INSTITUTO FEDERAL DO CEARÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

INTELIGÊNCIA COMPUTACIONAL APLICADA

Carlos Estevão Bastos Sousa

RELATÓRIO DE APLICAÇÃO DE ALGORITMOS GENÉTICOS EM REDES NEURAIS ARTIFICIAIS

Docente: Ajalmar R. Rocha Neto

FORTALEZA – CE, 2018

RESUMO

O presente trabalho tem como objetivo explanar sobre a aplicação de GA's (*Genetic Algorithms*) em ANN's (*Artificial Neural Networks*), o mesmo é dividido em dois problemas: A atualização de pesos em um SLP (*Single-Layer Perceptron*) aplicado a porta AND na qual inserimos entradas binárias (0 e 1) e uma segunda aplicação para o mesmo problema na qual inserirmos ruídos as entradas, e a poda em uma rede ELM (*Extreme Learning Machine*) aplicada a base de dados *Iris Data Set* e *Vertebral Column Data Set* (column_3C.dat).

Palavras-chave: Algoritmo Genético, Perceptron Simples, Máquina de Aprendizado Extremo, Atualização de pesos.

SUMÁRIO

1. Introdução 5
2. *SLP (Single-Layer Perceptron)* 6
3. *ELM (Extreme Learning Machine)* 6
4. *GA (Genetic algorithm)* 8
5. *GA-ELM (Genetic algorithm with Extreme Learning Machine)* 9
6. Aplicação prática 9
   - 6.1. *SLP* (*Single-Layer Perceptron*) 9
   - 6.2. SLP (*Single Layer Perceptron*) com atualização de Pesos através de GA (*Genetic Algorithm*) 10
   - 6.3. ELM (*Extreme Learning Machine*) 10
   - 6.4. ELM (*Extreme Learning Machine*) com podagem de Neurônios através de GA (*Genetic Algorithm*) 11
7. Simulações Computacionais 11
8. Conclusões 12
9. Referências 13
10. Apêndices 14
    - 10.1. SLP (Código em Matlab) 14
    - 10.2. SLP com GA (Código em Matlab) 15
    - 10.3. ELM (Código em Matlab) 17
    - 10.4. GA-ELM (Código em Matlab) 18

1. INTRODUÇÃO

O presente trabalho visa explanar a aplicação de um GA (*Genetic Algorithm*) na elaboração de dois algoritmos, um SLP (*Single-Layer Perceptron*) para resolução do problema da porta AND e uma ELM (*Extreme Learning Machine*) aplicada a base de dados *Iris Data Set e Vertebral Column Data Set*, no primeiro utilizamos o GA para a escolha de pesos, e no segundo, para a poda dos neurônios da rede.

Segundo HAYKIN (2008) uma rede neural artificial pode ser definida como um processador maciçamente paralelo e distribuído formado de simples unidades capazes de armazenar conhecimento baseado em experiência.

Para COPPIN (2004) um algoritmo genético pode ser usado para determinar a conectividade de uma rede. Deste modo, o número de neurônios e as conexões entre estes neurônios podem evoluir, para produzir uma arquitetura ótima.

Redes neurais com uma única camada oculta e alimentação direta são hoje em dia muito aplicadas a problemas de classificação de padrões e aproximação de funções. O algoritmo mais comumente utilizado para ajustar os pesos de uma rede com essa arquitetura é o de retropropagação do erro *error backpropagation*, que usa técnicas de otimização com base no gradiente descendente. Essa técnica apresenta sensibilidade a ocorrência de mínimos locais e possui convergência lenta (ALENCAR e NETO, 2014).

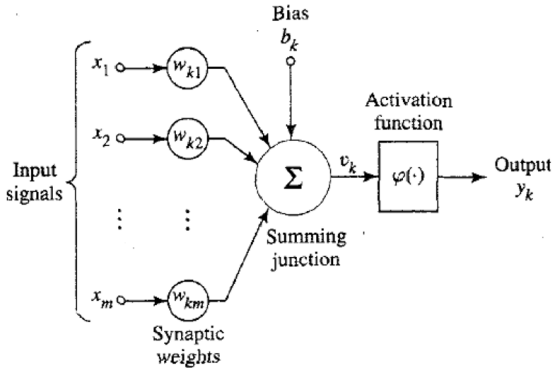
A *Iris Data Set* é uma base de dados para classificação bem clássica para quem estuda ou trabalha com Inteligência Artificial a mesma consiste em 150 entradas, com 4 atributos (*sepal length in cm, sepal width in cm, petal length in cm and petal width in cm*) e 3 classes (*Iris Setosa, Iris Versicolour and Iris Virginica*).

A base de dados *Vertebral Column Data Set* consiste na classificação de 3 classes (*normal, disk hernia or spondilolysthesis*) ou 2 classes (*normal or abnormal*), conforme podemos perceber a mesma é relacionadas a patologias na coluna vertebral, ela possui 310 entradas e 6 atributos, neste trabalho utilizamos a base com 3 classes.

