REDN EURONAL PYTHON PARTE 2

Optimización y Heurística

3º Curso de Ciencia e Ingeniería de Datos
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Aris Vazdekis Soria y Alejandra Ruiz de Adana Fleitas
25/12/2023

MEMORIA SOBRE EL PROYECTO DE RED CONVOLUTIVA

El proyecto se centra en mejorar un modelo de red neuronal aplicado al conjunto de datos Fashion MNIST, inicialmente tratado con una versión simple de red neuronal. En esta nueva etapa, la red se expande y mejora mediante el uso de técnicas convolucionales, capas adicionales y estrategias de optimización para potenciar su precisión y rendimiento.

La mejora se logra al implementar una arquitectura más sofisticada con capas convolucionales seguidas de max pooling y capas completamente conectadas. Además, se inicia la red con pesos y sesgos utilizando la inicialización de Xavier, junto con funciones de activación ReLU y Softmax.

La manipulación de datos, evaluación y visualización se lleva a cabo exclusivamente con las bibliotecas NumPy, scikit-learn y Matplotlib. Durante el entrenamiento, se escalan los datos al rango [0, 1] para normalizarlos y se aplican capas ocultas de tamaños 128, 64 y 32 con un kernel de 3.

Una de las mejoras más significativas se logra al emplear el algoritmo de optimización Adam, que actualiza los pesos y sesgos con momentos corregidos en cada epoch, aprovechando hiperparámetros beta.

El proyecto inicial implementó una red neuronal con tres capas ocultas de menores tamaños y una capa de salida para la clasificacion de prendas del mismo conjunto Fashion MNIST. Ciertas funciones como activacion ReLU y inicializacion de pesos con Xavier estuvieron en el modelo original. Se opto por el algoritmo Adam y este se mantiene en el nuevo proyecto ya que es el algoritmo de optimizacion mas adecuado al conjunto de datos. La mayor precision lograda en el anterior modelo se obtuvo tras un entrenamiento de 2,000 epoch con una precision del 77%.

INDICE

CONJUNTO FASHION-MNIST

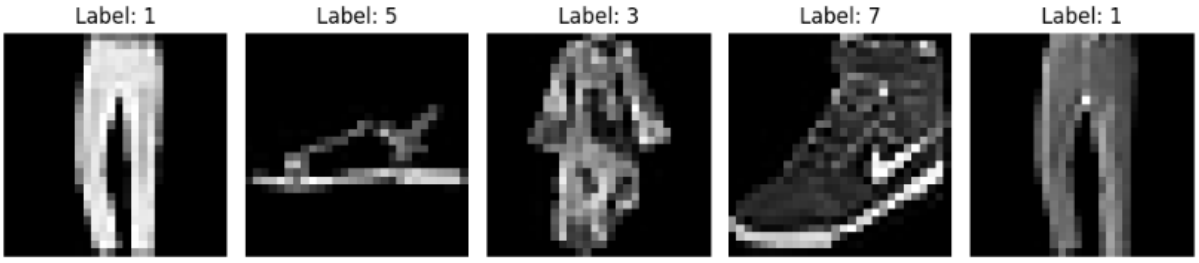
El conjunto de datos Fashion-MNIST es un conjunto de imagenes de prendas de vestir y calzados en una escala de grises de imagenes con tamaño 28x28, utilizado comunmente como un reemplazo directo del conjunto de datos MNIST para probar algoritmos de clasificacion de imagenes como el que hemos desarrollado. Contiene 60,000 imagenes de entrenamiento y 10,000 imagenes de prueba, etiquetadas en 10 clases distintas.

Las clases en Fashion-MNIST son:

0. Camiseta/top
1. Pantalon
2. Jersey
3. Vestido
4. Abrigo
5. Sandalia
6. Camisa
7. Tenis deportivo
8. Bolso
9. Bota

Cada imagen en el conjunto de datos representa una prenda de la lista, como mencionamos antes en escala de grises, donde los valores de los pixeles oscilan entre 0 y 255, siendo 0 el negro y 255 el blanco. Es por esta escala por lo que la normalizacion de las imagenes en nuestro proyecto fue dividir entre 255 las imagenes, para que asi el valor numerico pase a ser un valor situado en el rango [0,1]

Un ejemplo de 5 imagenes del conjunto que recibe nuestro modelo junto con su etiquetado.



FUNCIONES

En esta seccion vamos a hablar sobre las diferentes funciones matematicas utilizadas en el programa, tanto para el calculo de la convolucion, como funciones de activacion, ajuste de pesos, etc.

- **FUNCIONES 1.1 —> *RELU***

  La **función ReLU** es una funcion de activacion que implementamos en el feedforward y se define matematicamente como f(x) = max(0,x). Esto significa que devuelve 0 si el valor de entrada es negativo y devuelve el mismo valor de entrada cuando es positivo. Por otro lado, la **derivada de la función ReLU** fue necesaria para el proceso de backpropagation en el entrenamiento de la neurona. Lo que nos dice es que si la derivada es positiva tendrá valor 1 y cuando sea negativa o sea 0 tendrá valor 0.

