UNSAM 2024

Matemática III

Trabajo práctico final:
Desarrollo de Red Neuronal

Profesores: Tomas Prudente y Josefina Bompensieri

Integrantes: Martin Schubert

Fecha de entrega: 01/11/24

Parte 1:

Introducción:

Este informe presenta el desarrollo de una red neuronal para clasificar dígitos manuscritos del data set MNIST, implementada manualmente utilizando numpy. Se realiza un análisis detallado de la base de datos, describiendo sus características, correlaciones y preparaciones como la normalización de los datos. La red neuronal se diseña con múltiples capas y se entrena mediante retropropagación y descenso de gradiente estocástico, evaluando el rendimiento en términos de precisión y sobreajuste. Además, se implementa una red similar con scikit-learn para comparar su eficiencia, reflexionando sobre las ventajas y desafíos de construir una red desde cero frente al uso de herramientas de machine learning.

1. Descripción de columnas:

Cada columna del data set representa la posición de un píxel (10,000 píxeles en total, correspondientes a 10,000 muestras), y cada fila representa una muestra con 784 píxeles, es decir, una imagen. La última columna contiene el valor de salida, que representa un dígito del 0 al 9. Para facilitar los cálculos, el data set se invertirá, de modo que cada columna representará una muestra y cada fila, un píxel.

En este contexto hay dos tipos de variable:
Variable de entrada (píxeles): Es una variable continua, ya que los valores de cada píxel oscilan generalmente entre 0 y 255, representando distintos niveles de intensidad o luminosidad.
Variable de salida (dígitos): Es una variable categórica, ya que representa clases discretas (del 0 al 9) que corresponden a cada dígito manuscrito que el modelo debe clasificar.

2. Análisis de Correlaciones:

Durante el análisis de correlaciones del data set MNIST, se observó que la mayoría de las correlaciones resultaron en valores NaN. Este fenómeno se debe a que muchas columnas (píxeles) contienen sólo valores constantes, como ceros, lo que impide el cálculo de una correlación válida. Las correlaciones son indefinidas cuando no hay variación en los datos, lo que explica la prevalencia de NaN en la matriz de correlación.

Sin embargo, se identificaron algunas correlaciones significativas entre ciertos píxeles y la columna objetivo con sigo misma. Un caso por ejemplo fue el pixel 28x19, que mostró una correlación positiva con la variable objetivo.

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
label	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.033682	0.027747	0.021902	0.015736	NaN	NaN	NaN	NaN	NaN	NaN
1x1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1x2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1x3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1x4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28x24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28x25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28x26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28x27	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28x28	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3. Análisis de Factibilidad:

Más allá de este resultado de la correlación que pareciera ser no muy alentador para el desarrollo de una red neuronal, sabía que los pixeles que tienen una correlación estarían más en el centro del data set, y además de que se trata de una base de datos muy conocida y ampliamente utilizada en la comunidad de aprendizaje automático, decidí seguir adelante y emplearla de todos modos en mi análisis.

El propósito de entrenar la red neuronal es desarrollar un modelo capaz de reconocer y clasificar imágenes de dígitos manuscritos. En particular, se utiliza el conjunto de datos MNIST, que contiene miles de ejemplos de dígitos del 0 al 9, cada uno representado como una imagen en escala de grises de 28x28 píxeles. Cada imagen se transforma en un vector de características, donde cada elemento del vector corresponde a la intensidad de un píxel.

La red neuronal intentará predecir la clase correspondiente a cada imagen de dígito manuscrito, es decir, identificar correctamente el número que representa. La columna objetivo del conjunto de datos MNIST proporciona la etiqueta asociada a cada imagen, que va del 0 al 9. Así, el modelo aprenderá a mapear las características visuales de los dígitos a su representación numérica.

