**实验报告**

专业：计算机科学与技术

姓名：贺情怡

学号：3180105438

日期：2020/1/18

课程名称： 计算机视觉 指导老师： 宋明黎 成绩：

实验名称： HW#5: 学习CNN

**一、实验目的和要求**

利用CNN进行手写数字识别

1. 框架：TenserFlow（PyTorch也行）
2. 数据集：The Mnist Database of handwritten digits
3. 网络结构：LeNet-5;
4. 具体任务：
   1. 利用上述数据集/网络结构/框架实现手写数字识别，可以参考代码
   2. 参考链接：
      1. <https://www.tensorflow.org/versions/r0.12/tutorials/mnist/pros/index.html>
      2. <http://wiki.jikexueyuan.com/project/tensorflow-zh/tutorials/mnist_beginners.html>
      3. <http://blog.csdn.net/kkk584520/article/details/51477537>

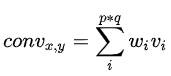
**二、实验内容和原理**

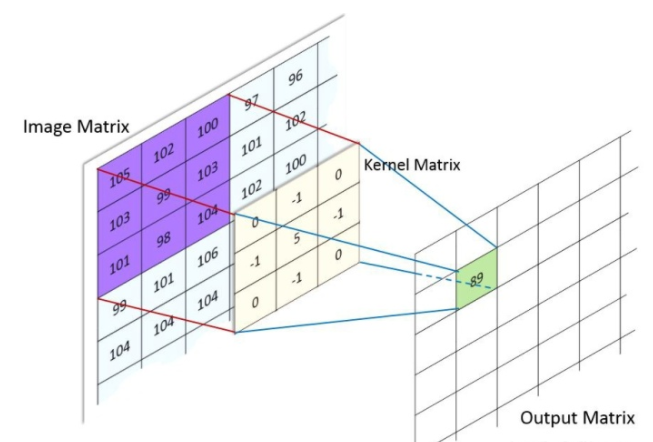
**CNN**

卷积神经网络（CNN）是一种特殊的多层神经网络。像几乎所有其他的神经网络一样，它也是用反向传播算法来训练的，只是用特殊结构来以更少的预处理识别像素图中的pattern。而且它对一些简单的几何变换有一定的鲁棒性。

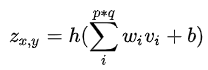
基础的CNN由卷积(convolution), 激活(activation), 和池化(pooling)三种结构组成。当处理图像分类任务时，我们会把CNN输出的特征空间作为全连接层或全连接神经网络(fully connected neural network, FCN)的输入，用全连接层来完成从输入图像到标签集的映射，即分类。当然，整个过程最重要的工作就是如何通过训练数据迭代调整网络权重，也就是后向传播算法。目前主流的卷积神经网络(CNNs)，比如VGG, ResNet都是由简单的CNN调整，组合而来。

