Conhecimento e Raciocínio

João Choupina – 2020151878
Ricardo Tavares - 2021144652

09/05/2024

—

Redes Neuronais

—

Licenciatura em Engenharia Informática

Índice

Introdução

Este relatório descreve o desenvolvimento e análise de um trabalho prático desenvolvido no âmbito da unidade curricular de Conhecimento e Raciocínio, que explora os conceitos de raciocínio baseado em casos e redes neurais. Utilizamos a plataforma MATLAB, especificamente a toolbox Deep Learning, para implementar redes neurais.

O objetivo principal deste trabalho é aplicar os conhecimentos adquiridos nas aulas para explorar e aprofundar os conceitos de raciocínio baseado em casos e redes neurais do tipo Feedforward. Optamos por selecionar o conjunto de dados "stroke" com o intuito de realizar essa aplicação. Adicionalmente desenvolvemos uma interface gráfica básica onde podemos treinar e verificar as redes neuronais.

Decisões tomadas

Deteção e Isolamento de Casos com Valores Ausentes (NaN)

O algoritmo começa por identificar as linhas na tabela principal (*caseLibrary*) que contêm valores ausentes ("NaN") na coluna "stroke", a variável alvo que indica a presença ou ausência de AVC. Essas linhas com dados ausentes são agrupadas em tabelas separadas chamadas *nanCases*. As restantes linhas, sem valores ausentes em "stroke", são mantidas na tabela original *caseLibrary*.

```
formatSpec = '%f%f%f%f%f%f%f%f%f%f%f';
caseLibrary = readtable('Train.csv', 'Delimiter', ',', 'Format', formatSpec);

% Guardando os casos com NaN
nanCases = caseLibrary(isnan(caseLibrary{:, 11}), :);

disp('Casos com NaN:');
disp(nanCases);
```

Figura 1 - Identificadores casos NaN

Casos Semelhantes para Imputação de Valores Ausentes

Para cada linha na tabela *nanCases*, o algoritmo tenta encontrar linhas semelhantes na tabela *caseLibrary*.

A similaridade é calculada usando uma função especializada de nome *retrieve*. Essa função considera nove atributos dos pacientes, cada um com um peso específico:

- Género (peso 1)
- Idade (peso 5)
- Hipertensão (peso 5)
- Doença cardíaca (peso 5)
- Já foi casado (peso 2)
- Tipo de residência (peso 1)
- Nível médio de glicose (peso 4)
- BMI (peso 4)
- Situação de tabagismo (peso 5)

Para cada atributo, a distância entre os valores do paciente na linha *nanCases* e os valores dos pacientes na tabela caseLibrary é calculada.

- Para atributos numéricos (como idade e nível de glicose), a função *linear_distance* é utilizada para calcular a diferença absoluta entre os valores.
- Para o atributo de situação de tabagismo (categórico), a função *smoking_status*_distance é aplicada.

A dissimilaridade final é calculada a partir da soma ponderada das distâncias dos atributos e normalizada para um intervalo entre 0 e 1, onde 0 indica total similaridade e 1 indica total diferença.

```
% Para cada caso na biblioteca de casos é calculada a distância
for i=1:size(caseLibrary,1)
    factorDistances = zeros(1,9);

    factorDistances(1,1) = linear_distance(caseLibrary{i,'gender'}/maxValues('gender'),...
        inputCase.gender / maxValues('gender'));

    factorDistances(1,2) = linear_distance(caseLibrary{i,'age'}/maxValues('age'),...
        inputCase.age / maxValues('age'));

    factorDistances(1,3) = linear_distance(caseLibrary{i,'hypertension'}/maxValues('hypertension'),...
        inputCase.hypertension / maxValues('hypertension'));

    factorDistances(1,4) = linear_distance(caseLibrary{i,'heart_disease'}/maxValues('heart_disease'),...
        inputCase.heart_disease / maxValues('heart_disease'));

    factorDistances(1,5) = linear_distance(caseLibrary{i,'ever_married'}/maxValues('ever_married'),...
        inputCase.ever_married / maxValues('ever_married'));

    factorDistances(1,6) = linear_distance(caseLibrary{i,'Residence_type'}/maxValues('Residence_type'),...
        inputCase.Residence_type / maxValues('Residence_type'));

    factorDistances(1,7) = linear_distance(caseLibrary{i,'avg_glucose_level'}/maxValues('avg_glucose_level'),...
        inputCase.avg_glucose_level / maxValues('avg_glucose_level'));

    factorDistances(1,8) = linear_distance(caseLibrary{i,'bmi'}/maxValues('bmi'),...
        inputCase.bmi / maxValues('bmi'));

    factorDistances(1,9) = smoking_status_distance(caseLibrary{i,'smoking_status'}/maxValues('smoking_status'),...
        inputCase.smoking_status / maxValues('smoking_status'));
```

