INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA E SISTEMAS

Conhecimento e Raciocínio

Redes Neuronais

João Duarte **Nº 2020122715**
João Santos **Nº 2020136093**

COIMBRA

12 de maio de 2024

Índice

Introdução *3*

Decisões Tomadas *4*

3.3 – Preparação do Dataset *5*

Retrieve **5**

Funções de similariedade local **5**

Função de similariedade global **6**

Pesos dos atributos **6**

3.4 Estudo e análise de redes neuronais *7*

3.4 a) – START **7**

3.4 b) – TRAIN **8**

3.4 c) – TEST **11**

Aplicação Gráfica *12*

Configuração da rede neuronal **13**

Switch + Botão "Procurar Dataset" **14**

Caso único **16**

Menu **17**

Conclusão *18*

Introdução

Este trabalho foi realizado no contexto da disciplina de Conhecimento e Raciocínio, com o propósito de desenvolver e estudar redes neurais para a identificação de padrões, especificamente para categorizar análises sanguíneas.

Utilizou-se a linguagem e tecnologia MATLAB para o desenvolvimento prático, empregando redes neurais do tipo feedforward. Foram realizadas variações em diversos parâmetros para comparação e teste.

Além disso, o trabalho prático inclui uma aplicação gráfica que possibilita a criação, treinamento e simulação de redes neurais em conjuntos de dados específicos ou casos individuais.

Decisões Tomadas

- Para fazer a conversão do target (este podia variar entre 0 e 4) para binário, foi necessário utilizar a função "onehotencode" que converte variáveis categóricas em codificação one-hot. A codificação one-hot é um método de representação de variáveis categóricas onde cada categoria é representada por um vetor binário.

- Para fazer o tratamento do ficheiro "Train.csv", o qual continha valores numéricos e não numéricos (strings), foram realizados os seguintes processos:

    1. Os valores da coluna "Category" ("0 – Blood Donor", "0s – suspect Blood Donor", "1 – Hepatitis", "2 – Fibrosis", "3 – Cirrhosis") seriam redefinidos para "0", "1", "2", "3", "4" respetivamente.

    2. Da coluna "Category", os valores não numéricos ("NA") foram redefinidos para "-1".

    3. Da coluna "Sex", todos os valores ("f", "m") foram redefinidos para "0" e "1" respetivamente.

    4. No fim de todos os processos anteriores, todos os valores foram convertidos de string para valores numéricos com a função "str2double", permitindo assim a formação de uma matriz.

- Para melhorar a eficácia do cálculo da precisão e erro de uma rede neuronal após a simulação sobre o dataset, decidimos fazer uma média de ambas após 50 iterações para a mesma rede (mesmos parâmetros).

- Para normalizar valores do dataset "anormais" decidimos fazer um random entre um valor consideravelmente baixo e alto. Estes valores foram obtidos a partir de várias pesquisas que fizeram perceber qual o intervalo de valores de cada parâmetro. Assim, qualquer valor acima de 1000 será redefinido para um valor random em um intervalo adequado.
  No caso do dataset escolhido 'Hepatitis', apenas o parâmetro "CREA" requeria tal processo.

- Para visualizar o desempenho da rede neuronal, pelo target variar para além de valores 0 e 1, não seria possível utilizar o "plot confusion", então optámos por usar o "confusion chart".

3.3 – Preparação do Dataset

Para o ponto 3.3 do enunciado, usámos a função "readtable" para carregar o dataset, pois este continha valores não númericos ("NA" e outros resultados string na coluna target – Category e "m","f" na coluna "Sex"). Definimos os inputs e target do dataset e após isso, fizemos o processo de converter as strings lidas na coluna Category e Sex para valores numéricos (verificar Decisões Tomadas).

Depois de ter uma tabela de valores numéricos, usámos o "retrieve": Extrai da memória casos semelhantes sobre situações semelhantes ao novo problema. Uma medida de semelhança estima a diferença entre o novo caso e os que estão armazenados. Com isto, conseguimos substituir os valores -1 (antigos "NA") pelo valor do target mais parecido de todos os estudos do dataset (obviamente que apenas usámos o dataset sem valores -1 para conseguir obter o caso mais parecido a um "NA").

Para os valores que ultrapassam os 999, fizemos uma normalização simples de trocar esse valor por um valor random presente em um intervalo adaptado ao parâmetro.

Por fim guardámos toda esta tabela alterada em um novo ficheiro CSV para que este dataset preparado possa ser usado posteriormente.

Retrieve

Funções de similariedade local

- calculate_linear_distance (calcula distância linear): Esta função calcula a diferença absoluta entre dois valores normalizados de atributos (val1 e val2). É usada para atributos como idade e sexo, que se espera que tenham uma relação linear com a similaridade. São constituídos por valores inteiros. Uma distância linear menor indica maior similaridade.

- calculate_euclidean_distance (calcula distância euclidiana): Esta função calcula a distância euclidiana entre dois valores normalizados de atributos. É usada para atributos como ALB, ALP, ALT, etc., que são constituídos por valores decimais. Uma distância euclidiana menor indica maior similaridade.

