作业一报告

> 姓名：徐恒达
>
> 学号：1120151811

作业一报告
- 一、监督学习
  - 1. 概述
  - 2. 数据集
  - 3. 多层感知器网络MLP
  - 4. 随机梯度下降SGD
    - 随机梯度下降
    - 动量
  - 5. 自适应学习率Adam算法
  - 4. 确定隐藏层节点数
  - 5. 确定学习率
  - 6. 训练模型
- 二、非监督学习
  - 1. 算法比较
  - 2. 降维可视化

一、监督学习

1. 概述

解决手写数字字符识别问题，给出一张有手写数字字符的灰度图像，算法输出对图像的识别结果，即输出0~9中的一个数字。

采用多层感知器模型，反相传播算法计算最终结果对网络中各个权重的导数，采用随机梯度下降的方法优化参数，目标是使得总的错误率尽量小。并在具体实践中进行了一些调整和优化。

使用Python编程语言实现，以numpy包中的多维数组(ndarray)为核心数据结构处理矩阵和向量，以sci-kit learn包为主要工具进行算法构建，并进行调参和性能度量，用matplotlib库进行效果的可视化展现，比对不同算法，不同参数对效果的影响。

2. 数据集

使用UCL ML手写数字字符数据集的一部分，数据被标注为10类，每类一种数字。数据包括来自于43个人的1797张图片，原始图像为32×32像素的尺寸，使用NIST的预处理工具将其转化为8×8像素的图像，每个像素有16种灰度值。

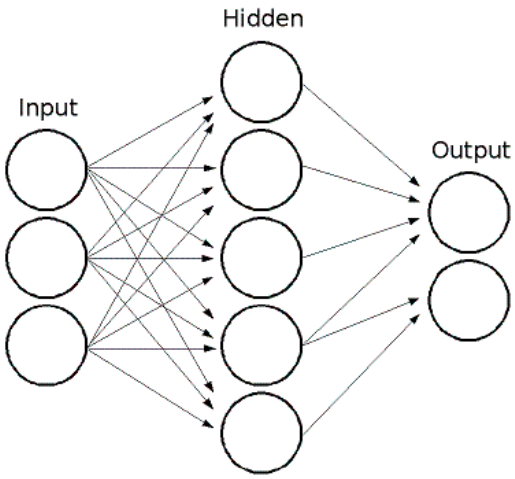
因此，数据集中共有1797个向量，每个向量的维度为64维，每一维度上有0到15共16种取值。所有的训练、调参、度量和测试都在这个数据集上展开。

3. 多层感知器网络MLP

搭建3层的感知器网络，输入图像为8×8像素，因此输入神经元有64个，每个神经元对应一个像素点；因为最终将数据分为10类，于是输出神经元有10个，最后对这个10个神经元的输出取softmax以决定最后的分类结果。

隐藏层的神经元数目在几百的量级上，具体数目将在下文中通过比对不同神经元数量对分类效果的影响来确定。

同时，对于梯度下降方法来说，学习率同样是对分类效果影响非常大的一个参数，对学习率的选择同样在下文中讨论。



4. 随机梯度下降SGD

随机梯度下降

梯度下降法是使得参数向着目标损失函数（loss function）变化最快的方向变化，即逆梯度方向变化，因此梯度下降法也称为最速下降法。但随着数据规模的增长，迭代训练一次的时间也随之变长，即存在训练数据量和算法效率上的矛盾。

按照数据生成分布抽取m个小批量（独立同分布的）样本，通过计算它们的梯度均值我们可以得到总体梯度的无偏估计，这就是随机梯度下降SGD方法。

SGD及相关的小批量抑或更广义的基于梯度优化的在线学习算法，一个重要的性质是每一步更新的计算时间不依赖于训练样本数量的多寡。即使训练样本数目非常大时，它们也能收敛。对于足够大的数据集，SGD可能会在处理整个训练集之前就收敛到最终测试集误差的某个固定容差范围内。

动量

虽然随机梯度下降效果不错，但其学习过程有时会很慢。动量方法旨在加速学习，特别是处理高曲率、小但一致的梯度，或是带噪声的梯度。动量算法累积了之前梯度算法累积了之前梯度指数级衰减的移动平均，并且继续延该方向移动。

5. 自适应学习率Adam算法

神经网络的学习率是难以设置的超参数之一，因为它对模型的性能有显著的影响。因为损失通常高度敏感于参数空间中的某些方向，而不敏感于其他。在这种情况下，如果我们相信方向敏感度在某种程度是轴对齐的，那么每个参数设置不同的学习率，在整个学习过程中自动适应这些学习率是有道理的。Adam (Kingma and Ba, 2014)就是一种学习率自适应的算法。

