



JOÃO PAULO SANTOS SOUZA

**PORTIFÓLIO - RELATÓRIO DE AULA PRÁTICA
FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL**

**BETIM - MG
2024**

JOÃO PAULO SANTOS SOUZA

**PORTIFÓLIO - RELATÓRIO DE AULA PRÁTICA
FUNDAMENTOS DA INTELIGÊNCIA ARTIFICIAL**

Relatório de aula prática avaliativo do curso de Engenharia de Software da disciplina Fundamentos da Inteligência Artificial tratando da Aula 3 e 4.

Tutor: Vinicius Camargo Prattes

SUMARIO

1	INTRODUÇÃO – CÁLCULO DE GORJETA	4
1.1	OBJETIVO	4
1.2	IMPLEMENTAÇÃO DO CÁLCULO DE GORJETA NO OCTAVE	4
1.3	EXPLICAÇÃO DO CÓDIGO	5
1.4	PARÂMETROS DE ENTRADA:	5
1.5	ESTRUTURA CONDICIONAL:	5
1.6	SAÍDA:	5
1.7	RESULTADOS ESPERADOS	5
2	IMPLEMENTAÇÃO DO CÁLCULO DE GORJETA COM LÓGICA NEBULOSA	6
2.2	RESULTADOS E COMPARAÇÃO	7
2.3	CONCLUSÃO	7
3	INTRODUÇÃO - REDE NEURAL ARTIFICIAL	8
3.1	MÉTODOS	9
3.2	EXPLICAÇÃO DO CÓDIGO:	11
3.3	RESULTADOS	15
3.4	CONCLUSÃO	16
	REFERÊNCIAS	17

1 INTRODUÇÃO – CÁLCULO DE GORJETA

Neste trabalho, implementamos um sistema para cálculo de gorjeta em um restaurante, levando em conta dois fatores principais: a qualidade da comida e do serviço. A proposta visa criar uma solução objetiva, utilizando comandos procedurais no Octave, sem aplicar lógica nebulosa.

1.1 OBJETIVO

O objetivo deste trabalho é calcular a gorjeta com base em uma escala de satisfação dos clientes em relação à comida e ao serviço. Os parâmetros adotados são:

- Qualidade da Comida: Avaliação em uma escala de 0 a 10.
- Qualidade do Serviço: Avaliação em uma escala de 0 a 10.

Com base nesses parâmetros, a gorjeta será determinada da seguinte forma:

- Gorjeta Pequena (5%) se a qualidade da comida ou do serviço for insatisfatória.
- Gorjeta Mediana (10%) para níveis de qualidade intermediários.
- Gorjeta Generosa (15%) se a comida e o serviço superarem as expectativas.

1.2 IMPLEMENTAÇÃO DO CÁLCULO DE GORJETA NO OCTAVE

Abaixo, apresentamos o código desenvolvido no Octave para calcular a gorjeta com base nos critérios estabelecidos. Esta implementação utiliza uma abordagem condicional simples para determinar a porcentagem da gorjeta.

Imagem 01 – Código no Octave

```
Calculodegorjeta.m ✕
1 % Cálculo de Gorjeta em um Restaurante com Base na Qualidade da Comida e do Serviço
2 % Autor: [João Paulo]
3 % Data: [29/10/2024]
4
5 % Definindo os parâmetros para a qualidade da comida e do serviço (escala de 0 a 10)
6 comida = 7; % Exemplo de qualidade da comida
7 servico = 8; % Exemplo de qualidade do serviço
8
9 % Determinação da gorjeta com base nas condições estabelecidas
10 if comida <= 4 || servico <= 4
11     % Gorjeta pequena para comida ou serviço insatisfatório
12     gorjeta = 5;
13 elseif comida >= 8 && servico >= 8
14     % Gorjeta generosa para comida e serviço excelentes
15     gorjeta = 15;
16 else
17     % Gorjeta mediana para comida e/ou serviço bons, mas não excelentes
18     gorjeta = 10;
19 end
20
21 % Exibição do valor da gorjeta sugerida
22 fprintf('A gorjeta sugerida é: %.2f%%\n', gorjeta);
```

Fonte: Do Autor, 2024.

