CENTRO UNIVERSITÁRIO INTERNACIONAL UNINTER
ESCOLA SUPERIOR POLITÉCNICA
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA DE INTELIGÊNCIA ARTIFICIAL APLICADA

ATIVIDADE PRÁTICA – TREINAMENTO DO NEURÔNIO DE ROSENBLATT

CIRO MENESES – RU: 2732171
PROF. LUCIANO FRONTINO DE MEDEIROS

ITAPIPOCA – CEARÁ
2021

1 - Objetivo

Este trabalho descreve o processo de desenvolvimento de um algoritmo em C++, que vai construir e realizar o treinamento supervisionado de um neurônio de Rosenblatt e servirá como avaliação prática da disciplina “Inteligência Artificial Aplicada”.

2 - Detalhes

Decidi desenvolver e treinar o neurônio utilizando a linguagem C++ porque é uma das linguagem mais utilizada nas diversas aplicações que requerem desempenho e também por ser orientada a objetos, o que facilita na codificação do neurônio.

Seguindo a risco as fórmulas do livro “Inteligência Artificial Aplicada – uma abordagem introdutória” do autor Luciano Frontino de Medeiros, consegui modelar a classe do neurônio de rosenblatt, bem como implementar o algoritmo de treinamento LMS descrito no livro. Descrevendo de uma forma simplista, criei uma classe para representar o Neurônio, logo após isso eu defini a quantidade de neurônios amostra e preenchi suas entradas com os números de RUs aleatórios, após isso, é iniciado o treinamento que automaticamente calibra os pesos e consequentemente delimitando a separação linear do conjunto de números que formam as entradas dos neurônios.

Por ser um treinamento supervisionado, houve a necessidade de gerar uma classificação ideal, onde o algoritmo une as sete entradas do neurônio e compara com o RU do aluno, no meu caso, com o número 2732171 e gerando a classificação ideal.

3 – Implementação em C++

O primeiro passo foi modelar a classe do neurônio para depois utiliza-lo no treinamento. A classe foi modelada conforme descrita abaixo com o nome Neuronio.h, onde foi definido que o neurônio teria sete(7) entradas, taxa de aprendizagem de 0.00001, vai ter os atributos: entradas, pesos e deltas como sendo arrays de tamanho 7, saída, classificação, classificação ideal, erro local e bias.

Apesar dos atributos pesos e bias pertencerem globalmente, a inclusão deles na classe facilitou a implementação do algoritmo.

```
//arquivo Neuronio.h
#pragma once

#include <iostream>
#define QTD_DE_ENTRADAS 7 //Número de entradas do neuronio
#define RU_ALUNO 2732171.0 //Limiar que divide os valores classificados
#define TAXA_APRENDIZAGEM 0.00001  //Velocidade da aprendizagem dos neurônios

using namespace std;

class Neuronio
```

```
{
public:

    double entradas[QTD_DE_ENTRADAS]; //Entrada de dados
    double pesos[QTD_DE_ENTRADAS]; //Pesos que serão calibrados no treinamento
    double deltas[QTD_DE_ENTRADAS]; //Deltas para os calculos
    double saida;
    double classificacao; //Classificação que será dada pelo treinamento
    double classificacaoIdeal; //Classificação precisa criada para auxiliar no
treinamento assistido
    double bias; //BIAS neh!
    double erroLocal;


    void imprime(); //Imprime os valores atuais do neurônio
    void classificaIdeal(); //Une as entradas e gera a classificação corretamente sem
necessitar do processamento dos neuronios. Serve apenas para treinamento assistido
    void classifica(); //Função ativadora que classifica o calculo da saída intermediaria
em +1 ou -1
    void calculaSaida(); //Realiza o calculo para gerar a saída intermediaria
    void setEntradas(double e[]); //Seta as entradas do Neurônio
    void setPesos(double p[]); //Seta os pesos do neurônio
    void calculaErroLocal(); //Calcula o erro local
    void calculaDeltas(); //Calcula os Deltas
    Neuronio();
    ~Neuronio();
};
```