El objetivo principal del modelo es maximizar la precisión de las predicciones en un conjunto de datos de prueba, es decir, identificar correctamente el dígito que aparece en cada imagen. Para lograr esto, la red neuronal se entrenará mediante el ajuste de sus pesos y sesgos a través de un proceso de retropropagación y descenso por gradiente. A medida que el modelo se entrena, debe aprender a generalizar las características que permiten diferenciar entre los dígitos, incluso ante variaciones en la escritura a mano.

4. Datos Atípicos y Limpieza de Datos:

Esta base de datos no presenta outliers o valores atípicos, ya que ha sido ampliamente utilizada por investigadores y profesionales en el campo del aprendizaje automático. MNIST es un conjunto de datos bien curado y estandarizado, lo que asegura que no contenga registros erróneos o extremos. Por lo tanto, no es necesaria ninguna limpieza de datos en este caso, lo que permite centrarse directamente en el entrenamiento del modelo y en el análisis de su rendimiento.

5. Transformaciones Preliminares:

Para preparar el conjunto de datos MNIST para el entrenamiento de la red neuronal, se realizan varias transformaciones importantes:

1. Normalización de los Datos

La normalización es un paso crucial en el preprocesamiento de datos, especialmente para redes neuronales. En este caso, se utiliza la normalización mediante la derivación estándar (también conocida como z-score normalization). Esta técnica transforma los

datos para que tengan una media de 0 y una desviación estándar de 1. El proceso se realiza de la siguiente manera:

$$Z = \frac{(x - \mu)}{\sigma}$$

donde x es el valor original, μ es la media del conjunto de datos y σ es la desviación estándar.

Beneficios de normalizar:

- Mejora la Convergencia: Al tener características en la misma escala, la red neuronal puede aprender de manera más eficiente, ya que evita que algunas características dominen el proceso de aprendizaje debido a su escala.
- Facilita el Aprendizaje: Las funciones de activación, como la ReLU, funcionan mejor cuando los valores de entrada están normalizados. Esto permite que la red se entrene más rápidamente y con mayor eficacia.

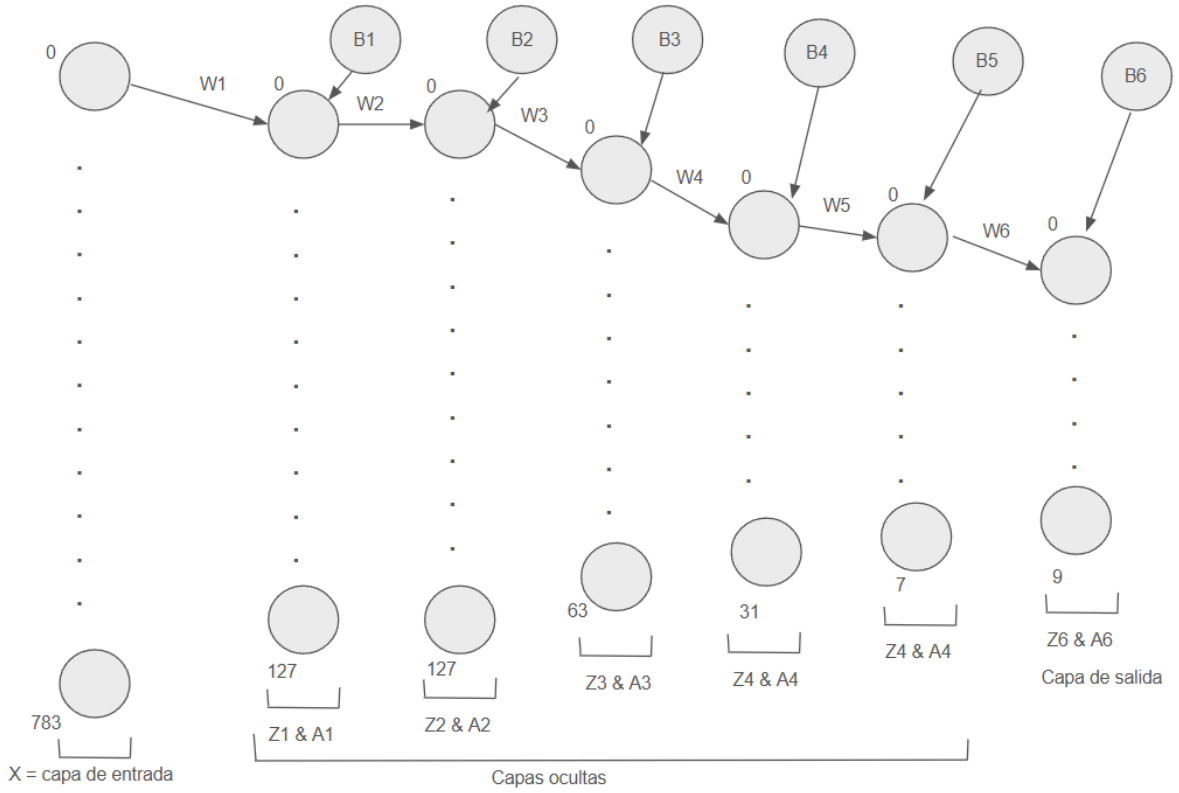
2. One-Hot encoding para la Salida:

Dado que la tarea consiste en clasificar los dígitos del 0 al 9, es fundamental convertir las etiquetas de salida en un formato adecuado para el modelo. El one-hot encoding es una técnica que convierte categorías en vectores binarios, donde cada categoría se representa por un vector en el que solo un elemento es 1 (indica la clase) y los demás son 0. Por ejemplo, el dígito "3" se representaría como: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

Parte 2:

La red contará con un total de 6 capas, sin contar la capa de entrada, 5 de estas serán llamadas capas internas que utilizan como función de activación a ReLU, y la última será la capa de salida que utiliza la función softmax como activación.
El funcionamiento es el siguiente, cuando ingresan los 784 inputs estos serán ingresados a cada neurona de la capa 1, y en cada una están multiplicándose por un peso y sumándose un bias, luego, serán activados con la ReLU y este proceso se repite

con las capas que le sigue, hasta llegar a la última capa, en donde el cambio es la activación que ahora la efectúa la función softmax.



Una vez separada la cantidad de muestras para entrenamiento y para la prueba, se inician los parámetros. Los pesos se inician como valores pseudoaleatorios (para recrear resultados) y los bias en 0, que luego estos se ajustarán mediante back propagation, pero primero se debe pasar por el **forward propagation**, que utiliza las siguientes fórmulas:

$$Z_l = W_l * A_{l-1} + B_l$$

$$A_l = f(Z_l)$$

Donde l es el valor de la capa actual y () es la función de activación de esa capa.

$$ReLU = max\{x, 0\}$$

Devuelve el máximo entre 0 y x

$$Softmax(Z_i) = \frac{e^{Z_i}}{\sum_{j=1}^{k} e^{Zj}}$$

Donde Z_i representa un valor sin activar de la capa de salida, y $\sum_{j=1}^{k} e^{Zj}$ suma de todos los valores de la capa de salida, para devolver el porcentaje que representa Z_i teniendo en cuenta todos los valores que integran la capa de salida.

Después de hallar los valores correspondientes de todos los Z y todos los A, estos se utilizarán en el **back propagation** para poder realizar ese ajuste que buscamos y así la red tenga valores que permitan predecir correctamente. Las fórmulas utilizadas son las siguientes:

$$Loss = -\sum_{i=1}^{n} y_i * log(a_i)$$

donde n = total neuronas de salida e i es la neurona actual.
luego se deriva a_i respecto de z_i, siendo a_i la función softmax lo que nos permite aplicar un logaritmo sobre a_i (para facilitar el proceso de derivación), ya que la softmax nunca dará como resultado 0 o un negativo. Entonces quedaría:

$$\frac{\partial log(a_i)}{\partial z_i} = \frac{1}{a_i} * \frac{\partial a_i}{\partial z_i}$$
$$\frac{\partial a_i}{\partial z_i} = \frac{\partial log(a_i)}{\partial z_i} * a_i$$

