Universidade Federal do Maranhão
Programa de Pós Graduação Em Ciência da Computação

Aluno
Carlos Eduardo Nascimento Cajado

Professor
Prof. Dr. Areolino de Almeida Neto

RECONHECIMENTO DE FONEMAS VIA REDE DE KOHONEN

São Luís, MA
junho- 2024

1. Introdução

1.1 Contexto

Ao longo da história, com o crescimento da comunidade científica, é indubitável que o fascínio pela anatomia e fisiologia do corpo humano tem sido uma das principais fontes de pesquisa para tentar compreender a complexidade do raciocínio-matemático pertencente ao Homo sapiens. Onde, para Frank Rosenblatt, em artigo intitulado “The Perceptron: a Probabilistic Model For Information Storage And Organization In The Brain”, esse fascínio se estende especialmente nas investigações para desenvolver e simular, computacionalmente, o processamento lógico cerebral.

Posteriormente, em consonância aos estudos de Rosenblatt, se apresentou o conceito de redes neurais artificiais que, para Haykin (2009) rede neural é uma máquina projetada para modelar a maneira como o cérebro executa uma determinada tarefa ou função de interesse. Sendo, geralmente, implementada por meio de componentes eletrônicos ou é simulada em software em um computador digital.

Ademais, é importante salientar a existência de redes neurais não supervisionadas, um exemplo relevante é o mapa de Kohonen, ou Rede Neural de Mapas Auto-Organizáveis (Self-Organizing Map, SOM). O SOM organiza os dados de maneira que pontos semelhantes fiquem próximos no mapa, facilitando a visualização e análise de padrões complexos. Atualmente, é utilizado em tarefas de agrupamento, redução de dimensionalidade e visualização de dados, destacando-se pela sua capacidade de auto-organização e preservação das relações topológicas (Dias, 2022).

.

1.2 Problema

Neste estudo, utilizaremos a rede neural Kohonen (Self-Organizing Map, SOM) visando aprender a reconhecer as sílabas fonéticas das palavras “DIREITA” e “ESQUERDA”.

1.3 Objetivos

- Desenvolver e Treinar a Rede Neural Kohonen.
- Analisar a Precisão do algoritmo.
- Verificar o reconhecimento das palavras.
- Determinar o melhor ponto de parada do treinamento.

1.4 Configuração do Ambiente

Para desenvolvimento do projeto, temos as seguintes configurações:

1.3.1 Software:

- Windows 11
- Audacity
- Matlab

1.3.2 Hardware:

- Processador: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
- Memória RAM: 16,0 GB (utilizável: 15,9 GB)

2. Bases de Dados

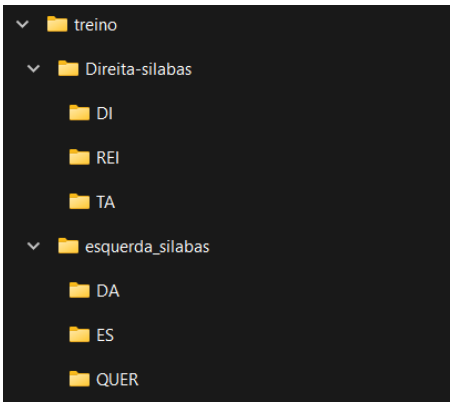
2.1 pré-processamento:

Primeiramente, utilizando o aplicativo "Audacity", é fundamental segmentar as sílabas fonéticas (DI, REI, TA, ES, QUER, DA) para serem utilizadas no treinamento. Nesse caso, temos 73 amostras da palavra "esquerda" e 58 amostras de "direita" como conjunto totais.

2.2 Dividindo o *Database*:

Treino: 75%, Validação: 15%, Teste: 10% . Temos a seguinte divisão de diretórios, a exemplo na figura 1 para treino:

figura 1: Particionamento das Sílabas



Autor: próprio

Essa divisão se repete para pasta de validação e teste.

