Red Neuronal

Jaime Ballesteros Domínguez
jaime.ballesteros101@alu.ulpgc.es

José Gabriel Reyes Rodríguez
jose.reyes121@alu.ulpgc.es

7 de enero de 2024

Resumen

En este trabajo se nos pedía mejorar la red neuronal realizada en la anterior práctica. El resultado de tal trabajo resultaría en una red mucho más flexible y adaptativa a los diferentes conjuntos de datos y resultados esperados. Para ello hicimos uso de cuatro datasets de características distintas, tanto para relizar una clasificación binaria como para una clasificación multiclase. Entre los elementos creados para la realización encontramos la función de backpropagation, capas como la convolutiva, flatten y fully connected. Además, se añadieron varias funciones de pérdida y de optimización para ampliar nuestra capacidad de ejecución frente a diferentes casos. Al final y tras realizar varias ejecuciones, nuestro porcentaje de acierto se situó finalmente entorno al 55 % para conjuntos de datos conformados por imágenes y un porcentaje superior al 75 % para conjuntos de datos extraídos de archivos de Excel.

1. Introducción

En la actualidad, presenciamos el auge de la inteligencia artificial y, en consecuencia, el interés en las redes neuronales. Es esencial comprender tanto la estructura como el funcionamiento interno de estos algoritmos. En esta investigación, hemos desarrollado una librería propia para redes neuronales desde cero, explorando su funcionamiento mediante la experimentación con diversos conjuntos de datos. Nuestro enfoque no busca competir en rendimiento con frameworks consolidados como TensorFlow o PyTorch, sino profundizar en los fundamentos matemáticos que sustentan estas redes. Por tanto, nuestros análisis no se centrarán exclusivamente en la precisión de los resultados, sino en comprender los aspectos teóricos que respaldan a las redes neuronales.

2. Metodología

Para la ejecución de esta nueva red neuronal, se diseñaron tres capas distintas, siendo estas la convolutiva, la flatten y la fully connected.

2.1. Convolución

Una convolución es un operador matemático que combina dos funciones f y g en una tercera función que en cierto sentido representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

$$(f * g)(t) \doteq \int_{-\infty}^{\infty} f(\eta) g(t - \eta) d\eta$$

Figura 1: Fórmula de la convolución.

Las capas convolutivas entonces, contiene un conjunto de núcleos convolucionales(también llamados filtros o kernels), que se convolucionan con la imagen de entrada para generar un mapa de características de salida.

2.1.1. Kernel

Un kernel puede describirse como una cuadrícula de valores discretos o numéricos, donde cada valor se conoce como el peso de este núcleo. No es más que un filtro que se utiliza para extraer las características de las imágenes.

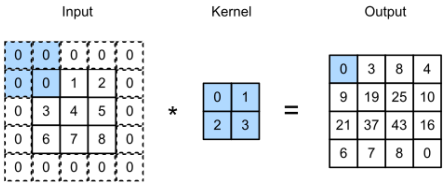

Figura 2: Ejemplo de uso del kernel.

2.2. Flatten (aplanamiento)

Se utiliza para convertir todas las matrices 2D resultantes de los mapas de características, es decir, los resultados de la capa convolutiva, agrupados en un solo vector lineal continuo largo. La matriz aplanada alimenta a la capa fully connected que clasifica la imagen.

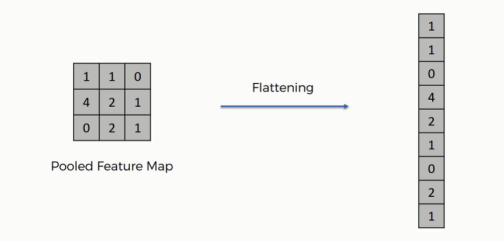

Figura 3: Ejemplo del flattening.

2.3. Fully Connected (completamente conectada)

La última parte de una red neuronal utilizada para la clasificación consta de capas completamente conectadas, donde cada neurona dentro de una capa está conectada con cada neurona de la correspondiente capa anterior.

Las capas fully connected (FC) toman de entrada la salida de la capa convolutiva o de agrupación final, que tiene la forma de un conjunto de métricas las cuales se usan para crear un vector y este vector luego alimenta a la FC para generar la salida final de la red neuronal.

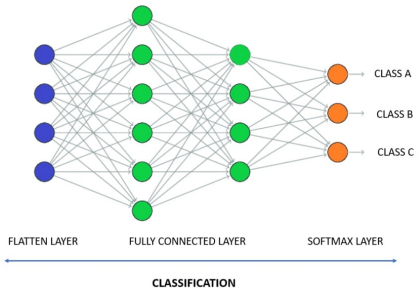

Figura 4: Ejemplo de capa fully connected.