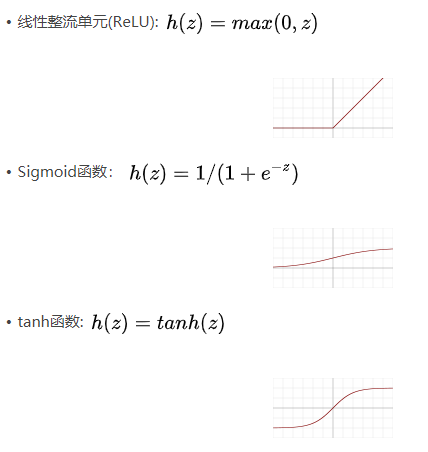
**卷积**

我们假设单一通道输入图像的空间坐标为，卷积核大小是，kernel权重为,图像亮度值是 ，卷积过程就是kernel 所有权重与其在输入图像上对应元素亮度之和，即



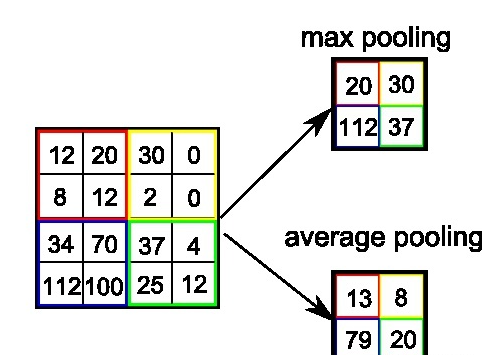
**激活**

卷积之后，通常会加入偏置(bias), 并引入非线性激活函数(activation function)，这里定义bias为b，activation function 是  ，经过激活函数后，得到的结果是：。bias不与元素位置相关，只与层有关。



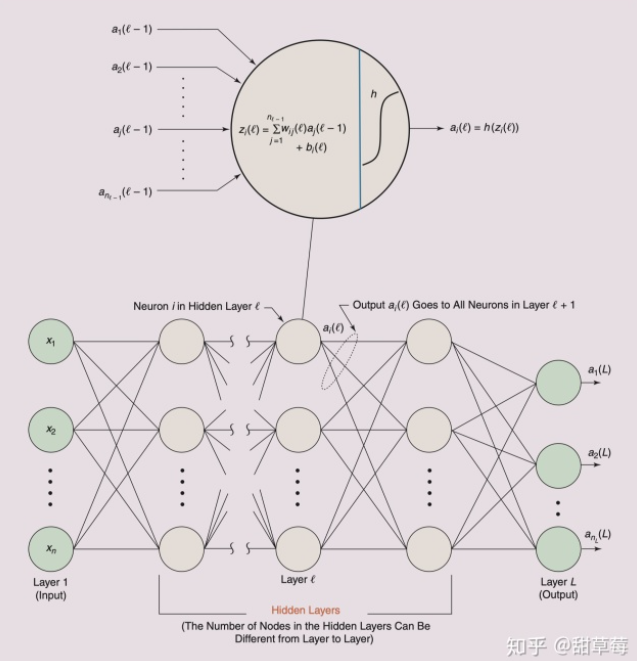
**池化**

池化(pooling），是一种降采样操作(subsampling)，主要目标是降低feature maps的特征空间，或者可以认为是降低feature maps的分辨率。因为feature map参数太多，而图像细节不利于高层特征的抽取。

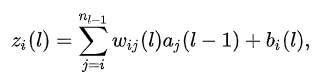


Pooling操作会降低参数，降低feature maps的分辨率，但是这种暴力降低在计算力足够的情况下是不是必须的，并不确定。目前一些大的CNNs网络只是偶尔使用pooling。

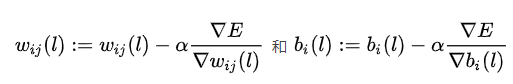
**全连接网络**



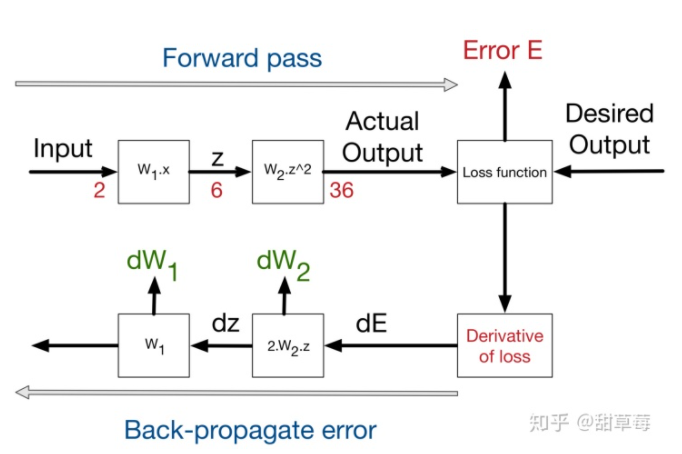
不同于CNN的滑动卷积，全连接网络每一层的所有单元与上一层完全连接。对于第l 层的第i个神经元，它的输出计算方式是,