Função de similariedade global

- Distância Combinada (DG): Esta função calcula uma média ponderada das distâncias de similaridade local para todos os atributos considerados. Os pesos (weighting_factors) determinam a importância relativa de cada atributo na pontuação geral de similaridade. Um valor DG menor indica um caso mais similar.

final_similarity: Esta variável tem o valor do cálculo de uma pontuação de similaridade global subtraindo a distância combinada (DG) de 1. Essa transformação garante que uma pontuação mais alta represente maior similaridade (mais próxima de 1).

Pesos dos atributos

O **vetor weighting_factors** atribui pesos a cada atributo na biblioteca de casos. Esses pesos refletem o conhecimento do especialista no domínio sobre a importância relativa de cada atributo na determinação da similaridade do caso. Por exemplo, um peso maior pode ser dado a atributos considerados mais críticos para decisões de diagnóstico ou tratamento.
Neste caso, foi atribuído o mesmo peso 5, a cada atributo biomédico pois após uma pesquisa, percebemos que os atributos biométricos são sensivelmente críticos.
Ao atributo sexo foi atribuído o peso 1, pois considerámos que este não tinha grande influência na classificação do estudo.
Para o atributo idade foi atribuído o peso 3, que numa escala de 1 a 6 seria um atributo de criticalidade média.

3.4 Estudo e análise de redes neuronais

Perante a realização da tarefa 3.3, foi necessário estudar e analisar o desempenho de redes neuronais.
No 3.4 a) foi necessário criar uma rede, treiná-la com o dataset "Start.csv", simulá-la e depois verificar o seu desempenho (precisão), com targets_previstos_corretos / length(target) * 100.
No 3.4 b) foi necessário criar uma rede, treiná-la e simulá-la, mas usando o dataset "Train.csv" foram guardadas as 3 redes com melhor desempenho depois de várias iterações destes processos.
No 3.4 c) foram usadas as redes previamente guardadas para as testar usando o dataset "Test.csv". Aqui as redes seriam carregadas e simuladas para depois analisar o desempenho destas perante os resultados obtidos vs resultados reais.

Apontamento: Para cada camada de neurónios, a rede terá nºcamadas + 1 funções de ativação (esta última para o target). Para cada exercício, os inputs e o target da rede são obtidos através de:

```
% Carrega o arquivo CSV
data = readmatrix('Dataset1 - Hepatitis/Start.csv', 'Delimiter', ';',
'DecimalSeparator', '.');
inputs = data(:,3:14)'; % inputs: colunas 3 a 14
target = data(:,2)';    % target: coluna 2 - Category
```

3.4 a) – START

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos		Precisão Global	Erro	Tempo
Configuração por defeito	1	10	tansig, purelin	trainlm	dividerand = {1.0, 0.0, 0.0}		100%	0	1.83s

Fig. 1 - Resultados da configuração por defeito - START

A função de treino influencia o desempenho?									
Conf1	1	10	tansig, purelin	traingd	dividerand = {1.0, 0.0, 0.0}		80%	0.0835	1.53s
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {1.0, 0.0, 0.0}		100%	0	3.75s
Conf3	1	10	tansig, purelin	trainscg	dividerand = {1.0, 0.0, 0.0}		100%	0	2.20s
Conf4	1	10	tansig, purelin	traincgf	dividerand = {1.0, 0.0, 0.0}		100%	0	1.72s
Conf5	1	10	tansig, purelin	traingdx	dividerand = {1.0, 0.0, 0.0}		100%	0	2.33s

Fig. 2 - Resultados em função da variação da função de treino - START

Sobre a tabela com resultados da variação da função de treino, conseguimos perceber que para o dataset "Start.csv" não há grande influência nos resultados, pelo que foi obtida 100% precisão em 4 das 5 funções de treino (para além da default). Conseguimos perceber que o tempo de execução varia de função para função mas com valores relativamente próximos entre eles.

As funções de ativação influenciam o desempenho?								
Conf1	1	10	poslin,softmax	trainlm	dividerand = {1.0, 0.0, 0.0}	90%	0.2175	1.20s
Conf2	1	10	netinv,hardlim	trainlm	dividerand = {1.0, 0.0, 0.0}	30%	0.4850	0.64s
Conf3	1	10	tribas, radbas	trainlm	dividerand = {1.0, 0.0, 0.0}	20%	0.2490	5.78s
Conf4	1	10	compet,hardlims	trainlm	dividerand = {1.0, 0.0, 0.0}	40%	0.4800	0.53s
Conf5	1	10	logsig,radbasn	trainlm	dividerand = {1.0, 0.0, 0.0}	90%	0.2163	1.61s
Conf6	1	10	poslin,purelin	trainlm	dividerand = {1.0, 0.0, 0.0}	100%	0	0.77s

Fig. 3 - Resultados em função da variação das funções de ativação - START

Para a tabela de resultados da variação das funções de ativação percebemos uma grande influência no valor da precisão. Dependendo das combinações de funções de ativação podemos ter valores entre os 20% e os 100%, isto porque há funções de ativação mais compativeis com os dados do dataset e até compativeis entre si.
Em geral, as configurações que utilizam as funções de ativação **poslin** e **logsig** obtêm melhor precisão.
Usar uma porcentagem alta de dados para treinamento (por exemplo, 100%) pode levar a uma precisão de treinamento mais baixa, mas pode resultar em melhor desempenho na simulação.

