1.可视化数据集

In [1]:

```
import numpy as np
import scipy.special
import matplotlib.pyplot as plt
import os
import struct
```

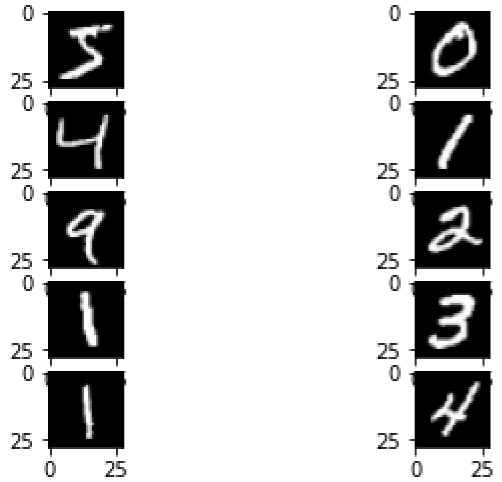
In [2]:

```
## 定义读取MNIST 数据
def load_mnist(path, kind='train'):
    """Load MNIST data from `path`"""
    labels_path = os.path.join(path,
                               '%s-labels.idx1-ubyte'
                               % kind)
    images_path = os.path.join(path,
                               '%s-images.idx3-ubyte'
                               % kind)
    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II',
                                 lbpath.read(8))
        labels = np.fromfile(lbpath,
                             dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack('>IIII',
                                               imgpath.read(16))
        images = np.fromfile(imgpath,
                             dtype=np.uint8).reshape(len(labels), 784)
    ## 返回MNIST图像序列，以及label
    return images, labels
```

In [3]:

```
## 读取数据,训练集、测试集
X_train, Y_train = load_mnist("C:\\Users\\Lenovo\\Desktop\\pics\\Day04_Baseline\\MNIST\\", "train")
X_test, Y_test = load_mnist("C:\\Users\\Lenovo\\Desktop\\pics\\Day04_Baseline\\MNIST\\", "t10k")
for i in range(1,11):
    plt.subplot(5,2,i)
    pic=X_train[i-1].reshape(28,28)
    plt.imshow(pic,cmap="gray")
```



2.尝试修改节点个数

In [4]:

```
## 定义前馈全连接神经网络
class NetworkNP:
    ## 网络初始化
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        """
        :param inputnodes: 网络输入节点数
        :param hiddennodes: 隐藏层 节点数
        :param outputnodes: 输出层 节点数
        :param learningrate: 学习率
        :param weit_simple:
        """
        # 网络层定义，以及初始化
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        self.lr = learningrate
        # 权重初始、偏执初始化
        self.w_ih = (np.random.rand(self.hnodes, self.inodes) - 0.5)
        self.w_ho = (np.random.rand(self.onodes, self.hnodes) - 0.5)

        # 激活函数定义Sigmoid
        self.activaltion_function = lambda x: scipy.special.expit(x)

    # 交叉熵损失函数 输入即为网络的输出结果 Y正确的类别 返回交叉熵损失值
    def cross_entropy(self, out, Y):
        loss = np.sum(np.nan_to_num(-Y*np.log(out)-(1-Y)*np.log(1-out)))
        return loss

    # 多分类softmax 返回预测的概率
    def softmax(self, x):
        max_per_row = np.max(x)
        exp_scores = np.exp(x - max_per_row)
        probs = exp_scores / np.sum(exp_scores)
        return probs

    def train(self, inputs_list, targets_list):
        """
        训练函数
        :param inputs_list: 输入数据
        :param targets_list: 输出返回数据
        """
        ## 前向传播过程
        inputs = np.array(inputs_list, ndmin=2).T
        targets = np.array(targets_list, ndmin=2).T
        hidden_inputs = np.matmul(self.w_ih, inputs)
        hidden_outputs = self.activaltion_function(hidden_inputs)
        final_inputs = np.dot(self.w_ho, hidden_outputs)
        # print(final_inputs)
        final_outputs = self.softmax(final_inputs)

        output_errors = targets - final_outputs
        hidden_errors = np.dot(self.w_ho.T, output_errors)
        # 反向梯度更新过程 这里使用的反向传播用的是S型函数的公式
        self.w_ho += self.lr * np.dot((output_errors * final_outputs * (1.0 - final_outputs)),
                                        np.transpose(hidden_outputs))
        self.w_ih += self.lr * np.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), np
        return self.cross_entropy(final_outputs, targets)

    def query(self, inputs_list):
```

```
        inputs = np.array(inputs_list, ndmin=2).T
        hidden_inputs = np.dot(self.w_ih, inputs)
        hidden_outputs = self.activaltion_function(hidden_inputs)
        final_inputs = np.dot(self.w_ho, hidden_outputs)
        final_outputs = self.activaltion_function(final_inputs)
        return final_outputs
```

