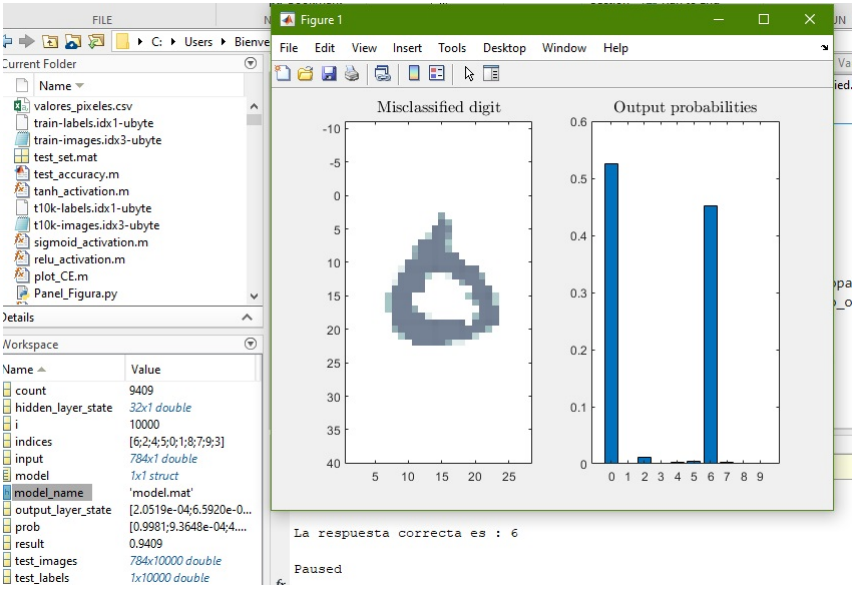


Figura 5.12: Resultado de la red entrenada al reconocimiento de un dígito escrito a mano.

Conclusión

Llegar a desarrollar una aplicación que simule una de las características del cerebro humano, nos llevó a pensar cómo el cerebro trabaja de forma estructurada, en recibir la información y procesar para determinar la salida. El proyecto que se realizó fundamentó parte del trabajo al llevar y estudiar varios campos de estudio, el reto de aprender a elegir qué lenguaje de programación será accesible, e incluso crear nuestra propia base de datos. Los modelos y ecuaciones matemáticas que el cerebro llega manejar para realizar distintas tareas funcionales.

Concluir que si hemos estudiado toda la capacidad del cerebro humano actualmente es algo que aún no podemos decidir a pesar de tener la tecnología y modelos que ayudan a plasmar esta abstracción, solo hemos sentado un pequeño porcentaje para fundamentar y seguir con investigaciones que a futuro la humanidad podrá llegar a experimentar y demostrar.

Bibliografía

[1] S Ramon Cajal y col.((Histology of the nervous system of man and vertebrates)). En: History of Neuroscience (OxfordUniv Press, New York) 6 (1995).

[2] Ongay Fernando Carrillo Antonio Mendoza Miguel Angel.((Integrate and fire neurons and circle maps)). En: WSEAS TRANSACTIONS ON BIOLOGY AND BIOMEDICINE1 (2004), p´ags. 287-293.

[3] Simon Haykin y N Network. ((A comprehensive foundation)). En: Neural networks 2.2004 (2004), pág. 41.

[4] Jeff Heaton. ((AIFH, volume 3: deep learning and neural networks)). En: Journal of Chemical Information and Modeling 3 (2015).

[5] Donald Olding Hebb. The organization of behavior: A neuropsychological theory. Psychology Press, 2005.

[6] John Hertz, Anders Krogh y Richard G Palmer. Introduction to the theory of neural computation. CRC Press, 2018.

[7] Eugene Isaacson y Herbert Bishop Keller. Analysis of numerical methods. Courier Corporation, 2012.

[8] Pedro Isasi e Ines Galvan. ((Redes de neuronas artificiales. Un enfoque practico)). En: Madrid. España, Pearson Educacion, SA (2004).

[9] Eric R Kandel y col. Principles of neural science. Vol. 4. McGraw-hill New York, 2000.

[10] Teuvo Kohonen. Self-Organizing Maps, 3rd Edition. 3rd. 2000.

