Here's my solution to the last quiz.

```
class Sigmoid(Node):
    """
    Represents a node that performs the sigmoid activation function.
    """
    def __init__(self, node):
        # The base class constructor.
        Node.__init__(self, [node])

    def _sigmoid(self, x):
        """
        This method is separate from `forward` because it
        will be used with `backward` as well.

        `x`: A numpy array-like object.
        """
        return 1. / (1. + np.exp(-x))

    def forward(self):
        """
        Perform the sigmoid function and set the value.
        """
        input_value = self.inbound_nodes[0].value
        self.value = self._sigmoid(input_value)

    def backward(self):
        """
        Calculates the gradient using the derivative of
        the sigmoid function.
        """
        # Initialize the gradients to 0.
        self.gradients = {n: np.zeros_like(n.value) for n in self.inbound_n
        # Sum the derivative with respect to the input over all the outputs
        for n in self.outbound_nodes:
            grad_cost = n.gradients[self]
```

```
        sigmoid = self.value
        self.gradients[self.inbound_nodes[0]] += sigmoid * (1 - sigmoid
```

The `backward` method sums the derivative (it's a normal derivative when there's only one variable) with respect to the only input over all the output nodes. The last line implements the derivative, $\frac{\partial sigmoid}{\partial x} \frac{\partial cost}{\partial sigmoid}$.

Replacing the math expression with code:

$\frac{\partial sigmoid}{\partial x}$ is `sigmoid * (1 - sigmoid)` and $\frac{\partial cost}{\partial sigmoid}$ is `grad_cost`.

Now that you have the gradient of the cost with respect to each input (the return value from `forward_and_backward()`) your network can start learning! To do so, you will implement a technique called **Stochastic Gradient Descent**.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a version of Gradient Descent where on each forward pass a batch of data is randomly sampled from total dataset. Remember when we talked about the batch size earlier? That's the size of the batch. Ideally, the entire dataset would be fed into the neural network on each forward pass, but in practice, it's not practical due to memory constraints. SGD is an approximation of Gradient Descent, the more batches processed by the neural network, the better the approximation.

A naïve implementation of SGD involves:

1. Randomly sample a batch of data from the total dataset.
2. Running the network forward and backward to calculate the gradient (with data from (1)).
3. Apply the gradient descent update.
4. Repeat steps 1-3 until convergence or the loop is stopped by another mechanism (i.e. the number of epochs).

If all goes well, the network's loss should generally trend downwards, indicating more useful weights and biases over time.

So far, MiniFlow can already do step 2. In the following quiz, steps 1 and 4 are already implemented. It will be your job to implement step 3.

As a reminder, here's the gradient descent update equation, where αrepresents the learning rate:

$$x = x - \alpha * \frac{\partial cost}{\partial x}$$

We're also going to use an actual dataset for this quiz, the **Boston Housing dataset**. After training the network will be able to predict prices of Boston housing!

Boston's Back Bay

By Robbie Shade (Flickr: Boston's Back Bay) [CC BY 2.0 (**http://creativecommons.org/licenses/by/2.0**)], via Wikimedia Commons

Each example in the dataset is a description of a house in the Boston suburbs, the description consists of 13 numerical values (features). Each example also has an associated price. With SGD, we're going to minimize the MSE between the actual price and the price predicted by the neural network based on the features.

If all goes well the output should look something like this:

When the batch size is 11:

```
Total number of examples = 506
Epoch: 1, Loss: 140.256
Epoch: 2, Loss: 34.570
```

```
Epoch: 3, Loss: 27.501
Epoch: 4, Loss: 25.343
Epoch: 5, Loss: 20.421
Epoch: 6, Loss: 17.600
Epoch: 7, Loss: 18.176
Epoch: 8, Loss: 16.812
Epoch: 9, Loss: 15.531
Epoch: 10, Loss: 16.429
```

When the batch size is the same as the total number of examples (batch is the whole dataset):

```
Total number of examples = 506
Epoch: 1, Loss: 646.134
Epoch: 2, Loss: 587.867
Epoch: 3, Loss: 510.707
Epoch: 4, Loss: 446.558
Epoch: 5, Loss: 407.695
Epoch: 6, Loss: 324.440
Epoch: 7, Loss: 295.542
Epoch: 8, Loss: 251.599
Epoch: 9, Loss: 219.888
Epoch: 10, Loss: 216.155
```

Notice the *cost* or *loss* trending towards 0.

Instructions

1. Open `nn.py`. See how the network runs with this new architecture.
2. Find the `sgd_update` method in `miniflow.py` and implement SGD.
3. Test your network! Does your loss decrease with more epochs?

Note! The virtual machines on which we run your code have time limits. If your network takes more than 10 seconds to run, you will get a timeout error. Keep this in mind as you play with the number of epochs.