实验三：手写数字识别

学号:2018329621262　　姓名:刘恒玮

一、实验目的

理解 MLP 模型，使用 MLP 模型解决实际问题。

二、实验环境

PC 机，Python，numpy 库，sklearn 库。

三、实验内容

给定训练数据，利用 MLP 模型对手写数字进行分类。

四、实验原理（算法）

多层感知机（MLP，Multilayer Perceptron）除了输入输出层，它中间可以有多个隐层，最简单的 MLP 只含一个隐层，即三层的结构。多层感知机层与层之间是全连接的：上一层的任何一个神经元与下一层的所有神经元都有连接。多层感知机最底层是输入层，中间是隐藏层，最后是输出层。隐藏层的神经元怎么得来？首先它与输入层是全连接的，假设输入层用向量 X 表示，则隐藏层的输出就是 $f(\omega_1 x+b_1)$，ω_1 是权重（也叫连接系数），b_1 是偏置，函数 f 可以是常用的 sigmoid 函数或者 tanh 函数。其实隐藏层到输出层可以看成是一个多类别的逻辑回归，也即 softmax 回归，所以输出层的输出就是 $\text{softmax}(\omega 2x_1+b_2)$，$x_1$ 表示隐藏层的输出 $f(\omega_1 x+b_1)$。因此，MLP 所有的参数就是各个层之间的连接权重以及偏置，包括 ω_1、b_1、ω_2、b_2。对于一个具体的问题，怎么确定这些参数？求解最佳的参数是一个最优化问题，解决最优化问题，最简单的就是梯度下降法了（SGD）：首先随机初始化所有参数，然后迭代地训练，不断地计算梯度和更新参数，直到满足某个条件为止（比如误差足够小、迭代次数足够多时）。这个过程涉及到代价函数、规则化（Regularization）、学习速率（learning rate）、梯度计算等。

五、实验步骤（分析过程）

1. 导入实验过程中所需要的相关库。

[1]:
```
import gzip
import numpy as np
import struct
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
```

2. 编写导入数据的函数。

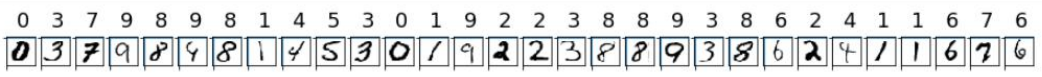
[2]:
```
# Load compressed MNIST gz files and return numpy arrays
def load_data(filename, label=False):
    with gzip.open(filename) as gz:
        struct.unpack('I', gz.read(4))
        n_items = struct.unpack('>I', gz.read(4))
        if not label:
            n_rows = struct.unpack('>I', gz.read(4))[0]
            n_cols = struct.unpack('>I', gz.read(4))[0]
            res = np.frombuffer(gz.read(n_items[0] * n_rows * n_cols), dtype=np.uint8)
            res = res.reshape(n_items[0], n_rows * n_cols)
        else:
            res = np.frombuffer(gz.read(n_items[0]), dtype=np.uint8)
            res = res.reshape(n_items[0], 1)
    return res
```

3. 导入手写数字的训练集和测试集。

[3]:
```
X_train = load_data("data/MNIST/train-images-idx3-ubyte.gz") / 255.0
X_test = load_data("data/MNIST/t10k-images-idx3-ubyte.gz") / 255.0
y_train = load_data("data/MNIST/train-labels-idx1-ubyte.gz",True).reshape(-1)
y_test = load_data("data/MNIST/t10k-labels-idx1-ubyte.gz",True).reshape(-1)
```

4. 随机在训练集中选取 30 个数据，展示出手写数字图像及其标签。

[4]:
```
count = 0
sample_size = 30
plt.figure(figsize=(16, 6))
for i in np.random.permutation(X_train.shape[0])[:sample_size]:
    count = count + 1
    plt.subplot(1, sample_size, count)
    plt.axhline('')
    plt.axvline('')
    plt.text(x=10, y=-10, s=y_train[i], fontsize=18)
    plt.imshow(X_train[i].reshape(28, 28), cmap=plt.cm.Greys)
plt.show()
```



5. 输出手写数字图像数据集和测试集的数量。

```
[5]: print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, sep = '\n')
```

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
```

6. 利用 help 查看 MLPClassifier 的各项参数的意义。

