认知科学与类脑计算

实验报告七

日期：2019/5/31

班级：16 人工智能

姓名：贾乘兴

学号：201600301304

一.需求分析：

卷积神经网络（Convolutional Neural Networks, CNN）是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表算法之一。由于卷积神经网络能够进行平移不变分类，因此也被称为“平移不变人工神经网络”。对卷积神经网络的研究始于二十世纪 80 至 90 年代，时间延迟网络和 LeNet-5 是最早出现的卷积神经网络；在二十一世纪后，随着深度学习理论的提出和数值计算设备的改进，卷积神经网络得到了快速发展，并被大量应用于计算机视觉、自然语言处理等领域。卷积神经网络仿造生物的视知觉机制构建，可以进行监督学习和非监督学习，其隐含层内的卷积核参数共享和层间连接的稀疏性使得卷积神经网络能够以较小的计算量对格点化特征，例如像素和音频进行学习、有稳定的效果且对数据没有额外的特征工程要求。本次实验目的在于加深对卷积神经网络模型的理解，能够使用卷积神经网络模型解决简单问题。

二.概要设计：

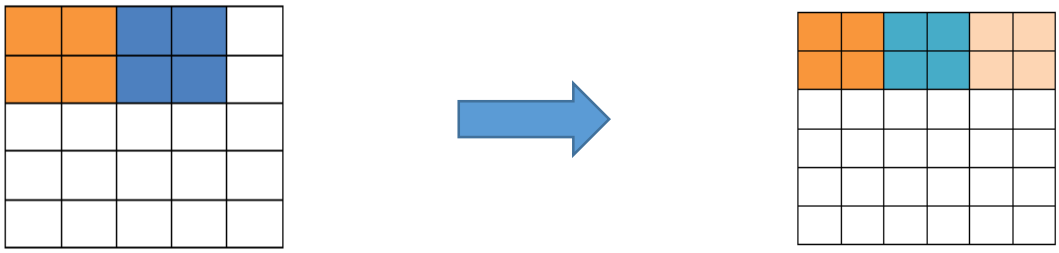
下载 MNIST 数据集，构建卷积神经网络，使用 MNIST 数据集中的训练集训练网络，使用测试集测试训练好的网络，测试准确率可达到 98%以上。

三.详细设计：

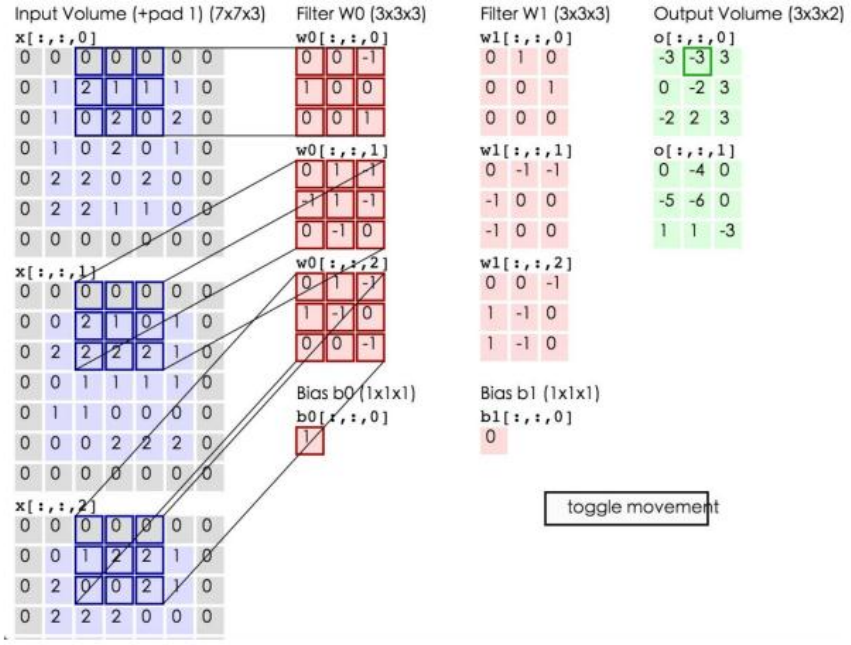
卷积神经网络的层级结构一般为：数据输入层、卷积计算层、激励层（最常用的为 ReLU 激活函数）、池化层（平均池化和最大池化）、全连接层（不必须）、输出层。

