

Red Neuronal Artificial en C++

Documentación Técnica del Proyecto

Grupo: group_3_custom_name
Curso: CS2013 Programación III
Año: 2025-1

Índice

1. Descripción del Proyecto	3
2. Características Principales	3
3. Estructura del Proyecto	3
4. Requisitos del Sistema	4
5. Instalación y Compilación	4
6. Fundamentos Teóricos	4
6.1. Arquitectura Multicapa	4
6.2. Propagación Hacia Adelante	4
6.3. Funciones de Activación	4
6.4. Función de Pérdida MSE	5
6.5. Backpropagation	5
7. Arquitectura de la Solución	5
7.1. Clase Matrix	5
7.2. Interfaz Layer	5
7.3. Capa Densa	6
7.4. Activaciones	6
7.5. Network	6
8. Función de Pérdida: MSE	6
9. Uso y Ejemplos	7
9.1. Ejemplo XOR	7
9.2. Red Personalizada	7
9.3. Demo de Dataset Sintético Grande	7
10. Pruebas Unitarias	7

11. Análisis de Rendimiento	8
11.1. XOR	8
11.2. Dataset Sintético Grande	8
12. Mejoras Futuras	8
13. Contribuciones	8
14. Licencia	8
15. Referencias	9

1. Descripción del Proyecto

Este proyecto implementa una **red neuronal artificial multicapa** desde cero en C++, sin utilizar bibliotecas de aprendizaje automático externas. Incluye una clase de matrices personalizada, capas densas, activaciones, función de pérdida MSE y un algoritmo completo de entrenamiento basado en **retropropagación (backpropagation)**.

El enfoque es completamente educativo y busca mostrar los fundamentos internos de cómo aprende una red neuronal.

2. Características Principales

- Implementación 100 % desde cero.
- Clase `Matrix` personalizada.
- Capas densas (`Dense`) y de activación (`Tanh`, `Sigmoid`).
- Función de pérdida MSE.
- Backpropagation implementado manualmente.
- Tests unitarios completos.
- Demo XOR + demo de dataset sintético grande.

3. Estructura del Proyecto

```
proyecto-final-2025-2-novo/
src/
    Matrix.h
    Network.h
    Network.cpp
    main.cpp
    main_large.cpp
    layers/
        Layer.h
        Dense.h
        Dense.cpp
        Activation.h
    losses/
        MSE.h
tests/
    test_matrix.cpp
    test_dense.cpp
    test_activation.cpp
    test_xor.cpp
DOCUMENTACION.md
```

4. Requisitos del Sistema

- Compilador con soporte C++17 (GCC 11+, Clang 12+, MSVC 2019+).

5. Instalación y Compilación

Compilación Directa con g++ (Recomendada)

```
1 # Demo XOR
2 g++ src/main.cpp src/Network.cpp src/layers/Dense.cpp \
3     -o neural_net_demo -I src -std=c++17
4
5 # Demo de dataset sint tico grande
6 g++ src/main_large.cpp src/Network.cpp src/layers/Dense.cpp \
7     -o neural_net_large_demo -I src -std=c++17
```

Ambos programas generan resultados por consola.

6. Fundamentos Teóricos

6.1. Arquitectura Multicapa

Una red neuronal multicapa típica incluye:

- Capa de entrada.
- Una o varias capas ocultas.
- Capa de salida.

6.2. Propagación Hacia Adelante

$$Y = XW + B$$

$$A = f(Y)$$

6.3. Funciones de Activación

Tangente Hiperbólica

$$\tanh'(x) = 1 - \tanh^2(x)$$

Sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

6.4. Función de Pérdida MSE

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

6.5. Backpropagation

Basado en la regla de la cadena:

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

7. Arquitectura de la Solución

7.1. Clase Matrix

Soporta:

- Suma y resta.
- Multiplicación matricial.
- Multiplicación por escalar.
- Producto Hadamard.
- Transposición.

```
1 class Matrix {
2 public:
3     int rows, cols;
4     std::vector<std::vector<double>> data;
5
6     Matrix(int r, int c);
7     void setRandom();
8     Matrix transpose() const;
9     static Matrix multiply(const Matrix& A, const Matrix& B);
10    Matrix hadamard(const Matrix& other) const;
11};
```

