Processamento Digital do Sinal

Diego António Nunes Cabral e Silva - a33872
José João Pimenta Oliveira - a35466

Relatório do Primeiro Trabalho Prático de Processamento Digital do Sinal.

Docente:

Prof. Rui Fernandes, Prof. João Paulo Teixeira

Bragança

2018

Resumo

Este trabalho tem como objetivo o reconhecimento de dez letras, que serão apresentadas de seguida, criadas em matrizes 5x5 que posteriormente foram organizadas numa só matriz coluna, essa que será a nossa matriz *input*.

Posteriormente, são produzidos propositadamente, um e dois erros para demonstrar a taxa de sucesso da rede neuronal. Após o treino desta rede com diferente funções de treino, e variando também o número de nós na camada escondida.

Palavras-chave: Rede Neuronal, Treino, MatLab

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Uma rede neuronal artificial são modelos computacionais, inspirados pelo cérebro, que são capazes de "aprender" e a partir daí melhorar a sua performance sobre uma dada tarefa como por exemplo, reconhecimento de imagens.

O problema apresentado consiste no reconhecimento de imagens. Esta poderá ter variadas aplicações, tais como o desenvolvimento de uma aplicação para deteção dos mais variados sinais, imagens e símbolos.

Capítulo 2

Metodologias

2.1 Matriz de caracteres

A matriz representa um caracter alfabético numa matriz [5x5]. A figura 2.1 mostra as matrizes que foram escolhidas, sendo que o fundo preto representa o bit 1 e o branco representa o bit 0.

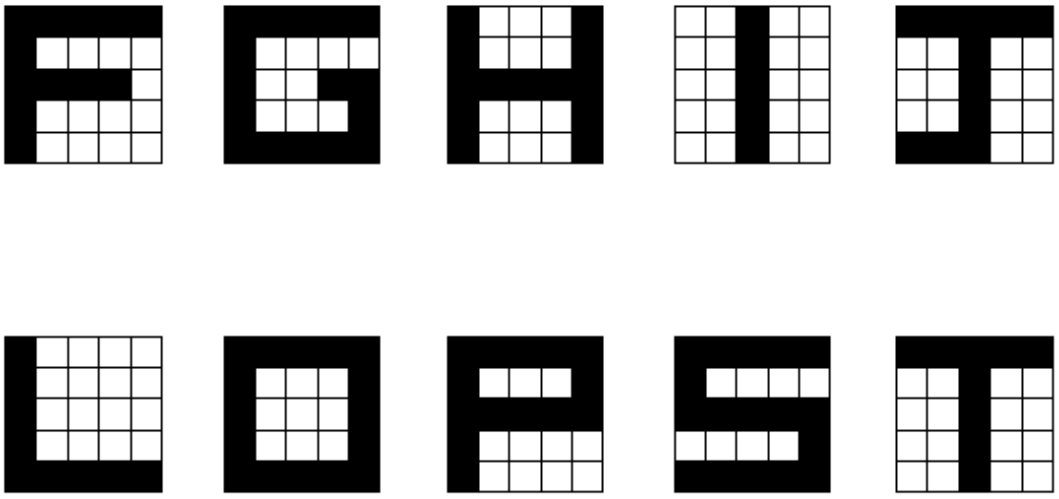

Figura 2.1: Matriz de caracteres escolhida.

2.2 Codificação das matrizes de entrada p, e saída t

A matriz p é composta por 25 linhas e 10 colunas (Figura 2.2). Cada coluna representa um caracter que já havia sido representado (Figura 2.1).

1	1	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	0	0	1
1	1	0	1	1	1	1	0	0	1
1	1	0	1	1	1	0	0	0	1
1	1	0	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	0	1
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	1	0
1	0	1	0	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	0
0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	0	1	1	0
1	0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	1	0	0	1
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	0	0
1	1	0	0	1	1	1	1	1	1
0	1	0	0	1	1	0	0	0	0
1	1	0	0	1	1	1	0	0	0
1	1	0	0	1	0	1	0	0	0
1	1	0	1	1	0	1	0	0	0