数据输入层。该层要做的处理是对原始图像数据进行预处理，其中主要包括归一化操作，即将图像数据处理成均值为 0，方差为 1 的数据等。

卷积计算层。在卷积层，有两个关键操作：局部关联，每个神经元看做一个滤波器；窗口滑动，滤波器对局部数据进行计算。先介绍卷积层遇到的几个名词：步长(stride)，即窗口一次滑动的长度；填充值(padding)，一般为 0 值填充。以下图为例子，比如有一个 5*5 的图片（一个格子一个像素），我们滑动窗口取 2*2，步长取 2，那么我们发现还剩下 1 个像素没法滑完，就可以在原先的矩阵边缘加一层填充值，使其变成 6*6 的矩阵，那么窗口就可以刚好把所有像素遍历完。这就是填充值的作用。



卷积的计算。下图的蓝色矩阵就是输入的图像，粉色矩阵就是卷积层的神经元，这里表示了有两个神经元（w_0,w_1）。绿色矩阵就是经过卷积运算后的输出矩阵，这里的步长设置为 2。

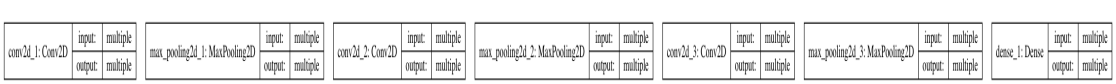


激励层。把卷积层输出结果做非线性映射，最常用的为 ReLU 激活函数。池化层夹在连续的卷积层中间，用于压缩数据和参数的量，减小过拟合。简而言之，如果输入是图像的话，那么池化层的最主要作用就是压缩图像。

四.调试分析：

神经网络结构如下

卷积层 1→池化层→卷积层 2→池化层→卷积层 3→池化层→全联接层



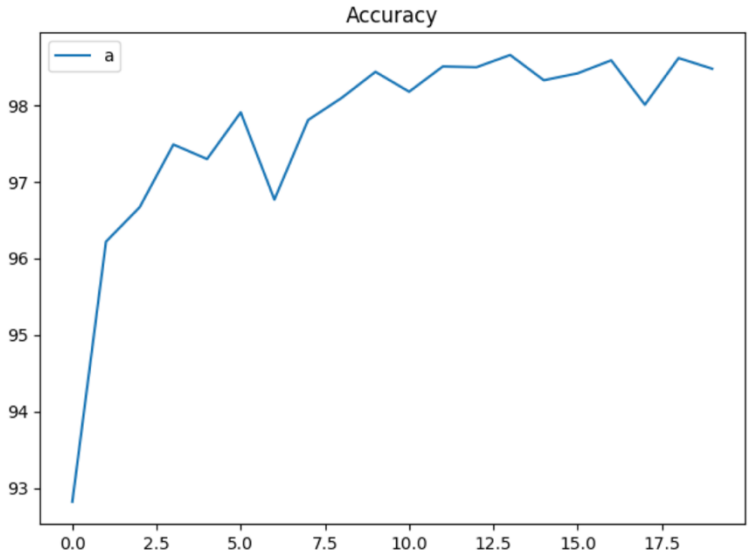

```
self.conv1 = nn.Conv2d(in_channels=1, out_channels=10, kernel_size=5)
self.conv2 = nn.Conv2d(10, 20, 5)
self.conv3 = nn.Conv2d(20, 40, 3)
self.mp = nn.MaxPool2d(2)
self.fc = nn.Linear(40, 10)# (in_features, out_features)
```