2. SLP (SINGLE-LAYER PERCEPTRON)

Um único perceptron pode ser usado para aprender uma tarefa de classificação, na qual ele recebe uma entrada e a classifica em uma de duas categorias: 1 ou 0. Podemos considerá-las como representando *verdadeiro* ou *falso* e, neste caso, o perceptron pode aprender a representar um operador booleano, tal como E ou OU. (COPPIN, 2004)

Figura 1: Single-Layer Perceptron

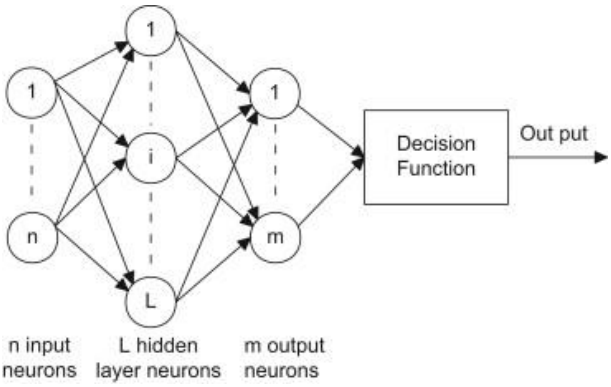


Fonte: Haykin, 1999

3. ELM (EXTREME LEARNING MACHINE)

ELM é uma rede neural artificial com apenas uma camada oculta, a mesma segue a mesma metodologia de uma RNA, porém não faz uso de gradiente descendente, desta forma, a mesma pode se sobressair sobre as demais redes pois não possui convergência lenta nem convergência para mínimos locais. Segundo HUANG (2006), o treinamento dessa rede pode ser milhares de vezes mais rápido do que o treinamento via backpropagation.

Figura 2: Representação esquemática do AG-ELM



Fonte: http://dovgalecs.com/blog/extreme-learning-machine-matlab-mex-implementation/

1 a N representam as entradas da rede, os pesos da camada de entrada ficam entre a entrada e a camada oculta (L), os pesos de L são representados por uma matriz β. Abaixo segue a organização da ELM.

$$X = [x1, \dots xn]$$

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{md} \\ b_1 & \cdots & b_d \end{bmatrix}$$

$$\beta = \begin{bmatrix} b_{11} & \cdots & b_{1s} \\ \vdots & \ddots & \vdots \\ b_{d1} & \cdots & b_{ds} \end{bmatrix}$$

$$Y = [y_1, \dots, y_s]$$

O bias da camada oculta é inserido na última linha dos pesos, na camada de saída não o utilizamos, **m** representa o número de neurônios da camada de entrada, **d** a quantidade de neurônios da camada de saída e **s**, o número de saídas da rede.

A matriz de pesos é gerada de forma aleatória e não é alterada até o fim da execução, podemos afirmar que a ELM tem como objetivo encontrar a matriz de pesos **β,** de acordo com as saídas e com os pesos, para isso devemos encontrar a matriz **H** conforme segue abaixo.

$$H^i = \left[x_1^i, \dots, x_m^i\right] \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{md} \\ b_1 & \cdots & b_d \end{bmatrix} \rightarrow H = \begin{bmatrix} f(H^1) \\ f(H^2) \\ \vdots \\ f(H^N) \end{bmatrix}_{N\ x\ d}$$

f é a função de transferência da camada e **H** possui o resultado dos neurônios da camada oculta, obtendo esses valores agora basta adquirir os pesos da matriz β como segue abaixo.

$$H\,\beta = Y \rightarrow \beta = H^T Y$$

Com a obtenção de β teremos a rede treinada, podemos perceber que a mesma possui um treinamento bem rápido, a partir desse momento basta elaborar a fase de testes e dar continuidade a rede em questão.

4. GA (GENETIC ALGORITHM)

Algoritmos Genéticos são técnicas estocásticas de otimização, métodos inspirados na teoria da Evolução de Darwin, a ideia mais simples foi descrita por Holland em 1975, na qual uma cadeia de bits é conhecida como cromossomo (um indivíduo completo) e cada bit da mesma como um gene.

De acordo com VIANA (1998) a ideia básica consiste em que, de forma similar a teoria biológica dos sistemas naturais, os "melhores" indivíduos sobrevivem e geram descendentes com suas características hereditárias.

Figura 3: Algoritmo Genético básico

Algorithm 1 Basic GENETIC ALGORITHM

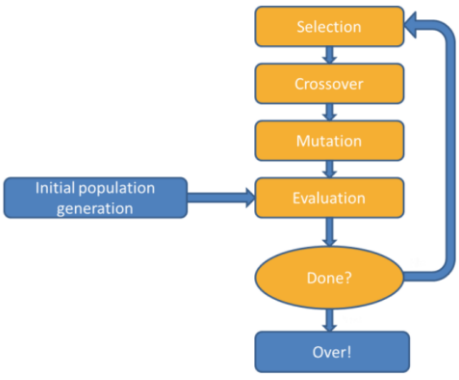
```
1: initialize population
2: repeat
3:    repeat
4:       crossover
5:       mutation
6:       phenotype mapping
7:       fitness computation
8:    until population complete
9:    selection of parental population
10: until termination condition
```

Fonte: KRAMER(2017), GENETIC ALGORITHM ESSENTIALS

De acordo com VIANA(1998) a ideia básica consiste em que, de forma similar a teoria biológica dos sistemas naturais, os "melhores" indivíduos sobrevivem e geram descendentes com suas características hereditárias.