[11] Roland E. Larson. Introduccion Al Algebra Lineal. Limusa, 2002.

[12] Antonio Carrillo Ledesma. ((Dinámica y multiestabilidad de mapeos en el circulo)). Universidad Nacional Autónomade la Ciudad de México, 2003.

[13] Cheng Soon Ong Marc Peter Deisenroth A. Aldo Faisal. Mathematics For Machine Learning. Cambridge University Press, 2019.

[14] Warren S McCulloch y Walter Pitts. ((A logical Calculus of the ideas immanent in nervous activity)). En: 5.4 (1943), págs. 115-133.

[15] Raul Rojas. Neural networks: a systematic introduction. Springer Science & Business Media, 2013.

[16] Yousef Saad. Iterative methods for sparse linear systems. SIAM, 2003.

[17] Alan M Turing. ((Computing machinery and intelligence)). En: Parsing the turing test. Springer, 2009, págs. 23-65.

[18] Sergio Theodoridis Konstantinos. Pattern Recognition, Reino Unido: Elsevier(2009).

[19] Carolina Barriga Montoya ((Modelación y análisis de la sincronización de una célula nervisosa periódicamente estimulada: una analogía mecánica)). Universidad Nacional Autónomade la Ciudad de México, 2005

[20] Natalia Carrilo Martinez de la Escalera((Modelación de la actividad neuroeléctrica)). Universidad Nacional Autónomade la Ciudad de México, 2010.

[21] Jerrold E. Marsden,Anthony Tromba. Vector Calculus. W.H. Freeman. 5th Ed. 2003

[22] Phil Kim. MATLAB Deep Learning: ((MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence.)) Aprress. 2017

Apéndice A

Código en Python

```
Código en Python para generar un panel de figura, realizar la
escritura a mano, guardar y exportar el archivo en formato
DIGITO.txt en un vector de [1,784].
import tkinter as tk

# Dimensiones del lienzo y del archivo de imagen
CANVAS_WIDTH = 560
CANVAS_HEIGHT = 560
IMAGE_SIZE = 28

#Inicializar el lienzo
root = tk.Tk()
canvas = tk.Canvas(root, width=CANVAS_WIDTH,
height=CANVAS_HEIGHT, bg='white')
canvas.pack()

# Lista para almacenar los puntos dibujados
points = []
# Variable para indicar si se está dibujando o no
is_drawing = False

# Función para capturar el evento de clic del ratón
```

```
def start_drawing(event):
    global is_drawing
     is_drawing = True

# Función para capturar el evento de movimiento del ratón
def draw(event):
    if is_drawing:
     x = event.x
     y = event.y
     points.append((x, y))
     canvas.create_oval(x, y, x+25, y+25, fill='black')

#Función para capturar el evento de liberación del botón del
ratón
def stop_drawing(event):
     global is_drawing
     is_drawing = False

#Función para guardar la imagen dibujada
def save_image():
 #Crear un archivo de imagen de 28x28 de ceros y unos
 image = [[0] * IMAGE_SIZE for _ in range(IMAGE_SIZE)]

#Escalar y dibujar los puntos en la imagen
 for point in points:
     x = int(point[0] * IMAGE_SIZE / CANVAS_WIDTH)
     y = int(point[1] * IMAGE_SIZE / CANVAS_HEIGHT)
     image[x][y] = 1

#Guardar la imagen en un archivo de texto
 with open('DIGITO.txt', 'w') as file:
     for row in image:
     file.write(' '.join(str(pixel) for pixel in row) + ' ')
     print('Imagen guardada como DIGITO.txt')
```

```
#Registrar los eventos del ratón
canvas.bind('<Button-1>', start_drawing)
canvas.bind('<B1-Motion>', draw)
canvas.bind('<ButtonRelease-1>', stop_drawing)

#Crear el botón para guardar la imagen
save_button = tk.Button(root, text='Guardar',
command=save_image)
save_button.pack()
#Iniciar la ventana principal
root.mainloop()
```

