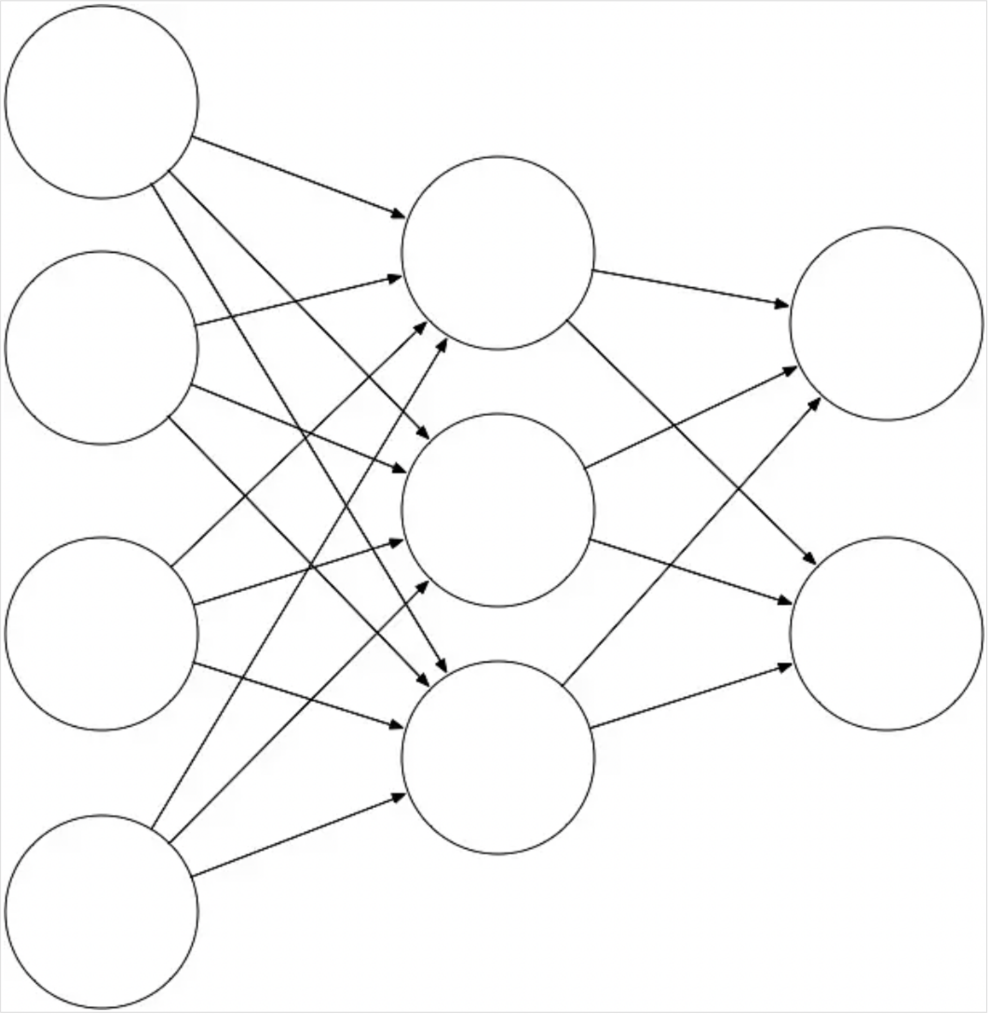
Assignment 4

Cary Jardin · Nov 6, 2021 · Notified 40 people

This is the graph you will be creating in code :)



gr.jpg · 53.3 KB · View full-size · Download

Motivation Of Assignment:
As we discussed in class, an Artificial Neural Network (ANN) is a directed, connected graph that we use to "guess" the correct answers. In this assignment you will be creating the "core" of the ANN that we will be using in the next assignment

assignment.

Objective of Assignment:

- Utilize lists, trees, and recursion to produce an Artificial Neural Network
- Be able to create a program from written requirements
- Complete an assignment on time

Scope:
Will be using this assignment for the next assignment

Submitting Your Assignment:
Email me the following:

- Name of your GitHub user and branch name

How the Assignment will be Graded:

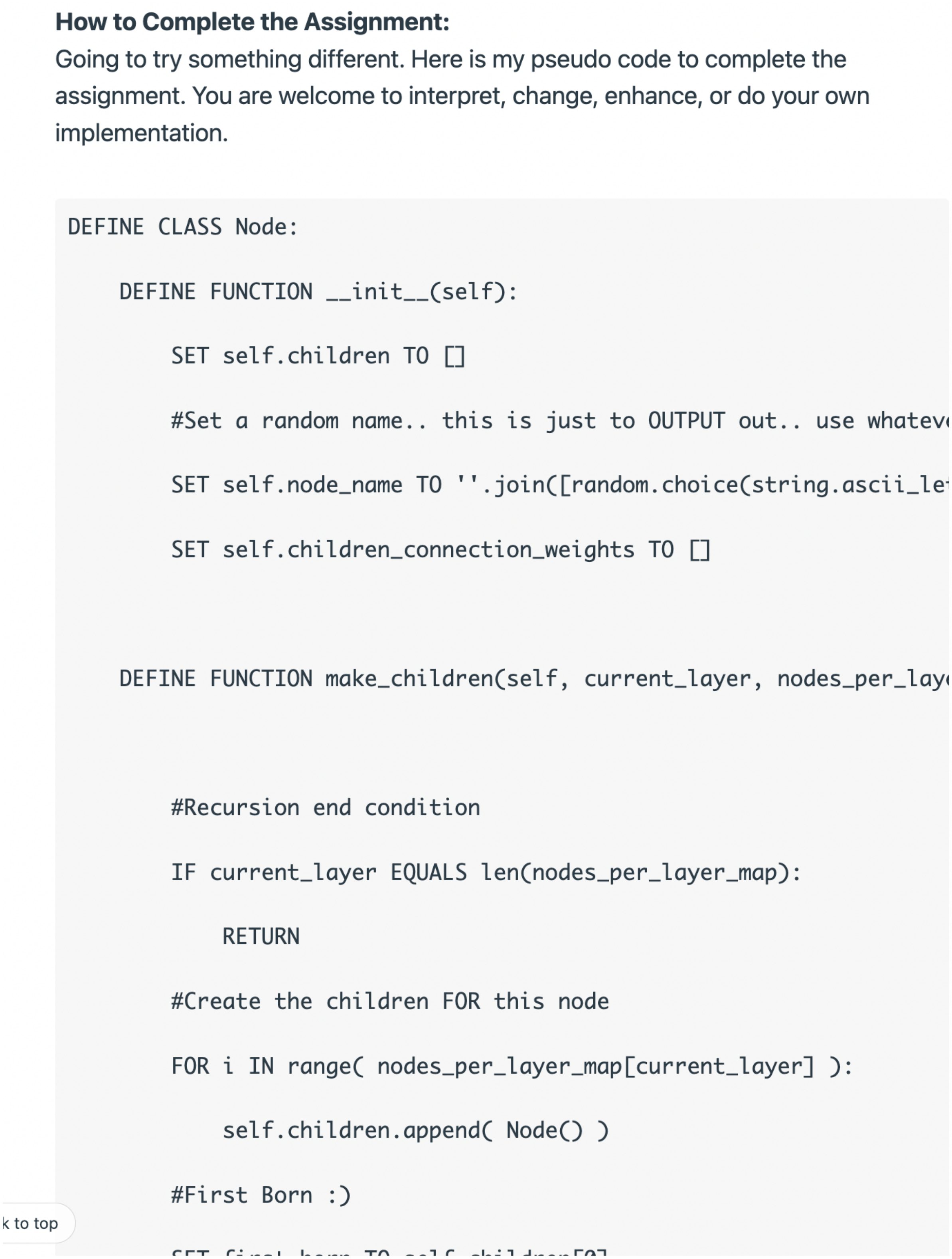
- All working, and submitted assignments get a C
  - Working is me being able to:
    - Clone your repo
    - Running your code and producing an output that shows connections
- Grade B
  - Produces output, but not fully connected and not correct
- Grade A
  - Produces output, all nodes are correctly connected, and all weights are displayed

Completeness:
Code, and executables need to be committed into your branch so I can run them on my ec2 instance