- **FUNCIONES 1.2 —> *SOFTMAX***

  La funcion de softmax es una funcion de activacion que utilizamos en la capa de salida de la red neuronal. Matematicamente, el softmax transforma un vector de numeros reales en un vector de probabilidades, asegurandose de que la suma de todas las probabilidades sea igual a 1.

  La funcion softmax toma como entrada un vector z de K numeros reales y produce un vector a de la misma longitud, donde cada elemento a_i se calcula de la siguiente manera:

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

  Donde e es la base del nogaritmo natural (aprox 2.718), Z_i es el i-esimo elemento del vector de entrada Z y $\sum_{j=1}^{k} e^{z_j}$ es la suma de las exponenciales de todos los elementos en el vector Z.

  Entonces softmax es una funcion de normalizacion exponencial que toma el input lineal y lo transforma en una distribucion de probabilidad. Cada elemento del vector de salida a representa la probabilidad de que el input pertenezca a una de las clases.

El enumerador e^{z_i} juega un elemento clave, ya que este amplifica los valores mas grandes y los diferencia mas claramente. Luego la suma del exponenciador normaliza estas exponenciales para que la salida sea una probabilidad valida. El resultado de utilizar la funcion de activacion softmax es realmente magnificar las diferencias entre los valores de entrada, lo que hace que el elemento con mayor puntuacion tenga una probabilidad mas alta y los demas elementos tengan probabilidades mas bajas.

- **FUNCIONES 1.3 —> *INICIALIZACIÓN DE XAVIER***

La **inicialización de Xavier** es una técnica para inicializar los pesos de una red neuronal de manera que se reduzca la varianza de las activaciones a lo largo de las capas. La fórmula de la inicialización de Xavier es:

$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

donde n_{in} es el número de unidades de la capa de entrada y n_{out} es el número de unidades de la capa salida. La idea detrás de esta inicialización es que al multiplicar los pesos inicializados por una distribución normal con media 0 y varianza $\frac{2}{n_{in} + n_{out}}$ se logre que la varianza de las activaciones sea aproximadamente la misma en todas las capas.

En la función “xavier_initialization” primero se genera una matriz de pesos con valores aleatorios extraídos de una distribución normal estándar y luego esta matriz se multiplica por $\sqrt{\frac{2}{n_{in} + n_{out}}}$, que es la fórmula de Xavier para escalar los valores.

- **FUNCIONES 1.4 —> *METODO DE OPTIMIZACION ADAM***

Como **método de optimización** elegimos el método **ADAM.** Este método modifica las bases del algoritmo SGD al introducir conceptos como el momentum y la estabilidad de los promedios de los gradientes.

SGD Original:

$\theta_{t+1}^{SGD} = \theta_t - \alpha \nabla J(\theta_t)$

(donde α es la tasa de aprendizaje y $\nabla J(\theta_t)$ es el gradiente de la función de pérdida con respecto a los parámetros en el tiempo t).

Modificación con Momentum:

$m_t = \beta \cdot m_{t-1} + (1 - \beta) \cdot \nabla J(\theta_t)$

$\theta_{t+1}^{Momentum} = \theta_t - \alpha \cdot m_t$

(donde m_t representa el momentum que incorpora promedios de los gradientes)

Incorporación de RMSprop en Adam:

$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla J(\theta_t))^2$

$\theta_{t+1}^{Adam} = \theta_t - \alpha \cdot \frac{m_t}{\sqrt{v_t} + \epsilon}$

Esta versión completa de Adam incluye el RMSprop en donde el v_t es el momento del segundo orden y tanto el v_t como el m_t son versiones sesgadas corregidas de los momentos. El término ϵ se añade para evitar la división por 0 rompiendo el funcionamiento. El método Adam combina la estabilidad de los promedios de los gradientes para que las actualizaciones tengan inercia y utiliza el historial de gradientes adaptativo del RMSprop para ayudarse a superar problemas donde la magnitud de los gradientes varia ampliamente entre parámetros, de manera que la tasa de aprendizaje evita que sea demasiado pequeña o demasiado grande.

- **FUNCIONES 1.5 —> *ENTROPIA CRUZADA***

Una **fórmula** que utilizamos para calcular la pérdida en el error fue la **entropía cruzada**. Esta medida se emplea para evaluar cuánto se ajustan las probabilidades predichas por el modelo a las etiquetas reales. La formula de la entropía cruzada es:

$$H(y, \widehat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} log(\widehat{y}_{ij})$$

N = Número total de muestras
C = Número de clases
y_{ij} = Es 1 si la muestra i pertenece a la clase j y 0 de lo contrario (Etiqueta real)
$\widehat{y}_{ij}$ = Es la probabilidad predicha por el modelo de que la muestra i pertenezca a la clase j.

