Instituto Tecnológico y de Estudios Superiores de Monterrey

Inteligencia Artificial Avanzada para la Ciencia de Datos I (Grupo 101)

Momento de Retroalimentación: Módulo 2 Implementación de una técnica de aprendizaje máquina sin el uso de un framework. (Portafolio Implementación)

Daniel Kaled Bernal Ayala A01750047

5 de Septiembre del 2025

1. Introducción

En el presente trabajo se realiza la implementación del algoritmo de Backpropagation para una red neuronal, la cual clasificará una base de datos creada desde cero. Además este será un código desarrollado desde python para su funcionamiento independiente. De igual forma, se explicará dicha implementación y se abordará el desempeño de esta por medio de una matriz de confusión y métricas relacionadas con el desempeño de la red neuronal. A continuación se presenta la implementación del algoritmo y las pruebas realizadas para comprobar el desempeño.

El algoritmo de Backpropagation, como lo vimos en clase, es uno de los pilares en el entrenamiento de redes neuronales. Permite actualizar los parámetros de mejor manera y buscar el error dentro de la red neuronal para así corregir. Esto debido a que el algoritmo se especializa en eso, ir hacia atrás para poder actualizar y aprender los pesos y bias correspondientes a la red y así pueda predecir de mejor manera lo que se busca.

2. Fundamentación Teórica

2.1. Base de Datos y Arquitectura de la red

Para el funcionamiento del algoritmo primero se define que se quiere predecir, es por ello que se realizó una base de datos para clasificar números definidos en una matriz de 3x3. Los cuales son representados de la siguiente manera.