Figura 2 - Cálculo distância de fatores

```
DG = (factorDistances * factorWeights')/sum(factorWeights);
finalSimilarity = 1 - DG;

if finalSimilarity >= similarityThreshold
    caseIndexes = [caseIndexes i];
    caseSimilarities = [caseSimilarities finalSimilarity];
end
```

Figura 3 - Cálculo final similaridades

Ajuste do Limite de Similaridade e Imputação do Valor Ausente

Se nenhuma linha similar for encontrada com base no limite de dissimilaridade inicial (*similarityThreshold*), o algoritmo não para. Ele diminui gradualmente esse limite até encontrar uma linha na tabela *caseLibrary* que seja suficientemente similar à linha em questão na tabela *nanCases*.

Uma vez que essa linha é encontrada, o valor ausente de "stroke" na linha *nanCases* é preenchido usando o modo (valor mais frequente) do atributo "stroke" nas linhas similares. Essa estratégia garante que o valor imputado seja o mais plausível possível, considerando os pacientes mais semelhantes.

A linha preenchida de *nanCases* é então adicionada de volta à tabela principal *caseLibrary*, reunindo todos os dados com valores completos.

Observações Importantes

O processo de busca por casos similares e imputação de valores ausentes é iterativo, ou seja, é realizado para cada linha em *nanCases* até que todos os valores ausentes sejam preenchidos.

```
for i = 1:size(nanCases, 1)
```

Figura 4 - Iteração sobre casos NaN

O limite de dissimilaridade (*similarityThreshold*) é um parâmetro crucial que influencia a qualidade dos valores imputados. Um valor muito alto pode levar à imputação de valores incorretos, enquanto um valor muito baixo pode resultar em números excessivos de casos sem similaridade suficiente. A escolha do valor ideal pode depender das características dos dados e do objetivo final da análise.