En el programa la entropía cruzada se calcula de la siguiente manera "*-np.log(probs[range(len(X_train)), y_train])"*, donde se selecciona las probabilidades predichas correspondientes a las clases reales de las muestras que calculamos utilizando Softmax, y a esta se le aplica el logaritmo negativo (−log) y se toma la media para obtener la pérdida promedio.

- **FUNCIONES 1.6 —> *PROPAGACION EN LA RED***

El proceso de propagacion en la red es imprescindible ya que es donde la red aprende patrones complejos. Este proceso consta de dos fases claves, el feedforward, donde los datos se mueven hacia adelante generando predicciones, y el backpropagation, que es donde se ajustan los pesos de la red en funcion del error, permitiendo asi que la red mejore su precision. A continuacion explicaremos como funcionan ambos procesos.

  - **FUNCIONES 1.6.1 —> FEEDFORWARD**

Primeramente explicaremos el feed foward. Aqui es donde los datos de entrada se alimentan a la red, pasando a traves de cada capa. Para cada neurona en cada capa, se realiza una combinacion lineal de las entradas ponderadas por los pesos, seguida de la aplicacion de la funcion de activacion utilizada (ReLU). Esto se expresa matematicamente de la siguiente manera:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}$$
$$a^{(l)} = f(z^{(l)})$$

Donde l denota la capa actual, $W^{(l)}$ son los pesos, $a^{(l-1)}$ es la activacion de la capa anterior, $b^{(l)}$ es el sesgo y f es la funcion de activacion.

Despues de que los datos de entrada hayan pasado por la red y se haya generado una salida, esta pasa por la funcion de softmax que se aplica para obtener las probabilidades de las clases. Esta se compara con la etiqueta real utilizando una funcion de perdida, que ya previamente explicada, utilizamos la entropia cruzada.

$$L = Funcion\ de\ pérdida(y_{predicho}, y_{real})$$

  - **FUNCIONES 1.6.2 —> BACKPROPAGATION**

El objetivo del backpropagation es minimizar esta funcion de perdida ajustando los pesos de la red. Se calcula el gradiente de la funcion de perdida L con respecto a los pesos de la red utilizando la regla de la cadena y se utiliza para actualizar los momentos del primer y segundo orden de Adam.

El gradiente indica la direccion en la que los pesos deben ser ajustados para minimizar la perdida.

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial W^{(l)}}$$

Entonces el momento de primer orden del algoritmo Adam:

$$m^{(l)} = \beta_1 \cdot m^{(l)} + (1 - \beta_1) \cdot \frac{\partial L}{\partial W^{(l)}}$$

Donde $m^{(l)}$ es el momento del primer orden para los pesos $W^{(l)}$, $\beta_1$es el coeficiente de decaimiento del momento y $\frac{\partial L}{\partial W^{(l)}}$ es el gradiente de la funcion de perdida con respecto a los pesos.

En los momentos de segundo orden del algoritmo Adam:

$$v^{(l)} = \beta_2 \cdot v^{(l)} + (1 - \beta_2) \cdot \left(\frac{\partial L}{\partial W^{(l)}}\right)^2$$

Donde $v^{(l)}$ es el momento del segundo orden para los pesos $W^{(l)}$, β_2 es el coeficiente de decaimiento del momento y $\left(\frac{\partial L}{\partial W^{(l)}}\right)^2$ es cuadrado del gradiente.

Luego se aplica una corrección de los momentos sesgados:

$$m^{(l)}_{corregido} = \frac{m^{(l)}}{1-\beta_1^t}$$
$$v^{(l)}_{corregido} = \frac{v^{(l)}}{1-\beta_2^t}$$

Donde t es el numero de iteraciones.

Todo este proceso se realiza para cada capa, propagando el error desde la capa de salida hacia la capa de entrada, ajustando los pesos en funcion de la tasa de aprendizaje y el gradiente calculado con Adam de la siguiente manera:

$$W^{(l)}_{nuevo} = W^{(l)}_{viejo} - \ Tasa\ de\ aprendizaje\ \times \frac{m^{(l)}_{corregido}}{\sqrt{v^{(l)}_{corregido}} + \epsilon}$$

EXPERIMENTACIÓN

Los datos de entrenamiento y prueba se cargan desde archivos CSV y, como parte del preprocesamiento, normalizamos los valores de píxeles dividiéndolos por 255. Esta normalización tiene como objetivo escalar los valores de cada píxel al rango de 0 a 1. Este proceso es crucial, ya que ayuda al modelo a converger más rápidamente durante el entrenamiento al mantener los valores en una escala manejable.

La elección de una tasa de aprendizaje de 0.001 se basa en experimentos previos. Tasas de aprendizaje más bajas tendían a provocar que el modelo clasificara todas las imágenes en una sola clase, mientras que tasas más altas resultaban en un sobreajuste. La tasa seleccionada busca un equilibrio que permita un aprendizaje efectivo sin perder generalización.

