


▼ 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 camada 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 o 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.



	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

```
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.

```
#Tamanho do DataSet de Treinamento
n_records, n_features = X_train.shape

#Arquitetura da MPL
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:
[[ 0.06437303 -0.13626963 -0.0183772  -0.05371014]
 [-0.0919449  -0.09168483 -0.19670838  0.07139244]
 [ 0.08186578 -0.03777275  0.02786526 -0.0461809  ]]
Pesos da Camada de Saída:
[[-0.01966511 -0.01092425]
 [ 0.05457469  0.1447764 ]
 [-0.17469743 -0.03738838]
 [ 0.08715709  0.12952557]]
```

Algoritmo Backpropagation

```
output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)

#Aplicado a função de ativação
output = sigmoid(output_layer_in)
#print('As saídas da rede são',output)
#-----

# 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 Oculta)
hidden_error_term = hidden_error * hidden_layer_output * (1 - hidden_layer_output)

# 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]

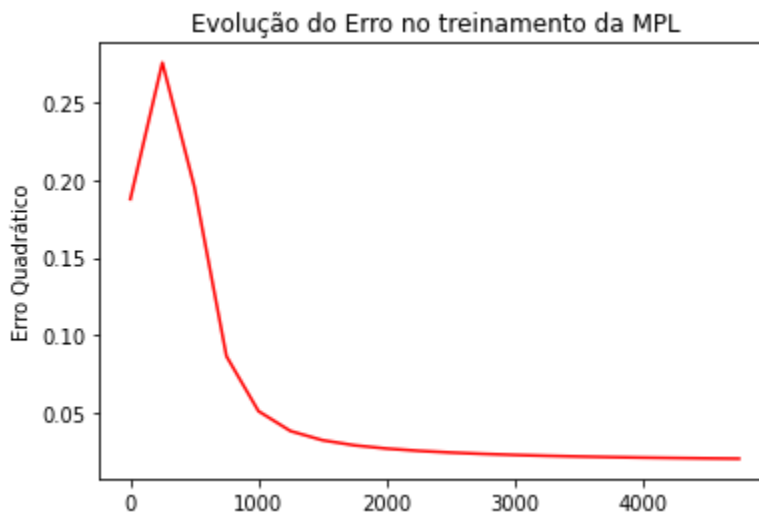
#Atualização dos pesos na época em questão
weights_input_hidden += learnrate * delta_w_i_h / n_records
weights_hidden_output += learnrate * delta_w_h_o / n_records

# Imprimir o erro quadrático médio no conjunto de treinamento
```

```
Erro quadrático no treinamento: 0.023211515809159005
Erro quadrático no treinamento: 0.02411268020836287
Erro quadrático no treinamento: 0.023253440025671243
Erro quadrático no treinamento: 0.022564488493381503
Erro quadrático no treinamento: 0.022003936330157552
Erro quadrático no treinamento: 0.02154362430804599
Erro quadrático no treinamento: 0.021162815949794644
Erro quadrático no treinamento: 0.020845258278021327
Erro quadrático no treinamento: 0.020577810602487995
Erro quadrático no treinamento: 0.020349775575473496
Erro quadrático no treinamento: 0.020152509682942677
```

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()
```



```
output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)

#Aplicado a função de ativação
output = sigmoid(output_layer_in)

#-----

#Cálculo do Erro
## TODO: Cálculo do Erro
error = yi - output
MSE_Output1 += (yi[0] - output[0])**2
MSE_Output2 += (yi[1] - output[1])**2

#Erro Quadrático Médio
MSE_Output1/=n_records
MSE_Output2/=n_records

print('Erro Quadrático Médio da Saída Output1 é: ',MSE_Output1)
print('Erro Quadrático Médio da Saída Output2 é: ',MSE_Output2)
```

