Red neuronal feed-forward junto a Fashion-MNIST

Ignacio Martinez Goloboff*
Redes Neuronales, Licenciatura en Ciencias de la Computación,
Facultad de Matemática, Astronomía, Física y Computación
(Dated: November 19, 2024)

Este escrito explora el análisis y desarrollo de una red neuronal feed-forward diseñada para clasificar imágenes del conjunto de datos Fashion-MNIST [1], un recurso ampliamente utilizado en experimentos de reconocimiento de patrones en artículos de moda. Utilizando PyTorch como marco principal, se investigan los efectos del ajuste de varios hiperparámetros del modelo, incluyendo la tasa de aprendizaje, el dropout y el tamaño de las capas intermedias, sobre el rendimiento de la red [2]. El objetivo es optimizar la precisión de clasificación y comprender la influencia de cada hiperparámetro en la capacidad de generalización del modelo.

I. INTRODUCCIÓN

En la actualidad, la clasificación de imágenes representa uno de los desafíos más importantes en el ámbito del aprendizaje automático, con aplicaciones que abarcan desde el reconocimiento de patrones hasta la toma de decisiones basada en datos visuales. En este contexto, el conjunto de datos Fashion-MNIST se ha convertido en un referente para evaluar el rendimiento de modelos de clasificación.

Este trabajo tiene como objetivo desarrollar e implementar una red neuronal que sea capaz de clasificar dichas imágenes de manera precisa. Se examinan los procesos de preprocesamiento de datos, diseño del modelo, entrenamiento y evaluación, explorando el impacto de diferentes hiperparámetros en el desempeño del modelo y comparando los resultados obtenidos en función de estos ajustes.

II. TEORÍA

El entrenamiento de redes neuronales supervisadas, como las redes feed-forward, pueden dividirse en una secuencia de pasos perfectamente definidos:

1. **Preparación del Dataset:** El conjunto de datos Fashion-MNIST [1] es una herramienta ampliamente utilizada en experimentos de aprendizaje automático. Este dataset consta de 70,000 imágenes en escala de grises (28x28 píxeles) divididas en 10 categorías, ver FIG 1. Las categorías incluyen prendas de vestir como camisetas, zapatos y abrigos. Para este trabajo, se divide en 60,000 imágenes para entrenamiento y 10,000 para validación, asegurando que las particiones representen equilibradamente las clases.

2. **Diseño de la Arquitectura:** Las redes feed-forward [3] constan de capas completamente conectadas, donde cada neurona en una capa está conectada a todas las neuronas de la capa siguiente.

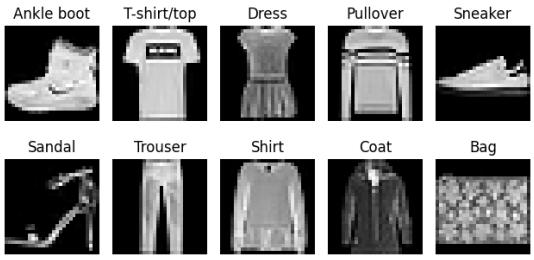

FIG. 1: Ejemplo en el dataset.

En este proyecto, se analiza cómo diversos hiperparámetros impactan el desempeño del modelo, incluyendo la tasa de aprendizaje, que regula la tasa de actualización en los pesos; el dropout, que apartir de una probabilidad p "apaga" neuronas de una capa específica para prevenir sobreajuste; el número de neuronas en las capas ocultas, que influye en la capacidad de aprender patrones complejos; el número de épocas, que define los ciclos de entrenamiento; y el tamaño de los batches, el cual agrupa los datos de entrenamiento en diferentes lotes [2].

3. **Función de pérdida y optimización:** Se utiliza la Cross Entropy Loss como función de pérdida para medir el error entre las probabilidades predichas por el modelo y las clases verdaderas. Usaremos el optimizador [4] SGD que ajusta iterativamente los pesos para mejorar el rendimiento del modelo y minimizar esta función de pérdida.