El logaritmo nos sirve para el siguiente paso que es hallar $\frac{\partial log(a_i)}{\partial z_i}$:

$$log(a_i) = log(\frac{e^{z_i}}{\sum_{l=1}^{k} e^{z_l}}) = z_i - log(\sum_{l=1}^{k} e^{z_l})$$

$$\frac{\partial log(a_i)}{\partial z_i} = \frac{\partial z_i}{\partial z_j} - \frac{\partial log(\sum_{l=1}^{k} e^{z_l})}{\partial z_j}$$

Cuando $i = j$, luego usando la regla de la cadena y sabiendo que al derivar una sumatoria de e^{Z_l} respecto de z_j todas las otras partes en la sumatoria quedan como 0, lo que nos deja con:

$$\frac{\partial log(a_i)}{\partial z_i} = 1\{i == j\} - \frac{1}{\sum_{l=1}^{k} e^{Z_j}} * \ e^{Z_j}$$

Pero se sabe que $\frac{e^{Z_j}}{\sum_{l=1}^{k} e^{Z_j}} = a_i$ entonces:

$$\frac{\partial log(a_i)}{\partial z_i} = 1\{i == j\} - a_i$$

Con esto resuelto volveos a la ecuación original y reemplazamos por lo hallado

$$\frac{\partial a_i}{\partial z_i} = (1\{i == j\} - a_i) * a_i$$

Ahora lo que queda el hallar la $\frac{\partial L_i}{\partial z_i}$ siendo *L* la fórmula de cross entropy, para eso se usara nuevamente la regla de la cadena

$$\frac{\partial L_i}{\partial z_i} = \frac{\partial L_i}{\partial a_i} * \frac{\partial a_i}{\partial z_i} = -\frac{\sum_{i=1}^{c} y_i}{a_i} * \frac{\partial a_i}{\partial z_i} = -\frac{\sum_{i=1}^{c} y_i}{a_i} * a_i * (1\{i == j\} - a_i)$$

Distribuir la sumatoria de y_i, también se simplifica $\frac{a_i}{a_i}$ que nos da 1

$$\frac{\partial L_i}{\partial z_i} = \sum_{i=1}^{c} y_i * a_i - \sum_{i=1}^{c} y_i * 1\{i == j\}$$

Necesito que se siga cumpliendo la condición de que $i == j$, entonces $\sum_{i=1}^{c} y_i$ me deja con y_j

$$\frac{\partial L_i}{\partial z_i} = \sum_{i=1}^{c} y_i * a_i - y_j$$

$\sum_{i=1}^{c} y_i$ es one hot, por ende, esta sumatoria nos da 1, llegando al resultado final que es:

$$\frac{\partial L_i}{\partial z_i} = a_i - y_j$$

con este resultado ahora se puede proseguir con el back propagation, que cuenta con las siguientes fórmulas:

$$\text{BP1: } \delta^L = \nabla_a C \odot \sigma'(Z^L)$$

Donde $\nabla_a C$ se lo toma como el cambio de C respecto de las activaciones de salida y el segundo termino mide que tan rápido la función σ cambia respecto de Z^L, otra manera sería expresarlo en forma vectorial: $\delta^L = (a^L - y) \odot \sigma'(Z^L)$, lo que nos permitiría expresarlo en numpy más fácilmente.

$$\text{BP2: } \delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(Z^L)$$

Una ecuación para el error de δ^l en base del error de la capa siguiente, donde $(W^{l+1})^T$ es el peso matriz de la capa $l + 1$. Suponiendo que sabemos el error de δ^{l+1}, le aplicaos la matriz del peso transpuesto, y luego tomamos el producto Hadamard (multiplicación elemento a elemento) $\odot \sigma'(Z^L)$. Se puede pensar de una manera que movemos el error hacia atrás en la red. Combinando BP1 y BP2 podemos computar el error para cualquier capa de la red, usamos BP1 para la capa de salida, y luego BP2 para calcular δ^{L-1}, luego nuevamente BP2 para hallar δ^{L-2}, y así sucesivamente para todas las capas.