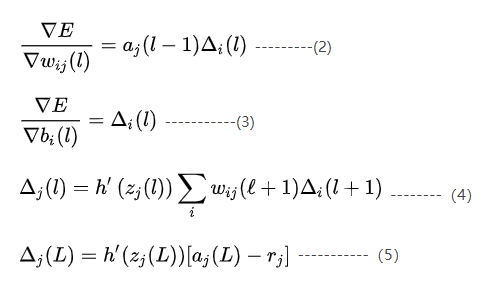
**目标函数和训练方法**

假设得到的误差为E，则在更新时有有

其中是learning rate，相当于每次学习的步长。然后通过BP来一层层往前调整参数进行学习。

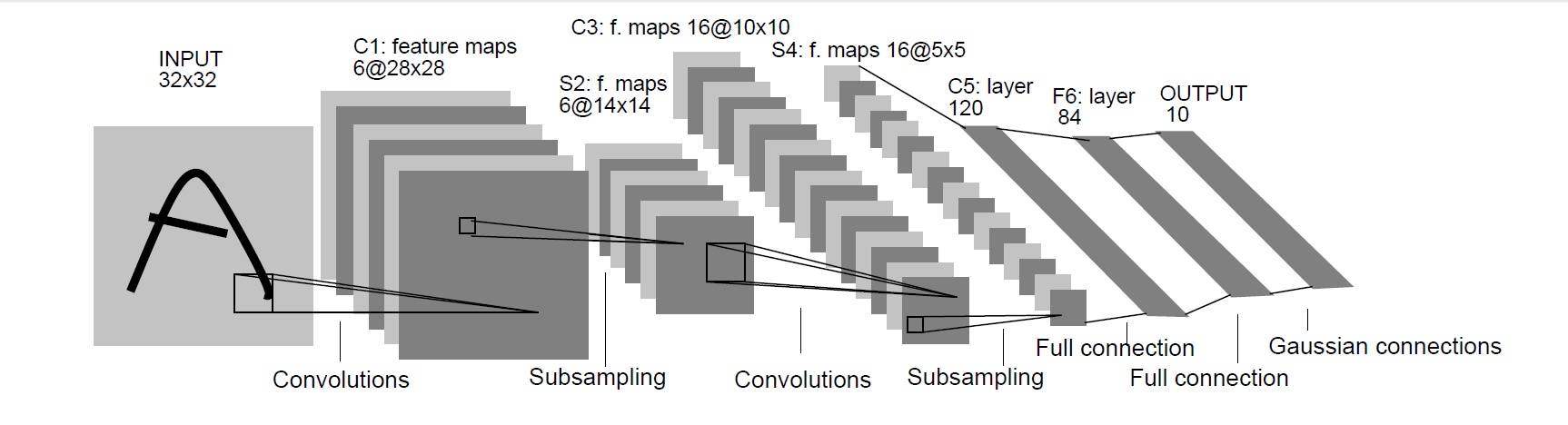


BP算法通过以下四个公式更新：



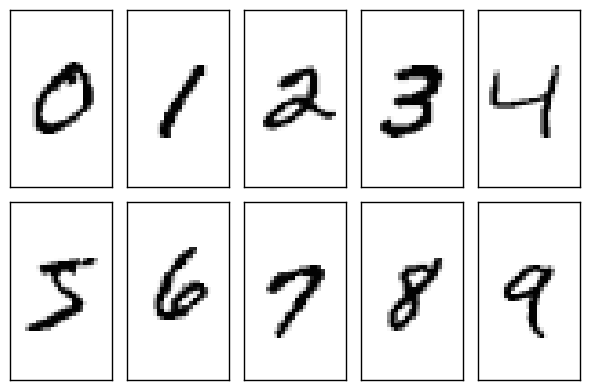
**LeNet-5**

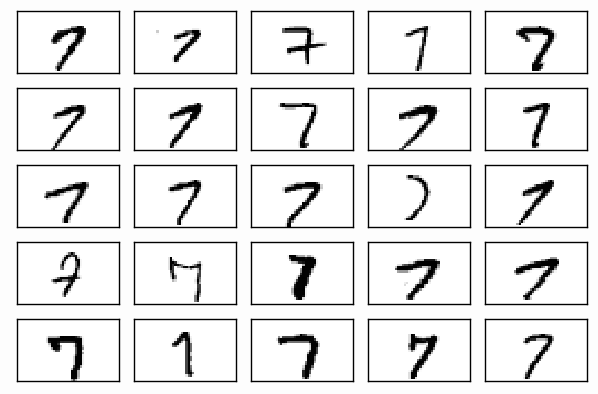
是一种用来进行手写字母识别的网络，LeNet-5中主要有2个卷积层、2个下抽样层（池化层）、3个全连接层，如图。



**MNIST**

MNIST 数据集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。





**三、实验步骤与分析**

实验中试了各种optimizer和loss函数，发现效果差距很大。

然后尝试了对自己用画图制作的数据集进行识别，效果也不错。

**网络结构**

|  |
| --- |
| class Net(nn.Module):  def \_\_init\_\_(self):  super().\_\_init\_\_()    self.conv1 = nn.Sequential(  nn.Conv2d(1, 6, 5, 1, 2),  nn.ReLU(),  nn.MaxPool2d(kernel\_size=2, stride=2),  )  self.conv2 = nn.Sequential(  nn.Conv2d(6, 16, 5),  nn.ReLU(),  nn.MaxPool2d(2, 2)  )  self.fc1 = nn.Sequential(  nn.Linear(16 \* 5 \* 5, 120),  nn.ReLU()  )  self.fc2 = nn.Sequential(  nn.Linear(120, 84),  nn.ReLU()  )  self.fc3 = nn.Linear(84, 10)    def forward(self, x):  x = self.conv1(x)  x = self.conv2(x)  x = x.view(x.size()[0], -1)  x = self.fc1(x)  x = self.fc2(x)  x = self.fc3(x)  return x |

**训练函数**

|  |
| --- |
| def train(model, device, train\_loader, optimizer, epoch):  model.train()  for batch\_idx, (data, target) in enumerate(train\_loader):  data, target = data.to(device), target.to(device)  data, target = Variable(data), Variable(target)    optimizer.zero\_grad()  output = model(data)  loss = F.cross\_entropy(output, target)  # loss = F.nll\_loss(output, target)  loss.backward()  optimizer.step()  if(batch\_idx+1)%30 ==0:  print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(  epoch, batch\_idx \* len(data), len(train\_loader.dataset),  100. \* batch\_idx / len(train\_loader), loss.item())) |

**训练函数**

|  |
| --- |
| def test(epoch\_num, model, device, test\_loader):  model.eval()  test\_loss = 0  correct = 0  with torch.no\_grad():  for i, (data, target) in enumerate(test\_loader):  data, target = data.to(device), target.to(device)  output = model(data)  test\_loss += F.cross\_entropy(output, target, reduction='sum').item()  pred = output.max(1, keepdim=True)[1]  correct += pred.eq(target.view\_as(pred)).sum().item()  test\_loss /= len(test\_loader.dataset)  print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(  test\_loss, correct, len(test\_loader.dataset),  100. \* correct / len(test\_loader.dataset))) |

