Instituto Superior
de Engenharia
Politécnico de Coimbra

Conhecimento Raciocínio
2023-2024

Casos de Uso e Redes neuronais

Trabalho Realizado:
Rodrigo Reis, 2022137090
João Pais, 2020131717

Índice

Introdução

No âmbito da unidade curricular de Conhecimento e Raciocínio, do 2º ano da Licenciatura em Engenharia Informática, foi proposto um trabalho prático com o objetivo de explorar e aprofundar os conceitos de raciocínio baseado em casos de uso e de redes neuronais. O trabalho envolve a implementação de um sistema utilizando o MATLAB, nomeadamente a toolbox Deep Learning para redes neuronais.

Escolha e Descrição do Dataset

O dataset escolhido pelo nosso grupo foi o "Stroke", que contém informações sobre pacientes e a ocorrência ou não de Acidente Vascular Cerebral (AVC). Este dataset possui 11 atributos descritos:

- **ID:** Identificador do caso.
- **Gender:** Género do paciente.
- **Age:** Idade do paciente.
- **Hypertension:** Valor binário que determina se o paciente sofre de hipertensão (1) ou não (0).
- **Heart_Disease:** Valor binário que determina se o paciente sofre de doença cardíaca (1) ou não (0).
- **Ever_Married:** Campo que determina se o paciente já foi casado (‘Yes’) ou não (‘No’).
- **Residence_Type:** Campo que descreve o tipo de residência do paciente como ‘Rural’ ou ‘Urban’.
- **AVG_Glucose:** Nível médio de glicose.
- **BMI:** Valor do índice de massa corporal.
- **Smoking_Status:** Campo que descreve se o paciente é fumador (‘smokes’), se foi fumador (‘formerly smoked’), se nunca fumou (‘never smoked’) ou se essa informação não é conhecida (‘Unknown’).
- **Stroke:** Campo que indica se é provável vir a ter um AVC (1) ou pouco provável (0).

E é composto por 3 ficheiros:

- **Start:** Contém 10 casos que não requerem preparação prévia. Está balanceado com 50% dos casos na classe ‘stroke = 0’ e 50% na classe ‘stroke = 1’.
- **Train:** Inclui 610 casos que necessitam de análise e preparação, contendo valores em falta (NA) na coluna ‘stroke’. Está balanceado com 60% dos casos na classe ‘stroke = 0’ e 40% na classe ‘stroke = 1’.
- **Test:** Contém 10 casos para testar a capacidade de generalização das redes neuronais treinadas. Está balanceado com 50% dos casos na classe ‘stroke = 0’ e 50% na classe ‘stroke = 1’.

Preparação do Dataset

3.3 a) Conversão de Atributos

Na alínea 3.3 a) nós começamos a preparar o dataset "Train" para ser utilizado. Os atributos do tipo texto foram convertidos para valores numéricos utilizando o script "converte_dados_stroke.m". Este script realiza as seguintes conversões:

- Género: Convertido de texto para numérico, onde 'Male' = 0 e 'Female' = 1.
- Estado Civil (Ever Married): Convertido para booleano, onde 'Yes' = 1 e 'No' = 0.
- Tipo de Residência: Transformado em booleano, com 'Urban' = 1 e 'Rural' = 0.
- Estado de Fumador: Codificado como numérico, atribuindo valores distintos para diferentes estados (0 para 'never smoked', 1 para 'formerly smoked', 2 para 'smokes', e 3 para 'Unknown').

3.3 b) Identificação do Atributo Alvo

O atributo identificado como sendo o alvo (target) foi a coluna "stroke", que indica se é provável o paciente vir a sofrer ou não um AVC. Esta coluna possui alguns valores em falta, identificados como "NA".

3.3 c) Preenchimento de Valores em Falta

Esta tarefa foi executada através dos scripts 'split.m' e 'retrieve.m'.

Split.m

O script 'split.m' executa as seguintes operações sobre o conjunto de dados do arquivo 'train.csv':

1. Carregamento de Dados: Inicialmente, o script carrega todos os dados disponíveis no arquivo 'train.csv'.
2. Separação de Dados: Os dados são então divididos em dois conjuntos:
    - Dados Completos: Linhas que contêm todos os valores necessários, sem qualquer valor ausente (NA).
    - Dados Incompletos: Linhas que possuem um ou mais valores ausentes (NA).
3. Processamento de Dados Incompletos: Para cada linha com dados incompletos, o script realiza as seguintes etapas:
    - Utiliza um ciclo 'for' para percorrer cada caso de linha incompleta.
    - Recuperação de Similaridade: Invoca a função 'retrieve', que identifica o caso mais semelhante no conjunto de dados que não possui valores ausentes. Esta função retorna o índice do caso semelhante e a medida de similaridade correspondente.
    - Substituição de Valores Ausentes: O valor ausente (NA) na linha incompleta é substituído pelo valor correspondente do caso mais semelhante identificado.
4. Unificação de Dados: Após o processamento de todas as linhas incompletas, os dados agora completos são reunidos com o conjunto inicial de dados completos, formando um conjunto de dados unificado e sem valores ausentes.

retrieve.m

A função retrieve é projetada para identificar o caso mais semelhante a um novo caso fornecido, dentro de uma biblioteca de casos preexistentes. A função utiliza várias métricas de similaridade e ponderações específicas para cada atributo, a fim de calcular a semelhança geral entre casos.

