Implementación del Algoritmo de Back-Propagation Para una Red Neuronal de Clasificación Multi-Clase

Carlos Santana Esplá y
Ricardo Juan Cárdenes Pérez

Noviembre, 2023

Resumen

Presentamos en este trabajo las bases matemáticas para implementar el algoritmo de retropropagación, utilizado en aprendizaje profundo para el entrenamiento de modelos basados en redes neuronales, para una red neuronal fully-connected. También ofrecemos dos formas de implementar dicho algoritmo, utilizando los lenguajes de programación Python y Matlab, obteniendo una precisión superior a 92 % para el conjunto de datos MNIST, para una arquitectura con una capa oculta de 10 neuronas.

1. Introducción

Desde sus primeros pasos en la década de 1970 hasta su integración en arquitecturas neuronales más complejas en la actualidad, el algoritmo de retropropagación ha desempeñado un papel esencial en la evolución y mejora continua de las capacidades de las redes neuronales. Durante sus primeras instancias, el backpropagation sentó las bases para el desarrollo de sistemas de aprendizaje automático, proporcionando un mecanismo eficiente para ajustar los pesos y sesgos de las redes en función de los errores de predicción. A medida que la investigación avanzaba, este algoritmo se hizo crucial para superar desafíos en la captura de patrones complejos en conjuntos de datos de alta dimensionalidad.

Con el tiempo, el backpropagation ha evolucionado, y se ha adaptado para su implementación en arquitecturas neuronales más avanzadas, permitiendo la construcción de modelos más profundos y sofisticados, como se hará en este artículo. Su contribución se ha vuelto aún más significativa a medida que las aplicaciones de inteligencia artificial se expanden a campos como el reconocimiento de imágenes, el procesamiento de lenguaje natural y la conducción autónoma, donde la capacidad de las redes para discernir patrones sutiles y abordar problemas complejos es esencial. En este contexto, el backpropagation se destaca como una herramienta fundamental que ha allanado el camino para el

desarrollo y despliegue exitoso de diversas aplicaciones de inteligencia artificial.

Inspirado por los principios del cálculo diferencial, este método comienza dada una inicialización aleatoria de pesos y sesgos de una red neuronal, y a través de la propagación hacia adelante, las entradas se procesan capa a capa, aplicando sobre ellas funciones de activación, generando predicciones. Luego, se evalúa la discrepancia entre estas predicciones y las salidas reales a través de una función de pérdida, lo cual se verá más en detalle en la sección 3. La retropropagación del error sigue, calculando las derivadas parciales de la función de pérdida respecto a los pesos y sesgos en cada capa, permitiendo ajustar los parámetros de la red mediante técnicas de optimización, como detallaremos más adelante.

2. Arquitectura de red Fully-Connected

Supongamos que tenemos una red neuronal FC compuesta por l capas. Cada una de las capas, ocultas o de salida, computará una combinación lineal con las salidas de la capa anterior, las cuales, para la capa k, se denotan con el vector $A^{[k-1]}$. Para ello, cada neurona i de la capa k contará con un vector de pesos $w_i = (w_{1i} \cdots w_{n(k-1),i}) \in \mathbb{R}^{n(k-1)}$, donde $n(k-1)$ representa el número de neuronas de la capa anterior, y un sesgo o bias $b_i^{[k]}$. Estos, en conjunto, actuarán como los coeficientes en dicha operación. A su salida, se le aplicará una no linealidad que será general para toda la capa, evitando que la red colapse en una única combinación lineal compuesta de las entradas. Así, tenemos que la salida de la neurona en cuestión para la capa arbitraria escogida no es más que $A_i^{[k]}$, donde

$$z_i^{[k]} = w_i^{[k]} \cdot A^{[k-1]} + b_i^{[k]} \tag{1}$$

$$A_i^{[k]} = g_k\left(z_i^{[k]}\right) \tag{2}$$

Siendo $g_k : \mathbb{R} \to \mathbb{R}$ la función de activación definida para dicha capa y $z_i^{[k]}$ la combinación lineal descrita. La notación utilizada para los pesos, algo que cobrará gran importancia en las siguientes secciones, es $w_{ji}^{[k]}$, donde los índices i y j representan la neurona de entrada en la capa k, y la de salida en la capa $k-1$, respectivamente.

Podemos calcular así para una muestra cualquiera $x \in \mathbb{R}^n$ su salida en la red como

$$A^{[l]} = g_l(w^{[l]} \cdot (g_{l-1}(w^{[l-1]} \cdot (\cdots(g_1(w^{[1]}x^t + b^{[1]}))) + b^{[l-1]})) + b^{[l]}) \tag{3}$$

Hemos definido así el feedfoward de nuestra red. Ahora bien, ¿cómo podemos optimizar esta red para que la salida se ajuste correctamente a la realidad de los datos?

