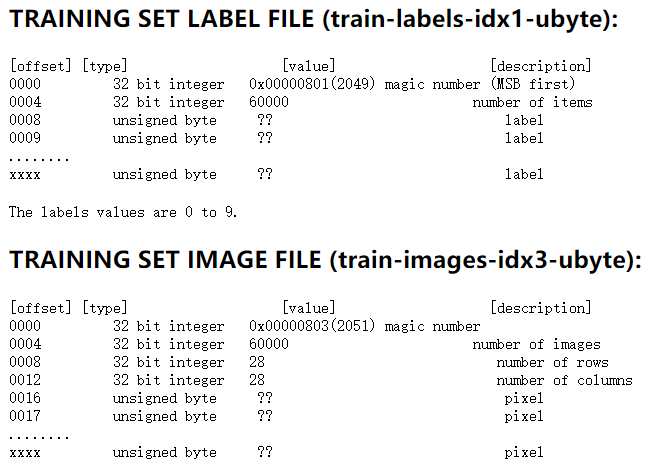
网络的构建

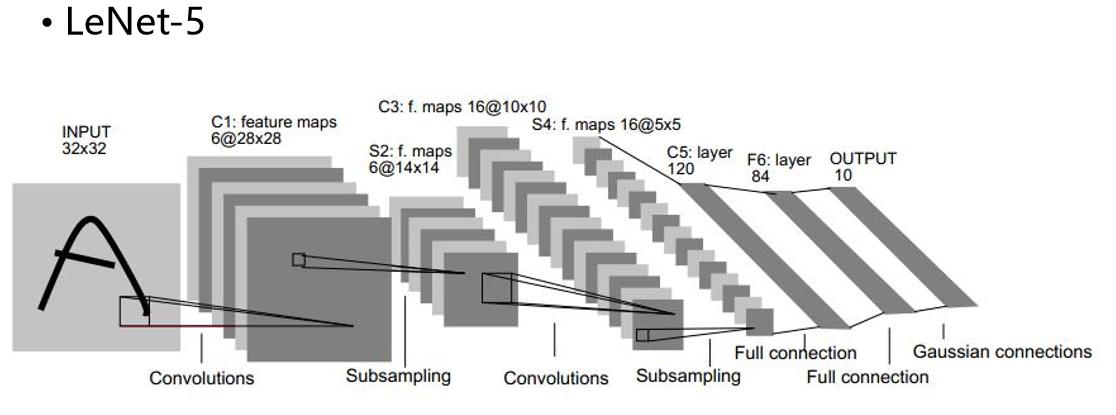
由于所给手写识别数据有训练集与测试集，所以我的思路是利用训练集来对网络模型进行训练，然后将网络模型的参数保存下来。再创建一个相同结构的网络模型，直接加载参数，然后对于测试集进行验证，这样就将模型的训练与测试分隔开来，如果切换不同的测试集也不在需要重新训练网络。

首先需要读入数据，训练集数据的格式如下



在label文件中，前8个字节为两个整数，一个为标识，另一个为数据个数。剩下的数据都为8位无符号整数，代表每幅图片的数字标签。在image文件中，前16个字节为四个整数，代表标识，数据个数，图片高，图片宽，剩下的为按行排列的图片的像素值。测试集的数据类似。

接下来是网络模型的构建，参考了LeNet-5的网络结构如下：



由于mnist的图片格式为28x28，根上述网络结构有所差异，所以对网络进行了如下调整，主要是在卷积层的调整。

对于第一个卷积层，命名为C1，第一个下采样层成为S1，则

|  |  |  |
| --- | --- | --- |
| Layer | Input\_size | Output\_size |
| C1 | 28x28 | 6x24x24 |
| S1 | 6x24x24 | 6x12x12 |
| C2 | 6x12x12 | 16x8x8 |
| S2 | 16x8x8 | 16x4x4 |

并且需要在全连接层中增加一层，因为16x4x4展开后维度即为256

|  |  |  |
| --- | --- | --- |
| FC | 256 | 120 |

在最后输出时，加上了softmax函数来处理得分。

*class* convolution\_neural\_network(*nn*.*Module*):

*def* \_\_init\_\_(*self*):

*super*(convolution\_neural\_network, *self*).\_\_init\_\_()

