Exercício 4

Frederico Schardong

1 Enunciado do trabalho

O enunciado do exercício propõem a criação de uma rede neural com objetivo de aprender e classificar instâncias do `thyroid_dataset`. Os parâmetros da rede devem ser explorados pelo aluno. Este conjunto de dados consiste em 215 instâncias com 5 dimensões cada e a respectiva categoria de cada instância (1, 2 ou 3).

2 Resolução do Trabalho

Nesta seção são apresentados: (i) diferentes formas de normalização dos dados; (ii) os parâmetros utilizados; e (iii) detalhes de implementação para reproduzir o trabalho.

2.1 Normalização dos dados

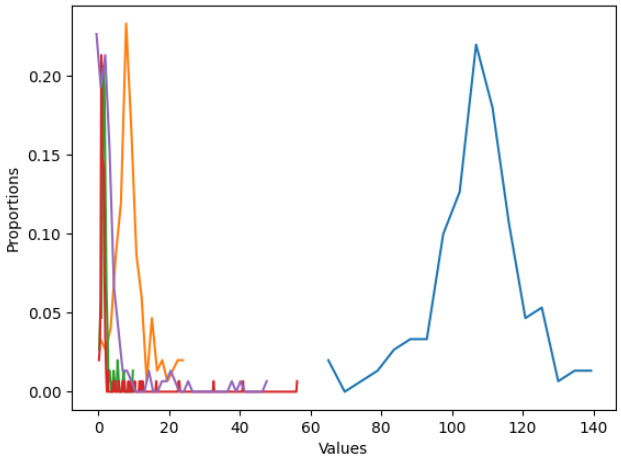
A distribuição da intensidade das dimensões 3 (verde), 4 (vermelho) e 5 (roxo) do problema são assimétricas, enquanto que as dimensões 1 (azul) e 2 (laranja) são próximas de uma distribuição normal, conforme pode ser visto nas Figuras 1a, 1b e 1c e na Tabela 1. Duas técnicas de normalização de dados foram utilizadas, a `minmax` e a `Yeo-Johnson` [1]. A primeira transforma os dados de entrada para o intervalo $[0; 1]$, porém não muda a distribuição os dados, enquanto que a segunda normaliza a distribuição dos dados mas o intervalo é variado, sendo definido em função dos dados de entrada.

	Dimensão	Mínimo	Máximo	Média	Desvio Padrão
Sem normalização	1	65.00	144.00	109.05	13.12
Normalização `minmax`		0.00	1.00	0.56	0.17
Normalização `Yeo-Johnson`		-2.75	3.20	0.00	1.00
Sem normalização	2	0.50	25.30	9.84	4.87
Normalização `minmax`		0.00	1.00	0.38	0.20
Normalização `Yeo-Johnson`		-2.81	2.38	0.00	1.00
Sem normalização	3	0.20	10.00	2.06	1.54
Normalização `minmax`		0.00	1.00	0.19	0.16
Normalização `Yeo-Johnson`		-2.99	2.42	0.00	1.00
Sem normalização	4	0.10	56.40	109.05	13.12
Normalização `minmax`		0.00	1.00	0.05	0.12
Normalização `Yeo-Johnson`		-2.04	2.56	0.00	1.00
Sem normalização	5	-0.60	48.80	4.09	7.15
Normalização `minmax`		0.00	1.00	0.09	0.14
Normalização `Yeo-Johnson`		-1.73	2.26	0.00	1.00

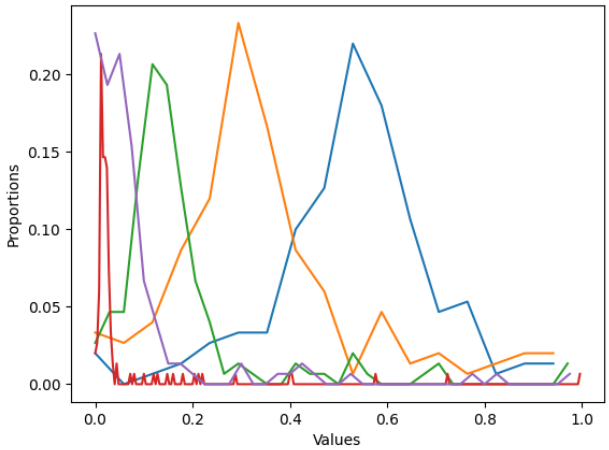
Table 1: Características da distribuição antes e depois das normalizações.