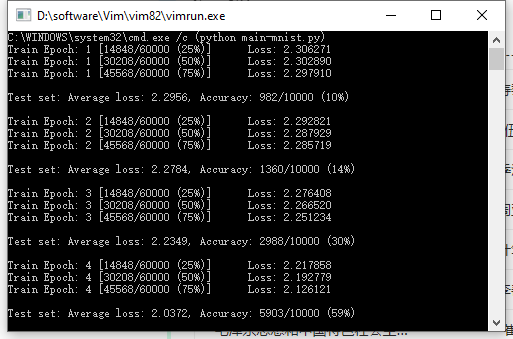
**测试程序**

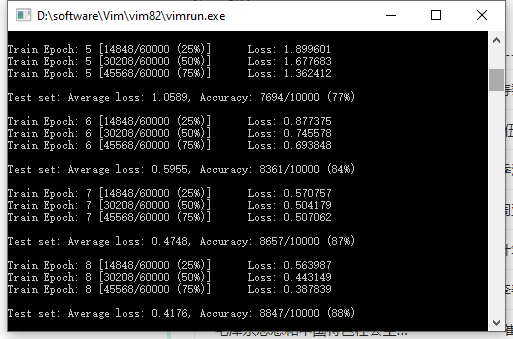
|  |
| --- |
| img = cv2.imread(PATH)  trans = transforms.Compose(  [  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,))  ])  img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  img = trans(img)  img = img.to(device)  img = img.unsqueeze(0)  output = model(img)  prob = F.softmax(output, dim=1)  prob = Variable(prob)  prob = prob.cpu().numpy()  prob = prob[0]  print(prob)  pred\_out = np.argmax(prob)  plt.title('pred is {}'.format(pred\_out))  img = cv2.imread(PATH)  plt.imshow(img)  plt.show() |

