一. 问题介绍

1. 解决的问题

由于每个人字体和书写习惯的差异，就阿拉伯数字 0 到 1 而言，同一个数字在不同人的书写下可能差别甚大。但是对于人类，即使字体差异很大，人眼也可以辨认每一个数字。那么，如何训练机器识别手写数字呢？在本文中，本文作者设计并实现了一个用以解决手写数字识别问题的自组织映射（Self Organizing Map，SOM）神经网络。

2. 选择的数据集：MNIST 数据集

本文使用的是 MNIST 数据集。这是一组由美国高中生和人口调查局员工手写的 70000 个数字的图片。本文使用其中的一个子集，并将其命名为“SOM_MNIST_data.txt”。该子集是一个784×5000的矩阵。5000 表示数据集中有 5000 张图片，784 是每张图片的特征。因为对于每一张数字图片，将其切割成了28×28的像素点，所以共有 784 个像素点，即 784 个特征，每个特征代表了一个像素点的强度，范围是 0 到 255，0 是白色，255 是黑色。在 Python 中，图像灰度是 uint8 类型的，为了方便计算，我们这里使用的图像灰度是 float 类型的，这样就把数字控制在了 0-1 的范围内。

3. 选择的算法：SOM 神经网络

首先对于“手写数字识别”，可以看做是一个分类问题，就是将每个样本分为 0-9 这十个类别中的一类。所以我们这里需要一个可以应用于分类的算法。

自组织映射神经网络（Self Organizing Map，SOM）是一种无监督学习的神经网络，不同于一般神经网络基于损失函数的优化训练，SOM 运用竞争学习策略以及聚类思想来优化网络。其聚类的基本思想是将距离小的个体集合划分为同一类别，将距离大的个体集合划分为不同的类别。

SOM 的思想是模拟生物神经网络接收外界刺激时候的反应。生物神经网络在接收到外界刺激（例如气味）的时候，与该刺激相关的神经元会被激活，被激活的神经元分布在同一个拓扑结构之中，越是位于拓扑领域中心位置的神经元活动越是剧烈，越是远离拓扑领域中心位置的神经元活动越不明显。

那么选择 SOM 的原因如下：

（1）SOM 神经网络本身就是用于解决分类问题的。并且该神经网络的特点是可以将高维的数据样本进行降维处理，从而将其映射到一维或二维上，而我们的 MNIST 数据集正是一个需要降维处理的高维数据集。

（2）SOM 是无监督学习的，不同于监督学习，它不需要分训练集以及测试集，那么对于数据量较小的数据集，无监督学习更加合适。而数据集“SOM_MNIST_data.txt”中只有 5000 条数据，即使使用交叉验证的方式，数据量依旧很小，所以无监督学习更加合适。

（3）SOM 作为一个神经网络，它只有两层，一层输入层一层输出层（SOM 的输出层通常被称作竞争层）。这使得 SOM 的实现较为简单。

（4）SOM 的权重迭代公式中不存在导数、梯度下降等计算过程，这也使得 SOM 的权重迭代过程较为简单。

二. 算法设计与说明

1. SOM 神经网络

前面提到，SOM 神经网络接收外界输入数据的方式是模拟生物神经网络接收外界刺激的方式。就 SOM 神经网络本身而言，该神经网络由两层构成，一层输入层和一层输出层（在 SOM 中，输出层称为竞争层），其结构如下图所示：

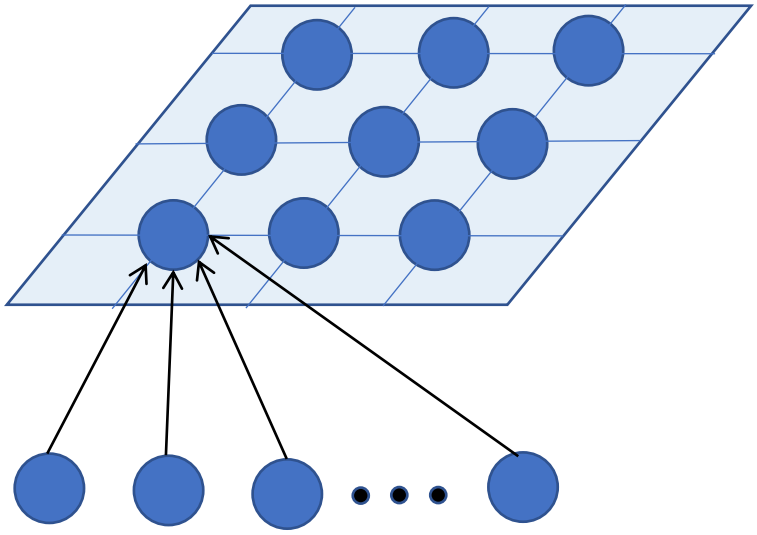

图 1 SOM 结构图

从图上可以看出，下面的一层是 SOM 的输入层，上面的平面是 SOM 的竞争层。SOM 的过程是：首先随机给出竞争层神经元对应权重的值。对于输入 x_i，计算每个权重与 x_i 的距离，距离最小的神经元就是获胜神经元，获胜神经元被理解为兴奋最强的神经元；之后获取获胜神经元的拓扑领域，在该领域内，获胜神经元位于领域中心，越靠近获胜神经元，兴奋越强，反之，兴奋越弱。

