In [1]:

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

Função de ativação e sua derivada

In [2]:

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

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

Definindo o dataset

In [3]:

```
DataSet=pd.read_csv('arruela.csv')
DataSet.head()
```

Out[3]:

	Hora	Tamanho	Referencia	NumAmostra	Area	Delta	Output1	Output2
0	17:56:39	53	25	69	81	68	1	0
1	17:56:41	53	26	89	87	56	1	0
2	17:56:52	53	27	68	69	55	1	0
3	17:56:55	53	28	36	50	80	1	0
4	17:56:58	53	29	71	72	50	1	0

In [4]:

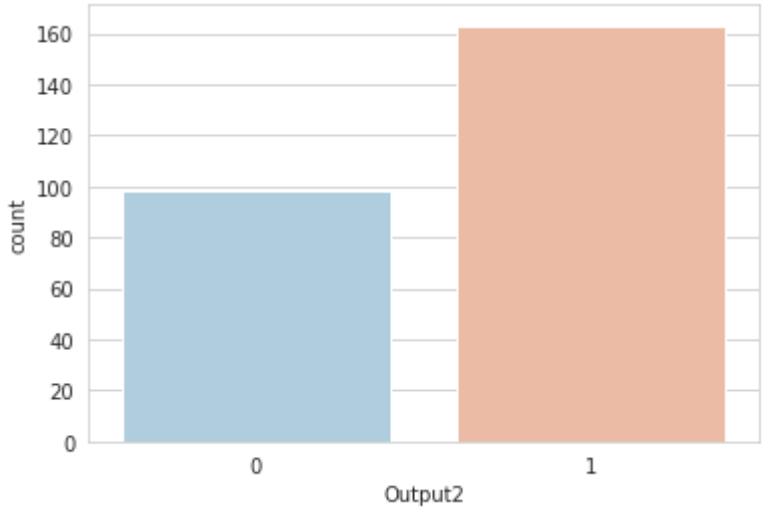
```
DataSet.drop(['Hora','Tamanho','Referencia'],axis=1,inplace=True)
DataSet.head()
```

Out[4]:

	NumAmostra	Area	Delta	Output1	Output2
0	69	81	68	1	0
1	89	87	56	1	0
2	68	69	55	1	0
3	36	50	80	1	0
4	71	72	50	1	0

In [5]:

```
sns.set_style('whitegrid')
sns.countplot(x='Output2',data=DataSet,palette='RdBu_r')
plt.show()
```



Separando os dados para validação e treinamento

In [6]:

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

In [7]:

```
#selecionando randomicamente 30% do conjunto total de dados para os testes e o resto pa
ra fazer o treinamento supervisionado
X_train, X_test, Y_train, y_test = train_test_split(DataSet.drop(['Output1', 'Output2'
],axis=1), DataSet[['Output1', 'Output2']], test_size=0.30, random_state=101)
#print(X_train)#entradas
#print(y_test)#saídas
```

Configuração da MPL

In [8]:

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

#Arquitetura da MPL
N_input = 3 #num neuronios de entrada
N_hidden = 5 #num neuronios na camada oculta 1
N_hidden_2 = 5 #num neuronios na camada oculta 2
N_output = 2 #num de saidas sinalizando uma ou duas peças
N = 0.1 #taxa de aprendizado
```

Inicialização dos pesos da MPL (Aleatório)

In [9]:

```
#Pesos da Camada Oculta (Inicialização Aleatória)
w_hidden_layer_1 = np.random.normal(0, scale=0.1, size=(N_input, N_hidden))
print('Pesos da Primeira Camada Oculta:')
print(w_hidden_layer_1,'\n')

#Pesos da Segunda Camada Oculta (Inicialização Aleatória)
w_hidden_layer_2 = np.random.normal(0, scale=0.1, size=(N_hidden, N_hidden_2))
print('Pesos da Segunda Camada Oculta:')
print(w_hidden_layer_2,'\n')

#Pesos da Camada de Saída (Inicialização Aleatória)
w_output_layer = np.random.normal(0, scale=0.1, size=(N_hidden_2, N_output))
print('Pesos da Camada de Saída:')
print(w_output_layer)
```

```
Pesos da Primeira Camada Oculta:
[[ 0.04046384 -0.05594798 -0.24485347  0.01653624  0.04185853]
 [ 0.18378658 -0.02566283 -0.19929081  0.04712941 -0.06087608]
 [-0.01583966 -0.08128932  0.00660977 -0.08681393  0.07780869]]

Pesos da Segunda Camada Oculta:
[[ 0.00822845  0.10331953 -0.20313476 -0.05798073 -0.03019671]
 [-0.09187643  0.12212762 -0.09268177 -0.09150123  0.05285974]
 [ 0.02872155  0.0304913  -0.06110498 -0.02964616  0.04909495]
 [ 0.07568319  0.04494204  0.05313464  0.11986361  0.10057469]
 [-0.02597454 -0.17700388  0.04879387 -0.07130941  0.12003141]]

Pesos da Camada de Saída:
[[ 0.03274233  0.00264324]
 [-0.01062858 -0.03102634]
 [ 0.06636754 -0.06447654]
 [-0.05783332  0.01855802]
 [-0.09875077 -0.14905813]]
```

Treino da rede

In [10]:

```
#Algoritmo Backpropagation
epochs = 30000
last_loss=None
EvolucaoError=[]
IndiceError=[]
# Calcule a precisão dos dados de teste
n_records, n_features = X_test.shape
MSE_Output1=0
MSE_Output2=0
predicao=0

for e in range(epochs):
    delta_hidden_layer_1 = np.zeros(w_hidden_layer_1.shape)
    delta_hidden_layer_2 = np.zeros(w_hidden_layer_2.shape)
    delta_output_layer = np.zeros(w_output_layer.shape)

    for X, Y in zip(X_train.values, Y_train.values):

        activation_hidden_layer_1 = sigmoid(np.dot(X, w_hidden_layer_1))
        activation_hidden_layer_2 = sigmoid(np.dot(activation_hidden_layer_1, w_hidden_
layer_2))
        activation_output_layer = sigmoid(np.dot(activation_hidden_layer_2, w_output_la
yer))

        delta_output_layer = (Y - activation_output_layer) * sigmoid_derivative(activat
ion_output_layer)
        delta_hidden_layer_2 = np.dot(delta_output_layer, w_output_layer.T) * sigmoid_d
erivative(activation_hidden_layer_2)
        delta_hidden_layer_1 = np.dot(delta_hidden_layer_2, w_hidden_layer_2.T) * sigmo
id_derivative(activation_hidden_layer_1)

    #weights_input_hidden += learnrate * delta_w_i_h / n_records
    w_output_layer += N * delta_output_layer / n_records
    w_hidden_layer_2 += N * delta_hidden_layer_2 / n_records
    w_hidden_layer_1 += N * delta_hidden_layer_1 / n_records

    if  e % (epochs / 20) == 0:

        hidden_output = sigmoid(np.dot(X, w_hidden_layer_1))
        #print("hidden_output",hidden_output)
        hidden_output_2 = sigmoid(np.dot(w_hidden_layer_1, w_hidden_layer_2))
        #print("hidden_output_2",hidden_output_2)
        out = sigmoid(np.dot(w_hidden_layer_2, w_output_layer))
        #print("out",out)

        loss = np.mean((out - Y) ** 2)

        if last_loss and last_loss < loss:
            print("Erro quadrático no treinamento: ", loss, " Atenção: O erro está aume
ntando")
        else:
            print("Erro quadrático no treinamento: ", loss)
        last_loss = loss

        EvolucaoError.append(loss)
        IndiceError.append(e)
        #'''
```

```
Erro quadrático no treinamento:  0.25016501131443153
Erro quadrático no treinamento:  0.24695979914902694
Erro quadrático no treinamento:  0.23696030866381554
Erro quadrático no treinamento:  0.22372987935484834
Erro quadrático no treinamento:  0.21026376618269738
Erro quadrático no treinamento:  0.19776826193327804
Erro quadrático no treinamento:  0.18655375925221346
Erro quadrático no treinamento:  0.17659214255254083
Erro quadrático no treinamento:  0.1677501904762772
Erro quadrático no treinamento:  0.15987649087902917
Erro quadrático no treinamento:  0.15283088822057347
Erro quadrático no treinamento:  0.1464923896650568
Erro quadrático no treinamento:  0.1407593568386367
Erro quadrático no treinamento:  0.13554724170775329
Erro quadrático no treinamento:  0.13078582684383183
Erro quadrático no treinamento:  0.1264166549977852
Erro quadrático no treinamento:  0.12239084251315843
Erro quadrático no treinamento:  0.1186672892636799
Erro quadrático no treinamento:  0.11521123759064578
Erro quadrático no treinamento:  0.11199311952534378
```

Gráfico da Evolução do Erro

In [11]:

```
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()
```

In [ ]:

```
#Pesos da Camada Oculta
print(w_hidden_layer_1)
print()
print(w_hidden_layer_2)

#Pesos da Camada de Saída
print()
print(w_output_layer)
```

Validação do Modelo

In [ ]:

```
# Calcule a precisão dos dados de teste
n_records, n_features = X_test.shape
MSE_Output1=0
MSE_Output2=0
predicao=0

for xi, yi in zip(X_test.values, y_test.values):

# Forward Pass
    '''
    activation_hidden_layer_1 = sigmoid(np.dot(xi, w_hidden_layer_1))
    activation_hidden_layer_2 = sigmoid(np.dot(activation_hidden_layer_1, w_hidden_laye
r_2))
    output = sigmoid(np.dot(activation_hidden_layer_2, w_output_layer))
    #delta_output_layer = (yi - activation_output_layer) * sigmoid_derivative(activatio
n_output_layer)
    #delta_hidden_layer_2 = np.dot(delta_output_layer, w_output_layer.T) * sigmoid_deri
vative(activation_hidden_layer_2)
    #delta_hidden_layer_1 = np.dot(delta_hidden_layer_2, w_hidden_layer_2.T) * sigmoid_
derivative(activation_hidden_layer_1)
    '''
    #Camada oculta 1
    #Calcule a combinação linear de entradas e pesos sinápticos
    n_neurons_hidden_layer_1 = np.dot(xi, w_hidden_layer_1)
    #Aplicado a função de ativação
    hidden_layer_output_1 = sigmoid(n_neurons_hidden_layer_1)

    #Camada oculta 2
    #Calcule a combinação linear de entradas e pesos sinápticos
    n_neurons_hidden_layer_2 = np.dot(hidden_layer_output_1, w_hidden_layer_2)
    #Aplicado a função de ativação
    hidden_layer_output_2 = sigmoid(n_neurons_hidden_layer_2)

    #Camada de Saída
    #Calcule a combinação linear de entradas e pesos sinápticos
    output_layer_in = np.dot(hidden_layer_output_2, w_output_layer)
    #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

#calculando o erro de predição
    if(output[0]>output[1]):
        if(yi[0]>yi[1]):
            predicao+=1
    if(output[1]>output[0]):
        if(yi[1]>yi[0]):
            predicao+=1

#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)
print("A acuracia da predicao é:{:.3f}".format(predicao/n_records))
```

In [ ]:

```
```

In [ ]:

```
```