Os pesos dos atributos foram ajustados com base numa breve pesquisa no Chat GPT e posterior consideração. Fatores como a Idade, Hipertensão e Doenças Cardíacas receberam pesos mais altos por serem fortemente ligados ao risco de AVC, sendo a idade considerada mais impactante do que os outros dois. Atributos como Estado Civil, Género e Tipo de Residência receberam pesos menores, refletindo a sua influência menos direta no risco de AVC. O peso de Status de Fumador e do nível médio de Glicose reflete o facto de estes fatores terem um impacto significativo, ainda assim menor que a Hipertensão e as Doenças Cardíacas. Pela mesma ordem de ideias, o IMC (BMI), embora sendo considerado relevante, mas ligeiramente menos impactante que Status de Fumador e o Nível Médio de Glicose.

	Gender	Age	Hypertension	Heart_Disease	Ever_Married	Residence_Type	Avg_Glucose_Level	BMI	Smoking_Status
Pesos	0.2	0.8	0.7	0.7	0.2	0.2	0.6	0.5	0.6

Para a função de similaridade global, diferentes métodos de cálculo de distância são aplicados com base no tipo de dados de cada atributo. Distâncias Lineares são utilizadas para atributos binários ou categoricos (‘gender’, ‘age’, ‘hypertension’, ‘heart_disease’, ‘ever_married’, ‘Residence_type’), porque estes representam estados claramente definidos (ex.: sim ou não), assim, a distância linear é eficaz para captar a discrepância direta entre tais estados, adequando-se à simplicidade desses dados. Para atributos mais complexos como ‘avg_glucose_level’ e ‘bmi’ foi usada a distância euclidiana, visto estes apresentarem uma gama mais ampla de valores e variações.

No caso do atributo "smoking_status", foi utilizada uma função de similaridade local personalizada, definida por uma matriz de similaridades. Esta matriz captura as semelhanças entre as diferentes categorias deste atributo, como "Never smoked", "formerly smoked", "Smokes" e "Unknown". A matriz foi definida manualmente com base em conhecimento de domínio geral, atribuindo maiores semelhanças a categorias mais próximas conceitualmente. (a melhorar)

	Never Smoked	Formerly Smoked	Smokes	Unknwon
Never Smoked	1.0	0.3	0.0	0.5
Formerly Smoked	0.3	1.0	0.5	0.4
Smokes	0.0	0.5	1.0	0.2
Unknwon	0.5	0.4	0.2	1.0

A distância total ponderada (‘DG’) é calculada como o produto das distancias pelo vetor de pesos, dividido pela soma dos pesos. A similaridade final é então calculada como ‘1 - DG’, transformando a distância em uma medida de similaridade (onde 0 indica sem similaridade e 1 indica identidade completa).

```
DG = (distances * weighting_factors')/sum(weighting_factors);
final_similarity = 1-DG;
```

Por fim, a cada iteração é comparada a similaridade final do caso em avaliação com a maior similaridade final até ao momento, assim, no fim da execução da função ‘retrieve’ é devolvido apenas o índice do caso mais similar.

Estudo e Análise das Redes Neuronais

Nas 3 fases seguintes (alíneas a, b e c), dividimos os datasets em dois, os ‘inputs’ (entradas) e os ’targets' (alvos). Como target, utilizamos a coluna ‘stroke’ como referido anteriormente. Para os ‘inputs’, incluímos todos os campos restantes, excluindo o ‘id’, que foi descartado por não ter qualquer correlação com o resultado final (AVC).

3.4 a) Uso do Dataset Start

Nos testes realizados utilizando o dataset "Start", observámos uma precisão bastante elevada. Contudo, esta alta precisão deve-se à falta de segmentação, o que implica que o sistema apenas testa exemplos previamente vistos durante o treino da rede. Adicionalmente, o dataset "Start" contém um número reduzido de exemplos, contribuindo para uma precisão elevada que pode ser considerada enganosa. Por exemplo, quando testámos a mesma rede com o dataset "Test.csv", que inclui exemplos não utilizados no treino, a precisão alcançada foi de apenas 52,03%. Em contraste, uma rede treinada com o dataset "Train.csv" e testada posteriormente com o dataset "Test.csv" alcançou uma precisão de 75% utilizando a mesma configuração.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Precisão Global	Tempo Global
Configuração por defeito	1	10	tansig, purelin	trainlm	100.00	0.0735

A função de treino/ativação influencia o desempenho?

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Precisão Global	Tempo Global
Conf1	1	10	tansig, purelin	traingd	99.86	0.0014
Conf2	1	10	tansig, purelin	trainbfg	100.00	0.0001
Conf3	1	10	logsig, purelin	trainlm	100.00	0.0001
Conf4	1	10	hardlim, hardlims	trainlm	50.00	0.5000

3.4 b) Uso do Dataset Train

Os valores estudados neste capítulo são a média do resultado de 30 treinos. Isto é, o treino foi realizado repetindo cada configuração 30 vezes, no final das iterações registámos a média dos resultados para as seguintes métricas: precisão global, precisão de teste, tempo global e tempo de teste.