Lendo os arquivos de áudio no matlab :

Utilizando o software MATLAB, serão desenvolvidos vários scripts de pré-processamento. Cada pasta de áudios, incluindo as de treino, teste e validação, terá um

script dedicado que seguirá as etapas abaixo:

1. Leitura dos caminhos de cada sílaba.
2. Processar cada faixa de áudio utilizando a biblioteca “audioRead”.
3. Calcular a Transformada Rápida de Fourier (FFT) da primeira coluna dos dados de áudio, retornando a magnitude, abs (valor absoluto).
4. Atribuir a cada sílaba , utilizado “mear”, o valor da média dos elementos.
5. Salvar variáveis do MatLab, em arquivo com extensão mat.

A exemplo temos, na figura 2, um trecho correspondente ao arquivo treino. Visualizamos os passos para a fonema “DI”.

Figura 2: Trecho pré-processamento de treino.

```
% Diretório para cada sigla
diretorio_DI = 'treino\Direita-silabas\DI';
diretorio_REI = 'treino\Direita-silabas\REI';
diretorio_TA = 'treino\Direita-silabas\TA';

% Carregando audios e inicializando células para armazenar dados de FFT
audio_di = dir(fullfile(diretorio_DI, '*.wav'));
audio_rei = dir(fullfile(diretorio_REI, '*.wav'));
audio_ta = dir(fullfile(diretorio_TA, '*.wav'));


dados_audio_di = cell(1, numel(audio_di));
dados_audio_rei = cell(1, numel(audio_rei));
dados_audio_ta = cell(1, numel(audio_ta));

N_partes = 128;
% áudios DI
for i = 1:numel(audio_di)

     arquivo_atual = fullfile(diretorio_DI, audio_di(i).name);
     [audio_data, ~] = audioread(arquivo_atual);
     fft_data = abs(fft(audio_data(:,1)));

    % Dividindo fft_data em partes
    tamanho_parte = floor(numel(fft_data) / N_partes);
    partes_fft = reshape(fft_data(1:N_partes*tamanho_parte), tamanho_parte, N_partes);

    % Calculando a média de cada parte
     dados_audio_di{i} = mean(partes_fft) ;
end
```

Autor: próprio

O vetor resultante da FFT será dividido em “N_partes” visando reduzir a dimensionalidade dos dados de áudio e extrair características representativas de cada segmento do espectro de frequências.

3 Treinamento:

Partindo da configuração:

- Criando as correspondências das classes.

Tabela 1: correspondências das classes.

Fonema	Número classe correspondente
“DI”	1
“REI”	2
“TA”	3
“ES”	4
“QUER”	5
“DA”	6

Autor: próprio

- Carregando os arquivos de áudios salvos na figura 3.

Figura 3: carregando áudios.

```
% Carregar os dados DIREITA!!
load('dados_audio_DI.mat');
load('dados_audio_REI.mat');
load('dados_audio_TA.mat');

% Carregar os dados ESQUERDA!!
load('dados_audio_ES.mat');
load('dados_audio_QUER.mat');
load('dados_audio_DA.mat');
```

Autor: próprio

- Os pesos são aleatoriamente a cada treinamento.

pesos = rand(size(X, 2), Neuronios);

- Na figura 4, podemos verificar os parâmetros iniciais para funcionamento, Seguimos:

Figura 4: Parâmetros da rede.

