SCC5809 - Redes Neurais e Aprendizado Profundo
MLP - XOR e Matriz Identidade
Exercício 2

Gustavo Rocha Barbosa - Nº USP: 8927634
Dhyogo Nunes Costa - Nº USP: 13096109
Herlisson Maciel Bezerra - Nº USP: 10224760

September 2, 2024

Abstract

Implementar uma rede MLP para os seguintes casos:

I - A rede implementada no problema do XOR.

II - Considere o problema de auto-associador (encoding problem) no qual um conjunto de padrões ortogonais de entrada são mapeados num conjunto de padrões de saída ortogonais através de uma camada oculta com um número pequeno de neurônios. A figura em anexo mostra a arquitetura básica para se resolver este problema.

Essencialmente, o problema é aprender uma codificação de padrão com p-bit em um padrão de log2 p-bit, e em seguida aprender a decodificar esta representação num padrão de saída.

Pede-se: Construir o mapeamento gerado por uma rede multi-camadas com o algoritmo back-propagation (BP), para o caso do mapeamento identidade, considerando dois casos:

a) Padrão de entrada e Padrão de Saída: Id(8x8) e Id(8X8)
b) Padrão de entrada e Padrão de Saída: Id(15x15) e Id(15X15)
Onde Id denota a matriz identidade.

1 Introdução

O presente projeto foi feito em Python utilizando as bibliotecas Numpy e Tensorflow. Portanto, antes de mais nada precisamos importar as bibliotecas:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Numpy é utilizado para manipular as matrizes e arrays. A biblioteca Tensorflow e Keras serão utilizadas para a criação das arquiteturas e funcionamento das Redes Neurais.

2 I - Problema XOR

```
#Dados de treinamento para o problema XOR:
X_xor = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_xor = np.array([[0], [1], [1], [0]])
```

```
#Construindo o modelo com função Sigmoid e ReLu:
model_xor = Sequential([Dense(2, input_dim=2, activation='sigmoid'), Dense(1, activation='relu')])
# Camada intermediária com 2 neurônios e camada de saída com 1 neurônio

#Compilando o modelo:
model_xor.compile(optimizer='adam', loss='mean_squared_error')

#Treinando o modelo:
model_xor.fit(X_xor, y_xor, epochs=5000, verbose=0) #5000 épocas

#Testando o modelo:
outputs_xor = model_xor.predict(X_xor)
print("Resultados para o problema XOR:")
print(np.round(outputs_xor, decimals=4))
```

O presente problema é resolvido usando a biblioteca Tensorflow Sequential, usando a função sigmoide para a camada de neurônio intermediária e a função ReLU para a camada de saída. A arquitetura da Rede é formada por 2 neurônios na camada intermediária e 1 neurônio na camada de saída.

Input: [[0, 0], [0, 1], [1, 0], [1, 1]]

Targets: [[0], [1], [1], [0]]

Output depois de 5000 épocas: (54ms/época)
[[0.0023]
[0.9969]
[0.997]
[0.0041]]

3 II - Matriz Identidade

Foi criada uma função em python que recebe a dimensão da matriz identidade a ser calculada, por exemplo, se a matriz identidade que for passar pelo mapeamento (problema auto-associador) for de 8x8, a entrada deverá ser: mapeamento_identidade(8), caso for 15x15, a entrada deverá ser: mapeamento_identidade(15).

```
def mapeamento_identidade(dimensao_matriz_identidade):

  # Matriz Identidade:
  input_size = dimensao_matriz_identidade
  hidden_size = int(np.log2(input_size))  # Quantidade de Neurônios intermediária
  output_size = input_size
  X = np.eye(input_size)

  # Construindo o modelo:
  model_autoencoder = Sequential([
      Dense(hidden_size, input_dim=input_size, activation='sigmoid'),
      # Camada oculta - Função sigmoid
      Dense(output_size, activation='sigmoid')  # Camada de saída - Função sigmoide
  ])

  # Compilando o modelo:
  model_autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

```
  # Treinando o modelo
  model_autoencoder.fit(X, X, epochs=5000, verbose=0) #5000 épocas