Durante cada execução, após o treino da rede com a matriz de 'inputs' e 'targets', normalizámos os 'outputs' para garantir que os resultados fossem consistentes. Isso foi feito ajustando o output no intervalo 0 a 1 usando a função 'mapminmax' do MATLAB. Posteriormente, aplicámos um limiar de 0.5um para determinar a classificação final como 1 (AVC) ou 0 (~AVC). Esta normalização é essencial porque as funções de ativação na camada de saída podem produzir outputs em escalas diferentes. Por exemplo, funções com 'logsig' ou 'softmax' já produzem resultados normalizados entre 0 e 1, enquanto 'purelin' gera resultados numa escala linear. Sem a normalização, aplicar um limiar de 0.5um diretamente aos outputs de 'purelin' (e outras funções semelhantes) poderia levar a classificações incorretas devido à variação mais ampla nos valores de output.

Para medir o erro, utilizamos o erro quadrático médio (MSE - Mean Squared Error). O MSE é calculado como a média dos quadrados das diferenças entre os valores alvo e os valores previstos.

Na tabela abaixo temos os resultados da configuração por defeito, que vão servir como principal comparação nos próximos testes.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Melhor Rede
Configuração por defeito	1	10	tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	85.58	85.39	0.1443	0.1461	1

Influência da quantidade de neurônios

Podemos observar que o aumento do número de camadas e neurônios tende a melhorar a precisão global dos modelos. As configurações testadas variam de redes com 2 camadas até redes com cinco camadas, aumentando progressivamente o número de neurônios de 5 para 10 em cada camada. As redes mais densas conseguem capturar mais detalhes nos dados, melhorando a capacidade de generalização do modelo quando aplicado a novos dados no teste.

Observamos também, que o tempo global e de teste tende a diminuir com o aumento do número de camadas e neurônios, o que pode indicar que as redes mais complexas convergem mais rapidamente, possivelmente devido a uma representação mais robusta das características dos dados aprendidas durante o treino.

Como é possível notar na tabela (nº x) abaixo, a melhor configuração é a 6ª, o que suporta as afirmações anteriores.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Melhor Rede
Conf1	2	5, 5	tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	85.52	85.25	0.1449	0.1475	2
Conf2	2	10,10	tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	86.34	85.88	0.1367	0.1412	3
Conf3	3	5,10,5	tansig, tansig,tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	85.56	85.49	0.1451	0.1451	4
Conf4	3	10,10,10	tansig, tansig,tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	86.24	86.25	0.1380	0.1375	5
Conf5	4	10,10,10,10	tansig, tansig,tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	85.63	85.04	0.1422	0.1496	6
Conf6	5	10,10,10,10,10	tansig, tansig,tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	86.56	86.26	0.1336	0.1374	7

Influência da função de treino

No geral, ao mudar a função de treino, os resultados foram piores em comparação com a função ‘trainlm’, e o tempo de treino também foi maior. Isto acontece devido ao facto de a função ‘trainlm’ ser normalmente mais rápida e eficiente para problemas menores devido ao seu método de aproximação da segunda derivada (mas pode ser mais pesada em termos computacionais para problemas maiores).

Por outro lado, ‘traingd’ e ‘traingdm’ são mais simples e tendem a rodar até às 1000 épocas. Em contraste, ‘trainlm’ frequentemente converge mais rapidamente, como indicado pelos tempos de treino menores e pelo menor número de épocas necessárias.

A precisão global e de teste varia entre configurações, mas não parece haver uma diferença drástica que favoreça uma função de treino em particular. No entanto, visto a ‘trainlm’ ter apresentado desempenho superior, consideramo-la a melhor entre as testadas.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Melhor Rede
Conf1	1	10	tansig, purelin	traingd	dividerand = {0.7, 0.15, 0.15}	83.52	83.55	0.1642	0.1645	8
Conf2	1	10	tansig, purelin	trainbfg	dividerand = {0.7, 0.15, 0.15}	84.78	84.09	0.1504	0.1591	9
Conf3	1	10	tansig, purelin	traingdm	dividerand = {0.7, 0.15, 0.15}	84.09	83.96	0.1593	0.1604	10
Conf4	1	10	tansig, purelin	trainscg	dividerand = {0.7, 0.15, 0.15}	85.10	85.13	0.1494	0.1487	11
Conf5	1	10	tansig, purelin	traingdx	dividerand = {0.7, 0.15, 0.15}	84.08	84.24	0.1596	0.1576	12
Conf6	1	10	tansig, purelin	trainrp	dividerand = {0.7, 0.15, 0.15}	84.54	84.15	0.1540	0.1585	13
Conf7	1	10	tansig, purelin	trainbr	dividerand = {0.7, 0.15, 0.15}	83.64	83.60	0.1644	0.1640	14

Influência da função de ativação

As funções de ativação têm um impacto significativo na precisão global e no tempo de teste dos modelos. A primeira combinação (logsig, purelin) foi a que mais se aproximou da configuração default com uma diferença de menos de 1 ponto percentual (valor da precisão global). Este resultado pode ser considerado comparável ao default, levando em conta a variabilidade inerente a estes tipos de treino.