4. **Entrenamiento y Validación:** Durante el entrenamiento, los datos se procesan en batches para optimizar el cálculo de gradientes. Se evalúa el rendimiento en un conjunto de validación y se ajustan hiperparámetros para maximizar la generalización, garantizando métricas sobre su capacidad de clasificación.

III. RESULTADOS

En esta sección, se analizan los resultados obtenidos durante la primera prueba de entrenamiento de la red neuronal. Para esta configuración inicial, los hiperparámetros utilizados se detallan en la Tabla I. A continuación, se presentan los gráficos obtenidos y su análisis correspondiente.

Hiperparámetro	**Valor utilizado**
Tasa de aprendizaje (lr)	0.001
Dropout (p)	0.2
Tamaño del batch	100
Épocas	30
Capas intermedias (ocultas)	$n_1 = 128$, $n_2 = 64$
Optimizador	SGD

TABLE I: Hiperparámetros de la prueba inicial.

Los resultados iniciales, ver FIG 2, muestran que el modelo logra aprender de manera consistente, con una disminución progresiva de la pérdida en el conjunto de entrenamiento y una estabilización en la validación. La precisión alcanza valores cercanos al 85% en entrenamiento y validación, reflejando una generalización razonable, pero no óptima. Estos resultados sugieren que el modelo puede beneficiarse de ajustes adicionales en los hiperparámetros. Además observamos que la convergencia del modelo se vuelve lenta después de la época 30, con mejoras marginales en las métricas. Por lo tanto, en adelante se optará por usar 30 épocas para maximizar la eficiencia sin comprometer el rendimiento y además los siguientes resultados obtenidos se compararán con este entrenamiento inicial.

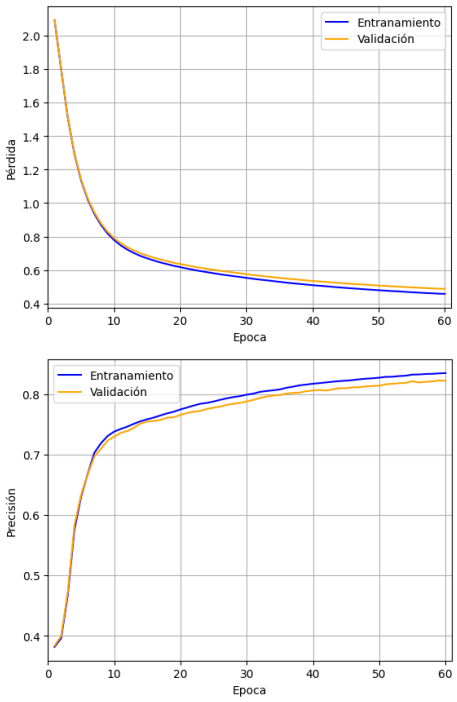

FIG. 2: Entrenamiento inicial; Pérdida y Precisión

A. Tasa de Aprendizaje

La tasa de aprendizaje (lr) impacta significativamente el rendimiento del modelo. Valores bajos, como 10^{-4}, estabilizan el entrenamiento, pero ralentizan la convergencia, requiriendo más épocas para alcanzar un rendimiento aceptable. Por otro lado, tasas más altas, como 0.01, permiten una convergencia más rápida y mejores resultados en precisión y pérdida, ver FIG **??**. Sin embargo, valores excesivamente altos pueden generar fluctuaciones en las métricas, dificultando la estabilidad del modelo. Esto resalta la importancia de elegir un valor equilibrado que combine rapidez y estabilidad, igualmente para las próximas pruebas seguiremos usando $lr = 0.001$, ya que con valores mas bajos empieza a aparecer un sobreajuste.