我们使用三层全联接层进行对比，对比的全联接网络参数为 784-1024-128-10

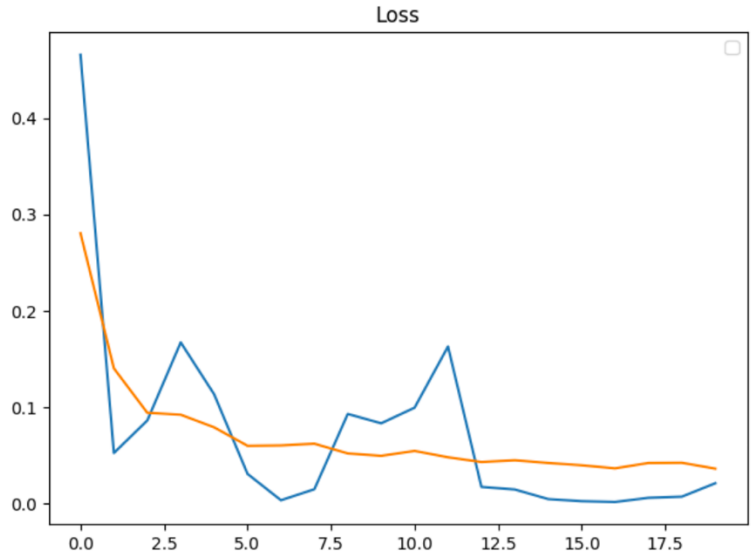
五.测试结果：

cnn 的结果

Accuracy 变化曲线如下，可见准确性较高，很快达到了 98%以上

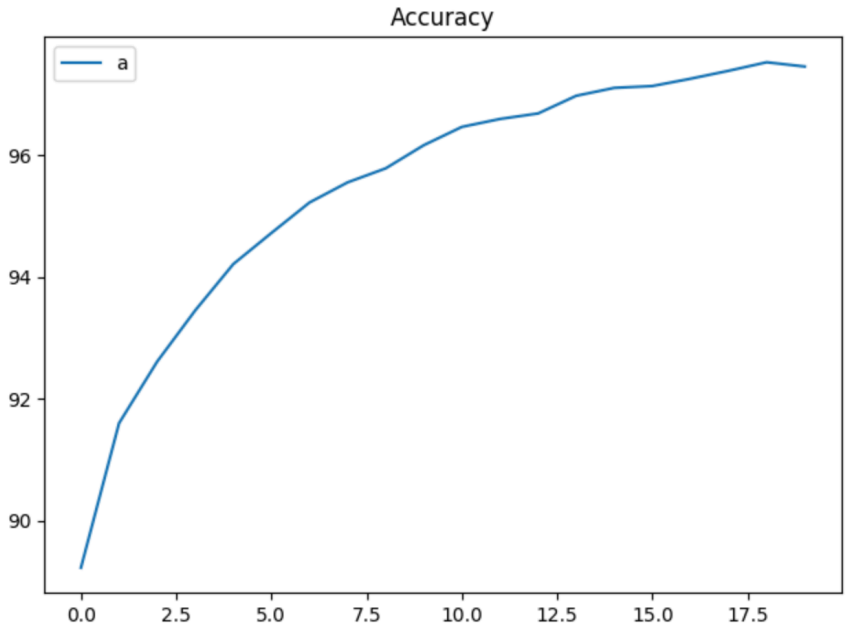


loss 变化曲线如下（蓝色为测试集，黄色为训练集）

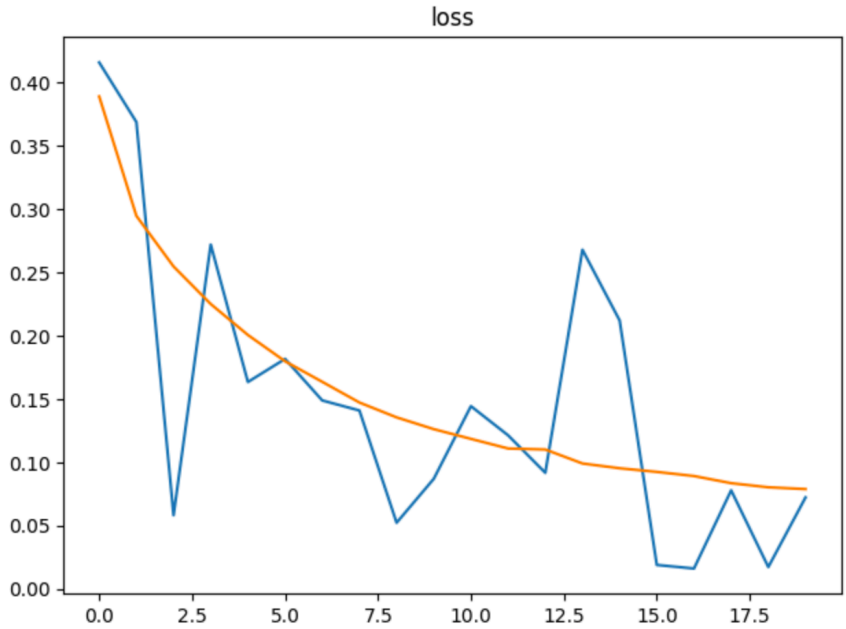


普通的全联接层结果

准确率最高到达 97%以上



loss 曲线



可见使用了 cnn 后收敛速度和准确率都提升了

我们在使用了 batch-normalization 后，准确率达到了 99.2%

六.附录：

附录代码如下 文件为 cnn.py

cnn.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable

# Training settings
batch_size = 64

# MNIST Dataset
# MNIST 数据集已经集成在 pytorch datasets 中，可以直接调用
train_dataset = datasets.MNIST(root='./data/',
                               train=True,
                               transform=transforms.ToTensor(),
                               download=True)

test_dataset = datasets.MNIST(root='./data/',
                              train=False,
                              transform=transforms.ToTensor())

# Data Loader (Input Pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 输入 1 通道，输出 10 通道，kernel 5*5
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, 5)
        self.conv3 = nn.Conv2d(20, 40, 3)

        self.mp = nn.MaxPool2d(2)
        # fully connect
        self.fc = nn.Linear(40, 10)# (in_features, out_features)
```

```
    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = F.relu(self.mp(self.conv3(x)))

        x = x.view(in_size, -1)
        x = self.fc(x)
        return F.log_softmax(x)  #64*10


class Net0(nn.Module):
    def __init__(self):
        super(Net0, self).__init__()

