深度学习——典型算法复现

——pytorch实现简单的ResNet对MNIST进行分类

1854062 许之博

开发环境

- 操作系统：Windows10
- 开发工具：Pycharm2020.3
- 主要依赖：python3.8，pytorch1.8.0，torchvision0.9.0

1.背景

1.1 深度学习

深度学习是机器学习领域中一个研究方向。深度学习的概念源于人工神经网络的研究，含多个隐藏层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层表示属性类别或特征，以发现数据的分布式特征表示。典型的深度学习模型有卷积神经网络、深度置信网络和堆栈自编码网络模型等，本实验中选用的卷积神经网络。

1.2 卷积神经网络

卷积神经网络是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表性算法。卷积神经网络具有表征学习能力，能够按其阶层结构对输入信息进行平移不变分类，因此也被称为“平移不变人工神经网络“。一个传统的卷积神经网络模型通常包括输入层，隐含层和输出层。输出层通常包括卷积层，池化层和全连接层。

1.2.1 卷积层

卷积神经网络中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法最佳化得到的。卷积运算的目的是提取输入的不同特征，更多层的网络能从低级特征中迭代提取更复杂的特征。

1.2.2 激活函数

引入激活函数是为了增加神经网络模型的非线性。没有激活函数的每层都相当于矩阵相乘。如果不引入激活函数，就算叠加了若干层之后，无非还是个矩阵相乘罢了。

1.2.3 批标准化

Batch Normalization, 批标准化, 和普通的数据标准化类似, 是将分散的数据统一的一种做法, 也是优化神经网络的一种方法。BN一般添加在卷积层和激活函数之间。BN算法（批标准化）则可以用来规范化某些层或者所有层的输入，从而固定每层输入信号的均值与方差。

1.2.4 池化层

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。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。池化层用的方法有Max pooling 和 average pooling。本实验选用最大池化。

1.2.5 全连接层

全连接层位于卷积神经网络隐含层的最后部分。按表征学习观点，卷积神经网络中的卷积层和池化层能够对输入数据进行特征提取，全连接层的作用则是对提取的特征进行非线性组合以得到输出。全连接层的每一个结点都与上一层的所有结点相连。

1.3 ResNet

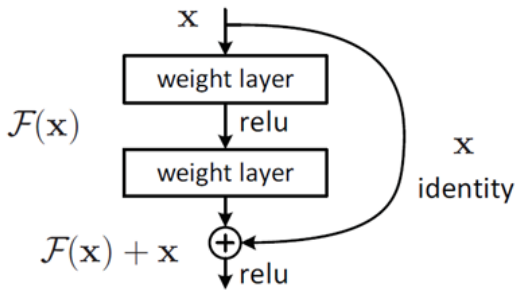
经典的卷积神经网络包括LeNet，AlexNet，GoogleNet，VGG，ResNet等，本实验选取实现一个简单一个简单的ResNet结构，并在MNIST数据集上测试了网络的效果。ResNet的作者何凯明凭借着ResNet摘得了CVPR2016最佳论文奖。

1.3.1 深度网络的退化

从经验来看，网络的深度对模型的性能至关重要，当增加网络层数后，网络可以进行更加复杂的特征模式的提取，所以当模型更深时理论上可以取得更好的结果。但是实验发现深度网络出现了退化问题（Degradation problem）：网络深度增加时，网络准确度出现饱和，甚至下降。

1.3.2 残差学习

何凯明提出了残差学习来解决退化问题。对于一个堆积层结构，当输入为 x时其学习到的特征记为$H(x)$，现在我们希望学习到残差$F(x)=H(x)-x$，这样原始的学习特征是$F(x)+x$。之所以这样是残差学习相比原始直接学习较为容易。当残差为0时，此时堆积层仅仅做了恒等映射，至少网络性能不会下降，这也使得堆积层在输入特征基础上学习到新的特征，从而拥有更好性能。下图为残差学习结构图，也可以理解为短路机制。