A figura seguir representa o funcionamento de um AG

Figura 4: Estrutura básica de um Algoritmo Genético



Fonte: https://genetic.io/en/introduction-genetic-algorithms/

5. *GA-ELM (GENETIC ALGORITHM WITH EXTREME LEARNING MACHINE)*

Conforme citado anteriormente, as ELM's possuem convergência rápida e foge de mínimos locais, porém, a mesma pode ser uma alternativa ruim quando os recursos computacionais se tornam escassos. A ideia de implementar o GA surge com a intenção de reduzir a quantidade de neurônios da ELM e assim, torná-la mais adequada a ser trabalhada em ambientes onde a quantidade de processamento seja baixa. No tópico a seguir explanaremos a metodologia para a aplicação da meta-heurística citada na rede em questão.

6. APLICAÇÃO PRÁTICA

6.1. *SLP* (*Single-Layer Perceptron*)

A aplicação do SLP ocorreu em 2 etapas, a seguir explanamos sobre as mesmas:

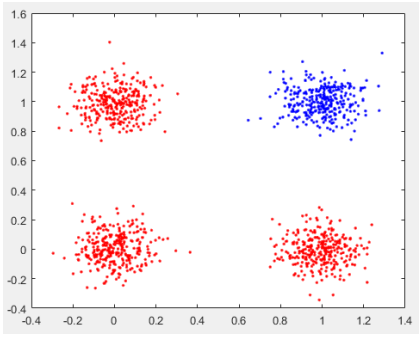
- Aplicação com instâncias binárias para o problema da porta AND, conforme segue na tabela abaixo:

Tabela 1- Resultados Obtidos SLP para análise em 100% de acertos

Entradas		Saídas
0	0	0
0	1	0
1	0	0
1	1	1

- Instâncias com ruídos de acordo com a imagem a seguir (figura 5):

Figura 5: Dados com ruídos



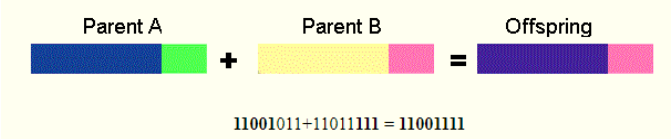
Fonte: Elaborada pelo autor

6.2. SLP (*Single Layer Perceptron*) com atualização de Pesos através de GA (*Genetic Algorithm*)

Seguindo o mesmo padrão de entradas do item anterior, iniciamos o algoritmo com uma população randômica (a mesma representa os pesos), implementamos mutações e cruzamentos na qual a probabilidade é dada nas variáveis iniciais.

Na mutação sorteamos um gene e substituímos seu valor por um número diferente do atual, no cruzamento, utilizamos uma espécie de roleta para sortear os indivíduos nos quais os que possuem maior fitness (maior possibilidade de ser a solução procurada) têm uma maior chance de serem sorteados, após esta etapa o filho desse cruzamento substituirá o indivíduo com menor fitness, fazendo assim, com que a população tenda a uma solução ótima.

Figura 6: Cruzamento em um ponto (utilizado no projeto)

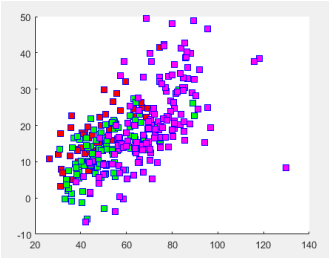


Fonte: http://www.obitko.com/tutorials/genetic-algorithms/portuguese/crossover-mutation.php

6.3. ELM (*Extreme Learning Machine*)

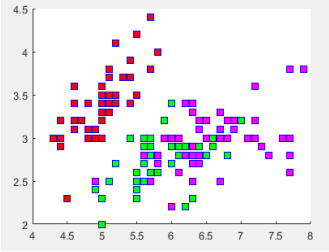
Conforme citado anteriormente, aplicamos a ELM nas bases de dados *Vertebral Column Data Set* e *Iris Data Set*, a seguir segue algumas imagens representando as mesmas.

Figura 7: *Vertebral Column Data Set* (atributos *pelvic_incidence numeric* e *pelvic_tilt numeric*)



Fonte: Elaborada pelo autor

Figura 8: *Iris Data Set* (atributos *sepal length* e *sepal width*)

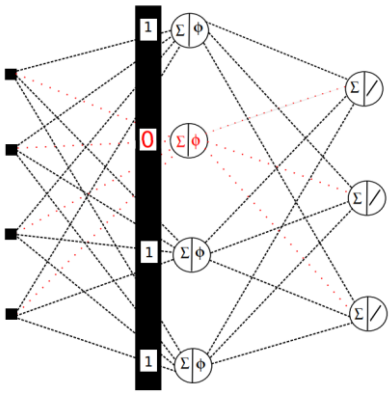


Fonte: Elaborada pelo autor

6.4. ELM (*Extreme Learning Machine*) com podagem de Neurônios através de GA (*Genetic Algorithm*)

Seguindo o mesmo padrão do SLP com atualização de pesos através GA, aplicamos mutações e cruzamentos, o primeiro manteve a mesma ideia aplicada anteriormente e o segundo, mantém a roleta ponderada para o sorteio dos indivíduos a serem cruzados, a alteração ficou na população, nesta aplicação a mesma não representa os pesos e sim os neurônios da rede ELM, assim, cada gene representa um deles, por exemplo, o indivíduo 1011 utiliza apenas os neurônios 1, 3 e 4 sendo o neurônio 2 (com o valor 0), não utilizado.