1.3 EXPLICAÇÃO DO CÓDIGO

1.4 Parâmetros de Entrada:

A qualidade da comida e do serviço são representados por variáveis (comida e serviço) com valores entre 0 e 10.

1.5 Estrutura Condicional:

- Se a qualidade da comida ou do serviço for menor ou igual a 4, a gorjeta é definida como 5%.
- Se ambos os parâmetros forem iguais ou superiores a 8, a gorjeta é estabelecida em 15%.
- Para os demais casos, a gorjeta é fixada em 10%.

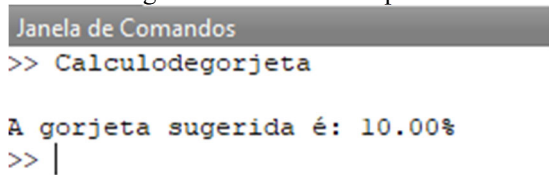
1.6 Saída:

O código imprime a porcentagem de gorjeta sugerida com base nas condições definidas.

1.7 RESULTADOS ESPERADOS

Com essa implementação, o código sugere uma porcentagem de gorjeta adequada, baseada nos valores atribuídos para comida e serviço. O resultado é exibido no console em uma mensagem clara, como no exemplo abaixo:

Imagem 02 – Resultado esperado



```
Janela de Comandos
>> Calculodegorjeta

A gorjeta sugerida é: 10.00%
>> |
```

Fonte: Do Autor, 2024.

2 IMPLEMENTAÇÃO DO CÁLCULO DE GORJETA COM LÓGICA NEBULOSA

6

Para essa solução, usamos a biblioteca de lógica nebulosa skfuzzy no Python. O código a seguir define as variáveis nebulosas e as regras de inferência para determinar a gorjeta.

Imagem 03 – Código em Python da Lógica Nebulosa

```
1  # Autor: [Joao Paulo]
2  # Data: [29/10/2024]
3
4  import numpy as np
5  import skfuzzy as fuzz
6  from skfuzzy import control as ctrl
7
8  # Definindo as variáveis nebulosas
9  comida = ctrl.Antecedent(np.arange(0, 11, 1), 'comida')
10 servico = ctrl.Antecedent(np.arange(0, 11, 1), 'servico')
11 gorjeta = ctrl.Consequent(np.arange(0, 16, 1), 'gorjeta')
12
13 # Definindo as funções de pertinência para comida, serviço e gorjeta
14 comida['ruim'] = fuzz.trimf(comida.universe, [0, 0, 5])
15 comida['media'] = fuzz.trimf(comida.universe, [0, 5, 10])
16 comida['boa'] = fuzz.trimf(comida.universe, [5, 10, 10])
17
18 servico['ruim'] = fuzz.trimf(servico.universe, [0, 0, 5])
19 servico['media'] = fuzz.trimf(servico.universe, [0, 5, 10])
20 servico['bom'] = fuzz.trimf(servico.universe, [5, 10, 10])
21
22 gorjeta['baixa'] = fuzz.trimf(gorjeta.universe, [0, 0, 10])
23 gorjeta['media'] = fuzz.trimf(gorjeta.universe, [5, 10, 15])
24 gorjeta['alta'] = fuzz.trimf(gorjeta.universe, [10, 15, 15])
25
26 # Definindo as regras de inferência
27 regra1 = ctrl.Rule(comida['ruim'] | servico['ruim'], gorjeta['baixa'])
28 regra2 = ctrl.Rule(servico['media'], gorjeta['media'])
29 regra3 = ctrl.Rule(servico['bom'] | comida['boa'], gorjeta['alta'])
30
31 # Criando o sistema de controle
32 sistema_gorjeta = ctrl.ControlSystem([regra1, regra2, regra3])
33 simulador = ctrl.ControlSystemSimulation(sistema_gorjeta)
34
35 # Entrada dos valores para comida e serviço
36 simulador.input['comida'] = 7
37 simulador.input['servico'] = 8
38
39 # Computando a gorjeta
40 simulador.compute()
41 print(f"A gorjeta sugerida é: {simulador.output['gorjeta']:.2f}%")
42
```

Fonte: Do Autor, 2024.