Figura 2.2: Matriz de entrada p.

```
p0 = [1,1,1,1,1;1,0,0,0,0;1,1,1,1,0;1,0,0,0,0;1,0,0,0,0;]
p0 = p0(:);
p1 = [1,1,1,1,1;1,0,0,0,0;1,0,0,1,1;1,0,0,0,1;1,1,1,1,1;]
p1 = p1(:);
p2 = [1,0,0,0,1;1,0,0,0,1;1,1,1,1,1;1,0,0,0,1;1,0,0,0,1;]
p2 = p2(:);
p3 = [0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;]
p3 = p3(:);
p4 = [1,0,0,0,0;1,0,0,0,0;1,0,0,0,0;1,0,0,0,0;1,1,1,1,1;]
p4 = p4(:);
p5 = [1,1,1,1,1;1,0,0,0,1;1,0,0,0,1;1,0,0,0,1;1,1,1,1,1;]
p5 = p5(:);
p6 = [1,1,1,1,1;1,0,0,0,1;1,1,1,1,1;1,0,0,0,0;1,0,0,0,0;]
p6 = p6(:);
p7 = [1,1,1,1,1;1,0,0,0,0;1,1,1,1,1;0,0,0,0,1;1,1,1,1,1;]
p7 = p7(:);
p8 = [1,1,1,1,1;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;]
p8 = p8(:);
p9 = [1,1,1,1,1;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;1,1,1,1,0;]
p9 = p9(:);

p = [p1, p2, p3, p4, p5, p6, p7, p8, p9, p0;]
```

Figura 2.3: Codificação da matriz p.

A matriz de saída t corresponde à matriz identidade, ou seja, cada elemento da diagonal é igual a 1 sendo esta um matriz de 10x10, tal como é possível verificar através da Figura 2.4.

1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1

Figura 2.4: Matriz de saída t.

```
inputs = p;
saida = sim(net,inputs)

saida_final = zeros(10,10);
for i = 1:10
    saida_final(saida(:,i) == max(saida(:,i)),i) = 1;
end
```

Figura 2.5: Codificação da matriz de saída t.

2.3 Interpretação da Saída

Após o treino desta rede neuronal para os diferentes caracteres alfabéticos pode-se então analisar a sua saída.

Como é possível observar nas imagens anteriores, a matriz identidade tem o mesmo número de linhas que a matriz de entrada. A seguir ao treino, é suposto a rede neuronal já saber qual o caracter a imprimir na saída t. Para a criação desta rede usou-se a

função *newff*, disponível no MatLab. Optou-se pela mesma pois iremos utilizar uma rede neuronal do tipo *feed-forward*, sendo esta das mais usadas.

Para a realização do treino desta mesma rede usou-se a função *train* que recebe como parâmetros de entrada as matrizes de entrada, de saída e a estrutura da rede.

2.4 Arquitetura da Rede Neuronal

Nas redes *feed-forward* cada camada conecta-se à próxima, porém não existe um caminho de retorno, portanto todas as conexões têm a mesma direção, partindo da camada de entrada para a camada de saída. Estas redes têm a tendência natural de armazenar conhecimento experimental, tal como o cérebro humano.

No treino desta rede neuronal foram usadas 4 funções diferentes, sendo que todas elas apresentam resultados diferentes, uns melhores que outros. Foram realizados 30 testes para cada função, variando entre o número de erros e o número de nós na camada escondida.

Analisando os resultados, pode-se então concluir que a função que fornece resultados mais constantes é a *trainbr*, pois obtemos a melhor taxa de acerto nos diferentes ambientes de teste. Esta função atualiza os valores de peso e polarização de acordo com a otimização *Levenberg-Marquardt*. Ela minimiza uma combinação de erros e pesos quadrados e, em seguida, determina a combinação correta para produzir uma rede que generalize bem. Este processo é chamado de regularização *bayesiana*.

2.5 Taxa de sucesso para 1 bit errado

Através desta instrução de código (ver Figura 2.6), calcula-se a taxa de acerto para 1 bit errado. O ciclo *for* percorre os dez caracteres com o acréscimo de outro ciclo para percorrer a matriz onde é negada a primeira posição antes da saída para cada coluna. Após a saída, volta-se a negar essa mesma posição para passar à próxima e assim sucessivamente.

```
%com 1 erro%
for j=1:col
    for i = 1 : lin
        erro = p(:,j);
        erro(i) = ~erro(i);
        mat_erro = [mat_erro erro];
        mat_target = [mat_target t(:,j)];
    end
end

output = sim(net, mat_erro);

%converter a saida para 0's e 1's%

final_output = round(output);
figure();
plotconfusion(mat_target,final_output);
```

Figura 2.6: Código da taxa de acerto para um bit errado.

2.6 Taxa de sucesso para 2 bits errados

Neste caso, Figura 2.7, foram utilizados 3 ciclos *for*, o primeiro para correr a coluna, o segundo para correr a linha e por fim, o terceiro para negar as duas posições seguintes.

```
%Gerar todas com 2 erros%
mat_erro2 = [];
mat_target2 = [];
for j = 1 : col
    for i = 1 : lin-1
        for k = i+1 :lin
            erro2 = p(:,j);
            erro2(i) = ~erro2(i);
            erro2(k) = ~erro2(k);
            mat_erro2 = [mat_erro2 erro2];
            mat_target2 = [mat_target2 t(:,j)];
        end
    end
end
```

Figura 2.7: Código da taxa de acerto para dois bits errados.