Logo após modelar comecei a implementar os métodos da classe, as entradas e saídas de dados e os métodos que implementam as fórmulas matemáticas do neurônio, como o cálculo da saída intermediária, método ativador(classificação em +1 ou -1)calculo do erro local e calculo dos deltas, conforme descrito no arquivo Neuronio.cpp logo abaixo.

```
//arquivo Neuronio.cpp
#include "Neuronio.h"



Neuronio::Neuronio()
{   //inicializa BIAS
    this->bias = -1.0;

    //inicializa entradas e pesos
    for (int i = 0; i < QTD_DE_ENTRADAS; i++) {
        this->entradas[i] = 1.0;
        this->pesos[i] = 1.0;
        this->deltas[i] = 0.0;

    }
}


Neuronio::~Neuronio()
{
}
```

```
void Neuronio::imprime() {


        //Imprime Entradas
        cout << " E: ";
        for (int i = 0; i < QTD_DE_ENTRADAS; i++)
            cout << (int)this->entradas[i];
        //Imprime Pesos
        cout << " P:";
        for (int i = 0; i < QTD_DE_ENTRADAS; i++)
            cout << " " << this->pesos[i];


        cout << " C: " << (int)this->classificacao;
        cout << " CI: " << (int)this->classificacaoIdeal;
        cout << " S: " << this->saida;
        cout << " Erro: " << this->erroLocal;
        //Imprime Deltas
        cout << " D:";
        for (int i = 0; i < QTD_DE_ENTRADAS; i++)
            cout << " " << this->deltas[i];


        cout << endl;

}

//junta os números do vetor em um numero para encontrar a classificação ideal
//sem intermedio do neuronio, para fins de treinamento assistido
void Neuronio::classificaIdeal() {
    double valor = 0;

    //Pega as entradas e une elas em um double (transforma um array de double em um
double)
    for(int i = 0; i < QTD_DE_ENTRADAS; i++)
        valor = valor * 10.0 + entradas[i];

    //Compara o numero com o RU para gerar a classificação correta. Serve apenas para a
aprendizagem supervisionada.
    if (valor >= RU_ALUNO)
        this->classificacaoIdeal = 1;
    else
        this->classificacaoIdeal = -1;
}
//Seta as entradas recebendo um vetor como argumento
void Neuronio::setEntradas(double e[]) {
    for (int i = 0; i < QTD_DE_ENTRADAS; i++)
        this->entradas[i] = e[i];
}

//Seta os pedo recebendo um vetor como argumento
void Neuronio::setPesos(double p[]) {
    for (int i = 0; i < QTD_DE_ENTRADAS; i++)
        this->pesos[i] = p[i];
}

void Neuronio::calculaSaida() {
    this->saida = 0;
```

```
    for (int i = 0; i < QTD_DE_ENTRADAS; i++)
        this->saida = this->saida + (this->entradas[i] * this->pesos[i]);

    this->saida = this->saida + this->bias;
}

void Neuronio::classifica() {
    if (this->saida >= 0)
        this->classificacao = 1;
    else
        this->classificacao = -1;
}

void Neuronio::calculaErroLocal() {
    this->erroLocal = this->classificacaoIdeal - this->classificacao;
}



void Neuronio::calculaDeltas() {

    for (int i = 0; i < QTD_DE_ENTRADAS; i++) {
        this->deltas[i] = 0;
        this->deltas[i] =  this->entradas[i]  * this->erroLocal * TAXA_APRENDIZAGEM;
//taxa_aprendizagem * erro_local_n * entrada_n
    }


}
```

Após modelar e implementar a classe, o ideal seria modelar e implementar outra classe chamada RNA, porém, optei por implementar o restante do código no arquivo main.cpp, sem a utilização de outra classe, onde esse arquivo ficou responsável pela instanciação das amostras geradas por um array de objetos da classe Neuronio.

