Clasificando Fashion-MNIST con PyTorch

Máximo Caprari[1]

[1]Facultad de Ciencias Económicas, Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina

Noviembre 19, 2024

Abstract

Este trabajo presenta la implementación de un modelo de red neuronal utilizando PyTorch para la clasificación del conjunto de datos Fashion-MNIST. Se describen los aspectos teóricos del modelo, el proceso de entrenamiento y validación, y se analizan los resultados obtenidos en términos de precisión y pérdida. Además, se examina el impacto de variar hiperparámetros clave como el *batch size*, la probabilidad de *dropout* y el número de épocas.

I Introducción y fundamentos teóricos

El conjunto de datos Fashion-MNIST es un estándar ampliamente utilizado para comparar algoritmos de clasificación en visión por computadora. Este trabajo desarrolla un modelo de red neuronal (*feedforward*) con dos capas ocultas y funciones de activación ReLU.Además, se utilizó una capa lineal para realizar las transformaciones entre las capas, permitiendo que el modelo aprenda relaciones complejas entre las características de entrada y salida. Se implementaron técnicas como *dropout* [B] para evitar el sobreajuste, y se emplearon los optimizadores Adam [C] y SGD [D] para el entrenamiento, minimizando la pérdida de entropía cruzada.

El modelo base inicial (Figura 1) consta de las siguientes configuraciones: tasa de aprendizaje (*learning rate*) de 0.01, 30 épocas, tamaño de lote (*batch size*) de 100, probabilidad de *dropout* de 0.2, y los optimizadores Adam y SGD. Este modelo sirvió como punto de partida para evaluar el impacto de los parámetros iniciales y establecer una línea base en términos de precisión y pérdida.

Las métricas utilizadas para evaluar el rendimiento se calculan como:

$$\text{Pérdida} = \frac{\sum \text{Pérdida promedio por lote}}{\text{Número de lotes}} \tag{1}$$

$$\text{Precisión} = \frac{\sum \text{Predicciones correctas}}{\text{Número total de muestras}} \tag{2}$$

Este documento está estructurado como sigue: en las subsecciones se presentan detalles sobre las redes neuronales *feedforward*, la técnica de *dropout*, los optimizadores y los conceptos de épocas y *batches*. Luego se presentan los resultados de las simulaciones de la red, variando diversos parametros, y finalmente, se discuten los resultados obtenidos.

A Red neuronal feedforward

Una red neuronal feedforward es una arquitectura donde la información fluye en una sola dirección, desde las entradas hasta las salidas, pasando por capas ocultas, sin ciclos ni retroalimentación [1].

B Técnica de Dropout

El dropout es una técnica de regularización utilizada para evitar el sobreajuste. Consiste en apagar aleatoriamente ciertas neuronas durante el entrenamiento, lo que obliga a la red a aprender representaciones más robustas [2].

C Optimizador Adam

Adam combina los beneficios del gradiente estocástico (SGD) y el método de momentos, adaptando la tasa de aprendizaje para cada parámetro, lo que mejora la convergencia [3].

D Optimizador SGD

El gradiente estocástico (SGD) ajusta los parámetros de una red utilizando un solo ejemplo de entrenamiento a la vez, lo que puede ser eficiente en memoria, pero ruidoso [4].

E Épocas y batches

Las épocas indican cuántas veces el modelo recorre todo el conjunto de datos durante el entrenamiento. Los batches, subconjuntos del conjunto de entrenamiento, permiten una actualización eficiente de los parámetros [5].

