Implementación desde Cero de un Motor de Redes Neuronales Feedforward para IRIS, MNIST y Students Performance

Denys Kavkalo Gumeniuk Manuel Alejandro Torrealba Torrealba

7 de noviembre de 2025

Resumen

Se presenta un motor de redes neuronales *feedforward* implementado íntegramente en Python y NumPy, sin librerías de *deep learning*. La librería ofrece capas densas configurables, funciones de activación (Sigmoid, ReLU, Softmax), pérdidas (MSE y Cross-Entropy), inicialización de pesos (Xavier), entrenamiento por *mini-batches* y optimizadores SGD y Adam. La biblioteca se separa de los cuadernos experimentales y se acompaña de `utils` y `data loaders` propios. La validación se realiza en tres conjuntos de datos: IRIS (clasificación multiclase), MNIST (reconocimiento de dígitos manuscritos) y Students Performance (regresión). Los resultados muestran evidencia clara de aprendizaje y generalización: 100 % de precisión en IRIS, ∼97.5 % en MNIST y disminución sostenida de MSE en Students. El documento incluye la deducción mínima de ecuaciones de backpropagation y el esquema de optimización empleado, junto con pseudocódigo de alto nivel. La modularidad permite extender el motor con nuevas arquitecturas, regularización y *schedulers* de tasa de aprendizaje.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

El objetivo es construir y analizar un motor de redes neuronales desde cero, garantizando comprensión de los mecanismos matemáticos y de su traducción a código vectorizado. Se busca cumplir una implementación reproducible, modular y extensible, y verificarla en problemas de clasificación y regresión.

2. Métodos y Conjuntos de Datos

2.1. Formulación del modelo

Para cada capa l con activación σ se tiene

$$Z^{(l)} = W^{(l)} A^{(l-1)} + b^{(l)}, \tag{1}$$

$$A^{(l)} = \sigma\Big(Z^{(l)}\Big). \tag{2}$$

En la salida de clasificación se usa Softmax y pérdida de entropía cruzada:

$$\hat{Y} = \mathrm{softmax}(Z), \qquad \mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} Y_{ic}\log \hat{Y}_{ic}. \tag{3}$$

La retropropagación resulta

$$\delta^{(L)} = \hat{Y} - Y, \qquad \delta^{(l)} = \Big(W^{(l+1)}\Big)^{\top} \delta^{(l+1)} \odot \sigma'\Big(Z^{(l)}\Big), \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} \Big(A^{(l-1)}\Big)^{\top}, \qquad \frac{\partial \mathcal{L}}{\partial b^{(l)}} = \sum_{i} \delta_i^{(l)}. \tag{5}$$

2.2. Optimización

SGD aplica $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$. Adam mantiene momentos (m_t, v_t) con corrección de sesgo y actualiza

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \tag{6}$$

$$\theta \leftarrow \theta - \eta \frac{m_t/(1-\beta_1^t)}{\sqrt{v_t/(1-\beta_2^t)} + \epsilon}. \tag{7}$$

2.3. Algoritmo general (pseudocódigo)

Algorithm 1: Entrenamiento con mini-batches y optimizador genérico.

Inicializar pesos (Xavier)
for *época* $= 1 \dots T$ **do**
  Particionar datos en mini-batches
  for *cada mini-batch* B **do**
    $A \leftarrow \text{forward}(B)$; $\mathcal{L} \leftarrow \text{loss}(B, A)$; $G \leftarrow \text{backward}(\partial\mathcal{L}/\partial A)$;
    optimizer.update(θ, G)
  end
  Evaluar en validación
end

2.4. Conjuntos de datos

IRIS: 150 instancias, 4 atributos, 3 clases. Normalización min–max y *one-hot*.
MNIST: 70,000 imágenes 28×28 descargadas desde repositorio público, a vectores de 784 y normalización a $[0, 1]$.
Students Performance: predicción de notas finales a partir de variables socioeducativas (regresión).