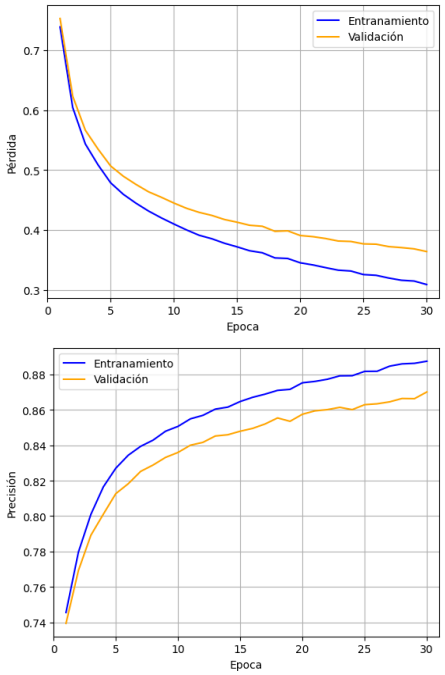

FIG. 3: Pérdida y Precisión con lr=0.01

B. Dropout

Al incrementar el dropout de $p = 0.2$ a $p = 0.5$, no se observó una mejora significativa; por el contrario, el modelo presentó una mayor pérdida y una disminución en

la precisión tanto en el conjunto de entrenamiento como en el de validación. Esto sugiere que un valor alto de dropout limita la capacidad de aprendizaje del modelo en esta arquitectura específica. Por otro lado, al reducir el dropout a $p = 0$, osea eliminarlo, el modelo mostró una ligera mejora en la precisión de entrenamiento, con un sobreajuste muy similar al anterior, ver FIG 4. Dejaremos el valor inicial $P = 0.2$ para los siguientes experimentos.

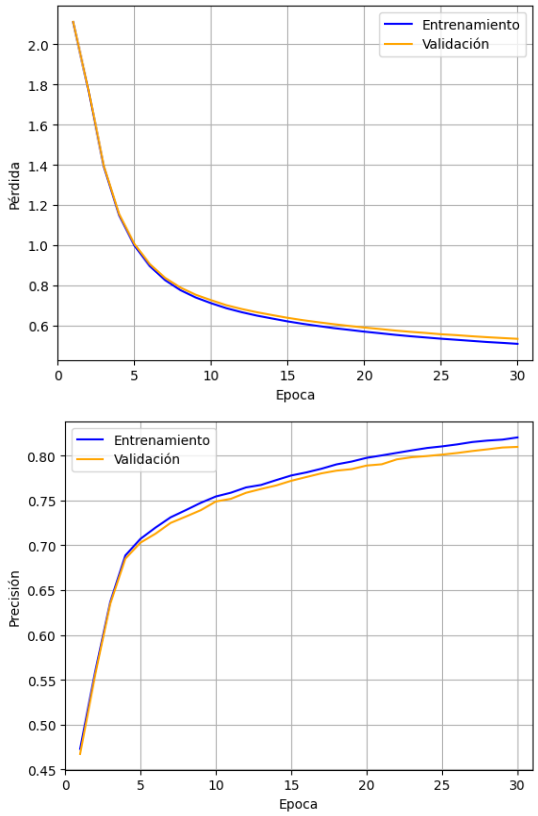

FIG. 4: Pérdida y Precisión sin dropout

C. Número de neuronas en las capas intermedias

El ajuste del número de neuronas en las capas intermedias tuvo un impacto significativo en el rendimiento del modelo. Al reducir el número de neuronas a $n_1 = 64, n_2 = 32$, el modelo mostró un descenso leve en la capacidad de aprendizaje, evidenciado por una mayor pérdida y menor precisión en ambos conjuntos, debido a la reducción en su capacidad para representar patrones complejos, ver en FIG 5. Por el contrario, al aumentar el número de neuronas a $n_1 = 256, n_2 = 128$, se observó una mejora escasa en la precisión de entrenamiento.

