import math  
import random  
from sklearn import preprocessing  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
import matplotlib.pyplot as plt  
import pandas as pd  
random.seed(0)  
mm = MinMaxScaler()  
frequency = []  
error\_s = []  
def rand(a, b):  
 return (b - a) \* random.random() + a  
def make\_matrix(m, n, fill=0.0):  
 mat = []  
 for i in range(m):  
 mat.append([fill] \* n)  
 return mat  
def sigmoid(x):  
 return (2.0\*1.0 / (1.0 + np.exp(-x)))-1  
def sigmoid\_derivative(x):  
 return 2.0\*np.exp(-x)/((1+np.exp(-x))\*\*2)  
class BPNeuralNetwork:  
 def \_\_init\_\_(self):  
 self.input\_n = 0  
 self.hidden\_n = 0  
 self.output\_n = 0  
 self.input\_cells = []  
 self.hidden\_cells = []  
 self.output\_cells = []  
 self.input\_weights = []  
 self.output\_weights = []  
 self.input\_correction = []  
 self.output\_correction = []  
  
 def setup(self, ni, nh, no):  
 self.input\_n = ni + 1  
 self.hidden\_n = nh  
 self.output\_n = no  
 # init cells  
 self.input\_cells = [1.0] \* self.input\_n  
 self.hidden\_cells = [1.0] \* self.hidden\_n  
 self.output\_cells = [1.0] \* self.output\_n  
 # init weights  
 self.input\_weights = make\_matrix(self.input\_n, self.hidden\_n)  
 self.output\_weights = make\_matrix(self.hidden\_n, self.output\_n)  
 # random activate  
 for i in range(self.input\_n):  
 for h in range(self.hidden\_n):  
 self.input\_weights[i][h] = rand(-0.2, 0.2)  
 for h in range(self.hidden\_n):  
 for o in range(self.output\_n):  
 self.output\_weights[h][o] = rand(-2.0, 2.0)  
 # init correction matrix  
 self.input\_correction = make\_matrix(self.input\_n, self.hidden\_n)  
 self.output\_correction = make\_matrix(self.hidden\_n, self.output\_n)  
  
 def predict(self, inputs):  
 # activate input layer  
 for i in range(self.input\_n - 1):  
 self.input\_cells[i] = inputs[i]  
 # activate hidden layer  
 for j in range(self.hidden\_n):  
 total = 0.0  
 for i in range(self.input\_n):  
 total += self.input\_cells[i] \* self.input\_weights[i][j]  
 self.hidden\_cells[j] = sigmoid(total)  
 # activate output layer  
 for k in range(self.output\_n):  
 total = 0.0  
 for j in range(self.hidden\_n):  
 total += self.hidden\_cells[j] \* self.output\_weights[j][k]  
 self.output\_cells[k] = sigmoid(total)  
 return self.output\_cells[:]  
  
 def back\_propagate(self, case, label, learn, correct):  
 # feed forward  
 self.predict(case)  
 # get output layer error  
 output\_deltas = [0.0] \* self.output\_n  
 for o in range(self.output\_n):  
 error = label[o] - self.output\_cells[o]  
 output\_deltas[o] = sigmoid\_derivative(self.output\_cells[o]) \* error  
 # get hidden layer error  
 hidden\_deltas = [0.0] \* self.hidden\_n  
 for h in range(self.hidden\_n):  
 error = 0.0  
 for o in range(self.output\_n):  
 error += output\_deltas[o] \* self.output\_weights[h][o]  
 hidden\_deltas[h] = sigmoid\_derivative(self.hidden\_cells[h]) \* error  
 # update output weights  
 for h in range(self.hidden\_n):  
 for o in range(self.output\_n):  
 change = output\_deltas[o] \* self.hidden\_cells[h]  
 self.output\_weights[h][o] += learn \* change + correct \* self.output\_correction[h][o]  
 self.output\_correction[h][o] = change  
 # update input weights  
 for i in range(self.input\_n):  
 for h in range(self.hidden\_n):  
 change = hidden\_deltas[h] \* self.input\_cells[i]  
 self.input\_weights[i][h] += learn \* change + correct \* self.input\_correction[i][h]  
 self.input\_correction[i][h] = change  
 # get global error  
 error = 0.0  
 for o in range(len(label)):  
 error += 0.5 \* (label[o] - self.output\_cells[o]) \*\* 2  
 return error  
  
 def train(self, cases, labels, limit=1000, learn=0.05, correct=0.1):  
 for j in range(limit):  
 error = 0.0  
 for i in range(len(cases)):  
 label = labels[i]  
 case = cases[i]  
 error += self.back\_propagate(case, label, learn, correct)  
 print("次数=", j, "error=", error)  
 frequency.append(j)  
 error\_s.append(error)  
 if error < correct:  
 break  
  
 def test(self):  
 data1 = open('first\_data.csv')  
 data2 = pd.read\_csv(data1)  
  
 X = data2[['人口', 'GDP', '垃圾处理厂数量', '城市化率', '污水排放量', '食品制造业固体废物产生量（万吨）']]  
 y = data2[['垃圾总产量']]  
 X = np.asarray(X)  
 y = np.asarray(y)  
 print(X)  
 print(y)  
 # 数据归一化  
 # cases = mm.fit\_transform(X)  
 X = mm.fit\_transform(X)  
 print(X)  
 y = mm.fit\_transform(y)  
 print(y)  
 # 数据归一化  
 # labels = mm.fit\_transform(y)  
 # print(labels)  
 self.setup(6, 8, 1)  
 self.train(X, y, 2000, 0.1, 0.01)  
 # for case in cases:  
 # # 归一化还原  
 # print(self.predict(case))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 nn = BPNeuralNetwork()  
 nn.test()  
 data3 = open('first\_test\_data.csv')  
 data4 = pd.read\_csv(data3)  
 X\_predict = data4[['人口', 'GDP', '垃圾处理厂数量', '城市化率', '污水排放量', '食品制造业固体废物产生量（万吨）']]  
 X\_predict = np.asarray(X\_predict)  
 print(np.asarray(nn.input\_weights).shape)  
 print(np.asarray(nn.output\_weights).shape)  
 print(np.dot(np.asarray(nn.input\_weights)[:-1], np.asarray(nn.output\_weights)))  
 X\_predict = (X\_predict - X\_predict.min())/(X\_predict.max() - X\_predict.min()) #最小-最大规范化  
 X\_predict = np.asarray(X\_predict)  
 # print(X\_predict)  
 # predict = np.asarray(np.asarray(nn.predict(X\_predict.ravel())).ravel())  
 for i in range(0,20):  
 predict = nn.predict(X\_predict[i])  
 # # 反归一化数据  
 result = mm.inverse\_transform(np.asarray(predict).reshape(1,1))  
 print(2019+i, "年预测值：", result)  
 plt.title('BP神经网络训练次数和误差值的关系', size=14)  
 plt.xlabel('训练次数', size=10)  
 plt.ylabel('误差值', size=10)  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.rcParams['axes.unicode\_minus'] = False  
 plt.plot(frequency, np.power(error\_s, 2), '.-')  
 plt.show()