3. Detalles de Implementación

La librería se organiza en módulos: `layers.py` (capa densa), `activations.py` (Sigmoid, ReLU, Softmax), `losses.py` (MSE, CE), `optimizers.py` (SGD, Adam), `network.py` (encadenado de capas), `trainer.py` (bucle de entrenamiento, validación y *mini-batches*) y `utils.py` (normalización, *one-hot*, métricas y gráficas). La derivada combinada CE+Softmax se aplica en la última capa para estabilidad.

4. Experimentos y Resultados

Se usa un protocolo sistemático: división *train/val/test*, semilla fija y registro de *loss* y *accuracy*.

4.1. IRIS (clasificación)

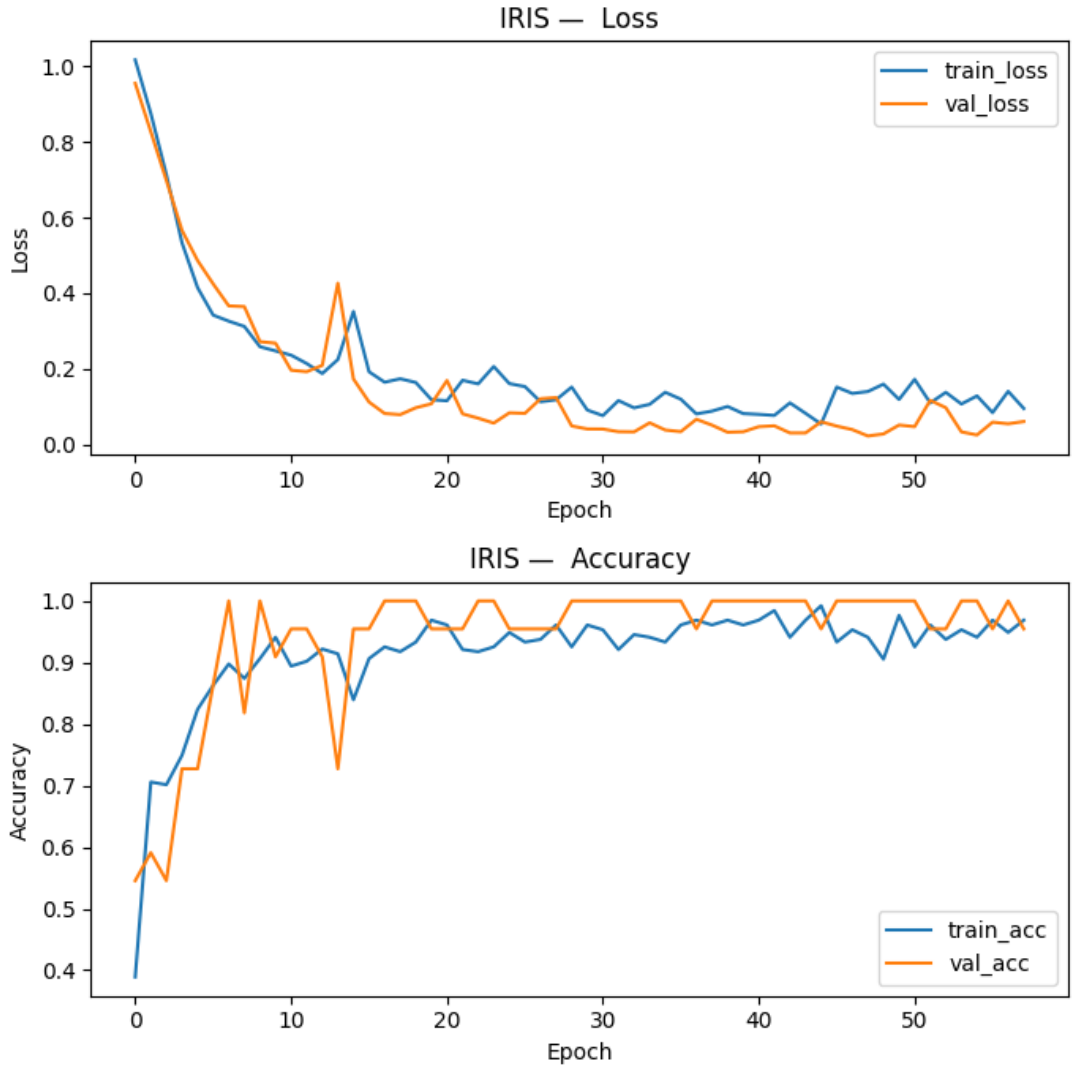

Figura 1: Curvas de pérdida y precisión en IRIS.