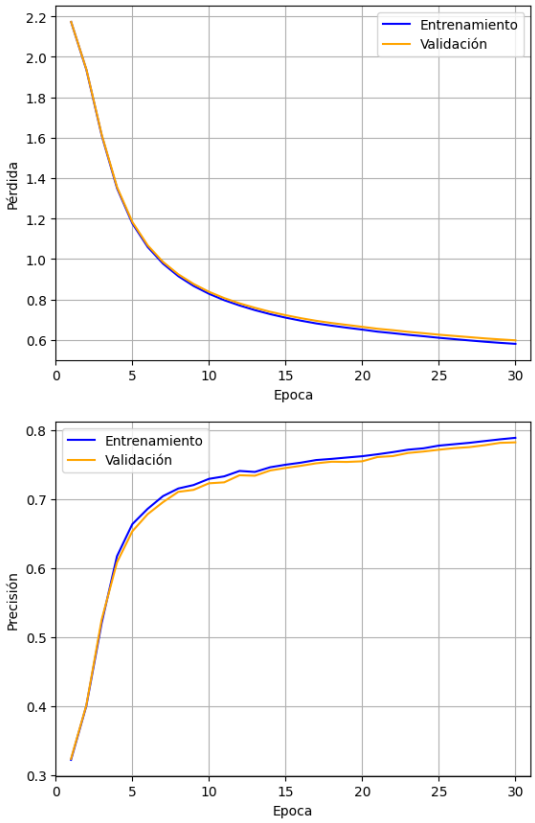

FIG. 5: Pérdida y Precisión con $n_1 = 64, n_2 = 32$

D. Optimizador

Al cambiar el optimizador a Adam, podemos ver en 6 que muestra ventajas en términos de rapidez de convergencia y mejor desempeño en el conjunto de entrenamiento y validación respecto a SGD, indicando una mayor capacidad de generalización. No obstante, el análisis revela que Adam también tiende a ajustarse más a los datos de entrenamiento, hay una mayor separación con respecto a las métricas de validación en las últimas épocas. Esta diferencia sugiere un riesgo de overfitting, donde el modelo optimizado con Adam podría estar capturando patrones específicos del conjunto de entrenamiento en detrimento de su capacidad de generalizar completamente a nuevos datos.

E. Batches

Se realizó una variación en el tamaño de los lotes (*batch size*), probando los valores de 50 y 200. Se observó que el lote de 50 introdujo una brecha leve en las curvas, pero obteniendo un mejor desempeño que en el entrenamiento inicial, ver FIG 7 . Por otro lado, un lote de 200 disminuyó dicha variabilidad, pero a costa de un aprendizaje más lento y una convergencia menos eficiente.