Apéndice B

Códigos en MATLAB P1

Código para generar un panel de dibujo y guardar la imagen en formato JPG.

```
function Panel_dibujo()
% Crear una ventana
 fig = figure('Name', 'PANEL28X28', 'NumberTitle', 'off');
 set(fig, 'WindowButtonDownFcn', @startDrawing);
 set(fig, 'WindowButtonMotionFcn', @continueDrawing);
 set(fig, 'WindowButtonUpFcn', @stopDrawing);

% Crear el panel de figura
 axesHandle = axes('Parent', fig);
 axis off;
 axis([0 1 0 1]);
 hold on;

% Variables de estado
 isDrawing = false;
 markerSize = 10;

% Calcular dimensiones de píxeles
 pixelWidth = 1 / 28;
 pixelHeight = 1 / 28;
```

```
% Función llamada al presionar el botón del mouse
 function startDrawing(src, ~)
     isDrawing = true;
     point = get(axesHandle, 'CurrentPoint');
     x = floor(point(1, 1) * 28) * pixelWidth;
     y = floor(point(1, 2) * 28) * pixelHeight;
     rectangle('Position', [x, y, pixelWidth, pixelHeight],
'FaceColor', 'k');
 end
% Función llamada al soltar el botón del mouse
 function stopDrawing(~, ~)
     isDrawing = false;
     exportImage();
 end
% Función llamada al mover el mouse después de presionar el
botón
 function continueDrawing(src, ~)
     if isDrawing
         point = get(axesHandle, 'CurrentPoint');
         x = floor(point(1, 1) * 28) * pixelWidth;
         y = floor(point(1, 2) * 28) * pixelHeight;
         rectangle('Position', [x, y, pixelWidth,
pixelHeight], 'FaceColor', 'k');
     end
 end
% Función para exportar la figura como imagen en formato JPG
 function exportImage()
 %Pedir al usuario el nombre del archivo de imagen
     [file, path] = uiputfile('*.jpg', 'Guardar imagen como');


 % Verificar si se seleccionó un archivo
     if isequal(file, 0) || isequal(path, 0)
         disp('Exportación de imagen cancelada.');
```

```
    return;
    end

 %Guardar la figura como imagen en formato JPG
    outputPath = fullfile(path, file);
    print(fig, outputPath, '-djpeg', '-r300');

    disp(['Imagen guardada en: ', outputPath]);
 end
end
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%Código para ajustar nuestra imagen de 28x28 pixeles
%Ruta del archivo de imagen
rutaImagen = 'DIGITO3.jpg';
% Leer la imagen
imagen = imread(rutaImagen);

%Ajustamos la imagen en 28x28 pixeles
resizedImage = imresize(imagen, [28,28]);

%Reescribimos y guardarmos nuestra nueva imagen ajustada
imwrite(resizedImage, 'VDIGITO3.jpg', 'jpg');
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
%Código para exportar una imagen a un vector de pixeles en un
formato .csv
% Ruta del archivo de imagen
rutaImagen = 'VDIGITO3.jpg';
imagen = imread(rutaImagen);

% Convertir la imagen a escala de grises (si no lo está ya)
imagen_gris = rgb2gray(imagen);

% Convertir la imagen a una matriz binaria (0's y 1's)
```

```
umbral = 128; % Umbral para binarizar la imagen (ajústalo
según sea necesario)
imagen_binaria = imagen_gris < umbral;

% Convertir la imagen en una matriz de una sola fila
fila=imagen_binaria(:,:,1);
imagenfila =zeros(1,784);
filaprueba = reshape(fila,[1,784]);

% Crear un archivo CSV y escribir los valores de píxeles
nombreArchivoCSV = 'valores_pixeles.csv';
csvwrite(nombreArchivoCSV, filaprueba);
```

- -

```
%Código para leer el archivo .csv e imprimir la imagen
data = csvread('valores_pixeles.csv');

for i=1:size(data,1)
     clc
     figure(1)
     clf(1)
     digit = data(i,1:784);
     kep = reshape(digit,[28,28]);
     imagesc(kep)
     colormap(bone)
     title('Visualizacion de
digitos','interpreter','latex','fontsize',12)
     axis equal
end
```