II Resultados

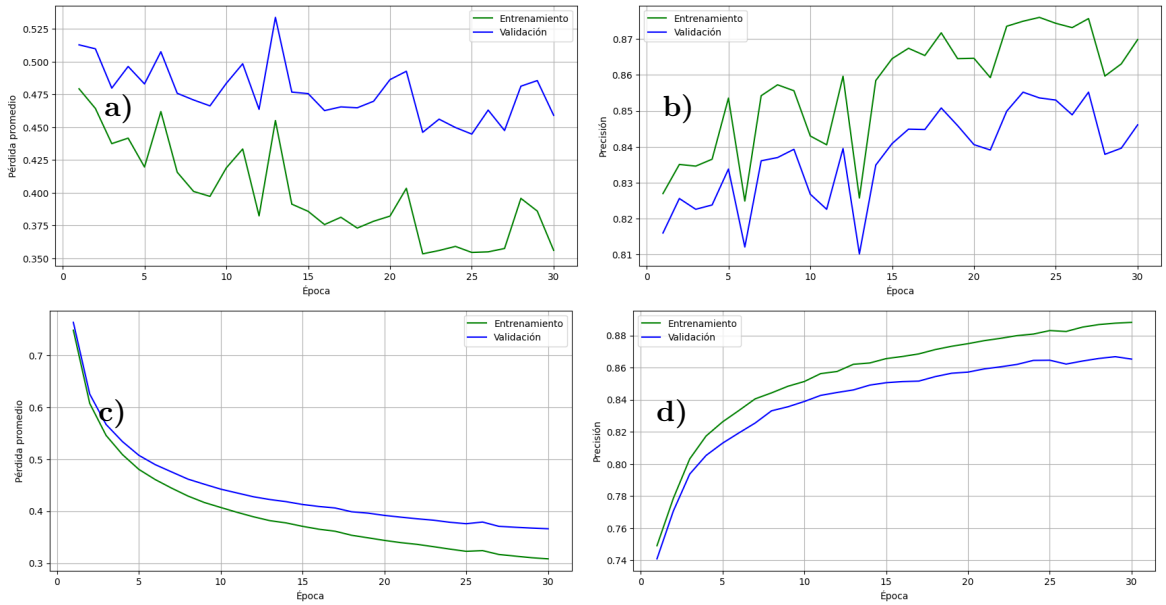

Figure 1: Desempeño del modelo base con Adam y SGD. **a)** Pérdida (1) (Adam) . **b)** Precisión (2)(Adam). **c)** Pérdida (1) (SGD). **d)** Precisión (2) (SGD).

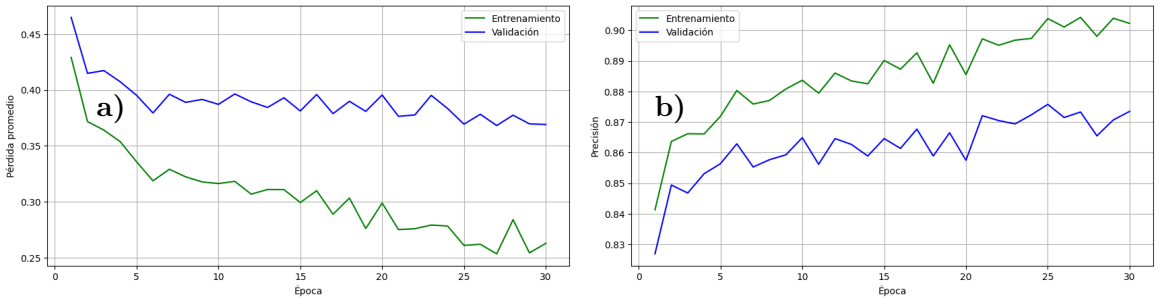

Figure 2: Impacto del tamaño del lote 500 en el desempeño con Adam. **a)** Pérdida (1). **b)** Precisión (2).

