实验报告：手写数字识别

韦俊林 (201928016029023)

2020 年 5 月 5 日

1 摘要

使用 MxNet 框架构建一个经典的 LeNet 卷积神经网络，对 MNIST 手写数字数据集进行训练和评估，经过简单的调参，实现测试集准确率达到 98% 以上，经过这次实验初步了解基于 MxNet 框架搭建一个深度学习网络以及整个训练评估过程。

2 问题描述

深度学习是学习样本数据的内在规律和表示层次，这些学习过程中获得的信息对诸如文字，图像和声音等数据的解释有很大的帮助。它的最终目标是让机器能够像人一样具有分析学习能力，能够识别文字、图像和声音等数据。深度学习是一个复杂的机器学习算法，在语音和图像识别方面取得的效果，远远超过先前相关技术。深度学习是一门即注重理论又注重实践的学课，而基于 MNIST 数据集实现手写数字识别便是最好的入门数据集。

利用当前主流的框架构建一个规范的卷积神经网络，完成从编码、训练、调参到测试的全过程，在 MNIST 手写数字数据集上，实现测试集准确率达到 98% 以上。

3 解决方案

3.1 损失函数

本次实验是十个类别的分类任务，因此损失函数首先想到的是使用 softmax 交叉熵损失函数。Softmax Regression 是 Logistic 回归的推广，能以更加紧凑的方式来处理 Logistic 回归中所面临的多分类问题。因此，Softmax 回归的任务是给定 n 个训练样本 $(x_1, y_1), ..., (x_n, y_n)$，其中 $x_i \in R^d$，$i = [1, n]$ 为 d 维空间中的样本特征，$y_i \in \{1, 2, ..., c\}$ 为其对应的类别标签，训练出一个解决多分类问题的模型。给定测试样本 x，期望基于所学的假设函数能对每个类别 j 估算出概率值 $P(y = j|x)$。其以概率形式的假设函数为：

$$h(x; W) = \begin{pmatrix} P(y=1|x;W) \\ P(y=2|x;W) \\ ... \\ P(y=c|x;W) \end{pmatrix} = \frac{1}{\sum_{i=1}^{c} e^{w_i^T x}} = \begin{pmatrix} e^{w_1^T x} \\ e^{w_2^T x} \\ ... \\ e^{w_c^T x} \end{pmatrix} \in R^c$$

其中 x 为齐坐标表示，即 x 的维数是 $d+1$ 维，齐次坐标则对应着平移量 b，$W = [w_1, w_2, ..., w_c] \in R^{(d+1)c}$。

使用 softmax 运算后可以方便的与离散标签计算误差，这里我使用交叉熵损失函数，其中交叉熵为：

$$H(y^{(i)}, \hat{y}^{(i)}) = -\sum_{j=1}^{c} y_j^{(i)} \log \hat{y}_j^{(i)}$$

这里 $y^{(i)}$ 表示第 i 个样本的真实标签，$\hat{y}^{(i)}$ 表示第 i 个样本的预测标签。因此，有损失函数：

$$\zeta(\Theta) = \frac{1}{n} \sum_{i=1}^{n} H(y^{(i)}, \hat{y}^{(i)})$$

其中 Θ 代表模型的参数。

Gluon 提供了一个包括 softmax 运算和交叉熵损失函数计算的函数，它的数值稳定性更好，因此本次实验中直接调用该开源函数，SoftmaxCrossEntropyLoss()。

```
class Net(model):
    def __init__(self, x_train, y_train, x_test, y_test, labels_num, ctx):
        super().__init__(x_train, y_train, x_test, y_test, labels_num, ctx)
        #softmax交叉熵损失函数
        self.loss_func=gloss.SoftmaxCrossEntropyLoss()
        self.net=None
```