3. Optimización de la red neuronal

Usaremos la función de error de Cross-Entropy, la cual resulta ser de las mejores opciones a la hora de desarrollar redes de clasificación, aunque otras como MSE nos traen al mismo resultado ajustando adecuadamente las constantes implicadas. La función se define, para una muestra $X \in \mathbb{R}^n$ y su etiqueta $Y \in \{0,1\}^C$, como

$$\mathcal{L}(X,Y) = -\sum_{i=1}^{C} y_i \cdot log(A_i^{[l]}) \tag{4}$$

donde $A_i^{[l]}$ es la salida de la red para la $i-$ésima clase de la muestra X. Ante la posibilidad de trabajar en un espacio muestral con múltiples clases, usamos la función softmax como función de activación para la capa de salida. Esto es, sea $z^{[l]}$ el vector que contiene los cálculos lineales realizados en dicha capa, la salida de la misma se define como

$$A^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{C} e^{z_j^{[l]}}} \tag{5}$$

Sustituyendo este vector de salida en la función de coste dada en la ecuacion (4), obtenemos que

$$\mathcal{L}(X,Y) = -\sum_{i=1}^{C} y_i \cdot log\left(\frac{e^{z_i^{[l]}}}{\sum_{j=1}^{C} e^{z_j^{[l]}}}\right) \tag{6}$$

Y al aplicar las propiedades básicas del logaritmo

$$\mathcal{L}(X,Y) = -\sum_{i=1}^{C} y_i \left(log\left(e^{z_i^{[l]}}\right) - log\left(\sum_{j=1}^{C} e^{z_j^{[l]}}\right)\right) \tag{7}$$

El algoritmo de retropropagación (Back-Propagation) se fundamenta en el cálculo de las derivadas del error con respecto a los diferentes pesos de la red. Estas derivadas se utilizan en técnicas de descenso del gradiente para actualizar los pesos, uno por uno, epoch tras epoch (cada ciclo de entrenamiento). Por consiguiente, procederemos a calcular las derivadas correspondientes a los pesos de la capa de salida. Para lograr esto, aplicaremos la regla de la cadena. Empezamos derivando la función de error con respecto a $z_i^{[k]}$, donde lo primero será aplicar la regla de la derivada para la suma de funciones. Así

$$\frac{\partial \mathcal{L}}{\partial z_i^{[l]}} = -\frac{\partial}{\partial z_i^{[l]}}\left(y_i \cdot log\left(e^{z_i^{[l]}}\right)\right) + \sum_{k=1}^{C} \frac{\partial}{\partial z_i^{[l]}}\left(y_k \cdot log\left(\sum_{j=1}^{C} e^{z_j^{[l]}}\right)\right) \tag{8}$$

Veáse que en el primer término de la derivada nos hemos quedado únicamente con el término del sumatorio original que depende del parámetro respecto al cual estamos derivando. Centrándonos en dicho término, vemos que

$$-\frac{\partial}{\partial z_i^{[l]}}\left(y_i \cdot log\left(e^{z_i^{[l]}}\right)\right) = -\frac{\partial}{\partial z_i^{[l]}}\left(y_i z_i^{[l]}\right) = -y_i \tag{9}$$

Por otra parte,

$$\sum_{k=1}^{C}\frac{\partial}{\partial z_i^{[l]}}\left(y_k \cdot log\left(\sum_{j=1}^{C} e^{z_j^{[l]}}\right)\right) = \sum_{k=1}^{C} y_k \frac{e^{z_i^{[l]}}}{\sum_{j=1}^{C} e^{z_j^{[l]}}} = \frac{e^{z_i^{[l]}}}{\sum_{j=1}^{C} e^{z_j^{[l]}}}\sum_{k=1}^{C} y_k \tag{10}$$

Hemos de tener en cuenta que las etiquetas de las muestras vienen dadas en codificación one-hot, y por tanto $\sum_k y_k = 1$. Bastaría así con aplicar esta propiedad junto con la igualdad (5) para obtener que

$$\sum_{k=1}^{C}\frac{\partial}{\partial z_i^{[l]}}\left(y_k \cdot log\left(\sum_{j=1}^{C} e^{z_j^{[l]}}\right)\right) = A_i^{[l]} \tag{11}$$

Podemos ahora sustituir los resultados (9) y (11) en la expresión (8), de forma que

$$\frac{\partial \mathcal{L}}{\partial z_i^{[l]}} = A_i^{[l]} - y_i \tag{12}$$

Expresado en forma matricial,

$$\frac{\partial \mathcal{L}}{\partial z^{[l]}} = A^{[l]} - Y \tag{13}$$

Teniendo esto claro, uno podría calcular las derivadas del error con respecto a los pesos de la última capa con tan solo aplicar la regla de la cadena, donde se tiene que, por definición de $z^{[l]}$,

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[l]}} = \frac{\partial \mathcal{L}}{\partial z_i^{[l]}}\frac{\partial z_i^{[l]}}{\partial w_{ji}^{[l]}} \tag{14}$$

Teniendo en cuenta la ecuación (1), vemos que para una capa $k \in \{1, ..., l\}$,

$$\frac{\partial z_i^{[k]}}{\partial w_{ji}^{[k]}} = \frac{\partial}{\partial w_{ji}^{[k]}}\left(w_i^{[k]} \cdot A^{[k-1]} + b^{[k]}\right) = \frac{\partial}{\partial w_{ji}^{[k]}}\left(w_{ji}^{[k]} A_j^{[k-1]}\right) = A_j^{[k-1]} \tag{15}$$