En la Figura 1 se observa la evolución del entrenamiento. Las curvas de pérdida (arriba) para entrenamiento (`train_loss`) y validación (`val_loss`) descienden consistentemente, indicando un aprendizaje efectivo. Paralelamente, las curvas de precisión (abajo) ascienden rápidamente, alcanzando y manteniéndose cerca del 100 % (`val_acc`) en pocas épocas. La volatilidad observada, especialmente en validación, es esperable dado el reducido tamaño del conjunto de datos IRIS, donde fallos en unas pocas instancias alteran significativamente la métrica.

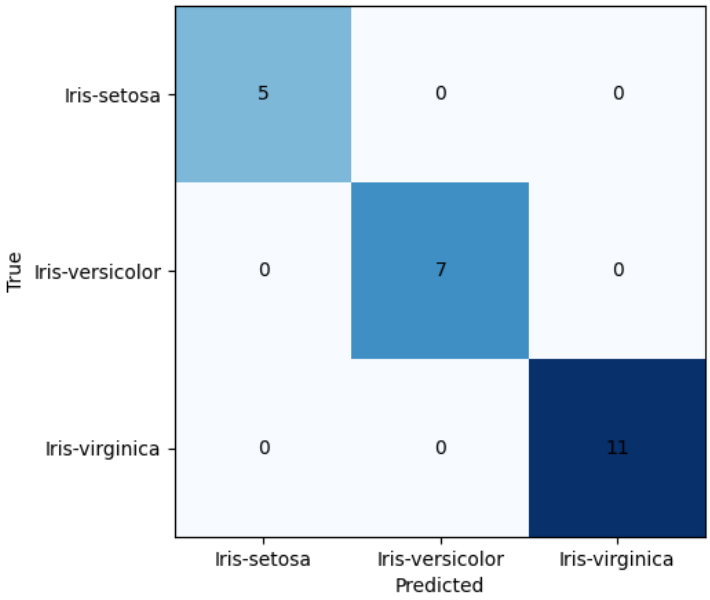

Figura 2: Matriz de confusión en IRIS. Resultados perfectos en el conjunto de test.

La Figura 2 presenta la matriz de confusión resultante sobre el conjunto de *test*. Se obtiene una matriz diagonal perfecta, con 5, 7 y 11 instancias clasificadas correctamente para Iris-setosa, Iris-versicolor e Iris-virginica, respectivamente. Esto confirma una precisión final del 100 % en datos no vistos, validando el correcto funcionamiento del optimizador SGD y la implementación general en un problema separable.