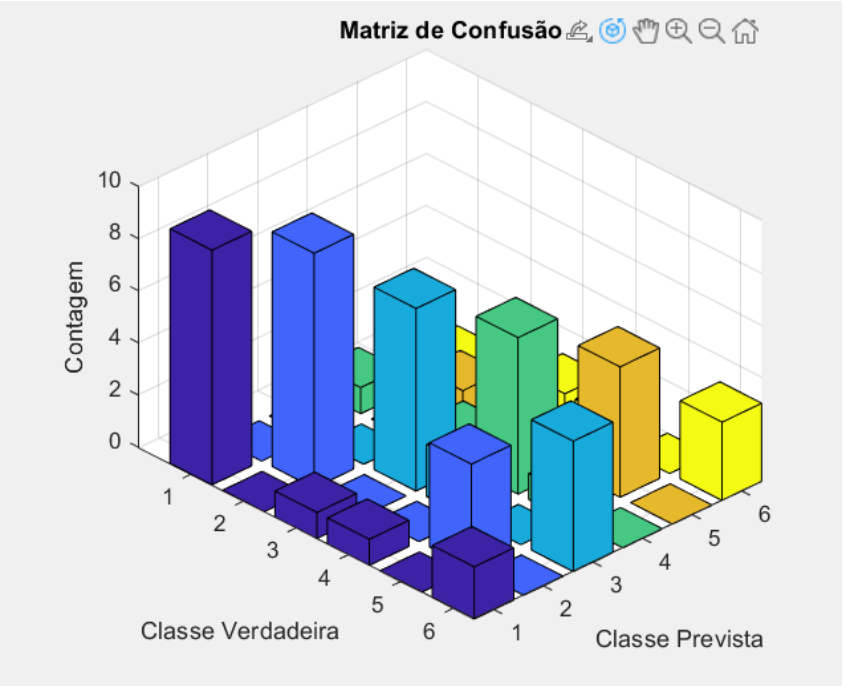
```
sigma0 = 2;          % Sigma inicial
eta0 = 0.01;         % Taxa de aprendizado inicial
tau_sigma = 10000;   % Tempo de decaimento do sigma
tau_eta = 12000;     % Tempo de decaimento da taxa de aprendizado
epocas = 10000;      % Número de épocas
Neuronios = 24;      % Número de neurônios
```

Autor: próprio

3.1 Validação:

Rodando o algoritmo de validação temos a seguinte matriz de confusão.

Figura 5: Matriz de confusão validação.



Autor: próprio

3.2 Algoritmo de treinamento:

Adicionado Atributos (X) e classes (Y). Seguimos:

Figura 6: Definindo, dados , atributos e classes.

```
dados_di = cat(1, dados_audio_di{:});
dados_rei= cat(1, dados_audio_rei{:});
dados_ta = cat(1, dados_audio_ta{:});

dados_es = cat(1, dados_audio_es{:});
dados_quer = cat(1, dados_audio_quer{:});
dados_da = cat(1, dados_audio_da{:});


%6 CLASSES; DI, REI, TA, ES, QUER, DA
X = [dados_di; dados_rei; dados_ta; dados_es;  dados_quer; dados_da];

Y = [1*ones(size(dados_di', 2), 1); 2*ones(size(dados_rei', 2), 1); 3*ones(size(dados_ta', 2), 1);
        4*ones(size(dados_es', 2), 1); 5*ones(size(dados_quer', 2), 1); 6*ones(size(dados_quer', 2), 1)];
```

Autor: próprio

Primeiramente, na figura 7 iniciamos o processo competitivo, com uma amostra X(a,:) é selecionada e a distância euclidiana entre ela e os vetores de pesos dos neurônios é calculada para encontrar o neurônio vencedor, ou seja, aquele com a menor distância.

Figura 7: distância euclidiana, neurônio vencedor

```
for t = 1:epocas
    for a = 1:size(X, 1)

        %Processo competitivo:
        amostra = X(a, :)';

        % Encontrar o neurônio vencedor
        %distância euclidiana entre o vetor x e os vetores de pesos

        distancias = sum((pesos - amostra).^2, 1);
        [~, vencedor] = min(distancias);

        % Iniciando os Pesos
        pesosAnteriores = pesos;
```

Autor: próprio

Após isso, os pesos anteriores são armazenados e, para cada neurônio, é iniciado o processo cooperativo, calculando a função de vizinhança baseada na distância entre o neurônio atual e o vencedor.

