Neural Network Theory and Applications Homework Assignment 2

林煜

2017 年 3 月 20 日

1. **Solution.** 利用python3 结合numpy和matplotlib库实现，代码见lab2.py。

   大体实现过程：首先将训练数据中两类数据均随机分成两份，$X_1^1, X_1^2, Y_0^1, Y_0^2$ 分别表示正类第一二份，负类第一二份。将正类和负类数据两两组合，构成四份训练数据

   $$\{\{X_1^i + Y_0^j\}_{i=1,j=1}^{N_i=2,N_j=2}\}$$

   将这四份数据分别使用上次作业实现的MLQP网络训练，最后将四个神经元组织成Min-Max网络。网络如下

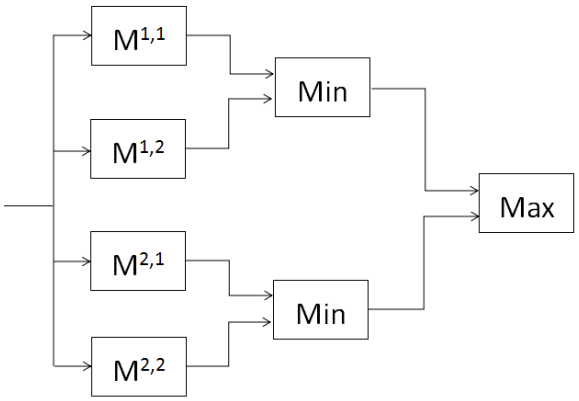


   Figure 1: Min-Max结构

   其中M^{ij} 代表某一个神经元，其训练数据为$\{X_1^i + Y_0^j\}$。对正类的输入数据相同的神经元的输出执行Min操作，最后执行Max操作。

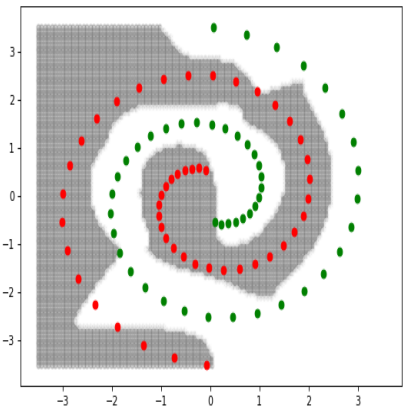
   以下是程序网络训练部分伪代码。

Algorithm 1: Min-Max网络训练

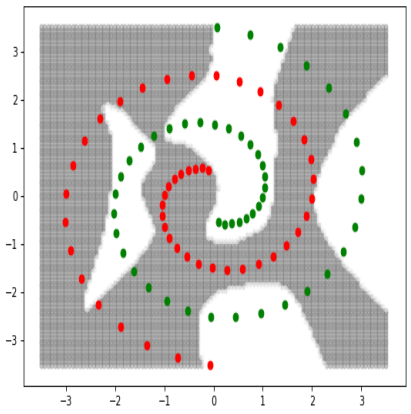
Input: 训练数据data
Output: Min-Max网络

X, Y ← data按类别划分;
X_1^1, X_1^2 ← X随机分成两份;
Y_0^1, Y_0^2 ← Y随机分成两份;
$xNum, yNum \leftarrow 2, 2$;
for *i=1 to xNum* **do**
    for *j=1 to yNum* **do**
        $M^{i,j}$ ← 新建一个MLQP神经网路;
        $M^{i,j}.tranning(X_1^i + Y_0^j)$;
按照Figure 1的形式组织网络返回;

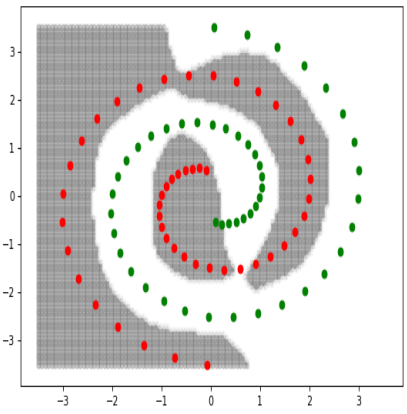
2. 训练完成后，对每一个子模块以及Min-Max网络，使用测试集进行测试。并在x轴[-3.5,3.5]，y轴[-3.5,3.5]区域内，等间距生产10000个点，绘制决策边界。以下是每一个神经元的决策边界图。



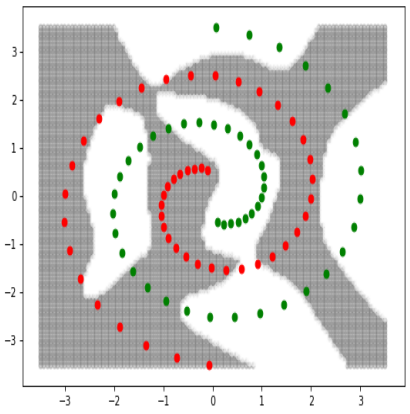
(a) $M^{1,1}$(93/96)



(b) $M^{1,2}$(82/96)



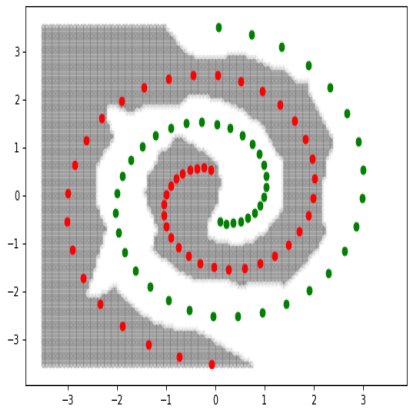
(c) $M^{2,1}$(93/96)



(d) $M^{2,2}$(83/96)

其中红绿点对应测试集的点的分布。红色点为输出为1的点，绿色为输出0。为了明显看出

决策边界，我采用灰色和白色作为背景色，红色的点对应灰色区域，绿色的点对应白色区域，括号中对应（测试正确个数/测试集总数）。下面是Min-Max网络对应的测试结果。



(e) $Min - Max$(96/96)

3. 由于在做作业二的过程中，开始的时候$Min - Max$的训练效果非常差，基本正确率仅在70%左右。于是我仔细检查了作业一中的程序，发现把各个神经元的权值及偏置初始值设成随机且每次迭代时打乱数据顺序，性能大大提高，不论是作业一还是作业二，均能基本达到100% 的测试正确率。在打包中$hw1_3.py$是新的作业一实现。另外我没有将四个子模块并行训练，所以就粗略的将四个模块中运行时间最长的神经元作为$Min - Max$训练时间。以下将他们俩进行对比，其中学习速率为0.5。

运行结果 / 网络	训练时间(s)	测试正确率
MLQP	7.73	100%(96/96)
Min-Max	6.62	100%(96/96)

而且有的时候Min-Max训练时间还会多于MLQP，表现为子神经元模块恰好训练过慢，会出现这种情况。需要注意的是，我的程序花费时间较长，这是由于我将神经元写成对象的形式，所以本来整个权值矩阵变为了一个个权值向量，导致numpy库矩阵运算的优化没有发挥作用，所以程序比较慢。

另外，Min-Max的正确率依赖于子模块的正确性。综上，Min-Max在训练时间上由于可以并行处理小模块，故训练时间普遍减少了。