Reto (sin solución): Fashion-MNIST desde cero con NumPy

Curso de Aprendizaje Automático

1 de septiembre de 2025

Objetivo

Implementar **desde cero (solo NumPy)** un clasificador para **Fashion-MNIST** (10 clases de prendas). No se permite usar librerías de Deep Learning (Keras/TensorFlow/PyTorch/JAX). El reto incluye carga de datos, preprocesamiento, construcción de una **red neuronal multicapa**, entrenamiento con **mini-batches**, visualizaciones y análisis de errores.

Reglas y Alcance

- **Permitido**: `numpy`, `pandas`, `matplotlib`. (`sklearn` solo para métricas como matriz de confusión).
- **Prohibido**: Keras, TensorFlow, PyTorch, JAX u otras librerías de DL.
- **Datos**: Fashion-MNIST (formato MNIST: 28×28 gris, 10 clases). Use la versión CSV (columna `label` + 784 píxeles) o convierta desde `idx`/`npz`.
- **Entrega**: código reproducible + figuras + breve informe (README).

Clases (para títulos en gráficas)

0=T-shirt/top, 1=Trouser, 2=Pullover, 3=Dress, 4=Coat, 5=Sandal, 6=Shirt, 7=Sneaker, 8=Bag, 9=Ankle boot

Estructura sugerida de proyecto

```
fashion_reto/
 data/                  # train.csv (o .npz/.idx)
 figs/                  # imágenes de salida
 src/
   reto_fashion.py     # implementación principal
   utils.py            # helpers (opcional)
 README.md              # cómo ejecutar y hallazgos
```

Entregables

Guarde las figuras en `figs/` con estos nombres:

- `fig_fm_imagen_0_con_valores.png`
- `fig_fm_10_ejemplos.png`
- `fig_fm_loss_train_val.png`
- `fig_fm_confusion_matrix.png`
- `fig_fm_errores_grid.png`

Reporte en README: accuracy en *train/valid/test* y 2–3 observaciones sobre confusiones típicas.

Pasos (qué hacer, sin solución)

0) Preparación

1) Cree un entorno e instale: `numpy pandas matplotlib`.
2) Fije semilla: `np.random.seed(100)`.

1) Carga + EDA breve

a) Lea `data/train.csv`. Separe: `y = df['label'].values`, `X_raw = df.drop('label', axis=1).values`.
b) Imprima `X_raw.shape`, distribución de clases.
c) Visualice:
   - Imagen 0 (28×28) con **valores por píxel** (0–255) sobre la celda.
   - Cuadrícula 2×5 con los **primeros 10** ejemplos (título: nombre de clase).

2) Preprocesamiento

a) Normalice: `X = X_raw.astype(np.float32)/255.0`.
b) Codifique etiquetas con **One-Hot**: `Y = one_hot(y, num_classes=10)`.
c) Divida en **train/valid/test** (80/10/10). Si no usa `sklearn`, baraje índices manualmente.

3) Arquitectura

Use base: **784** $\rightarrow$ **64** $\rightarrow$ **32** $\rightarrow$ **10**. Active en ocultas con **ReLU** o **Sigmoid**. En salida, **Softmax** (recomendado) y pérdida **Cross-Entropy**.

4) Inicialización

Inicialice pesos pequeños (escala 0.01) y sesgos en 0. Imprima número de parámetros por capa.

5) Forward

Implemente `forward(X, params)` que devuelva:

- `P`: probabilidades de clase por lote.

- `cache`: diccionario con intermedios (`a0,z0,a1,z1,a2,z2`) para backprop.

6) Pérdida

Implemente **Cross-Entropy** multiclase con estabilidad numérica (restar `max` por fila antes de `exp`).

7) Backpropagation

Calcule gradientes para todas las capas. Con Softmax+CE:

$$\delta^{(L)} = P - YB, \quad \delta^{(1)} = (\delta^{(L)} W_2^\top) \odot g'(z^{(1)}), \quad \delta^{(0)} = (\delta^{(1)} W_1^\top) \odot g'(z^{(0)})$$

donde g es la activación de las ocultas y B el tamaño de *batch*.

8) Entrenamiento (mini-batch)

Baraje cada época. Para cada mini-batch: `forward` → `loss` → `backward` → `update`. Registre `loss_train` y `loss_val`. Grafique y guarde `fig_fm_loss_train_val.png`.

9) Evaluación + Errores

- Calcule **accuracy** en train/valid/test.
- Genere **matriz de confusión** (`sklearn.metrics.confusion_matrix` permitido).
- Muestre 12 errores en cuadrícula 3×4 con títulos: `Real=<nombre>, Pred=<nombre>`. Guarde `fig_fm_errores_grid.png`.