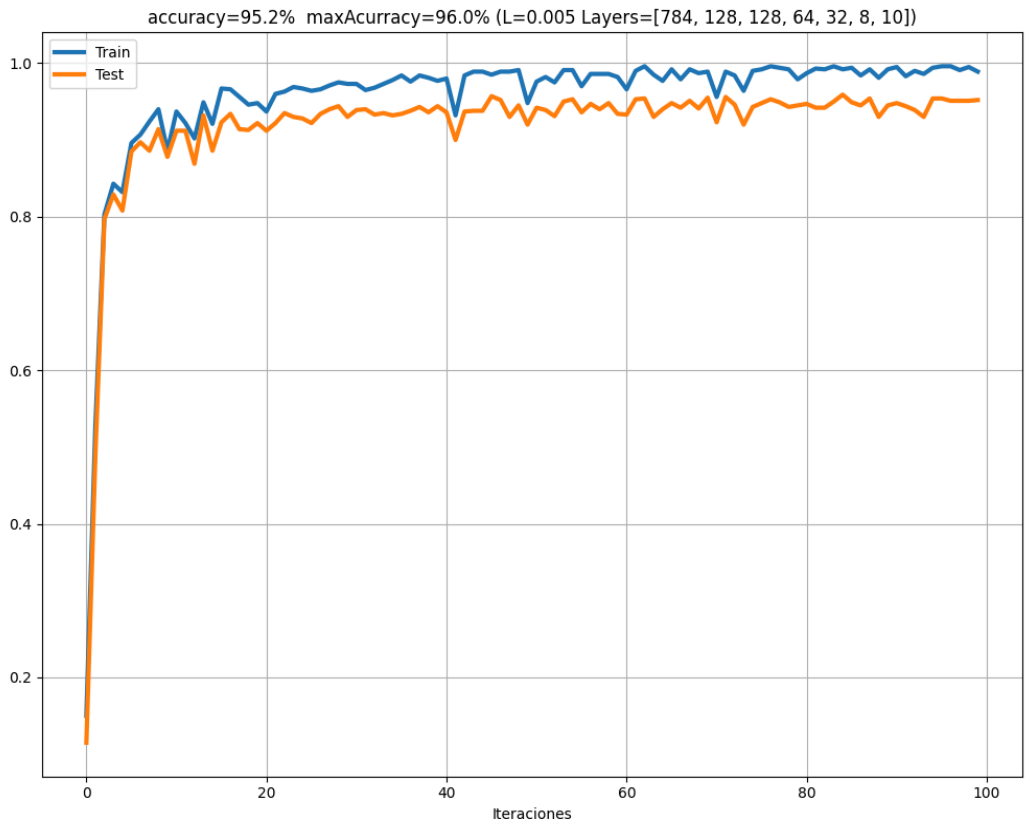
$$\text{BP3: } \frac{\partial C}{\partial b} = \delta$$