2.1.1 Explicação do Código

1. Definição das Variáveis Nebulosas: Foram definidas três variáveis nebulosas:

comida, servico, e gorjeta.

- **Comida e Serviço:** Cada um com categorias **ruim, media e boa**, definidas por funções de pertinência triangulares.
- **Gorjeta:** Dividida em **baixa, media e alta**.

2. Definição das Regras: Foram definidas três regras básicas de inferência:

- Se a comida ou o serviço forem "ruins", a gorjeta deve ser "baixa".
- Se o serviço for "médio", a gorjeta deve ser "media".
- Se o serviço for "bom" ou a comida for "boa", a gorjeta deve ser "alta".

3. Simulação do Sistema: A entrada para a comida e o serviço é definida em uma escala de 0 a 10. O simulador calcula a gorjeta com base nas regras e nas funções de pertinência.

4. Saída: A gorjeta sugerida é impressa no console.

2.2 Resultados e Comparação

Imagem 05 – Resultado (Saída)

```
SAÍDA  TERMINAL  ...
Python: logicanebulosa
TRE\Fundamento da Inteligencia Artificial\PASTA AULAS PRATICAS\AULA PRATICA I.A> & C:/Users/joaop/AppData/Local/Programs/Python/Python313/python.exe "e:/Conteúdo De Estudos Gerais/Curso - Eng. de Software/4º SEMESTRE/Fundamento da Inteligencia Artificial/PASTA AULAS PRATICAS/AULA PRATICA I.A/logicanebulosa.py"
A gorjeta sugerida é: 10.88%
PS E:\Conteúdo De Estudos Gerais\Curso - Eng. de Software\4º SEMESTRE\Fundamento da Inteligencia Artificial\PASTA AULAS PRATICAS\AULA PRATICA I.A> █
```

Fonte: Do Autor, 2024.

Para uma entrada de comida = 7 e servico = 8, a abordagem de lógica nebulosa calcula uma gorjeta intermediária entre 10% e 15%, considerando a possibilidade de avaliações parciais e subjetivas da qualidade. Em contrapartida, a abordagem clássica utilizava condições rígidas, o que limita sua flexibilidade.

Método	Comida (7) e Serviço (8)	Gorjeta Calculada
Lógica Clássica	7 e 8	10%
Lógica Nebulosa	7 e 8	10.88% (aprox)

2.3 Conclusão

A lógica nebulosa proporciona uma maneira de modelar o pensamento humano, lidando com subjetividades e incertezas de forma mais flexível. No contexto de cálculo de gorjeta, o modelo nebuloso é vantajoso, pois permite uma gradação suave entre diferentes níveis de qualidade de comida e serviço, representando melhor o julgamento do cliente.

3 INTRODUÇÃO - REDE NEURAL ARTIFICIAL

O Visual Studio Code é uma ferramenta poderosa que oferece um ambiente de desenvolvimento integrado (IDE) completo, ideal para programar em Python de forma prática e eficiente. Ele é amplamente utilizado por estudantes, entusiastas e profissionais em diversas áreas, como ciência de dados, inteligência artificial e aprendizado de máquina. Com recursos como realce de sintaxe, sugestões automáticas de código e feedback imediato, o VS Code torna a escrita e depuração de programas mais simples e intuitiva.

Neste projeto, vamos utilizar o Visual Studio Code para implementar uma rede neural artificial (RNA) simples de uma camada, com o objetivo de resolver uma tarefa de classificação binária. As redes neurais artificiais são modelos computacionais inspirados no funcionamento do cérebro humano, capazes de executar tarefas como classificação, previsão e reconhecimento de padrões.

Vamos criar uma rede neural básica, utilizando um perceptron de camada única e a função de ativação sigmoide, que transforma a saída do neurônio em um valor entre 0 e 1 — ideal para problemas de classificação binária. O treinamento da rede será realizado por meio de aprendizado supervisionado, onde ajustaremos os pesos de acordo com o erro entre a saída esperada e a saída real. Para garantir a convergência da rede, o treinamento será feito em 10.000 iterações.