- -

```
Código para importar la imagenes en un archivo .csv
```

```
% Ruta de la carpeta que contiene las imágenes en formato
JPEG
%C:\Users\Bienvenido\Desktop\PRUEBA_BASE_DE_DATOS
carpetaImagenes =
'C:\Users\Bienvenido\Desktop\CAPTESIS_CODIGOS_PROGRAMACION';

%carpetaImagenes = 'ruta_de_tu_carpeta_de_imagenes/';
% Obtener la lista de archivos JPEG en la carpeta
archivos = dir(fullfile(carpetaImagenes, '*.jpg'));
numArchivos = length(archivos);

% Leer la primera imagen para obtener las dimensiones
rutaPrimeraImagen = fullfile(carpetaImagenes,
archivos(1).name);
imagen = imread(rutaPrimeraImagen);
[filas, columnas, val] = size(imagen);

% Crear una matriz para almacenar los valores de píxeles de
todas las imágenes
valoresPixeles = zeros(numArchivos ,filas * columnas);
imagenfila =zeros(1,784);

% Leer y procesar cada imagen
for i = 1:numArchivos
     rutaImagen = fullfile(carpetaImagenes, archivos(i).name);

     imagen = imread(rutaImagen);
 %Convertir la imagen a escala de grises (si no lo está ya)
     imagen_gris = rgb2gray(imagen);
% Convertir la imagen a una matriz binaria (0's y 1's)
     umbral = 128;% Umbral para binarizar la imagen (ajústalo
según sea necesario)
     imagen_binaria = imagen_gris < umbral;
     fila=imagen_binaria(:,:,1);
     imagenfila =zeros(1,784);
```

```
     valoresPixeles(i,:) = reshape(fila,[1,784]);
end

% Crear un archivo CSV y escribir los valores de píxeles en
columnas separadas
nombreArchivoCSV = 'dataMnistPrueba.csv';
csvwrite(nombreArchivoCSV, valoresPixeles);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

dataMnist = csvread('dataMnistPrueba.csv');
dataMnistT=(dataMnist)';
prueba = dataMnistT(:,1);
dataMnist_EtiquetaM = csvread('dataMnistPrueba_Etiqueta.csv');

Etiqueta = dataMnist_EtiquetaM(:,1);
test_images = dataMnistT;
test_labels = (Etiqueta)';
save example.mat test_images test_labels ;
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%Código para guardar nuestra bases de datos y etiquetas en
formato .mat
dataMnist = csvread('dataMnistPrueba.csv');
dataMnistT=(dataMnist)';

prueba = dataMnistT(:,1);
dataMnist_EtiquetaM = csvread('dataMnistPrueba_Etiqueta.csv');

Etiqueta = dataMnist_EtiquetaM(:,1);

test_images = dataMnistT;
test_labels = (Etiqueta)';

save example.mat test_images test_labels ;
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%funcion de activacion arctan(x)
function output = atan_activation(input)
 output = atan(input);
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%función de activación ReLU
function output = relu_activation(input)
 output = max(0, input);
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%funcion de activacion sigmoid(x)
function output = sigmoid_activation(input)
 output = 1 ./ (1 + exp(-input));
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%funcion de activacion tangente hiperbolica
function output = tanh_activation(input)
 output = tanh(input);
end
```