Figura 9- Representação esquemática do AG-ELM



Fonte: ALENCAR e NETO, 2014

7. SIMULAÇÕES COMPUTACIONAIS

7.1.SLP simples e SLP com atualização de pesos através de GA

Tabela 2- Resultados Obtidos SLP para análise em 100% de acertos

Porta AND	Tempo de Execução		População inicial	Gerações
	SLP	GA-SLP		
Sem ruídos	± 0.042	± 0.044	50	± 1
Com ruídos	± 0.229	± 14.351	50	± 98.0

7.2.ELM simples e ELM com poda através de GA

Tabela 3- Resultados Obtidos ELM

Bases	Tempo de Execução	Número de Neurônios	Acurácia
IRIS	± 0.274478	30	± 0.9867
VCD	± 0.283425	30	± 0.8387

Tabela 4- Resultados Obtidos ELM com GA

Bases	Tempo de Execução	Número de Neurônios	População inicial	Gerações	Acurácia
IRIS	± 2.982342	18	50	10	± 0.9833
VCD	± 1.573378	14	50	10	± 0.8871

8. CONCLUSÕES

Como podemos analisar no tópico anterior, o SLP na aplicação do problema da porta AND e sem ruídos aplicados as entradas tem uma melhor performance que a mesma aplicação com atualização de pesos através de GA, mas, isso ocorre apenas para um problema mais simples (como é o caso em questão). Ao deixar o problema um pouco mais complexo (inserção de ruídos nas entradas) em alguns casos nos aproximamos de 100% de acertos, porém, alguns erros não foram solucionados com sucesso, desta forma o algoritmo ficou com uma taxa de acerto média de 99,5%; por outro lado, através da aplicação de GA para atualização dos pesos conseguimos 100% de acertos um tempo superior aos algoritmos citados anteriormente mas com uma taxa de acerto conforme esperada (100%).

Para a ELM podemos perceber que na base de dados *Iris Data Set* conseguimos reduzir a quantidade de neurônios em 40%, a taxa de acerto foi reduzida, porém de forma quase que insignificante (0.0034 segundos). Ao analisar a base de dados *Vertebral Column Data Set*, podemos afirmar que houve uma redução em cerca de 53% da quantidade de neurônios e ocorreu um aumento médio de cerca de 1,28995 segundos em sua execução.

9. REFERÊNCIAS

ALENCAR, ASC. NETO, ARR. (2014). UMA ABORDAGEM DE PODA PARA MAQUINAS DE APRENDIZADO EXTREMO VIA ALGORITMOS GENÉTICOS, LTC.

AN INTRODUCTION TO GENETIC ALGORITHMS. Disponível em: <https://genetic.io/en/introductiongenetic-algorithms/>. Acesso em: 18 jun. 2018.

COPPIN, B. (2004). INTELIGÊNCIA ARTIFICIAL, LTC.

HAYKIN, S. (2001). REDES NEURAIS ARTIFICIAIS: PRINCÍPIOS E PRÁTICA, Bookman.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. EXTREME LEARNING MACHINE: THEORY AND APPLICATIONS. Neurocomputing, v. 70, n. 1, p. 489–501, 2006.

INTRODUÇÃO AOS ALGORITMOS GENÉTICOS - XI. CRUZAMENTO E MUTAÇÃO. OBITKO, Disponível em: <http://www.obitko.com/tutorials/genetic-algorithms/portuguese/crossover-mutation.php>. Acesso em: 20 nov. 2018.

VIANA, V. META-HEURÍSTICAS E PROGRAMAÇÃO PARALELA EM OTIMIZAÇÃO COMBINATÓRIA. 1. ed. Fortaleza-CE: EUFC, 1998. 250 p.

