BP 神经网络手写字体识别（Minst）

1.实验环境

- OS: Windows 10
- CPU: intel CORE i5
- Python 3.6
- Sckikit_learn 0.18.1
- 其他 Python 科学计算模块
- 编译器：PythonCharm 2017.3.3

2.主要参考文档：

- Sklearn 0.18 官方文档

3.实验过程

3.1 问题描述

BP 神经网络(Back Propagation)网络是 1986 年由 Rumelhart 和 McCelland 为首的科学家小组提出，是一种按照误差传播算法训练的多层前馈网络，是目前应用最为广泛的神经网络模型之一。BP 神经网络能学习和储藏大量的输入-输出模式映射关系，而无须事前揭示这种映射关系的数学方程，或者可以这样说，神经网络就是在一个函数。

Minst 问题是一个非常基础和简单的问题，指辨识一个手写数字，这个问题可以使用 BP 神经网络得到较好的解决。

3.2 实验资源库

本实验采用的资源库为 THE MNIST DATABASE，选取的训练集 60000 张数字图片，测试集选取了 10000 张数字图片，最后选取了 100 张图片进行了读取。所有的数字图片都包含了 28*28 的像素，每个像素点有两种取值：0 和 1，1 代表黑色，0 代表无色（白色）；

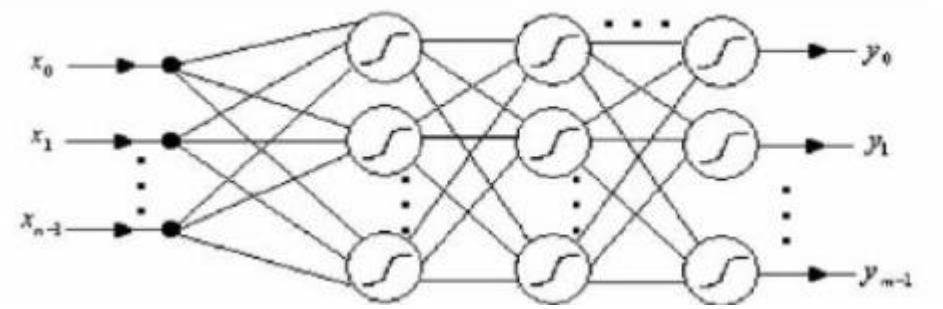
资源库网址：http://yann.lecun.com/exdb/mnist/

3.3 实验原理与过程：

神经网络这个概念天天都在谈，所以我就决定多做一个实验，写一写神经网络中的”Hello, world” 实验，使用 BP 神经网络来解决 Minst 问题。

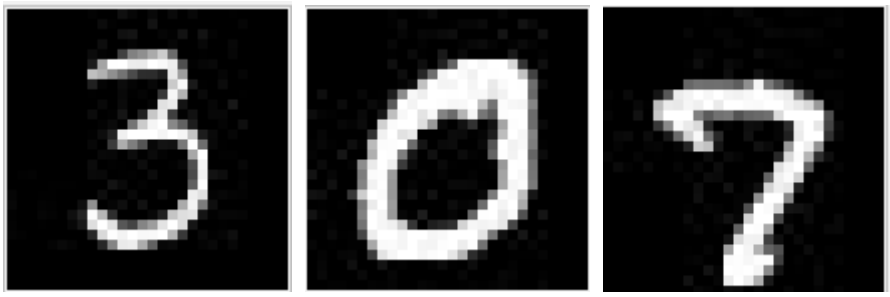
这个实验参考了一些互联网上的博客和 gitbub 等，并且在充分理解的基础上做出了自己的修改和补充。首先是 BP 神经网络在本实验的运用：

神经网络的基本结构如图所示：



包括输入层（input），隐藏层(hide layer)以及输出层(output layer)。输入层神经元的个数由样本属性的维度决定，输出层神经元的个数由样本的分类个数决定。隐藏层的层数和每层的神经元个数由用户指定。