Assignment Due Date:

o top

Due by November 20th by 11:59pm



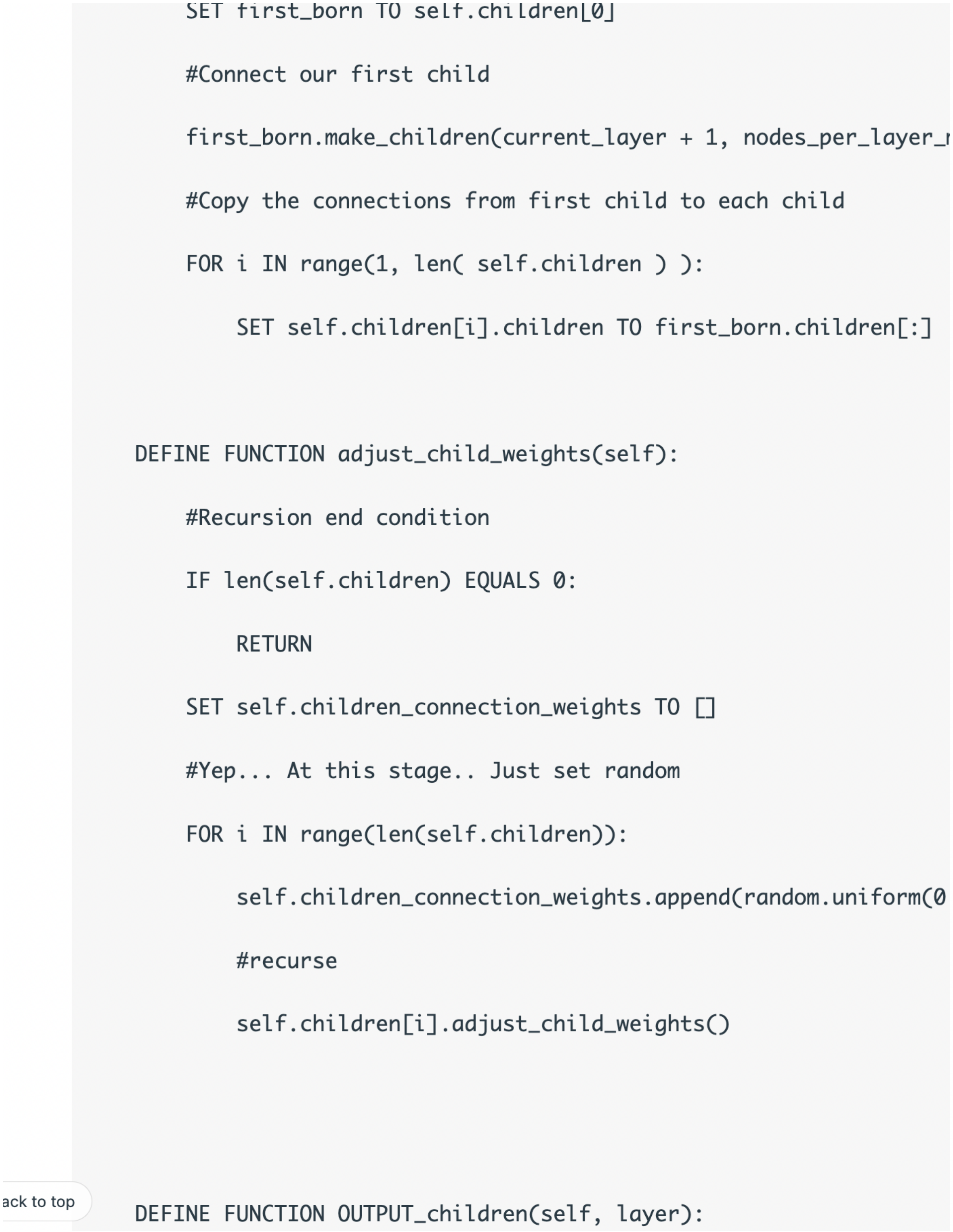
How to Complete the Assignment:

Going to try something different. Here is my pseudo code to complete the assignment. You are welcome to interpret, change, enhance, or do your own implementation.

```
DEFINE CLASS Node:
    DEFINE FUNCTION __init__(self):
        SET self.children TO []
        #Set a random name.. this is just to OUTPUT out.. use whateve
        SET self.node_name TO ''.join([random.choice(string.ascii_le
        SET self.children_connection_weights TO []

    DEFINE FUNCTION make_children(self, current_layer, nodes_per_lay

        #Recursion end condition
        IF current_layer EQUALS len(nodes_per_layer_map):
            RETURN
        #Create the children FOR this node
        FOR i IN range( nodes_per_layer_map[current_layer] ):
            self.children.append( Node() )
        #First Born :)
        SET first_born TO self.children[0]
```



```
    SET first_born TO self.children[0]
    #Connect our first child
    first_born.make_children(current_layer + 1, nodes_per_layer_r
    #Copy the connections from first child to each child
    FOR i IN range(1, len( self.children ) ):
        SET self.children[i].children TO first_born.children[:]


DEFINE FUNCTION adjust_child_weights(self):
    #Recursion end condition
    IF len(self.children) EQUALS 0:
        RETURN
    SET self.children_connection_weights TO []
    #Yep... At this stage.. Just set random
    FOR i IN range(len(self.children)):
        self.children_connection_weights.append(random.uniform(0
        #recurse
        self.children[i].adjust_child_weights()



DEFINE FUNCTION OUTPUT_children(self, layer):
        #just used to indent per level
        SET indent TO '    ' * layer
        #Recursion end case
        IF len(self.children) EQUALS 0:
            OUTPUT(f"{indent}{self.node_name}")
            RETURN
        OUTPUT(f"{indent}{self.node_name} is connected to ")
        FOR i IN range(len(self.children)):
            self.children[i].OUTPUT_children(layer+1)

            #OUTPUT the weight IF we have it
            IF i < len(self.children_connection_weights):
                OUTPUT(f"{indent}with weight {self.children_connecti

#Create a master node that we can use to connect to all the layers
SET INPUT_nodes TO []
SET master_node TO Node()
#make our first node
```