3.4 b) – TRAIN

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Erro	Precisão Teste	Erro Teste
Configuração por defeito	1	10	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	94.6469%	0.0182	91.5824%	0.0283

Fig. 4 - Resultados da configuração por defeito - TRAIN

A arquitetura influencia o desempenho?									
Conf1	1	5	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	93.9901%	0.0197	91.6044%	0.0284
Conf2	2	5	tansig, purelin, poslin	trainlm	dividerand = {0.7, 0.15, 0.15}	89.2046%	0.2233	88.2857%	0.2247
Conf3	3	5	tansig, purelin, poslin,softmax	trainlm	dividerand = {0.7, 0.15, 0.15}	90.1518%	0.2194	88.9670%	0.2217
Conf4	2	10	tansig, purelin, poslin	trainlm	dividerand = {0.7, 0.15, 0.15}	90.1419%	0.2177	89.8242%	0.2194
Conf5	3	10	tansig, purelin,poslin,softmax	trainlm	dividerand = {0.7, 0.15, 0.15}	91.1980%	0.2174	90.5055%	0.2186
Conf6	1	20	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	95.1056%	0.0179	91.8462%	0.0293
Conf7	2	25	tansig, purelin,poslin	trainlm	dividerand = {0.7, 0.15, 0.15}	91.2706%	0.2213	89.5824%	0.2280

Fig. 5 - Resultados em função da variação da arquitetura - TRAIN

As funções de ativação parecem ter pouco impacto significativo na precisão da rede neural. Em geral, as configurações que utilizam as funções de ativação **tansig** e **purelin** obtêm melhor precisão.

Isto ocorre devido à complexidade dos dados do dataset. Por serem dados muito simples, qualquer rede "decente" mesmo com apenas 1 camada de neurónios consegue processá-los e ter uma boa precisão, de vez em quando melhor que redes com mais camadas e neurónios.

A função de treino influencia o desempenho?									
Conf1	1	10	tansig, purelin	traingd	dividerand = {0.7, 0.15, 0.15}	87.6502%	0.0581	87.3407%	0.0545
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}	90.9637%	0.0291	89.5604%	0.0369
Conf3	1	10	tansig, purelin	trainscg	dividerand = {0.7, 0.15, 0.15}	91.3795%	0.0270	89.1648%	0.0381
Conf4	1	10	tansig, purelin	traincgf	dividerand = {0.70, 0.15, 0.15}	91.4323%	0.0274	89.8901%	0.0368
Conf5	1	10	tansig, purelin	traingdx	dividerand = {0.7, 0.15, 0.15}	90.8647%	0.0297	89.6703%	0.0378

Fig. 6 - Resultados em função da variação da função de treino - TRAIN

A variação das funções de treino também parecem ter pouco impacto na precisão da rede. A função que obteve o menor resultado foi a **traingd**.
Se o conjunto de dados é relativamente simples e bem estruturado, as diferenças nas funções de treinamento podem não ter um impacto significativo na precisão. Em conjuntos de dados menores e mais simples, as redes neurais podem ser capazes de aprender facilmente padrões mesmo com uma função de treinamento básica.

As funções de ativação influenciam o desempenho?									
Conf1	1	10	poslin,softmax	trainlm	dividerand = {0.70, 0.15, 0.15}	92.9241%	0.2139	90.6154%	0.2180
Conf2	1	10	netinv,hardlim	trainlm	dividerand = {0.70, 0.15, 0.15}	44.2442%	0.5185	43.8681%	0.5170
Conf3	1	10	tribas, radbas	trainlm	dividerand = {0.70, 0.15, 0.15}	87.8812%	0.2134	87.2967%	0.2162
Conf4	1	10	compet,hardlims	trainlm	dividerand = {0.70, 0.15, 0.15}	47.7162%	0.4981	48.0879%	0.4970
Conf5	1	10	logsig,radbasn	trainlm	dividerand = {0.70, 0.15, 0.15}	85.9604%	0.2253	83.6264%	0.2294
Conf6	1	10	poslin,purelin	trainlm	dividerand = {0.70, 0.15, 0.15}	94.2013%	0.0205	91.4945%	0.0318

Fig. 7 - Resultados em função da variação das funções de ativação - TRAIN

A variação das funções de ativação têm em parte influência na precisão da rede neuronal como podemos analisar na tabela acima. Podemos verificar que netinv + hardlim e compet + hardlims têm as piores precisões, enquanto que as outras combinações têm uma precisão relativamente próxima.
Algumas funções de ativação são mais eficientes perante certo tipos de dados e/ou têm comportamentos diferentes de acordo com os padrões de dados.