Ao analisar as restantes combinações, concluímos que a função de ativação da camada de saída tem um grau de importância superior às restantes. Isto porque, na configuração 2 por exemplo, a única alteração foi a função de ativação da camada de saída, e o resultado produzido foi inferior. Também na configuração 8, observamos que a única diferença para a configuração 1 é a função da camada de saída, o que neste caso provocou uma queda de cerca de 6 pontos percentuais na precisão global e um aumento significativo nos tempos de teste.

As restantes combinações apresentam resultados menos significativos, levando-nos a concluir que para o nosso dataset, configurações que incluem ‘purelin’ na camada de saída e ‘tansig’ ou ‘logsig’ nas restantes camadas são as mais eficazes.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Melhor Rede
Conf1	1	10	logsig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	84.62	84.48	0.1544	0.1552	15
Conf2	1	10	tansig, logsig	trainlm	dividerand = {0.7, 0.15, 0.15}	78.35	78.12	0.2156	0.2188	16
Conf3	1	10	hardlims,hardlim	trainlm	dividerand = {0.7, 0.15, 0.15}	58.57	58.03	0.4146	0.4197	17
Conf4	1	10	compet,softmax	trainlm	dividerand = {0.7, 0.15, 0.15}	74.99	74.95	0.2501	0.2505	18
Conf5	1	10	compet,logsig	trainlm	dividerand = {0.7, 0.15, 0.15}	75.53	75.50	0.2450	0.2450	19
Conf6	1	10	hardlim,hardlims	trainlm	dividerand = {0.7, 0.15, 0.15}	32.62	31.85	0.6695	0.6815	20
Conf7	1	10	tansig, softmax	trainlm	dividerand = {0.7, 0.15, 0.15}	78.26	78.23	0.2174	0.2177	21
Conf8	1	10	logsig,softmax	trainlm	dividerand = {0.7, 0.15, 0.15}	78.99	78.16	0.2087	0.2184	22

Influência da divisão de exemplos

Ao testar várias combinações de divisão, observa-se que a configuração que foca mais na fase de treino do que na validação e teste foi a que produziu os melhores resultados até ao momento. Este resultado pode ser atribuído ao facto de quanto mais dados estão disponíveis para treino, a rede tem mais oportunidades de aprender e ajustar os seus parâmetros de forma eficaz, o que geralmente melhora o desempenho da rede para este dataset.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Melhor Rede
Conf1	1	10	tansig, purelin	trainlm	dividerand = {0.35, 0.35, 0.30}	83.82	83.78	0.1608	0.1622	23
Conf2	1	10	tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	86.33	86.10	0.1364	0.1390	24
Conf3	1	10	tansig, purelin	trainlm	dividerand = {0.8, 0.10, 0.10}	85.40	84.68	0.1459	0.1532	25
Conf4	1	10	tansig, purelin	trainlm	dividerand = {0.1, 0.80, 0.10}	82.17	82.56	0.1702	0.1744	26
Conf5	1	10	tansig, purelin	trainlm	dividerand = {0.6, 0.20, 0.20}	84.98	84.67	0.1497	0.1533	27
Conf6	1	10	tansig, purelin	trainlm	dividerand = {0.4, 0.20, 0.40}	84.66	84.63	0.1548	0.1537	28
Conf7	1	10	tansig, purelin	trainlm	dividerand = {0.2, 0.30, 0.50}	83.72	83.39	0.1543	0.1661	29

Outros Testes

Nesta fase, testamos diversas configurações combinando os melhores resultados dos testes anteriores. Mantivemos a função de treino constante (‘trainlm’), visto ter sido a melhor anteriormente. Pela mesma ordem de ideias mantivemos a divisão dos exemplos constante ({0.9, 0.05, 0.05}). Focámo-nos em alterar o nº de camadas, de neurônios e as funções de ativação das camadas intermédias, mantendo a de camada de saída constante (‘purelin’) devido aos testes anteriores.

Os resultados foram promissores, subiu a máxima até ao momento que se situava nos 86.34% de precisão global e aproximámo-nos dos 90% em várias configurações e os tempos de treino diminuíram significativamente.

Dados estes testes, observámos que o aumento do número de camadas nem sempre melhora a precisão, como tinha sido concluído anteriormente, visto que a melhor média de precisão global foi de uma configuração com apenas 3 camadas.