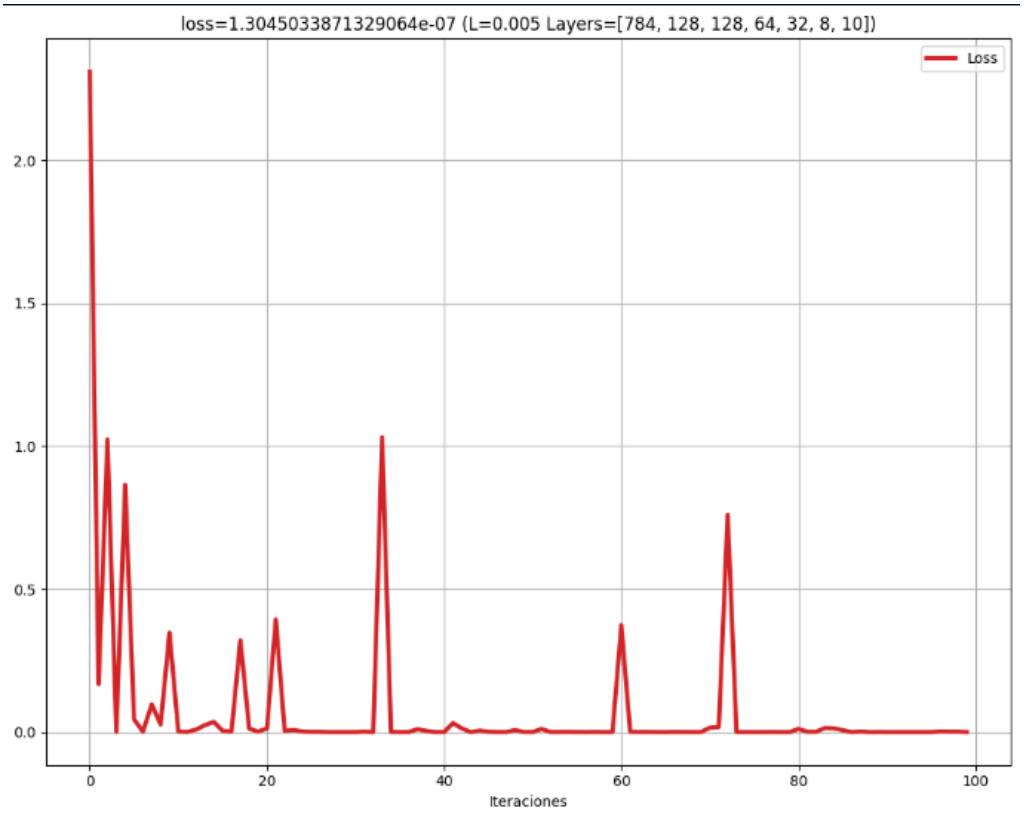
Para hallar el cambio del error con respecto de bias usamos esta ecuación, y como ya se sabe cómo hallar δ, donde se entiende que δ está siendo evaluado en la misma neurona que b

$$\text{BP4: } \frac{\partial C}{\partial W^l} = a^{l-1} * \delta^l$$

Y por último para hallar el cambio del error respecto de los pesos, donde se entiende a^{l-1} como la activación de la neurona de entrada del peso W^l , y δ^l es el error de la neurona de salida del peso W^l

Luego de implementar las bases, se buscó el paso de aprendizaje (L) y la cantidad de iteraciones que me permiten llegar a los mejores resultados, lo que me llevó a decantarme por estos valores, siendo los mas óptimos de los utilizados. Lo que me llevó a reducir el error final a un valor extremadamente más bajo que el inicial, que era lo que se buscaba llevarlo o más posible a 0 que se pueda