残差学习相对更加容易，可以进行如下推导。从数学角度分析，残差单元可以表示如下：

$$y_l = h(x_l) + F(x_l, W_l); \quad x_{l+1} = f(y_l)$$

其中x_l和x_{l+1}分别表示的是第l个残差单元的输入和输出，注意每个残差单元一般包含多层结构。

F是残差函数，$h(x_l)=x_l$表示恒等映射，f是ReLu激活函数，基于上式，从浅层l到深层L的学习特征为：$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$, 利用链式法则，反向求梯度：

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot (1 + \frac{\partial}{\partial x_L} \sum_{i=l}^{L-1} F(x_i, W_i))$$

小括号中的1表示短路机制可以无损地传播梯度，而另外一项残差梯度则需要经过带有weights的层，梯度不是直接传递过来的。残差梯度不会那么巧全为-1，而且就算其比较小，有1的存在也不会导致梯度消失。所以残差学习会更容易。

1.4 训练

训练过程需要计算每轮网络的输出结果与实际值的损失函数，然后使用梯度下降等优化方法对网络的参数进行调整。

1.4.1 损失函数

在模型训练中，损失函数用来衡量预测值与真实值之间的误差，数值越小表示误差越小。梯度下降法在机器学习中应用十分的广泛，不论是在线性回归还是Logistic回归中，它通过不断迭代更新w的参数，从而使损失函数最小化。在pytorch框架中，反向传播是利用backward方法实现的。

```
loss = F.nll_loss(output, target)#负对数似然函数损失
```

1.4.2 优化器

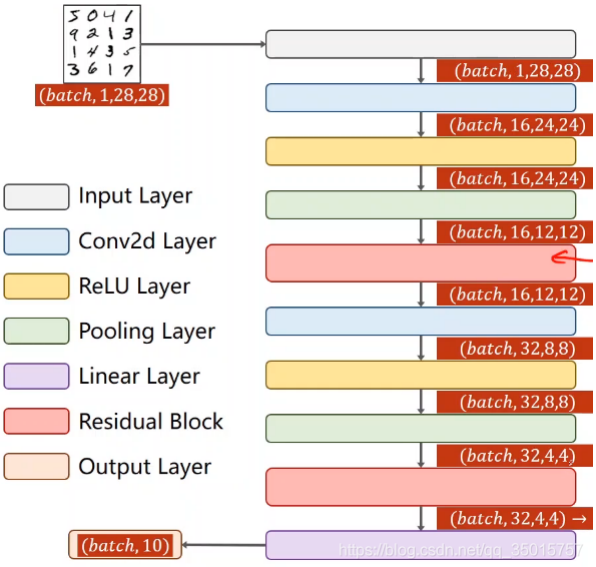
利用损失函数来训练神经网络，本质上，需要做的就是计算损失并尽量减少损失，最小化（或最大化）任何数学表达式的过程称为优化。优化器就是在深度学习反向传播过程中，指引损失函数的各个参数往正确的方向更新合适的大小，使得更新后的各个参数让损失函数值不断逼近全局最小。常用的优化器有随机梯度下降(SGD)，自适应梯度下降(AdaGrad)等等。

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)#随机梯度下降
```

2.实现过程

2.1 神经网络搭建

本实验中所实现的神经网络结构如下图所示



2.1.1 残差块

```
class ResidualBlock(nn.Module):
    """
    每一个ResidualBlock,需要保证输入和输出的维度不变
    所以卷积核的通道数都设置成一样
    """
    def __init__(self, channel):
        super().__init__()
        self.conv1 = nn.Conv2d(channel, channel, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(channel)#批标准化
        self.relu = nn.ReLU(inplace=True)#ReLu激活函数
        self.conv2 = nn.Conv2d(channel, channel, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(channel)
    def forward(self, x):
        identify = x
```

