Clasificación de Fashion-MNIST con una Red Neuronal Feedforward Multicapa en PyTorch

Juan R. Anabalón,[*] Ignacio Benjamín Ceballos,[**] and Bruno D'Ambrosio[***]
Facultad de Matemática, Astronomía, Física y Computación,
Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina and
Universidad Nacional de Cuyo, Padre Jorge Contreras 1300. Parque General San Martín. M5502JMA. Mendoza, Argentina

Presentamos la implementación de una red neuronal feedforward multicapa para clasificar imágenes del conjunto de datos Fashion-MNIST, utilizando PyTorch. El modelo se entrenó durante 30 épocas utilizando el algoritmo de optimización Adam. Para analizar el rendimiento se utiliza una función de perdida Cross Entropy Loss, cuyos resultados muestran una disminución consistente de la pérdida (tanto en entrenamiento como en validación) y un aumento en la precisión a lo largo de las épocas.

I. INTRODUCCIÓN

El objetivo del informe es implementar, entrenar y evaluar una Red Neuronal Artificial Feedforward Multicapa utilizando la librería PyTorch para clasificar prendas de vestir del dataset Fashion-MNIST [1]. Las redes neuronales feed-forward multicapa son un tipo de redes neuronales que procesan los datos de tal manera que fluyen en una sola dirección, sin bucles recursividad o ciclos. La clasificación de datos con respecto a este tipo de red funciona de la siguiente manera: Una imagen de entrada se propaga a través de la red pasando por cada capa, en cada uno de los nodos la información se transforma hasta llegar a la capa de salida mediante pesos y sesgos pasando también por una función de activación. Por último, la capa de salida realiza una predicción, esta se compara con la etiqueta real de la imagen usando una función de perdida, la cual cuantifica que tan mal resultó la predicción inicial. El error se propaga hacia atrás a traves de la red ajustando los pesos y sesgos de cada nodo con ayuda de un optimizador, que busca minimizar la función de pérdida a lo largo del tiempo, haciendo que las prediciones del modelo sean cada vez mas precisas. [1].

II. TEORÍA

Las redes neuronales feedforward multicapa (MLP, por sus siglas en inglés) son consideradas como uno de los modelos fundamentales del aprendizaje profundo. Su evolución se centra en la optimización de pesos, arquitecturas y funciones de activación, con aplicaciones en clasificación, regresión y reconocimiento de patrones [2]. En estas redes la información fluye en una sola dirección, desde la capa de entrada hacia la salida, pasando por una o más capas ocultas. El algoritmo de retropropagación (backpropagation) ha sido el método clásico para entrenarlas, permitiendo ajustar los pesos mediante gradiente descendente.

A. Conjunto de datos Fashion-MNIST

El conjunto de datos *Fashion-MNIST* contiene imágenes en escala de grises de 28x28 píxeles, que representan distintos artículos de ropa y accesorios. Estas imágenes muestran diferentes vistas (frontal, trasera o detalles) y pertenecen a diferentes categorías de productos (hombre, mujer, niño y neutro). Adoptamos el enfoque de mantener las 10 clases y 70.000 imágenes en escala de grises de tamaño 28 28 como en el MNIST original. El conjunto de datos fue dividido en un conjunto de 60 mil imágenes de entrenamiento y 10 mil imágenes de conjunto de validación. Cabe destacar que el dataset no incluye productos de color blanco, debido al bajo contraste con el fondo.

Las imágenes y etiquetas están almacenadas en el mismo formato de archivo que el conjunto de datos *MNIST* original, el cual está diseñado para vectores y matrices multidimensionales. Los ejemplos se encuentran ordenados por etiquetas, lo que reduce el tamaño de los archivos tras la compresión y facilita la recuperación de ejemplos pertenecientes a una clase específica.