A segmentação influencia o desempenho?									
Conf1	1	10	tansig, purelin	trainlm	dividerand = {0.8, 0.1, 0.1}	95.1815%	0.0169	91.5082%	0.0326
Conf2	1	10	tansig, purelin	trainlm	dividerand = {0.5, 0.25, 0.25}	93.6172%	0.0225	91.3947%	0.0310
Conf3	1	10	tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	95.8350%	0.0179	90.5333%	0.1044
Conf4	1	10	tansig, purelin	trainlm	dividerand = {0.4, 0.3, 0.3}	92.9109%	0.0254	91.4615%	0.0320
Conf5	1	10	tansig, purelin	trainlm	dividerand = {0.2, 0.6, 0.2}	91.2343%	0.0339	90.0496%	0.0379

Fig. 8 - Resultados em função da variação da segmentação - TRAIN

Também se concluiu que uma variação de segmentação influencia pouco a precisão final da rede neuronal:

- Estabilidade do modelo: Se os modelos são treinados de maneira estável e convergem para soluções semelhantes, independentemente da segmentação dos dados, isso pode resultar em resultados consistentes em termos de precisão.

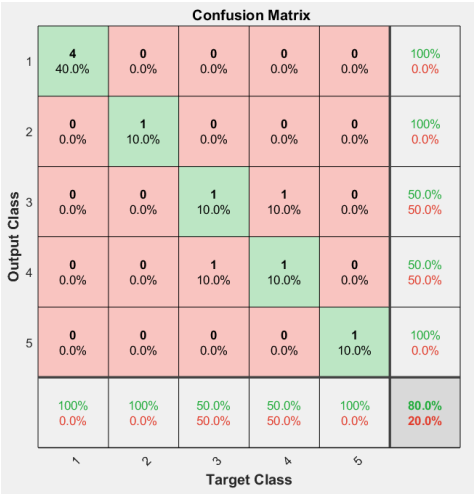
- Homogeneidade dos dados: Se os dados são relativamente homogêneos e representativos do problema que a rede neural está a tentar resolver, diferentes segmentações dos dados podem levar a resultados semelhantes em termos de precisão.

3.4 c) – TEST

Usando uma rede que teve uma precisão global de 98.6799% no treino para simulação no dataset "Test.csv", podemos perceber que esta teve uma precisão de 80% (acertou 8 de 10 resultados).

Isto deve-se ao facto de a rede estar a ser simulada para dados nunca antes "vistos" pela rede neuronal.

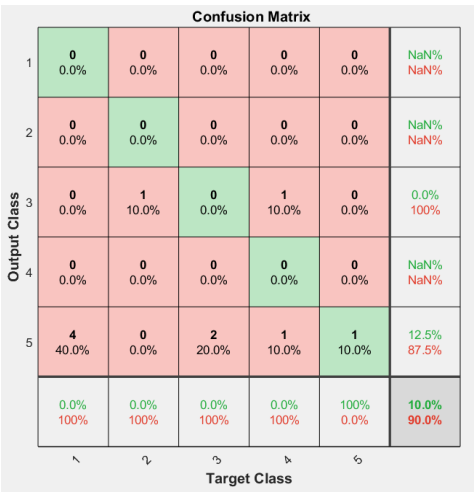
```
Precisão total (nos 10 exemplos): 80.0000%
Erro: 0.6208
```



Se por outro lado usármos uma rede com uma precisão global de 4.6205% no treino no dataset "Test.csv", obtemos uma precisão de 10% (1 target certos de 10).

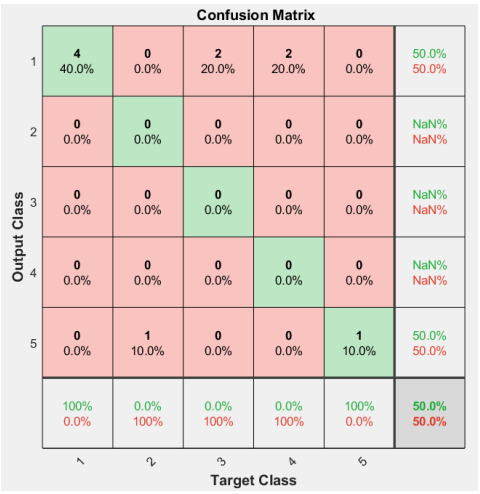
Valores "normais" de acordo com a justificação dada em cima.

```
Precisão total (nos 10 exemplos): 10.0000%
Erro: 0.3800
```



Com uma rede de 90.9241% de precisão de treino a rede consegue 50% de precisão com o dataset "Test.csv". 50% está entre 80% (rede de 98.6799%) e 10% (rede de 4.6205%).

```
Precisão total (nos 10 exemplos): 50.0000%
Erro: 0.1245
```



Conclusão: Como os dados do dataset "Test.csv" contêm valores e estudos que não estiveram presentes no treino da rede, as precisões destas perante este dataset fazem sentido, e será muito dificil obter uma precisão de 100%.

