Proyecto Final Red Neuronal
Miguel Angel Patiño Caicedo - 2369650

Este proyecto se centró en desentrañar el corazón de los algoritmos de aprendizaje profundo, construyendo y analizando un **Perceptrón Multicapa (MLP)** de principio a fin, sin depender de librerías de alto nivel como TensorFlow o PyTorch. Utilizamos el popular conjunto de datos **Fashion-MNIST** para la clasificación de imágenes simples.

El objetivo principal no fue solo obtener alta precisión, sino también **validar la eficiencia algorítmica** de cada componente, desde las operaciones de la red neuronal hasta las estructuras de datos auxiliares, todo medido con la notación Big-O.

1. El Núcleo de la Red Neuronal (Fase 1: Análisis Asintótico)

Implementamos la arquitectura base del MLP: una capa de entrada, una capa oculta con activación **ReLU**, y una capa de salida con **Softmax** para la clasificación.

- **Entrenamiento (SGD):** La clave fue implementar la propagación hacia adelante (*Forward Pass*) y la **Retropropagación** (*Backpropagation*) para ajustar los pesos de la red. Esto se hizo a mano, utilizando álgebra lineal (multiplicaciones de matrices con NumPy) para calcular las pérdidas y los gradientes.
- **Complejidad del Entrenamiento:** El costo de cada época de entrenamiento (pasar por todos los datos) depende linealmente del tamaño del *dataset* (N), la dimensión de entrada (D), y el número de neuronas ocultas (H). Analizamos que la complejidad temporal de una época es aproximadamente O(N * D * H), y lo validamos empíricamente midiendo el tiempo de ejecución al aumentar H y el tamaño del lote (B). El requisito de **Correctitud** fue verificado mediante el **Gradiente *Checking***. Este proceso compara los gradientes analíticos (Backpropagation) con una estimación numérica, asegurando que el corazón matemático del modelo funciona correctamente.

2. Optimización con Estructuras de Datos Clásicas (Fases 2 y 3)

Para demostrar cómo las estructuras de datos tradicionales mejoran el rendimiento, integramos módulos de selección y ordenamiento.

Selección de Elementos (Quickselect y Top-k)

- **Búsqueda de la Mediana:** Implementamos **Quickselect** para encontrar la mediana de un conjunto de datos en tiempo promedio O(N). Esto es significativamente más rápido que ordenar la lista completa (O(N log N)), lo cual se confirmó en las pruebas de escalado. * **Selección de los Top-k (Heap):** Implementamos un **MinHeap** (Montículo de Mínimos) para mantener eficientemente los k elementos más grandes de una lista. La complejidad de esta operación es O(N log K) lo que supera la eficiencia de un ordenamiento completo O(N log N) cuando K es mucho menor que N.

Integración para el "Hard Mining"

El *Heap* se integró directamente en el proceso de entrenamiento del MLP mediante una técnica llamada **Hard Mining**.

- **Muestreo Inteligente:** En lugar de entrenar con todos los datos por igual, la red procesa primero todas las muestras. Luego, utiliza el *MinHeap* para seleccionar solo el **10% de las muestras con mayor pérdida** (las más "difíciles" de clasificar).
- **Impacto en el Tiempo Total:** Al entrenar en cada época solo con un subconjunto de datos (los "difíciles"), se logró un **ahorro de tiempo considerable** en el entrenamiento por época, demostrando una mejora práctica gracias al uso eficiente de la estructura de datos.

3. Comparación con el Modelo Baseline (Fase 4: k-NN)

Finalmente, comparamos el rendimiento de nuestro MLP con un algoritmo *baseline* clásico: el clasificador **k-Vecinos más Cercanos (k-NN)**.

Metrica	MLP	K-NN
Tiempo entrenamiento	Lento (segundos)	Instantaneo (O(1))
Tiempo prediccion	Rápido (milisegundos)	Lento (O (M * N * D))
Precision	Alta (alrededor de 85%)	Moderada

Conclusión de la Comparativa: El **k-NN** es trivial de entrenar, pero su predicción es **extremadamente lenta** porque debe comparar la nueva muestra con *todos* los datos de entrenamiento. En contraste, nuestro **MLP** requiere un entrenamiento costoso, pero una vez listo, su predicción es **casi instantánea** (solo requiere dos multiplicaciones de matrices), lo que lo hace ideal para aplicaciones en tiempo real.