Além disso, foram criadas constantes para a quantidade de neurônios e de épocas de treinamento, bem como os métodos para calcular a quantidade de acertos e para calculo e reajuste dos novos pesos do neurônio. O arquivo main.cpp está descrito abaixo.

```
//Arquivo Main.cpp
#include "Neuronio.h"
#include <vector>
#include <iomanip>
#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <windows.h>


#define QTD_NEURONIOS 1000
#define QTD_EPOCAS 200000

//Atualiza os novos pesos de acordo com a formula
```

```
void calculaNovosPesos(Neuronio neu[]);
//Calcula quantos acertos o neuronio conseguiu usando o treinamento assistido e imprime a
qtd de erros e acertos
void calculaAcertos(Neuronio neu[]);

// QTD_DE_ENTRADAS 2. Ajustar aqui a QTD de entradas
double pesos[QTD_DE_ENTRADAS] = { 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 };
double entradas[QTD_DE_ENTRADAS] = { 5.0, 9.0, 9.0, 9.0, 9.0, 9.0, 9.0 };
double entradaAlta[QTD_DE_ENTRADAS] = { 2.0, 7.0, 3.0, 2.0, 1.0, 7.0, 2.0 };
double entradaBaixa[QTD_DE_ENTRADAS] = { 2.0, 7.0, 3.0, 2.0, 1.0, 7.0, 0.0 };
Neuronio n[QTD_NEURONIOS];




int main() {
    srand(time(NULL));//Gera semente aleatória

    //Seta a precisão do double
    std::cout << fixed << setprecision(5);

    //inicializa alguns neurônios para testes
    for (int i = 0; i < QTD_NEURONIOS; i++)
        n[i] = Neuronio();

    //preenche as entradas com numeros aleatórios
    for (int i = 0; i < QTD_NEURONIOS; i++) {
        //cout << "Entrada " << i << ": "; //imprime Entradas para verificação
        for (int x = 0; x < QTD_DE_ENTRADAS; x++) {
            entradas[x] = rand() % 9 + 1;
            //cout << entradas[x]; //imprime Entradas para verificação
        }

        //cout << endl; //imprime Entradas para verificação
        n[i].setEntradas(entradas); //Injeta o array de entradas geradas no array de
entradas do neuronio
    }
    n[0].setEntradas(entradaBaixa); //Injeta o array de 1 numero abaixo do RU
    n[1].setEntradas(entradaAlta); //Injeta o array de 1 numero acima do RU

    //treinamento dos neuronios
    for (int x = 0; x < QTD_EPOCAS; x++)
    {

        for (int i = 0; i < QTD_NEURONIOS; i++) {
            //insere pesos padrões
            n[i].setPesos(pesos);
            //n[i].setEntradas(entradas);
            n[i].calculaSaida(); //calcula saida
            n[i].classificaIdeal(); //calcula a classificação ideal
            n[i].classifica(); //Calcula a classificação usando as fórmulas matemáticas
            n[i].calculaErroLocal(); //Calcula o Erro local
            n[i].calculaDeltas(); //Calcula os Dentas para poder atualizar os pesos
            //n[i].imprime(); //Imprime todo o conjunto de neuronios
        }
        //Atualiza os pesos
        calculaNovosPesos(n);
        n[0].imprime(); //imprime apenas a amostra do conjunto de neuronios para
verificar as modificação nos pesos
```

```
        n[1].imprime(); //imprime apenas a amostra do conjunto de neuronios para
verificar as modificação nos pesos
        n[2].imprime(); //imprime apenas a amostra do conjunto de neuronios para
verificar as modificação nos pesos
        n[3].imprime(); //imprime apenas a amostra do conjunto de neuronios para
verificar as modificação nos pesos
        cout << "Epoca " << x << ": " << endl;
        calculaAcertos(n); //Calcula quantos acertos obteve

    }
    return 0;
}

//Atualiza os novos pesos de acordo com a formula
void calculaNovosPesos(Neuronio neu[]) {
    for (int n = 0; n < QTD_NEURONIOS; n++)
        for (int i = 0; i < QTD_DE_ENTRADAS; i++)
        {
            pesos[i] = pesos[i] + neu[n].deltas[i];
        }
}

//Calcula quantos acertos o neuronio conseguiu usando o treinamento assistido e imprime a
qtd de erros e acertos
void calculaAcertos(Neuronio neu[]) {
    double erros = 0, acertos = 0;

    for (int n = 0; n < QTD_NEURONIOS; n++)
    {
        if (neu[n].classificacao == neu[n].classificacaoIdeal)
            acertos += 1;
        else
            erros += 1;
    }

    cout << "Erros: " << erros << endl << "Acertos: " << acertos << endl << endl;

    if (erros == 0)
        exit(1);


}
```

4 – Treinamento supervisionado do Neurônio de Rosenblatt

Configurando os parâmetros do programa, defini que existiriam mil amostras, todas com valores aleatórios, exceto uma amostra semelhante ao meu RU e outra amostra com o valor um pouco abaixo do meu RU. O bias foi inicializado com o valor -1.0, os pesos foram inicializados todos com o valor 1.0.

O funcionamento do treinamento consiste em recalcular os pesos até encontrar os valores que permitem uma precisão alta na classificação dos números. Para chegar nessa classificação, utilizei o treinamento supervisionado, onde se consegue a classificação ideal e a partir dela, pode-se comparar com a classificação gerada pelas fórmulas matemáticas do neurônio, caso haja um erro na classificação, é calculado os deltas, um delta para cada entrada do neurônio.

Obtido os deltas, pode-se então avançar para o calculo dos novos pesos e ao final desse processo, finaliza-se uma época de treinamento, podendo partir para a próxima época de treinamento, onde serão novamente geradas as classificações dos neurônios e realizados os cálculos de erro, deltas e novos pesos novamente.

Sempre ao final de uma época, optei por imprimir apenas quatro amostras escolhidas, pois ao imprimir todas as amostras, o tempo de execução do treinamento aumentava drasticamente, inviabilizando o treinamento. Segue abaixo a saída da primeira linha do programa realizando o treinamento.