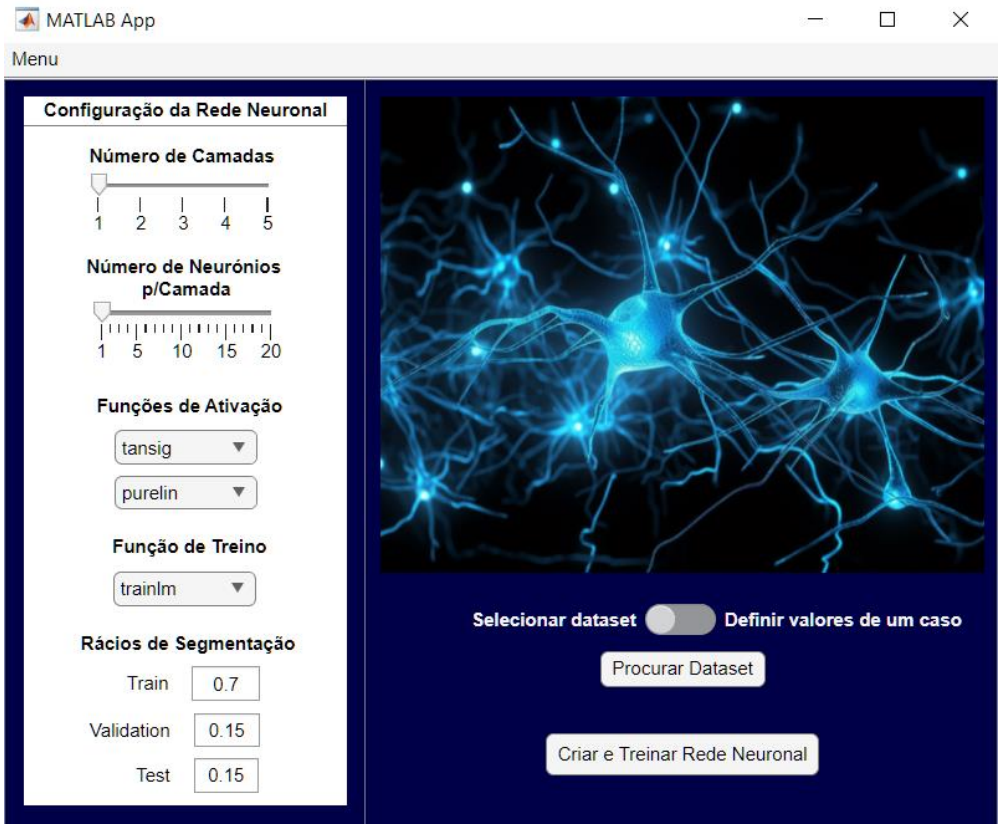
Aplicação Gráfica

Na interface que criámos, colocámos botões, dropdowns, sliders, input fields e switches, necessários para a implementação correta do que era pedido no enunciado.

A aplicação oferece várias funcionalidades:

- É possível carregar uma rede neuronal para avaliar o seu desempenho no dataset escolhido ou para um caso único especificado. *
- É possível criar a partir de valores definidos nos dropdowns, sliders e nos input fields e treinar uma nova rede neuronal a partir de um dataset escolhido. **

** a escolha entre dataset ou definir um caso único específico é feita através de um switch.*
*** a primeira função de ativação será igual para todas as camadas da rede neuronal, e a segunda será usada para o target.*