Concluimos así que para un peso $w_{ji}^{[l]}$, la deriva de la función de error con respecto a dicho peso no es más que

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[l]}} = \frac{\partial \mathcal{L}}{\partial z_i^{[l]}} \frac{\partial z_i^{[l]}}{\partial w_{ji}^{[l]}} = \left(A_i^{[l]} - y_i\right) A_j^{[l-1]} \tag{16}$$

y, para toda capa $k \in \{1, ..., l\}$,

$$\frac{\partial \mathcal{L}}{\partial b_i^{[k]}} = \frac{\partial \mathcal{L}}{\partial z_i^{[k]}} \tag{17}$$

por lo que

$$\frac{\partial \mathcal{L}}{\partial b_i^{[l]}} = \left(A_i^{[l]} - y_i\right) \tag{18}$$

3.1. Derivadas para parámetros en capas ocultas

Veamos como se comportan las derivadas para la última capa oculta. Comenzaremos derivando con respecto a $z_i^{[l-1]}$ para luego aplicar la regla de la cadena y obtener la derivada respecto a cada uno de sus pesos. Teniendo en cuenta que estamos tratando de derivar la ecuación (6), que consiste en un sumatorio de C términos todos dependientes de $z_i^{[l-1]}$, por tratarse de una red FC, la derivada consistirá en un sumatorio, por la propiedad de la derivada de la suma de funciones, con las derivadas parciales con respecto a $z_i^{[l-1]}$ para cada uno de sus sumandos. Esto es,

$$\frac{\partial \mathcal{L}}{\partial z_i^{[l-1]}} = \sum_{n_l=1}^{C} \frac{\partial \mathcal{L}}{\partial A_{n_l}^{[l]}} \frac{\partial A_{n_l}^{[l]}}{\partial z_{n_l}^{[l]}} \frac{\partial z_{n_l}^{[l]}}{\partial A_i^{[l-1]}} \frac{\partial A_i^{[l-1]}}{\partial z_i^{[l-1]}} \tag{19}$$

Para simplificar los cálculos, supondremos que todas las capas ocultas de la red utilizan una función de activación sigmoide, $\sigma : \mathbb{R} \to \mathbb{R}$, la cual es derivable $\forall x \in \mathbb{R}$ y cumple que $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Gracias a esta suposición, sea $k \in \{1, ..., l-1\}$, sabemos que

$$A_i^{[k]} = \sigma(z_i^{[k]}) \tag{20}$$

Y, consecuentemente,

$$\frac{\partial A_i^{[k]}}{\partial z_i^{[k]}} = \sigma(z_i^{[k]}) \left(1 - \sigma(z_i^{[k]})\right) = A_i^{[k]} \left(1 - A_i^{[k]}\right) \tag{21}$$

Por tanto, al sustituir los resultados obtenidos en las ecuaciones (12) y (21) en la expresión (16), y observar que la derivada parcial de $z_{n_l}^{[l]}$ con respecto a $A_i^{[l-1]}$ no es más que su coeficiente $w_{i,n_l}^{[l]}$, obtenemos la siguiente expresión

$$\frac{\partial \mathcal{L}}{\partial z_i^{[l-1]}} = \sum_{n_l=1}^{C} \left(A_{n_l}^{[l]} - y_{n_l}\right) w_{i,n_l}^{[l]} A_i^{[l-1]} \left(1 - A_i^{[l-1]}\right) \tag{22}$$

Si ahora queremos la derivada con respecto a los pesos de una neurona i de la capa $l-1$, obtenemos, aplicando (14), que

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[l-1]}} = \sum_{n_l=1}^{C} \left(A_{n_l}^{[l]} - y_{n_l}\right) w_{i,n_l}^{[l]} A_i^{[l-1]} \left(1 - A_i^{[l-1]}\right) A_j^{[l-2]} \tag{23}$$

El conjunto MNIST puede ser clasificado con un alto accuracy con una única capa oculta, por lo que ya tendríamos los cálculos necesarios para desarrollar el algoritmo. No obstante, para generalizar este algoritmo a cualquier red FC, necesitamos saber que ocurre con los pesos más allá de la penúltima capa.

Veamos así que sucede para la capa $l-2$. Para ello, hemos de tener en cuenta que, dado un peso $w_{ji}^{[l-2]}$, su valor únicamente pondera a la salida de la neurona j de la capa $l-3$, como entrada a la neurona i de la capa $l-2$. Ahora bien, al tratarse de una red neuronal FC, todas las neuronas de la capa $l-1$ estarán conectadas con todas las de la capa $l-2$. Por tanto, el peso $w_{ji}^{[l-2]}$ afectará a todas las salidas de esta $l-1$. Es por esta razón por la que aparece un nuevo sumatorio en la expresión, ya que todas las $z_i^{[l]}$ consisten en una suma ponderada de las distintas coordenadas del vector $A^{[l-1]}$, las cuales dependen todas de dicho peso y, por tanto, actúara la regla de la derivada para la suma de funciones sobre ellas. Esto es,

$$\frac{\partial}{\partial w_{ji}^{[l-2]}} \left(w_i^{[l]} \cdot A^{[l-1]}\right) = \sum_{n_{l-1}=1}^{n(l-1)} \frac{\partial}{\partial w_{ji}^{[l-2]}} \left(w_{n_{l-1},i}^{[l]} A_{n_{l-1}}^{[l-1]}\right) \tag{24}$$