3.2 网络结构

本次实验搭建经典的 LeNet 网络，而 LeNet 一开始就是通过梯度下降训练卷积神经网络完成手写数字识别任务。

LeNet 分为卷积层和全连接层块两个部分，卷积层块基本单位是卷积层后接最大池化层。卷积层都使用 5×5 的窗口，并在输出上使用 sigmoid 激活函数，第一个卷积层输出通道数为 6，第二个卷积层输出通道数则增加到 16，而卷积层的两个最大池化层的窗口形状均为 2×2，且步幅为 2。全连接层块含 3 个全连接层，它们的输出个数分别是 120、84 和 10，其中 10 为输出的类别数。

```python
def LeNet(self):
    '''
    实现经典的LeNet
    第一层，卷积层，输出通道：6，卷积核：5*5，激活函数：sigmoid；
    第二层，池化层，窗口形状：2*2，步幅：2；
    第三层，卷积层，输出通道：16，卷积核：5*5，激活函数：sigmoid；
    第四层，池化层，窗口形状：2*2，步幅：2；
    第五层，全连接层，输出个数：120，激活函数：sigmoid；
    第六层，全连接层，输出个数：84，激活函数：sigmoid；
    第七层，输出层，输出个数：10(标签数)；
    '''
    net=nn.Sequential()
    net.add(nn.Conv2D(channels=6,kernel_size=5,activation='sigmoid'),\
            nn.MaxPool2D(pool_size=2,strides=2),\
            nn.Conv2D(channels=16,kernel_size=5,activation='sigmoid'),\
            nn.MaxPool2D(pool_size=2,strides=2),\
            nn.Dense(120,activation='sigmoid'),\
            nn.Dense(84,activation='sigmoid'),\
            nn.Dense(self.labels_num))
    print('init net...')
    print('--------------------')
    print(net)
    print('--------------------')
    net.collect_params().initialize(force_reinit=True, ctx=self.ctx)
    net.initialize(force_reinit=True,ctx=self.ctx,init=init.Xavier())
    return net
```

```
--------------------
Sequential(
  (0): Conv2D(None -> 6, kernel_size=(5, 5), stride=(1, 1), Activation(sigmoid))
  (1): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False, global_pool=False, pool_type=max, layout=NCHW)
  (2): Conv2D(None -> 16, kernel_size=(5, 5), stride=(1, 1), Activation(sigmoid))
  (3): MaxPool2D(size=(2, 2), stride=(2, 2), padding=(0, 0), ceil_mode=False, global_pool=False, pool_type=max, layout=NCHW)
  (4): Dense(None -> 120, Activation(sigmoid))
  (5): Dense(None -> 84, Activation(sigmoid))
  (6): Dense(None -> 10, linear)
)
```

4 实验分析

4.1 数据集介绍

MNIST 来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST). 训练集 (training set) 由来自 250 个不同人手写的数字构成, 其中 50% 是高中学生, 50% 来自人口普查局 (the Census Bureau) 的工作人员. 测试集 (test set) 也是同样比例的手写数字数据。MNIST 包含以下四个部分：

Training set images: train-images-idx3-ubyte.gz (9.9MB, 解压后 47MB, 包含 60,000 个样本)

Training set labels: train-labels-idx1-ubyte.gz (29KB, 解压后 60KB, 包含 60,000 个标签)

Test set images: t10k-images-idx3-ubyte.gz (1.6MB, 解压后 7.8MB, 包含 10,000 个样本)

Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 解压后 10KB, 包含 10,000 个标签)

MNIST 数据集中的每张图片由 28×28 个像素点构成, 每个像素点用一个灰度值表示。图片数据集中，前 12 个字节分别表示幻数、图片数、行数以及列数，标签数据集中，前 4 个字节分别表示幻数和

