MiniFlow Architecture

Let's consider how to implement this graph structure in MiniFlow. We'll use a Python class to represent a generic node.

```
class Node(object):
    def __init__(self):
        # Properties will go here!
```

We know that each node might receive input from multiple other nodes. We also know that each node creates a single output, which will likely be passed to other nodes. Let's add two lists: one to store references to the inbound nodes, and the other to store references to the outbound nodes.

```
class Node(object):
    def __init__(self, inbound_nodes=[]):
        # Node(s) from which this Node receives values
        self.inbound_nodes = inbound_nodes
        # Node(s) to which this Node passes values
        self.outbound_nodes = []
        # For each inbound Node here, add this Node as an outbound Node to _th
        for n in self.inbound_nodes:
            n.outbound_nodes.append(self)
```

Each node will eventually calculate a value that represents its output. Let's initialize the valueto Noneto indicate that it exists but hasn't been set yet.

```
class Node(object):
    def __init__(self, inbound_nodes=[]):
        # Node(s) from which this Node receives values
        self.inbound_nodes = inbound_nodes
        # Node(s) to which this Node passes values
        self.outbound_nodes = []
        # For each inbound Node here, add this Node as an outbound Node to _th
        for n in self.inbound_nodes:
            n.outbound_nodes.append(self)
        # A calculated value
        self.value = None
```

Each node will need to be able to pass values forward and perform backpropagation (more on that later). For now, let's add a placeholder method for forward propagation. We'll deal with backpropagation later on.

```
class Node(object):
    def __init__(self, inbound_nodes=[]):
        # Node(s) from which this Node receives values
        self.inbound_nodes = inbound_nodes
        # Node(s) to which this Node passes values
        self.outbound_nodes = []
        # For each inbound Node here, add this Node as an outbound Node to _th
        for n in self.inbound_nodes:
            n.outbound_nodes.append(self)
        # A calculated value
        self.value = None

    def forward(self):
        """
        Forward propagation.

        Compute the output value based on `inbound_nodes` and
        store the result in self.value.
        """
        raise NotImplemented
```

Nodes that Calculate

While Nodedefines the base set of properties that every node holds, only specialized subclassesof Nodewill end up in the graph. As part of this lab, you'll build the subclasses of Nodethat can perform calculations and hold values. For example, consider the Inputsubclass of Node.

```
class Input(Node):
    def __init__(self):
        # An Input node has no inbound nodes,
        # so no need to pass anything to the Node instantiator.
        Node.__init__(self)

    # NOTE: Input node is the only node where the value
    # may be passed as an argument to forward().
    #
    # All other node implementations should get the value
    # of the previous node from self.inbound_nodes
    #
    # Example:
    # val0 = self.inbound_nodes[0].value
    def forward(self, value=None):
        # Overwrite the value if one is passed in.
        if value is not None:
            self.value = value
```

Unlike the other subclasses of `Node`, the `Input`subclass does not actually calculate anything. The `Input`subclass just holds a `value`, such as a data feature or a model parameter (weight/bias).

You can set `value`either explicitly or with the `forward()`method. This value is then fed through the rest of the neural network.

The Add Subclass

`Add`, which is another subclass of `Node`, actually can perform a calculation (addition).

```
class Add(Node):
    def __init__(self, x, y):
        Node.__init__(self, [x, y])

    def forward(self):
        """

        You'll be writing code here in the next quiz!
        """
```

Notice the difference in the `__init__`method, `Add.__init__(self, [x, y])`. Unlike the `Input`class, which has no inbound nodes, the `Add`class takes 2 inbound nodes, `x` and `y`, and adds the values of those nodes.