Así, cobra más sentido el siguiente resultado

$$\frac{\partial \mathcal{L}}{\partial z_i^{[l-2]}} = \sum_{n_l=1}^{C} \frac{\partial \mathcal{L}}{\partial A_{n_l}^{[l]}} \frac{\partial A_{n_l}^{[l]}}{\partial z_{n_l}^{[l]}} \sum_{n_{l-1}=1}^{n(l-1)} \frac{\partial z_{n_l}^{[l]}}{\partial A_{n_{l-1}}^{[l-1]}} \frac{\partial A_{n_{l-1}}^{[l-1]}}{\partial z_{n_{l-1}}^{[l-1]}} \frac{\partial z_{n_{l-1}}^{[l-1]}}{\partial A_i^{[l-2]}} \frac{\partial A_i^{[l-2]}}{\partial z_i^{[l-2]}} \tag{25}$$

pues no es más que

$$\frac{\partial \mathcal{L}}{\partial z_i^{[l-2]}} = \sum_{n_l=1}^{C} \frac{\partial \mathcal{L}}{\partial A_{n_l}^{[l]}} \frac{\partial A_{n_l}^{[l]}}{\partial z_{n_l}^{[l]}} \sum_{n_{l-1}=1}^{n(l-1)} \frac{\partial}{\partial w_{ji}^{[l-2]}} \left(w_{n_{l-1},i}^{[l]} A_{n_{l-1}}^{[l-1]}\right) \tag{26}$$

3.2. Generalización de las derivadas

Uno podría seguir calculando las derivadas para capas inferiores, y observaría un patrón que se repite. Existirían $l-k$ sumatorios anidados en la expresión, uno por cada capa comprendida entre la capa k que se esté evaluando y la de salida, incluyendo a esta. Los $l-k-1$ sumatorios correspondientes a las capas $\{l, l-1, ..., k+2\}$ computarían la derivada del cálculo lineal que hace cada una de las neuronas de dicha capa con respecto a la salida de la capa anterior, multiplicado por su sumatorio. Esto, en conjunto, supone la derivada

de la función de error con respecto al cálculo lineal realizado en la capa $k+1$, tras aplicar sucesivas veces la regla de la cadena. Es decir,

$$S_{l:(k+2)} = \sum_{j=1}^{C} \frac{\partial \mathcal{L}}{\partial A_j^{[l]}} \frac{\partial A_j^{[l]}}{\partial z_j^{[l]}} \sum_{n_1=1}^{n(l-1)} \frac{\partial z_j^{[l]}}{\partial A_{n_1}^{[l-1]}} \frac{\partial A_{n_1}^{[l-1]}}{\partial z_{n_1}^{[l-1]}} \cdots \sum_{n_{k+2}=1}^{n(k+2)} \frac{\partial z_{n_{k+3}}^{[k+3]}}{\partial A_{n_{k+2}}^{[k+2]}} \frac{\partial A_{n_{k+2}}^{[k+2]}}{\partial z_{n_{k+2}}^{[k+2]}} \tag{27}$$

El sumatorio de la capa $k+1$ calcula la derivada de cada $z_{n_{k+2}}^{[k+2]}$ con respecto al $z_i^{[k]}$ que utiliza el peso $w_{ji}^{[k]}$ que nos intersea, que se reduce a lo siguiente

$$\frac{\partial \mathcal{L}}{\partial z_i^{[k]}} = S_{l:(k+2)} \cdot \sum_{n_{k+1}=1}^{n(k+1)} \frac{\partial z_{n_{k+2}}^{[k+2]}}{\partial A_{n_{k+1}}^{[k+1]}} \frac{\partial A_{n_{k+1}}^{[k+1]}}{\partial z_{n_{k+1}}^{[k+1]}} \frac{\partial z_{n_{k+1}}^{[k+1]}}{\partial A_i^{[k]}} \frac{\partial A_i^{[k]}}{\partial z_i^{[k]}} \tag{28}$$

Finalmente, sea la k la capa en la que se encuentra un peso respecto al cual queremos calcular la derivada, tenemos que

$$\frac{\partial \mathcal{L}}{\partial z_i^{[k]}} = \sum_{j=1}^{C} \frac{\partial \mathcal{L}}{\partial A_j^{[l]}} \frac{\partial A_j^{[l]}}{\partial z_j^{[l]}} \sum_{n_1=1}^{n(l-1)} \left(\cdots \sum_{n_{k+1}=1}^{n(k+1)} \frac{\partial z_{n_{k+2}}^{[k+2]}}{\partial A_{n_{k+1}}^{[k+1]}} \frac{\partial A_{n_{k+1}}^{[k+1]}}{\partial z_{n_{k+1}}^{[k+1]}} \frac{\partial z_{n_{k+1}}^{[k+1]}}{\partial A_i^{[k]}} \frac{\partial A_i^{[k]}}{\partial z_i^{[k]}} \right) \tag{29}$$