Aplicação

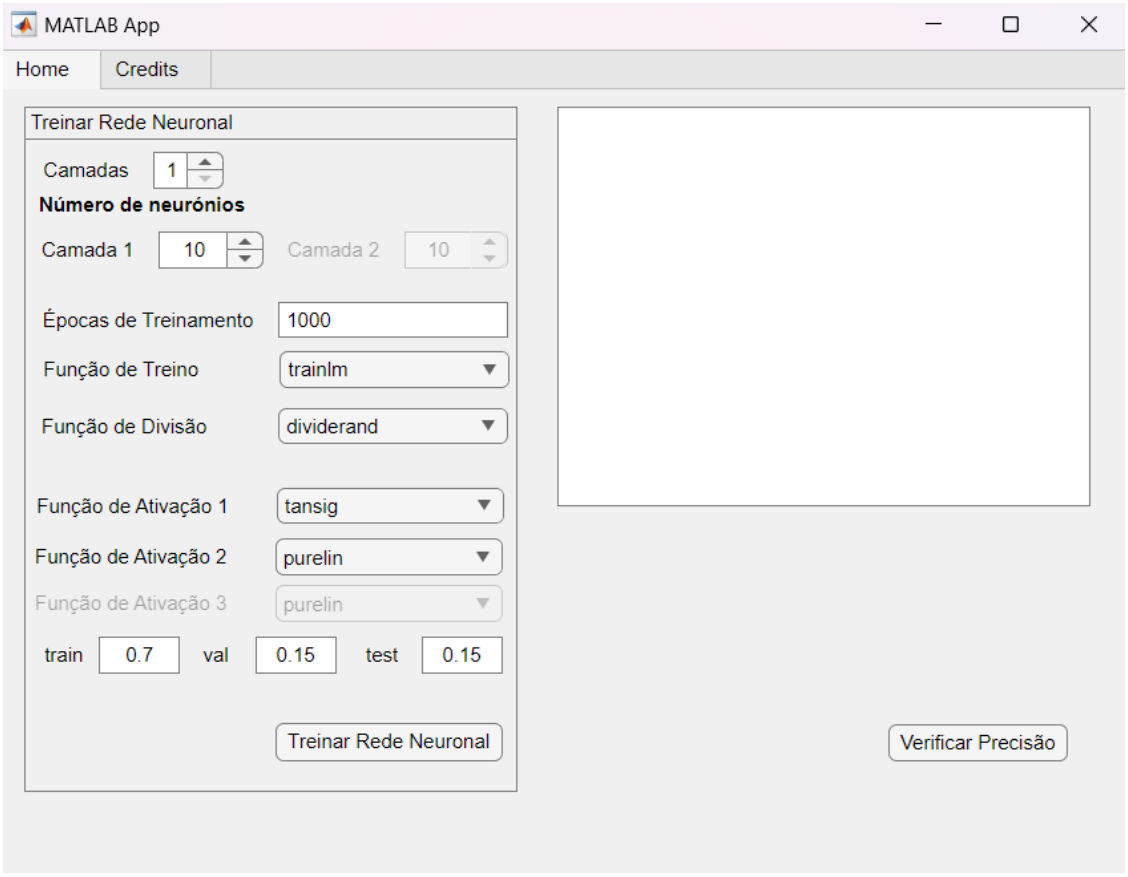

Figura 5 - Interface gráfica

Como previamente referido, o trabalho inclui também uma interface gráfica, que permite treinar uma rede neuronal podendo escolher alguns dos seus parâmetros de entrada, tais como numero de camadas, uma ou duas, o numero de neurónios por camada, épocas de treinamento, funções de treino, divisão e ativação e também os valores de train, val e test.

Além disto, está também preparada para validar todos os dados inseridos de modo a mostrar mensagens de erro ao utilizador e impedir comportamentos inesperados.

