The "A Basic Graph" Section Explained

Step 1: Initializes the graph (g)

Step 2: Sets a global variable (_default_graph) as the graph's instance, "self"

Step 3: Sets variables A and b, which can be thought of as the parameters of the NN. It's seen later that these represent the "split line" between the data.

Step 4: Initializes the placeholder as x, which has an empty list as the only instance attribute.

Step 5: Defines the **multiply** class as y with parameters A and x. There are a few things going on here due to Operation's __init__ magic method:

1. Sets **multiply**'s attribute *input_nodes* as [A, x]
2. Sets **multiply**'s attribute *output_nodes* as an empty list
3. For every element in [A, x] (ie. for both A and x): appends **multiply** to the element's list of output nodes
4. Appends **multiply** to the _default_graph's list of operations

Step 6: Defines the **add** class as z with parameters y and b. The same processes 1-4 are taken as in Step 5:

1. Sets **add**'s attribute *input_nodes* as [y, b]
2. Sets **add**'s attribute *output_nodes* as an empty list
3. For every element in [y, b]: appends **add** to the element's list of output nodes
4. Appends **add** to the_default_graph's list of operations

The "Session" Section Explained

Step 7: Defines a **Session** class

Step 8: Invokes the **Session**'s **run()** method and stores the output as the variable result. The inputs for the **run()** method are *operation* as z (see Step 6) and *feed_dict* as the single placeholder-input_value pair x: 10. Here, x is the placeholder (see Step 4) and 10 is the input value.

This is a meaty step, so I'll break it down into substeps.

Step 8a: Sets traverse_postorder(z) as the variable nodes_postorder. Recall that z is add(multiply(A,placeholder()),b) explicitly.

Here's the steps of traverse_postorder(z) (skip this if you want): Firstly, it defines *nodes_postorder* as an empty list. Then:

1. Checks if z is an instance of the super class **Operation**. In this case, this is true because z=**add** overwrites **Operation**'s *compute()* instance.
2. As previous is true, for each element in add's *input_nodes* instance (that is, y and b), perform the previous but with y and b instead of z.
3. y is an instance of Operation (as with z, it overwrites *compute()*). However, b is not - the **Variable** class does not inherit from the **Operation** class.
4. As previous true for y=**mult**(A,x), perform the above with A and x this time.
5. Neither Variable nor Placeholder inherit from Operation... etc. (see diagram to the right for all steps)

Hence, for this setup, the output of traverse_postorder(z) is [A, x, y, b, z]. This can be confirmed via the jupyter notebook on this topic by making Session.run() return *nodes_postorder!*

1. Proposition A: Is z an instance of **Operation**?

2. As Prop A is true, we ask Prop B: Is y an instance of **Operation**?

3. As Prop B is true, we ask Prop C: Is A an instance of **Operation**?

Prop C is false so no further recursions here

4. Appends A of the **Variable** class to *nodes_postorder*

5. As Prop B is true, we ask Prop D: Is A an instance of **Operation**?

Prop D is false so no further recursions here

6. Appends x of the **Placeholder** class to *nodes_postorder*

7. Appends y [of the **add** subclass] of the **Operation** class to *nodes_postorder*

8. As Prop A is true, we ask Prop E: Is b an instance of **Operation**?

Prop E is false so no further recursions here

9. Appends b of the **Variable** class to *nodes_postorder*

10. Appends z [of the **multiply** subclass] of the **Operation** class to *notes_postorder*

Step 8b: Enumerates through the elements of *nodes_postorder* and does different things depending on the super class of the current element.

If the element is of class **Placeholder**, the element's *output* instance is set to the element's corresponding input_value (see Step 8).

If the element is of class **Variable**, the element's *output* instance is set to the element's *value* instance.

If the element is of class Operation, the element's *output* instance is set to the element's *compute()* method applied to every element in the following list: Iterate over the element's *input_nodes* instance (ie. A and x when considering y), and take their (A and x's) *output* instances.

Step 8c: In the event of any of the outputs assigned in Step 8c were of type list, they are converted to a numPy array for compatibility with further operations such as matrix multiplication.

Step 8d: Step 8b and Step 8c were enumerating over notes_postorder. Now that's done, run() just returns the *output_instance* of the operation put into **run()**, ie. the *output_instance* of the final element of *nodes_postorder* (they're equivalent!)