```
        out = self.bn1(self.conv1(x))
        out = self.relu(out)
        out = self.bn2(self.conv2(out))
        out += identify
        return out
```

2.1.2 残差网络

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=5)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5)
        self.res_block_1 = ResidualBlock(16)#残差块
        self.res_block_2 = ResidualBlock(32)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(512, 10)
    def forward(self, x):
        in_size = x.size(0)
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)#池化
        x = self.res_block_1(x)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = self.res_block_2(x)
        x = x.view(in_size, -1)
        x = self.fc1(x)#全连接层
        return F.log_softmax(x, dim=1)
```

2.2 训练

```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()# 优化器梯度初始化为零
        output = model(data)# 负对数似然函数损失
        loss = F.nll_loss(output, target)
        loss.backward()#反向传播计算梯度
        optimizer.step()#调整网络参数
        if batch_idx % args.log_interval == 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()
            ))
```

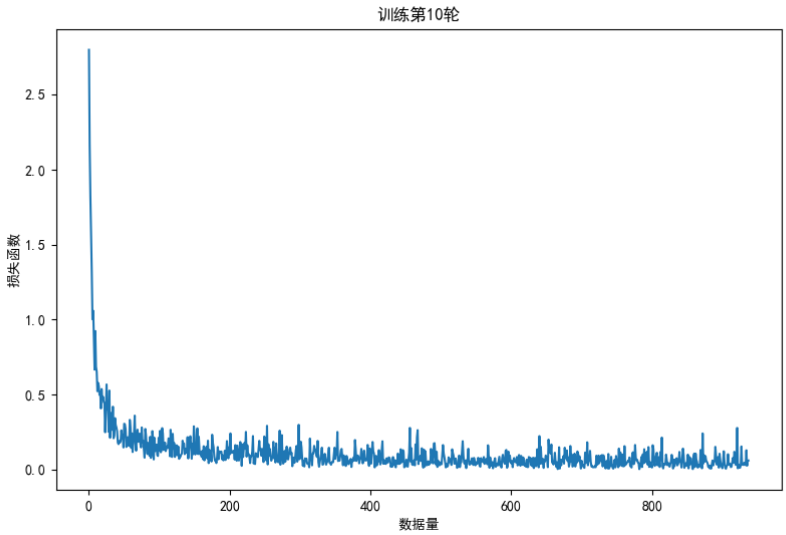
3. 效果和总结

3.1 测试效果

```
def test():
    # 设置为test模式
    model.eval()
    # 初始化测试损失值为0
```

```
    test_loss = 0
    # 初始化预测正确的数据个数为0
    correct = 0
    for data, target in test_loader:
        if args.cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        output = model(data)
        # 把所有loss值进行累加
        test_loss += F.nll_loss(output, target, size_average=False).item()
        # 获取最大对数概率值的索引
        pred = output.data.max(1, keepdim=True)[1]
        # 对预测正确的个数进行累加
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()
    # 因为把所有loss值进行累加,所以最后要除以总的数据长度才能得到平均loss
    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)
    ))
```

训练10轮后




```
Train Epoch: 10 [56960/60000 (95%)] loss: 0.023533
Train Epoch: 10 [57600/60000 (96%)] loss: 0.000676
Train Epoch: 10 [58240/60000 (97%)] loss: 0.002217
Train Epoch: 10 [58880/60000 (98%)] loss: 0.000730
Train Epoch: 10 [59520/60000 (99%)] loss: 0.001927

Test set: Average loss: 0.0261, Accuracy: 9910/10000 (99%)


Process finished with exit code 0
```

3.2 总结

1. pytorch内置有一个resnet18网络，可以直接调用，但通过自己搭建一个简单的ResNet更帮助我理解ResNet的实现逻辑。
2. 通过本次实践更加熟悉卷积神经网络的训练过程以及网络的准确率测试
3. 课堂学习与课外实践相结合，增强了对深度学习的了解
4. 深度学习的内容还很丰富，目前只是学习了九牛一毛，还需要加深理解并更多实践。