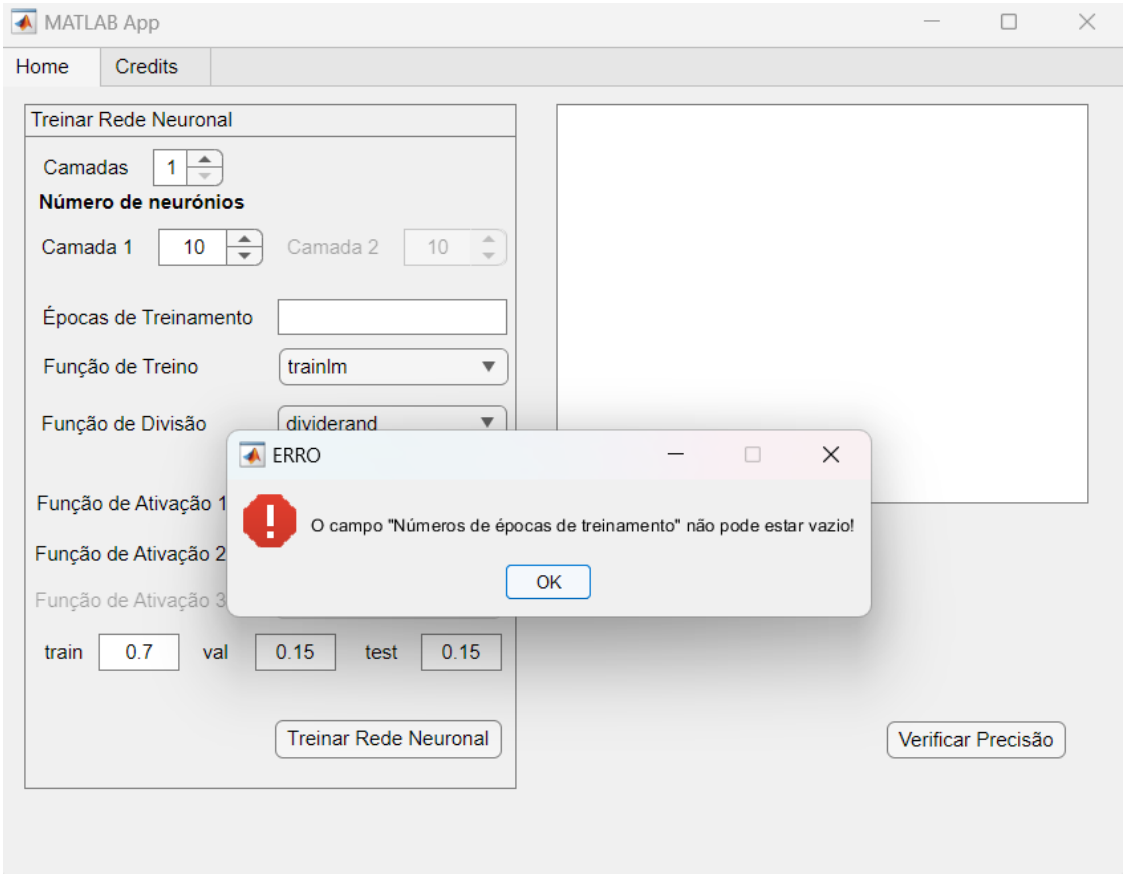

Figura 6 - Interface Gráfica - exemplo de uma mensagem de erro

Como podemos observar neste exemplo onde o número de épocas de treinamento não foi especificado e assim que colocamos a rede a treinar nos aparece este pop-up de erro de forma a melhorar a interação do utilizador com o programa.

Analise de Resultados

Ficheiro Train:

O número e dimensão das camadas encondidas influencia o desempenho?								
Conf1	2	5, 5	tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		77.4262	72.9891
Conf2	2	10,10	tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		78.5164	73.0435
Conf3	5	10,10,10,10,10	tansig,tansig, tansig,tansig tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		77.80833	74.1848
Conf4	6	10,10,10,10,10,10	tansig,tansig, tansig,tansig,tansig,tansig,purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		76.5246	71.7391

Figura 7 - Estudo número de cadamas escondidas

Com base nesses resultados, podemos concluir que o número e dimensão das camadas escondidas influenciam significativamente o desempenho da rede neural. Em geral, um aumento no número de camadas e neurónios pode melhorar o desempenho da rede até certo ponto, como visto nas Conf1 e Conf2. No entanto, adicionar camadas em excesso, como na Conf4, pode levar a uma degradação no desempenho devido a possíveis problemas de overfitting.

A função de treino influencia o desempenho?								
Conf1	1	10	tansig, purelin	traingd	dividerand = {0.7, 0.15, 0.15}		70.2623	68.9130
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}		75.7131	72.3370
Conf3	1	10	tansig, purelin	trainbr	dividerand = {0.7, 0.15, 0.15}		79.4426	73.0978
Conf4	1	10	tansig, purelin	traingdx	dividerand = {0.7, 0.15, 0.15}		75.6721	73.7500

Figura 8 - Estudo função de treino

Com base nesses resultados, podemos concluir que a função de treino influencia significativamente o desempenho da rede neural. Algumas funções de treino, como trainbr, podem resultar em melhorias significativas no desempenho em comparação com outras, como traingd. A escolha da função de treino adequada é crucial para obter o melhor desempenho da rede neural em termos de precisão global e precisão no teste.

As funções de ativação influenciam o desempenho?							
Conf1	1	10	logsig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	76.9672	73.9130
Conf2	1	10	tansig, logsig	trainlm	dividerand = {0.7, 0.15, 0.15}	40.1639	39.8913
Conf3	1	10	hardlim, compet	trainlm	dividerand = {0.7, 0.15, 0.15}	40.1639	41.1413
Conf4	1	10	tansig, softmax	trainlm	dividerand = {0.7, 0.15, 0.15}	40.5978	40.5978
Conf5	1	10	softmax, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	77.8607	73.7500
Conf6	1	10	tribas,purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	76.1557	72.8804

Figura 9 - Estudo função de ativação

Com base nessas observações, podemos concluir que as funções de ativação têm um impacto significativo no desempenho da rede neural. Algumas combinações de funções de ativação, como logsig e purelin na Conf1, podem resultar em desempenho superior, enquanto outras, como tansig e logsig na Conf2, podem levar a resultados muito inferiores. A escolha adequada das funções de ativação é essencial para obter o melhor desempenho da rede neural.