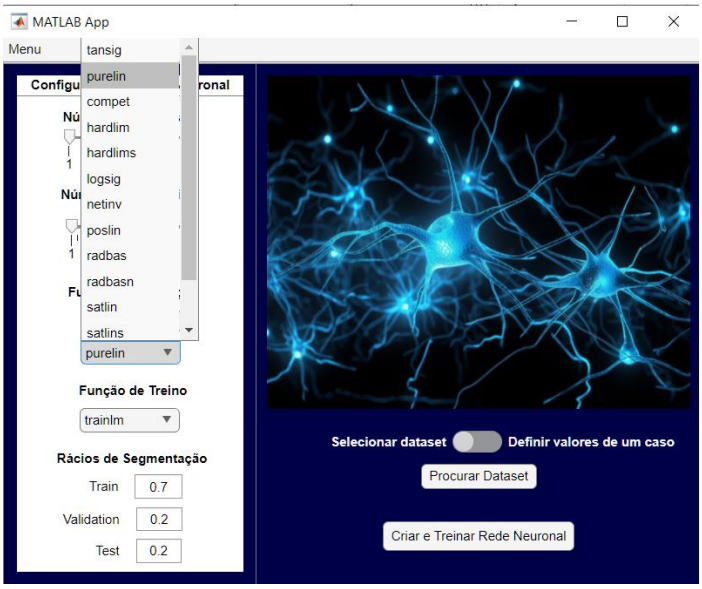
Configuração da rede neuronal

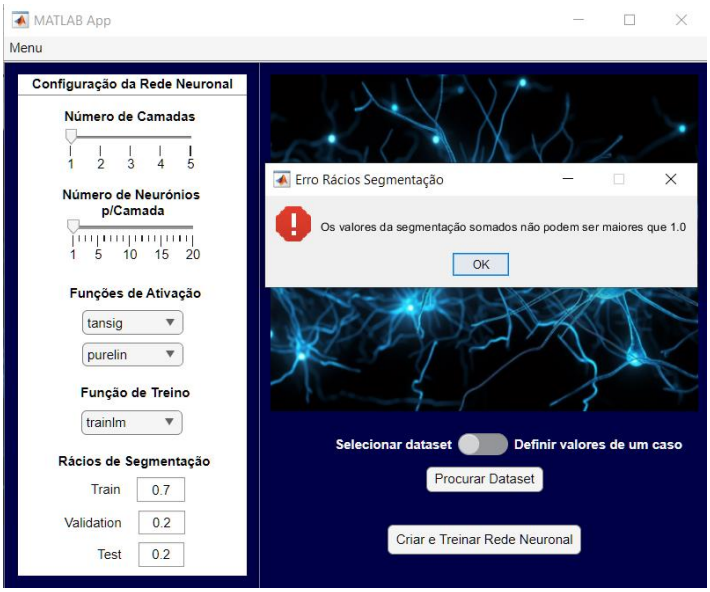
O slider do número de camadas, permite, como o nome indica, definir o número de camadas da futura rede neuronal (mínimo 1 camada, máximo 5 camadas).

O slider do número de neurónios por camada, permite definir o número de neurónios que cada camada irá ter (mínimo 1, máximo 20).

Os três dropdowns permitem definir as funções de ativação e a função de treino. Neste caso, a primeira função de ativação será usada para todas as camdas e a segunda será usada para o target.

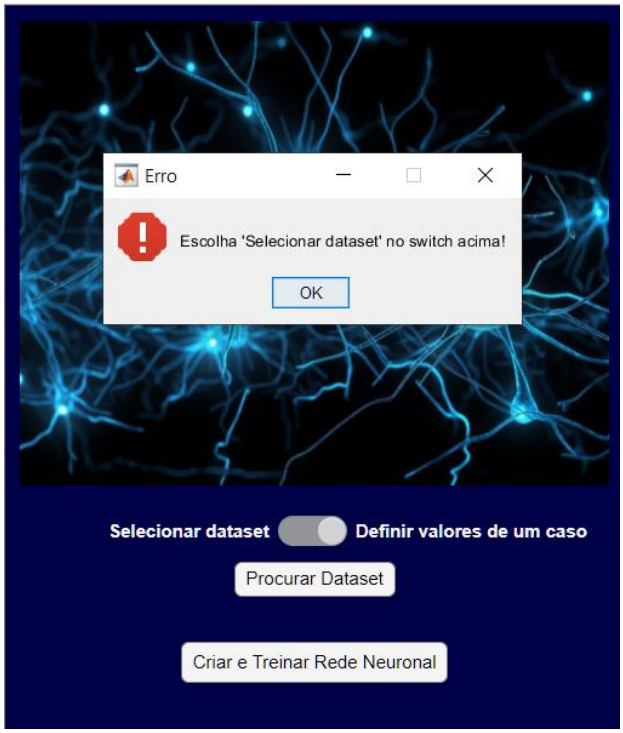
Os três input fields, permitem ao utilizador definir os valores dos rácios de segmentação (não sendo possível a soma dos três ser maior que 1).

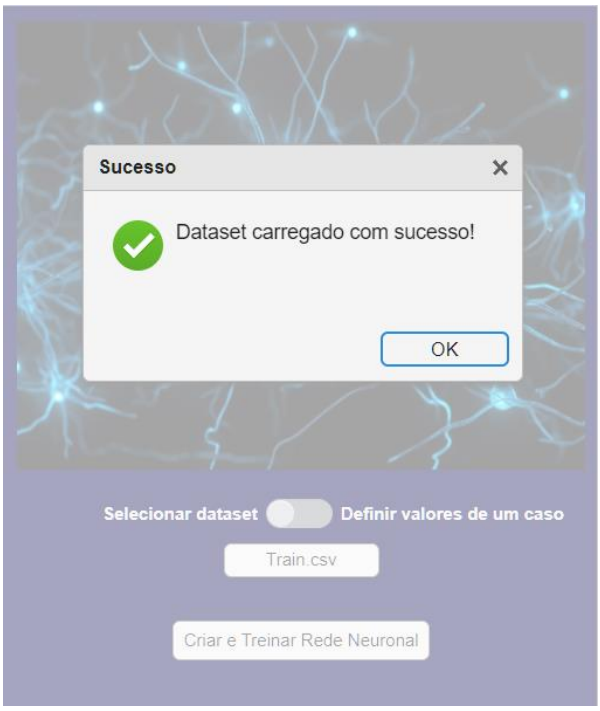




Switch + Botão "Procurar Dataset"

O switch permite o utilizador escolher entre selecionar um dataset guardado ou definir um caso único, sendo que se este último estiver selecionado, não será permitido procurar um dataset através do botão correspondente.