ack to top

```
SET my_first_node TO Node()

#make all the children FOR the first node

my_first_node.make_children(1, NODE_COUNT_PER_LAYER)

master_node.children.append(my_first_node)

#duplicate the first node FOR all INPUT nodes

FOR i IN range(1, len(NODE_COUNT_PER_LAYER)):

    SET new_node TO Node()

    #copy the children to the new node

    SET new_node.children TO my_first_node.children[:]

    master_node.children.append(new_node)

#OUTPUT out to see IF we are all connected

master_node.OUTPUT_children(0)

OUTPUT("!! Set Weights !!")

#init the weights

master_node.adjust_child_weights()

#OUTPUT out with weights

master_node.OUTPUT_children(0)
```

Here is my output:

ack to top

```
bpW is connected to
    Izz is connected to
        ghi is connected to
            kYm
            qwC
        WsP is connected to
            kYm
            qwC
        RpX is connected to
            kYm
            qwC
    coK is connected to
        ghi is connected to
            kYm
            qwC
        WsP is connected to
            kYm
            qwC
        RpX is connected to
            kYm
            qwC
    uSh is connected to
        ghi is connected to
            kYm
            qwC
        WsP is connected to
            kYm
            qwC
        RpX is connected to
            kYm
            qwC
!! Set Weights !!
bpW is connected to
    Izz is connected to
        ghi is connected to
            kYm
        with weight 0.31396430208687065
            qwC
```

‹ to top

```
        with weight 0.7543072430877908
    with weight 0.6829939315676103
        WsP is connected to
            kYm
        with weight 0.037485018064457254
            qwC
        with weight 0.474016988718973
    with weight 0.36673172094385786
        RpX is connected to
            kYm
        with weight 0.015303828558788646
            qwC
        with weight 0.15036432009054634
    with weight 0.5007743341078787
with weight 0.8579905098045468
    coK is connected to
        ghi is connected to
            kYm
        with weight 0.31396430208687065
            qwC
        with weight 0.7543072430877908
    with weight 0.21866974376947246
        WsP is connected to
            kYm
        with weight 0.037485018064457254
            qwC
        with weight 0.474016988718973
    with weight 0.40180077636151623
        RpX is connected to
            kYm
        with weight 0.015303828558788646
            qwC
        with weight 0.15036432009054634
    with weight 0.9014772771346441
with weight 0.006751699877545092
    uSh is connected to
        ghi is connected to
            kYm
        with weight 0.31396430208687065
```

to top

```
            qwC
        with weight 0.7543072430877908
    with weight 0.21866974376947246
        WsP is connected to
            kYm
        with weight 0.037485018064457254
            qwC
        with weight 0.474016988718973
    with weight 0.40180077636151623
        RpX is connected to
            kYm
        with weight 0.015303828558788646
            qwC
        with weight 0.15036432009054634
    with weight 0.9014772771346441
with weight 0.006751699877545092
    uSh is connected to
        ghi is connected to
            kYm
        with weight 0.31396430208687065
            qwC
        with weight 0.7543072430877908
    with weight 0.34629788998833144
        WsP is connected to
            kYm
        with weight 0.037485018064457254
            qwC
        with weight 0.474016988718973
    with weight 0.6203941710539433
        RpX is connected to
            kYm
        with weight 0.015303828558788646
            qwC
        with weight 0.15036432009054634
    with weight 0.7423199933426761
with weight 0.14711754058441984
```