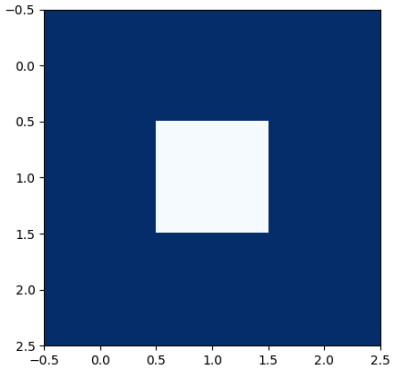

Figura 1. Matriz de 3x3 del número 0

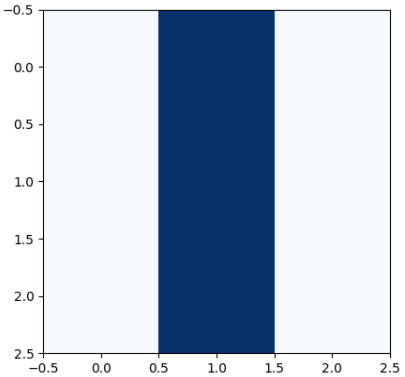

Figura 2. Matriz de 3x3 del número 1

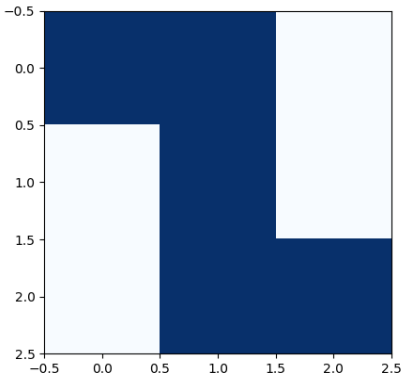

Figura 3. Matriz de 3x3 del número 2

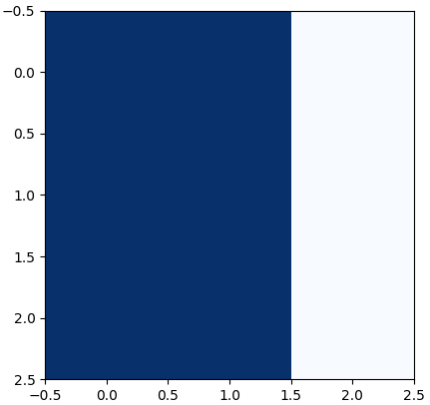

Figura 4. Matriz de 3x3 del número 3

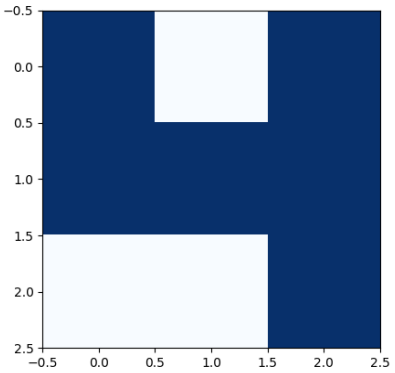

Figura 5. Matriz de 3x3 del número 4

Esta base de datos cuenta con los 5 datos mostrados en las figuras y su resultado de "y" es one hot encoding definiendo los 5 números como (0,0,0,0,1) para 0, (0,0,0,1,0) para 1, (0,0,1,0,0) para 2, (0,1,0,0,0) para 3 y (1,0,0,0,0) para 4. Haciendo que la red neuronal se vea de la siguiente manera:

- **Capa de entrada**: 9 neuronas (cada píxel de la matriz 3×3).
- **Capa oculta:** 10 neuronas con activación sigmoide.
- **Capa de salida:** 5 neuronas (clases 0, 1, 2, 3, 4), también con activación sigmoide.

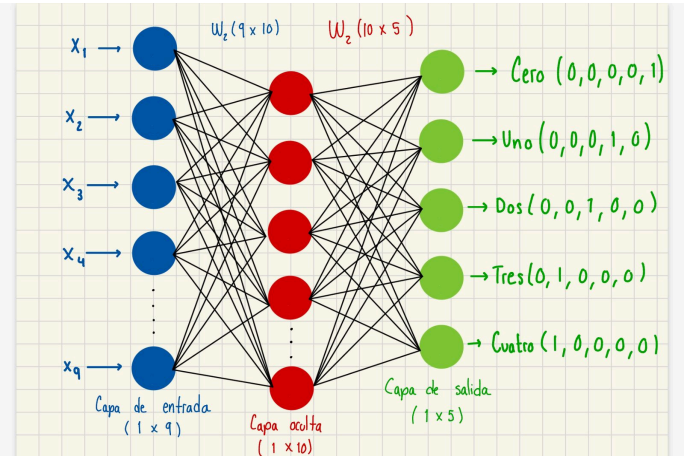

Figura 6. Arquitectura de la red neuronal implementada

Con esto definido, se realiza una expansión del dataset para poder tener una cantidad de datos estable para entrenamiento, validación y testeo. Es por ello que dentro del código se implementa una función llamada **agregar_ruido().** Lo que realiza esta función es sumar un número random que permite generar un tipo de ruido en las matrices 3x3 para tener 20 datos distintos de cada entrada. Por lo tanto, el aumento de los datos sería de 20x5 dándonos un total de 105 datos para la comprobación de su funcionamiento. Con esto hecho se dividen los datos correspondientes con una repartición 70% entrenamiento, 20% validación y 10% testeo generando lo siguiente:

- **70% de los 105 datos = 73**
- **20% de los 105 datos = 21**
- **10% de los 105 datos = 9**

2.2. Feed Forward

Para la implementación del Feed Forward tenemos que realizar la multiplicación de la entrada por el peso correspondiente. Después de esto se implementa la función de activación, en este caso una función sigmoide para fines prácticos. Quedando con las siguientes ecuaciones para la capa oculta.

$$Z = X \cdot W + b$$

$$A = \sigma(Z) = \frac{1}{1+e^{-Z}}$$

En donde:

- X es la entrada correspondiente a los 9 datos de entrada
- W son las matrices de los pesos
- b es el bias que solo existe en la capa oculta y en la capa de salida
- σ es la función sigmoide
- A es la salida de la capa

Posterior a esto se realizan dos cálculos, uno para la capa oculta y otra para la capa de salida quedando definida en la función **FeedForward().**

2.3. Cálculo del error utilizando MSE

Utilizando el error cuadrático medio que está representado por la siguiente ecuación.

$$s = \frac{1}{N}\sum_{i=1}^{N}(Y_i - A_i)^2$$

Una vez realizada la sumatoria del error que existe entre el valor esperado y el valor real, se elevan al cuadrado y se divide entre el total de datos para tener el error.