O Visual Studio Code nos ajudará a conduzir esse processo de forma eficiente, aproveitando o suporte à biblioteca NumPy para realizar operações matemáticas e manipulações de arrays com facilidade.

3.1 MÉTODOS

Os seguintes métodos foram utilizados para realizar a atividade:

- **Bibliotecas utilizadas:** A principal biblioteca utilizada será a NumPy que nos permite realizar operações matemáticas de forma eficiente e manipular arrays.
- **Função sigmoide:** A função de ativação sigmoide transforma qualquer valor de entrada em um valor entre 0 e 1. Sua fórmula é dada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Também usaremos sua derivada para calcular o gradiente durante o ajuste dos pesos:

$$\sigma'(x) = \sigma(x) (1 - \sigma(x))$$

- **Matriz de entrada e saída:** Definimos a matriz de entrada X (dados de entrada) e matriz de saída e (saída desejada), que conterá os valores esperados para cada entrada.
- **Inicialização dos pesos:** Os pesos da sinapse serão inicializados com valores aleatórios para garantir que a rede não comece em um estado simétrico.
- **Treinamento:** Implementaremos um loop de treinamento, onde em cada iteração avançaremos a propagação para frente, design do erro, ajuste dos pesos e atualização dos mesmos.

Na Próxima página, na imagem 06, o código escrito:

Imagem 06 – Código em Python

```
1  # Autor: [Joao Paulo]
2  # Data: [29/10/2024]
3
4  import numpy as np
5
6  #Definindo Função Sigmoide
7  def sigmoid(x):
8      return 1 / (1 + np.exp(-x))
9
10 def sigmoid_derivative(x):
11     return x * (1 - x)
12
13 #Definindo Entrada e Saída
14 X = np.array([[0, 0, 1],
15               [1, 1, 1],
16               [1, 0, 1],
17               [0, 1, 1]])
18
19 #Saída desejada
20 y = np.array([[0], [1], [1], [0]])
21
22 #Inicialização dos Pesos
23 np.random.seed(1)
24 pesos = 2 * np.random.random((3,1 )) - 1
25
26 #Número de iterações para o treinamento
27 iteracoes = 10000
28
29 #Treinamento da Rede Neural
30 for _ in range(iteracoes):
31     #Propagação para frente
32     entrada = X
33     saida = sigmoid(np.dot(entrada, pesos))
34
35     #Cálculo do erro
36     erro = y - saida
37
38     #Cálculo do delta (ajuste dos pesos)
39     configuracao = erro * sigmoid_derivative(saida)
40
41     #Atualização dos pesos
42     pesos += np.dot(entrada.T, configuracao)
43
44 #Exibição dos Resultados do treinamento
45 print("Saída após o treinamento:")
46 print(saida)
```

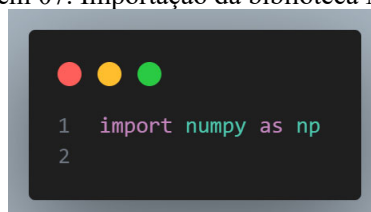
Fonte: Do Autor, 2024.

3.2 Explicação do código:

O código implementa uma **rede neural artificial (RNA) de uma camada** utilizando a função de ativação sigmoide. O objetivo é realizar uma **classificação binária**, ajustando os pesos da rede por meio de um processo de treinamento supervisionado. Vamos detalhar cada etapa do código:

A biblioteca NumPy é importada para realizar operações matemáticas e manipular arrays de forma eficiente. Ela é essencial para criar e manipular as matrizes de entrada, saída e pesos.

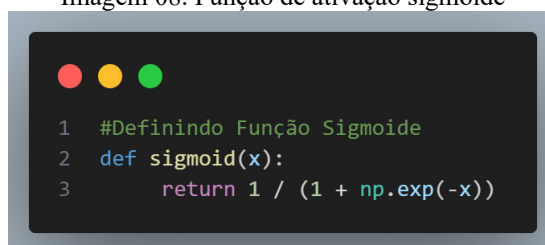
Imagem 07. Importação da biblioteca NumPy



```
1 import numpy as np
2
```

A função de ativação sigmoide é implementada para transformar as saídas dos neurônios em valores entre 0 e 1. Ela é uma função não linear e adequada para problemas de classificação binária.