B. Estructura de la red

La red es una arquitectura **feedforward totalmente conectada** de 4 capas:

$$\text{Flatten}(28 \times 28) \rightarrow \text{FC}_1 \rightarrow \text{FC}_2 \rightarrow \text{FC}_3 \qquad (1)$$

donde cada capa se compone de operaciones lineales seguidas de funciones de activación y regularización (excepto la capa de salida):

- **Capa de entrada:** Constituye el punto de entrada de la red. Cada neurona representa una característica del conjunto de datos. Esta capa no realiza cálculos por sí misma, sino que únicamente transmite los datos sin procesar. En este caso, cada imagen de 28x28 píxeles se aplana en un vector de 784 características.

- **Primera capa oculta:** Es una capa completamente conectada compuesta por $\mathbf{n}_1$ neuronas. Cada neurona recibe las 784 entradas de la capa anterior. Se aplica una función de activación *ReLU* para permitir el aprendizaje de relaciones no lineales, y una capa de *Dropout* como técnica de regularización, que desactiva temporalmente un porcentaje de neuronas durante el entrenamiento para prevenir el sobreajuste.
- **Segunda capa oculta:** También es completamente conectada, con $\mathbf{n}_2$ neuronas. Cada una recibe las salidas de la primera capa oculta, aplicando nuevamente la activación *ReLU* y una capa de *Dropout*.
- **Capa de salida:** está formada por 10 neuronas, correspondientes a las 10 clases del conjunto de datos *Fashion-MNIST*. Se utiliza la función de pérdida *Cross Entropy Loss*, que cuantifica la diferencia entre las probabilidades predichas por el modelo y la distribución real de etiquetas, permitiendo que el modelo asigne mayores probabilidades a las clases correctas.

C. Topología y Expresión Matemática

La red es una arquitectura feedforward totalmente conectada definida como:

$$\text{Flatten} \to \text{FC}_1(784, n_1) \to \text{FC}_2(n_1, n_2) \to \text{FC}_3(n_2, 10) \tag{2}$$

donde cada capa intermedia incluye normalización por lotes, activación ReLU y dropout:

$$\mathbf{z}_i = \mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b}_i \tag{3}$$

$$\mathbf{x}_i = \text{Dropout}(\text{ReLU}(\text{BatchNorm}(\mathbf{z}_i)), p) \quad \text{para } i = 1, 2 \tag{4}$$

$$\mathbf{y} = \text{softmax}(\mathbf{W}_3 \mathbf{x}_2 + \mathbf{b}_3) \tag{5}$$

Los parámetros $\mathbf{W}_i$ y $\mathbf{b}_i$ se optimizan mediante backpropagation.

D. Preprocesamiento

Las imágenes se transforman mediante:

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x}_{\text{PIL}}/255{,}0 - \mu}{\sigma} \tag{6}$$

donde:

- $\mathbf{x}_{\text{PIL}} \in [0, 255]$ es la imagen original (uint8)
- $\mu = 0{,}5$, $\sigma = 0{,}5$ (normalización estándar)
- $\mathbf{x}_{\text{norm}} \in [-1, 1]$ es la imagen normalizada

Cada imagen se convierte a tensor PyTorch con shape $\mathcal{C} = (1, 28, 28)$ (canal único, alto, ancho).

E. Métodos de optimización utilizados

Los experimentos se realizaron con dos optimizadores distintos: Adam (Adaptive Moment Estimation), y SGD (Descenso de Gradiente Estocástico), el objetivo de estos métodos es minimizar la función de pérdida bajo una tasa de aprendizaje establecida (tamaño de los pasos que se toman para alcanzar el mínimo).

SGD: Es un método de optimización iterativo que ejecuta una época de entrenamiento para cada ejemplo dentro del conjunto de datos y actualiza los parámetros de cada ejemplo de entrenamiento de uno en uno. De esta manera permite promediar una solución global y evitar el sobreajuste.