In [5]:

```
a = NetworkNP(784, 100, 10, 0.2)
b = NetworkNP(784, 200, 20, 0.25)
c = NetworkNP(784, 150, 10, 0.3)
```

3.Epoch

Epoch指数据完全通过神经网络的次数的次数，理论上epoch增加可以使得模型由欠拟合逐渐变得过拟合

In [6]:

```
input_nodes = 784   ## 图像大小为28*28
hidden_nodes = 200  ## 隐藏层，200个节点
output_nodes = 10   ## 输出层，10个节点分别表示0-9 10个数字
learning_rate = 0.2 ## 学习率

## 调用神经网络
n = NetworkNP(input_nodes, hidden_nodes, output_nodes, learning_rate)

## 读取数据,训练集、测试集
X_train, Y_train = load_mnist("C:\\Users\\Lenovo\\Desktop\\pics\\Day04_Baseline\\MNIST\\", "train")
X_test, Y_test = load_mnist("C:\\Users\\Lenovo\\Desktop\\pics\\Day04_Baseline\\MNIST\\", "t10k")
print("训练数据维度为：", X_train.shape, Y_train.shape) ## 输出数据维度
print("测试数据维度为：", X_test.shape, Y_test.shape)
```

```
训练数据维度为： (60000, 784) (60000,)
测试数据维度为： (10000, 784) (10000,)
```