Apéndice C

Códigos en MATLAB P2

- -

%%%Los códigos para esta parte fueron extraídos a partir
%%%de https://github.com/mkisantal para complementar
%%%los códigos anteriores.

```
function images = loadMNISTImages(filename)
%loadMNISTImages returns a 28x28x[number of MNIST images]
matrix containing
%the raw MNIST images
fp = fopen(filename, 'rb');
assert(fp ~= -1, ['Could not open ', filename, "]);
magic = fread(fp, 1, 'int32', 0, 'ieee-be');
assert(magic == 2051, ['Bad magic number in ', filename, "]);
numImages = fread(fp, 1, 'int32', 0, 'ieee-be');
numRows = fread(fp, 1, 'int32', 0, 'ieee-be');
numCols = fread(fp, 1, 'int32', 0, 'ieee-be');
images = fread(fp, inf, 'unsigned char');
images = reshape(images, numCols, numRows, numImages);
images = permute(images,[2 1 3]);
fclose(fp);
% Reshape to #pixels x #examples
```

```
images = reshape(images, size(images, 1) * size(images, 2),
size(images, 3));
% Convert to double and rescale to [0,1]
images = double(images) / 255;
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

function labels = loadMNISTLabels(filename)
%loadMNISTLabels returns a [number of MNIST images]x1 matrix
containing
%the labels for the MNIST images
fp = fopen(filename, 'rb');
assert(fp ~= -1, ['Could not open ', filename, '']);
magic = fread(fp, 1, 'int32', 0, 'ieee-be');
assert(magic == 2049, ['Bad magic number in ', filename, '']);
numLabels = fread(fp, 1, 'int32', 0, 'ieee-be');
labels = fread(fp, inf, 'unsigned char');
assert(size(labels,1) == numLabels, 'Mismatch in label
count');
fclose(fp);
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

function [train_images_batch, train_labels_batch,
valid_images, valid_labels, test_images, test_labels] =
load_images(N)
% Cargando imagenes de mnist
train_images2 = loadMNISTImages('train-images.idx3-ubyte');
train_labels2 = loadMNISTLabels('train-labels.idx1-ubyte');
test_images = loadMNISTImages('t10k-images.idx3-ubyte');
test_labels = loadMNISTLabels('t10k-labels.idx1-ubyte')';
train_images = train_images2(:,1:50000);
train_labels = train_labels2(1:50000,1);
size (train_images)
size(train_labels)
```

```
valid_images = train_images2(:,50001:60000);
valid_labels = train_labels2(50001:60000)';
M = floor(size(train_images,2)/N);
train_images_batch =
reshape(train_images(:,1:M*N),size(train_images,1),N,M);
train_labels_batch = reshape(train_labels(1:M*N)',1,N,M);%
labels transposed!
save test_set test_images test_labels
%save imported_imageset train_images valid_images test_images
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

function [] = plot_CE(CE_array,CE_array_avg,CE_array_valid)
[numbatches,epochs] = size(CE_array);
CE_values = reshape(CE_array,[1 epochs*numbatches]);
[num_valid,epochs] = size(CE_array_valid);
CE_valid_values = reshape(CE_array_valid,[1
num_valid*epochs]);
%Gráfico de error de entropía cruzada de validación
ticks = [0:10:epochs] * num_valid;
hold on
figure(1)
plot(CE_valid_values,'b')
xlabel('epoch'); ylabel('CE'); title('Cross-Entropy error')
set(gca,'XTick',ticks)
set(gca,'XTickLabel',[0:10:epochs] );
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%Evaluación del rendimiento de clasificación en el conjunto
de entrenamiento
clear all
model_name = 'model.mat';
try load(model_name)
catch
 load model_9896
```

```
 disp('model.mat not found, model_9896.mat evaluated
instead.')
end
load test_set
%contar imágenes clasificadas correctamente
count=0;
for i=1:size(test_images,2)

 input = test_images (:,i);

 [hidden_layer_state, output_layer_state] =
forward_propagation...
 (input, model.input_to_hidden_weights,
model.hidden_to_output_weights,...
 model.hidden_bias, model.output_bias);

 [prob, indices] = sort(output_layer_state, 'descend');
 indices = indices-1;

 if indices(1) == test_labels(i)
 count = count+1;
