UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA — UTEC

Facultad de Ingeniería de la Información

Proyecto Final 2025-1

AI Neural Network

Curso: CS2013 — Programación III

Informe Final

Grupo: group_3_custom_name

Integrantes:

- Gracia Mendoza – 202410390 (Informe completo)

Lima, Perú — 2025

Índice

1. Datos Generales

- **Tema:** Redes Neuronales Artificiales aplicadas en C++
- **Proyecto:** Implementación de una red neuronal multicapa desde cero.
- **Curso:** CS2013 Programación III
- **Ciclo:** 2025–1
- **Repositorio:** `https://github.com/EJEMPLO/proyecto-final`

—

2. Requisitos e Instalación

Compilador

- GCC 11+ o Clang 12+
- Soporte para C++17

Dependencias

Este proyecto **no requiere CMake ni Eigen**. Toda la arquitectura está codificada manualmente con operaciones matriciales propias.

Compilación con g++

```
# Demo XOR
g++ src/main.cpp src/Network.cpp src/layers/Dense.cpp \
    -o neural_net_demo -I src -std=c++17

# Demo de dataset sint tico
g++ src/main_large.cpp src/Network.cpp src/layers/Dense.cpp \
    -o neural_net_large_demo -I src -std=c++17
```

—

3. Investigación Teórica

3.1. Historia y evolución

Las redes neuronales surgieron en los años 50, evolucionaron con el perceptrón, decayeron en los 70 y resurgieron con el *backpropagation* en los 80. Hoy impulsan el aprendizaje profundo moderno.

3.2. Principales arquitecturas

- **MLP**: redes densas de múltiples capas.
- **CNN**: especializadas en imágenes.
- **RNN / LSTM**: secuencias y series de tiempo.

3.3. Backpropagation y optimización

$$Y = XW + B$$

$$L = \frac{1}{n}\sum(y - \hat{y})^2$$

$$\theta_{\text{nuevo}} = \theta_{\text{viejo}} - \alpha\frac{\partial L}{\partial \theta}$$

La regla de la cadena permite propagar el error desde la salida hacia las capas anteriores.

—

4. Diseño e Implementación

4.1. Arquitectura del proyecto

```
src/
 Matrix.h
 Network.h / Network.cpp
 main.cpp
 main_large.cpp
 layers/
    Layer.h
    Dense.h / Dense.cpp
    Activation.h
 losses/
     MSE.h
```

4.2. Clases clave

Clase Matrix

Respaldada por operaciones propias: suma, multiplicación, transposición, Hadamard, aleatorización.

Capa Dense

$$Y = XW + B$$

Activaciones

$$\tanh(x), \quad \sigma(x)$$

Función de pérdida MSE

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{n}(\hat{y} - y)$$

4.3. Manual de uso

```
./neural_net_demo
./neural_net_large_demo
```

4.4. Casos de prueba

- Test unitario de Matrix.
- Test de Dense.
- Test de activación.
- Test de convergencia XOR.

—

5. Ejecución

5.1. Demo XOR

Entrenamiento con arquitectura:

$$2 \to 3 \to 1$$

Resultados mostrados por consola.

5.2. Demo sintético (1000 muestras)

Pasos:

1. Ejecutar `neural_net_large_demo`.
2. Se generan features reales aleatorios.
3. Se calcula label binaria con funciones no lineales.
4. La red entrena y muestra la pérdida por época.

—

6. Análisis del Rendimiento

6.1. Métricas obtenidas

XOR

- Épocas: 10000
- LR: 0.1
- Error final: $< 0{,}01$

Dataset sintético

- Muestras: 1000
- Arquitectura: $10 \rightarrow 50 \rightarrow 30 \rightarrow 10 \rightarrow 1$
- Épocas: 1000
- Tiempo promedio: 1.7 segundos
- Estabilidad: sin oscilaciones fuertes

6.2. Ventajas y desventajas

- + Código ligero, entendible y modular.
- + No depende de frameworks externos.
- – No usa paralelización.
- – Multiplicación de matrices no optimizada.

6.3. Mejoras futuras

- Uso de BLAS (OpenBLAS / MKL).
- Mini–batch SGD.
- OpenMP para paralelización.

—

7. Conclusiones

- Se implementó una red neuronal desde cero en C++.
- Se validó correctamente con XOR y un dataset sintético.
- Se comprendió el funcionamiento interno de backpropagation.
- Se identificaron mejoras reales para escalar a producción.

—

8. Bibliografía

- I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016.
- M. Nielsen, *Neural Networks and Deep Learning*, 2015.
- C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
- Y. LeCun et al., "Deep Learning", *Nature*, 2015.

—

9. Licencia