Gradiente Descendente

O gradiente descendente é um algoritmo de otimização que usa as derivadas da função objetivo para encontrar o ponto com maior inclinação. No processo, as variáveis a otimizar são deslocadas em uma direção negativa o qual reduzirá o valor da função objetivo.

Algoritmo geral para atualizar os pesos com gradiente descendente:

- Defina o passo como zero: $\Delta w_i = 0$
- Para cada observação nos dados de treinamento:
  - Passe adiante na rede, calculando o output $\hat{y} = f(\sum_i w_i x_i)$
  - Calcule o erro para aquela unidade de output, $\delta = (y - \hat{y}) * f'(\sum_i w_i x_i)$
  - Atualize o passo $\Delta w_i = \Delta w_i + \delta x_i$
- Atualize os pesos $w_i = w_i + \eta \Delta w_i / m$ onde η é a taxa de aprendizado e m é o número de observações. Aqui estamos tirando a média dos passos para ajudar a reduzir quaisquer variações nos dados de treinamento
- Repita por e rodadas.

Vamos implementar o algoritmo do Gradiente Descendente!

Importando a biblioteca

In [ ]:

```
import numpy as np
```

Função do cáculo da sigmóide

In [ ]:

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

Derivada da função sigmóide

In [ ]:

```
def sigmoid_prime(x):
    return sigmoid(x) * (1 - sigmoid(x))
```

Vetor dos valores de entrada e saídas

In [ ]:

```
x = np.array([1, 3, 4, 8])
y = np.array(0.5)
b= 0.5
```

Pesos iniciais das ligações sinápticas

Nota: Inicializados aleatóriamente

In [ ]:

```
w = np.random.randn(4)/10
print(w)
```

Taxa de Aprendizagem

In [ ]:

```
learnrate = 0.5
```

Calcule um degrau de descida gradiente para cada peso

Critério de parada

- **Epochs**: Número de Épocas
- **MinError**: Erro mínimo estipulado

In [ ]:

```
#Número de Épocas
epochs=100

#Inicilizando del_w
del_w=0

for e in range(epochs):
    # TODO: Calcule a combinação linear de entradas e pesos sinápticos
    h = np.dot(x, w)

    # TODO: Calcule a saída da Rede Neural
    output = sigmoid(h)

    # TODO: Calcule o erro da Rede Neural
    error = y - output

    # TODO: Calcule o termo de erro
    error_term = error * sigmoid_prime(h)

    # TODO: Calcule a variação do peso
    del_w = learnrate * error_term * x

    # TODO: Atualização do Peso
    w = w + del_w

#    print(w)
#    print(output)
#   print(error)
```

In [ ]:

```
print('Saída da Rede Neural:')
print(output)
print('Erro:',error)
```