Figura 8: processo cooperativo

```
%Para cada neurônio de saída temos:
for n = 1:Neuronios
    %Processo cooperativo:
    distancia = (n - vencedor)^2;
    h = funcao_vizinhanca(distancia, sigma_atual);
```

Autor: próprio

Temos a definição da função vizinhança:

funcao_vizinhanca = @(distancia, sigma) exp(-distancia^2 / (2 * sigma^2));

Seguimos agora para o processo adaptativo onde os pesos dos neurônios são ajustados usando a taxa de aprendizado atual, a função de vizinhança e a diferença entre a amostra e os pesos.

pesos(:, n) = pesos(:, n) + eta_atual * h * (amostra - pesos(:, n));

Finalmente, é definido o conjuntos dos vencedores e a cada um a uma classe atribuída ao Y, valor conhecido previamente, na figura 9 temos:

Figura 9: conjunto vencedores

```
% Determinação dos Vencedores
numAmostras = size(X, 1);
vencedores = zeros(numAmostras, 1);
for amostra = 1:numAmostras
    [~, vencedores(amostra)] = min(sum((pesos - X(amostra, :)').^2, 1));
end

% Atribuição de Classes
classificacao = zeros(Neuronios, 1);
for neuronio = 1:Neuronios
    amostras = find(vencedores == neuronio);
    if ~isempty(amostras)
        classificacao(neuronio) = mode(Y(amostras));
    end
end
```

Autor: próprio

Saída da rede Neural:

Para a impressão de progresso, usaremos a máxima alteração nos pesos (max_delta_w) onde é calculada como a maior diferença absoluta entre os pesos atuais e os pesos anteriores. A cada 100 épocas, é exibida uma mensagem mostrando a época atual e o valor da variável "max_delta_w". Além disso, se for menor que um valor significativo pré definido o loop de treinamento é interrompido.

Figura 10: critérios de parada

```
% Impressão de progresso
max_delta_w = max(max(abs(pesos - pesosAnteriores)));
if mod(t, 100) == 0
    disp(['Época ', num2str(t), ': max_delta_w = ', num2str(max_delta_w)])
end

% Critério de parada
if max_delta_w < max_delta_w_significativo
    disp(['Critério de parada atingido na época ', num2str(t)]);
    break;
```

Autor: próprio

Como saída no terminal, figura 4, para 24 neurônios temos:

Figura 11: Output terminal treinamento

```
Época 9600: max_delta_w = 0.035183
Época 9700: max_delta_w = 0.03486
Época 9800: max_delta_w = 0.03472
Época 9900: max_delta_w = 0.034408
Época 10000: max_delta_w = 0.034092
```

Autor: próprio

4 Testes :

Para o *script* de teste, serão passados os atributos X inéditos, ainda não vistos pela rede.

Figura 12: carregando amostras teste

```
% Carregar os dados DIREITA!!
load('dados_audio_DI_teste.mat');
load('dados_audio_REI_teste.mat');
load('dados_audio_TA_teste.mat');

% Carregar os dados ESQUERDA!!
load('dados_audio_ES_teste.mat');
load('dados_audio_QUER_teste.mat');
load('dados_audio_DA_teste.mat');

dados_teste_di = cat(1, dados_audio_di_teste{:});
dados_teste_rei= cat(1, dados_audio_rei_teste{:});
dados_teste_ta = cat(1, dados_audio_ta_teste{:});

dados_teste_es = cat(1, dados_audio_es_teste{:});
dados_teste_quer = cat(1, dados_audio_quer_teste{:});
dados_teste_da = cat(1, dados_audio_da_teste{:});
```

Autor: próprio

4.1 Classificando as amostras (TESTE):

Pode-se verificar o algoritmo de teste criado, figura 12:

Figura 12: algoritmos de teste

```
% Testar a rede
numAmostrasTeste = size(X_teste, 1);
numNeuronios = size(pesos, 2);
vencedores = zeros(numAmostrasTeste, 1);

% Encontrar os neurônios vencedores para cada amostra de teste.
for i = 1:numAmostrasTeste
    distancias = sum((pesos - X_teste(i, :)').^2, 1);
    [~, vencedor] = min(distancias);
    vencedores(i) = vencedor;
end

rotulosTeste = Y_teste;
numClasses = length(unique(rotulosTeste));
rotuloMaisComum = zeros(numNeuronios, 1);

% Encontrar o rótulo mais comum para cada neurônio, temos:
for neuronio = 1:numNeuronios
    indicesNeuronio = find(vencedores == neuronio);
    classesNeuronio = rotulosTeste(indicesNeuronio);
    if ~isempty(classesNeuronio)
        rotuloMaisComum(neuronio) = mode(classesNeuronio);
    end
end

numAcertos = 0;
classePrevista = zeros(numAmostrasTeste, 1);

% Prever a classe para cada amostra de teste
for i = 1:numAmostrasTeste
    neuronio = vencedores(i);
    classePrevista(i) = rotuloMaisComum(neuronio);
    if rotulosTeste(i) == classePrevista(i)
        numAcertos = numAcertos + 1;
    end
end
```

Autor: próprio

4.2 Saída arquivo teste:

Como saída, após alguns experimentos de treinamento e seleção dos melhores pesos variando o número neurônios:

Tabela 2: Métricas por quantidade de neurônios .

Número de neurônios	Número de acertos por amostras	Taxa de acerto	Precisão média:	Recall médio
6	21/42	50.00%	0.50	0.53
12	28/42	66.67%	0.67	0.68
24	35/42	83.33%	0.83	0.89
100	38/42	90.48%	0.90	0.94

Autor: próprio

Com a análise da tabela, é indubitável que a rede Kohonen conseguiu aprender e generalizar as previsões para os dados de teste. Salienta-se que, à medida que aumentamos o número de neurônios, o tempo de treinamento aumenta, mas facilita o aprendizado e melhora a acurácia dos testes.

5 Reconhecer palavras:

5.1 Algoritmo

Criando uma função que recebe 3 sílabas, na ordem correta, e retorna a palavra caso seja reconhecida pela rede. na Figura 12, temos:

Figura 12: algoritmos para reconhecer palavras

```
function palavra_reconhecida = reconhece_palavra(silaba_1, silaba_2, silaba_3)

    palavra_reconhecida = '';

    % Chamada da função Saida_rede para a primeira sílaba
    classe_prevista = Saida_rede(silaba_1);
    % Verifica a classe prevista e atualiza a palavra reconhecida
    if classe_prevista == 1
        disp('Sílaba DI reconhecida');
        classe_prevista = Saida_rede(silaba_2);
        if classe_prevista == 2
            disp('Sílaba REI reconhecida');
            % Chamada da função Saida_rede para a terceira sílaba
            classe_prevista = Saida_rede(silaba_3);
            if classe_prevista == 3
                disp('Sílaba TA reconhecida');
                palavra_reconhecida = 'DIREITA';
            end
        end
    elseif classe_prevista == 4
        disp('Sílaba ES reconhecida');
        classe_prevista = Saida_rede(silaba_2);
        if classe_prevista == 5
            disp('Sílaba QUER reconhecida');
            % Chamada da função Saida_rede para a terceira sílaba
            classe_prevista = Saida_rede(silaba_3);
            if classe_prevista == 6
                disp('Sílaba DA reconhecida');
                palavra_reconhecida = 'ESQUERDA';
            end
        end
    else
        palavra_reconhecida = 'PALAVRA NÃO RECONHECIDA';
    end
end
```

Autor: próprio

5.2 Testando reconhecendo as palavras:

Chamando a função, para dois teste, podemos encontrar:

Figura 13: aplicando o reconhecimento

```
% Atribuindo as três sílabas
silaba_1 = X_silaba_1(3,:); % usando a amostra Nº3
silaba_2 = X_silaba_2(5,:); % usando a amostra Nº5
silaba_3 = X_silaba_3(2,:); % usando a amostra Nº3


X_silaba_4 = cat(1, dados_audio_es_teste{:});
X_silaba_5 = cat(1, dados_audio_quer_teste{:});
X_silaba_6 = cat(1, dados_audio_da_teste{:});

% Atribuindo as três sílabas
silaba_4 = X_silaba_4(1,:); % usando a amostra Nº3
silaba_5 = X_silaba_5(4,:); % usando a amostra Nº5
silaba_6 = X_silaba_6(2,:); % usando a amostra Nº2

fprintf('Treinamento - Kohonen - reconhecimento palavras\n');
% Chamada da função reconhece_palavra com as três sílabas teste 01
palavra_reconhecida = reconhece_palavra(silaba_1, silaba_2 , silaba_3);

% Exibe a palavra reconhecida
disp(['Saída: ', palavra_reconhecida]);

fprintf('_________________Teste 02_____________________________\n');

% Chamada da função reconhece_palavra com as três sílabas teste 02
palavra_reconhecida = reconhece_palavra(silaba_4, silaba_5 , silaba_6);

% Exibe a palavra reconhecida
disp(['Saída: ', palavra_reconhecida]);
```

Autor: próprio

Então, na figura 14, podemos utilizar o método para calcular a classe prevista.

Figura 14: Calcula a classe prevista

```
function classePrevista = Saida_rede(X)
    load('pesos_bias_treinados.mat');

    numAmostras = size(X, 1);
    numNeuronios = size(pesos, 2);
    vencedores = zeros(numAmostras, 1);

    % Encontrar os neurônios vencedores para cada amostra de entrada
    for i = 1:numAmostras
        distancias = sum((pesos - X(i, :)').^2, 1);
        [~, vencedor] = min(distancias);
        vencedores(i) = vencedor;
    end

    % Prever a classe para cada amostra de entrada
    classePrevista = zeros(numAmostras, 1);
    for i = 1:numAmostras
        neuronio = vencedores(i);
        classePrevista(i) = rotuloMaisComum(neuronio);
    end
end
```

Autor: próprio

Como saída temos:

Figura 15: Output terminal teste (Reconhecendo palavras) .

```
Treinamento - Kohonen - reconhecimento palavras
Sílaba DI reconhecida
Sílaba REI reconhecida
Sílaba TA reconhecida
Saída: DIREITA
___________________Teste 02_____________________________
Sílaba ES reconhecida
Sílaba QUER reconhecida
Sílaba DA reconhecida
Saída: ESQUERDA
```

Autor: próprio

É notório que, para as sílabas escolhidas, a rede conseguiu prever com eficiência e reconhecer a palavra gerada pelos fonemas.

Conclusão

O modelo via rede de Kohonen foi capaz de identificar corretamente as sílabas fonéticas das palavras "DIREITA" e "ESQUERDA", alcançando uma alta taxa de acerto nas amostras de teste, principalmente com o aumento do números de neurônios na camada de saída . Este trabalho destaca a capacidade das SOMs em tarefas de reconhecimento de padrões abrindo espaço para futuras pesquisas e aprimoramentos, incluindo a otimização de parâmetros e a aplicação em outras áreas de reconhecimento de fala.

REFERÊNCIAS :

ROSENBLATT, Frank. **The perceptron: a probabilistic model for information storage and organization in the brain**. Psychological review, v. 65, n. 6, p. 386, 1958.

HAYKIN, Simon. **Neural networks and learning machines**, 3/E. Pearson Education India, 2009.

Dias, Isabelle. **MAPAS DE KOHONEN: ESTUDO APLICADO NO APRENDIZADO NÃO SUPERVISIONADO DE DÍGITOS MANUSCRITOS**, (Trabalho conclusão de curso, 2022)