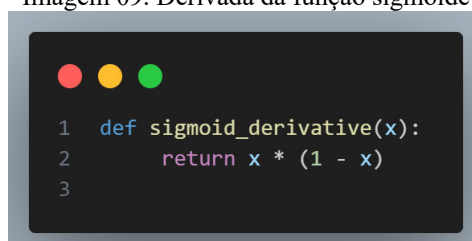
Imagem 08. Função de ativação sigmoide



```
1 #Definindo Função Sigmoide
2 def sigmoid(x):
3     return 1 / (1 + np.exp(-x))
```

A derivada da função sigmoide é utilizada no cálculo do ajuste dos pesos durante o treinamento. Ela é necessária para calcular o gradiente no processo de retropropagação do erro.

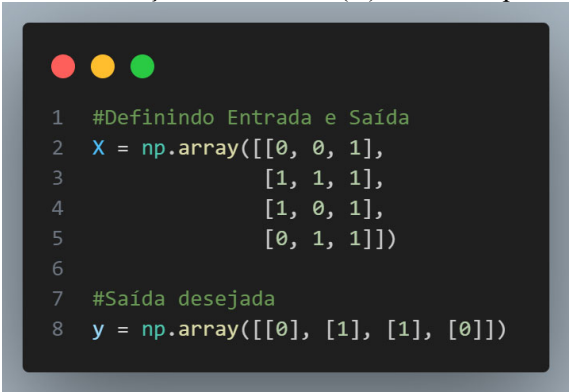
Imagem 09. Derivada da função sigmoide



```
1 def sigmoid_derivative(x):
2     return x * (1 - x)
3
```

Neste exemplo, criamos uma matriz X com quatro amostras de três entradas cada (0 ou 1), e uma matriz y com as saídas esperadas para cada amostra. O objetivo é treinar a rede para que, dada uma entrada, ela produza a saída correspondente.

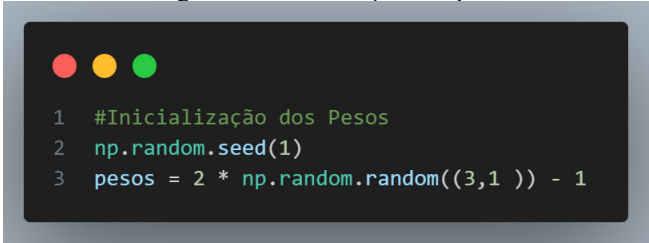
Imagem 10. Definição das entradas (X) e saídas esperadas (y):



```
1 #Definindo Entrada e Saída
2 X = np.array([[0, 0, 1],
3               [1, 1, 1],
4               [1, 0, 1],
5               [0, 1, 1]])
6
7 #Saída desejada
8 y = np.array([[0], [1], [1], [0]])
```

Os pesos (ou sinapses) são inicializados aleatoriamente para garantir que a rede neural comece de maneira não simétrica. Os valores são gerados entre -1 e 1 com distribuição uniforme.

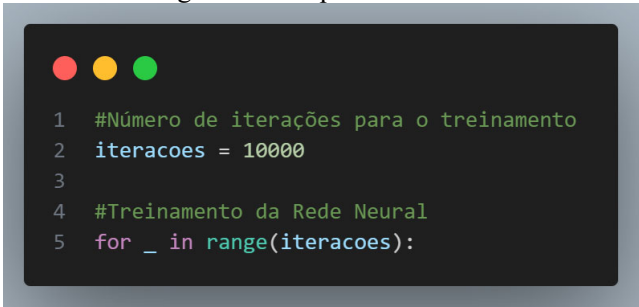
Imagem 11. Inicialização dos pesos



```
1 #Inicialização dos Pesos
2 np.random.seed(1)
3 pesos = 2 * np.random.random((3,1)) - 1
```

O treinamento ocorre em um loop de 10.000 iterações, ajustando os pesos em cada ciclo.