También, en este nuevo trabajo se ha hecho uso de variedad de funciones de activación, como pueden ser la ReLU, la softmax, la leaky ReLU y la sigmoide.

- **ReLU**: La función de activación ReLU (Rectified Linear Unit) se define matemáticamente como:

$$f(x) = \text{máx}(0, x)$$

En otras palabras, para cualquier valor de entrada x, la función ReLU devuelve cero si x es negativo, y devuelve x mismo si x es no negativo. De manera concisa, la función ReLU activa una unidad neuronal solo si la entrada es positiva, y no tiene efecto si la entrada es negativa. Su forma gráfica es una línea recta con pendiente uno para $x \geq 0$ y cero para $x < 0$. Esta función es comúnmente utilizada en redes neuronales debido a su simplicidad y efectividad.

- **Softmax**: La función de activación Softmax se utiliza en la capa de salida de una red neuronal y transforma un vector $\mathbf{x}$ de K números reales en un vector $\mathbf{y}$ de la misma dimensión. Cada elemento y_i está en el rango de 0 a 1 y la suma de todos los elementos es igual a 1. La fórmula matemática para el elemento i de la función Softmax es:

$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Esta función asigna probabilidades a cada clase en un problema de clasificación.

- **Leaky ReLU** : La función de activación Leaky ReLU (ReLU con fuga) es una variante de la función ReLU que permite que los valores negativos tengan una pequeña pendiente en lugar de simplemente ser cero. La formulación matemática de Leaky ReLU es:

$$\text{Leaky ReLU}(x) = \begin{cases} x, & \text{si } x \geq 0 \\ \alpha x, & \text{si } x < 0 \end{cases}$$

- **Sigmoide** : La función de activación sigmoide, también conocida como función logística, se utiliza comúnmente para transformar valores a un rango entre 0 y 1. La formulación matemática de la función sigmoide es:

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

También hemos hecho uso de cuatro funciones de pérdida, entre las que se encuentran la MSE, RMSE, Binary Crossed Entropy y Crossed Entropy:

- **Crossed Entropy**: La función de pérdida de entropía cruzada (Cross Entropy) es comúnmente utilizada en problemas de clasificación. La formulación matemática de la pérdida de entropía cruzada para un problema de clasificación binaria es:

$$\text{Entropía Cruzada}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

En el caso de la clasificación multiclase, la fórmula se extiende a:

$$\text{Entropía Cruzada}(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i)$$

- **MSE**: La función de pérdida Mean Squared Error (MSE), o error cuadrático medio, se utiliza comúnmente en problemas de regresión. La formulación matemática de MSE es:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **RMSE**: La función de pérdida Root Mean Squared Error (RMSE), o error cuadrático medio raíz, es una variante del Mean Squared Error (MSE) que calcula la raíz cuadrada de la media de las diferencias al cuadrado entre las predicciones del modelo y los valores reales. La formulación matemática de RMSE es:

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Binary Crossed Entropy**: La función de pérdida Binary Cross Entropy (entropía cruzada binaria) es comúnmente utilizada en problemas de clasificación binaria. La formulación matemática de Binary Cross Entropy para un solo ejemplo es:

$$\text{Binary Cross Entropy}(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right)$$

  En el caso de múltiples ejemplos, la fórmula se extiende a:

$$\text{Binary Cross Entropy}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right)$$

3. Experimentos

En este trabajo se ha hecho uso de cuatro datasets distintos, siendo estos Digit Recognizer (dataset .csv de clasificación múltiple), HotDog (dataset de imágenes de clasificación binaria), Stroke Prediction (dataset .csv de clasificación binaria) y DogsvsCats (dataset de imágenes de clasificación binaria):

3.1. Digit Recognizer

Este dataset contiene 42.000 imágenes de números entre el 0 y el 9 en formato csv. En este formato cada número está representado por 256 columnas, una para el número que representa la imágen (label) y el resto para representar cada pixel de la imagen en la escala de grises (0 a 255)

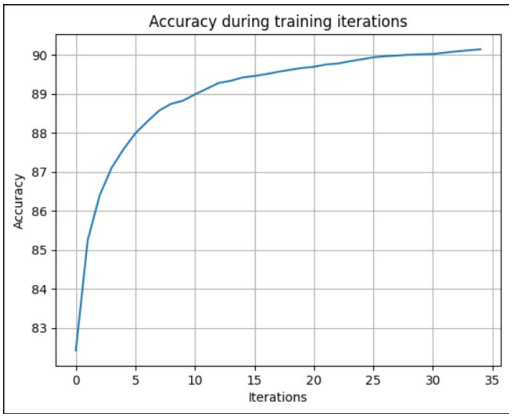

Figura 5: Precisión del 89,73 %.