在这个实验中：



每张数字图片都包含 28*28 个像素点。在官方网站下载的 Minst 文件都为二进制文件，得到上面的图片还需要进行二进制的文件转化为像素特征数据，将每张二进制图片的 Label 转化为数字特征，代码如下图所示

```
1.  def getLabel(self):
2.          """
3.          将 MNIST 中 label 二进制文件转换成对应的 label 数字特征
4.          """
5.          binFile = open(self._filename,'rb')
6.          buf = binFile.read()
7.          binFile.close()
8.          index = 0
9.          magic, numItems= struct.unpack_from(self._twoBytes2, buf,index)
10.         index += struct.calcsize(self._twoBytes2)
11.         labels = [];
12.         for x in range(numItems):
13.             im = struct.unpack_from(self._labelByte2,buf,index)
14.             index += struct.calcsize(self._labelByte2)
15.             labels.append(im[0])
16.         return np.array(labels)
17.
18.     def outImg(self, arrX, arrY):
19.         """
```

```
        根据生成的特征和数字标号，输出 png 的图像
        """
        m, n = np.shape(arrX)
        #每张图是 28*28=784Byte
        for i in range(1):
            img = np.array(arrX[i])
            img = img.reshape(28,28)
            outfile = str(i) + "_" +  str(arrY[i]) + ".png"
            plt.figure()
            plt.imshow(img, cmap = 'binary') #将图像黑白显示
            plt.savefig(self._outpath + "/" + outfile)
```

由于是 28*28 的输入像素点，所以输入层的神经元的个数为 784，由于输出的结果是 0 到 9 的一个数字，所以，设置输出层的神经元为 10 个，隐含层根据经验公式为 15 个神经元。

每一个神经元包含一个阈值θ_j，用来改变神经元的状态，网络中的弧线W_{ij}表示前一层神经元和后一层神经元之间的权值。每个神经元都有输入和输出。输入层和输出层都是训练样本的属性。

本实验中的神经网络配置如下。

```
 dsamp_num = len(sample)      # 样本总数
inp_num = len(sample[0])    # 输入层节点数
out_num = 10                # 输出节点数
hid_num = 15  # 隐层节点数(经验公式)
w1 = 0.2*np.random.random((inp_num, hid_num))- 0.1   # 初始化输入层权矩阵
w2 = 0.2*np.random.random((hid_num, out_num))- 0.1   # 初始化隐层权矩阵
hid_offset = np.zeros(hid_num)     # 隐层偏置向量
out_offset = np.zeros(out_num)     # 输出层偏置向量
inp_lrate = 0.2             # 输入层权值学习率
hid_lrate = 0.2             # 隐层权值学习习率
```

隐藏层和输出层的输入为$I_j = \sum_i W_{ij} O_i + \theta_j$，其中，$W_{ij}$是由上一层的单元 i 到单元 j 的连接的权值。$O_j$是上一层的单元 i 的输出；而$\theta_j$是单元 j 的阈值。

神经网络中的神经元的输出层是经由激活函数得到的。该符号表现单元代表的神经元活性。激活函数一般使用 simoid 函数。神经元的输出为：

$$O_j = \frac{1}{1 + e^{-I_j}}$$

除此之外需要有一个学习率，在 0 到 1 之间，其对学习的速率和效率有较大的影响。在这个实验中，输入层和学习层的学习率都取值为 0.2。

下面是本实验中的激活函数：

```
ddef get_act(x):
    act_vec = []
    for i in x:
        act_vec.append(1/(1+math.exp(-i)))
    act_vec = np.array(act_vec)
    return act_vec
```

接下来是 BP 网络的学习过程

对于样本中的每一个训练样本，都需要执行两个部分：向前传播输入和向后传播误差，并调整权值和阈值。

- 初始化 Network 的权值和阈值
- while(终止条件不满足){
- for sample 中的每一个训练样本 X{
- for 隐藏层或输出层每个神经元 j{ //向前传播输入
- $I_j = \sum_i W_{ij} O_i + \theta_j$ //相对于前一层单元 j 的净输入

  //$O_j = \frac{1}{1+e^{-I_j}}$计算单元 j 的输出
- }
- for 输出层每个单元 j{
- $E_{rr_j} = O_j(1 - O_j)(T_j - O_j)$ } //计算误差
- for 由最后一个到第一个隐藏层，对于隐藏层每一个单元 j{
- $E_{rr_j} = O_j(1 - O_j)\sum_k E_{rr_k} w_{kj}$; } //k 是 j 的下一层中的神经元
- for network 中每个权W_{ij}{
- $\Delta W_{ij} = (l) E_{rr_j} O_i;$ //权增值
- $W_{ij} = W_{ij} + \Delta W_{ij};$ //权更新
- }
- for network 中的每一个偏差 θ_j{
- $\Delta\theta_j = (l) E_{rr_j};$
- $\theta_j = \theta_j + \Delta\theta_j;$
- }
- }
- }