Y, consecuentemente, la derivada del error con respecto a dicho peso no es más que

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[k]}} = \sum_{j=1}^{C} \frac{\partial \mathcal{L}}{\partial A_j^{[l]}} \frac{\partial A_j^{[l]}}{\partial z_j^{[l]}} \sum_{n_1=1}^{n(l-1)} \left(\cdots \sum_{n_{k+1}=1}^{n(k+1)} \frac{\partial z_{n_{k+2}}^{[k+2]}}{\partial A_{n_{k+1}}^{[k+1]}} \frac{\partial A_{n_{k+1}}^{[k+1]}}{\partial z_{n_{k+1}}^{[k+1]}} \frac{\partial z_{n_{k+1}}^{[k+1]}}{\partial A_i^{[k]}} \frac{\partial A_i^{[k]}}{\partial z_i^{[k]}} A_j^{[k-1]} \right) \tag{30}$$

4. Algoritmo de Back-Propagation

El algoritmo de retropropagación es fundamental en el aprendizaje supervisado de las redes neuronales, pues desencadena un proceso esencial para la optimización de los parámetros de la red. Al abordar la complejidad inherente de la optimización, la retropropagación calcula las derivadas parciales de la función de pérdida con respecto a cada peso y sesgo en la red neuronal. Este cálculo preciso de las derivadas se lleva a cabo, generalmente, mediante el uso de la notación matricial, un enfoque que simplifica la representación y manipulación de las operaciones matemáticas implicadas, ya desarrolladas en la sección 3.

4.1. Notación matricial para el Back-Propagation

A la hora de implementar estas derivadas para que puedan calcularse de forma sencilla en un algoritmo de aprendizaje, se hace impresindible buscar

una manera de emcapsular los cálculos en alguna estructura que nos permita reducir la notación planteada. Para los pesos de la primera capa, por ejemplo, cuyas derivadas vienen expresadas de forma genérica en la expresión (16), uno tenría que tener en cuenta las ixj combinaciones de subíndices posibles, a fin de calcular todas las derivadas necesarias para la corrección de errores en la capa de salida. Esto podría resolverse cómodamente haciendo uso del producto de matrices.

Por ser la matriz de pesos de la capa de salida, $W^{[l]}$, de dimensión $n(l)$ x $n(l-1)$, tenemos que $\frac{\partial \mathcal{L}}{\partial W^{[l]}}$, que no es más que la derivada parcial del error con respecto a dicha matriz, es también una matriz de orden $n(l)$ x $n(l-1)$. En ella, cada entrada d_{ji} nos indica la derivada del error con respecto al peso $w_{ji}^{[l]}$.

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}^{[l]}} & \frac{\partial \mathcal{L}}{\partial w_{12}^{[l]}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{1,n(l-1)}^{[l]}} \\ \frac{\partial \mathcal{L}}{\partial w_{21}^{[l]}} & \frac{\partial \mathcal{L}}{\partial w_{22}^{[l]}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{2,n(l-1)}^{[l]}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{n(l),1}^{[l]}} & \frac{\partial \mathcal{L}}{\partial w_{n(l),2}^{[l]}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{n(l),n(l-1)}^{[l]}} \end{pmatrix} \tag{31}$$

Aplicando las propiedades de matrices, y apoyándonos en la ecuación (16), podemos expresar dicha derivada en la siguiente forma matricial

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial z_1^{[l]}} \\ \frac{\partial \mathcal{L}}{\partial z_2^{[l]}} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial z_{n(l)}^{[l]}} \end{pmatrix} \cdot \begin{pmatrix} A_1^{[l-1]} & A_2^{[l-1]} & \cdots & A_{n(l-1)}^{[l-1]} \end{pmatrix} = \frac{\partial \mathcal{L}}{\partial z^{[l]}} \cdot (A^{[l-1]})^t \tag{32}$$

Hemos conseguido no solo reducir la notación, sino simplificar la implementación del cálculo, lo cual se hace imprescindible para capas más profundas. Veamos que ocurre para la penúltima capa del modelo. Recordemos la expresión de la derivada del error respecto a sus pesos, dada en la ecuación (23)

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[l-1]}} = \sum_{n_l=1}^{C} \left(A_{n_l}^{[l]} - y_{n_l} \right) w_{i,n_l}^{[l]} A_i^{[l-1]} \left(1 - A_i^{[l-1]} \right) A_j^{[l-2]}$$

Podemos reordenar el segundo miembro, extrayendo del sumatorio aquellos términos que no dependen de su índice. Así,

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{[l-1]}} = A_i^{[l-1]} \left(1 - A_i^{[l-1]} \right) A_j^{[l-2]} \sum_{n_l=1}^{C} \left(A_{n_l}^{[l]} - y_{n_l} \right) w_{i,n_l}^{[l]} \tag{33}$$

Y nos bastaría con hacer uso del producto matricial para llegar al siguiente resultado