2.2 Parâmetros testados

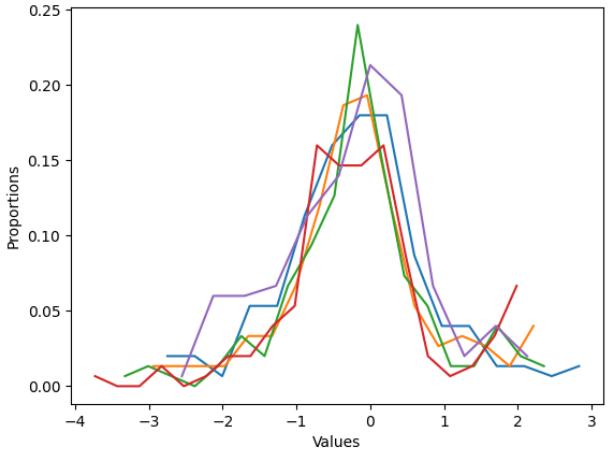
Para testar diferentes configuração e encontrar a que produz os melhores resultados, dezenas de configurações foram testadas. A biblioteca `scikit-learn` [5], utilizada para implementar a rede, fornece a classe `GridSearchCV` que recebe como parâmetro as possíveis configurações de uma rede neural e executa cada possível combinação de configuração, utilizando nesta implementação a métrica `f1` para comparar o desempenho das diferentes configurações. Esta métrica foi escolhida pois é a média harmônica da precisão e acurácia [2], produzindo valores no intervalo $[0; 1]$. A permutação de todos os parâmetros detalhados na Tabela 2 foram executados para cada normalizador utilizado.



(a) Antes da normalização.



(b) Depois da normalização `minmax`.



(c) Depois da normalização `Yeo-Johnson`.

Figure 1: Histograma dos dados não-normalizados e normalizados.

2.3 Detalhes de implementação

O trabalho foi implementado em Python 3, e as seguintes bibliotecas foram utilizadas:

- `scipy` [7] para carregar o dataset do arquivo `thyroid.mat`;
- `scikit-learn` [5] para: (i) normalizar os dados; (ii) gerar a permutação de todas as configurações de teste; e (iii) criar a rede neural, treiná-la e testá-la;
- `numpy` [6] para obter os valores mínimos, máximos, média e desvio padrão de cada dimensão antes e depois das normalizações;
- `matplotlib` [3] para geração dos gráficos dos histogramas;
- `pandas` [4] para converter os resultados dos testes em tabelas LaTeX.

Todas as bibliotecas podem ser instaladas através do gerenciador de pacotes padrão do Python, o `pip`, através do comando:

```
$ pip install scipy scikit-learn numpy matplotlib pandas
```

Parâmetro	Possíveis Valores
Função de ativação	Tangente Hiperbólica (tanh), Unidade Linear Retificada (relu), Sigmóide, Linear
Solucionador/Algorítmo	Gradiente Descendente (sgd), Adam, lbfgs
Taxa de aprendizado	0.00001, 0.0001, 0.001, 0.01
Épocas máximas	100, 1000, 10000
Camadas escondidas	(5), (10), (20), (50), (100), (5,5), (10,10), (20,20), (5,10,20), (20,10,5)

Table 2: Todas as configurações da rede testadas.

O código fonte desta tarefa é publico[1] e possui a licensa MIT. Ele foi submetido junto com este relatório. Para reproduzir as imagens e tabelas listas neste relatório, basta executa o comando abaixo.

```
$ python3 main.py
```

3 Resultados

Para treinar e medir o `f1-score` de todas as combinações possíveis de parâmetros de treinamento da rede (listados na Tabela 2), cada combinação de parâmetros foi testada usando 5-*fold cross-validation* no conjunto de treinamento, que consiste em 70% dos dados do *dataset*, *i.e.* 150 instâncias, aleatoriamente selecionados no começo da execução do programa.