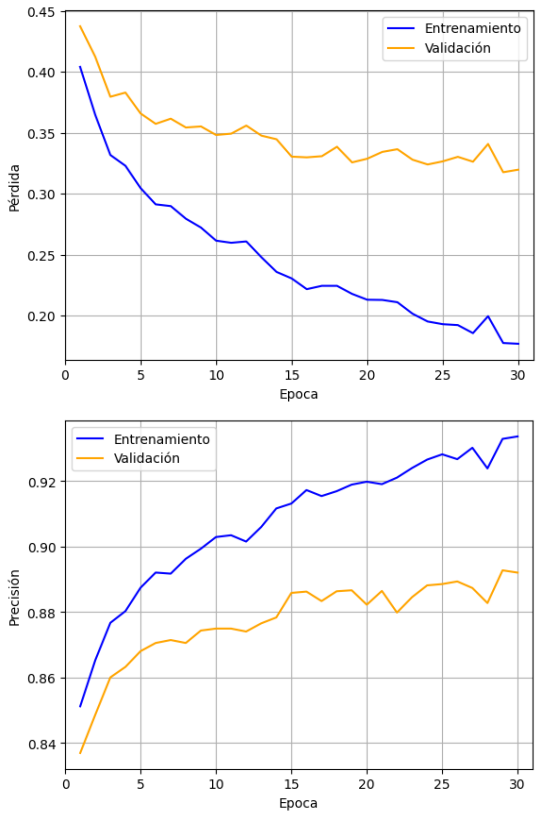

FIG. 6: Pérdida y Precisión con optim. Adam

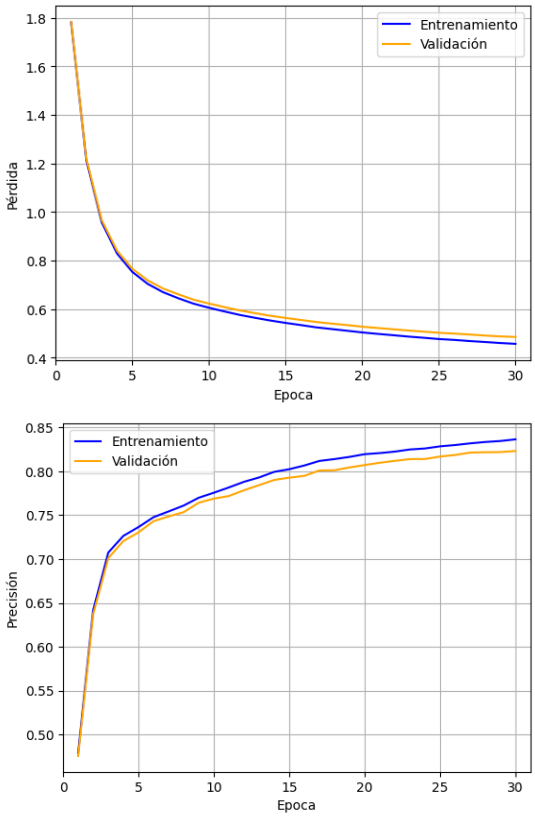

FIG. 7: Pérdida y Precisión con $batchsize = 50$ y SGD

IV. DISCUSIÓN

Los resultados obtenidos evidencian el impacto de cada hiperparámetro en la capacidad de la red para generalizar y aprender eficientemente. La tasa de aprendizaje influye en la velocidad y estabilidad de convergencia, resaltando la necesidad de un valor equilibrado. Un tamaño de batch intermedio ofrece un buen balance entre estabilidad y eficiencia computacional. Ajustar el dropout permite controlar el sobreajuste; valores bajos aumentan el riesgo de sobreajuste, mientras que valores altos reducen la precisión. El cambio de SGD a Adam mejora la convergencia, destacando la efectividad de optimizadores adaptativos. Finalmente, un número adecuado de neuronas en capas intermedias evita el sobreajuste sin limitar la precisión. En conjunto, estos ajustes optimizan el rendimiento del modelo al balancear aprendizaje y generalización.

V. CONCLUSIONES

Este trabajo destacó cómo el ajuste de los hiperparámetros afecta directamente el rendimiento de una red neuronal en tareas de clasificación. A través de la experimentación con el dropout, la tasa de aprendizaje, el optimizador, el tamaño del batch y la cantidad de neuronas en capas ocultas, se logró identificar configuraciones que permiten mejorar tanto la precisión como la capacidad de generalización. Los hallazgos obtenidos subrayan la importancia de optimizar cada parámetro en función de los datos y el objetivo del modelo.

VI. AGRADECIMIENTOS

IM agradece a los profes por la oportunidad y la orientación, que fueron clave para la realización de este trabajo.

* ignacio.martinez.goloboff@mi.unc.edu.ar

[1] Fashion-MNIST, Fashion-mnist — Wikipedia, the free encyclopedia (2024), [Online; accessed 19-November-2024].
[2] M. A. Nielsen, *Deep Learning* (Determination Press, 2016) [Online; accessed 19-November-2024].
[3] Feed-forward neural network, Wikipedia, the free encyclopedia (2024), [Online; accessed 19-November-2024].
[4] PyTorch, torch.optim — pytorch documentation (2024), [Online; accessed 19-November-2024].