4.2. MNIST (clasificación)

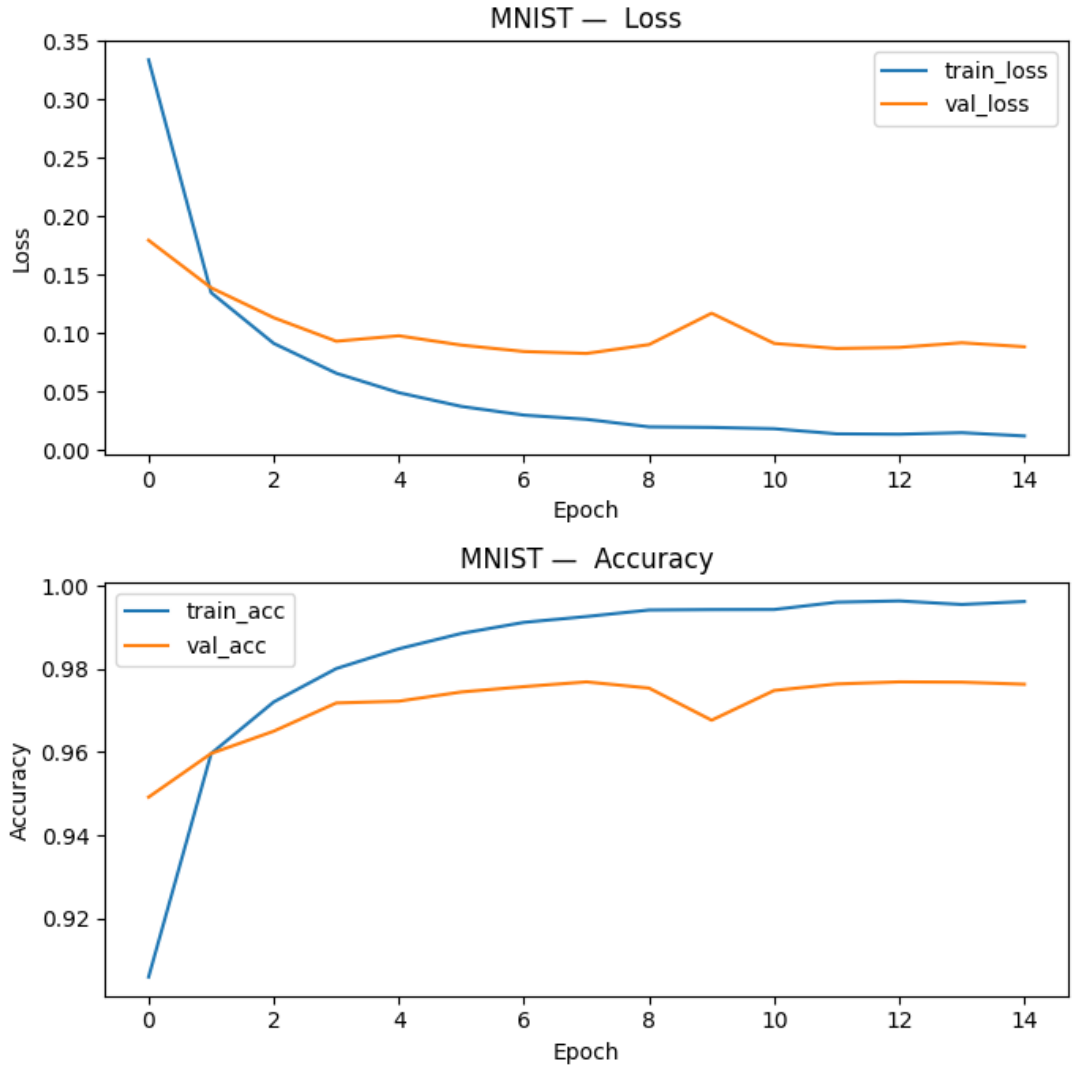

Figura 3: Curvas de pérdida y precisión en MNIST.

Las curvas de entrenamiento para MNIST (Figura 3), un problema de mayor complejidad, muestran un comportamiento de aprendizaje robusto usando el optimizador Adam. La pérdida de entrenamiento (arriba, `train_loss`) desciende de forma suave y continua, mientras que la pérdida de validación (`val_loss`) la acompaña, estabilizándose en un valor bajo. Esto indica una buena generalización, con una ligera brecha (gap) entre ambas, esperable en este problema. Las curvas de precisión (abajo) son consistentes, con la precisión en validación (`val_acc`) estabilizándose en torno al 97.5 %, tal como se resume en el Abstract y la Tabla 1.