10) Experimentos (elija 2+)

Cambie tamaño de capas (p.ej., 128/64), compare **ReLU vs Sigmoid**, pruebe distintas tasas de aprendizaje, agregue **L2** o **early stopping**. Documente hallazgos en README.

Plantilla (esqueleto, sin solución)

Listing 1: Esqueleto de src/reto$_fashion.pyconTODOs$

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

CLASS_NAMES = ["T-shirt/top","Trouser","Pullover","Dress","Coat",
               "Sandal","Shirt","Sneaker","Bag","Ankle boot"]

# ---------- utils ----------
def one_hot(y, num_classes=10):
    # TODO: devolver matriz (N,num_classes) one-hot.
    raise NotImplementedError

def relu(z):
```

```
    # TODO: implementar ReLU.
    raise NotImplementedError

def relu_derivative(z):
    # TODO: derivada de ReLU.
    raise NotImplementedError

def softmax(z):
    # TODO: softmax estable num ricamente.
    raise NotImplementedError

def cross_entropy(y_true, y_pred):
    # TODO: p rdida CE multiclase estable.
    raise NotImplementedError

# ---------- modelo ----------
def init_params(input_dim=784, h1=64, h2=32, num_classes=10,
   scale=0.01):
    # TODO: inicializar W0,W1,W2 y b0,b1,b2.
    raise NotImplementedError

def forward(X, params, activation="relu"):
    # TODO: calcular a0,z0,a1,z1,a2,z2 y salida P.
    # Usa 'activation' para ocultas; salida softmax.
    # Retorna P y cache.
    raise NotImplementedError

def backward(Y, cache, params, activation="relu"):
    # TODO: gradientes con softmax+CE:
    #   delta2 = (P - Y) / B
    #   delta1 = (delta2 @ W2.T) * g'(z1)
    #   delta0 = (delta1 @ W1.T) * g'(z0)
    raise NotImplementedError

def update(params, grads, lr):
    # TODO: descenso por gradiente.
    raise NotImplementedError

def accuracy(y_true_labels, y_pred_probs):
    # TODO: argmax y promedio de aciertos.
    raise NotImplementedError

# ---------- visualizaciones obligatorias ----------
def show_first_image_with_values(X_raw, y, outpath="figs/
   fig_fm_imagen_0_con_valores.png"):
    # TODO: imagen 0 con valores 0..255 impresos en cada pixel.
    raise NotImplementedError

def show_first_10_grid(X_raw, y, outpath="figs/fig_fm_10_ejemplos
   .png"):
    # TODO: cuadr cula 2x5 con t tulos CLASS_NAMES[y[i]].
```

```
    raise NotImplementedError

def show_errors_grid(X_raw, y_true, y_pred, k=12, outpath="figs/
    fig_fm_errores_grid.png"):
    # TODO: 3x4 ejemplos err neos con t tulos 'Real= , Pred='.
    raise NotImplementedError

# ---------- entrenamiento ----------
def train(X_train, Y_train, X_val, Y_val, params, lr=0.01,
    batch_size=128, epochs=20, activation="relu"):
    # TODO: bucle mini-batch:
    #   - barajar por  poca
    #   - forward -> loss -> backward -> update
    #   - registrar p rdidas y devolver history
    raise NotImplementedError

def main():
    np.random.seed(100)
    df = pd.read_csv("data/train.csv")
    y = df["label"].values
    X_raw = df.drop("label", axis=1).values.astype(np.float32)

    show_first_image_with_values(X_raw, y)
    show_first_10_grid(X_raw, y)

    X = X_raw / 255.0
    Y = one_hot(y, num_classes=10)

    # TODO: split train/val/test
    # TODO: params = init_params()
    # TODO: history = train(...)
    # TODO: evaluaci n + matriz de confusi n + show_errors_grid
        (...)
    # TODO: guardar curva de p rdidas

if __name__ == "__main__":
    main()
```

Criterios de evaluación (sugerencia)

- **Correctitud técnica (40 %)**: forward/backprop correctos, convergencia, métricas.
- **Calidad de código (20 %)**: claridad, modularidad, comentarios, reproducibilidad.
- **Visualizaciones (20 %)**: figuras solicitadas, legibles y guardadas.
- **Análisis (20 %)**: interpretación de errores y propuestas de mejora.

Cómo compilar

Ejecute: `pdflatex reto_fashion_mnist.tex` (dos pasadas recomendadas).

Nota: Este documento describe el *reto* y proporciona una plantilla con `TODOs`. No incluye soluciones.