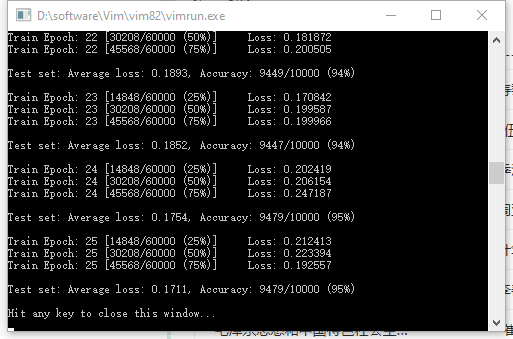
**四、实验结果**

学习率都是0.01，只是换了Loss和Optimizer，效果差异显著x

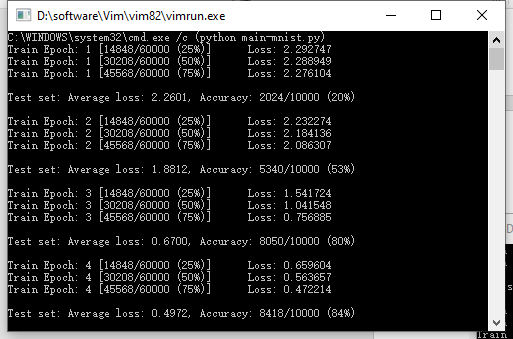
**Loss：直接返回X，Optimizer：Adadelta**

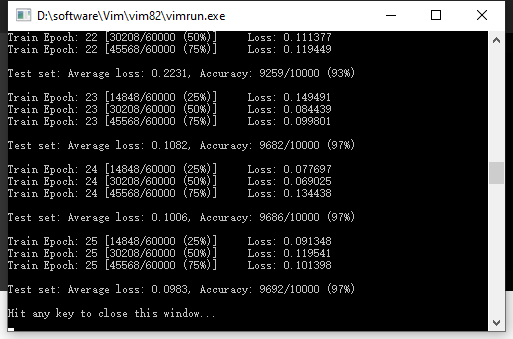
****



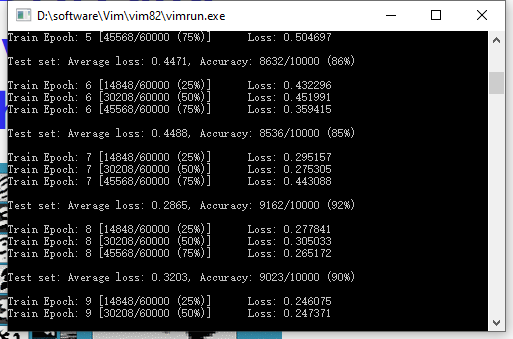