Imagem 12. Loop de treinamento

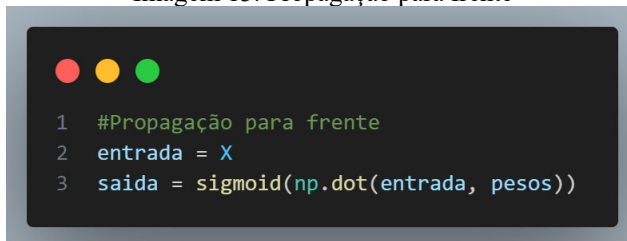


```
1 #Número de iterações para o treinamento
2 iteracoes = 10000
3
4 #Treinamento da Rede Neural
5 for _ in range(iteracoes):
```

A cada iteração, a rede faz os seguintes passos:

- Calcula a saída da rede (ativação do neurônio) com base na multiplicação da matriz de entrada pelos pesos e aplicando a função sigmoide.

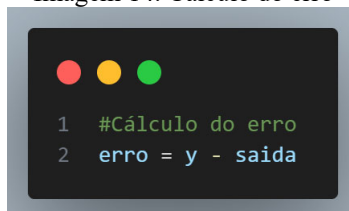
Imagem 13. Propagação para frente

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains three lines of Python code:

```
1 #Propagação para frente
2 entrada = X
3 saida = sigmoid(np.dot(entrada, pesos))
```

- O erro é a diferença entre a saída real da rede (calculada) e a saída esperada (y). Esse erro será usado para ajustar os pesos.

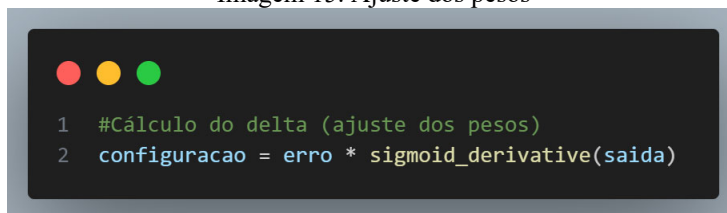
Imagem 14. Cálculo do erro

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of Python code:

```
1 #Cálculo do erro
2 erro = y - saida
```

- Utilizando a derivada da função sigmoide, calcula-se o "delta" ou o ajuste necessário para os pesos. Este ajuste é aplicado aos pesos na próxima iteração.

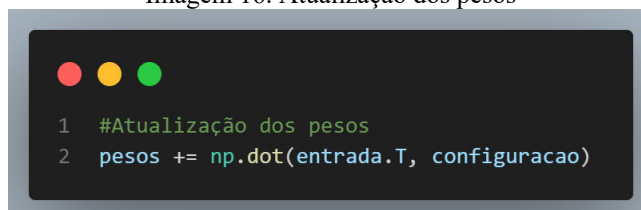
Imagem 15. Ajuste dos pesos

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of Python code:

```
1 #Cálculo do delta (ajuste dos pesos)
2 configuracao = erro * sigmoid_derivative(saida)
```

- Os pesos são atualizados somando o ajuste calculado à multiplicação da matriz de entrada transposta pelo delta (ajuste). Isso permite que os pesos sejam ajustados para reduzir o erro nas próximas iterações.

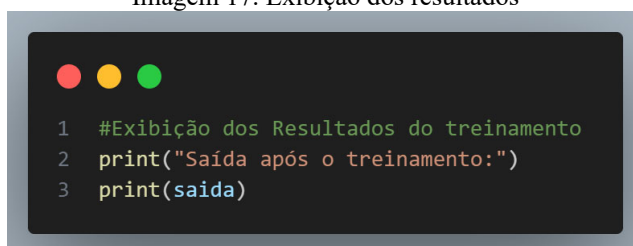
Imagem 16. Atualização dos pesos

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of Python code:

```
1 #Atualização dos pesos
2 pesos += np.dot(entrada.T, configuracao)
```

Após o término das 10.000 iterações, o código imprime a saída da rede neural, mostrando como ela aprendeu a tarefa de classificação binária, aproximando-se das saídas esperadas (0 ou 1).