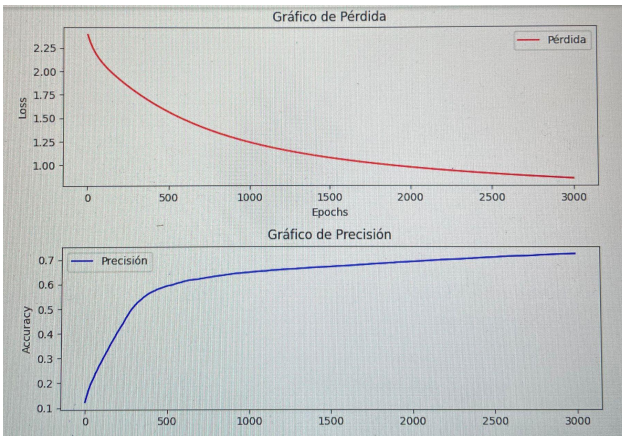
En la evaluación del modelo, hemos implementado la matriz de confusión para examinar su comportamiento en cada ejecución. Además, hemos incorporado una función de pérdida que nos proporciona información sobre la mejora potencial del modelo y una funcion de accuracy para conocer que tan precisa es la red

Hemos analizado el comportamiento de la red variando algunos hiperparametros y optimizadores. A continuacion presentaremos los siguientes experimentos que hicimos y las conclusiones de cada uno.

- **MODELO 1**

El primer modelo que creamos ajustaba los pesos utilizando SGD y con 3,000 epoch. Este nos arrojo los siguientes datos:

```
Matriz de Confusión:
[[791  12  29  93   6   0  43   3  23   0]
 [ 16 920  20  32   8   0   4   0   0   0]
 [ 21   3 648   6 235   1  60   0  26   0]
 [ 41  20   6 869  31   0  29   0   4   0]
 [  3  11 189  84 668   0  38   0   7   0]
 [  5   2   5   1   0 441   2 329  24 191]
 [246   5 179  61 268   1 199   0  40   1]
 [  0   0   0   0   0  26   0 854   1 119]
 [  2   0  15  21   8   3  16  21 912   2]
 [  0   0   0   0   0   7   1  67   0 925]]
```



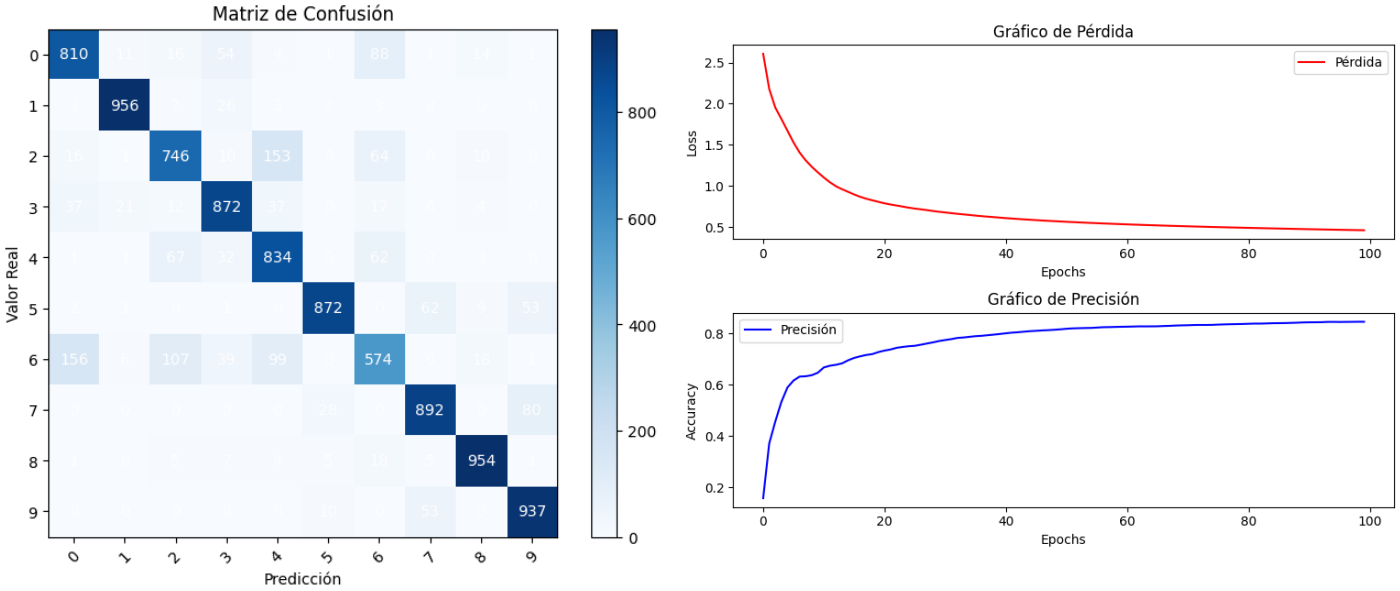
Podemos ver como tras una larga espera de 3,000 epoch (57 minutos de ejecucion) este algoritmo dio una precision del 72% con un learning rate de 0.001. La matriz esta en linea general bien formada a excepcion de la clase numero 6 que coincide con el tipo de prenda de camisa, llegando a fallar el 80% de las muestras.

