TRABAJO PRÁCTICO: REDES NEURONALES Y APRENDIZAJE PROFUNDO

Tobio Santiago
Departamento de ingeniería en Inteligencia Artificial
Universidad de San Andres
Buenos Aires, Argentina
stobio@udesa.edu.ar

October 25, 2025

ABSTRACT

acá va el abstract

1 Introducción

En este Trabajo Práctico se aborda el desarrollo e implementación de modelos de redes neuronales profundas (MLP's) desde cero en Python. El objetivo principal es comprender los fundamentos teóricos y prácticos detrás de las redes neuronales y evaluar el rendimiento de diferentes arquitecturas y técnicas de optimización sobre esta familia de modelos.

Evaluamos el desempeño de los modelos sobre el dataset de EMNIST Bymerge. Este conjunto de datos es una extensión del clásico MNIST, que incluye imágenes de dígitos manuscritos y letras mayúsculas y minúsculas, proporcionando un desafío adicional para la clasificación de imágenes.

Los modelos reciben como input un vector de 784 dimensiones (28x28 píxeles aplanados) y tienen como output una probabilidad para cada una de las 47 clases posibles (dígitos del 0 al 9, letras mayúsculas A-Z y letras minúsculas a-z, excluyendo algunas letras para evitar confusiones). La clasificación se realiza determinando la clase a la que le corresponde la mayor probabilidad en el output del modelo.

Para acelerar el proceso de entrenamiento y aprovechar el computo de operaciones matrices a través de la GPU, se utilizó la biblioteca CuPy, que ofrece una interfaz similar a NumPy pero con soporte para cálculos en GPU.

Definimos 5 configuraciones experimentales en función de lo especificado en el enunciado del trabajo práctico, variando la cantidad de capas ocultas, la cantidad de neuronas por capa, la función de activación , el optimizador utilizado y técnicas adicionales. A continuación, se detallan las configuraciones:

- **M0:** Modelo base con 2 capas ocultas de 128 y 64 neuronas respectivamente, función de activación ReLU , softmax en la capa de salida y función de pérdida de entropía cruzada. Optimización mediante SGD con mini batch de tamaño 256, tasa de aprendizaje de 0.01 y sin momentum.
- **M1:** Mejor modelo (según validación) variando la cantidad de capas ocultas y neuronas por capa. Y aplicando diversas optimizaciones en el proceso de entrenamiento. Algunas de ellas son: Rate scheduling, tanto lineal como exponencial ; uso de optimizador Adam; Regularización L2; Early Stopping.
- **M2:** Implementación de un MLP en PyTorch, entrenado con los hiperparámetros óptimos encontrados en M1.
- **M3:** Implementación de un MLP en PyTorch , explorando distintas arquitecturas y técnicas de optimización adicionales a las utilizadas en M1. Algunas de ellas son: Funciones de activación alternativas (LeakyReLU, SiLU, Swish o GELU); Batch Normalization; Dropout

Como aclaración: si bien sería interesante estudiar el Modelo M0 usando batch GD, no se realizó dicha experimentación debido a limitaciones de memoria al intentar cargar todo el dataset en la GPU.

Posteriormente, se presentan los resultados obtenidos en cada una de las configuraciones experimentales, analizando su desempeño tanto en el conjunto de desarrollo como el de testeo. Finalmente, perturbamos las imágenes de testeo con ruido gaussiano para evaluar la robustez de los modelos entrenados frente a datos ruidosos.

2 Exploración y preprocesamiento de datos