In [7]:

```
epochs = 1
for e in range(epochs):
    cnt = 0
    err = 0.0
    for record in zip(X_train, Y_train):
        # 数据初始化, 把图像归一化到0-1
        inputs = (np.array(record[0]).astype(np.float) / 255.0 * 0.99) + 0.01
        # One-hot编码
        targets = np.zeros(output_nodes) + 0.01
        targets[int(record[1])] = 0.99
        ## 同学们可以在此处添加类别, 判断训练是否预测正确
        error= n.train(inputs, targets)
        err += error
        cnt+=1
        if cnt%1000 == 0:
            print("[Epoch %d/%d] 训练误差为 %.6f"%(cnt, X_train.shape[0], err.mean()/1000))
            err     = 0                           # 误差归零
    print("Epoch %d 完成训练"%e)
```

```
[Epoch 1000/60000] 训练误差为 1.910869
[Epoch 2000/60000] 训练误差为 1.431768
[Epoch 3000/60000] 训练误差为 1.330386
[Epoch 4000/60000] 训练误差为 1.285101
[Epoch 5000/60000] 训练误差为 1.327228
[Epoch 6000/60000] 训练误差为 1.274274
[Epoch 7000/60000] 训练误差为 1.307037
[Epoch 8000/60000] 训练误差为 1.359329
[Epoch 9000/60000] 训练误差为 1.478377
[Epoch 10000/60000] 训练误差为 1.288809
[Epoch 11000/60000] 训练误差为 1.284226
[Epoch 12000/60000] 训练误差为 1.340391
[Epoch 13000/60000] 训练误差为 1.386112
[Epoch 14000/60000] 训练误差为 1.438848
[Epoch 15000/60000] 训练误差为 1.379310
[Epoch 16000/60000] 训练误差为 1.338733
[Epoch 17000/60000] 训练误差为 1.263426
[Epoch 18000/60000] 训练误差为 1.369254
[Epoch 19000/60000] 训练误差为 1.175027
[Epoch 20000/60000] 训练误差为 1.253076
[Epoch 21000/60000] 训练误差为 1.300072
[Epoch 22000/60000] 训练误差为 1.200723
[Epoch 23000/60000] 训练误差为 1.299671
[Epoch 24000/60000] 训练误差为 1.297641
[Epoch 25000/60000] 训练误差为 1.316267
[Epoch 26000/60000] 训练误差为 1.232587
[Epoch 27000/60000] 训练误差为 1.309295
[Epoch 28000/60000] 训练误差为 1.307301
[Epoch 29000/60000] 训练误差为 1.239371
[Epoch 30000/60000] 训练误差为 1.326126
[Epoch 31000/60000] 训练误差为 1.316571
[Epoch 32000/60000] 训练误差为 1.377571
[Epoch 33000/60000] 训练误差为 1.313260
[Epoch 34000/60000] 训练误差为 1.244919
[Epoch 35000/60000] 训练误差为 1.333314
[Epoch 36000/60000] 训练误差为 1.302133
[Epoch 37000/60000] 训练误差为 1.200883
[Epoch 38000/60000] 训练误差为 1.407243
[Epoch 39000/60000] 训练误差为 1.227602
```

```
[Epoch 40000/60000] 训练误差为 1.297110
[Epoch 41000/60000] 训练误差为 1.256840
[Epoch 42000/60000] 训练误差为 1.370565
[Epoch 43000/60000] 训练误差为 1.392834
[Epoch 44000/60000] 训练误差为 1.237327
[Epoch 45000/60000] 训练误差为 1.298518
[Epoch 46000/60000] 训练误差为 1.389517
[Epoch 47000/60000] 训练误差为 1.413698
[Epoch 48000/60000] 训练误差为 1.322328
[Epoch 49000/60000] 训练误差为 1.309240
[Epoch 50000/60000] 训练误差为 1.311208
[Epoch 51000/60000] 训练误差为 1.325675
[Epoch 52000/60000] 训练误差为 1.272140
[Epoch 53000/60000] 训练误差为 1.435167
[Epoch 54000/60000] 训练误差为 1.275664
[Epoch 55000/60000] 训练误差为 1.205968
[Epoch 56000/60000] 训练误差为 1.228863
[Epoch 57000/60000] 训练误差为 1.246558
[Epoch 58000/60000] 训练误差为 1.268545
[Epoch 59000/60000] 训练误差为 1.153812
[Epoch 60000/60000] 训练误差为 1.147152
Epoch 0 完成训练
```

In [8]:

```
epochs = 2
for e in range(epochs):
    cnt = 0
    err = 0.0
    for record in zip(X_train, Y_train):
        # 数据初始化, 把图像归一化到0-1
        inputs = (np.array(record[0]).astype(np.float) / 255.0 * 0.99) + 0.01
        # One-hot编码
        targets = np.zeros(output_nodes) + 0.01
        targets[int(record[1])] = 0.99
        ## 同学们可以在此处添加类别, 判断训练是否预测正确
        error= n.train(inputs, targets)
        err += error
        cnt+=1
        if cnt%1000 == 0:
            print("[Epoch %d/%d] 训练误差为 %.6f"%(cnt, X_train.shape[0], err.mean()/1000))
            err     = 0                          # 误差归零
    print("Epoch %d 完成训练"%e)
```

```
[Epoch 39000/60000] 训练误差为 1.293857

[Epoch 40000/60000] 训练误差为 1.285237
[Epoch 41000/60000] 训练误差为 1.221875
[Epoch 42000/60000] 训练误差为 1.353992
[Epoch 43000/60000] 训练误差为 1.344747
[Epoch 44000/60000] 训练误差为 1.268285
[Epoch 45000/60000] 训练误差为 1.305151
[Epoch 46000/60000] 训练误差为 1.309223
[Epoch 47000/60000] 训练误差为 1.347002
[Epoch 48000/60000] 训练误差为 1.323420
[Epoch 49000/60000] 训练误差为 1.289740
[Epoch 50000/60000] 训练误差为 1.287329
[Epoch 51000/60000] 训练误差为 1.336786
[Epoch 52000/60000] 训练误差为 1.252403
[Epoch 53000/60000] 训练误差为 1.297288
[Epoch 54000/60000] 训练误差为 1.266265
[Epoch 55000/60000] 训练误差为 1.195680
[Epoch 56000/60000] 训练误差为 1.212944
[Epoch 57000/60000] 训练误差为 1.166318
[Epoch 58000/60000] 训练误差为 1.166717
```