7.2. Interfaz Layer

```
1 class Layer {
2 public:
3     virtual Matrix forward(const Matrix& input) = 0;
4     virtual Matrix backward(const Matrix& dOut, double lr) = 0;
5};
```

7.3. Capa Densa

$$Y = XW + B$$

```
1 class Dense : public Layer {
2 public:
3     Matrix weights, bias, input;
4
5     Dense(int input_size, int output_size);
6     Matrix forward(const Matrix& in) override;
7     Matrix backward(const Matrix& dOut, double lr) override;
8 };
```

7.4. Activaciones

```
1 class Activation : public Layer {
2 private:
3     std::function<double(double)> act, act_prime;
4     Matrix input;
5 public:
6     Activation(auto a, auto a_p);
7     Matrix forward(const Matrix& in) override;
8     Matrix backward(const Matrix& dOut, double lr) override;
9 };
```

7.5. Network

```
1 class Network {
2 private:
3     std::vector<Layer*> layers;
4 public:
5     void add(Layer* layer);
6     Matrix predict(const Matrix& input);
7     void train(const Matrix& X, const Matrix& Y,
8                 int epochs, double lr);
9 };
```

8. Función de Pérdida: MSE

```
1 class MSE {
2 public:
3     static double loss(const Matrix& y_true,
4                         const Matrix& y_pred);
5
6     static Matrix prime(const Matrix& y_true,
7                         const Matrix& y_pred);
8 };
```

9. Uso y Ejemplos

9.1. Ejemplo XOR

```
1 Matrix X(4,2);
2 X.data = {{0,0},{0,1},{1,0},{1,1}};
3
4 Matrix Y(4,1);
5 Y.data = {{0},{1},{1},{0}};
6
7 Network net;
8 net.add(new Dense(2,3));
9 net.add(new Tanh());
10 net.add(new Dense(3,1));
11 net.add(new Tanh());
12
13 net.train(X, Y, 10000, 0.1);
14 Matrix pred = net.predict(X);
15 pred.print();
```

9.2. Red Personalizada

```
1 Network net;
2 net.add(new Dense(4,8));
3 net.add(new Tanh());
4 net.add(new Dense(8,4));
5 net.add(new Sigmoid());
6 net.add(new Dense(4,1));
7 net.add(new Sigmoid());
```

9.3. Demo de Dataset Sintético Grande

Dataset de 1000 muestras, 10 features:

- Arquitectura: $10 \rightarrow 50 \rightarrow 30 \rightarrow 10 \rightarrow 1$
- Épocas: 1000
- LR: 0.01

10. Pruebas Unitarias

Ejemplo de test de Matrix

```
1 void test_matrix_creation() {
2     Matrix m(3,4);
3     assert(m.rows == 3 && m.cols == 4);
4 }
```

Test de integración: XOR

```
1 void test_xor() {  
2     // Se entrena la red para aprender XOR  
3 }
```

11. Análisis de Rendimiento

11.1. XOR

- Arquitectura: $2 \rightarrow 3 \rightarrow 1$
- Épocas: 10000
- Error final: ± 0.01

11.2. Dataset Sintético Grande

- Arquitectura profunda
- 1000 muestras
- Comportamiento estable del entrenamiento

12. Mejoras Futuras

- Implementar ReLU y Softmax
- Añadir Adam y RMSprop
- Añadir Dropout
- Agregar capas convolucionales

13. Contribuciones

1. Hacer fork del repositorio
2. Crear rama
3. Realizar commits claros
4. Crear Pull Request

14. Licencia

Proyecto bajo licencia MIT.

15. Referencias

- Goodfellow, Bengio y Courville. *Deep Learning*.
- Nielsen. *Neural Networks and Deep Learning*.
- Bishop. *Pattern Recognition and Machine Learning*.
- LeCun, Bengio, Hinton. *Deep Learning*.