实验中分别采用带动量的随机梯度下降算法和adam算法进行测试，尝试不同的隐藏层节点数和学习率，以找到尽可能好的参数设置以达到尽可能好的效果。

4. 确定隐藏层节点数

使用网格搜索法调节隐藏层节点数，设置初始学习率为0.001，隐藏层的节点数从100到500变化，步长为20，使用随机梯度下降SGD和Adam算法各测试一编，比较测试结果。

评估方法采用交叉验证法(cross validation)，即将数据集划分为k个大小相似的互斥子集，每个子集尽可能保持数据分布的一致性。每次使用k-1个子集的并集作为训练集，余下的那个子集作为测试集，从而进行k次训练和测试，称为k折交叉验证。在本次实验中k取5，即对每种参数组合进行一轮5折交叉验证。

对性能的度量采用micro-f1 score指标。对于二分类问题，可将样例根据其实际类别与学习器预测类别的组合划分为真正例(true positive)、假正例(false positive)、真反例(true negative)、假反例(false negative)四种情形，令TP、FP、TN、FN分别为其对应的样例数，显然有TP+FP+TN+FN=样例总数。

查准率P与查全率R分别定义为

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

F1值就是P和R的调和平均数，即

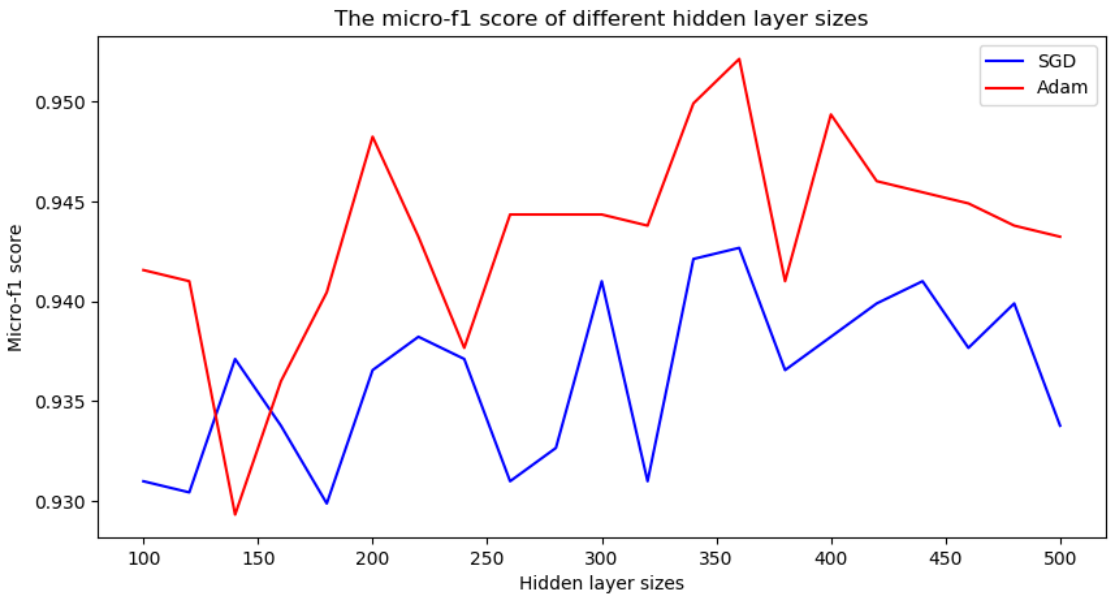
$$F1 = \frac{2 \times P \times R}{P + R}$$

对于手写字符数字识别这样的多分类任务，可将每一类分别作为正例进行统计，对每一类的计算TP、FP、TN和FN值，取平均值后再计算得出“微查准率”(micro-P)、“微查全率”(micro-R)和“微F1值”(micro-F1)。实验中采用微F1值作为性能指标进行度量。

核心代码如下：

```
clf = MLPClassifier(learning_rate_init=0.001)
hidden_layer_sizes = list(range(100, 501, 20))
param_grid = [{
    'solver': ['sgd', 'adam'],
    'hidden_layer_sizes': hidden_layer_sizes,
}]
grid = GridSearchCV(clf, param_grid, scoring='f1_micro', cv=5, n_jobs=4,
                    pre_dispatch='2*n_jobs', verbose=3, refit=False)
grid.fit(data, digits.target)
```

将计算任务划分为多个进程分布在不同的CPU上运行可以减少测试时间，实验中划分为4个进程。最终得到分别使用SGD和Adam算法时，micro-f1值与隐藏层节点数的关系，如下图所示：



从图中可以看出，隐藏层节点数与最终的性能没有太严密的关系，但大值在200到400个节点左右时训练效果更好一些；此外还可以看出，Adam比SGD整体效果更好，得分平均值更高，也更稳定。