具体的流程代码如下

```
for count in range(0, samp_num):
    t_label = np.zeros(out_num)
    t_label[label[count]] = 1
```

```
4.
5.      #前向过程
6.      hid_value = np.dot(sample[count], w1) + hid_offset       # 隐层值
7.      hid_act = get_act(hid_value)                 # 隐层激活值
8.      out_value = np.dot(hid_act, w2) + out_offset             # 输出层值
9.      out_act = get_act(out_value)                 # 输出层激活值
10.
11.     #后向过程
12.     e = t_label - out_act                            # 输出值与真值间的误差
13.     out_delta = e * out_act * (1-
    out_act)                                         # 输出层 delta 计算
14.     hid_delta = hid_act * (1-
    hid_act) * np.dot(w2, out_delta)                    # 隐层 delta 计算
15.     for i in range(0, out_num):
16.         w2[:,i] += hid_lrate * out_delta[i] * hid_act    # 更新隐层到输出层权向
    量
17.     for i in range(0, hid_num):
18.         w1[:,i] += inp_lrate * hid_delta[i] * sample[count]      # 更新输出层
    到隐层的权向量
19.
20.     out_offset += hid_lrate * out_delta                              # 输出层
    偏置更新
21.     hid_offset += inp_lrate * hid_delta
22. print('Training Finished!')
```

整个实验的神经网络的训练过程可以分为三步：

1. 初始化网络权值和神经元的阈值（此实验中为随机初始化）
2. 前向传播：按照公式一层一层的计算隐藏层神经元的输出层神经元的输入与输出。
3. 后向传播：根据公式修正权值和阈值。

将训练好的网络的权值与阈值参数存储好了之后，计算两种误差，即训练误差和测试误差。

训练误差由训练集再次在网络中运行，得到结果，计算误差值；测试误差是将测试集在网络上运行，计算误差值。

测试误差代码部分如下：

```
1.  temp=0
2.  for count in range(len(sample)):
3.      hid_value = np.dot(sample[count], w1) + hid_offset       # 隐层值
4.      hid_act = get_act(hid_value)                 # 隐层激活值
5.      out_value = np.dot(hid_act, w2) + out_offset             # 输出层值
6.      out_act = get_act(out_value)                 # 输出层激活值
7.      if np.argmax(out_act) == label[count]:
8.          temp+=1
9.  print('Train_Set Error is: %.2f%%'%((1-float(temp)/len(sample))*100))
```

训练误差部分代码如下：

```
temp=0
for count in range(len(test_s)):
    hid_value = np.dot(test_s[count], w1) + hid_offset        # 隐层值
    hid_act = get_act(hid_value)                  # 隐层激活值
    out_value = np.dot(hid_act, w2) + out_offset              # 输出层值
    out_act = get_act(out_value)                  # 输出层激活值
    if np.argmax(out_act) == test_l[count]:
        temp+=1
print('Test_Set Error is: %.2f%%'%((1-float(temp)/len(test_s))*100))
```

最后，这个实验我运用得到的神经网络，对 100 张得到的图片进行了实际的测试，发现识别错误的数量 8 张。

3.4 实验结果分析与思考

得到的实验结果为：

```
Train_Set Error is: 8.60%
Test_Set Error is: 8.52%
```

训练误差和测试误差分别为 8.60%和 8.52%

这个结果相对来说还是较为理想的，通过 100 张的图片的观察，发现了识别错误的图片确实是在人为识别的情况下都会存在识别错误的，属于客观的情况。通过这个实验总结，在实际 BP 神经网络中还会出现以下几个问题：

1. 样本处理。对于输出，如果只有两类，那么输出为 0 和 1，只有当U_i趋于正负无穷大的时候才会输出 0 和 1；因此可以适当放宽松。输出>0.9 时就可以认为是 1，输出<0.1 时认为是 0。对于随意输入，在读入参数的时候一样需要做一个归一化处理。

2. 网络结构的选择。主要是指隐含层层数和神经元决定了网络规模，网络规模和性能学习效果密切相关。规模大，计算量大，而且可能导致过度拟合；但是规模小，也可能导致欠拟合。

3. 增量学习和批量学习。上面的实验过程都是基于批量学习的，批量学习适用于离线学习，学习效果稳定性好；增量学习使用在线学习，它对样本的噪声是比较敏感的，不适合剧烈变化的输入模式。

5. 对于激活函数和误差函数也有其他的选择，这个还需要更多的总结和尝试。

综合以上，其实 BP 针对特定的数据还有比较大的优化空间，特别是对于特定问题。