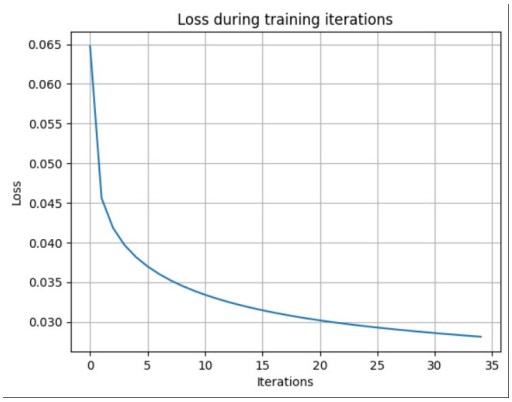

Figura 6: Pérdida de 0,028.

Como podremos ver al tratar los siguientes datasets, Digit Recognizer es el que mejor encaja con la red neuronal que hemos creado, teniendo una precisión bastante elevada y un error casi inexistente.

3.2. HotDog

El siguiente dataset contiene 4242 imágenes divididas a partes iguales entre las que contiene perritos calientes y las que no. Dentro de las imágenes que no contienen hot dogs se pueden encontrar tanto otras comidas como animales.

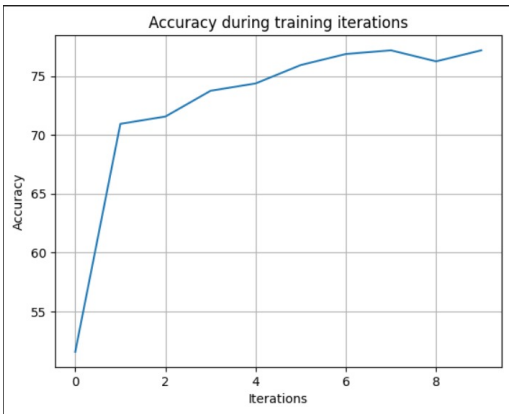

Figura 7: Precisión del 61,25 %.

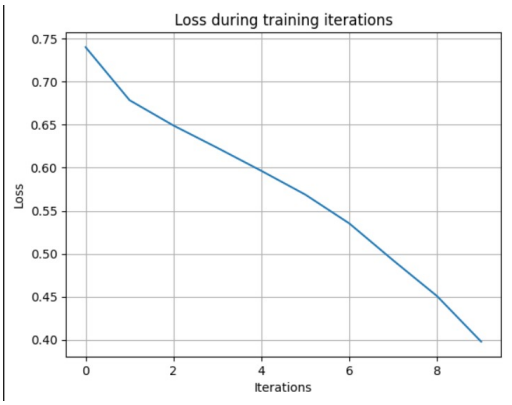

Figura 8: Pérdida de 0,398.

El dataset HotDog presenta unos resultados peores que los obtenidos con Digit Recognizer. Esto puede ser debido a que estamos tratando con imágenes en sí y no con tablas de información. También puede ser el resultado de la reducción de las imágenes, la cual haya podido empeorar la vizualización del elemento a clasificar.

3.3. Stroke Prediction

Este dataset contiene 5110 casos en los que se predicen si es probable que el sujeto pueda llegar a sufir un infarto. El dataset en sí tiene 12 columnas, siendo la primera el identificador del entrevistado, la última el resultado en 1 o 0 con respecto a la posibilidad de tener un infarto y las 10 de en medio los parámetros en los que se basa dicha suposición. Entre los parámetros de estas 10 columnas podemos encontrar la edad, el sexo, si se tiene hipertensión o enfermedades cardíacas entre ellos.

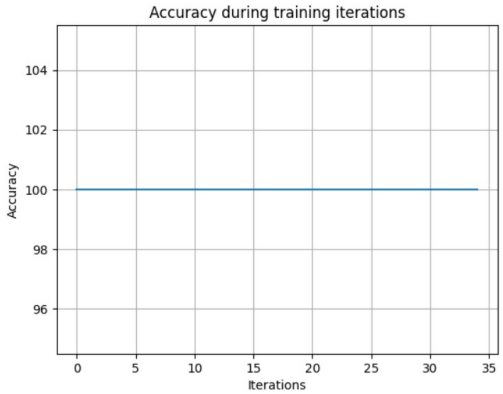

Figura 9: Precisión del 100 %.