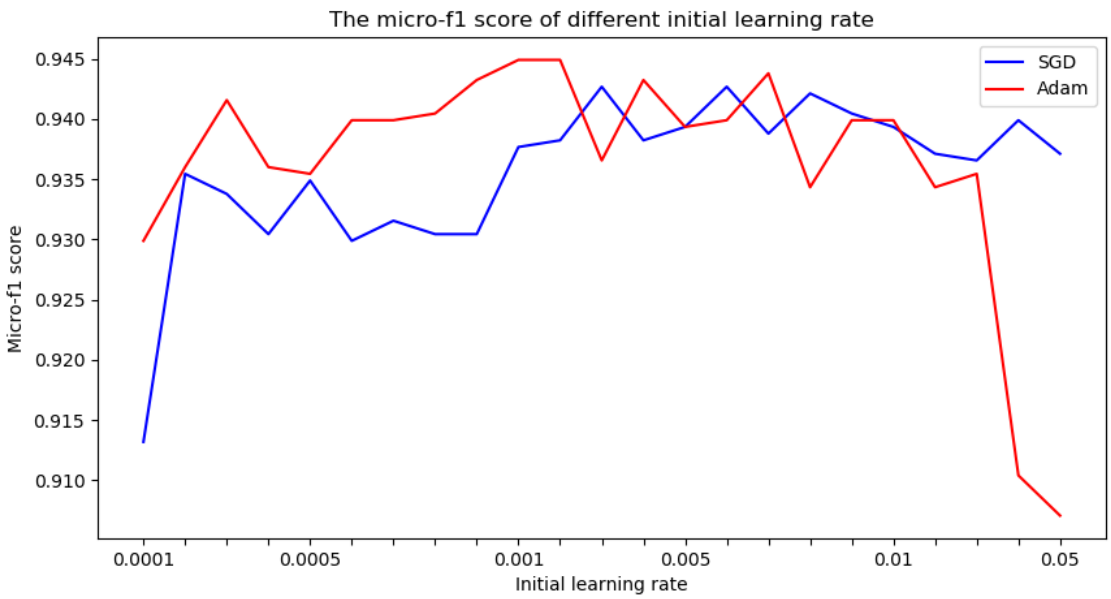
5. 确定学习率

同样采用5折交叉验证的评估方法和微f1值性能指标，测试初始学习率对训练效果的影响。隐藏层节点数固定为350个，初始学习率从0.0001到0.05进行变化。

核心代码如下：

```
clf = MLPClassifier(hidden_layer_sizes=350)
learning_rate_init = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005,
                      0.0006, 0.0007, 0.0008, 0.0009,
                      0.001, 0.002, 0.003, 0.004, 0.005,
                      0.006, 0.007, 0.008, 0.009,
                      0.01, 0.02, 0.03, 0.04, 0.05]
param_grid = [{
    'solver': ['sgd', 'adam'],
    'learning_rate_init': learning_rate_init
}]
grid = GridSearchCV(clf, param_grid, scoring='f1_micro', cv=5, n_jobs=4,
                    pre_dispatch='2*n_jobs', verbose=3, refit=False)
grid.fit(data, digits.target)
```

SGD和Adam算法的micro-f1值随初始学习率的变化曲线如下图所示：



从图中可以清晰地看出，初始学习率在0.001数量级时两算法均有较好的表现，学习率过高或过低都会导致训练效果明显下降。

6. 训练模型

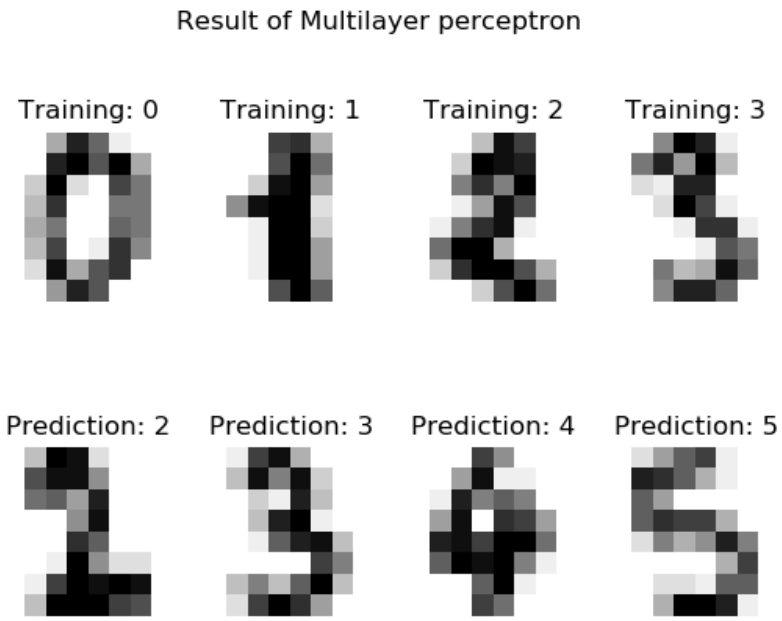
最终选择350个隐藏层节点数，Adam算法，初始学习率为0.001进行训练分类器，中止条件设为迭代200轮或连续两次迭代micro-f1值的变化量不超过0.001时停止。用数据集的80%进行训练模型，20%用来评估性能，得到的评估报告如下。从报告中可以看出识别率整体在94%左右。

```
Classification report:
      class  precision    recall  f1-score   support
          0       1.00      0.97      0.99        35
          1       0.85      0.97      0.91        36
          2       1.00      1.00      1.00        35
          3       1.00      0.76      0.86        37
          4       0.94      0.92      0.93        37
          5       0.93      1.00      0.96        37
          6       1.00      0.97      0.99        37
          7       0.92      0.94      0.93        36
          8       0.81      0.91      0.86        33
          9       0.94      0.92      0.93        37
avg / total       0.94      0.94      0.94       360
```