2.4. Backpropagation

Para la función Backpropagation se realiza como todo un FeedForward para obtener la salida de la capa oculta y la salida de la capa de salida. Con esto elaborado, se calculan los deltas correspondientes al descenso del gradiente y así actualizar los pesos y los bias dependiendo del error obtenido. Dando las siguientes ecuaciones para los Deltas tanto k como h.

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

En donde:

- t_k es el valor esperado de y
- o_k es la salida de la última capa

Reescribiendo así la ecuación dentro del código para el cálculo del error de la siguiente manera

$$error = (t_k - o_k)$$

$$\delta_k = error \cdot (o_k(1 - o_k))$$

Con esto obtenido, encontramos $\delta_h$de la capa oculta que está representada por la siguiente ecuación.

$$\delta_h = o_h(1 - o_h) \sum_{k \in outputs} W_{kh}\delta_k$$

Por lo cual, en el código se implementa donde $A_1 = o_h$, la transpuesta de la matriz de los pesos W_2, transmite estos hacia atrás, y la multiplicación por A_1*$(1 - A_1)$ es la derivada de la sigmoide. Por último, se realiza la actualización de los pesos en donde se obtienen los gradientes de los pesos multiplicando la transpuesta de las activaciones de la capa anterior por los deltas de error de la capa actual. Se utiliza la transpuesta para respetar las dimensiones de las matrices y así obtener los resultados correspondientes. Terminando con la actualización con las ecuaciones.

$$W_1 = W_1 + \alpha \cdot \Delta W_1$$

$$W_2 = W_2 + \alpha \cdot \Delta W_2$$

Donde **alpha** corresponde a la tasa de aprendizaje (*learning rate*). Lo mismo sucede con la actualización del bias en donde:

$$b_1 = b_1 + \alpha \cdot \Delta b_1$$

$$b_2 = b_2 + \alpha \cdot \Delta b_2$$

3. Entrenamiento

Una vez implementada la red neuronal, se necesita entrenar para que pueda identificar los números descritos en la base de datos. El procedimiento se organiza en **épocas**, que representan el número de veces que la red neuronal procesa el conjunto completo de datos de entrenamiento. Dentro de cada época, se recorre muestra por muestra. En cada paso, primero se calcula la salida de la red mediante el **Feed Forward** propagation. Una vez hecho el **Feed Forward**, se calcula el **MSE** y guardando el error en una matriz. Después, se realiza el **Backpropagation** para obtener la actualización de pesos y del bias. Al final de la época, se calcula la pérdida del promedio que se guarda en una variable para después graficar y analizar cómo disminuyó el error en el entrenamiento.

En la segunda parte se pasa el set de entrenamiento en la red neuronal para obtener el porcentaje de accuracy durante en el entrenamiento. Esto nos da una medida de qué tanto la red está memorizando o aprendiendo el patrón de los datos. De igual forma, se realiza un accuracy para el set de validación. Evaluando el desempeño de la red en datos que no se usaron para el ajuste de parámetros. Este valor es crucial, ya que permite identificar si la red neuronal está generalizando correctamente o si está cayendo en overfitting. Por último, se imprime el error al terminar las épocas y el accuracy tanto del entrenamiento como el de validación.

4. Evaluación

Después de entrenar la red neuronal y tener el modelo, encontramos el siguiente gráfico sobre el nivel de accuracy entre el entrenamiento y el set de validación.