10. APÊNDICES

10.1. SLP (Código em Matlab)

```
entradas =  [0.1*randn(300,2);
                repmat([1 0], 300,1) + 0.1*randn(300,2);
                repmat([0 1], 300,1) + 0.1*randn(300,2);
                repmat([1 1], 300,1) + 0.1*randn(300,2)];
saida = [zeros(900,1);ones(300,1)]';
pesos = [0.1 * randn(1,3)]
TaxaApren = 0.1;

%% Insere uma última coluna com o valor -1 para multiplicar com o bias
for i = 1: size(entradas,1)
   entradas(:,3) = -1;
end

%% Execução do treinamento
PesosAtualizados = treinar(entradas, pesos, saida, TaxaApren);

% Execução do teste
teste(entradas, PesosAtualizados);

% Plotagem dos dados
plot(entradas(saida==0,1), entradas(saida==0,2),'r.', entradas(saida==1,1),
entradas(saida==1,2), 'b.');
toc
% Calcula o produto
function Saida = CalculaSaida(entradas, pesos)
    Saida = stepFunction(sum(entradas .* pesos));
end

%% Função de ativação
function step = stepFunction(soma)
    if (soma >= 1)
        step = 1;
    else
        step = 0;
    end
end

%% Treino
function pesosAtualizados = treinar(entradas, pesos, saida, TaxaApren)
    ErroTotal  = 1;
    iteracao = 1;
    PercentualAcerto = 0;
    while (PercentualAcerto < 98)
        ErroTotal = 0;
        disp("Execução "+ iteracao);
        for i = 1:size(saida, 2)
            SaidaCalculada = CalculaSaida(entradas(i,:), pesos);
            erro = abs(saida(:,i) - SaidaCalculada)
            ErroTotal = ErroTotal + erro;
            for j = 1:size(pesos, 2)
                pesos(:,j) = pesos(:,j) + (TaxaApren * entradas(i,j) * erro);
            end
            PercentualAcerto = ((size(entradas, 1) - ErroTotal ) * 100)/size(entradas,1)
        end
        iteracao = iteracao + 1;
    end
    pesosAtualizados = pesos;
end

%% Teste
function teste(entradas, PesosAtualizados)
    entradasOriginais = entradas;
    entradasOriginais(:,size(entradas,2)) = [];
    for i = 1 : size(entradas, 1)
        disp("Saída para ");
        disp(entradas(i,:));
        saida = (CalculaSaida(entradas(i,:), PesosAtualizados));
        disp(saida);
    end
```

```
end
```

10.2. SLP com GA (Código em Matlab)

```
entradas =  [0.1*randn(300,2);
                repmat([1 0], 300,1) + 0.1*randn(300,2);
                repmat([0 1], 300,1) + 0.1*randn(300,2);
                repmat([1 1], 300,1) + 0.1*randn(300,2)];
    saida = [zeros(900,1);ones(300,1)];

    populacao = 50;
    pesos = IniciaPopulacao(populacao, 3);
    TaxaApren = 0.0;

    percentualMutacao = 0.01;       % Probabilidade de mutação - 1%
    percentualCruzamento = 0.8;     % Probabilidade de cruzamento - 80%

%% Insere uma última coluna com o valor 1 para multiplicar com o bias
    [tam, ~] = size(entradas);
    entradas = [entradas ones(tam,1)];

%% Execução do treinamento
    PesosAtualizados = treinar(entradas, pesos, saida, populacao, percentualMutacao,
percentualCruzamento);

%% Execução do teste
    teste(entradas, PesosAtualizados);

%% Plotagem dos dados
    plot(entradas(saida==0,1), entradas(saida==0,2),'r.', entradas(saida==1,1),
entradas(saida==1,2), 'b.');
toc
%% Treinamento
    function pesosAtualizados = treinar(entradas, pesos, saida,tamanhoPopulacao,
percentualMutacao, percentualCruzamento)
        contaMutacoes = 0;
        ErroTotal = 1;
        iteracao = 1;
        while (ErroTotal ~= 0)
            disp("Geração "+ iteracao)
           for i = 1:size(saida, 2)
               ErroTotal = 0;
               for j = 1: size(pesos, 1)
                  SaidaCal(j,:) = entradas * pesos(j,:)';
               end
%% Recebimento das saídas calculadas através da função StepFunction
               SaidaCalculada = StepFunction(SaidaCal');

%% Cálculo do erro e condição de parada
                erro = abs(saida(:,i) - SaidaCalculada);
                [erros, posicao] = min(sum(erro));
                if (min(sum(erro)) == 0)
                   disp("Solução encontrada");
                   disp(posicao)
                   disp("Pesos");
                   disp(pesos(posicao,:));
                   break;
                end
               sum(erro)
%% verificar fitness
               for j = 1: size(erro, 2)
                 FFitness(:,j) = Fitness(erro(:,j), size(saida, 1));
% neste caso, quanto menor o fitness melhor é o indivíduo
                 end

%% Cruzamento
                probcruzamento = rand;
             if (probcruzamento <= percentualCruzamento)
