北京交通大學

信息网络综合专题研究课

大作业

（应用层-机器学习）

用 pytorch 实现 CNN 卷积神经网络的搭建、训练和测试

班级：通信 1706 班

姓名：崔昭阳

学号：17211111

时间：2020 年 5 月

一 实验目的

用 python 语言实现深度学习的卷积神经网络（CNN）模块，包括 CNN 的搭建、训练和测试，学习 CNN 使用的结构和算法，研究 CNN 的运行过程和特点。

二 实验原理

卷积神经网络 CNN 是深度学习的主要算法之一，CNN 的网络结构是分层的，主要结构是卷积层，池化层和全连接层。卷积层的主要功能是根据卷积核的参数来提取数据特征，卷积层通常用激活函数加入非线性因素的方法来模拟细微的变化，最常用的激活函数是 ReLU 函数，池化层的主要功能是降低数据维度，突出主要特征。一般采用多个卷积层和池化层，每个卷积核的参数不同，这样就可以提取到不同的重要特征。连接层用来连接所有的特征，将输出值送给分类器，实现前向传播。训练主要是训练权重参数，测试是检验结果，最终得到准确率来评估 CNN 的性能。

Pytorch 是专门为深度学习设置的 python 环境，里面包含了深度学习大部分的函数和结构。逐行分析讲解 CNN 代码，并在 Pytorch 环境中运行来完成实验目的。

程序代码来源：https://github.com/yunjey/pytorch-tutorial

三 代码讲解

（1）基本参数设置：

调用 torch 和 torchvision，判定 GPU 是否存在，定义超参数：训练次数 num_epochs = 5，识别的类别数量 num_classes = 10，每个批次加载的样本容量的大小 batch_size = 100，学习率 learning_rate = 0.001。

（2）数据设置：

下载 MNIST 数据集，训练数据和测试数据使用的都是 torchvision.datasets 模块的 MNIST 数据集，并构建相应的数据通道。

（3）定义卷积神经网络模型：

1.通过 nn.Module 类来实现 ConvNet（两个卷积池化层）的定义，使用函数 nn.Sequential：

第一层：①定义卷积层，首先使用函数 nn.Conv2d 设置：输入信号通道数为 1，输出信号的通道数为 16，卷积核的大小 kernel_size=5（长和宽相同），步长 stride=1，每一维补零的数量 padding=2。然后使用函数 nn.BatchNorm2d 设置卷积的输出通道数为 16，最后使用激活函数 nn.ReLU 激活。② 定义池化层，使用函数 nn.MaxPool2d 设置：卷积核的大小 kernel_size=2（长和宽相同），步长 stride=2。第一层的输入维度为（1，28，28），输出维度为（16，14，14）。

第二层：函数 nn.Conv2d 的输入信号通道数为 16，输出信号的通道数为 32，函数 nn.BatchNorm2d 设置卷积输出通道数为 32，其余参数与第一层一样。第二层与第一层的结

构也完全相同。第二层的输入维度为（16，14，14），输出维度为（32，7，7）。

最后是全连接层，输出 10 个类别，输出维度为（7，7，32）。

层次结构：卷积层+池化层+卷积层+池化层+全连接层

2.再使用 nn.Module 类中的向前传播函数 forward，定义可学习参数的卷积神经网络结构 self，输入数据集 x，使用定义的 CNN 结构进行处理，转化为合适的输入形式后依次经过第一层（卷积层+池化层）、第二层（卷积层+池化层）和全连接层，这样就完成了 CNN 的前向传播，最后得到 output，这就是向前传播的过程。

把以上定义卷积神经网络模型用 model 表示。

（4）损失和优化设置：

使用 pytorch 中的交叉熵损失函数 nn.CrossEntropyLoss 来初始化 loss 函数。使用函数 torch.optim 的 Adam 算法构建优化器 optimizer，使用 0.001 的学习率，优化器能够根据计算得到的梯度来更新参数。