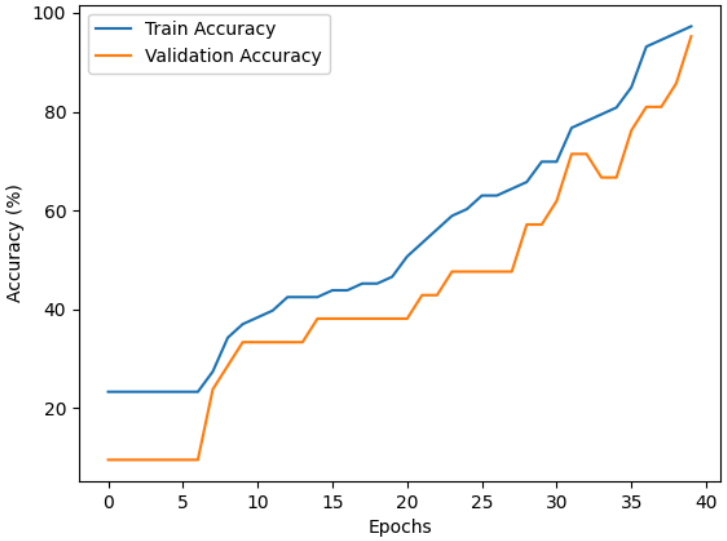

Figura 7. Porcentaje de accuracy del entrenamiento y la validación

Podemos observar, cómo después de 40 épocas, el modelo alcanzó un 97.26% en el entrenamiento y 95.24% en la validación. Con esto podemos podemos ver que el modelo está generalizando correctamente. Haciendo que la red haya aprendido correctamente la identificación del dataset.

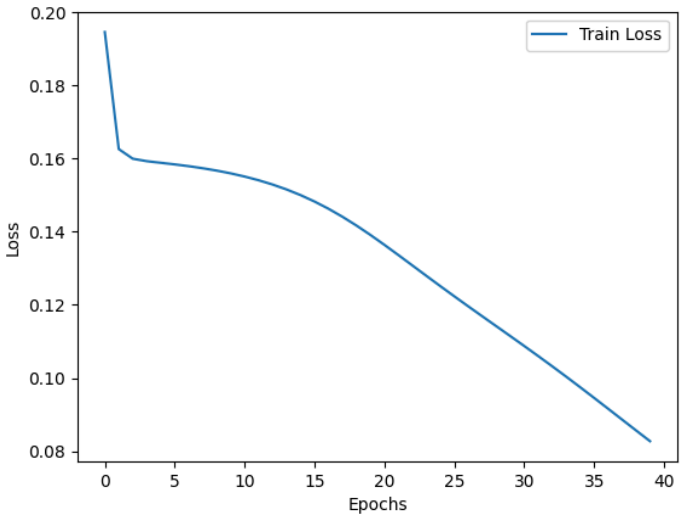

Figura 8. Gráfico del cambio del error entre época

De igual manera, en la Figura 8 podemos ver como el entrenamiento se realizó de manera correcta. Teniendo en cuenta que el learning rate es de 0.1 lo que es un ritmo lento para las pocas épocas que entrenó. Sin embargo, la disminución es adecuada, dando un valor de 0.08 lo que significa un 8% de error en predecir el dato correcto del database.

Después de ello, se realiza la prueba con los datos reservados del test para así realizar una matriz de confusión y analizar los datos obtenidos.