	Número de camadas escondidas	Número de neurónios	Funções de ativação	Função de treino	Divisão dos exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Melhor Rede
Conf1	1	10	logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	86.18	85.81	0.1384	0.1419	30
Conf2	2	10,10	tansig, tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	87.20	86.38	0.1273	0.1362	31
Conf3	2	10,10	logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	89.56	89.34	0.1042	0.1066	32
Conf4	3	5,10,5	logsig, logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	88.67	87.04	0.1125	0.1296	33
Conf5	3	10,10,10	logsig, logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	89.82	88.63	0.1006	0.1137	34
Conf6	3	10,10,10	tansig, tansig, tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	89.00	87.78	0.1096	0.1222	35
Conf7	4	5,5,5,5	logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	86.39	85.93	0.1358	0.1407	36
Conf8	4	10,10,10,10	logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	89.13	88.23	0.1078	0.1172	37
Conf9	5	10,10,10,10	tansig, tansig, tansig, tansig, purelin	trainlm	divederand = {0.9,0.05,0.05}	87.30	86.25	0.1273	0.1375	38
Conf10	5	10,10,10,10,10	logsig, logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	87.60	87.21	0.1236	0.1279	39
Conf11	5	10,10,10,10,10	tansig, tansig, tansig, tansig, tansig, purelin	trainlm	divederand = {0.9,0.05,0.05}	86.38	86.42	0.1360	0.1358	40

No final dos testes, organizámo-los numa folha excel “Resultados Ordenados” do livro “estudoNN_TP” por ordem de precisão global, para ter outro tipo de consulta.

Tabela do Top 10 (de 40):

TOP	Nº Camadas Escondidas	Nº Neurónios	Funções de Ativação	Função de Treino	Divisão dos Exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste	ID Rede Melhor
1	3	10,10,10	logsig, logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	89.82	88.63	0.1006	0.1137	34
2	2	10,10	logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	89.56	89.34	0.1042	0.1066	32
3	4	10,10,10,10	logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	89.13	88.23	0.1078	0.1172	37
4	3	10,10,10	tansig, tansig, tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	89.00	87.78	0.1096	0.1222	35
5	3	5,10,5	logsig, logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	88.67	87.04	0.1125	0.1296	33
6	5	10,10,10,10,10	logsig, logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	87.60	87.21	0.1236	0.1279	39
7	5	10,10,10,10	tansig, tansig, tansig, tansig, purelin	trainlm	divederand = {0.9,0.05,0.05}	87.30	86.25	0.1273	0.1375	38
8	2	10,10	tansig, tansig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	87.20	86.38	0.1273	0.1362	31
9	5	10,10,10,10,10	tansig, tansig,tansig, tansig, purelin	trainlm	dividerand = {0.7, 0.15, 0.15}	86.56	86.26	0.1336	0.1374	7
10	4	5,5,5,5	logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	86.39	85.93	0.1358	0.1407	36

Melhores Redes

Após as 30 execuções de cada configuração, a rede com melhor desempenho é selecionada e gravada. Para determinar a melhor rede de cada configuração utilizámos uma fórmula ponderada que combina a precisão global e a precisão de teste: (0.7 * Precisao_Global + 0.3 * Precisao_Teste) > Melhor_Precisao. Esta abordagem permite dar maior importância à precisão global, sem negligenciar o desempenho no conjunto de teste. Desta forma, evitamos selecionar redes que, embora apresentem alta precisão global, possam ter desempenho insatisfatório nos testes. Este cuidado é crucial para evitar a escolha de redes que, apesar de terem um bom desempenho no treino, podem revelar-se fracas nos testes e, consequentemente, ter dificuldades em generalizar para novos conjuntos de dados.

Concluídos os testes, correrm o script ‘top3.m’ que percorre todas as redes e classificando-as usando a fórmula anterior ((0.7 * Precisao_Global + 0.3 * Precisao_Teste) > Melhor_Precisao), ordena-as por ordem decrescente do valor calculado. Para finalizar, seleciona as 3 melhores eliminando as restantes do diretório e alterando o nome destas para ‘best1’, ‘best2’ e ‘best3’ respetivamente. Obtendo os seguintes resultados:

TOP	Renomeação	Rede	Nº Camadas Escondidas	Nº Neurónios	Funções de Ativação	Função de Treino	Divisão dos Exemplos	Precisão Global	Precisão Teste	Tempo Global	Tempo Teste
1	best1.mat	network_32.mat	2	10,10	logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	90.48	94.64	0.0947	0.0536
2	best2.mat	network_34.mat	3	10,10,10	logsig, logsig, logsig, purelin	trainlm	dividerand = {0.9, 0.05, 0.05}	90.90	93.1606	0.0910	0.0684
3	best3.mat	network_35.mat	4	10,10,10,10	logsig, logsig, logsig, logsig, purelin	trainlm	divederand = {0.9,0.05,0.05}	90.37	92.0541	0.0970	0.0795

Dada a seleção do Top 3 (disponível para consulta na folha “TOP 3” do livro excel “estudoNN_TP”) , podemos observar que não corresponde ao Top 10 do capítulo anterior. Isto deve-se ao facto de os resultados do capítulo anterior serem a média das 30 execuções e estarem ordenados apenas pela precisão global, enquanto o Top 3 apresenta os resultados da rede específica. Assim podemos concluir que embora uma configuração tenha produzido melhores resultados gerais, não significa que a sua melhor rede seja a melhor também.

