


▼ Implementação do Algoritmo Backpropagation

A ideia do algoritmo backpropagation é, com base no cálculo do erro ocorrido na camada de saída da rede neural, recalculando o valor dos pesos do vetor w da camada última de neurônios e assim proceder para as camadas anteriores, de trás para a frente (fase *backward*), ou seja, atualizar todos os pesos w das camadas a partir da última até atingir a camada de entrada da rede, para isso realizando a retropropagação do erro obtido pela rede.

A imagem a seguir mostra a nossa rede, com as unidades de entrada marcadas como Input1, Input2 e Input3 (**Input Layer**) conectadas com os *nós* da camada oculta (**Hidden Layer**). Por sua vez as saídas dos *nós* da camada oculta servem como entrada para os *nós* da camada de saída (**Output Layer**). 

O DataSet utilizado para o treinamento da MPL 3x4x2 é o "**Data.csv**", o qual possui informações dispostas em colunas:

- **Input1**: Entrada 1 da MPL.
- **Input2**: Entrada 2 da MPL.
- **Input3**: Entrada 3 da MPL.
- **Output1**: Saída 1 da MPL.
- **Output2**: Saída 2 da MPL.

Bibliotecas

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#Função do cálculo da sigmóide
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

▼ Carregando os dados

Para que uma rede dessas funcione, é preciso treiná-la. O treinamento de uma rede MLP insere-se no contexto de aprendizado de máquina supervisionado, em que cada amostra de dados utilizada apresenta um rótulo informando a que classificação ela se encaixa. Assim, utilizaremos um arquivo Data.csv como dataset para treinamento da nossa MPL.

Vamos começar lendo o arquivo Data.csv em um dataframe do pandas.

```
DataSet=pd.read_csv('Data.csv')
```

✓ 0s conclusão: 22:12



1	0.49	0.85	0.50	0.41	0.81
2	0.86	0.04	0.68	0.35	0.22
3	0.71	0.29	0.30	0.24	0.67
4	0.96	0.78	0.82	0.56	0.89

`DataSet.head()`

	Input1	Input2	Input3	Output1	Output2
0	0.93	0.23	0.73	0.41	0.42
1	0.49	0.85	0.50	0.41	0.81
2	0.86	0.04	0.68	0.35	0.22
3	0.71	0.29	0.30	0.24	0.67
4	0.96	0.78	0.82	0.56	0.89

Váriaveis do *Dataset*

`DataSet.columns`

```
Index(['Input1', 'Input2', 'Input3', 'Output1', 'Output2'], dtype='object')
```

Separando os dados de treinamento e de validação

Agora vamos dividir os dados em um conjunto de treinamento e um conjunto de testes.

Vamos treinar o modelo no conjunto de treinamento, em seguida, usar o conjunto de teste para validar o modelo.

Em nosso exemplo iremos separar de forma randômica 33% dos dados para validação. Estes dados não serão utilizados para determinação dos coeficientes preditores do modelo.

```
from sklearn.model_selection import train_test_split
```

```

N_input = 3
N_hidden = 4
N_output = 2
learnrate = 0.5

```

Inicialização dos pesos da MPL (Aleatório)

```

#Pesos da Camada Oculta (Inicialização Aleatória)
weights_input_hidden = np.random.normal(0, scale=0.1, size=(N_input, N_hidden))
print('Pesos da Camada Oculta:')
print(weights_input_hidden)

#Pesos da Camada de Saída (Inicialização Aleatória)
weights_hidden_output = np.random.normal(0, scale=0.1, size=(N_hidden, N_output))
print('Pesos da Camada de Saída:')
print(weights_hidden_output)

```

```

Pesos da Camada Oculta:
[[ 0.00382119  0.07776583 -0.16405714 -0.02603675]
 [ 0.06230804  0.14077199  0.04212939 -0.06926527]
 [ 0.02251131  0.04672114  0.05940119  0.0297294 ]]
Pesos da Camada de Saída:
[[-0.02384312 -0.03271044]
 [-0.00116809  0.1463324 ]
 [-0.07443877 -0.07459082]
 [-0.1450492  -0.2306215 ]]

```

Algoritmo Backpropagation

```

epochs = 5000
last_loss=None
EvolucaoError=[]
IndiceError=[]

for e in range(epochs):
    delta_w_i_h = np.zeros(weights_input_hidden.shape)
    delta_w_h_o = np.zeros(weights_hidden_output.shape)

```

```

# backward pass
    ## TODO: Cálculo do Erro
    error = yi - output

    # TODO: Calcule o termo de erro de saída (Gradiente da Camada de Saída)
    output_error_term = error * output * (1 - output)

    # TODO: Calcule a contribuição da camada oculta para o erro
    hidden_error = np.dot(weights_hidden_output,output_error_term)

    # TODO: Calcule o termo de erro da camada oculta (Gradiente da Camada Oc
    hidden_error_term = hidden_error * hidden_layer_output * (1 - hidden_lay

    # TODO: Calcule a variação do peso da camada de saída
    delta_w_h_o += output_error_term*hidden_layer_output[:, None]

    # TODO: Calcule a variação do peso da camada oculta
    delta_w_i_h += hidden_error_term * xi[:, None]

#Atualização dos pesos na época em questão
weights_input_hidden += learnrate * delta_w_i_h / n_records
weights_hidden_output += learnrate * delta_w_h_o / n_records

# Imprimir o erro quadrático médio no conjunto de treinamento

if e % (epochs / 20) == 0:
    hidden_output = sigmoid(np.dot(xi, weights_input_hidden))
    out = sigmoid(np.dot(hidden_output,
                          weights_hidden_output))
    loss = np.mean((out - yi) ** 2)

    if last_loss and last_loss < loss:
        print("Erro quadrático no treinamento: ", loss, " Atenção: O erro es
    else:
        print("Erro quadrático no treinamento: ", loss)

```

Gráfico da Evolução do Erro

```
plt.plot(IndiceError, EvolucaoError, 'r') # 'r' is the color red
plt.xlabel('')
plt.ylabel('Erro Quadrático')
plt.title('Evolução do Erro no treinamento da MPL')
plt.show()
```



Validação do modelo

```
MSE_output2/=n_records
```

```
print('Erro Quadrático Médio da Saída Output1 é: ',MSE_output1)  
print('Erro Quadrático Médio da Saída Output2 é: ',MSE_output2)
```

```
Erro Quadrático Médio da Saída Output1 é:  0.010986807501155889  
Erro Quadrático Médio da Saída Output2 é:  0.004380918304143488
```

Métricas de Avaliação

Aqui estão três métricas comuns de avaliação para problemas de regressão:

- **Erro Médio Absoluto (MAE):** é a média do valor absoluto dos erros.
- **Erro Quadrático Médio (MSE):** é a média do quadrado dos erros.
- **Raiz do Erro Médio Quadrático (RMSE):** é a raiz da média do quadrado dos erros.

O objetivo é sempre minimizar estas funções de Erro.