样本数。
具体处理数据集代码如下图：

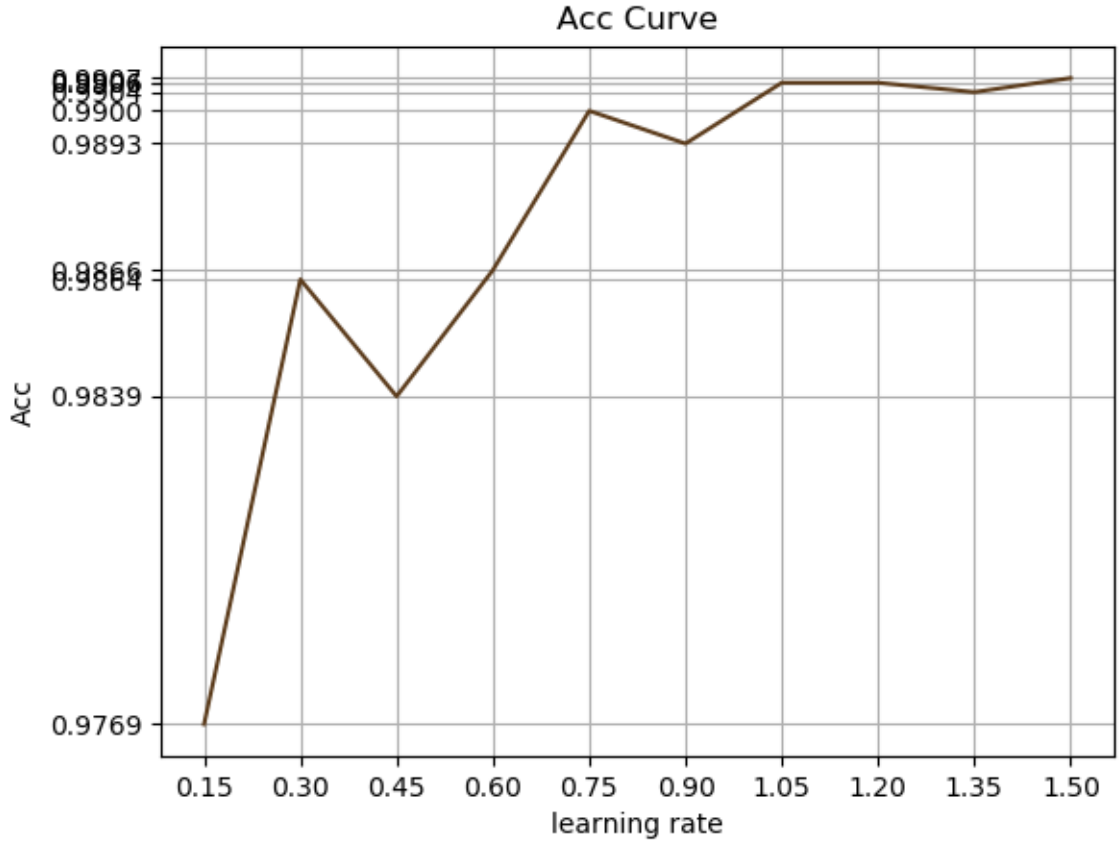
```python
def load_mnist():
    '''
    加载本地MNIST数据集
    '''
    PATH=path.abspath(path.join(path.dirname("__file__"),'MNIST'))
    TRAIN_IMAGES=os.path.join(PATH,'train-images-idx3-ubyte.gz')
    TRAIN_LABELS=os.path.join(PATH,'train-labels-idx1-ubyte.gz')
    TEST_IMAGES=os.path.join(PATH,'t10k-images-idx3-ubyte.gz')
    TEST_LABELS=os.path.join(PATH,'t10k-labels-idx1-ubyte.gz')

    x_train=_load_img(TRAIN_IMAGES)
    y_train=_load_label(TRAIN_LABELS)
    x_test=_load_img(TEST_IMAGES)
    y_test=_load_label(TEST_LABELS)

    return x_train,y_train,x_test,y_test
```

