Gradient Descent: The Code

From before we saw that one weight update can be calculated as:

$\Delta w_i = \alpha * \delta * x_i$

where α is the learning rate and δ is the error term.

Previously, we utilized the loss function for logistic regression, which was because we were performing a binary classification task. This time we'll try to get the function to learn a value instead of a class. Therefore, we'll use a simpler loss function, as defined below in the error term δ.

$\delta = (y - \hat{y}) f'(h) = (y - \hat{y}) f'(\sum w_i x_i)$

Note that $f'(h)$ is the derivative of the activation function $f(h)$, and h is defined as the output, which in the case of a neural network is a sum of the weights times the inputs.

Now I'll write this out in code for the case of only one output unit. We'll also be using the sigmoid as the activation function $f(h)$.

Gradient Descent: The Code

From before we saw that one weight update can be calculated as:

$\Delta w_i = \alpha * \delta * x_i$

where α is the learning rate and δ is the error term.

Previously, we utilized the loss function for logistic regression, which was because we were performing a binary classification task. This time we'll try to get the function to learn a value instead of a class. Therefore, we'll use a simpler loss function, as defined below in the error term δ.

$\delta = (y - \hat{y}) f'(h) = (y - \hat{y}) f'(\sum w_i x_i)$

Note that $f'(h)$ is the derivative of the activation function $f(h)$, and h is defined as the output, which in the case of a neural network is a sum of the weights times the inputs.

Now I'll write this out in code for the case of only one output unit. We'll also be using the sigmoid as the activation function $f(h)$.

So, the derivative of the sigmoid function is

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Derivative of the Sigmoid Function

In [2]:

```
import numpy as np

#Sigmoid Function
def sigmoid(x):
    return 1/(1+np.exp(-x))

# Derivative of Sigmoid Function
def sigmoid_derivative(x):
    return sigmoid(x)*(1-sigmoid(x))

# Input data
x = np.array([0.1, 0.3])
```

```
# Target
y = 0.2

# Input to output weights
weights = np.array([-0.8, 0.5])

# The learning rate
learnrate = 0.5

#Output of neural network y_hat
# nn_output = sigmoid(x[0]*weights[0] + x[1]*weights[1])
nn_output = sigmoid(np.dot(x,weights))

#Output Error (y - y_hat)
error = y - nn_output

#Error term δ
error_term = error * sigmoid_derivative(np.dot(x,weights))

#Gradient descent step
del_w = [learnrate * error_term * x[0], learnrate * error_term * x[1]]
# or del_w = learnrate * error_term * x

print('Neural Network output:')
print(nn_output)
print('Amount of Error:')
print(error)
print('Change in Weights:')
print(del_w)
```

```
Neural Network output:
0.5174928576663897
Amount of Error:
-0.31749285766638974
Change in Weights:
[-0.003963803079006883, -0.011891409237020648]
```

In [3]:

```
import numpy as np

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

learnrate = 0.5
x = np.array([1, 2])
y = np.array(0.5)

# Initial weights
w = np.array([0.5, -0.5])

# Calculate one gradient descent step for each weight
# Calculate output of neural network
nn_output = sigmoid(np.dot(x, w))

#Calculate error of neural network
error = y - nn_output

# Calculate change in weights
del_w = learnrate * error * nn_output * (1 - nn_output) * x
```

```
print('Neural Network output:')
print(nn_output)
print('Amount of Error:')
print(error)
print('Change in Weights:')
print(del_w)
```

```
Neural Network output:
0.3775406687981454
Amount of Error:
0.1224593312018546
Change in Weights:
[0.0143892 0.0287784]
```

In [4]:

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

# Derivative of Sigmoid Function
def sigmoid_derivative(x):
    return sigmoid(x)*(1-sigmoid(x))


learnrate = 0.5
x = np.array([1, 2])
y = np.array(0.5)

# Initial weights
weights = np.array([0.5, -0.5])


# The learning rate
learnrate = 0.5

#Output of neural network y_hat
# nn_output = sigmoid(x[0]*weights[0] + x[1]*weights[1])
nn_output = sigmoid(np.dot(x,weights))

#Output Error (y - y_hat)
error = y - nn_output

#Error term δ
error_term = error * sigmoid_derivative(np.dot(x,weights))

#Gradient descent step
del_w = [learnrate * error_term * x[0], learnrate * error_term * x[1]]
# or del_w = learnrate * error_term * x

print('Neural Network output:')
print(nn_output)
print('Amount of Error:')
print(error)
print('Change in Weights:')
print(del_w)
```

```
Neural Network output:
0.3775406687981454
Amount of Error:
0.1224593312018546
```

Change in Weights:
[0.01438919871308019, 0.02877839742616038]