3.4 c) Uso do Dataset TEST

Para executar esta tarefa, criamos o script ‘alinea34c.m’. Este script começa por dividir o dataset em input e target como já foi explicado anteriormente e carrega as 3 melhores redes numa estrutura para facilitar o acesso durante os testes. A seguir, cada rede realiza a simulação sobre o conjunto de ‘input’, os resultados são normalizados para o intervalo [0,1], e posteriormente é aplicado um limiar (0.5) para converter os outputs em classificações binárias (esta normalização já foi explicada no ponto 3.4 b). A precisão de cada rede é calculada comparando as classificações com os ‘targets’ reais e expressa como uma percentagem de classificações corretas sobre o total.

Apresentam-se abaixo os resultados (também disponíveis na folha “Alínea C” do livro “estudoNN_TP”):

Rede "best1.mat"		
Output	Target	Correto
1	1	1
1	1	1
0	1	0
1	1	1
0	1	0
0	0	1
0	0	1
1	0	0
0	0	1
1	0	0
Precisão = 60.00%		

Rede "best2.mat"		
Output	Target	Correto
1	1	1
1	1	1
0	1	0
0	1	0
0	1	0
0	0	1
0	0	1
1	0	0
0	0	1
1	0	0
Precisão = 50.00%		

Rede "best3.mat"		
Output	Target	Correto
1	1	1
1	1	1
0	1	0
1	1	1
0	1	0
0	0	1
0	0	1
1	0	0
0	0	1
1	0	0
Precisão = 60.00%		

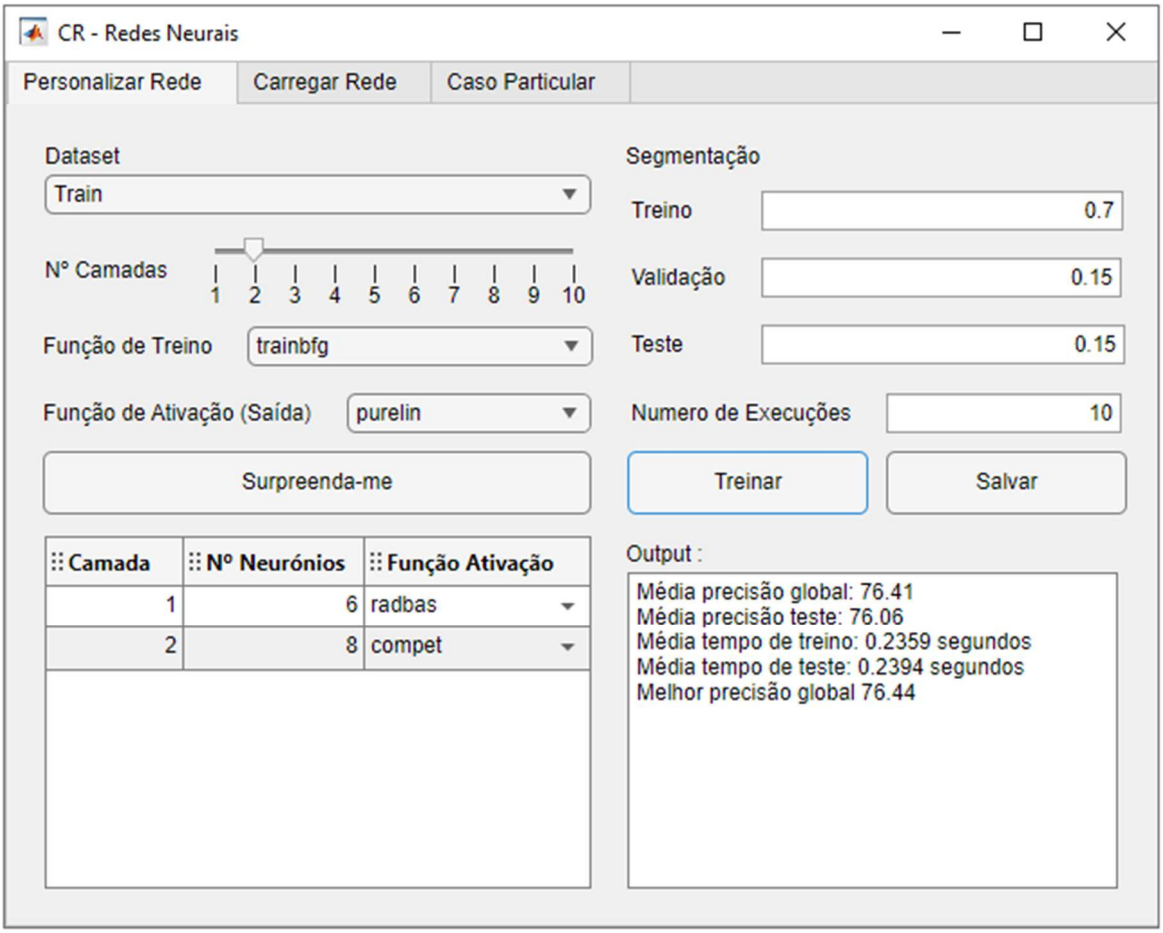
Os resultados obtidos são pouco satisfatórios, especialmente considerando que, apesar de um sucesso de 90% nos dados de treino, apenas conseguimos 60% e 50% nos testes. Esta discrepância deve-se à configuração das melhores redes obtidas no ponto 3.4 B, com uma divisão de dados de (0.9, 0.05, 0.05). Esta proporção elevada para treino resultou em redes muito especializadas nos dados específicos, mas com uma capacidade reduzida de generalizar para novos conjuntos de dados, o que é evidente pelos fracos resultados nos testes com novos dados. Este exemplo sublinha a importância de uma distribuição equilibrada entre dados de treino e teste, essencial para evitar o sobre ajuste e garantir uma boa generalização.