*# 定义卷积层*

*self*.conv = nn.Sequential(

*#卷积  b\*1\*28\*28 -> b\*6\*24\*24*

            nn.Conv2d(*in\_channels*=1, *out\_channels*=6, *kernel\_size*=5, *stride*=1, *padding*=0),

            nn.Sigmoid(),

*#下采样 b\*6\*24\*24 -> b\*6\*12\*12*

            nn.MaxPool2d(*kernel\_size*=2, *stride*=2),

*#卷积  b\*6\*12\*12 -> b\*16\*8\*8*

            nn.Conv2d(*in\_channels*=6, *out\_channels*=16, *kernel\_size*=5, *stride*=1, *padding*=0),

            nn.Sigmoid(),

*#下采样 b\*16\*8\*8 -> b\*16\*4\*4*

            nn.MaxPool2d(*kernel\_size*=2, *stride*=2)

        )

*self*.fc = nn.Sequential(

*#将特征图展开成向量作为输入，跟上3个全连接层,最后再加上softmax得分*

*#全连接 b\*256 -> b\*120*

            nn.Linear(*in\_features*=256, *out\_features*=120),

            nn.Sigmoid(),

*#全连接 b\*120 -> b\*84*

            nn.Linear(*in\_features*=120, *out\_features*=84),

            nn.Sigmoid(),

*#全连接 b\*84 -> b\*10*

            nn.Linear(*in\_features*=84, *out\_features*=10),

            nn.Softmax(*dim*=1))

然后重写前向传播函数forward，当输入到全连接层时，将特征张量展开。

*def* forward(*self*, *img*):

*#对图像卷积*

        feature = *self*.conv(img)

*#reshap形状为[batch\_size, size]*

        output = *self*.fc(feature.view(-1,256))

        return output

网络的训练

在进行网络训练时，首先指定设备，如果有GPU可用，那么就用GPU，否则才用CPU。然后获取训练集数据，初始化模型，并且将模型放到设备上。

if \_\_name\_\_ == "\_\_main\_\_":

*#如果能用cuda就用cuda*

    device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

    [train\_image, train\_label, n, size] = getData()

    LeNet = convolution\_neural\_network()

    LeNet.to(device)

接下来定义训练的各种参数，如果每次能够处理一个batch的数据，那么网络的训练可以通过并行的计算快速很多。

    epochs = 10  *#epochs次数*

    batch\_size = 300 *#batch大小*

    batchs = n // batch\_size *#有多少个batch*

    learning\_rate = 0.001 *#学习率*

    loss\_func = nn.CrossEntropyLoss() *#交叉熵损失*

    optimizer = optim.Adam(*params*=LeNet.parameters(), *lr*=learning\_rate) *#梯度下降使用Adam优化*

    loss\_list = [] *#存储损失*

接下来对于每一个epoch，都训练所有的batch，并且计算个epoch结束的正确率和损失。

    for epoch in range(epochs):

        accuracy = 0.0

        for batch in range(batchs):

接下来是具体的训练过程，首先将网络的梯度清零，如果不清零的话梯度会累加。

*#梯度清零*

            optimizer.zero\_grad()

然后计算出当前batch所对应的索引范围，假设当前是batch=0，那么下标范围就为[batch×batch\_size, (batch+1)×batch\_size)。

*#得到一个batch索引的上下界*

            l = batch \* batch\_size

            r = l + batch\_size

然后将输入的图片数据与标签转化为规定的张量格式，并放到设备上

*#nn.Conv2d的输入为四维B(bachsize),C(channel),H(high),W(widt)*

*#损失函数的输入参数为predict(N,C),label(N),其中C代表种类数,并且label需要为tensor.long类型*

            image = torch.Tensor(train\_image[l:r]).reshape(-1, 1, size[0], size[1])

            label = torch.tensor(train\_label[l:r]).long()

            image = image.to(device)

            label = label.to(device)

接下来将image输入到网络模型中，得到分类的得分向量后，统计分类正确的个数

*#得到预测向量*

            prediction = LeNet(image)

*#统计正确个数*

            for idx in range(batch\_size):

                if torch.argmax(prediction[idx]) == label[idx]:

                    accuracy += 1

最后，计算预测的标签与真实标签的损失，再进行反向传播，然后再更新权重等信息。

*#计算损失*

            loss = loss\_func(prediction, label)