Capítulo 3

Resultados

3.1 Comportamento da Rede

A rede neuronal acertou quase sempre no caracter, quando não se estava a provocar nenhum erro, usando as funções *trainbr* e *trainlm*, a taxa de acerto para 0 erros foi de 100%, mesmo no primeiro treino.

```
%simular para todas as saidas geradas%
output2 = sim (net, mat_erro2);

%formar as saidas a 0 ou 1%
final_output2 = round(output2);
%aplicar o confusion%
figure();
plotconfusion(mat_target2,final_output2);
[c, cm] = confusion(mat_target2,final_output2);

fprintf('Percentagem de classificação correta com 2 erros: %f%%\n', 100*(1-c));
fprintf('Percentagem de classificação incorreta com 2 erros: %f%%\n', 100*c);
```

Figura 3.1: Trecho de código da entrada correta (2 bits errados).

3.2 Tabelas de resultados

Como é possível ver nas tabelas seguintes, foram realizados vários testes, com quatros funções de treino diferentes, *trainlm*, *trainbr*, *traingda* e *trainscg*, pela análise destes resultados pode-se concluir que a *trainbr* foi a mais estável e a que obteve as melhores percentagens de acerto.

9 nós na camada escondida / 0 erros					
Nr. de Testes / Função de treino	1	2	3	4	5
trainlm	100,0%	100,0%	100,0%	100,0%	100,0%
trainbr	100,0%	100,0%	100,0%	100,0%	100,0%
traingda	100,0%	100,0%	100,0%	80,0%	100,0%
trainscg	70,0%	100,0%	100,0%	100,0%	90,0%

Figura 3.2: 9 nós na camada escondida com 0 erros.

9 nós na camada escondida / 1 erros					
Nr. de Testes / Função de treino	1	2	3	4	5
trainlm	93,6%	94,8%	92,0%	90,8%	92,0%
trainbr	97,6%	96,4%	94,8%	94,4%	95,6%
traingda	91,2%	94,4%	94,8%	90,8%	92,8%
trainscg	52,0%	90,8%	91,6%	93,6%	74,8%

Figura 3.3: 9 nós na camada escondida com 1 erro.

9 nós na camada escondida / 2 erros					
Nr. de Testes / Função de treino	1	2	3	4	5
trainlm	79,8%	82,1%	81,6%	75,9%	81,2%
trainbr	90,4%	90,0%	87,6%	87,8%	88,9%
traingda	78,6%	81,8%	84,3%	74,1%	77,9%
trainscg	41,9%	80,7%	81,3%	82,7%	62,0%

Figura 3.4: 9 nós na camada escondida com 2 erros.

11 nós na camada escondida / 0 erros					
Nr. de Testes / Função de treino	1	2	3	4	5
trainlm	100,0%	100,0%	100,0%	100,0%	100,0%
trainbr	100,0%	100,0%	100,0%	100,0%	100,0%
traingda	100,0%	100,0%	100,0%	80,0%	100,0%
trainscg	90,0%	100,0%	100,0%	100,0%	100,0%

Figura 3.5: 11 nós na camada escondida com 0 erros.

11 nós na camada escondida / 1 erro					
Nr. de Testes / Função de treino	1	2	3	4	5
trainlm	92,0%	93,2%	94,0%	95,2%	97,2%
trainbr	96,8%	94,8%	96,8%	96,8%	96,8%
traingda	94,0%	92,4%	92,8%	75,6%	94,4%
trainscg	94,4%	85,2%	96,4%	95,2%	94,0%

Figura 3.6: 11 nós na camada escondida com 1 erro.

11 nós na camada escondida / 2 erros					
Nr. de Testes / Função de treino	1	2	3	4	5
trainlm	81,7%	83,2%	80,1%	81,5%	74,4%
trainbr	91,1%	90,7%	91,5%	91,0%	90,8%
traingda	78,1%	82,4%	82,5%	80,1%	77,4%
trainscg	80,8%	74,5%	88,7%	83,9%	40,4%

Figura 3.7: 11 nós na camada escondida com 2 erros.

Capítulo 4

Conclusão

Este trabalho permitiu-nos ter uma ideia mais clara sobre o funcionamento das redes neuronais e sobre tudo o que envolve um algoritmo de treino e o sobre o que está por de trás de aplicações que fazem uso de *machine learning*. O presente trabalho acabou também por nos deixar mais familiarizados com a ferramenta de trabalho MatLab, o que nos irá certamente ajudar com os restantes trabalhos a realizar.