               disp("------------------ CRUZAMENTO ------------------");
               pesos = Cruzamento(pesos, FFitness);
               disp("------------------------------------------------");
             end

%% Mutação
                probmutacao = rand;
```

```
                if (probmutacao <= percentualMutacao)
                  disp("------------------ MUTAÇÃO ------------------");
                  individuo = randi([1 tamanhoPopulacao],1);                          %
Sorteio do indivíduo para a mutação
                  disp("Mutação no indivíduo: "+ individuo);
                  pesos = Mutacao(pesos, individuo);
                  contaMutacoes = contaMutacoes + 1;
                  disp("-------------------------------------------");
                end
                ErroTotal = sum(sum(erro));
                disp("Erros encontrados: "+ErroTotal);
                disp(" ");
                iteracao = iteracao + 1;
            end
        end
%     posicao
%     pesos(posicao,:)
    pesosAtualizados = pesos(posicao,:);
end

    %% step function
    function SaidasBinarias = StepFunction(SaidaCalculada)
        for k = 1:size(SaidaCalculada, 1)
            for l = 1: size(SaidaCalculada, 2)
                if (SaidaCalculada(k,l) >= 1)
                    SaidaCalculada(k,l) = 1;
                else
                    SaidaCalculada(k,l) = 0;
                end
            end
        end
        SaidasBinarias = SaidaCalculada;
    end

    %% Teste
    function teste(entradas, PesosAtualizados)
         for j = 1: size(PesosAtualizados, 1)
             SaidaCal(j,:) = entradas * PesosAtualizados(j,:)';
         end
            saida = StepFunction(SaidaCal');
%             disp(saida);
    end

    %% Inicia População
    function Pesos = IniciaPopulacao(tamanhoPopulacao, quantidadePesos)
        for i = 1: tamanhoPopulacao
            Pesos(i,:) = rand(1,quantidadePesos);
        end
    end

    %% Fitness
    function fitness = Fitness(erro, saida)
        fitness = saida - sum(erro);
    end

    %% Cruzamento
    function Populacao = Cruzamento(pesos, fitness)
        total = sum(fitness);
        numeracao = zeros(1, size(pesos,1));
        for i = 1: size(pesos, 1)
            probabilidade(:,i) = (fitness(:,i) * 100)/total;                 %
probabilidade de cruzamento
            if (i > 1)
                numeracao(:,i+1) = numeracao(:,i) + probabilidade(:,i);     % valor
obtido através de seu percentual
            else
                numeracao(:,i+1) = probabilidade(:,i);                      % valor
obtido através de seu percentual
            end
        end

        %% Sorteio (uso da roleta)
        ValorIndividuo1 = (100-1).*rand(1,1) + 1;
        ValorIndividuo2 = (100-1).*rand(1,1) + 1;
        individuo1 = 0;
        individuo2 = 0;
```

```
        %% Pega a linha do indivíduo
        for j = 1: size(pesos, 1)
            if (ValorIndividuo1 >= numeracao(:,j) && ValorIndividuo1 <=
numeracao(:,j+1))
                individuo1 = j;
            end
            if (ValorIndividuo2 >= numeracao(:,j) && ValorIndividuo2 <=
numeracao(:,j+1))
                individuo2 = j;
            end
        end

       [~, MaisFraco] = min(fitness);

        %% O cruzamento - (Ponto de cruzamento único)
        PontoDeCruzamento = randi([1 size(pesos,2)],1);
        for k = 1: size(pesos, 2)
            if (k <= PontoDeCruzamento)
                pesos(MaisFraco, k, :) = pesos(individuo1, k, :);
            else
                pesos(MaisFraco, k, :) = pesos(individuo2, k, :);
            end
        end

        %% Dados do cruzamento
        disp("Ponto de cruzamento: ")
        PontoDeCruzamento
        disp("Pais: ");
        individuo1
        pesos(individuo1,:)
        individuo2
        pesos(individuo2,:)
        disp("Filho: ");
        MaisFraco
        pesos(MaisFraco,:)
        % retorno dos novos pesos
        Populacao = pesos;
    end

    %% Mutação
    function Pesos = Mutacao(pesos, l)
      min = 1;
      max = size(pesos, 2);
      c = randi([min max],1); % peso (gene) sorteado para ser alterado
      disp("Gene: "+ c);
      valor = rand(1);      % valor a ser inserido no peso escolhido
      pesos(l, c,:) = valor; % peso recebendo a mutação
      Pesos = pesos;         % passando os pesos novos para o treino
    end
```