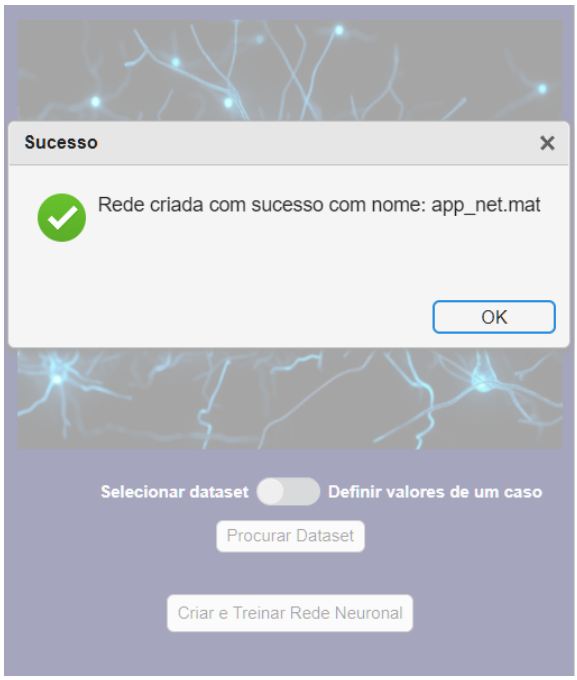




Caso o switch esteja do lado de "Selecionar dataset" e no fim de selecionar o ficheiro CSV, surgirá um popup com uma mensagem de sucesso e o texto de botão mudará para o nome do ficheiro selecionado.

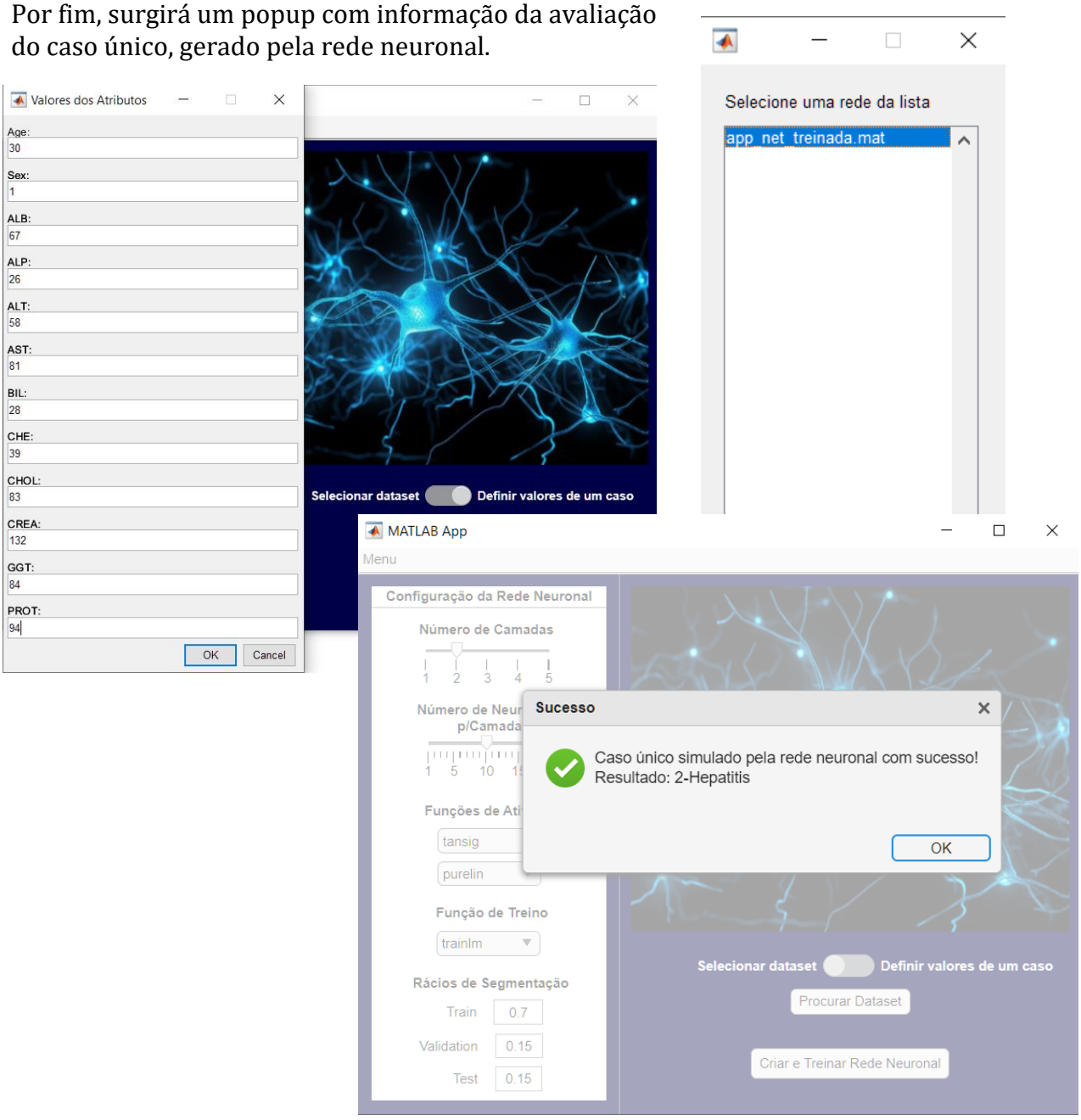
Caso o ficheiro CSV selecionado seja inválido, surgirá um popup com uma mensagem de erro, "forçando" o utilizador a escolher um outro. O texto do botão voltará a "Procurar Dataset".