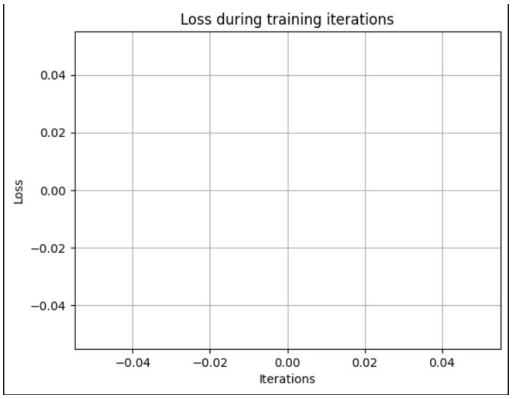

Figura 10: Pérdida de NaN.

Estos resultados nos llevan a concluir que, en este caso específico, la aplicación de una red neuronal resulta excesiva. Este hallazgo demuestra que este enfoque no necesariamente se ajusta óptimamente a todos los

problemas planteados.

3.4. DogsvsCats

El siguiente dataset contiene 25000 imágenes divididas a partes iguales entre las que son perros y las que son gatos.

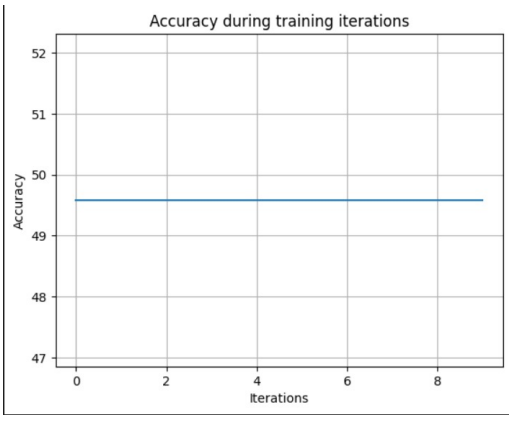

Figura 11: Precisión del 51,67 %.

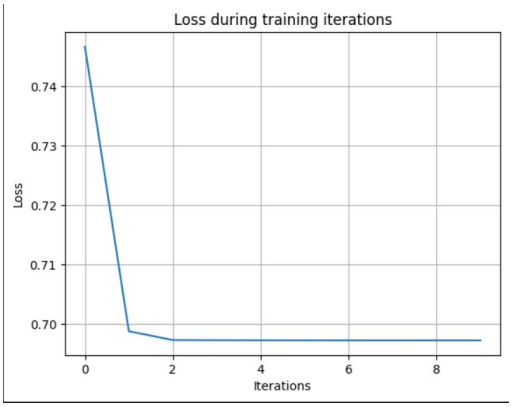

Figura 12: Pérdida de 0,697.

La falta de variabilidad en la precisión de este Dataset puede ser debido a que solo hemos utilizado una pequeña parte de este (200 imágenes de cada animal) y al reducido número de épocas. Esto se ha hecho principalmente para evitar elevados tiempos de ejecución.

4. Conclusiones

Como hemos podido ver a la hora de probar los diferentes datasets, podemos deducir que las redes neuronales no son aptas para todos los problemas, ya sea por el tipo de datos presentados o por la cantidad de estos. También podemos ver el gran peso que tiene la resolución de las imágenes a tratar, como podemos comprobar con la ejecución de los datasets compuestos por imágenes. Ya que en dichos datasets se tuvo que reducir las dimensiones de las imágenes con el objetivo de reducir los tiempos de ejecución y evitar un uso excesivo de la CPU.

5. Trabajo futuro

Como parte del desarrollo futuro de este trabajo, se pueden plantear las siguientes líneas de investigación:

- Implementar una variedad más amplia de capas neuronales, como MaxPooling o BatchNormalization, entre otras, para enriquecer la versatilidad de la librería.
- Incorporar otras arquitecturas neuronales, por ejemplo, AutoEncoders, con el fin de ampliar la gama de modelos disponibles.
- Realizar investigaciones orientadas a mejorar el rendimiento de la red neuronal desarrollada, explorando y aplicando estrategias de optimización y ajustes de hiperparámetros.

6. Bibliografía

- Digit Recognizer
- HotDog
- Stroke Prediction
- DogsvsCats