En el anterior proyecto utilizamos Adam y sin ser una red convolucional, este consigue superar este modelo 1, logrando una precision del 77% en 1,000 epoch, tres veces menos que en el presente modelo. En esta red convolutiva con el optimizador SGD no logra

alcanzar nuestro criterio pues el nivel de precision esperado dentro de un numero limitado de epoch de entrenamiento no es optimo, por lo que concluimos que debido la complejidad de los datos junto con la seleccion del optimizador SGD para esta red convolutiva no alcanza un rendimiento esperado.

- **MODELO 2**

Este modelo implementa el metodo de optimizacion Adam en la red convolucional. Se han realizado 100 epoch (2 minutos y 32 segundos) y este ha sido su rendimiento:



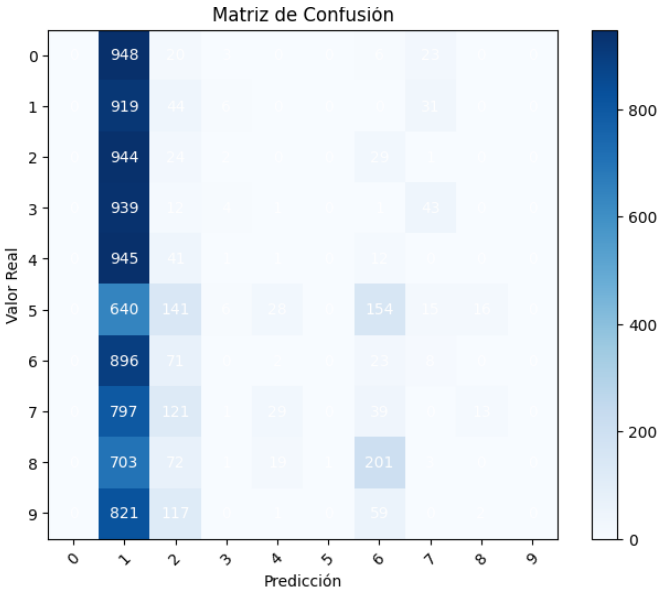
Este modelo 2 ha tenido una sorprendete mejora frente al modelo 1, dando una precision del 84% (lr=0.001) en tan solo 100 epoch de entrenamiento frente a las 3.000 del modelo 1. La incorporacion de Adam al modelo ha bajado el valor de la perdida al 0.44 lo que demuestra que el modelo es mas consistente y veloz que con el algoritmo de optimizacion SGD. Tambien encontramos la observacion de la clase 6 (camisa) que pese a que el valor es bastante mas adecuado que en el modelo previo, continua siendo la clase con mayor dificultad de clasificacion y tan solo el 57% de las muestras reales de clase 6 fueron true positive.

- **MODELO 3**

Este modelo se divide en dos experimentaciones. Nos hemos centrado en variar el learning rate a un valor mas alto 0.1 y a un valor mas bajo 0.00001 para conocer como el modelo se comporta segun variamos la tasa de aprendizaje.

  - **VARIACION LEARNING RATE BAJO**

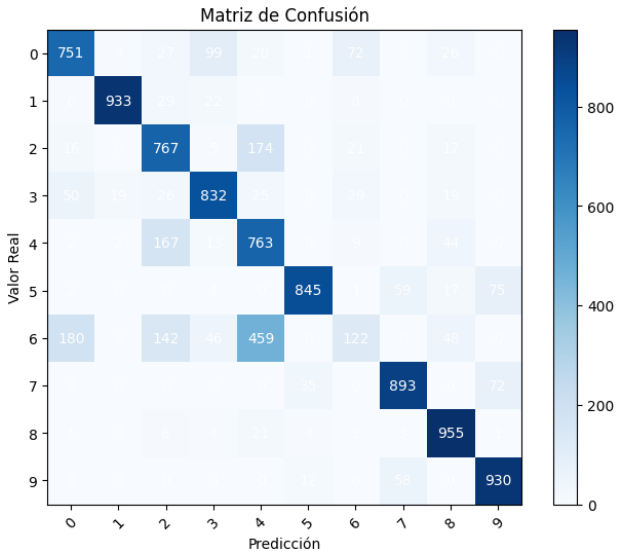
En esta variacion utilizaremos el modelo 2 pero con un learning rate de 0.00001 frente al del modelo 2 (0.001).