评测得到的混淆矩阵如下：

```
Confusion matrix:
[[34  0  0  0  1  0  0  0  0  0]
 [ 0 35  0  0  0  0  0  0  0  1]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 28  0  2  0  2  5  0]
 [ 0  2  0  0 34  0  0  0  0  1]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  1  0  0  0  0 36  0  0  0]
 [ 0  1  0  0  0  0  0 34  1  0]
 [ 0  2  0  0  1  0  0  0 30  0]
 [ 0  0  0  0  0  1  0  1  1 34]]
```

从数据集和训练集中各选择四个样本进行图示展示：



最后，作为对比，同样的数据集同样的评估指标，训练一个gamma值为0.001的支持向量机SVM模型进行测试，关于SVM的细节在这里不展开描述，此处仅作为对比参照。分类报告和混淆矩阵如下所示，可以看出SVM的识别率在96%左右，效果较多层感知器略胜一筹。

```
Classification report:
      class  precision    recall  f1-score   support
          0       1.00      0.97      0.99        35
          1       0.97      1.00      0.99        36
          2       1.00      1.00      1.00        35
          3       0.97      0.81      0.88        37
          4       0.97      0.92      0.94        37
          5       0.93      1.00      0.96        37
          6       1.00      1.00      1.00        37
          7       0.97      1.00      0.99        36
          8       0.84      0.94      0.89        33
          9       0.95      0.95      0.95        37
avg / total       0.96      0.96      0.96       360
```

```
Confusion matrix:
[[34  0  0  0  1  0  0  0  0  0]
 [ 0 36  0  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 30  0  2  0  1  4  0]
 [ 0  0  0  0 34  0  0  0  2  1]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 37  0  0  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  1  0  0  0  0  0  0 31  1]
 [ 0  0  0  1  0  1  0  0  0 35]]
```

二、非监督学习

1. 算法比较

与监督学习一样，解决手写数字字符识别问题，在同样的数据集上进行聚类，使用k-means算法。给定足够的时间，K-means将总是收敛，然而这可能会达到局部最小值。 这高度依赖于初始中心点的选择。 因此计算通常会进行多次，对中心的进行不同的初始化。解决这个问题的一种简单的方法是sci-kit learn中的一种名为k-means++初始化方案，将中心的初始化为彼此远离，从而导致比随机初始化更好的结果。

实验中分别采用k-means++初始化方法，随机初始化方法，基于PCA降维的处理方法进行聚类，比较它们的时间和均方误差值。因为数据的真实标签是已知的，实验中还计算了聚类结果的几个外部指标，分别是homogeneity score、completeness score、V measure、adjusted Rand index、adjusted mutual information和silhouette coefficient，不同指标的定义和细微差别不再展开描述，但结果是显而易见的，k-mean++初始化方法比起随机初始化方法效果有一些提高，使用PAC预处理之后，收敛时间明显减少了，各方面指标也具有较明显的提高。具体数据如下表所示：

```
init         time    inertia homo    compl   v-meas  ARI     AMI
silhouette
----         ----    ------- ----    -----   ------  ---     ---      -------
---
k-means++    0.56s   69742   0.662   0.711   0.685   0.540   0.658    0.161
random       0.39s   69683   0.674   0.713   0.693   0.562   0.671    0.147
PCA-based    0.07s   70794   0.666   0.694   0.680   0.552   0.663    0.139
```

2. 降维可视化

为了更直观地观察效果，这次首先用PCA降维将64维的数据降到2维，然后使用k-means算法将其分为10类。因为数据已经降到了2维，就可以在一张平面图中表示每个数据，将k-means聚类后的每一类用一种颜色标注，类别的中心用白色×表示，得到下图。