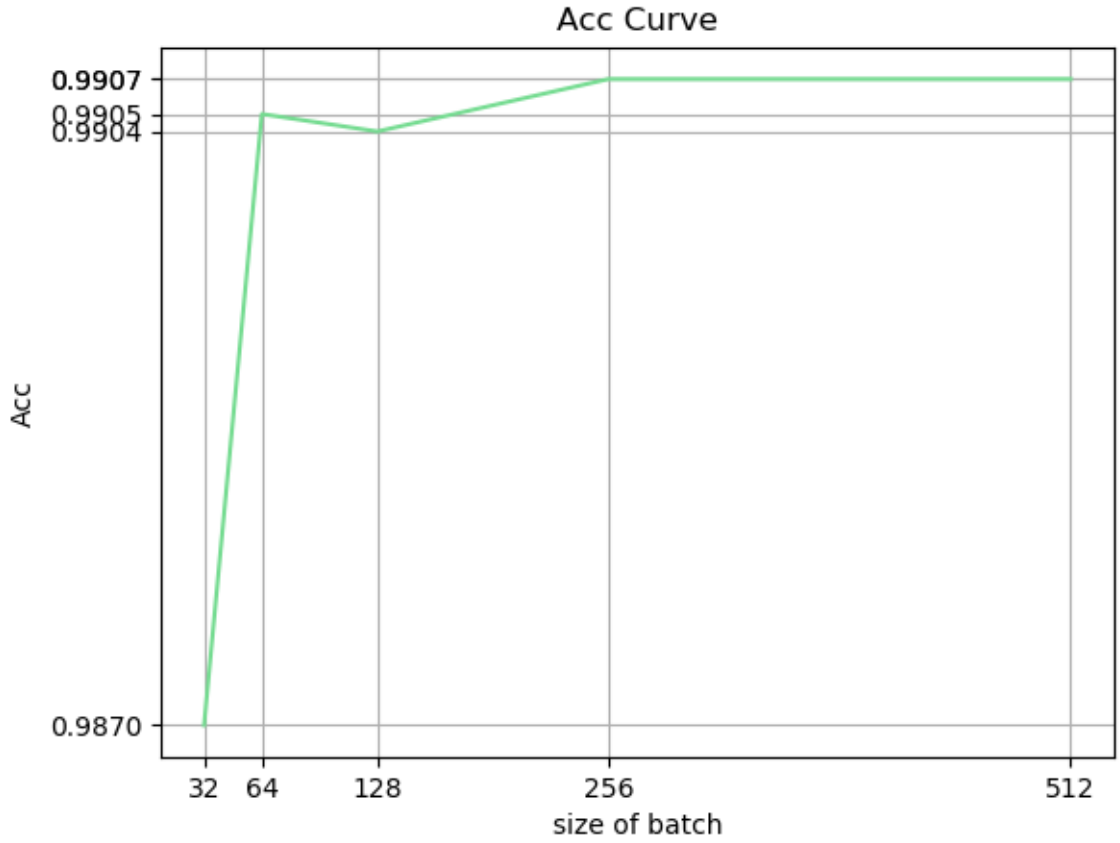
```python
def _load_img(file_path):
    '''
    加载图片
    '''
    with gzip.open(file_path,'rb') as bytestream:
        img_data=np.frombuffer(bytestream.read(),np.uint8,offset=16).astype(np.float32)
    #normalize : 将图像的像素值正规化为0.0~1.0
    img_data/=255.0
    #index * channels * columns * rows
    img_data=img_data.reshape(-1,1,28,28)
    return img_data
```