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ji}^{[l-1]}} &= A_i^{[l-1]} \left(1 - A_i^{[l-1]} \right) A_j^{[l-2]} w_i^{[l]} \left(A^{[l]} - y \right) \\ &= A_i^{[l-1]} \left(1 - A_i^{[l-1]} \right) w_i^{[l]} \left(A^{[l]} - y \right) A_j^{[l-2]} \end{aligned} \tag{34}$$

Aplicando la misma estrategia seguida para los pesos de la primera capa, obtenemos la siguiente forma matricial para la derivada con respecto a los pesos de la capa $l-1$

$$\frac{\partial \mathcal{L}}{\partial W^{[l-1]}} = \left(\left(A^{[l-1]} \odot \left(1 - A^{[l-1]}\right)\right) \odot \left(W^{[l]}\right)\left(A^{[l]} - y\right)\right)\left(A^{[l-2]}\right)^t \tag{35}$$

y

$$\frac{\partial \mathcal{L}}{\partial b^{[l-1]}} = \left(\left(A^{[l-1]} \odot \left(1 - A^{[l-1]}\right)\right) \odot \left(W^{[l]}\right)\left(A^{[l]} - y\right)\right) \tag{36}$$

De forma análoga, es posible representar matricialmente las derivadas para el resto de capas ocultas. No obstante, su formulación matemática formará parte de la segunda entrega.

4.2. Implementación del algoritmo

Implementaremos el algoritmo para una red neuronal de apenas dos capas, una oculta y una de salida, lo cual resulta ser suficiente para la clasificación de regiones convexas en el espacio. De esta manera, el algoritmo de back-propagation únicamente necesitará calcular las derivadas dadas por las ecuaciones (18), (32), (35) y (36). Dispondremos, por tanto, del siguiente algoritmo

Algorithm 1: Algoritmo de Back-Propagation

Data: Muestras y etiqueta para entrenar $X \in \mathbb{R}^n$, $Y \in \{0,1\}^C$
Result: Derivadas $\frac{\partial \mathcal{L}}{\partial W^{[l]}}$, $\frac{\partial \mathcal{L}}{\partial b^{[l]}}$, $\frac{\partial \mathcal{L}}{\partial W^{[l-1]}}$, $\frac{\partial \mathcal{L}}{\partial b^{[l-1]}}$ en (X, Y)
$A^{[l]} \leftarrow$ Resultado de la red para la muestra X ;
$\frac{\partial \mathcal{L}}{\partial z^{[l]}} \leftarrow A^{[l]} - Y$;
$\frac{\partial \mathcal{L}}{\partial b^{[l]}} \leftarrow \frac{\partial \mathcal{L}}{\partial b^{[l]}}$;
$\frac{\partial \mathcal{L}}{\partial W^{[l]}} \leftarrow \frac{\partial \mathcal{L}}{\partial z^{[l]}} \cdot \left(A^{[l-1]}\right)^t$;

$A^{[l-2]} \leftarrow$ Resultado de la capa $l-2$ para la muestra X ;
$\frac{\partial A^{[l-1]}}{\partial z^{[l-1]}} \leftarrow g'_{l-1}(z^{[l-1]})$;
$\frac{\partial \mathcal{L}}{\partial z^{[l-1]}} \leftarrow \frac{\partial A^{[l-1]}}{\partial z^{[l-1]}} \odot \left(W^{[l-1]}\right)^t \cdot \frac{\partial \mathcal{L}}{\partial z^{[l]}}$;
$\frac{\partial \mathcal{L}}{\partial b^{[l-1]}} \leftarrow \frac{\partial \mathcal{L}}{\partial z^{[l-1]}}$;
$\frac{\partial \mathcal{L}}{\partial W^{[l-1]}} \leftarrow \frac{\partial \mathcal{L}}{\partial z^{[l-1]}} \cdot \left(A^{[l-2]}\right)^t$;

return *Lista con las derivadas* $\frac{\partial \mathcal{L}}{\partial W^{[l]}}, \frac{\partial \mathcal{L}}{\partial b^{[l]}}, \frac{\partial \mathcal{L}}{\partial W^{[l-1]}}, \frac{\partial \mathcal{L}}{\partial b^{[l-1]}}$

La ventaja principal que nos ofrece esta implementación es que uno podría aplicar cualquiera de los métodos basados en descenso por el gradiente que existan, ya sea bien el descenso por el gradiente estocástisco, SGD, o el mismo Adam. En la implementación realizada en Python, se presentan varias alternativas, cuyos rendimientos se mostrarán más adelante.

5. Implementación de la red neuronal en Python

En la implementación en Python hemos decidido seguir el paradigma de programación orientada a objetos con el objetivo de desarrollar un código que siga los principios de código limpio, de forma que facilite su mejora para la siguiente entrega. En su arquitectura, la clase 'SequentialLayer' encapsula los componentes esenciales de una capa en la red neuronal. Sus atributos comprenden el número de neuronas de dicha capa (units), las matrices de pesos (weights) y los sesgos de cada neurona (bias), así como funciones de activación y sus derivadas (activation y activation derivative). Estos elementos definen la estructura y comportamiento de cada capa.