Aplicação gráfica

A aplicação está dividida em 3 partes: Personalizar Rede, Carregar Rede e Caso particular.

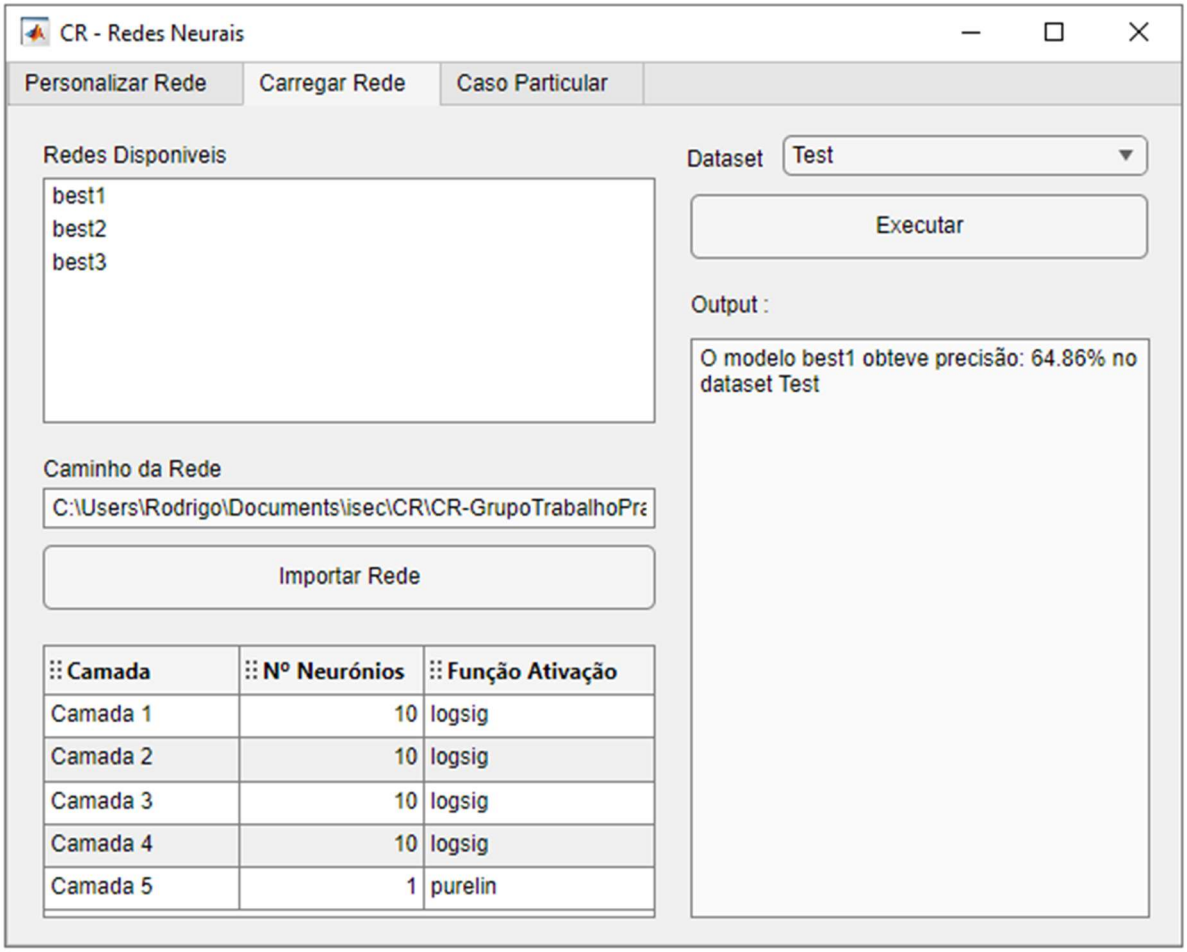
Personalizar Rede

Nesta secção, é possível configurar e treinar uma rede neuronal. O utilizador pode escolher o dataset para treino, o número de camadas, as funções de treino e ativação de cada camada, bem como a sua segmentação para treino, validação e teste. Além disso, o utilizador pode definir quantas vezes deseja executar o treino. Por exemplo, ao definir 10 número de execuções, a aplicação realizará 10 treinos e calculará a média, guardando a rede com a melhor precisão global.



