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

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

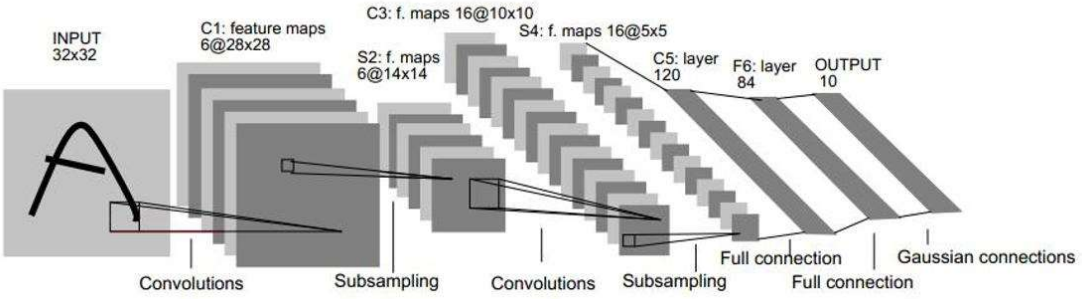
TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

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：

- LeNet-5



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。

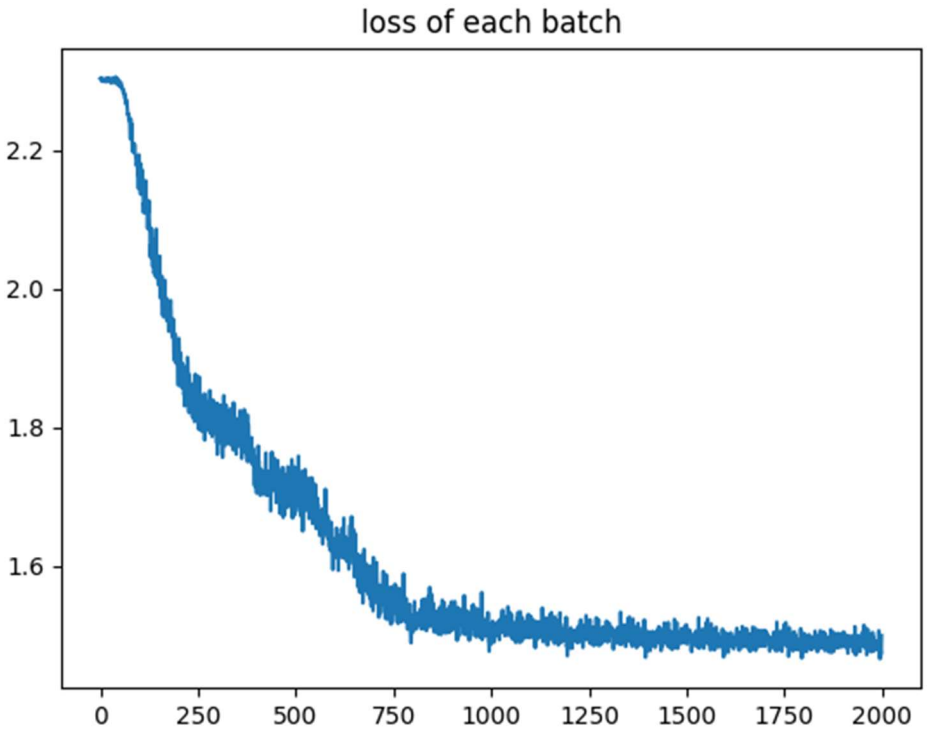
```
        accuracy = accuracy/n
        #每次输出的损失都为用所有图片训练完一次后的损失
        print("第%d 轮迭代: 损失为:%f 正确率为:%f" % (epoch,
loss_list[-1], accuracy))
```

结果如下图：

```
PS D:\作业\计算机视觉\第三次> & D:/Python/python.exe d:/作业/计算机视觉/第三次/LeNet_train.py
第0轮迭代: 损失为:1.892331 正确率为:0.302350
第1轮迭代: 损失为:1.720216 正确率为:0.672517
第2轮迭代: 损失为:1.619617 正确率为:0.781167
第3轮迭代: 损失为:1.523435 正确率为:0.911550
第4轮迭代: 损失为:1.510931 正确率为:0.951133
第5轮迭代: 损失为:1.506359 正确率为:0.961117
第6轮迭代: 损失为:1.505151 正确率为:0.967000
第7轮迭代: 损失为:1.503313 正确率为:0.970500
第8轮迭代: 损失为:1.502598 正确率为:0.973567
第9轮迭代: 损失为:1.499139 正确率为:0.975633
```

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：

```
PS D:\作业\计算机视觉\第三次> & D:/Python/python.exe d:/作业/计算机视觉/第三次/LeNet_test.py
正确率为:0.976800
```