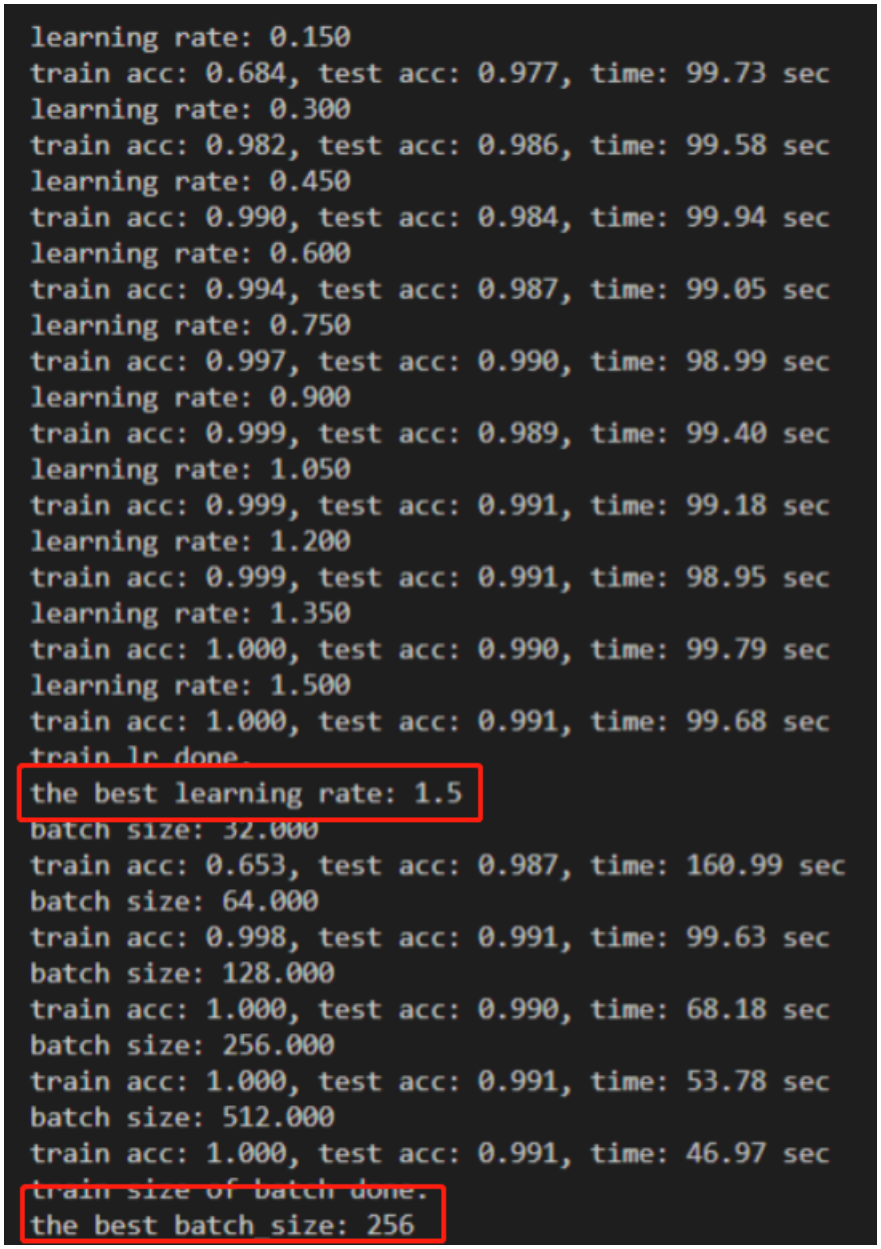
```python
def _load_label(file_path):
    '''
    加载标签
    '''
    with gzip.open(file_path,'rb') as bytestream:
        label_data=np.frombuffer(bytestream.read(),np.uint8,offset=8)
    return label_data
```