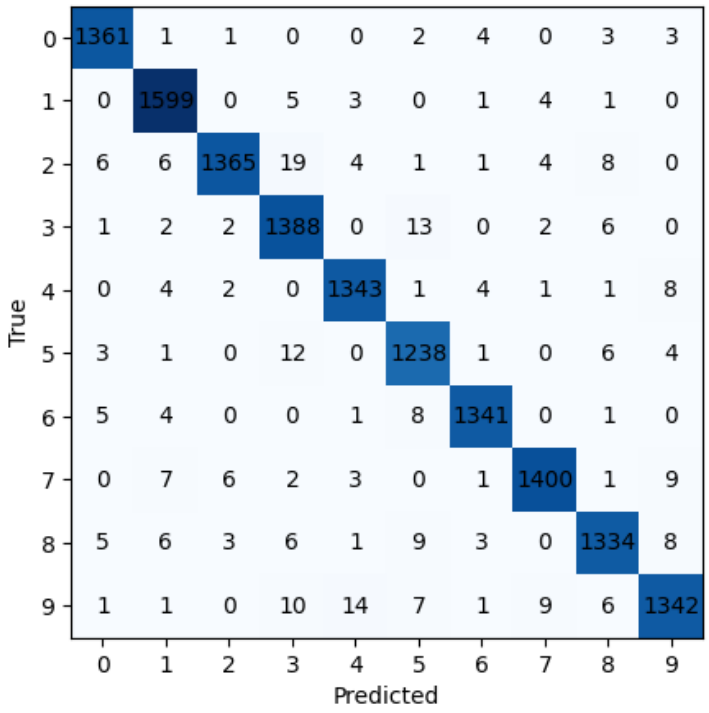

Figura 4: Matriz de confusión en MNIST.

La matriz de confusión del conjunto de *test* (Figura 4) corrobora este alto rendimiento. La diagonal principal concentra la gran mayoría de las predicciones (p.ej., 1361 aciertos para el '0', 1599 para el '1'). Los errores (valores fuera de la diagonal) son mínimos y se distribuyen de forma plausible (p.ej., 19 instancias del '2' confundidas con '3', o 14 del '9' con '4'), errores que son visualmente comprensibles.

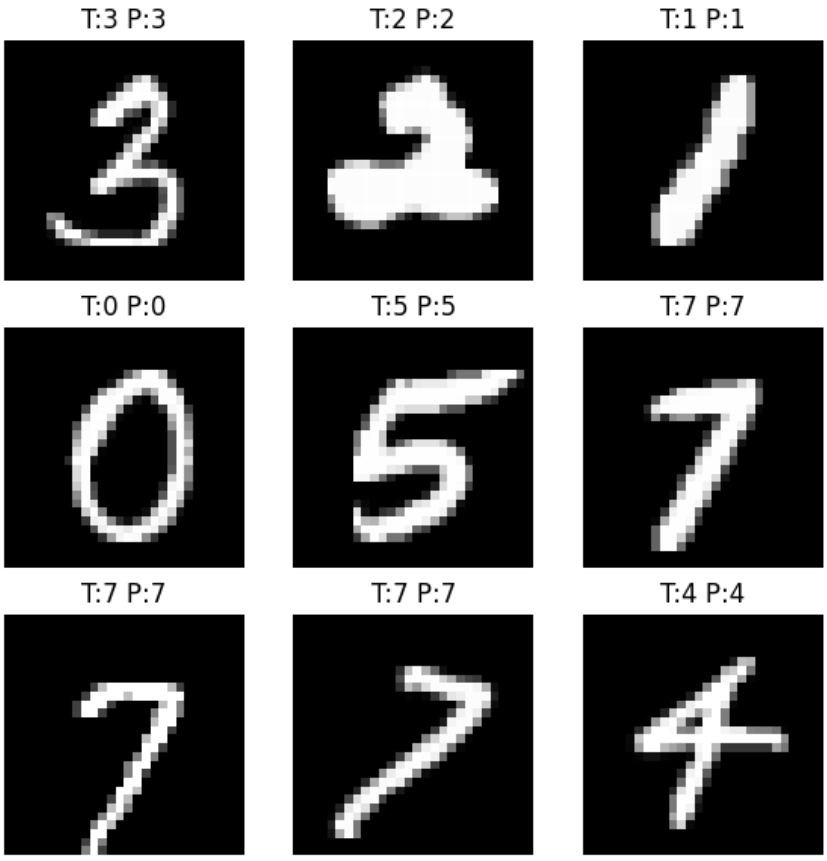

Figura 5: Ejemplos de predicción en MNIST (T: etiqueta real, P: predicción).