Por otra parte, la clase 'NeuralNetwork' administra la conexión y operación de estas capas. Su método 'set(layer)' agrega capas a la red. La función 'feedforward(X, n)' realiza la propagación hacia adelante, aplicando sucesivamente las funciones de activación de cada capa hasta la capa n (o la última capa por defecto). El método 'backpropagation(X, y, lr)' implementa la retropropagación para actualizar los pesos y sesgos de la red.

5.1. Optimizadores

En esta implementación se incluye una serie de funciones encargadas única y exclusivamente de optimizar nuestro modelo, haciendo uso del backpropagation que ofrece la clase 'NeuralNetwork', y que aplican algunos de los métodos de optimización ya vistos, como son: descenso del gradiente, descenso del gradiente con momento, y adam. Su estructura algorítmica podría resumirse en lo siguiente.

Algorithm 2: Algoritmo base para optimizadores

Data: Una instancia de la clase NeuralNetwork, modelo, un conjunto de entrenamiento, $L = \{(x, y) : x \in \mathbb{R}^n, y \in \{0, 1\}^C\}$, un factor de aprendizaje, α, y un número de épocas, epochs

for *cada epoch* $\in [1, epochs]$ **do**
  for *cada* $(x, y) \in L$ **do**
    AlgoritmoDeOptimizacion(modelo,x,y,α);
    $\alpha \leftarrow$ ActualizacionLearningRate(α);

5.2. Rendimiento del Modelo

A continuación, presentamos las medidas de accuracy obtenidas durante el entrenamiento de una red neuronal de 4 entradas, una capa oculta de 10 neuronas con una función de activación sigmoide, y una capa de salida de clasificación con 3 neuronas. Se han entrenado 3 modelos idénticos con los optimizadores del descenso del gradiente clásico, descenso del gradiente con momento, y adam.

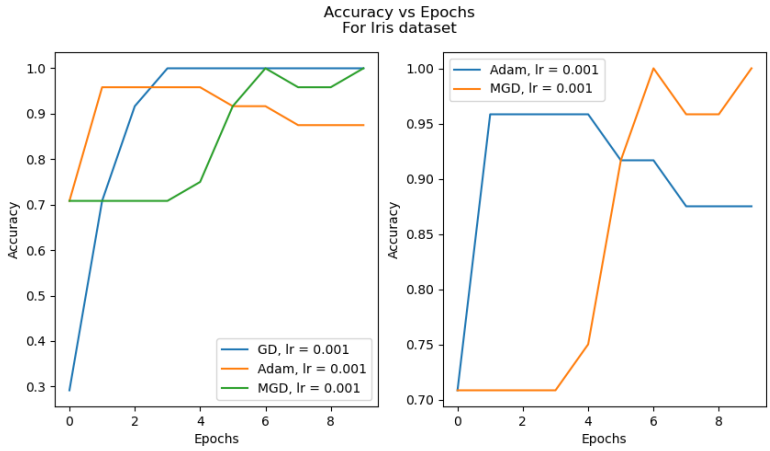

Figura 1: Accuracy en la clasificación del dataset Iris.

Presentamos, además, los resultados obtenidos para una red de 748 entradas, una capa oculta de 100 neuronas con función de activación sigmoide, y 10 salidas, con el objetivo de clasificar el dataset MNIST.

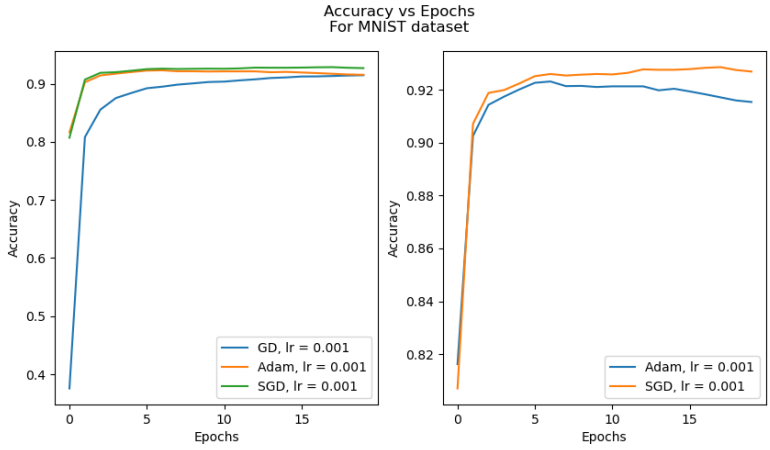

Figura 2: Accuracy en la clasificación del dataset MNIST.

Podemos observar como, para ambos modelos y conjuntos de datos, los optimizadores adam y descenso del gradiente con momento alcanzan un mayor accuracy en una menor cantidad de tiempo, algo que se apreciaría más notablemente al tomar una frecuencia de muestreo mayor en las distinta épocas. No obstante, el algoritmo del descenso por el gradiente clásico también consigue llegar al resultado deseado, aunque toma un mayor número de épocas.

Podemos además ver como varía el accuracy a medida que vamos aumentando el número de neuronas en la capa oculta de nuestra red. Lo probamos principalmente para el dataset Iris, obteniendo el siguiente gráfico sobre su evolución.