Como já foi referido anteriormente, a função que apresentou melhores resultados foi a *trainbr*, sendo que a *trainlm* também teve uma performance agradável neste caso. Podemos concluir também que mesmo reduzindo o número de nós na camada escondida as percentagens não diminuíram muito, evitando assim o uso de poder computacional desnecessário.

Assim sendo, basta-nos apenas agradecer a disponibilidade e ajuda por partes dos docentes para a realização deste trabalho.

Apêndice A

Anexos

```
clear all; close all;
p0 = [1,1,1,1,1;1,0,0,0,0;1,1,1,1,0;1,0,0,0,0;1,0,0,0,0;]
p0 = p0(:);
p1 = [1,1,1,1,1;1,0,0,0,0;1,0,0,1,1;1,0,0,0,1;1,1,1,1,1;]
p1 = p1(:);
p2 = [1,0,0,0,1;1,0,0,0,1;1,1,1,1,1;1,0,0,0,1;1,0,0,0,1;]
p2 = p2(:);
p3 = [0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;]
p3 = p3(:);
p4 = [1,0,0,0,0;1,0,0,0,0;1,0,0,0,0;1,0,0,0,0;1,1,1,1,1;]
p4 = p4(:);
p5 = [1,1,1,1,1;1,0,0,0,1;1,0,0,0,1;1,0,0,0,1;1,1,1,1,1;]
p5 = p5(:);
p6 = [1,1,1,1,1;1,0,0,0,1;1,1,1,1,1;1,0,0,0,0;1,0,0,0,0;]
p6 = p6(:);
p7 = [1,1,1,1,1;1,0,0,0,0;1,1,1,1,1;0,0,0,0,1;1,1,1,1,1;]
p7 = p7(:);
p8 = [1,1,1,1,1;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;]
p8 = p8(:);
p9 = [1,1,1,1,1;0,0,1,0,0;0,0,1,0,0;0,0,1,0,0;1,1,1,0,0;]
p9 = p9(:);

p = [p1, p2, p3, p4, p5, p6, p7, p8, p9, p0;]

[M, N] = size(p);
t = eye(10);

net = newff(p,t,9);

%training parameters%
net.trainParam.epochs = 2000;
net.divideParam.trainRatio = 1;
net.divideParam.valRatio = 0;
net.divideParam.testRatio = 0;

for aux = 1: net.numLayers
    net.layers{aux}.transferFcn = 'tansig';
end

net.trainFcn = 'trainscg'; %'traingda';

net = train(net,p,t);

inputs = p;
saida = sim(net,inputs)

saida_final = zeros(10,10);
for i = 1:10
    saida_final(saida(:,i) == max(saida(:,i)),i) = 1;
end

plotconfusion(t , saida_final)
[c, cm] = confusion(t, saida_final)

fprintf('Percentagem de classificação correta : %f%%\n', 100*(1-c));
```

```
fprintf('Percentagem de classificação incorreta : %f%%\n', 100*c);

input_final = [];

for j=1:10
    for i=1:M
        input_error(:,1) = p(:,j);
        input_error(i,1) = ~p(i,j);
        input_final = [input_final input_error];
    end
end

[lin, col] = size(p);

%sem erros%
inputs = p;
mat_erro = [];
mat_target = [];

%com 1 erro%
for j=1:col
    for i = 1 : lin
        erro = p(:,j);
        erro(i) = ~erro(i);
        mat_erro = [mat_erro erro];
        mat_target = [mat_target t(:,j)];
    end
end

output = sim(net, mat_erro);

%converter a saida para 0's e 1's%

final_output = round(output);
figure();
plotconfusion(mat_target,final_output);

[c, cm] = confusion(mat_target,final_output);

fprintf('Percentagem de classificação correta com 1 erro: %f%%\n',
100*(1-c));
fprintf('Percentagem de classificação incorreta com 1 erro: %f%%\n',
100*c);


%Gerar todas com 2 erros%
mat_erro2 = [];
mat_target2 = [];
for j = 1 : col
    for i = 1 : lin-1
        for k = i+1 :lin
            erro2 = p(:,j);
            erro2(i) = ~erro2(i);
            erro2(k) = ~erro2(k);
            mat_erro2 = [mat_erro2 erro2];
```

```
            mat_target2 = [mat_target2 t(:,j)];
        end
    end
end


%simular para todas as saidas geradas%
output2 = sim (net, mat_erro2);

%formar as saidas a 0 ou 1%
final_output2 = round(output2);
%aplicar o confusion%
figure();
plotconfusion(mat_target2,final_output2);
[c, cm] = confusion(mat_target2,final_output2);

fprintf('Percentagem de classificação correta com 2 erros: %f%%\n',
100*(1-c));
fprintf('Percentagem de classificação incorreta com 2 erros: %f%%\n',
100*c);
```