Adam: Es un algoritmo de optimización adaptativo que mantiene una tasa de aprendizaje separada para cada parámetro que se ajusta a medida que avanza el entrenamiento. Sabe tener una convergencia mas rápida y eficiente que SGD pero no ofrece la misma generalización.

III. RESULTADOS

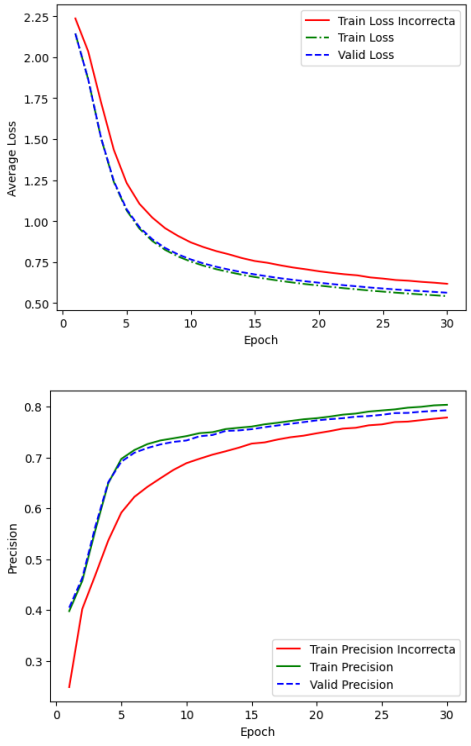

Figura 1: SGD - Resultados del entrenamiento con SGD y arquitectura inicial (n1 = 128, n2 = 64, $p = 0{,}2$).

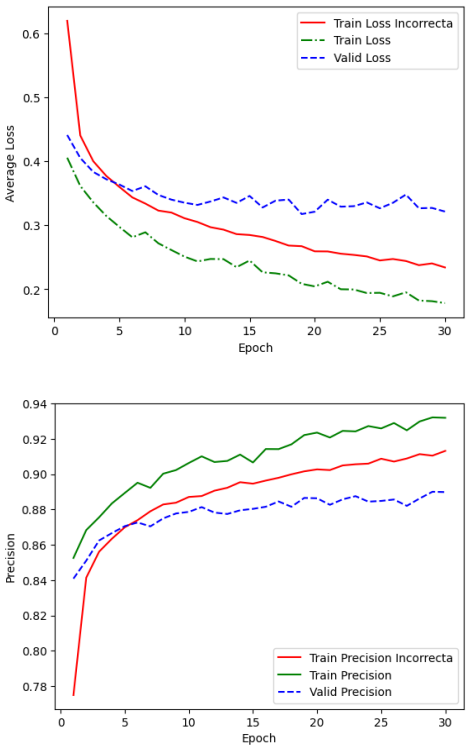

Figura 2: ADAM - Resultados del entrenamiento con ADAM y arquitectura inicial ($\mathrm{n1} = 128, \mathrm{n2} = 64, p = 0{,}2$).

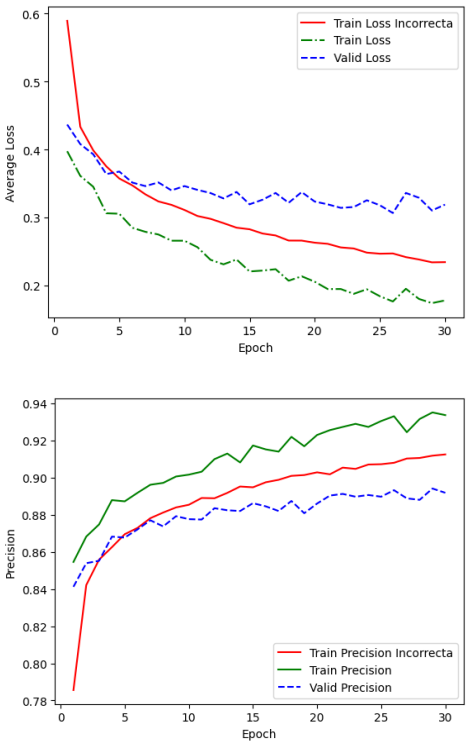

Figura 4: Resultados del entrenamiento con ADAM y arquitectura ampliada ($\mathrm{n1} = 256, \mathrm{n2} = 128, p = 0{,}3$).

