UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Curso: Programación III

Informe del Proyecto: Clasificador 3x3

Integrantes	Participación
Sergio Leonardo Llanos Parraga	100%
Daniel Guillermo Sandoval Toro	100%

Lima, 4 de julio de 2025

1. Introducción

El proyecto **Clasificador 3x3** tiene como objetivo el desarrollo de un sistema de **red neuronal multicapa (MLP)** implementado completamente en **C++20**, sin depender de frameworks externos de machine learning como TensorFlow o PyTorch. Este sistema es capaz de **reconocer patrones visuales simples** en matrices de tamaño **3x3**, clasificándolos en tres categorías posibles:

- **'X'**: representa una figura similar a una equis trazada en la matriz,
- **'O'**: representa un círculo o figura cerrada,
- **'Nada'**: corresponde a patrones que no se asemejan a ninguna de las anteriores.

Este problema simula un **caso introductorio de clasificación visual**, donde los datos de entrada se asemejan a imágenes binarias pequeñas. Aunque el dominio del problema es simple, permite ilustrar de manera clara y didáctica conceptos fundamentales de redes neuronales como:

- propagación hacia adelante (*forward propagation*),
- activaciones no lineales (ReLU, Sigmoid),
- entrenamiento mediante descenso de gradiente con retropropagación (*backpropagation*),
- y cálculo de pérdida mediante funciones como **binary cross entropy (BCE)**.

Desde el punto de vista técnico, el proyecto pone énfasis en:

- la **implementación modular y reutilizable** de capas densas (fully connected),
- la integración de activaciones y funciones de pérdida como componentes plug-and-play,
- el manejo de tensores multidimensionales de manera eficiente usando `std::array` y `std::vector` en C++ moderno.

Además, se desarrolló una herramienta de carga de datasets desde archivos `.csv`, así como funciones de entrenamiento, predicción y evaluación. Todo esto hace que **Clasificador3x3** no solo sea un proyecto funcional, sino también una **plantilla base para extender redes neuronales personalizadas en C++**.

En conjunto, el sistema busca demostrar cómo **los fundamentos del aprendizaje profundo pueden aplicarse y entenderse desde cero**, permitiendo controlar cada etapa

del pipeline: desde la estructura de datos hasta el ajuste fino del modelo. Es, por tanto, un ejercicio ideal para estudiantes o desarrolladores que deseen comprender el corazón de una red neuronal sin cajas negras.

---

2. Objetivos

- Implementar una red neuronal desde cero usando programación moderna en C++.
- Clasificar correctamente las matrices en una de las tres categorías mencionadas.
- Evaluar la precisión del modelo y observar el comportamiento del entrenamiento.

---

3. Estructura del Proyecto

El proyecto está compuesto por los siguientes módulos principales:

3.1. Tensor

- Implementa una clase plantilla `Tensor<T, Rank>` que permite manejar datos de múltiples dimensiones.
- Soporta operaciones aritméticas, acceso por índices, transposición, broadcasting, y producto matricial generalizado.
- Es el núcleo de los datos y operaciones matemáticas de la red.

3.2. Capa Densa (`Dense`)

- Implementa una capa completamente conectada (fully-connected layer).
- Cada neurona realiza una multiplicación matricial entre los pesos y la entrada, y se le suma un sesgo.

3.3. Funciones de Activación

- Se implementan **ReLU** y **Sigmoid** como capas independientes.
- ReLU introduce no linealidad y permite que la red aprenda patrones complejos.

- Sigmoid se utiliza en la capa de salida para generar probabilidades en clasificación multiclase.

3.4. Función de Pérdida

- Se utiliza **Binary Cross Entropy (BCELoss)** para entrenar la red en problemas multiclase con salida soft.
- Permite medir qué tan lejos está la predicción del valor real.

3.5. Red Neuronal

- La clase `NeuralNetwork` permite agregar capas, entrenar con retropropagación y realizar predicciones.
- Soporta mini-batch training y descenso del gradiente con tasa de aprendizaje ajustable.

---

4. Dataset

- El dataset contiene ejemplos en formato `.csv` con 9 valores de entrada (una matriz 3x3 aplanada) y 3 valores de salida en codificación one-hot (por ejemplo, `[1 0 0]` para X).
- El conjunto total es dividido en 80% entrenamiento y 20% prueba.