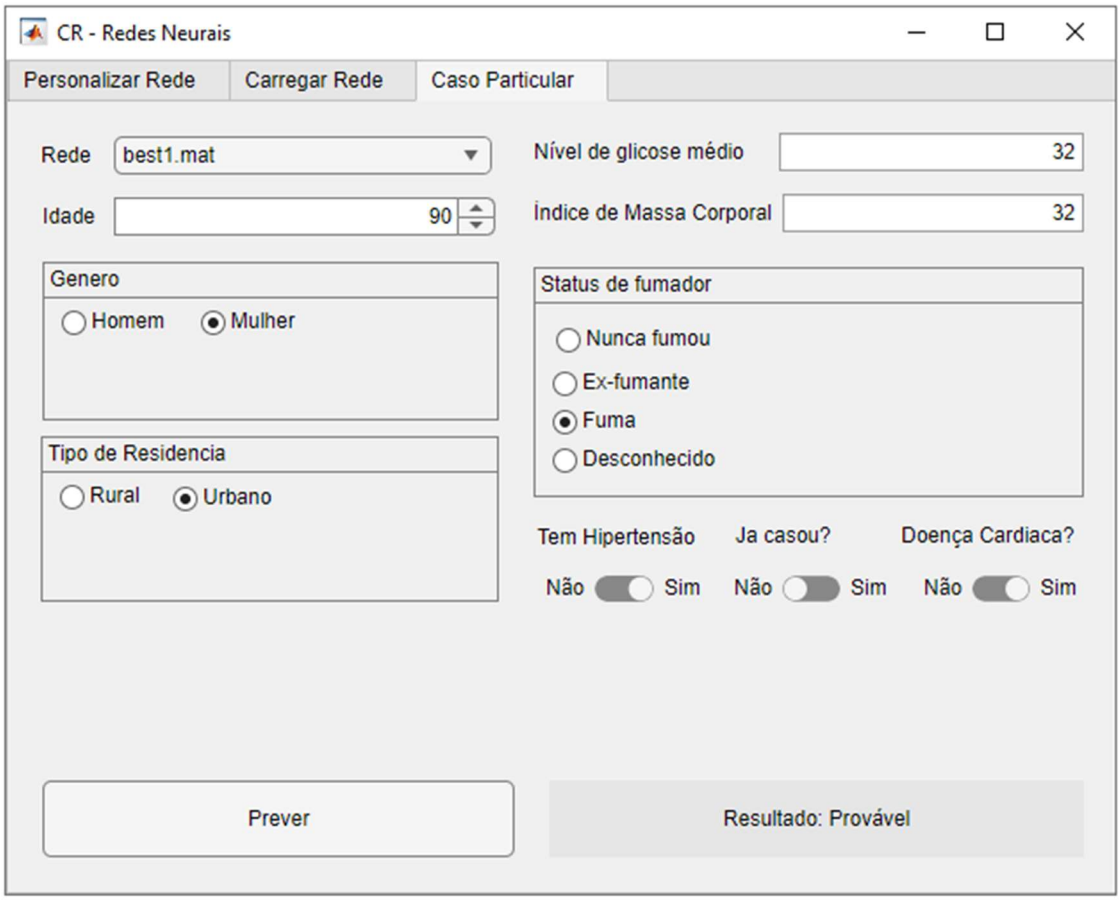
Carregar Rede

Esta secção permite visualizar todas as redes disponíveis, que estão localizadas na pasta "redes", e importar novas redes. Mostra a arquitetura da rede, ou seja, o número de neurónios por camada e a sua função de ativação. Além disso, é possível testar esta rede em diferentes datasets.



Caso Particular

Esta secção permite selecionar uma rede entre as disponíveis, indicar os valores dos atributos de um caso específico (tais como idade, género, tipo de residência - rural ou urbana -, histórico de tabagismo - nunca fumou, ex fumante, fumador desconhecido -, entre outros), e prever se esse caso pode ou não resultar em um acidente vascular cerebral (AVC). Após inserir os valores, basta clicar no botão "Prever" para obter a classificação dada pela rede.



Conclusão

Este trabalho prático permitiu-nos aprofundar o conhecimento sobre redes neuronais, destacando a influência da arquitetura e das metodologias de treino na eficácia dos modelos. Observámos que aumentar o número de camadas geralmente melhora a precisão, sendo que as funções de ativação "logsig" e o método de treino "trainlm" se mostraram particularmente eficazes.

No entanto, identificamos que uma precisão de 100% nos dados de treino não é sinónimo de um modelo robusto. Modelos treinados com o dataset "Start.csv", apesar de atingirem alta precisão, revelaram-se menos eficazes quando testados com novos dados. Isso demonstra que a alta precisão pode ser enganosa, especialmente em modelos treinados em datasets limitados sem uma adequada validação cruzada.

Os testes com o dataset "Test", que contém exemplos não vistos durante o treino, mostraram uma diminuição significativa na precisão, sublinhando a importância de testar os modelos em condições variadas para assegurar a sua generalização e aplicabilidade real. Este estudo ressalta a necessidade de uma distribuição equilibrada dos dados e de uma abordagem meticulosa no design do modelo para evitar o sobre ajuste e garantir a eficácia prática das redes neuronais.

Bibliografia

- Materiais da disciplina no Moodle: https://moodle.isec.pt/moodle/course/view.php?id=20428

- Documentação MATLAB: https://www.mathworks.com/help/matlab/ref/doc.html

- ChatGPT: https://chat.openai.com/