```
[6]: help(MLPClassifier)
```

```
Help on class MLPClassifier in module sklearn.neural_network._multilayer_perceptron

class MLPClassifier(sklearn.base.ClassifierMixin, BaseMultilayerPerceptron)
 |  MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', alph
01, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose
ng=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_
 |
 |  Multi-layer Perceptron classifier.
 |
 |  This model optimizes the log-loss function using LBFGS or stochastic
 |  gradient descent.
 |
 |  .. versionadded:: 0.18
 |
 |  Parameters
 |  ----------
 |  hidden_layer_sizes : tuple, length = n_layers - 2, default=(100,)
 |      The ith element represents the number of neurons in the ith
 |      hidden layer.
 |
 |  activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
 |      Activation function for the hidden layer.
 |
 |      - 'identity', no-op activation, useful to implement linear bottleneck,
 |        returns f(x) = x
 |
 |      - 'logistic', the logistic sigmoid function,
 |        returns f(x) = 1 / (1 + exp(-x)).
 |
 |      - 'tanh', the hyperbolic tan function,
 |        returns f(x) = tanh(x).
 |
 |      - 'relu', the rectified linear unit function,
 |        returns f(x) = max(0, x)
```

7. 采用 MLP 模型进行预测，并利用 GridSearchCV 选取相对最优的参数并输出训练集和验证集准确度。

[16]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {"hidden_layer_sizes": [(50,),(100,)],
                              "solver": ['adam', 'sgd', 'lbfgs'],
                              "max_iter": [200],
                              "verbose": [True],
                               "learning_rate_init":[0.001]
                              }
mlp = MLPClassifier()
grid = GridSearchCV(mlp, param_grid, refit = True, cv = 5, n_jobs = -1)
grid.fit(X_train, y_train)
```

[16]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=200, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state...e, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'hidden_layer_sizes': [(50,), (100,)],
                         'learning_rate_init': [0.001], 'max_iter': [200],
                         'solver': ['adam', 'sgd', 'lbfgs'],
                         'verbose': [True]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

[18]:
```
print("Training set score: %f" % grid.score(X_train, y_train))
print("Test set score: %f" %  grid.score(X_test, y_test))
```

```
Training set score: 1.000000
Test set score: 0.975400
```

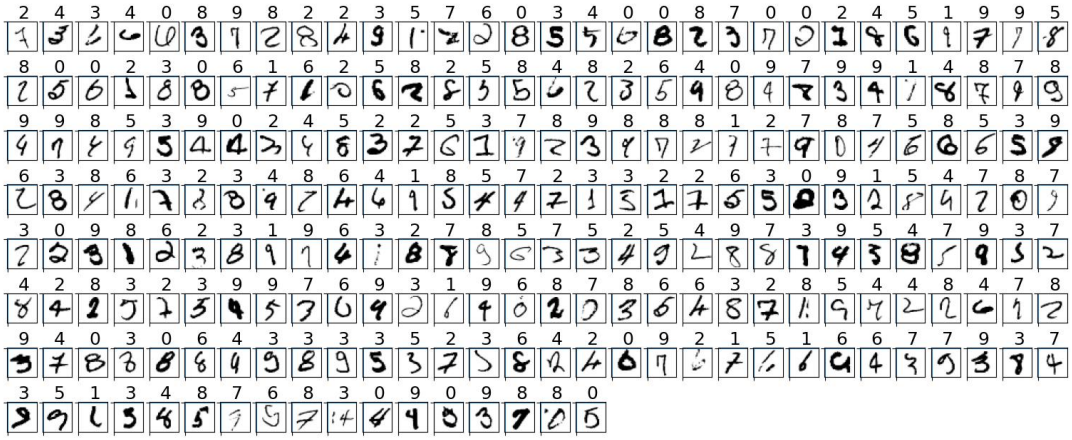
8. 将测试集中预测错误的数据的图片和预测标签输出。

六．实验结果（训练集准确度，验证集准确度等指标）

训练集和验证集的准确度：

```
Training set score: 1.000000
Test set score: 0.975400
```

预测错误的数据的图片和预测标签：



七．实验心得（碰到的问题，如何解决，与 SVM 算法进行对比，总结 MLP 算法的优缺点）

遇到的问题：

利用 GridSearchCV 选取参数时，程序运行时间较久，只选取一些关键的参数，比如隐藏层的数量和大小 hidden_layer_sizes，参数 slover：{‘lbfgs’，‘sgd’，‘adam’}，默认为 adam。权重优化的求解器：lbfgs 是准牛顿方法族的优化器；sgd 指的是随机梯度下降 adam 是指基于随机梯度的优化器。adam 在相对较大的数据集方面在训练时间和验证分数方面都能很好地工作，但是对于小型数据集，lbfgs 可以更快地收敛并且表现更好。

MLP 算法的优缺点：

（1）SVM 可以发现全局最优，而 MLP 算法通常只能获得局部最优。

（2）SVM 算法比较适合二分类问题，而对于对分类问题 MLP 算法会更加理想。

（3）SVM 算法是经典的分类算法，它内部可以通过核函数来增加特征抽象能力，因此很适合特征数比较小的数据集。MLP 算法是一种神经网络，这种网状结构特征抽象能力强，但是对算力的要求较高，适合大规模数据集。