 end
end
result = count/size(test_images,2);
fprintf(1, '\n\nImagenes correctamente clasificadas en el
equipo de prueba: %.2f%% \n', result*100);
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

function [hidden_layer_state, output_layer_state] = ...
 forward_propagation(input_batch, input_to_hidden_weights,...
 hidden_to_output_weights, hidden_bias, output_bias)
[input_number, batch_size] = size(input_batch);
[numhid, output_size] = size(hidden_to_output_weights);
inputs_to_hid_units = input_to_hidden_weights' *
input_batch...
```

```
 + repmat(hidden_bias,1,batch_size);

hidden_layer_state = 1 ./ (1 +
exp(-inputs_to_hid_units));%%%FUNCION DE ACTIVACION
%hidden_layer_state =
sigmoid_activation(inputs_to_hid_units);%%%FUNCION DE
ACTIVACION SIGMOIDE
%hidden_layer_state =
tanh_activation(inputs_to_hid_units);%%%FUNCION DE ACTIVACION
tangente hiperbolica
%hidden_layer_state =
atan_activation(inputs_to_hid_units);%%%FUNCION DE ACTIVACION
arctang
%hidden_layer_state =
relu_activation(inputs_to_hid_units);%%%FUNCION DE ACTIVACION
RELU
%hidden_layer_state = atan(inputs_to_hid_units);%%%FUNCION
DE ACTIVACION ARCTAN
%% Estado de la capa de salida
inputs_to_softmax = hidden_to_output_weights' *
hidden_layer_state...
 + repmat(output_bias,1,batch_size);
% making softmax inputs =< 0
inputs_to_softmax = inputs_to_softmax...
 - repmat(max(inputs_to_softmax), output_size, 1);
output_layer_state = exp(inputs_to_softmax);
output_layer_state = output_layer_state ./ ...
 repmat(sum(output_layer_state, 1), output_size, 1);
end
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%Entrenamiento de una red neuronal desde cero en MNIST
clc
clear all
start_time = clock;
```

```
%% Hyperparametros
batchsize = 100;% Mini-batch size.
learning_rate0 = 0.26;
momentum = 0.28;% Momentum
numhid = 32;% hidden layer size
epochs = 30;
lambda = 20;% L2 regularization
dropout = 0;% activating dropout
%Cargar base de datos MNIST
[train_images, train_labels, valid_images, valid_labels,...
 test_images, test_labels] = load_images(batchsize);
[input_size, batchsize, numbatches]=size(train_images);
output_size = 10;
%% Inicializacion
%Inicialización de peso
input_to_hidden_weights = 1/sqrt(input_size) *
randn(input_size, numhid);
hidden_to_output_weights = zeros(numhid, output_size);
hidden_bias = zeros(numhid,1);
output_bias = zeros(output_size,1);
CE_array = zeros(numbatches,epochs);
% initializing gradient matrices
input_to_hidden_weights_delta = zeros(input_size, numhid);
hidden_to_output_weights_delta = zeros(numhid, output_size);
hidden_bias_delta = zeros(numhid,1);
output_bias_delta = zeros(output_size,1);
tiny = exp(-30);
show_training_CE_after = 100;
show_validation_CE_after = 100;
count = 0;
CE_array_avg =
zeros(floor(numbatches/show_training_CE_after),epochs);
CE_array_valid =
zeros(floor(numbatches/show_validation_CE_after),epochs);
for epoch = 1:epochs;
```

```
 fprintf(1, 'Epoch %d\n', epoch);
 this_chunk_CE = 0;
 trainset_CE = 0;

%% Scheduling learning rate
learning_rate = learning_rate0 / (1+((epoch-1)/10)^2);
weight_dec = 1 - learning_rate * lambda / (batchsize *
numbatches);
disp(' Learning rate ')%%%% AQUI SE COMENTA PARA VER LA TASA
DE APRENDIZAJE
disp(learning_rate)
for m=1:numbatches %loop over mini-batches
%% Dropout
if dropout == 1

% storing complete matrices, scaling up!
 full_input_to_hidden_weights = 2 * input_to_hidden_weights;
 full_hidden_to_output_weights = 2 * hidden_to_output_weights;

 full_hidden_bias = 2 * hidden_bias;

 full_input_to_hidden_weights_delta = 2 *
input_to_hidden_weights_delta;
 full_hidden_to_output_weights_delta = 2 *
hidden_to_output_weights_delta;
 full_hidden_bias_delta = 2 * hidden_bias_delta;

% random element selection
 selecting_vector = sort(randperm(numhid,floor(numhid/2)));