E: Números que compõem a entrada do neurônio
C: Classicifação gerada pelo Neurônio
CI: Classicifação Ideal Gerada sem o Neurônio
S: Saída intermediária, antes de ser ativada em +1 e -1
Erro: valor do erro para o determinado neurônio
D: Array de deltas que serão utilizados para calcular os novos pesos

E: 2732170 P : 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 C : 1 CI : -1 S : 21.00000 Erro : -2.00000 D : -0.00004 - 0.00014 - 0.00006 - 0.00004 - 0.00002 - 0.00014 - 0.00000
E : 2732171 P : 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 C : 1 CI : 1 S : 22.00000 Erro : 0.00000 D : 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
E : 5445125 P : 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 C : 1 CI : 1 S : 25.00000 Erro : 0.00000 D : 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
E : 4499239 P : 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 C : 1 CI : 1 S : 39.00000 Erro : 0.00000 D : 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
Epoca 0 :
Erros : 192.00000
Acertos : 808.00000

Pode-se perceber que existem muitos erros de classificação, por volta dos 20%, mas conforme o treinamento vai avançando, os pesos vão sendo balanceados e conseguem um nível de precisão muito elevado. Segue abaixo a penúltima linha do treinamento.

E: 2732170 P: 0.41102 0.02632 0.00344 -0.00048 -0.00044 -0.00184 0.00002 C: 1 CI: -1 S: 0.00232 Erro: -2.00000 D: -0.00004 -0.00014 -0.00006 -0.00004 -0.00002 -0.00014 -0.00000
E: 2732171 P: 0.41102 0.02632 0.00344 -0.00048 -0.00044 -0.00184 0.00002 C: 1 CI: 1 S: 0.00234 Erro: 0.00000 D: 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
E: 5445125 P: 0.41102 0.02632 0.00344 -0.00048 -0.00044 -0.00184 0.00002 C: 1 CI: 1 S: 1.16772 Erro: 0.00000 D: 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
E: 4499239 P: 0.41102 0.02632 0.00344 -0.00048 -0.00044 -0.00184 0.00002 C: 1 CI: 1 S: 0.76978 Erro: 0.00000 D: 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
Epoca 81667:
Erros: 1.00000
Acertos: 999.00000

Na penúltima linha, percebe-se que os pesos já foram bastante modificados, porém o neurônio ainda possui 0.1% de erro, mas isso é corrigido na época 81667, na última linha do treinamento a seguir.

E: 2732170 P: 0.41098 0.02618 0.00338 -0.00052 -0.00046 -0.00198 0.00002 C: -1 CI: -1 S: -0.00000 Erro: 0.00000 D: 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
E: 2732171 P: 0.41098 0.02618 0.00338 -0.00052 -0.00046 -0.00198 0.00002 C: 1 CI: 1 S: 0.00002 Erro: 0.00000 D: 0.00000 0.00000

0.00000 0.00000 0.00000 0.00000 0.00000
E: 5445125 P: 0.41098 0.02618 0.00338 -0.00052 -0.00046 -0.00198 0.00002 C: 1 CI: 1 S: 1.16622 Erro: 0.00000 D: 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
E: 4499239 P: 0.41098 0.02618 0.00338 -0.00052 -0.00046 -0.00198 0.00002 C: 1 CI: 1 S: 0.76770 Erro: 0.00000 D: 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
Epoca 81668:
Erros: 0.00000
Acertos: 1000.00000

Finalizado o treinamento, ao final de 814665 épocas, o programa nos retornou os valores dos sete pesos calibrados “0.41098, 0.02618, 0.00338, -0.00052, -0.00046, -0.00198, 0.00002”. Vale ressaltar que ao obter 100% de precisão, o treinamento é encerrado, pois o objetivo do treinamento é calibrar os pesos a um nível onde se obtenha 100% de precisão na classificação dos números.

3 - Conclusão

O objetivo do trabalho foi concluído com sucesso, pois com a implementação do Neurônio de Rosenblatt em C++ e a utilização do programa compilado para realizar o treinamento supervisionado, foi possível calibrar os pesos de mil amostras, permitindo um alto nível de precisão na classificação dos números de RU de um aluno qualquer da UNINTER, onde esse número é composto por sete dígitos.

Observou-se também que o tempo de treinamento variou entre 1 e 50 segundos, pois dependendo dos RUs gerados, o programa pode demorar um pouco mais para calibrar os pesos.

TABELA DE INFORMAÇÕES SOBRE O TREINAMENTO

RU DO ALUNO	PESOS W DO 1 AO 7	BIAS
2	0.41098	-1
7	0.02618	
3	0.00338	
2	-0.00052	
1	-0.00046	
7	-0.00198	
1	0.00002	