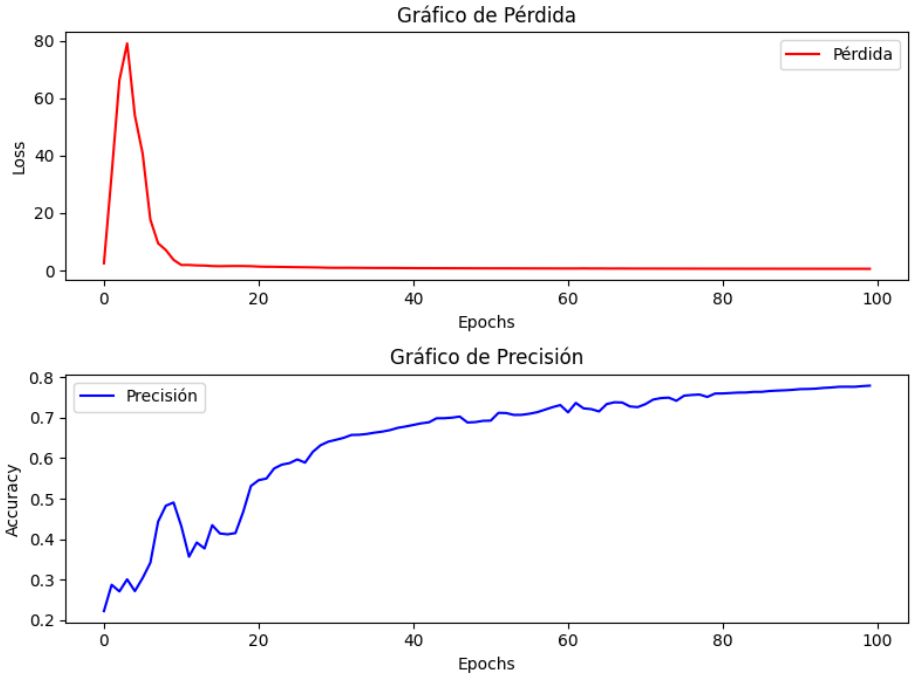
Observamos como curiosamente sucede un caso muy similar al del proyecto pasado en donde de igual manera bajando el learning rate sucede que todas las muestras del conjunto de prueba se acaban clasificando en una misma clase que curiosamente al igual que en la red neuronal simple es la clase 1 (pantalon). Bajar el learning rate hace que el modelo carezca de precision y logica alguna.

- **VARIACION LEARNING RATE ALTO**

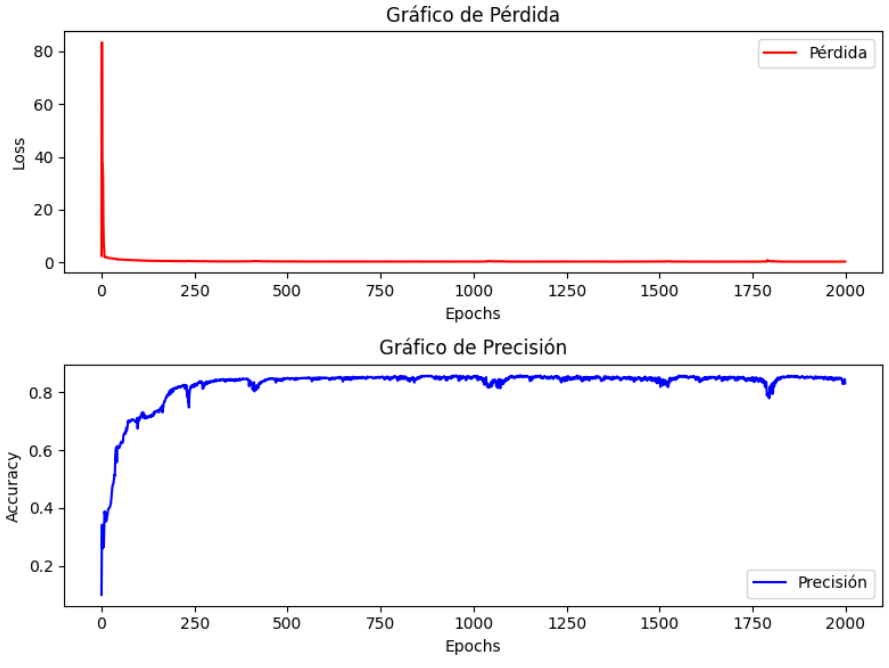
Ahora probaremos con un learning rate de 0.1 y variaremos las epoch para poder conocer como este modelo 3 se comporta, comenzando por 100 epoch.



Observamos como al subir el learning rate el modelo se comporta de manera adecuada pero insuficiente puesto que la precision es de un 77% respecto a un 84% que presentaba el modelo 2 en las mismas condiciones. A continuacion analizaremos las funciones de perdida y precision.