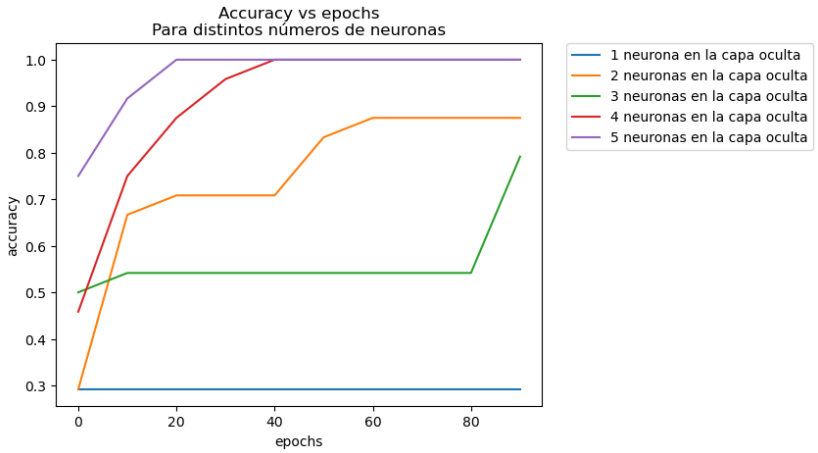

Figura 3: Accuracy para Iris con distintos número de neuronas.

Podemos ver, además, la disminución del error para los distintas número de neuronas en dicha capa, donde se ha tomado la función de error por defecto: Cross-Entropy.

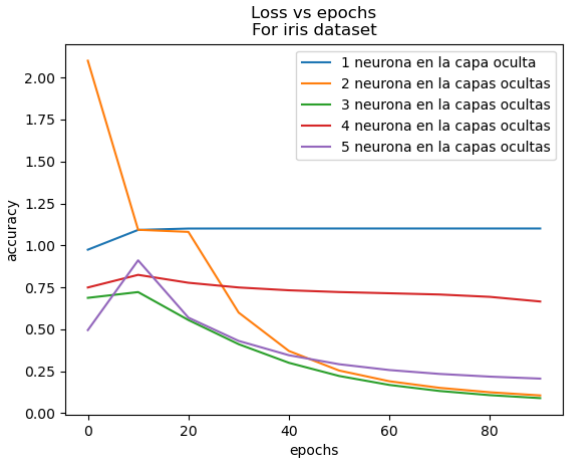

Figura 4: Error para Iris con distintos número de neuronas.

Para la siguiente entrega, se presentará un gráfico similar para el conjunto de datos MNIST, variando además el número de capas ocultas del modelo.

6. Implementación de la red neuronal en Matlab

El código realizado en Matlab refleja la misma arquitectura y lógica que la implementación en Python, utilizando conceptos similares para representar las capas y operaciones en la red neuronal. En esta implementación, se desarrolló todo el proceso de optimización en el mismo método del back-propagation, lo que nos hizo darnos cuenta de la necesidad de separar ambas funcionalidades. Esto es, un método de back-propagation para calcular las derivadas, y un método de óptimización iterativo que hace uso de ellas en cada iteración. Gracias a esta implementación, terminamos desarrollando la implementación en Python ya mencionada, prestando especial atención al principio de single responsability.

7. Conclusión

En conclusión, el algoritmo de retropropagación (Back-Propagation) desempeña un papel crucial en el de las redes neuronales, proporcionando un mecanismo eficiente para ajustar los parámetros de la red y mejorar su capacidad de generalización. A través de la aplicación de la regla de la cadena, el método de backpropagation permite calcular de manera sistemática las derivadas del error con respecto a los pesos y sesgos de cada capa, permitiendo así la optimización de la función de pérdida y obteniendo un mayor accuracy.

La notación matricial utilizada en este trabajo simplifica enormemente la implementación del algoritmo, haciendo que el cálculo de las derivadas sea más accesible y eficiente, especialmente en redes neuronales con múltiples capas. Esta eficiencia es esencial a medida que se aumenta la complejidad de las arquitecturas de red.

En última instancia, el método de retropropagación no solo ha demostrado ser una herramienta esencial para la optimización de redes neuronales, sino que también ha contribuido significativamente al avance de la inteligencia artificial al facilitar la creación de modelos más precisos y complejos. Su aplicación permite a las redes aprender patrones y representaciones útiles a partir de conjuntos de datos, allanando el camino para aplicaciones prácticas en diversas disciplinas, desde reconocimiento de imágenes hasta procesamiento de lenguaje natural.

Future Work

Para la siguiente entrega, el objetivo principal será la generalización de la red neuronal desarrollada para que sea adaptable a arquitecturas con múltiples clases ocultas, lo que permitiría resolver cualquier problema de clasificación que sea posbile resolver con redes Fully Connected. Se añadirán a su vez nuevos optimizadores, con el objetivo de comparar la eficacia de los distintos algoritmos de optimización basados en el descenso del gradiente en el ámbito del aprendizaje automático.

Referencias

[1] Cárdenes Pérez, Ricardo J. (2023) Neural Network Python Implementation https://github.com/ricardocardn/NeuralNetworkImplementation

[2] Santana Esplá, Carlos (2023) Neural Network Matlab Implementation https://github.com/carlos-esplaa/Backpropagation