```
hidden_error*inputs
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-3b59121cb809> in <module>()
----> 1 hidden_error*x

ValueError: operands could not be broadcast together with shapes (3,) (6,)
```

Also, w_{ij} is a matrix now, so the right side of the assignment must have the same shape as the left side. Luckily, NumPy takes care of this for us. If you multiply a row vector array with a column vector array, it will multiply the first element in the column by each element in the row vector and set that as the first row in a new 2D array. This continues for each element in the column vector, so you get a 2D array that has shape `(len(column_vector), len(row_vector))`.

```
hidden_error*inputs[:,None]
array([[ -8.24195994e-04,  -2.71771975e-04,   1.29713395e-03],
       [ -2.87777394e-04,  -9.48922722e-05,   4.52909055e-04],
       [  6.44605731e-04,   2.12553536e-04,  -1.01449168e-03],
       [  0.00000000e+00,   0.00000000e+00,  -0.00000000e+00],
       [  0.00000000e+00,   0.00000000e+00,  -0.00000000e+00],
       [  0.00000000e+00,   0.00000000e+00,  -0.00000000e+00]])
```

It turns out this is exactly how we want to calculate the weight update step. As before, if you have your inputs as a 2D array with one row, you can also do `hidden_error*inputs.T`, but that won't work if `inputs` is a 1D array.

Backpropagation exercise

Below, you'll implement the code to calculate one backpropagation update step for two sets of weights. I wrote the forward pass - your goal is to code the backward pass.

Things to do

- Calculate the network's output error.
- Calculate the output layer's error term.
- Use backpropagation to calculate the hidden layer's error term.
- Calculate the change in weights (the delta weights) that result from propagating the errors back through the network