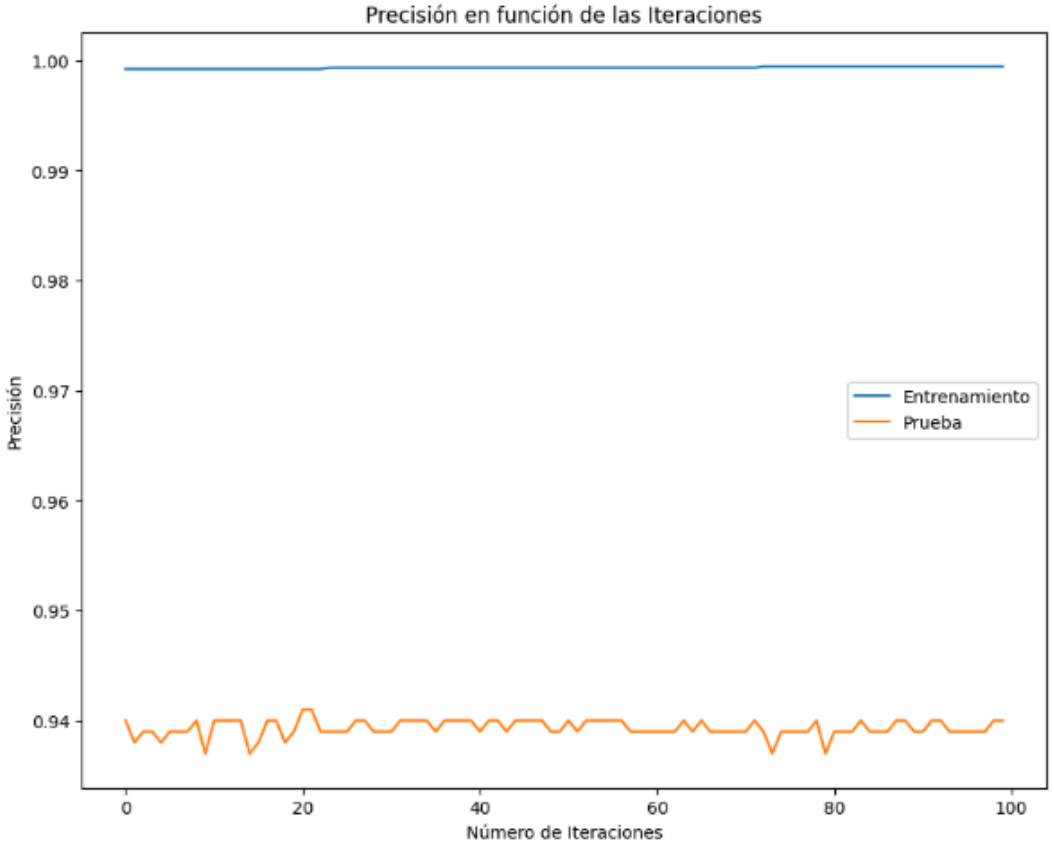


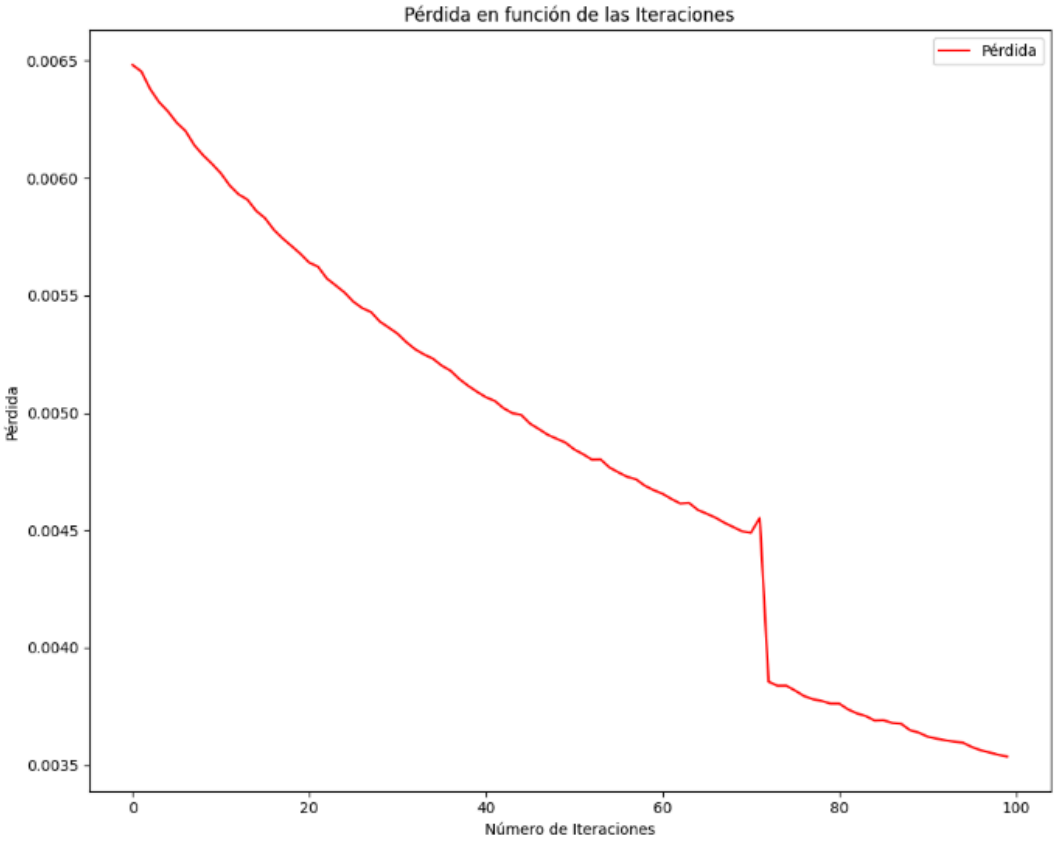
Observación: las iteraciones figuran como 100, pero esto no es así, 100 es la cantidad de veces que se toma una muestra de la precisión dentro del entrenamiento sin importar la cantidad de iteraciones, lo mismo sucede en el caso de la perdida