A divisão de exemplos pelos conjuntos influencia o desempenho?							
Conf1	1	10	tansig, purelin	trainlm	dividerand = {0.33, 0.33, 0.33}	74.6148	71.7488
Conf2	1	10	tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	76.8852	71.2903
Conf3	1	10	tansig, purelin	trainlm	dividerand = {0.8, 0.1, 0.1}	77.4508	73.7705
Conf4	1	10	tansig, purelin	trainlm	dividerand = {0.5, 0.25, 0.25}	76.1803	72.4837
Conf5	1	10	tansig, purelin	trainlm	dividerand = {0.6, 0.2, 0.2}	77.2459	72.7049
Conf6	1	10	tansig, purelin	trainlm	dividerand = {0.85, 0.1, 0.05}	77.5328	76.7742

Figura 10 - Estudo train, val e test

Com base nessas observações, podemos concluir que a divisão dos exemplos pelos conjuntos influencia significativamente o desempenho da rede neural. Uma distribuição equilibrada pode resultar em melhores resultados, como observado nas Conf3 e Conf6, enquanto distribuições desequilibradas podem levar a um desempenho inferior, como nas Conf1 e Conf2.

Ficheiro Start:

A função de treino influencia o desempenho?								
Conf1	1	10	tansig, purelin	traingd	dividerand = {0.7, 0.15, 0.15}		100	0	8.2874
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}		100	0	3.2759
Conf3	1	10	tansig, purelin	trainbr	dividerand = {0.7, 0.15, 0.15}		77	0.2300	1.8912
Conf4	1	10	tansig, purelin	traingdx	dividerand = {0.7, 0.15, 0.15}		100	0	3.3724

Figura 11 - Estudo função de treino

Com base nessas observações, podemos concluir que a função de treino tem um impacto significativo no desempenho da rede neural. Algumas funções de treino, como traingd e trainbfg, podem levar a um desempenho perfeito do modelo em conjunto com um tempo de execução relativamente baixo. Outras funções, como trainbr, podem resultar em uma precisão total menor e um tempo de execução mais rápido. A escolha adequada da função de treino depende das características do problema e dos requisitos de desempenho.

As funções de ativação influenciam o desempenho?								
Conf1	1	10	logsig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		100	0	1.6394
Conf2	1	10	tansig, logsig	trainlm	dividerand = {0.7, 0.15, 0.15}		50	0.5	1.7829
Conf3	1	10	hardlim, compet	trainlm	dividerand = {0.7, 0.15, 0.15}		50	0.5	1.5260
Conf4	1	10	tansig, softmax	trainlm	dividerand = {0.7, 0.15, 0.15}		50	0.5	1.7401
Conf5	1	10	softmax, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		100	0	1.7768
Conf6	1	10	tribas,purelin	trainlm	dividerand = {0.7, 0.15, 0.15}		100	0	1.7437

Figura 12 - Estudo função de ativação

Com base nessas observações, podemos concluir que as funções de ativação têm um impacto significativo no desempenho da rede neural. Algumas combinações de funções de ativação, como logsig e purelin na Conf1, podem resultar em desempenho superior, enquanto outras, como tansig e logsig na Conf2, podem levar a resultados muito inferiores. A escolha adequada das funções de ativação é essencial para obter o melhor desempenho da rede neural.

Conclusão

Com base nos resultados deste trabalho, conclui-se que vários fatores influenciam o desempenho das redes neurais em tarefas de classificação. Desde o número de camadas escondidas até a escolha das funções de ativação e de treino, cada especto desempenha um papel crucial. A distribuição equilibrada dos exemplos entre os conjuntos de treino, validação e teste também é essencial. Essas descobertas fornecem *insights* valiosos sobre como otimizar o desempenho das redes neurais em aplicações de classificação, destacando a importância da seleção criteriosa de parâmetros e configurações.