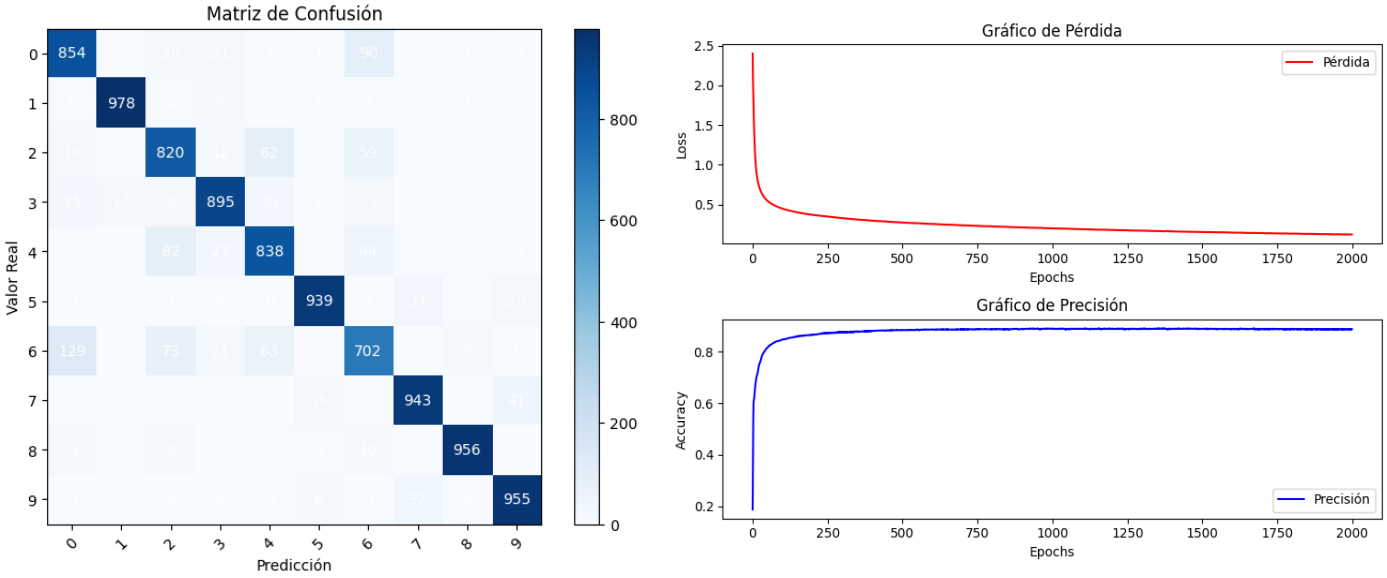
Observamos como el modelo durante su entrenamiento sufre de leves fluctuaciones en su precision y varios excesos en las primeras epocas en cuanto a la funcion de perdida. Seria interesante conocer si esto se trata de una simple inestabilidad en las primeras iteraciones debido a ciertos excesos en las operaciones matematicas de ajuste de peso o es que el modelo sufre de convergencia hacia minimos locales. Para entender este comportamiento aumentaremos los epoch a 2,000 para observar como el modelo se comporta en una ejecución intensa.



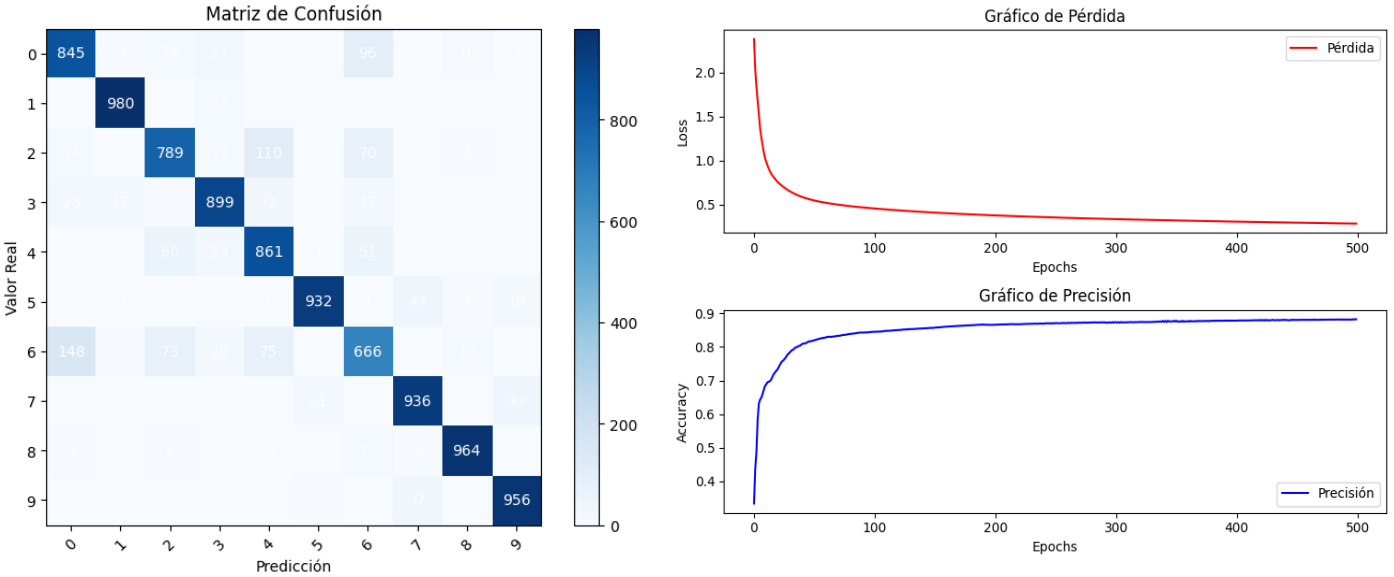
Como podemos observar el modelo en el transcurso del entrenamiento sufre de diversas fluctuaciones en donde converge hacia minimos locales. El modelo en una cantidad alta (asignando el concepto de alta en comparacion con el modelo 2) carece de estabilidad y robustez, siendo una opcion descartada.

