|  |  |  |
| --- | --- | --- |
| File | Code | Explanations |
| training | network\_dimensions = 5, 4, 2  Network(network\_dimensions) | Calls the Network class in network.py and passes along the number of layers of network. network[0] object is initiated |
| network | for i in range(len(dimensions) - 1) | The loop will process one less time than the number of dimensions because the Layer class processes pairs – e.g. 5&4 then 4&2 |
| self.layers = [] | An empty list is created and the result of the Layer function will be appended to it |
| Layer(dimensions[i], dimensions[i + 1]) | Calls Layer class in network.py and passes along the first two dimensions (5, 4) |
| \_\_init\_\_(self, inputs\_count, outputs\_count) | A Layer[0] object is created with dimension[i] =5 now inputs\_count and dimension [i+1] = 4 outputs\_count |
| self.outputs = [0.0 for \_ in range(outputs\_count)] | The self.outputs list is initialized in the object with “0.0” times the value of outputs\_count = 4 – [0.0, 0.0, 0.0, 0.0] |
| self.weights = [[random.random() \* 2 - 1  for \_i in range(inputs\_count)]  for \_o in range(outputs\_count)] | The self.weights list is initialized with a two-dimensional matrix of width inputs\_count (5) and height outputs\_count (4) with a random number between -1 and 1   |  |  |  |  |  | | --- | --- | --- | --- | --- | | r1 | r2 | r3 | r4 | r5 | | r6 | r7 | r8 | r9 | r10 | | r11 | r12 | r13 | r14 | r15 | | r16 | r17 | r18 | r19 | r20 | |
| Layer(dimensions[i], dimensions[i + 1]) | The second loop follows the same process, but the self-outputs list for the Layer[1] object is [0.0, 0.0] since dimensions[i+1] = 2, and the Layer[1] object gets a matrix that is 4 x 2   |  |  |  |  | | --- | --- | --- | --- | | r21 | r22 | r23 | r24 | | r25 | r26 | r27 | r28 | |
| self.layers.append… | The two matrices are combined and stored in the self.layer element of network[0], resulting in the network to be assigned to the first car   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | layer[0] | r1 | r2 | | r3 | | | r4 | r5 | | | r6 | r7 | | r8 | | | r9 | r10 | | | r11 | r12 | | r13 | | | r14 | r15 | | | r16 | r17 | | r18 | | | r19 | r20 | | | layer[1] | r21 | | r22 | | r23 | r24 | | | | r25 | | r26 | | r27 | r28 | | | |

|  |  |  |
| --- | --- | --- |
| File | Code | Explanations |
| car | acceleration, steer\_position = self.network.feed\_forward(measurements) | Sends the results of the five probes to the feed\_forward function in the Network class |
| network | feed\_forward(self, inputs) | inputs now equals the five radar values from car.py |
| for layer in self.layers: | The loop is going to run twice because we have two layers |
| layer.feed\_forward(inputs) | Sending first inputs (five radar values) to the feed\_forward function in the Layers class |
| for output\_index, output in enumerate(self.outputs): | The first loop will run four times. This is the number assigned to self.outputs from the network function: [i[0], i[1], i[2], i[4] and will assign the results of the calculation to output\_index. |
| for weight\_index, input in enumerate(inputs): | This inner loop will run five times since this is the number of radars |
| sum +=  input \* self.weights[output\_index][weight\_index] | sum[0] = i[0]r1 + i[0]r2 + i[0]r3 + i[0]r4 + i[0]r5  sum[1] = i[1]r6 + i[1]r7 + i[1]r8 + i[1]r9 + i[1]r10 etc.  each of the sums are added to the output\_index list |
| self.outputs[output\_index] = math.tanh(sum) | Each of the sums gets put through tanh function to get a number between -1 and 1 and then assigned to the self.outputs list  [t[0], t[1], t[2], t[3], t[4]] |
| inputs = [i for i in layer.outputs] | This copies the outputs from Layer[0] into the inputs variable that will then be sent to the feed\_forward function in the Layer class for the Layer[1] |
| This time the outer loop will run two times and the inner loop will run four times | sum[0] = i[0]r21 + i[0]r22 + i[0]r23 + i[0]r24  sum[1] = i[1]r25 + i[1]r26 + i[1]r27 + i[1]r28  each of the sums are added to the output\_index list |
| It is put back through the tanh function | [[t0],t[1]] |
| return self.layers[-1].outputs | This returns the last list in the outputs to accelerate and steer |

import random

import math

class Layer:

    def \_\_init\_\_(self, inputs\_count, outputs\_count):

        # initialize list of outputs

        self.outputs = [0.0 for \_ in range(outputs\_count)] # does this overwrite in the second loop?

        # create list of random numbers for qty of inputs x outputs (5x4=20)

        # I don't understand the use of \_i and \_o instead of just i and o

        self.weights = [[random.random() \* 2 - 1 for \_i in range(inputs\_count)] for \_o in range(outputs\_count)]

    def feed\_forward(self, inputs):

        # calculate outputs based on inputs and weights

        for output\_index, output in enumerate(self.outputs): # for each output

            sum = 0

            for weight\_index, input in enumerate(inputs):

                sum += input \* self.weights[output\_index][weight\_index] # multiply input by the weight

            # Activation function - without this, you just have linear calculations

            # addint the tanh function makes the model more powerful

            self.outputs[output\_index] = math.tanh(sum)

class Network:

    def \_\_init\_\_(self, dimensions): # dimensions describe qty and count of each layer (5, 4, 2)

        self.dimensions = dimensions

        self.layers = []

        # runs the loop one less time than the number of dimensions

        for i in range(len(dimensions) - 1):

            # because this function is processing two at a time

            # in 5,4,2 example, first processes from 5 to 4, then from 4 to 2

            self.layers.append(Layer(dimensions[i], dimensions[i + 1]))

    def feed\_forward(self, inputs):

        for layer in self.layers: # layer exists for each pair in dimensions 5:4 and 4:2 in our example

            layer.feed\_forward(inputs) # first inputs will be measurements from the probes

            inputs = [i for i in layer.outputs] # copies output of first iteration of layer.feedfoward into inputs

        return self.layers[-1].outputs # [-1] returns the set of outputs for steer and accelerate