 input_to_hidden_weights = full_input_to_hidden_weights( :,
selecting_vector) ;
 hidden_to_output_weights = full_hidden_to_output_weights(
selecting_vector, :);
```

```
 hidden_bias = full_hidden_bias( selecting_vector, :);

 input_to_hidden_weights_delta =
full_input_to_hidden_weights_delta(:, selecting_vector);
 hidden_to_output_weights_delta =
full_hidden_to_output_weights_delta(selecting_vector, :);
 hidden_bias_delta = full_hidden_bias_delta(selecting_vector,
:);
end
%% Forward propagate
input_batch = train_images(:,:,m);
[hidden_layer_state, output_layer_state] =
forward_propagation...
 (input_batch, input_to_hidden_weights,
hidden_to_output_weights,...
 hidden_bias, output_bias);


%% Error fcn
expansion_matrix = eye(output_size);
target_batch = train_labels(:,:,m);
target_vectors = expansion_matrix(:,target_batch+1);% +1
avoiding zero indices
% LOG Likelihood error function
CE = -sum(sum(target_vectors .* log(output_layer_state +
tiny)))/batchsize;
CE_array(m,epoch)=CE;
% show avg error
count = count + 1;
this_chunk_CE = this_chunk_CE + (CE - this_chunk_CE) / count;
trainset_CE = trainset_CE + (CE - trainset_CE) / m;
% fprintf(1, '\rBatch%d Train CE%.3f', m, this_chunk_CE);
if mod(m, show_training_CE_after) == 0
 fprintf(1, '\n');
 count = 0;
```

```
 CE_array_avg(m/show_training_CE_after,epoch) = this_chunk_CE;

 this_chunk_CE = 0;
end
%% Backpropagation
% Error fcn derivative
error_deriv = output_layer_state - target_vectors;
% output layer
hid_to_output_weights_grad = hidden_layer_state *
error_deriv';
output_bias_grad = sum(error_deriv, 2);
% hidden layer
back_propagated_deriv = (hidden_to_output_weights *
error_deriv) ...
 .* hidden_layer_state .* (1 - hidden_layer_state);
input_to_hid_weights_grad = input_batch *
back_propagated_deriv';
hidden_bias_grad = sum(back_propagated_deriv, 2);
%% Updating weights and biases
% input_to_hidden_weights
input_to_hidden_weights_delta = momentum .*
input_to_hidden_weights_delta...
 + input_to_hid_weights_grad ./ batchsize;
input_to_hidden_weights = input_to_hidden_weights *
weight_dec...
 - learning_rate * input_to_hidden_weights_delta;
% hidden_to_output_weights
hidden_to_output_weights_delta = momentum .*
hidden_to_output_weights_delta...
 + hid_to_output_weights_grad ./ batchsize;
hidden_to_output_weights = hidden_to_output_weights *
weight_dec...
 - learning_rate * hidden_to_output_weights_delta;
% hidden_bias
hidden_bias_delta = momentum .* hidden_bias_delta...
```

```
 + hidden_bias_grad ./ batchsize;
hidden_bias = hidden_bias - learning_rate * hidden_bias_delta;

% output_bias
output_bias_delta = momentum .* output_bias_delta...
 - output_bias_grad ./ batchsize;
output_bias = output_bias - learning_rate * output_bias_delta;

%% Merging weight and bias matrices for dropout
if dropout == 1
% update rows in complete matrices
full_input_to_hidden_weights( :, selecting_vector) =
input_to_hidden_weights;
full_hidden_to_output_weights( selecting_vector, :) =
hidden_to_output_weights;
full_hidden_bias( selecting_vector, :) = hidden_bias;
full_input_to_hidden_weights_delta(:, selecting_vector) =
input_to_hidden_weights_delta;
full_hidden_to_output_weights_delta(selecting_vector, :) =
hidden_to_output_weights_delta;
full_hidden_bias_delta(selecting_vector, :) =
hidden_bias_delta;
% switch back to standard matrix names, scaling!
input_to_hidden_weights = 0.5 * full_input_to_hidden_weights;
hidden_to_output_weights = 0.5 *
full_hidden_to_output_weights;
hidden_bias = 0.5 * full_hidden_bias;
input_to_hidden_weights_delta = 0.5 *
full_input_to_hidden_weights_delta;
hidden_to_output_weights_delta = 0.5 *
full_hidden_to_output_weights_delta;
hidden_bias_delta = 0.5 * full_hidden_bias_delta;
end

%% Validation
```

```
if mod(m, show_validation_CE_after) == 0