En el proceso de entrenamiento de esta red neurona, se ha observado que el modelo mantiene un rendimiento consistente en los conjuntos de entrenamiento y de validación, siendo que estos no se desvían el uno del otro en gran medida. Esto indica que la red no está experimentando overfitting.

Parte 3:

Por último, se utilizó scikit-learn para comparar resultados, donde se implementó la misma cantidad de capas y neuronas en cada una de ellas, así también como el paso de aprendizaje y la cantidad de iteraciones.


Precisión en función de las Iteraciones
Precisión
Número de Iteraciones
Entrenamiento
Prueba
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0
20
40
60
80
100


Pérdida en función de las Iteraciones
Pérdida
Número de Iteraciones
Pérdida
0.0065
0.0060
0.0055
0.0050
0.0045
0.0040
0.0035
0
20
40
60
80
100

Observación: pareciera que la red empieza con mas del 90% de precisión, pero esto es porque se utilizo el mismo método que en mi red para que no halla diferencias a la hora del tiempo de ejecución, es decir, scikit-learn alcanza ese porcentaje en las primeras 500 iteraciones que realiza, siendo estas las usadas para la medición 0 del gráfico. De la misma manera sucede en la perdida, que pareciera empezar desde un valor muy bajo.

Tras analizar los resultados, se observa que scikit-learn alcanzó una precisión de entrenamiento del 99.94% y una precisión de prueba del 94%, superando a mi red en entrenamiento, pero con un rendimiento inferior en la prueba. Es importante destacar que, con pocas iteraciones, scikit-learn ya lograba una precisión muy alta, mostrando una capacidad de aprendizaje considerablemente más rápida que la de mi red. Por último, el tiempo de ejecución fue de 84 segundos en scikit-learn frente a 52 segundos en mi implementación, lo cual puede deberse a que scikit-learn realiza procesos de verificación adicionales o más avanzados que requieren mayor cálculo.

Parte 4:

Conclusión:

El desarrollo de una red neuronal desde cero fue una experiencia muy educativa. La implementación de cada componente desde los principios básicos usando únicamente numpy y sin recurrir a librerías avanzadas como TensorFlow o PyTorch me permitió profundizar en el funcionamiento interno de estos modelos. Al programar manualmente pude comprender con detalle cómo cada capa y neurona contribuyen al proceso de aprendizaje.
Este enfoque a bajo nivel fue fundamental para obtener una visión clara de cómo funcionan las redes neuronales, desde la importancia de la inicialización de pesos hasta el impacto de las funciones de activación y el descenso de gradiente en el ajuste del modelo.
Más allá del cumplimiento de los objetivos académicos, este proyecto despertó en mí un interés por seguir explorando el campo del aprendizaje automático.