

▼ Rede Neural Multicamadas (MPL)

Uma rede MPL é uma classe de rede neural artificial *feedforward* (ANN). Um MLP consiste em pelo menos três camadas de nós: uma camada de entrada, uma camada oculta e uma camada de saída. Exceto para os nós de entrada, cada nó é um neurônio que usa uma função de ativação não linear. O MLP utiliza uma técnica de aprendizado supervisionado chamada *backpropagation* para treinamento.

Implementando uma RNA multicamadas


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**). 

Diagrama de uma MPL

Lembrando que em cada nó temos:

$$f(h) = \text{sigmoid}(h) = \frac{1}{1 + e^{-h}}$$

onde

$$h = \frac{1}{n} \sum_{i=1}^n (w_i * x_i) + b$$

▼ Configuração da MPL

```
#Importando a biblioteca
import numpy as np

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

#Arquitetura da MPL
N_input = 3
N_hidden = 4
N_output = 2

#Vetor dos valores de entrada
x = np.array([0.2, 0.12, 0.26])
```

```
[-0.07, 0.04, -0.05, 0.05]])
```

```
#Pesos da Camada de Saída
```

```
weights_hidden_out = np.array([[ -0.18, 0.11],  
                                [ -0.09, 0.05],  
                                [ -0.04, 0.05],  
                                [ -0.02, 0.07]])
```

Forward Pass

```
#Camada oculta
```

```
#Calcule a combinação linear de entradas e pesos sinápticos
```

```
hidden_layer_input = np.dot(x, weights_in_hidden)
```

```
#Aplicado a função de ativação
```

```
hidden_layer_output = sigmoid(hidden_layer_input)
```

```
#Camada de Saída
```

```
#Calcule a combinação linear de entradas e pesos sinápticos
```

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

```
#Aplicado a função de ativação
```

```
output = sigmoid(output_layer_in)
```

```
print('As saídas da rede são',output)
```

```
As saídas da rede são [0.45871946 0.53501657]
```

Backward Pass

```
## TODO: Cálculo do Erro
```

```
error = target - output
```

```
#print('Erro da Rede: ',error)
```

```

delta_w_h_o = learnrate * output_error_term*hidden_layer_output[:, None]
print('delta_w_h_o: ',delta_w_h_o)

# TODO: Calcule a variação do peso da camada oculta
delta_w_i_h = learnrate * hidden_error_term * x[:, None]
print('delta_w_i_h: ',delta_w_i_h)

delta_w_h_o: [[-0.00609329 -0.01577075]
 [-0.00624616 -0.01616643]
 [-0.00617724 -0.01598804]
 [-0.00611443 -0.01582547]]
delta_w_i_h: [[-6.41595715e-05 -2.41416349e-05 -5.47866254e-05 -9.87662420e-05]
 [-4.17037215e-05 -1.56920627e-05 -3.56113065e-05 -6.41980573e-05]
 [ 8.34074430e-05  3.13841253e-05  7.12226130e-05  1.28396115e-04]]

```

Atualização dos Pesos

```

weights_input_hidden = learnrate * delta_w_i_h
print('weights_input_hidden: ',weights_input_hidden)
weights_hidden_output = learnrate * delta_w_h_o
print('weights_hidden_output: ',weights_hidden_output)

weights_input_hidden: [[-3.20797858e-05 -1.20708174e-05 -2.73933127e-05 -4.91
 [-2.08518608e-05 -7.84603133e-06 -1.78056532e-05 -3.20990287e-05]
 [ 4.17037215e-05  1.56920627e-05  3.56113065e-05  6.41980573e-05]]
weights_hidden_output: [[-0.00304664 -0.00788538]
 [-0.00312308 -0.00808321]
 [-0.00308862 -0.00799402]
 [-0.00305721 -0.00791273]]

```