 fprintf(1, '\rRunning validation ...');

 target_vectors_validation =
expansion_matrix(:,valid_labels+1);% +1 avoiding zero indices
 datasetsize_validation = size(valid_images, 2);
 [hidden_layer_state, output_layer_state] =
forward_propagation...
 (valid_images, input_to_hidden_weights,
hidden_to_output_weights,...
 hidden_bias, output_bias);

 CE_valid = -sum(sum(target_vectors_validation .*
log(output_layer_state + tiny)))...
 /datasetsize_validation;

 CE_array_valid(m/show_validation_CE_after,epoch) = CE_valid;

 fprintf(1, ' Validation CE%.3f\n', CE_valid)
end
end% end loop over Mini-Batches
%fprintf(1, '\rAverage Training CE%.3f\n', trainset_CE);
end% end loop over Epochs
fprintf(1, 'Finished Training.\n');
fprintf(1, 'Final Training CE%.3f\n', trainset_CE);
fprintf(1, '\rRunning validation ...');
target_vectors_validation =
expansion_matrix(:,valid_labels+1);
datasetsize_validation = size(valid_images, 2);
[hidden_layer_state, output_layer_state] =
forward_propagation...
 (valid_images, input_to_hidden_weights,
hidden_to_output_weights,...
 hidden_bias, output_bias);
```

```
CE = -sum(sum(target_vectors_validation .*
log(output_layer_state + tiny)))...
 /datasetsize_validation;
fprintf(1, '\rFinal Validation CE%.3f\n', CE);
%% Evualuar Test_set
fprintf(1, '\rRunning test ...');
target_vectors_validation =
expansion_matrix(:,test_labels+1);% +1 avoiding zero indices
datasetsize_test = size(test_images, 2);
[hidden_layer_state, output_layer_state] =
forward_propagation...
 (test_images, input_to_hidden_weights,
hidden_to_output_weights,...
 hidden_bias, output_bias);

CE = -sum(sum(target_vectors_validation .*
log(output_layer_state + tiny)))...
 /datasetsize_test;
fprintf(1, '\rFinal Test CE%.3f\n', CE);
%%
end_time = clock;
diff = etime(end_time, start_time);
fprintf(1, 'Training took%.2f seconds\n', diff);
%% SAVE model
model.input_to_hidden_weights = input_to_hidden_weights;
model.hidden_to_output_weights = hidden_to_output_weights;
model.hidden_bias = hidden_bias;
model.output_bias = output_bias;
save model model
plot_CE(CE_array,CE_array_avg,CE_array_valid)
test_accuracy
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%Trazado de dígitos mal clasificados y distribuciones
%de probabilidad de salida correspondientes
```

```
clc
clear all
model_name = 'model.mat';
try load(model_name)
catch
 load model_9896
 disp('model.mat not found, model_9896.mat evaluated
instead.')
end
%load test_set.mat
load example.mat
%% selección de elementos mal clasificados
count=0;
misclass = zeros(1, size(test_images,2));
output = zeros(10, size(test_images,2));
for i=1:size(test_images,2)

 input = test_images (:,i);

 [hidden_layer_state, output_layer_state] =
forward_propagation...
 (input, model.input_to_hidden_weights,
model.hidden_to_output_weights,...
 model.hidden_bias, model.output_bias);

 [prob, indices] = sort(output_layer_state, 'descend');
 indices = indices-1;
 output(:,i) = output_layer_state;
 if indices(1) ~= test_labels(i)
 count = count+1;
 misclass(i) = 1;
 end
end
missed_images = test_images(:,misclass>0);
missed_output = output(:,misclass>0);
```

```
missed_labels = test_labels(:,misclass>0);
for i=1:size(missed_images,2)
clc
fprintf(1, 'Number of inorrectly classified images on test
set:%.3i \n', count);
fprintf(1,'%i/%i \n\n',count,i)
figure(1)
subplot(1,2,1)
digit = missed_images(:,i);
kep=reshape(digit,28,28);
colormap(bone);
clims = [-0.3 0.3];
imagesc(1-kep,clims)
colormap(bone)
title('Misclassified
digit','interpreter','latex','fontsize',12)
axis equal
% Salida de probabilidades
subplot(1,2,2)
x = 0:1:9;
y = missed_output(:,i);
bar(x,y)
title('Output
probabilities','interpreter','latex','fontsize',12)
fprintf(1,'La respuesta correcta es :%i
\n\n',missed_labels(i))
disp('Paused')
pause
end
```