4.2 实验结果

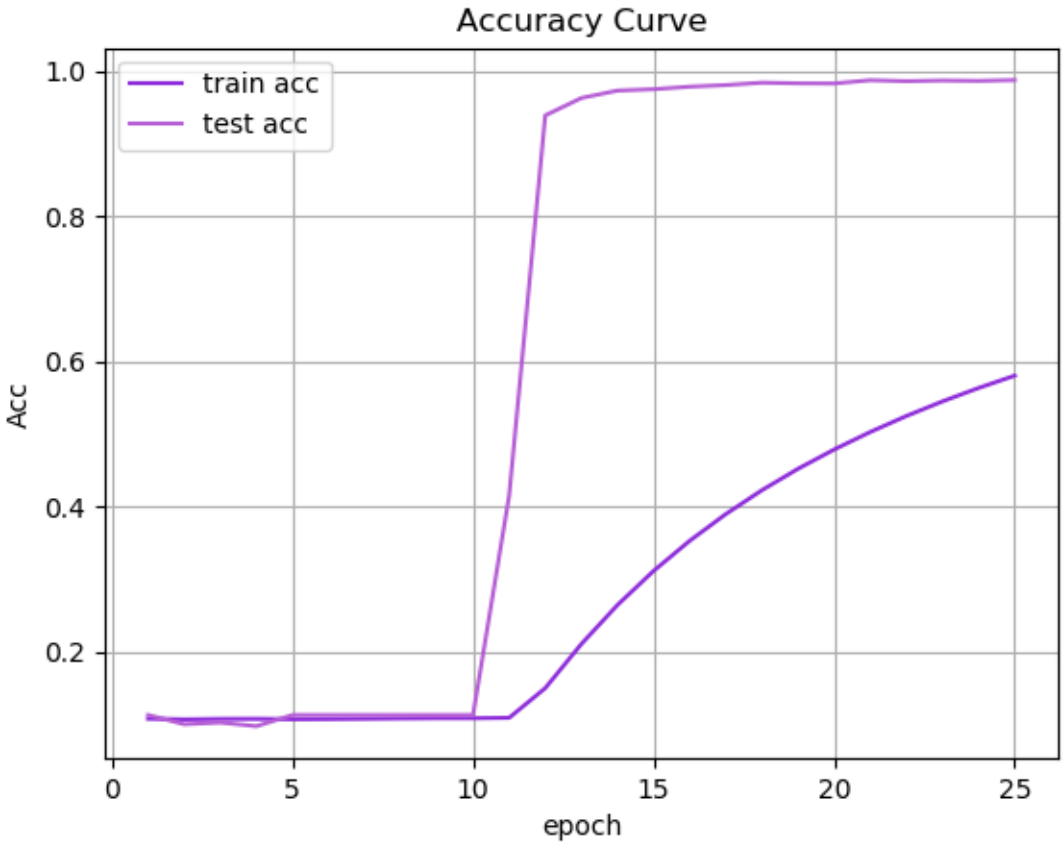
a) 由于笔记本性能较差以及 LeNet 对 MNIST 数据集进行分类效果非常好，所以只进行了简单的调参（学习率、小批量大小）。


Acc Curve
0.9900
0.9893
0.9839
0.9769
Acc
0.15
0.30
0.45
0.60
0.75
0.90
1.05
1.20
1.35
1.50
learning rate