（5）训练 CNN 模型：

首先遍历所有的训练数据，得出总长度 total_step，然后进行 num_epochs =5 次的重复循环训练。在每次重复训练中，主要有两步： ① Forward pass，先把训练集图像输入 model 模型中进行向前传播，通过运算得到输出 outputs，再将 output 和样本位置输入 loss 函数，计算得到 loss 的标量值；② Backward and optimize，先使用函数 optimizer.zero_grad 把梯度置零， 因为 PyTorch 默认会对梯度进行累加，所以如果不想先前的梯度影响当前梯度的计算，需要手动清零。再使用函数 loss.backward 实现误差反向传播，进而计算参数梯度，最后通过函数 optimizer.step 实现参数更新，使训练模型达到进一步优化的目的。

参数更新公式：新参数 = 原来的参数 + 学习速率 x 梯度向量 （参数为向量的形式）

在每次重复训练中，一个批次的样本容量为 100，,每批次训练完后都有一个输出 print，输出的内容为循环的次数，训练的样本位置，还有 loss 值（保留 4 位小数），具体格式如下：

```
print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}' .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

（6）测试 CNN 模型：

首先定义一个模型测试函数 model.eval，初始化 correct = 0，total = 0。从第一个数据开始，每一批的样本容量大小为 100，所以每次循环 total=total+100，直到没有数据时循环停止。每次循环从数据集中获取图像和样本位置，再把获得的图像输入 model 模型中得到 outputs，output.data 是列数为 100，行数为 10 的二维张量，用函数 torch.max 取张量中每行最大的值放到 predicted 中，predicted 就是包含 100 个数据的一维张量，将 predicted 与原来 100 个样本数据的一维张量的对应数据一一比较，如果相等则 correct 加 1，如果不相等则 correct 加 0，即 correct 不变。就可以用 correct/total 来表示准确率了。Print 格式如下：