可以将 SOM 的训练过程分成以下几个步骤：

（1）竞争过程

竞争过程就是选择获胜神经元的过程，而选择获胜神经元的原则是选取与输入数据距离最近的一个神经元。

如果输入空间是 D 维的，那么我们可以得到输入数据为：$X=\{x_i : i=1,2,\cdots,D\}$。输入的样本 x_i 与神经元之间的权重为：

$W_j = \{w_{ji} : j = 1, 2, \cdots, N; i = 1, 2, \cdots, D\}$，其中 N 是神经元的总数，D 是输入数据的总数。

权重向量最接近输入向量的神经元被宣告为获胜神经元。为了简化这个问题，针对竞争层中的某一个神经元 node i 而言，输入数据是：

$$X = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_d \end{Bmatrix}$$

从数据集 X 到神经元 node i 的权重为：

$$W_i = \begin{Bmatrix} w_{i1} \\ w_{i2} \\ w_{i3} \\ \vdots \\ w_{id} \end{Bmatrix}$$

那么 X 到神经元 node i 的距离是：

$$d_i = ||X - W_i||$$

那么我们需要找到的是 X 到所有 N 个神经元的距离中的最短距离，也就是：

$$min(d_1, d_2, \cdots, d_n)$$

然而距离公式可以推到如下：

$$d_i^2 = ||X - W_i||^2 = X^T X + W_i^T W_i - 2W_i^T X$$

上述公式的前两项都是固定值，所以要计算样本到神经元的距离的最小值，就是计算$W_i^T X$的最大值。

（2）合作过程

合作过程就是寻找获胜神经元所在的拓扑领域的过程。拓扑领域的获取与获胜神经元与其他神经元的纵横距离相关。该拓扑领域是伴随迭代次数动态减少的，或者说，该拓扑领域是随着时间的推移动态衰减的。

（3）适应过程

适应过程是 SOM 的学习过程，是权重值的更新过程。通过这个过程，输出结点自组织，形成输入与输出之间的特征映射。这

一过程的一个特点是，不仅获胜的神经元能够得到权重更新，它的邻居的权重也将被更新，并且距离获胜神经元越远，更新的幅度越小。

权重更新的规则是：

$$w_j(t+1) = w_j(t) + \eta(t)h_{j,i(x)}[x(t) - w_j(t)]$$

其中：

- 该公式表示对神经元 j 以及其拓扑领域内的其他神经元再第 t 次跌点中的更新；
- h 表示其拓扑领域，这是一个正数；
- 学习率η(t)同样也是一个正数，并且是一个随着时间推移而衰减的正数。其取值范围在 0 到 1 之间。

2. 数据预处理

本算法通过 Python3 实现。

在程序开始前，先要对数据进行预处理。前面已经提到，通常情况下 MNIST 数据集中存储的是 uint8 格式的灰度数据，大小从 0 到 255，我们需要的是 0-1 之间的 float 类型。但是由 MNIST 官方提供了这一类型的数据，所以这一步可以省略。

另外，在神经网络的训练工作中，数据的不同评价指标往往具有不同的量纲和量纲单位，这样的情况会影响到模型的训练以及最终的结果，为了消除这种影响，需要对数据进行归一化处理。原始数据经过数据归一化处理之后，更方便进行综合对比评价。

下表给出了归一化的代码（Python3），包括数据归一化以及权重归一化。

```
def normal_X(X):
    #数据归一化处理
    N, D = X.shape
    for i in range(N):
        temp = np.sum(np.multiply(X[i], X[i]))
        X[i] /= np.sqrt(temp)
    return X
```

```
def normal_W(W):
    #权重归一化处理
    N, D = X.shape
    for i in range(W.shape[1]):
        temp = np.sum(np.multiply(W[:,i], W[:,i]))
        W[:, i] /= np.sqrt(temp)
    return W
```