        self.fc1 = nn.Linear(784, 1024)
        self.fc2 = nn.Linear(1024, 128)
        self.fc = nn.Linear(128, 10)

    def forward(self, x):

        in_size = x.size(0)
        x = x.view(in_size, -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc(x)
        return F.log_softmax(x)  #64*10

model = Net0()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

def train(epoch):
    for batch_idx, (data, target) in enumerate(train_loader):#batch_idx 是
enumerate（）函数自带的索引，从 0 开始
        # data.size():[64, 1, 28, 28]
        # target.size():[64]
        output = model(data)
        #output:64*10
        loss = F.nll_loss(output, target)
        optimizer.zero_grad()   # 所有参数的梯度清零
        loss.backward()        #即反向传播求梯度
        optimizer.step()       #调用 optimizer 进行梯度下降更新参数
    print('loss train {}'.format(loss.data))
    tloss.append(loss.data)
```

```
def test():
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        # sum up batch loss
        test_loss += F.nll_loss(output, target)
        # get the index of the max log-probability
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    test_loss /= len(test_loader)
    print('loss test {}'.format(test_loss.data))
    print('acc test {}'.format(correct))


    eloss.append(test_loss.data)
    eacc.append(correct/10000)


tloss = []
eloss = []
eacc = []

for epoch in range(20):
    train(epoch)
    test()

from matplotlib import pyplot as plt

plt.plot(tloss)
plt.plot(eloss)
plt.title('loss')
plt.show()

plt.plot(eacc)
plt.title('accuracy')
plt.show()
```