As Tabelas 3 e 4 listam, respectivamente, os valores de `f1-score` e os parâmetros que produziram os melhores resultados (*i.e.* maior `f1`) para cada algorítmo de treinamento e limíte de épocas, divididos por método de normalização. A Tabela 3 subdivide os resultados ainda mais, mostrando os resultados para cada categoria de classificação. É importante observar que de todo o dataset, 150 instâncias são classificadas como categoria 1, 35 como categoria 2 e 30 como categoria, ou seja, a distribuição de instâncias por categoria não é igualitário.

Como pode ser observado na Tabela 3, quando os dados foram normalizados com `minmax`, o `f1-score` dos algorítmos de treinamento `sgd` e `adam` precisaram de 10000 épocas para convergirem, diferente de quando normalizados com `Yeo-Johnson`, que com apenas 100 épocas já produziram resultados próximos a 1. A diferença dos dois normalizadores é especialmente visível limitando o treinamento a 100 épocas para as categorias 2 e 3 (que possuem apenas 35 e 30 instâncias no *dataset*, respectivamente), que produzem resultados insatisfatórios.

Em relação aos parâmetros que produziram os maiores `f1-score`, pode ser observado na Tabela 4 que, de forma geral, a taxa de aprendizado (α) é maior quando os dados foram normalizados com `Yeo-Johnson`. Em relação as funções de ativação, em nenhum cenário `relu` produziu o melhor resultado.

Algorítmo \ Épocas		100		1000		10000	
Algorítmo	Cat.	MM	YJ	MM	YJ	MM	YJ
lbfgs	1	0,98	0,98	0,98	0,98	0,98	0,98
	2	0,95	0,92	0,95	0,92	0,95	0,92
	3	0,95	1,00	0,95	1,00	0,95	1,00
sgd	1	0,83	0,97	0,96	0,97	0,98	0,97
	2	0,00	0,92	0,84	0,88	0,95	0,88
	3	0,00	0,94	0,94	1,00	0,95	1,00
adam	1	0,85	0,96	1,00	0,97	1,00	0,97
	2	0,17	0,92	1,00	0,88	1,00	0,88
	3	0,50	0,88	1,00	1,00	1,00	1,00

Table 3: `f1-score` de cada categoria (1, 2 e 3) para os diferentes algoritmos de treinamento (linhas) e numero máximo de iterações (colunas), para os dois normalizadores (minmax (MM) e Yeo-Johnson (YJ)).

[1] Código fonte disponível em `https://github.com/fredericoschardong/programming-exercise-4-thyroid-hyper-parameterization`.

Algorítmo \ Épocas		100		1000		10000	
Algorítmo	Parâmetros	MM	YJ	MM	YJ	MM	YJ
lbfgs	α	0.00001	0.001	0.00001	0.001	0.00001	0.001
	Func. Ativ.	sigmoid	linear	sigmoid	linear	sigmoid	linear
	Neurônios	(5)	(5, 10, 20)	(5)	(5, 10, 20)	(5)	(5, 10, 20)
sgd	α	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001
	Func. Ativ.	tanh	tanh	linear	tanh	tanh	tanh
	Neurônios	(5)	(100)	(5, 10, 20)	(5, 5)	(5, 10, 20)	(5, 5)
adam	α	0.00001	0.0001	0.00001	0.01	0.00001	0.01
	Func. Ativ.	linear	linear	linear	sigmoid	linear	sigmoid
	Neurônios	(20, 10, 5)	(20, 20)	(20, 20)	(20, 20)	(20, 20)	(20, 20)

Table 4: Parâmetros que resultaram nos maiores `f1-score`.

4 Considerações Finais

Dezenas de diferentes parâmetros foram utilizados para classificar instâncias do `thyroid_dataset`, usando 70% dos dados para treinamento e 30% para teste. Para cada configuração da rede, os 70% dos dados usado no treinamento foram subdivididos em 5 para que cada teste fosse realizado usando 5-*fold cross-validation*.

Com os testes executados, é possível observar que a função de normalização tem grande importância para a rápida convergência da rede, principalmente quando o número de épocas máxima de treinamento é baixo.

References

[1] In-Kwon Yeo and Richard A Johnson. "A new family of power transformations to improve normality or symmetry". In: *Biometrika* 87.4 (2000), pp. 954–959.

[2] Tsong Yueh Chen, Fei-Ching Kuo, and Robert Merkel. "On the statistical properties of the f-measure". In: *Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings.* IEEE. 2004, pp. 146–153.

[3] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[4] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[7] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2.