- **MODELO 4**

En este modelo vamos a mantener el learning rate en 0.001 y aumentaremos las epocas a 500 y 2,000 para observar como se comporta la red. Comenzamos con 2,000.



Como podemos observar este modelo 4 en 2,000 epocas es bastante mas estable y robusto que el modelo 3 con su alta tasa de aprendizaje. La precision es de un 88% y la funcion de perdida desciende hasta adquirir un valor de 0.12 dando los mejores resultados hasta ahora. La matriz muestra como las predicciones son precisas y firmes. Observamos en la funcion de precision como a partir de las 500 epocas realmente el modelo no obtiene una mejora en la precision, asi que vamos a analizar el modelo con 500 epocas para concluir la mejor version.



Las funciones de perdida y precision junto con la matriz de confusion muestran como 500 epocas son suficiente para entrenar este conjunto de datos arrojandonos una precision de el 88.27% frente al 88,01%, lo que no es una diferencia notoria a diferencia de que esta ejecucion se realizo en 4 veces menos tiempo. Este modelo 4 de red convolutiva con algoritmo de optimizacion Adam, learning rate en 0.01 y 500 epocas se convierte en la mejor version ajustada de la red.

CONCLUSIONES

La implementación de este proyecto supuso un desafío significativo al fusionar todas las funciones de feedforward, backpropagation y el proceso de convolución dentro de la red neuronal. A medida que avanzábamos en el diseño y la integración de estas funciones, nos enfrentamos a múltiples obstáculos que dificultaron el progreso.

Inicialmente, nos encontramos con problemas de estabilidad en el entrenamiento debido a la complejidad de unificar la lógica de convolución con las capas tradicionales de la red neuronal del previo proyecto. El como empezar y como estructurar la convolucion utilizando unicamente la libreria NumPy fue abrumador. Uno de los problemas a los que mas nos tuvimos que enfrentar en este comienzo fue el manejo de las dimensiones de las imagenes en la red convirtiendose en una tarea compleja y que consumio una cantidad significativa de tiempo y esfuerzo.

Además, durante las etapas finales del desarrollo de la estructura, nos enfrentamos a problemas persistentes relacionados con la precisión de las predicciones. Este fenómeno se debió a una incorrecta inicializacion de los pesos utilizados en la red, lo que generó conflictos en la propagación de los gradientes.

A pesar de estos desafíos, perseveramos en nuestra búsqueda de soluciones. Después de numerosos intentos y pruebas exhaustivas, logramos identificar y corregir las incompatibilidades entre las funciones de convolución y la red tradicional. Esto nos permitió integrar con éxito el proceso de convolución en la red neuronal, superando los problemas de pesos y los errores de predicción.

En última instancia, el tiempo y esfuerzo dedicados a superar estos desafíos nos llevaron a una solución efectiva. Logramos entrenar y entender un modelo capaz de clasificar con precisión el conjunto Fashion MNIST completamente manual, consolidando así nuestra comprensión y habilidades en la implementación de redes neuronales convolucionales.

LÍNEAS FUTURAS

En busca de una red mas precisa e infalible podriamos utilizar ciertas tecnicas de regularizacion como dropout en donde podriamos lograr mejorar la capacidad de generalizacion y estabilidad del modelo.

Tambien podriamos investigar y experimentar con arquitecturas mas complejas de redes neuronales como la ResNet o redes recurrentes como GRU, para explorar su rendimiento en diferentes pruebas con otros conjuntos de datos.

En ultima instancia pensamos que aumentar el conjunto de datos generando de manera virtual mayor variabilidad en el conjunto con diferentes tecnicas de aumento como rotacion o reescalabilidad de dimensiones de las imagenes en las capas convolutivas.

RECURSOS UTILIZADOS

La estructura y contenido de la memoria se han desarrollado siguiendo las pautas del proyecto de red neuronal. Entre los recursos utilizados encontramos como entorno de desarrollo Google Collab, puesto que hemos recibido varias recomendaciones de dicha aplicacion web de Google. No se utilizó ninguna herramienta de control de versiones y como herramienta de documentación se utilizó Documentos de Google.

BIBLIOGRAFIA

- https://www.kaggle.com/code/milan400/cnn-from-scratch-numpy
- https://www.youtube.com/watch?v=3Dpclm3m1Cc
- https://es.wikipedia.org/wiki/Red_neuronal_convolucional
- Apuntes campus virtual