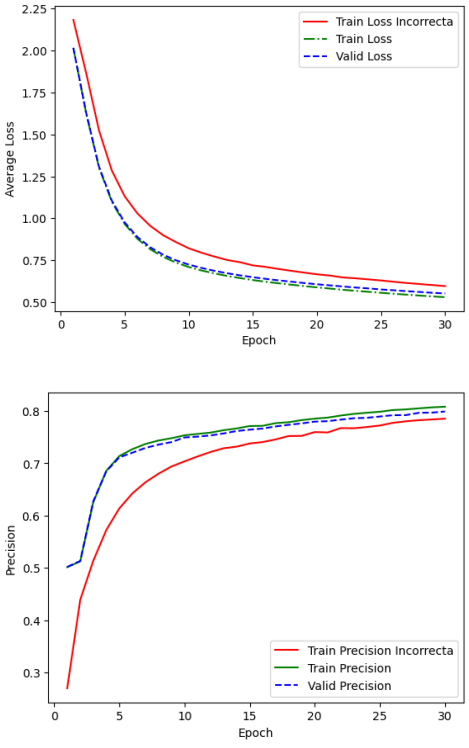

Figura 3: SGD - Resultados del entrenamiento con SGD y arquitectura ampliada ($\mathrm{n1} = 256, \mathrm{n2} = 128, p = 0{,}3$).

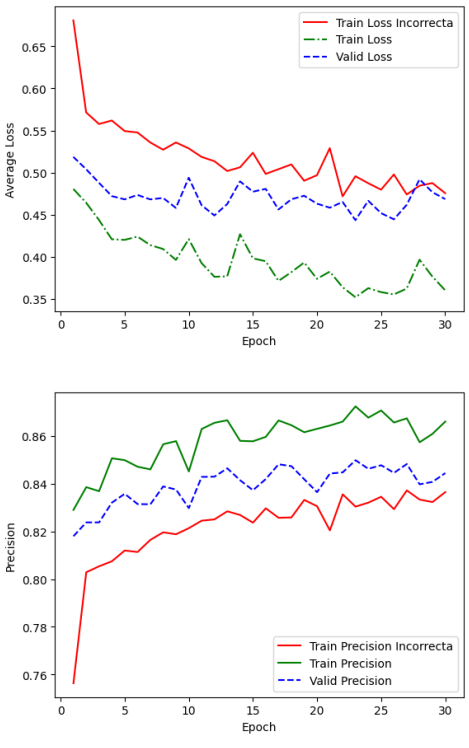

Figura 5: ADAM - Resultados del entrenamiento con ADAM y LR alto ($\mathrm{n1} = 128, \mathrm{n2} = 64, p = 0{,}2, \mathrm{LR} = 1e-2$).

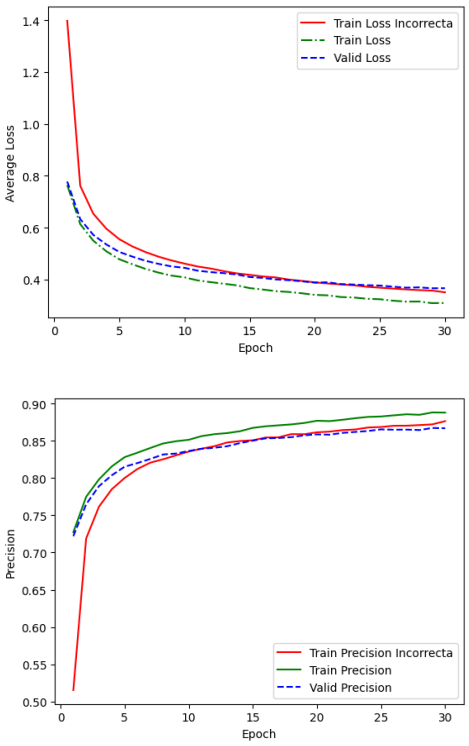

Figura 6: SGD - Resultados del entrenamiento con SGD y LR alto ($n1 = 128, n2 = 64, p = 0{,}2, \mathrm{LR} = 1e-2$).