**Loss：直接返回X，optimizer：SGD**



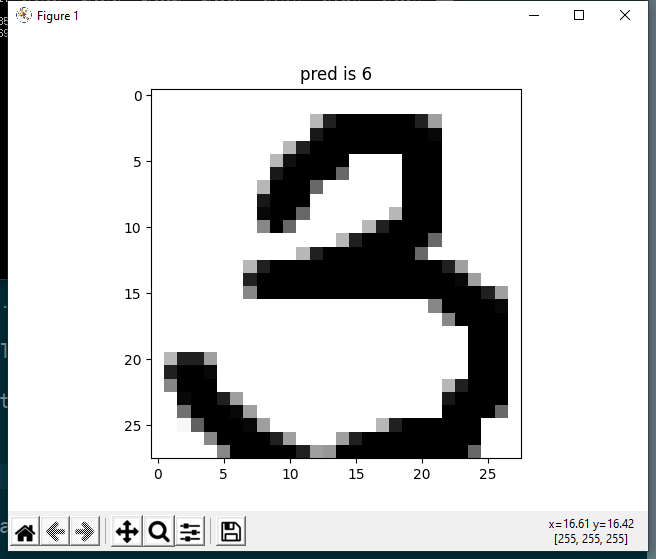


**Loss：log\_softmax，Optimizer：SGD**

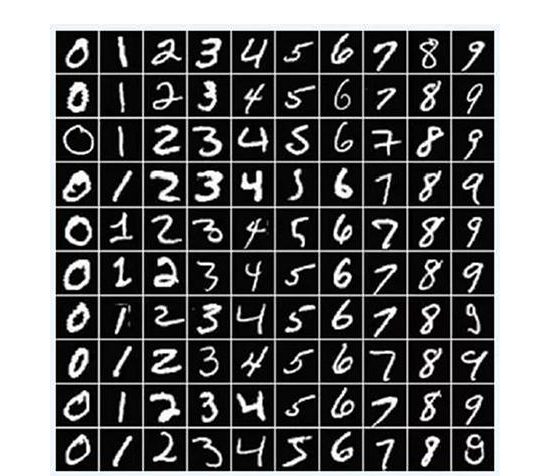


**用画图自制数据测试**

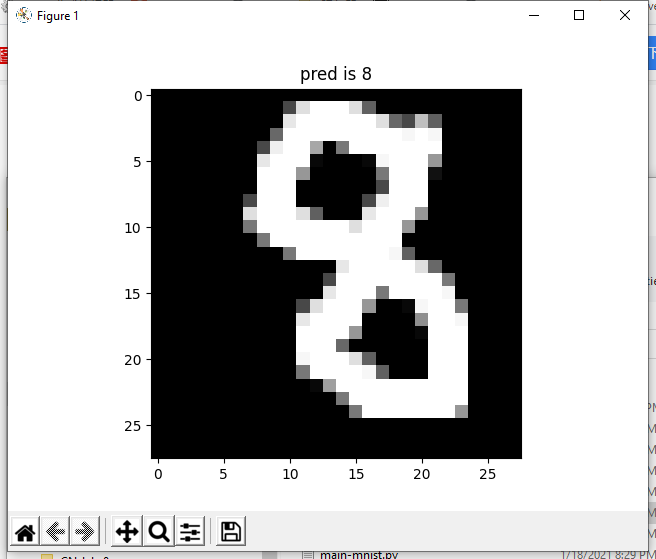
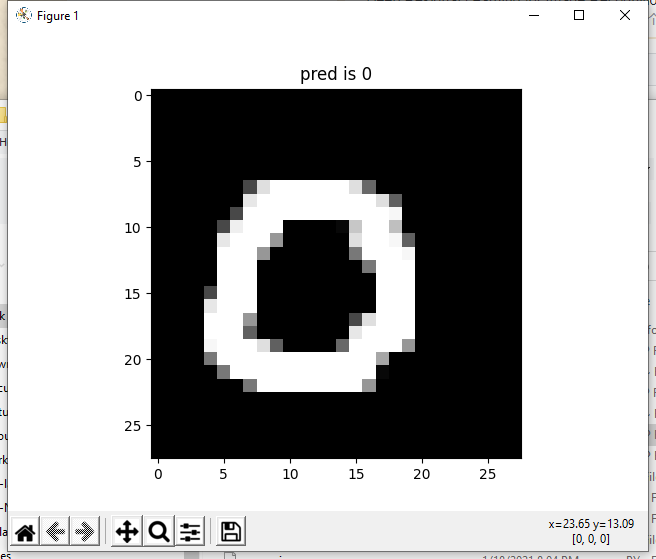
一开始使用白底黑字的图像，发现一直没有成功（虽然这个3真的挺像6的x）

