which conveniently calculates h for us. The dot product multiplies two arrays element-wise, the first element in array 1 is multiplied by the first element in array 2, and so on. Then, each product is summed.

```
# input to the output layer
output_in = np.dot(weights, inputs)
```

And finally, we can update Δw_i and w_i by incrementing them with `weights += ...` which is shorthand for `weights = weights + ...`.

Efficiency tip!

You can save some calculations since we're using a sigmoid here. For the sigmoid function, $f'(h) = f(h)(1 - f(h))$. That means that once you calculate $f(h)$, the activation of the output unit, you can use it to calculate the gradient for the error gradient.

Programming exercise

Below, you'll implement gradient descent and train the network on the admissions data. Your goal here is to train the network until you reach a minimum in the mean square error (MSE) on the training set. You need to implement:

- The network output: `output`.
- The output error: `error`.
- The error term: `error_term`.
- Update the weight step: `del_w +=`.
- Update the weights: `weights +=`.

After you've written these parts, run the training by pressing "Test Run". The MSE will print out, as well as the accuracy on a test set, the fraction of correctly predicted admissions.

Feel free to play with the hyperparameters and see how it changes the MSE.