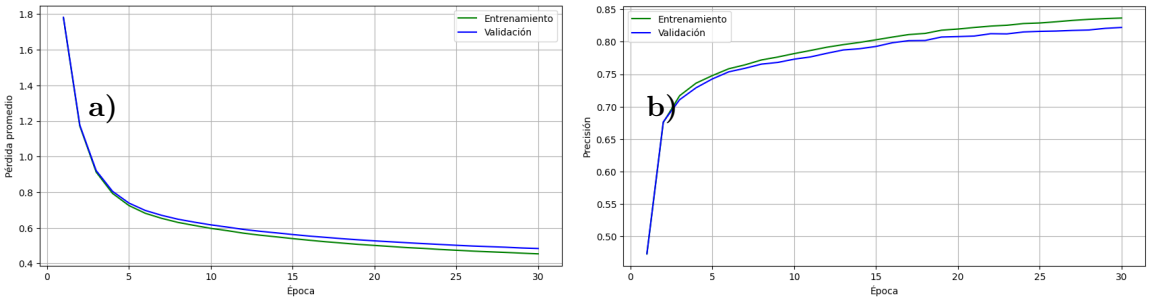

Figure 3: Impacto del tamaño del lote 500 en el desempeño con SGD. **a)** Pérdida(1). **b)** Precisión(2).

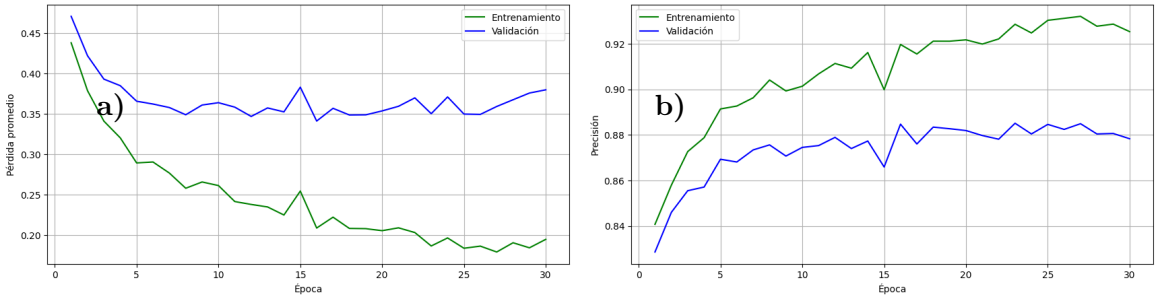

Figure 4: Impacto del tamaño del lote 1000 en el desempeño con Adam. **a)** Pérdida(1). **b)** Precisión(2).

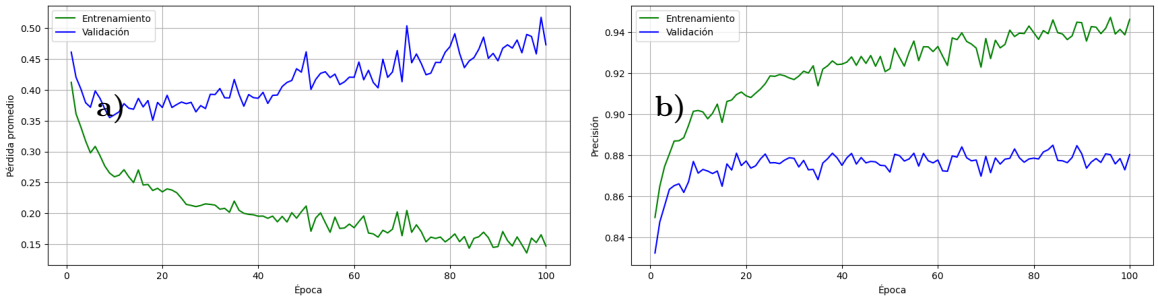

Figure 5: Impacto de 100 épocas en el desempeño con Adam. **a)** Pérdida(1). **b)** Precisión(2).

III Discusión

A Análisis del Modelo Base

El modelo base, una red *feedforward* con dos capas ocultas, fue entrenado utilizando los optimizadores Adam y SGD. Los parámetros iniciales de entrenamiento fueron: tasa de aprendizaje $lr = 0.01$, número de épocas $= 30$, tamaño de lote $= 100$, probabilidad de *dropout* $= 0.2$, y la función de pérdida utilizada fue la entropía cruzada, implementando todas las configuraciones mediante la biblioteca PyTorch [6] (Figura 1).

