Gradient Descent: The Code

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

$$\Delta w_i = \eta \, \delta x_i$$

with the error term δ as

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

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 $f(h)$.

```
# Defining the sigmoid function for activations
def sigmoid(x):
    return 1/(1+np.exp(-x))

# Derivative of the sigmoid function
def sigmoid_prime(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, eta in the weight step equation
learnrate = 0.5

# The neural network output (y-hat)
nn_output = sigmoid(x[0]*weights[0] + x[1]*weights[1])
# or nn_output = sigmoid(np.dot(x, weights))
# output error (y - y-hat)
error = y - nn_output
```

```
# error term (lowercase delta)
error_term = error * sigmoid_prime(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
```

gradient.py soultion.py

```
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
# TODO: Calculate output of neural network
nn_output = sigmoid(np.dot(x, w))

# TODO: Calculate error of neural network
error = y - nn_output

# TODO: 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:
0.377540668798
Amount of Error:
```

```
0.122459331202
Change in Weights:
[ 0.0143892  0.0287784]
```

```
Nice job!  That's right!
```

RESET QUIZ TEST RUN SUBMIT ANSWER

NEXT