3. SOM 算法步骤

SOM 算法的过程可以总结如下：

（1）初始化：设置竞争层神经元数量。初始化权重值（为权重分配随机值）。例如在本文实现的算法中，神经元的数量是 100 个（即 SOM 竞争层为 10 行×10 列的平面），权重矩阵 W 是一个784×100的矩阵，784 是样本特征，之所以是 784 行是为了计算$W_i^T X$方便。

（2）采样：确定从输入空间中抽取的 batch 的数量以及迭代次数。本算法中设置 batch 为 1000，迭代次数为 1000 次。

（3）匹配：找到权重向量最接近输入向量的获胜神经元。

（4）更新：更新权重向量。

（5）迭代：继续回到步骤（2），直到迭代次数结束。

各个步骤中对应的代码如下所示：

（1）获取获胜神经元的拓扑领域的代码

```
def getneighbor(self, index, N):
    #获得获胜神经元的拓扑领域
    a, b = self.output
    length = a*b
    def distence(index1, index2):
        i1_a, i1_b = index1 // a, index1 % b
        i2_a, i2_b = index2 // a, index2 % b
        return np.abs(i1_a - i2_a), np.abs(i1_b - i2_b)
```

```
    ans = [set() for i in range(N+1)]
    for i in range(length):
        dist_a, dist_b = distence(i, index)
        if dist_a <= N and dist_b <= N:
            ans[max(dist_a, dist_b)].add(i)
    return ans
```

（2）权重更新代码

```
def GetN(self, t):
    a = min(self.output)
    return int(a - float(a) * t/self.iteration)

def Geteta(self, t, n):
    #依赖于迭代次数的学习率
    return int(a - float(a) * t/self.iteration)

def updata_W(self, X, t, winner):
    #更新权重，包括获胜神经元的权重以及获胜神经元所处拓扑领域的权重
    N = self.GetN(t)
    for x, i in enumerate(winner):
        to_update = self.getneighbor(i, N)
        for j in range(N+1):
            e = self.Geteta(t, j)
            for w in to_update[j]:
                self.W[:, w] = np.add(self.W[:,w], e*(X[x,:] - self.W[:,w]))
```

（3）训练过程的代码

```
def train(self):
```

```
        #训练过程
        count = 0
        list_mse= []
        while self.iteration > count:
            train_X=self.X[np.random.choice(self.X.shape[0],
self.batch_size)]
            normal_W(self.W)
            normal_X(train_X)
            train_Y = train_X.dot(self.W)
            winner = np.argmax(train_Y, axis=1).tolist()
            self.updata_W(train_X, count, winner)
            count += 1
            mse = np.min(train_Y,axis=1).sum()/200
            list_mse.append(mse)
        return self.W,list_mse
```

（4）模型被成功训练出之后对数据进行聚合的代码

```
def cluster(self, X):
        X = normal_X(X)
        m = X.shape[0]
        cluster_labels = []
        for i in range(m):
            dis = X[i].dot(normal_W(self.W))
            re_dis = dis.reshape(self.output[0],self.output[1])
            l = np.where(re_dis == np.max(re_dis))
            cluster_labels.append(l)
        return np.array(cluster_labels)
```

三. 实验结果

1. 数据集

前面已经提到过，由于 SOM 是无监督学习的神经网络模型，所以不需要将数据集分解成训练集与测试集。但是由于模型需要多次迭代，所以本文代码中设置 1000 条数据为一个 batch，每次迭代都是从数据集中随机抽取 1000 条数据进行模型训练。

2. SOM 模型竞争层训练结果

由于数据集是将图片切割成 784 个像素点然后将每个像素点数值化后得到的，那么竞争层的每个神经元都可以转化成一张包含一个数字的图片，整个竞争层最后的结果将可以转化成一个包含$10 \times 10 = 100$个数字的图片，相同的数字集中在一起构成一个区域。这就是 SOM 模型训练成功的样子，类似于生物神经网络，当一张包含一个数字的图片被输入 SOM 模型之后，该数字将会落在竞争层相应的区域之上，就像生物神经网络在接收外界刺激后相应的区域会被激活一样。