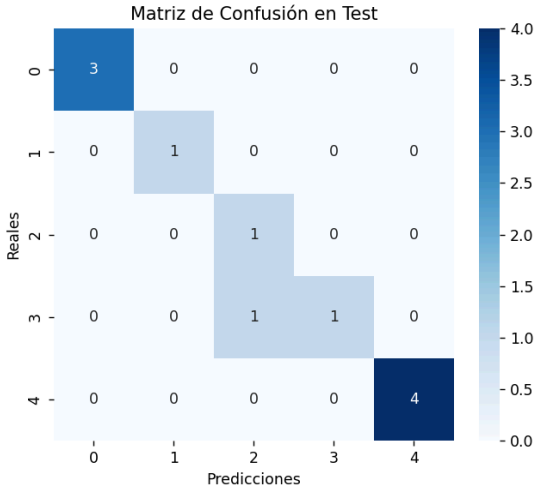

Figura 9. Matriz de confusión

Después de analizar la matriz de confusión mostrada en la Figura 9 podemos observar que existe una confusión con la salida predecida de la matriz 2 y 3, ya que se confundió con esos 2 datos reales, dando un falso positivo. El accuracy en el test es del 90.91% dentro de esta primera implementación con el dataset descrito anteriormente.

5. Resultados

Después de obtener el análisis de la Figura 9, encontramos la siguiente tabla de resultados.

Tabla I. Métricas del modelo con el test realizado

	Precisión	Recall	F1-score	support
Cero	1.00	1.00	1.00	3
Uno	1.00	1.00	1.00	1
Dos	0.50	1.00	0.67	1
Tres	1.00	0.50	0.67	2
Cuatro	1.00	1.00	1.00	4

Como se puede ver el modelo logra aprender a distinguir los patrones de dígitos 0–4 representados en 3×3. Sin embargo, no se puede concretar el funcionamiento correcto del modelo, debido a que como se observa en los datos de testeo, estos son muy pocos. Lo que se tendría que hacer es seguir aumentando el dataset para tener una mayor precisión en la predicción de los números representados en las matrices 3x3.

De igual manera, la precisión en el conjunto de prueba depende del ruido y número de épocas, pero típicamente se alcanza un alto desempeño en este problema. Para mejorar el accuracy sería bueno probar con más épocas pero teniendo cuidado con el overfitting. Esto se puede hacer comparando las salidas de las gráficas. Esto se puede observar en la siguiente figura.

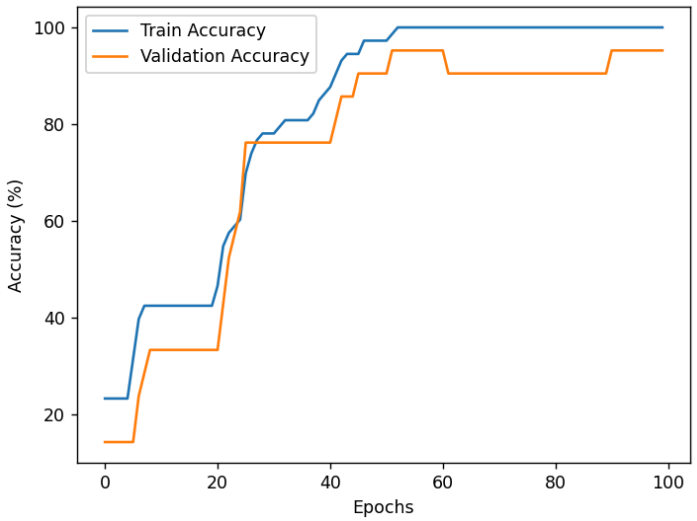

Figura 10. Representación del overfitting en el gráfico del accuracy

Aquí nos damos cuenta que ya existe overfitting debido a que el set de entrenamiento se lo aprendió al 100% y la validación deja de seguir de cerca al entrenamiento. Esto es que deja de generalizar de manera correcta y nuestro modelo no es funcional. Es por ello que se debe de tener cuidado con el entrenamiento y cómo se hará la división del dataset. De igual manera, para la implementación de esta red neuronal no se utiliza batch ya que es un dataset pequeño, pero se debe de considerar a futuro para datasets mayores. Finalizando así con la implementación del Backpropagation y comprobando los resultados con una base de datos creada desde cero.

Referencias

Uresti, J. (2025). Presentación Tema 6 - Redes Neuronales. Recuperado del material visto en clase.