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

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

Remember, in the above equation $(y - \hat{y})$ is the output error, and $f'(h)$ refers to the derivative of the activation function, $f(h)$. We'll call that derivative the output gradient.

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 linear combination performed by the node (h in f(h) and f'(h))
h = x[0]*weights[0] + x[1]*weights[1]
# or h = np.dot(x, weights)

# The neural network output (y-hat)
nn_output = sigmoid(h)

# output error (y - y-hat)
error = y - nn_output

# output gradient (f'(h))
output_grad = sigmoid_prime(h)

# error term (lowercase delta)
error_term = error * output_grad

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

Note: If you are wondering where the derivative of the `sigmoid` function comes from (`sigmoid_prime` above), check out the derivation in this post.

In the quiz below, you'll implement gradient descent in code yourself, although with a few differences (which we'll leave to you to figure out!) from the above example.

gradient.py | solution.py

```
21
22 ### Calculate one gradient descent step for each weight
23 ### Note: Some steps have been consolidated, so there are
24 ###       fewer variable names than in the above sample code
25
26 # TODO: Calculate the node's linear combination of inputs and weights
27 h = np.dot(x, w)
28
29 # TODO: Calculate output of neural network
30 nn_output = sigmoid(h)
31
32 # TODO: Calculate error of neural network
33 error = y - nn_output
34
35 # TODO: Calculate the error term
36 #       Remember, this requires the output gradient, which we haven't
37 #       specifically added a variable for.
38 error_term = error * sigmoid_prime(h)
39 # Note: The sigmoid_prime function calculates sigmoid(h) twice,
40 #       but you've already calculated it once. You can make this
41 #       code more efficient by calculating the derivative directly
42 #       rather than calling sigmoid_prime, like this:
```

```
43 # error_term = error * nn_output * (1 - nn_output)
44
45 # TODO: Calculate change in weights
46 del_w = learnrate * error_term * x
47
```

```
Neural Network output:
0.6899744811276125
Amount of Error:
-0.1899744811276125
Change in Weights:
[-0.02031869 -0.04063738 -0.06095608 -0.08127477]

Nice job!  That's right!
```

RESET QUIZ TEST RUN SUBMIT ANSWER

NEXT