Finalmente, la Figura 5 ilustra ejemplos cualitativos de la predicción. Se muestran 9 imágenes del conjunto de *test* donde la etiqueta real (T) y la predicción del modelo (P) coinciden en todos los casos expuestos, demostrando la capacidad del motor para identificar correctamente los dígitos manuscritos.

4.3. Students Performance (regresión)

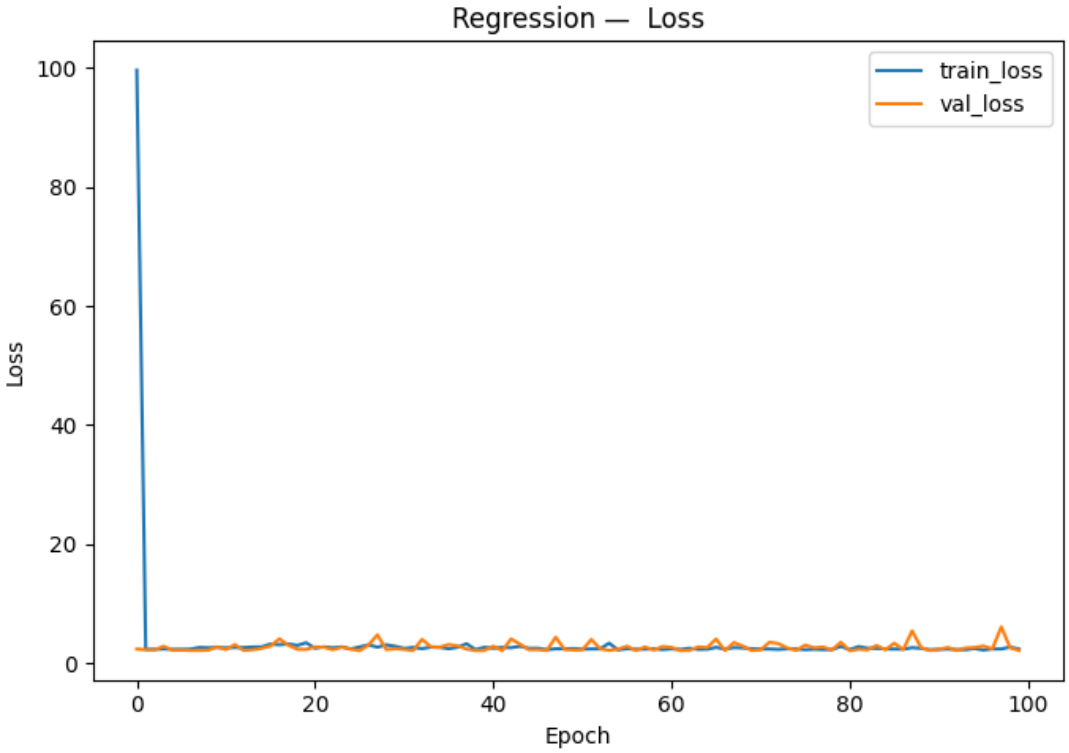

Figura 6: Evolución de la pérdida en entrenamiento y validación (MSE).

Para el problema de regresión, la Figura 6 muestra la evolución de la pérdida (MSE). Se aprecia un pico inicial muy elevado en la primera época, probablemente debido a la inicialización de pesos antes del primer ajuste con Adam. Inmediatamente después, tanto el MSE de entrenamiento (`train_loss`) como el de validación (`val_loss`) caen drásticamente y se estabilizan en un valor bajo, oscilando muy cerca el uno del otro, aunque obteniendo algunos picos no muy grandes a lo largo de las epocas. Este solapamiento casi perfecto sugiere que el modelo aprende la relación subyacente sin sufrir de sobreajuste en este conjunto de datos.