10.3. ELM (Código em Matlab)

```
     Base = readtable('column_3C.dat');
    QtdeEntradas = size(Base,1);
    Base = Base(randperm(QtdeEntradas),:);
    QtdeAtributos = size(Base,2)-1;
    Dados = Base{:,1:QtdeAtributos};
    ClassesCategoricas = categorical(Base{:,size(Base,2):size(Base,2)});
    Saida = grp2idx(ClassesCategoricas);
    QtdeClasses = size(unique(Saida), 1);

%% Padronização dos dados
    for i = 1:QtdeAtributos
        Dados(:, i) = ((Dados(:, i) - mean(Dados(:, i)))/std(Dados(:, i)));
    end

%% Organização das Saídas
    SaidaBinaria = zeros(QtdeEntradas,QtdeClasses);
    for x = 1: QtdeEntradas
         SaidaBinaria(x, Saida(x)) = 1;
    end

%% Parâmetros iniciais
    NTreino = QtdeEntradas * 0.8;                       % 80% para treino
    NTeste = QtdeEntradas-NTreino;                      % 20% para teste
    DadosTreino = Dados(1:NTreino,:);
```

```
    SaidasTreino = SaidaBinaria(1:NTreino,:);
    DadosTeste = Dados(NTreino+1:QtdeEntradas,:);
    SaidasTeste = SaidaBinaria(NTreino+1:QtdeEntradas,:);
    N_Neuronios = 30;
    pesos = rand(N_Neuronios,QtdeAtributos);
    bias = rand(N_Neuronios,1);

%% Treino
    for i=1:NTreino
        SaidaCalculada = pesos * DadosTreino(i,:)'+ bias;
        SaidaCalculada = sigmf(SaidaCalculada,[1 0]);
        H_Treino(:,i) = SaidaCalculada;
    end
    H_Treino = H_Treino';
    beta = pinv(H_Treino) * SaidasTreino;

%% Teste
    DadosTeste(1,:);
    for i = 1: NTeste
        SaidaCalculada = pesos * DadosTeste(i,:)'+ bias;
        SaidaCalculada = sigmf(SaidaCalculada,[1 0]);
        H_Teste(:,i) = SaidaCalculada;
    end
    H_Teste = H_Teste';
    SaidaTeste = H_Teste * beta;

%% Transformação da Saida do Teste em Saída Binária
    for i = 1:NTeste
        maior = max(SaidaTeste(i,:));
        for j = 1:QtdeClasses
            if(SaidaTeste(i,j) == maior)
                SaidaTeste(i,j) = 1;
            else
                SaidaTeste(i,j) = 0;
            end
        end
    end
%% Cálculo da acurácia
    x = 0;
    for i = 1: NTeste
        if(SaidaTeste(i,:) == SaidasTeste(i,:))
            x = x + 1;
        end
    end
    acuracia = x/NTeste

%% Plotagem da matriz de confusão
    plotconfusion(SaidasTeste' , SaidaTeste' );
    title("Dados da classificação")
    xlabel("Saídas corretas")
    ylabel("Saídas calculadas")
    set(gca,'yticklabel',{'classe 1' 'classe 2' 'classe 3' 'Acerto Total'})
    set(gca,'xticklabel',{'classe 1' 'classe 2' 'classe 3' 'Acurácia'})
    toc
```