In [9]:

```
epochs = 3
for e in range(epochs):
    cnt = 0
    err = 0.0
    for record in zip(X_train, Y_train):
        # 数据初始化, 把图像归一化到0-1
        inputs = (np.array(record[0]).astype(np.float) / 255.0 * 0.99) + 0.01
        # One-hot编码
        targets = np.zeros(output_nodes) + 0.01
        targets[int(record[1])] = 0.99
        ## 同学们可以在此处添加类别，判断训练是否预测正确
        error= n.train(inputs, targets)
        err += error
        cnt+=1
        if cnt%1000 == 0:
            print("[Epoch %d/%d] 训练误差为 %.6f"%(cnt, X_train.shape[0], err.mean()/1000))
            err     = 0                           # 误差归零
    print("Epoch %d 完成训练"%e)
```

```
[Epoch 1000/60000] 训练误差为 1.252901
[Epoch 2000/60000] 训练误差为 1.310770
[Epoch 3000/60000] 训练误差为 1.212178
[Epoch 4000/60000] 训练误差为 1.173024
[Epoch 5000/60000] 训练误差为 1.262143
[Epoch 6000/60000] 训练误差为 1.215345
[Epoch 7000/60000] 训练误差为 1.264772
[Epoch 8000/60000] 训练误差为 1.309632
[Epoch 9000/60000] 训练误差为 1.335365
[Epoch 10000/60000] 训练误差为 1.185021
[Epoch 11000/60000] 训练误差为 1.271008
[Epoch 12000/60000] 训练误差为 1.248407
[Epoch 13000/60000] 训练误差为 1.262120
[Epoch 14000/60000] 训练误差为 1.265926
[Epoch 15000/60000] 训练误差为 1.257513
[Epoch 16000/60000] 训练误差为 1.265849
[Epoch 17000/60000] 训练误差为 1.181073
[Epoch 18000/60000] 训练误差为 1.322866
[Epoch 19000/60000] 训练误差为 1.208183
```

In [10]:

```
# 4.输出网络检测准确率
```

In [11]:

```
epochs = 1
for e in range(epochs):
    cnt = 0
    err = 0.0
    for record in zip(X_train, Y_train):
        # 数据初始化, 把图像归一化到0-1
        inputs = (np.array(record[0]).astype(np.float) / 255.0 * 0.99) + 0.01
        # One-hot编码
        targets = np.zeros(output_nodes) + 0.01
        targets[int(record[1])] = 0.99
        ## 同学们可以在此处添加类别，判断训练是否预测正确
        error= n.train(inputs, targets)
        err += error
        cnt+=1
        if cnt%1000 == 0:
            print("[Epoch %d/%d] 训练误差为 %.6f"%(cnt, X_train.shape[0], err.mean()/1000))
            err     = 0                         # 误差归零
            print("网络准确率为：= ", cnt / float(len(Y_train)))
    print("Epoch %d 完成训练"%e)
```

```
[Epoch 1000/60000] 训练误差为 1.243687
网络准确率为：=  0.016666666666666666
[Epoch 2000/60000] 训练误差为 1.276924
网络准确率为：=  0.03333333333333333
[Epoch 3000/60000] 训练误差为 1.274603
网络准确率为：=  0.05
[Epoch 4000/60000] 训练误差为 1.152180
网络准确率为：=  0.06666666666666667
[Epoch 5000/60000] 训练误差为 1.253787
网络准确率为：=  0.08333333333333333
[Epoch 6000/60000] 训练误差为 1.221541
网络准确率为：=  0.1
[Epoch 7000/60000] 训练误差为 1.223085
网络准确率为：=  0.11666666666666667
[Epoch 8000/60000] 训练误差为 1.235268
网络准确率为：=  0.13333333333333333
[Epoch 9000/60000] 训练误差为 1.343013
网络准确率为：=  0.15
[Epoch 10000/60000] 训练误差为 1.222964
网络准确率为：=  0.16666666666666666
```

5.lenet（环境配置还没有完成）