IV. DISCUSIÓN

- **SGD vs ADAM - arquitectura inicial(figuras 1 y 2)**

  Aunque ambos optimizadores operan con el ruido que proviene del mini-batch estocástico junto con el dropout, la forma en que traducen ese ruido en un paso de actualización difiere fundamentalmente en que el SGD es .ªmortiguado"por su tasa de aprendizaje constante y baja.

- **SGD vs ADAM - learning rate mayor (lr = 1e-2) (Figuras 5 y 6)**

  SGD con LR alto tiene mejor rendimiento en este caso especifico. Ofrece una mejor estabilidad, menor pérdida final y la mayor precisión de validación entre las dos configuraciones de LR alto. SGD requiere más épocas, e indica una mejor generalización.

- **SGD vs ADAM - arquitectura ampliada(figuras 4 y 3)**

  En la arquitectura ampliada, ADAM es superior a SGD en términos de rendimiento final (pérdida y precisión), aunque el overfitting sea más pronunciado.

- **SGD arquitectura ampliada vs SGD arquitectura incial(figuras 1 y 3)**

  Para el optimizador SGD, aumentar la capacidad del modelo de (128,64) a (256,128) no justifica el incremento en complejidad y costo computacional, ya que las métricas finales apenas cambian.

- **ADAM arquitectura ampliada vs ADAM arquitectura incial(figuras 4 y 2)**

  Con el optimizador ADAM, la arquitectura ampliada no ofrece una mejora significativa en la precisión de validación y, al mismo tiempo, incrementa el riesgo de overfitting (observado en la mayor separación de las curvas de Train y Valid precisión).

Tabla I: Precisión de validación por caso y algoritmo de optimización (B.V.A = Best validation accuracy)

Arq	**B.V.A**
Figura 1 SGD (n1=128, n2=64, p=0.2)	80.6 %
Figura 2 ADAM (n1=128, n2=64, p=0.3)	80.10 %
Figura 3 SGD (n1=256, n2=128, p=0.3)	93.30 %
Figura 4 ADAM (n1=256, n2=128, p=0.3)	87.00 %
Figura 5 ADAM (n1=128, n2=64, p=0.2, lr=1e-2)	88.80 %
Figura 6 SGD (n1=128, n2=64, p=0.2, lr=1e-2)	93.04 %

V. CONCLUSIÓN

Al observar en la tabla los resultados, vemos que el mejor rendimiento se obtuvo mediante ADAM utilizando el n1=256, n2=128, p=0.3, con Accuracy = 89.09 %. Sin embargo, el experimento con la arquitectura inicial de ADAM esta muy cerca, aunque la arquitectura ampliada logra una pequeña mejora y una pérdida ligeramente menor, aunque como se observo aumenta el riesgo de overfitting. En contraste, el experimento con SGD y LR mas alto logra una precisión de validación de 86.8 % con una brecha de overfitting menor, lo que la convierte en una opción mas estable si es lo que se busca, pero el experimento con ADAM y arquitectura ampliada logra un mejor rendimiento final.

* juanrodrigo@anabalon.org; https://www.anabalon.org
** benjamin.ceballos@mi.unc.edu.ar
*** bruno.dambrosio@mi.unc.edu.ar

[1] V. Mehra, Explicación de las redes neuronales feedforward: Un tutorial completo.
[2] V. K. Ojha, A. Abraham, and V. Snášel, Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research (2017).