10.4. GA-ELM (Código em Matlab)

```
    Base = readtable('column_3C.dat');
    QtdeEntradas = size(Base,1);
    Base = Base(randperm(QtdeEntradas),:);
    QtdeAtributos = size(Base,2)-1;
    Dados = Base{:,1:QtdeAtributos};
    ClassesCategoricas = categorical(Base{:,size(Base,2):size(Base,2)});
    Saida = grp2idx(ClassesCategoricas);
    QtdeClasses = size(unique(Saida), 1);

%% Padronização dos dados
    for i = 1:QtdeAtributos
        Dados(:, i) = ((Dados(:, i) - mean(Dados(:, i)))/std(Dados(:, i)));
    end

%% Organização das Saídas
    SaidaBinaria = zeros(QtdeEntradas,QtdeClasses);
    for x = 1: QtdeEntradas
         SaidaBinaria(x, Saida(x)) = 1;
    end
```

```
%% Parâmetros iniciais
    NTreino = QtdeEntradas * 0.8;                              % 80% para treino
    NTeste = QtdeEntradas - NTreino;                           % 20% para teste
    DadosTreino = Dados(1:NTreino,:);
    SaidasTreino = SaidaBinaria(1:NTreino,:);
    DadosTeste = Dados(NTreino + 1:QtdeEntradas,:);
    SaidasTeste = SaidaBinaria(NTreino + 1:QtdeEntradas,:);
    N_Neuronios = 30;
    pesos = rand(N_Neuronios, QtdeAtributos);
    bias = rand(N_Neuronios, 1);
    TamPopulacao = 50;
    pesosEscolhidos = IniciaPopulacao(N_Neuronios, TamPopulacao);
    percentualMutacao = 0.1;
    percentualCruzamento = 1;
    geracoes = 10;

for l = 1: geracoes
    disp("Geração: "+ l);
    for k = 1: TamPopulacao
        contaMutacoes = 0;

    %% Faz a poda de acordo com a população
        pesosNovos = pesos;
        for i = 1: N_Neuronios
           if (pesosEscolhidos(k,i,:) ~= 1)
               pesosNovos(i,:) = zeros();
           end
        end
%         pesosEscolhidos(k,:)
%         pesosNovos

     %% Treinamento
        H_Treino = [];
        [SaidaCalculada, H_Treino, beta] = Treinamento(NTreino, pesosNovos, DadosTreino,
bias, SaidasTreino);

    %% Teste
        H_Teste = [];
        SaidaTeste = Teste(NTeste, pesosNovos, DadosTeste, bias, beta, QtdeClasses);

    %% Cálculo da acurácia/Fitness
       fitness(k,:) = Fitness(NTeste, SaidaTeste, SaidasTeste);
    end
    %% Mutação
        probmutacao = rand;
        if (probmutacao <= percentualMutacao)
           disp("----------------- MUTAÇÃO -----------------");
           individuo = randi([1 TamPopulacao],1);                          % Sorteio do
indivíduo para a mutação
           disp("Mutação no indivíduo: "+ individuo);
           pesosEscolhidos(individuo,:)
           pesosEscolhidos = Mutacao(pesosEscolhidos, individuo, N_Neuronios);
           contaMutacoes = contaMutacoes + 1;
           pesosEscolhidos(individuo,:)
           disp("-------------------------------------------");
        end
    fitness(k,:)
    %% Cruzamento
        probcruzamento = rand;
            if (probcruzamento <= percentualCruzamento)
                disp("----------------- CRUZAMENTO -----------------");
                pesosEscolhidos = Cruzamento(pesosEscolhidos,  fitness);
                disp("-------------------------------------------");
            end

     %% Encontrando o melhor indíviduo por geração
     fitness
     [acuracia individuo] = max(fitness)
     pesosEscolhidos(individuo,:)

end


%% Plotagem da matriz de confusão
    plotconfusion(SaidasTeste' , SaidaTeste' );
    title("Dados da classificação")
```

```
    xlabel("Saídas corretas")
    ylabel("Saídas calculadas")
    set(gca,'yticklabel',{'classe 1' 'classe 2' 'classe 3' ''})
    set(gca,'xticklabel',{'classe 1' 'classe 2' 'classe 3' 'Acurácia'})
    toc
%% Métodos da ELM
    %% Treinamento
    function [SaidaCalculada, H_Treino, beta] = Treinamento(NTreino, pesosNovos,
DadosTreino, bias, SaidasTreino)
    for i = 1: NTreino
        SaidaCalculada = pesosNovos * DadosTreino(i,:)'+ bias;
        SaidaCalculada = sigmf(SaidaCalculada,[1 0]);           % Função sigmoide
        H_Treino(:,i) = SaidaCalculada;
    end
    H_Treino = H_Treino';
    beta = pinv(H_Treino) * SaidasTreino;
end

    %% Testes
    function SaidaTeste = Teste(NTeste, pesosNovos, DadosTeste, bias, beta, QtdeClasses)
    for i = 1: NTeste
        SaidaCalculada = pesosNovos * DadosTeste(i,:)'+ bias;
        SaidaCalculada = sigmf(SaidaCalculada,[1 0]);           % Função sigmoide
        H_Teste(:,i) = SaidaCalculada;
    end
    H_Teste = H_Teste';
    SaidaTeste = H_Teste * beta;

    %% Transformação da Saida do Teste em Saída Binária
    for i = 1:NTeste
        maior = max(SaidaTeste(i,:));
        for j = 1:QtdeClasses
            if(SaidaTeste(i,j) == maior)
                SaidaTeste(i,j) = 1;
            else
                SaidaTeste(i,j) = 0;
            end
        end
    end
end

%% Métodos do AG
    %% Inicia População
    function Populacao = IniciaPopulacao(N_Neuronios, TamPopulacao)
        for i = 1: TamPopulacao
            Populacao(i,:) = randi([0 1],1,N_Neuronios);
        end
   end

    %% Mutação
    function Pesos = Mutacao(pesos, individuo, N_Neuronios)
        max = N_Neuronios;
        gene = randi([1 max],1)
        if (pesos(individuo,gene,:)) == 1
            pesos(individuo,gene,:) = 0;
        else
            pesos(individuo,gene,:) = 1;
        end
        Pesos = pesos;
    end

    %% Fitness
    function fitness = Fitness(NTeste, SaidaTeste, SaidasTeste)
         x = 0;
        for i = 1: NTeste
            if(SaidaTeste(i,:) == SaidasTeste(i,:))
                x = x + 1;
            end
        end
        fitness = x/NTeste;
    end

    %% Cruzamento
    function Populacao = Cruzamento(pesos, fitness)
        pesos
        fitness
        total = sum(fitness)
```

```
        for i = 1: size(pesos, 1)
            probabilidade(:,i) = (fitness(i,:) * 100)/total;
             if (i > 1)
                numeracao(:,i+1) = numeracao(:,i) + probabilidade(:,i);     % valor
obtido através de seu percentual
            else
                numeracao(:,i+1) = probabilidade(:,i);                      % valor
obtido através de seu percentual
            end
        end
        %% Sorteio (uso da roleta)
        ValorIndividuo1 = (100-1).*rand(1,1) + 1;
        ValorIndividuo2 = (100-1).*rand(1,1) + 1;
        individuo1 = 0;
        individuo2 = 0;

        %% Pega a linha do indivíduo
        for j = 1: size(pesos, 1)
            if (ValorIndividuo1 >= numeracao(:,j) && ValorIndividuo1 <=
numeracao(:,j+1))
                individuo1 = j;
            end
            if (ValorIndividuo2 >= numeracao(:,j) && ValorIndividuo2 <=
numeracao(:,j+1))
                individuo2 = j;
            end
        end

       [~, MaisFraco] = min(fitness)

        %% O cruzamento - (Ponto de cruzamento único)
        PontoDeCruzamento = randi([1 size(pesos,2)],1);
        for k = 1: size(pesos, 2)
            if (k <= PontoDeCruzamento)
                pesos(MaisFraco, k, :) = pesos(individuo1, k, :);
            else
                pesos(MaisFraco, k, :) = pesos(individuo2, k, :);
            end
        end
        %% Dados do cruzamento
        disp("Ponto de cruzamento: ")
        PontoDeCruzamento
        disp("Pais: ");
        individuo1
        fitness(individuo1,:)
        pesos(individuo1,:)
        individuo2
        fitness(individuo2,:)
        pesos(individuo2,:)
        disp("Filho: "+ MaisFraco);
        pesos(MaisFraco,:)
        % retorno dos novos pesos
        Populacao = pesos;
    end
```