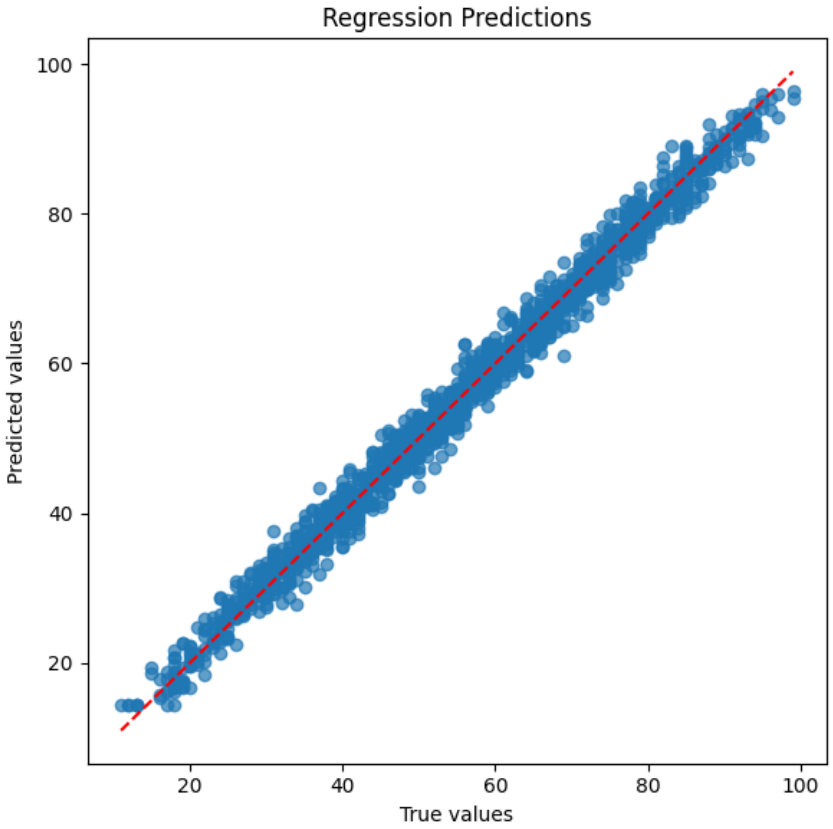

Figura 7: Dispersión predicción vs. valor real con línea identidad.

La Figura 7 ofrece una validación visual de la regresión. Este gráfico de dispersión compara los valores reales (*True values*) con los valores predichos por el modelo (*Predicted values*) en el conjunto de *test*. Los puntos se agrupan de forma compacta y lineal alrededor de la línea de identidad (línea roja discontinua, $y = x$). Esta fuerte correlación positiva indica que las predicciones del modelo son muy cercanas a los valores reales, confirmando el éxito del motor en la tarea de regresión.

Dataset	Tarea	Optimizador	Métrica test
IRIS	Clasificación	SGD	1.00 accuracy
MNIST	Clasificación	Adam	0.975 accuracy
Students	Regresión	Adam	MSE decreciente

Cuadro 1: Resumen de resultados en los tres conjuntos.

Finalmente, la Tabla 1 sintetiza los resultados clave obtenidos en el conjunto de *test* para cada experimento. Esta tabla consolida la métrica de rendimiento final, el tipo de tarea (clasificación o regresión) y el optimizador empleado. Los resultados demuestran que el motor implementado es capaz de manejar diferentes optimizadores (SGD y Adam) y resolver con éxito tanto problemas de clasificación multiclase (IRIS y MNIST), alcanzando métricas de precisión elevadas, como tareas de regresión (Students), logrando una reducción consistente del error (MSE).

5. Conclusiones y Trabajo Futuro

La implementación demuestra aprendizaje estable y generalización en clasificación y regresión. El diseño modular permite extender el motor con regularización (L2, *dropout*), *early stopping*, *learning-rate schedulers* y arquitecturas (p.ej., CNN).
Dificultades: manejo de dimensiones en backpropagation, normalización consistente entre particiones y depuración del último *batch*.

6. Bibliografía

Referencias

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. MIT Press, 2016.