  # Testando o modelo
  outputs = model_autoencoder.predict(X)
  print(f"\nMapeamento identidade {input_size}x{output_size} concluído.")
  return outputs
```

A arquitetura da Rede é composta por $log_2 N$ na camada intermediária, onde N é o número de entradas. Para uma matriz 8x8 a camada intermediária é composta de 3 neurônios e uma de 15x15 é composta por 4 neurônios. O número de neurônios de saída é iguail ao número de entrada, $Id_{8x8} = 8$ e $Id_{15x15} = 15$.

As funções de ativações para as camadas intermediárias e de saída é a função sigmoide:

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

O número de épocas é configurado dentro da função, e existe um valor padrão de 5000 épocas.

3.1 Saída Matriz Identidade

Temos as saídas das Matrizes Identidades depois de passar pelo mapeamento de 5000 épocas:

```
#Matriz 8x8:

48ms/step

Mapeamento identidade 8x8 concluído.

[[0.13  0.075 0.081 0.128 0.131 0.062 0.134 0.094]
 [0.114 0.771 0.093 0.178 0.023 0.001 0.138 0.133]
 [0.165 0.155 0.803 0.122 0.282 0.03  0.181 0.015]
 [0.123 0.13  0.041 0.141 0.075 0.038 0.128 0.154]
 [0.117 0.004 0.1   0.065 0.595 0.112 0.113 0.012]
 [0.151 0.005 0.033 0.117 0.265 0.796 0.132 0.192]
 [0.125 0.095 0.083 0.124 0.128 0.033 0.131 0.076]
 [0.104 0.09  0.001 0.176 0.019 0.134 0.101 0.723]]

#Matriz 15x15:

71ms/step

Mapeamento identidade 15x15 concluído.

[[0.054 0.048 0.046 0.048 0.054 0.072 0.1   0.055 0.056 0.059 0.08  0.055 0.046 0.066 0.049]
 [0.047 0.044 0.041 0.055 0.047 0.059 0.087 0.053 0.051 0.055 0.056 0.048 0.043 0.073 0.046]
 [0.053 0.048 0.046 0.052 0.053 0.071 0.094 0.057 0.056 0.059 0.074 0.055 0.047 0.074 0.05 ]
 [0.104 0.119 0.105 0.609 0.091 0.064 0.25  0.144 0.105 0.167 0.027 0.084 0.122 0.218 0.135]
 [0.053 0.046 0.046 0.035 0.055 0.081 0.087 0.055 0.055 0.054 0.1   0.058 0.045 0.067 0.047]
 [0.078 0.065 0.072 0.023 0.088 0.19  0.069 0.088 0.078 0.062 0.292 0.112 0.065 0.121 0.063]
 [0.1   0.087 0.092 0.188 0.094 0.076 0.401 0.065 0.091 0.131 0.088 0.07  0.08  0.035 0.093]
 [0.054 0.055 0.05  0.077 0.055 0.084 0.053 0.089 0.06  0.06  0.064 0.068 0.057 0.193 0.058]
 [0.05  0.046 0.043 0.047 0.05  0.07  0.078 0.058 0.053 0.055 0.071 0.054 0.045 0.082 0.047]
 [0.056 0.053 0.049 0.104 0.053 0.053 0.144 0.057 0.058 0.073 0.046 0.048 0.051 0.063 0.056]
```

```
[0.088 0.064 0.08  0.009 0.105 0.266 0.084 0.074 0.083 0.058 0.534 0.131 0.062 0.068 0.06 ]
[0.057 0.051 0.051 0.033 0.061 0.108 0.064 0.071 0.061 0.055 0.129 0.073 0.051 0.11  0.052]
[0.048 0.046 0.042 0.056 0.049 0.064 0.079 0.058 0.052 0.056 0.059 0.051 0.045 0.087 0.048]
[0.062 0.073 0.062 0.153 0.063 0.108 0.03  0.162 0.072 0.069 0.055 0.097 0.081 0.504 0.077]
[0.052 0.049 0.045 0.072 0.051 0.061 0.097 0.059 0.055 0.062 0.055 0.051 0.048 0.081 0.052]]
```

Os resultados foram satisfatórios para épocas maiores, mas também ocasionou um aumento no tempo de processamento do código à medida que houve um acréscimo no número de épocas.

4 Código

O presente código pode ser encontrado através do link e temos como referência o livro do Raschka [1] :

Exercício 2 - MLP - XOR e Matriz Identidade (auto-associado) em Python

References

[1] Sebastian Raschka. Machine Learning with Pytorch and Scikit-Learn. 2022.