Acc Curve
0.9907
0.9905
0.9904
0.9870
Acc
32
64
128
256
512
size of batch

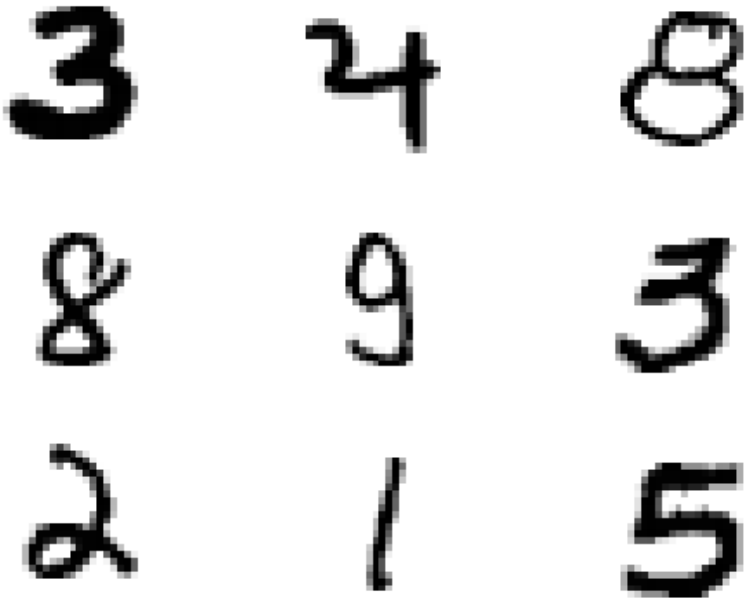


b) 确定最优超参数之后，进行最后的模型训练，可以看到到第 17 轮迭代后，测试的准确率达到了 98%，达到实验要求。




```
training on gpu(0)
epoch 1, loss 2.3183, train acc 0.107, test acc 0.114, time 5.4 sec
epoch 2, loss 2.3033, train acc 0.108, test acc 0.101, time 5.3 sec
epoch 3, loss 2.3028, train acc 0.107, test acc 0.103, time 5.4 sec
epoch 4, loss 2.3029, train acc 0.108, test acc 0.098, time 5.4 sec
epoch 5, loss 2.3029, train acc 0.108, test acc 0.114, time 5.4 sec
epoch 6, loss 2.3027, train acc 0.109, test acc 0.114, time 5.6 sec
epoch 7, loss 2.3025, train acc 0.111, test acc 0.114, time 5.4 sec
epoch 8, loss 2.3026, train acc 0.110, test acc 0.114, time 5.4 sec
epoch 9, loss 2.3024, train acc 0.110, test acc 0.114, time 5.4 sec
epoch 10, loss 2.3024, train acc 0.109, test acc 0.114, time 5.5 sec
epoch 11, loss 2.2776, train acc 0.140, test acc 0.415, time 5.5 sec
epoch 12, loss 0.6847, train acc 0.775, test acc 0.939, time 5.6 sec
epoch 13, loss 0.1771, train acc 0.946, test acc 0.963, time 5.5 sec
epoch 14, loss 0.1206, train acc 0.963, test acc 0.973, time 5.5 sec
epoch 15, loss 0.0934, train acc 0.971, test acc 0.975, time 5.5 sec
epoch 16, loss 0.0782, train acc 0.976, test acc 0.978, time 5.5 sec
epoch 17, loss 0.0691, train acc 0.979, test acc 0.980, time 5.6 sec
epoch 18, loss 0.0608, train acc 0.982, test acc 0.984, time 5.4 sec
epoch 19, loss 0.0564, train acc 0.983, test acc 0.983, time 5.5 sec
epoch 20, loss 0.0517, train acc 0.984, test acc 0.983, time 5.5 sec
epoch 21, loss 0.0487, train acc 0.985, test acc 0.987, time 5.5 sec
epoch 22, loss 0.0450, train acc 0.986, test acc 0.986, time 5.6 sec
epoch 23, loss 0.0423, train acc 0.987, test acc 0.987, time 5.5 sec
epoch 24, loss 0.0399, train acc 0.988, test acc 0.986, time 5.4 sec
epoch 25, loss 0.0364, train acc 0.989, test acc 0.987, time 5.5 sec
```

c) 从测试集中随机选取九个样本进行人工对比验证，检验结果。



```
test result.
true labels:
[[3. 4. 8.]
 [8. 9. 3.]
 [2. 1. 5.]]
pred labels:
[[3. 4. 8.]
 [8. 9. 3.]
 [2. 1. 5.]]
```

5 总结与分析

经过理论学习，总感觉自己对深度学习有种云里雾里的感觉。第一次实现深度学习项目没出意外遇到了很多困难，课程中不经意跳过的每一个细节都在真正实现中成为了绊脚石。个人认为实验中的难点在于数据的导入处理，充分理解数据集，能将数据规整的导入程序，以相应的变量的形式表示的时候，就已成功大半了。