*#反向传播*

            loss.backward()

*#参数更新*

            optimizer.step()

如果想要得到损失变化趋势的图片，需要将loss从tensor转化为numpy格式。首先无论loss是在GPU还是在CPU上，都将它搬到CPU上，然后取出梯度，在转化为numpy。

*#将损失转到cpu上,去除梯度后转化为numpy*

            loss = loss.cpu().detach().numpy()

            loss\_list.append(loss)

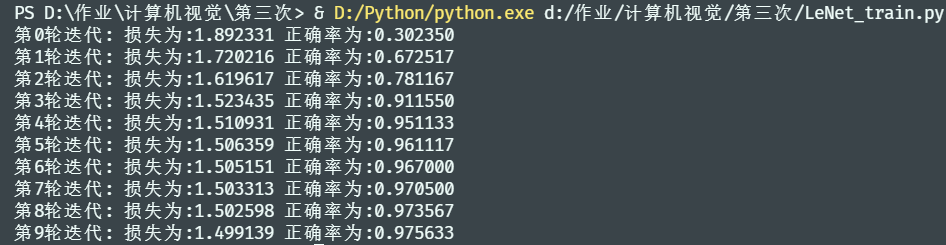
对于每一个eopch，我们都输出它的准确率与损失。

        accuracy = accuracy/n

*#每次输出的损失都为用所有图片训练完一次后的损失*

        print("第%d轮迭代: 损失为:%f 正确率为:%f" % (epoch, loss\_list[-1], accuracy))

结果如下图：



最后保存模型的参数，并且画出损失变化的图像。

*#保存模型的参数为LeNet\_parameter.pt,用字典对应*

    torch.save({"LeNet":LeNet.state\_dict()}, "LeNet\_parameter.pt")

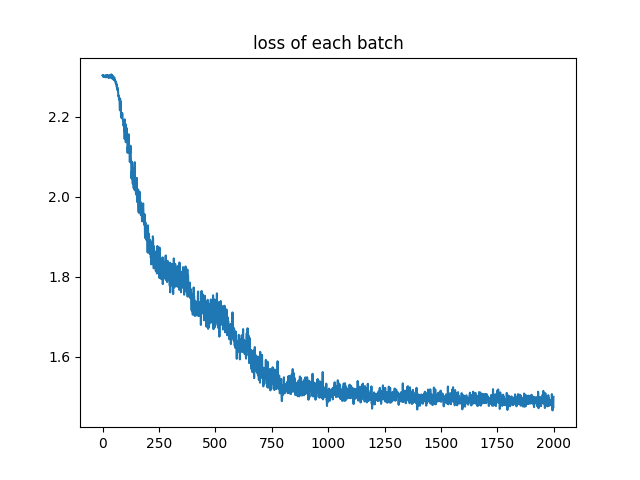
*#画损失图像*

    plt.plot(loss\_list)

    plt.title("loss of each batch")

    plt.savefig("loss.png")

    plt.show()



模型检验

在模型检验时，我创建了一个相同网络结构的模型，并且将先前保存的参数加载进来。

if \_\_name\_\_ == "\_\_main\_\_":

    [test\_image, train\_label, n, size] = getData()

    LeNet = convolution\_neural\_network()

*#加载参数*

    state\_dict = torch.load("LeNet\_parameter.pt")

    LeNet.load\_state\_dict(state\_dict["LeNet"])

然后遍历每一幅测试图片，通过网络得到分类的得分向量，并统计分类正确的个数。最终得到模型在测试集上的准确率。

accuracy = 0.0

for index in range(n):

        image = torch.Tensor(test\_image[index]).reshape(-1, 1, size[0], size[1])

        label =torch.tensor(train\_label[index]).long().reshape(1)

*#得到预测向量*

        prediction = LeNet(image)

        predict\_label = torch.argmax(prediction)

        if predict\_label == label:

            accuracy += 1

    accuracy = accuracy/n

    print("正确率为:%f" % accuracy)

最终在测试集上的结果如下图：

D:\QQ文档\1833791192\Image\C2C\U`PQMGXVT}NPDI}RIT}N~J1.png