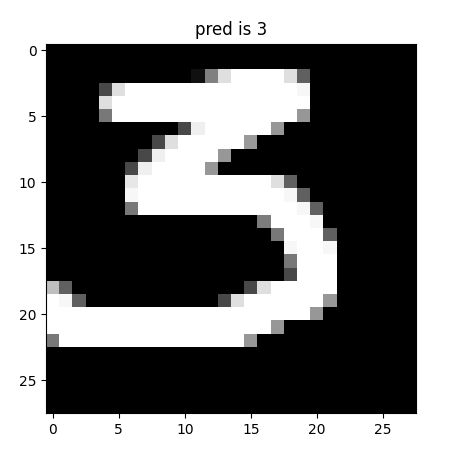
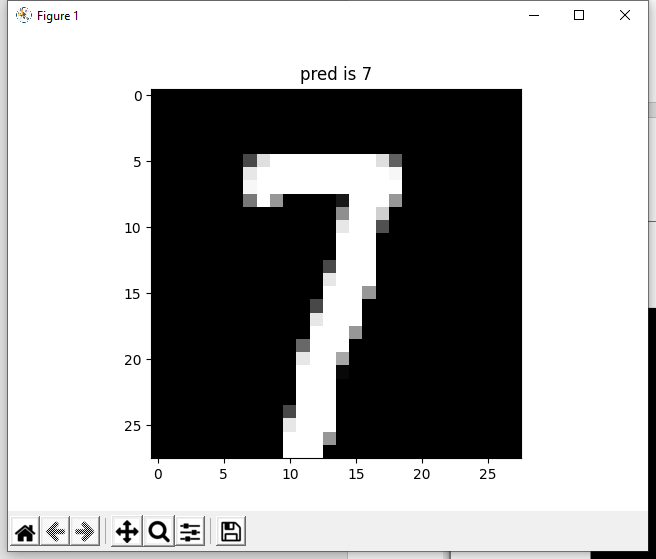


于是怀疑可能output的顺序并不是按照a[i]表示这是数字i对应的值这样的……但实在没有找到对应的资料，在找图像印证猜想时，发现图像都是黑底白字：



想起了当时学图像处理时遇到的坑，于是干脆试了试，发现好像真的是这个原因：





**五、心得体会**

以前有抄过一次MNIST，这次重新学习理解了一次，感觉真的好炼丹啊（改一个Loss函数或者Optimizer，效果差距就会很大）

学会了把torch里的张量可视化，懂了pytorch里的数据大概存放格式，寒假应该能会自制数据集并可视化了，学到了很多（炼丹真是复习考试的时候一个不错的放松方式x）

被白底黑字和黑底白字坑了闷闷不乐x