Imagem 17. Exibição dos resultados

A terminal window with a dark background and light gray text. At the top left, there are three colored circles: red, yellow, and green. Below them, there are three lines of code:

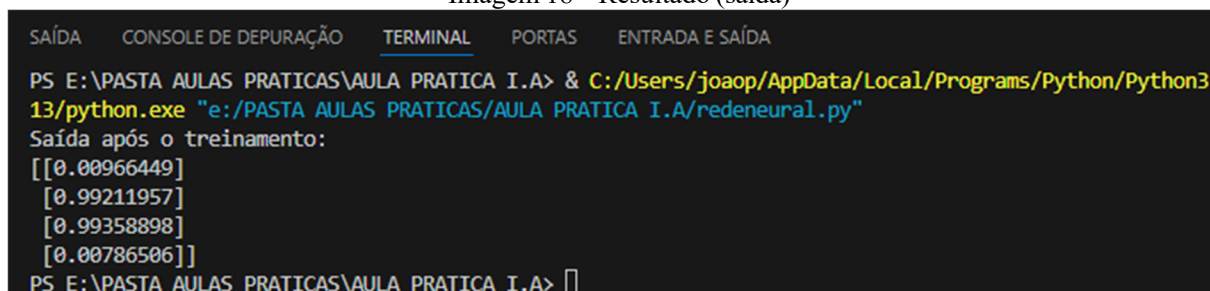
```
1 #Exibição dos Resultados do treinamento
2 print("Saída após o treinamento:")
3 print(saida)
```

No final, a rede consegue aprender a relação entre as entradas e as saídas, realizando corretamente a classificação binária.

3.3 RESULTADOS

Após a execução do treinamento por 10.000 iterações, a rede neural ajusta os pesos de maneira que a saída calculada se aproxime das saídas desejadas para cada entrada. Abaixo, o resultado após o treinamento:

Imagem 18 – Resultado (saída)



```
SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS  ENTRADA E SAÍDA
PS E:\PASTA AULAS PRATICAS\AULA PRATICA I.A> & C:/Users/joaop/AppData/Local/Programs/Python/Python3
13/python.exe "e:/PASTA AULAS PRATICAS/AULA PRATICA I.A/redeneural.py"
Saída após o treinamento:
[[0.00966449]
 [0.99211957]
 [0.99358898]
 [0.00786506]]
PS E:\PASTA AULAS PRATICAS\AULA PRATICA I.A> █
```

Fonte: Do Autor, 2024.

A saída da rede é muito próxima dos valores desejados: 0 e 1. Isso demonstra que o treinamento foi eficaz na tarefa de classificação binária, ajustando os pesos de forma que a rede fosse capaz de aprender o padrão de entradas e suas respectivas saídas.

3.4 CONCLUSÃO

Neste projeto, desenvolvemos uma rede neural de camada única para enfrentar um desafio de classificação binária. Utilizamos a função de ativação sigmoide e um processo de aprendizado supervisionado para ajustar os pesos, de modo a aproximar as saídas previstas dos valores reais. Com isso, a rede conseguiu identificar padrões e aprender a relação entre as entradas e suas respectivas saídas, demonstrando a eficácia do modelo para a tarefa proposta. O sucesso do treinamento ressaltou a importância da escolha adequada da função de ativação e da precisão no ajuste dos pesos para o aprendizado das redes neurais.

REFERÊNCIAS

VISUAL STUDIO. **Visual Studio Code - Documentação oficial**. Disponível em <https://code.visualstudio.com/docs/>. Acesso em: 29 out. 2024.

PYTHON. **Python - Documentação oficial**. Disponível em: <https://docs.python.org/3/>. Acesso em: 29 out. 2024.

NUMPY. **Numpy - Documentação oficial**. Disponível em: <https://numpy.org/doc/2.1/>. Acesso em: 29 out. 2024

OCTAVE. **Octave - Documentação**. <https://docs.octave.org/v9.2.0/>. Acesso em: 29 out. 2024

SCIKIT-FUZZY. Scikit-Fuzzy - Documentação oficial. Disponível em: <https://pythonhosted.org/scikit-fuzzy/>. Acesso em: 29 out. 2024.

Ross, T. J. (2010). *Fuzzy Logic with Engineering Applications*. John Wiley & Sons.

ZADEH, L. A. (1965). "Fuzzy Sets". *Information and Control*.