```
import numpy as np
from itertools import product
from functools import reduce

debug_print = print



class McCullochPitt:
    def __init__(self, dataset, target_row_val, name_of_gate):
        """
        :param dataset: This is the set of all the possible combinations that can be input to the nn.
                        For example:
                        for and gate:
                        x1 x2   y
                        0  0    0
                        0  1    0
                        1  0    0
                        1  1    1
        :param target: the value which is the target which will be classified / separated from other
class.
        :param function: the function for which the dataset is to be trained.
        """
        self.dataset = dataset
        self.target_row_val = target_row_val
        self.X = [row[:-1] for row in dataset]
        self.Y = [row[-1] for row in dataset]
        self.m = len(self.X[0])  # the number of inputs in the first row is the number of cols in all the
rows
        self.n = len(self.dataset)  # number of rows that is there in the dataset.
        self.function = Gates.get_gate(name_of_gate)
        self.weights = None  # this will be set after training.
        self.threshold = None  # this will be set while training.

    @staticmethod
    def get_threshold(op, target):
        """
        :param op: yin = sum(FiXi), this is applied to the thresholding function.
        :param target: The target value of the dataset which is 0/1
        :return: minimum of the values of op for which the target value is 1.
        """
        return min([i for i in zip(op, target) if i[1] == 1], key=lambda x: x[1])[0]

    @staticmethod
    def apply_threshold(row, threshold):
        """
        This will return the arrays of the value that will represent whether
        that specific value is greater than equal to the threshold value passed to the function.
        :param row: the row / list for which will be applied with threshold and returned 0/1
accordingly.
        :param threshold: The value based on which the value of the op will be dependent on.
        :return: list of the element with cardinality of the row.
        """
        return [0 if ele < threshold else 1 for ele in row]

    def train(self, combs=(-1, 0, 1)):
        """
        This assumes that, there is only one layer and having number of nodes = number of the
inputs in the dataset.
        This will be setting the weights and the threshold for which the model predicted the op
accurately.
        """
        # these are all the combinations of 0, 1, -1
        # basically the sample space.
        possible_weights = product(combs, repeat=self.m)

        # iterating through the sample space and finding out whether the weight chosen gave
correct output.
        # converting the weight to np array in order for me to enable to multiply without reduce and
zip.
        for weights in map(np.array, possible_weights):
            debug_print('Checking for weights {}'.format(weights))

            # calculating sum(FiXi).
            op = [sum(weights * row) for row in self.X]

            # trying to get threshold.
            threshold = McCullochPitt.get_threshold(op, target=self.Y)

            # applying the threshold to sum(FiXi) to get o/p of the neuron in 0/1.
            neuron_op = McCullochPitt.apply_threshold(op, threshold)

            # this is probably not a good approach for checking if the predicted and actual weights
were same.
            if neuron_op == self.Y:
                # setting the threshold and weights for which we got correct o/p.
                self.weights = weights
                self.threshold = threshold
                break
        if self.weights is None:
            raise ValueError("Couldn't train model")
        else:
            debug_print('Training successful!! ')


class DataSet:
    def __new__(cls, name_of_gate, repeat, ip_values):
        return cls.get_dataset(Gates.get_gate(name_of_gate), repeat, ip_values)

    @staticmethod
    def get_dataset(function, repeat, ip_values):
        return [list(row) + [function(row)] for row in product(ip_values, repeat=repeat)]



class Gates:
    @staticmethod
    def _and(l):
        return reduce(lambda x, y: x * y, l)

    @staticmethod
    def _or(l):
        if len(l) != 2:
            raise NotImplementedError("number of element should be 2")
        return l[0] or l[1]

    @staticmethod
    def nand(l):
        if len(l) != 2:
            raise NotImplementedError("number of element should be 2")
        return int(not Gates._and(l))

    @staticmethod
    def and_not(l):
        if len(l) != 2:
            raise NotImplementedError("number of element should be 2")
        return int(l[0] and not l[1])

    @staticmethod
    def nor(l):
        if len(l) != 2:
            raise NotImplementedError("number of element should be 2")
        return int(not Gates._or(l))

    @staticmethod
    def get_gate(name_of_gate):
        function_gate_mapping = {
            "AND": Gates._and,
            "OR": Gates._or,
            "NAND": Gates.nand,
            "NOR": Gates.nor,
            "AND_NOT": Gates.and_not
        }
        return function_gate_mapping.get(name_of_gate.upper())


for gate in ("and", 'or', 'nand', 'nor', 'and_not'):
    try:
        print('\nTrying to train model for {} gate:'.format(gate.upper()))
        dataset = DataSet(gate, 2, [0, 1])
        model = McCullochPitt(dataset=dataset, target_row_val=1, name_of_gate='naNd')
        model.train()
        print('Trained weights:', model.weights, model.threshold)
    except ValueError:
        print("Couldn't train the model for the gate.")


"""
Trying to train model for AND gate:
Checking for weights [-1 -1]
Checking for weights [-1  0]
Checking for weights [-1  1]
Checking for weights [ 0 -1]
Checking for weights [0 0]
Checking for weights [0 1]
Checking for weights [ 1 -1]
Checking for weights [1 0]
Checking for weights [1 1]
Training successful!!
Trained weights: [1 1] 2

Trying to train model for OR gate:
Checking for weights [-1 -1]
Checking for weights [-1  0]
Checking for weights [-1  1]
Checking for weights [ 0 -1]
Checking for weights [0 0]
Checking for weights [0 1]
Checking for weights [ 1 -1]
Checking for weights [1 0]
Checking for weights [1 1]
Training successful!!
Trained weights: [1 1] 1

Trying to train model for NAND gate:
Checking for weights [-1 -1]
Checking for weights [-1  0]
Checking for weights [-1  1]
Checking for weights [ 0 -1]
Checking for weights [0 0]
Checking for weights [0 1]
Checking for weights [ 1 -1]
Checking for weights [1 0]
Checking for weights [1 1]
Couldn't train the model for the gate.

Trying to train model for NOR gate:
Checking for weights [-1 -1]
Training successful!!
Trained weights: [-1 -1] 0

Trying to train model for AND_NOT gate:
Checking for weights [-1 -1]
Checking for weights [-1  0]
Checking for weights [-1  1]
Checking for weights [ 0 -1]
Checking for weights [0 0]
Checking for weights [0 1]
Checking for weights [ 1 -1]
Training successful!!
Trained weights: [ 1 -1] 1
"""
```