Caso o ficheiro CSV e os parâmetros da configuração da rede sejam válidos, a rede será criada, treinada e guardada no diretório do programa.

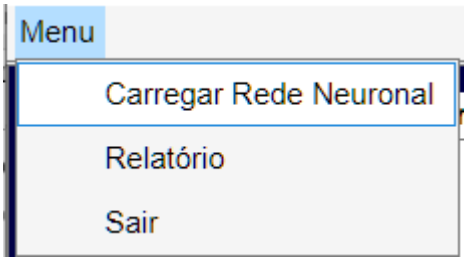


Caso único

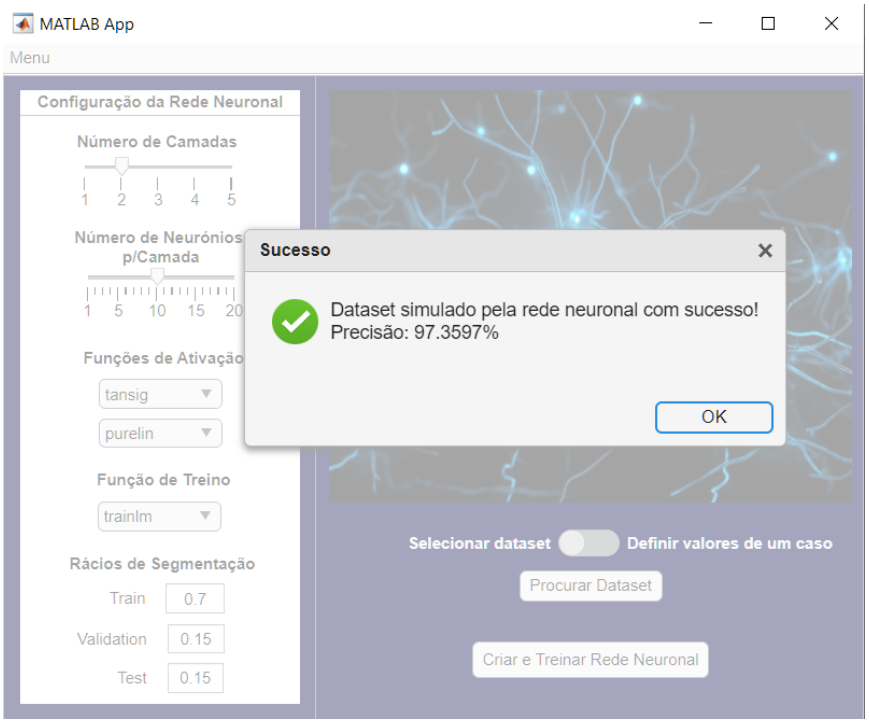
Caso o utilizador escolha definir valores de um caso único, surgirá uma janela com vários inputs para o utilizador. Depois de selecionar os valores, surgirá uma nova janela para que seja possível escolher a rede neuronal (previamente guardada) que simulará aquele caso único.
Por fim, surgirá um popup com informação da avaliação do caso único, gerado pela rede neuronal.



Menu



O menu permite carregar uuma rede reuronal e, se previamente tiver sido escolhido um dataset, a rede realizará a simulação e mostrará o seu desempenho. Para utilizar uma rede previamente guardada para um caso único, ver "Caso Único".



Conclusão

Com este trabalho, foi possível adquirir bastantes conceitos relativos à temática das redes neuronais e à linguagem de programação Matlab (inclusive no desenvolvimento de apps).
Verificamos que as redes neuronais têm um potencial muito grande a nível de processamento lógico.

No entanto, este processamento envolve muitos recursos e por isso, quem as prepara tem de ter o cuidado de gerir recursos de modo a não afetar a qualidade de análise.

Com isto, podemos concluir que as redes neuronais são idealmente desenvolvidas para ajudar a resolver problemas complexos em diversas situações da vida real ,como é o caso aqui de identificar caracteres.

Opinião: As melhores funções de ativação são a combinação de **tansig** e **purelin** para este tipo de dados ; no caso de dados do dataset muito simples, a configuração da rede, a função de treino e a segmentação não terão grande influência no desempenho da rede neuronal.