Modelo base con Adam: La pérdida en el entrenamiento disminuyó rápidamente desde 0.475 hasta aproximadamente 0.35 durante las primeras épocas, mientras que en validación se redujo de 0.52 a 0.475 (Figura 1a). La precisión en entrenamiento aumentó de 0.83 a 0.87, y en validación alcanzó valores cercanos a 0.84, evidenciando una mejora progresiva con baja variabilidad y alta volatilidad (Figura 1b).

Modelo base con SGD: La pérdida en entrenamiento disminuyó de forma más lenta, alcanzando 0.3, mientras que en validación se estabilizó en 0.35 (Figura 1c). La precisión en

entrenamiento mejoró de 0.75 a 0.89, mientras que en validación alcanzó 0.86, aunque con mayor variabilidad (Figura 1d). Esto sugiere que, aunque más lento, el optimizador SGD puede superar a Adam en precisión final.

B Impacto del Tamaño del Lote

El efecto del tamaño del lote se evaluó utilizando valores de 500 y 1000, en contraste con el tamaño base de 100, manteniendo constantes los demás parámetros. Los resultados se presentan en las Figuras 2, 3 y 4.

Tamaño de lote 500: Las curvas de pérdida y precisión para Adam (Figura 2) muestran una convergencia similar al modelo base, aunque con un ajuste más suave. La pérdida se estabiliza cerca de 0.4 en validación, y la precisión alcanza valores de 0.84. Con el optimizador SGD, la pérdida disminuye de manera constante, convergiendo en valores superiores a 0.4, mientras que la precisión aumenta de forma progresiva, alcanzando aproximadamente 0.83 (Figura 3).

Tamaño de lote 1000: Con Adam, se observa una estabilización más temprana, con la pérdida convergiendo en 0.4 y la precisión alcanzando 0.83 (Figura 4). Sin embargo, el rendimiento es inferior al observado con tamaños de lote menores. No se incluye el gráfico para SGD en este caso debido a su alta similitud con las curvas de la Figura 3. Además, se modificó la tasa de dropout de 0.2 a 0.05, no se observaron cambios significativos en el rendimiento, por lo que no se consideró necesario graficar la comparación de estos resultados.

C Impacto del Número de Épocas

Para evaluar el efecto de un entrenamiento más prolongado, el número de épocas se incrementó de 30 (modelo base) a 100. Los resultados se presentan en la Figura 5.

Al extender el entrenamiento, la pérdida en validación mostró un ligero aumento después de la época 30, estabilizándose cerca de 0.4, mientras que en el conjunto de entrenamiento continuó disminuyendo hasta alcanzar valores cercanos a 0.15. Esto sugiere la presencia de sobreajuste, ya que el modelo mejora en entrenamiento, pero pierde capacidad de generalización.

En términos de precisión, los valores en entrenamiento continuaron incrementándose hasta aproximadamente 0.95, mientras que en validación permanecieron estables alrededor de 0.84, mostrando que un mayor número de épocas no aporta beneficios significativos en validación y puede incluso perjudicar el rendimiento.

IV Conclusión

El análisis realizado evidenció que Adam permite una convergencia más rápida, aunque con mayor volatilidad, mientras que SGD alcanza una precisión superior al costo de una pérdida más alta y una convergencia más lenta. El tamaño del lote influye en la estabilidad y velocidad del entrenamiento: lotes pequeños favorecen ajustes rápidos, mientras que lotes grandes estabilizan el proceso con menor rendimiento. Por último, extender el entrenamiento a 100 épocas mostró indicios de sobreajuste, sin mejoras significativas en validación, resaltando la importancia de un ajuste adecuado del número de épocas.

V Referencias

1. Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.
2. Srivastava, N., et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. JMLR, 2014.
3. Kingma, D. P., Ba, J. *Adam: A Method for Stochastic Optimization*. arXiv, 2014.
4. Bottou, L. *Large-Scale Machine Learning with Stochastic Gradient Descent*. Proceedings of Compstat 2010.
5. Goodfellow, I., et al. *Deep Learning*. MIT Press, 2016.
6. Paszke, A., et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS, 2019.