```
print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct / total))
```

（7）保存模型参数：

最后，使用函数 torch.save 将模型参数 model.state_dict 保存到 model.ckpt 文件中。

四 实验步骤

1. 配置 pytorch 环境

要求使用 tensorflow、torch、torchvision、scipy、numpy 包。使用的 anaconda3 中 scipy 和 numpy 包已经存在，还需要安装其余的包。

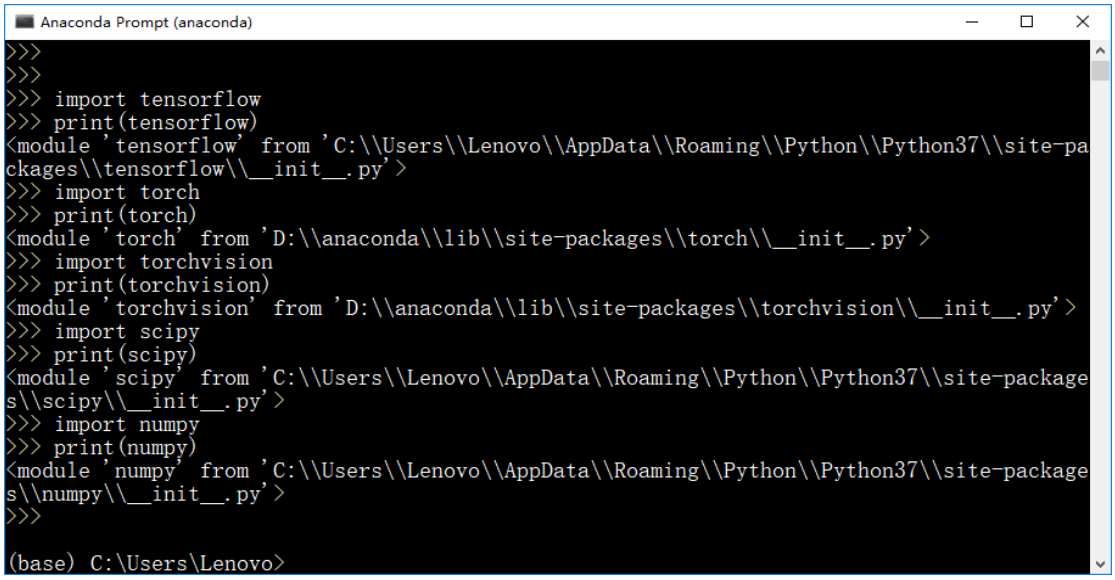
在 Anaconda Prompt 中经过多次调试最终确定了以下可用的指令：

conda install pytorch-cpu=1.1.0

conda install torchvision-cpu

pip install --user --upgrade tensorflow-gpu

测试环境安装成功：CNN 代码可以运行。测试结果如下图：



2. 运行结果

程序输出与之前分析的一致，一共循环训练 5 次，每次有 6 批，每个批次有 100 的样本容量，总共输出 30 个 loss 值，模型测试的准确率为 99.06 %，因为 loader 设置的 shuffle=True，所以 loss 值和准确率每次运行都会产生不同的值。以下是程序运行的一次输出：

runfile('C:/Users/Lenovo/.spyder-py3/CNN.py', wdir='C:/Users/Lenovo/.spyder-py3')

Epoch [1/5], Step [100/600], Loss: 0.1707

Epoch [1/5], Step [200/600], Loss: 0.1325

Epoch [1/5], Step [300/600], Loss: 0.0503

Epoch [1/5], Step [400/600], Loss: 0.0340

Epoch [1/5], Step [500/600], Loss: 0.1231

Epoch [1/5], Step [600/600], Loss: 0.0244

Epoch [2/5], Step [100/600], Loss: 0.0330

Epoch [2/5], Step [200/600], Loss: 0.0362

Epoch [2/5], Step [300/600], Loss: 0.0210

Epoch [2/5], Step [400/600], Loss: 0.0442

Epoch [2/5], Step [500/600], Loss: 0.0685

Epoch [2/5], Step [600/600], Loss: 0.0106

Epoch [3/5], Step [100/600], Loss: 0.0988

Epoch [3/5], Step [200/600], Loss: 0.0167

Epoch [3/5], Step [300/600], Loss: 0.0158

Epoch [3/5], Step [400/600], Loss: 0.0023

Epoch [3/5], Step [500/600], Loss: 0.0386

Epoch [3/5], Step [600/600], Loss: 0.0103

Epoch [4/5], Step [100/600], Loss: 0.0202

Epoch [4/5], Step [200/600], Loss: 0.0106

Epoch [4/5], Step [300/600], Loss: 0.0038

Epoch [4/5], Step [400/600], Loss: 0.0047

Epoch [4/5], Step [500/600], Loss: 0.0178

Epoch [4/5], Step [600/600], Loss: 0.0596

Epoch [5/5], Step [100/600], Loss: 0.0038

Epoch [5/5], Step [200/600], Loss: 0.0070

Epoch [5/5], Step [300/600], Loss: 0.0264

Epoch [5/5], Step [400/600], Loss: 0.0030

Epoch [5/5], Step [500/600], Loss: 0.0119

Epoch [5/5], Step [600/600], Loss: 0.0056

Test Accuracy of the model on the 10000 test images: 99.06 %

3. 结果分析

对输出结果做简要的分析：对每一次训练来说，把 6 个批次的 loss 值相加得到一次训练的总 loss 值，第 1 次总 loss 值为 0.5350，第 2 次总 loss 值为 0.2135，第 3 次总 loss 值为 0.1825，第 4 次总 loss 值为 0.1167，第 5 次总 loss 值为 0.0577，可以明显的看出随着训练次数的增加，loss 值在下降，即优化器对参数权重实现了优化，达到了模型训练的目的。测试的准确率也高达 99.06 %，pytorch 构建的 CNN 模型的效果很好。

五 实验总结

本次实验学到了很多知识，收获到了很多经验。本次实验用 python 的 pytorch 库实现 CNN 卷积神经网络的搭建、训练和测试，代码中结构和算法与 CNN 的理论分析相吻合，既用 pytorch 实现了 CNN 结构也仔细研究了其使用的函数和算法，因此对 CNN 的运行过程和特点有了很深刻的认识，对今后的机器学习产生了很大的帮助。另外，对 python 环境的搭建和 python 语言的学习也获得了很多重要的经验。

此外，在附录中还提供了可运行的实验代码、数据集的建立过程和程序运行的面板图。