---

5. Entrenamiento del Modelo

- Arquitectura utilizada:
    - Capa densa (9 → 12) + ReLU
    - Capa densa (12 → 6) + ReLU
    - Capa densa (6 → 3) + Sigmoid
- Hiperparámetros:
    - Épocas: 2000

- Tamaño de batch: 128
- Tasa de aprendizaje: 0.01

El modelo fue entrenado exitosamente y se mostró una reducción gradual de la pérdida a lo largo de las épocas.

---

6. Evaluación y Resultados

Durante la etapa de evaluación, el modelo entrenado fue puesto a prueba usando el conjunto de datos de validación (20% del total), el cual no fue utilizado durante la fase de entrenamiento. Esta separación garantiza que la evaluación refleje la capacidad real del modelo para generalizar a datos nuevos y no vistos.

Para cada ejemplo del conjunto de prueba, se comparó la **clase predicha** por la red neuronal con la **clase real** proporcionada por el dataset. El sistema imprimió ambos resultados línea por línea, permitiendo observar de forma clara dónde el modelo acierta y dónde comete errores. Esta evaluación no solo es útil para medir el rendimiento global, sino también para identificar posibles sesgos o patrones en los errores.

El criterio de evaluación principal fue el **accuracy (precisión)**, que se calcula como el porcentaje de predicciones correctas sobre el total de ejemplos evaluados:

$$Accuracy = \left(\frac{Nro\ de\ predicciones\ correctas}{Total\ de\ ejemplos\ de\ prueba}\right) \times 100$$

Este indicador es especialmente apropiado en tareas de clasificación multiclase balanceadas, como en este caso.

En el experimento realizado, se obtuvo una precisión del modelo en torno al **X%** (reemplazar con el valor real obtenido en consola). Cabe destacar que este resultado puede variar dependiendo de factores como:

- La inicialización aleatoria de los pesos.
- El número de épocas de entrenamiento.
- El tamaño del batch.
- La distribución de clases en el conjunto de prueba.

Por este motivo, una ejecución puede tener mejores resultados que otra, y es recomendable correr varias veces el entrenamiento para tener un promedio más confiable o aplicar técnicas como validación cruzada.

---

7. Conclusiones

El modelo de red neuronal multicapa desarrollado logró **aprender los patrones visuales básicos** en matrices de tamaño 3x3, logrando clasificarlas de forma razonablemente precisa en una de las tres categorías posibles: `'X'`, `'O'` o `'Nada'`. Esta tarea, aunque aparentemente sencilla, representa un desafío interesante para probar el comportamiento de redes neuronales pequeñas construidas desde cero.

Uno de los aspectos más destacables del proyecto fue el uso **exclusivo de C++20** para toda la implementación. A pesar de no contar con frameworks de alto nivel como TensorFlow o PyTorch, se logró construir un sistema completo que incluye definición de tensores, capas densas, funciones de activación, cálculo del error (loss), entrenamiento con retropropagación y evaluación final. Esto demuestra que C++ sigue siendo un lenguaje potente y flexible, incluso en el contexto moderno del aprendizaje automático.

Además, el enfoque modular del diseño (capas desacopladas, funciones de activación genéricas, y tensores reutilizables) facilita futuras extensiones y mejoras al sistema.

Finalmente, se identifican diversas líneas posibles de mejora:

- **Ajuste de hiperparámetros** como la tasa de aprendizaje, número de capas, tamaño del batch o número de neuronas por capa.
- **Incorporación de regularización** (como L2 o dropout) para mejorar la generalización.
- Uso de **funciones de activación más sofisticadas** como Softmax en la salida para clasificación multiclase, o incluso ReLU variantes como LeakyReLU.
- Aplicación de técnicas de **paralelización o uso de librerías optimizadas** (como Eigen o cuBLAS) para mejorar el rendimiento computacional.

En conclusión, el sistema Clasificador 3x3 cumple su objetivo pedagógico y funcional, siendo una base sólida sobre la cual construir experimentos más complejos en aprendizaje profundo con C++.

---

8. Trabajo Futuro

- Implementar visualización gráfica de los patrones.
- Extender el clasificador a patrones más complejos (matrices 5x5, imágenes, etc).
- Comparar el rendimiento con frameworks como TensorFlow o PyTorch en problemas similares.