另外，文档“final.txt”中保存的是最终的，模型训练结束后的权重值；文档“initial.txt”中保存的是初始状态下随机给定的权重值。

（1）首先绘制模型的初始状态。初始状态即是 SOM 竞争层刚被建成，权重被随机分配数值时的状态，那么此时的状态应当是紊乱的、无序的、无结构的。该状态可以见下图：

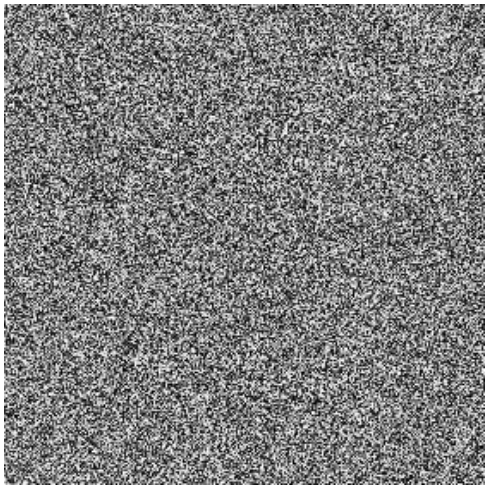

图 2 未训练的SOM竞争层可视化

（2）绘制模型的最终状态。下图是 SOM 竞争层的最终状态，可以看出此时的竞争层已经有了相对完整、清晰的拓扑结构，同一个数字处于同一个区域中。但是同样也可以看出，该算法的表现并不完美，图中有一部分“4”与“9”没有区分开。另外，这里需要注意的是，由于初始的权重值是随机的，所以每次运行时，最终的结构不尽相同。

图 3 经过训练后的 SOM 竞争层可视化

（3）对 SOM 进行测试。测试一下 SOM 竞争层的效果。这里为了可视化，选择抽取一条 mnist 中的数据进行测试并将结果可视化。首先随机从 mnist 中抽取一条数据，将其可视化之后可以看出，是一个手写的数字“4”：

然后利用算法中的聚类算法，输出这条数据在 SOM 竞争层中的坐标为：[3,6]。那么也就是，这条数据落在了 SOM 拓扑结构的第 1+3=4 行第 1+6=7 列上（注意 Python 中矩阵下标从 0 开始）。那么从图 3 可以看出，第 4 行第 7 列是“4”所在的拓扑范围。

3. 模型表现

本算法在训练模型的过程中加入了对均方误差（MSE）的计算，并将每次 MSE 的结果保存在文档“train.txt”中。最后算法调用“train.txt”文档并使用 Python3 的 matplotlib 模块将 MSE 绘制成折线图，横轴表示迭代次数，纵轴表示每次迭代的 MSE 的值，迭代结果如下图所示：

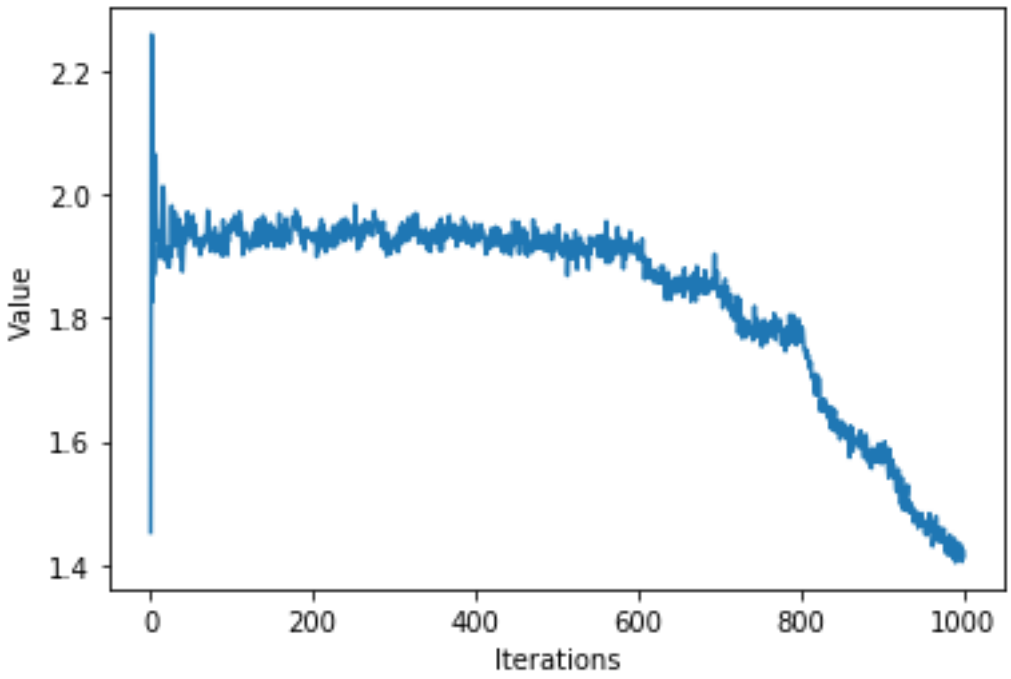

图 4 MSE 与迭代次数的关系

从图上可以看出，开始的 500 次迭代效果并不理想，但是 500 次迭代之后，MSE 的值开始迅速下降，直到接近第 1000 次降至最低。可以预见 1000 次之后模型效果将趋于稳定。