实验 13- PyTorch

姓名：王煌基　　学号：519030910100　　班级：F1903004

一、实验概览

基于本次对于机器学习与深度学习的初步认识，了解神经网络及神经元的概念，理解模型训练过程的参量变化及参量意义，在此基础上，利用深度学习实现对初等函数的拟合以及对 CIFAR-10 图片进行分类。

二、实验环境

1.个人笔记本电脑

2.操作系统：windows10 专业版

3.使用软件：Visual Studio Code；Docker Desktop

三、实验过程

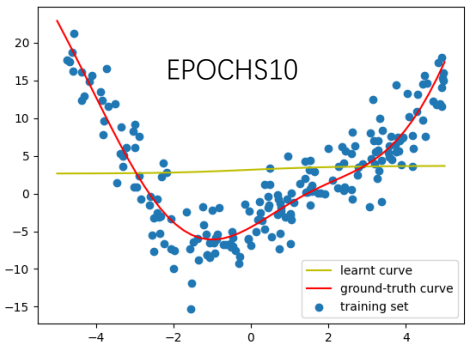
练习一：利用深度学习实现对非线性函数 f(x) = x^2 + 2sin(x) + cos(x-1) -5 的拟合（见 exp1.py）。本实验将帮助你理解深度学习的基本流程。

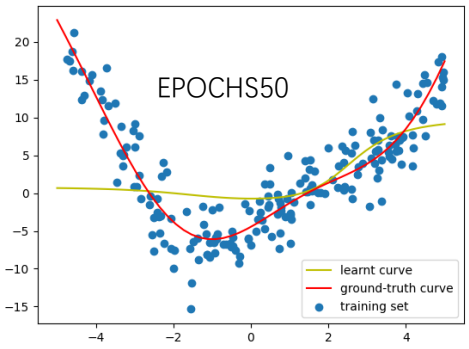
A. 不对代码其他部分进行改动，更改参数 NUM_TRAIN_EPOCHS 为 100,1000,10000,50000，你发现拟合得到的曲线有什么变化？

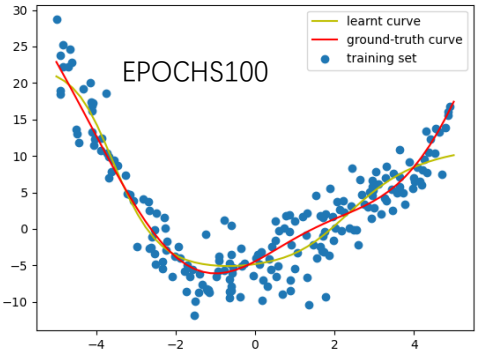
【解】

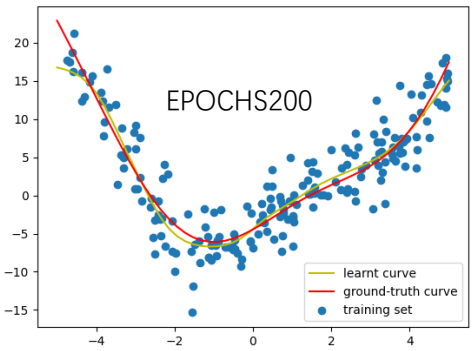
更改 NUM_TRAIN_EPOCHS 的大小之后，可以发现，随着该数值的增大，该学习曲线(learnt curve)越来越接近于点的原始数据，而该数值的小规模状态下由于数据规模过小，没有较好地进行模型的训练，拟合的效果也不好，故整体呈现的曲线拟合精度应该是随着 NUM_TRAIN_EPOCHS 的增大，曲线拟合的效果先增大，再减小，在某个具体的训练批次能够取到最大值，接近真实曲线(ground-truth curve)。为了验证我的想法，我额外进行了 NUM_TRAIN_EPOCHS=10、50、200 以及 100000 的数据规模，通过图像，可以确定我的想法基本是正确的，即准确率（拟合曲线的相关系数）应该满足类似于正态分布的曲线。

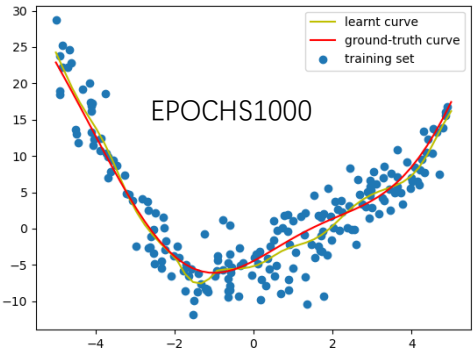
【实验图像】

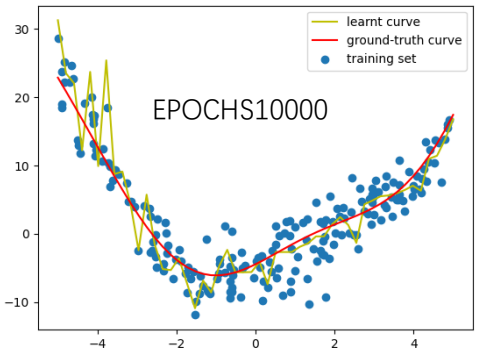

EPOCHS10
learnt curve
ground-truth curve
training set

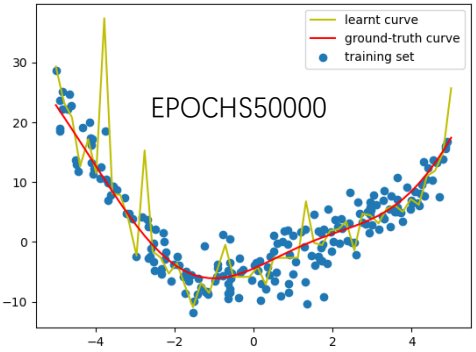

EPOCHS50
learnt curve
ground-truth curve
training set

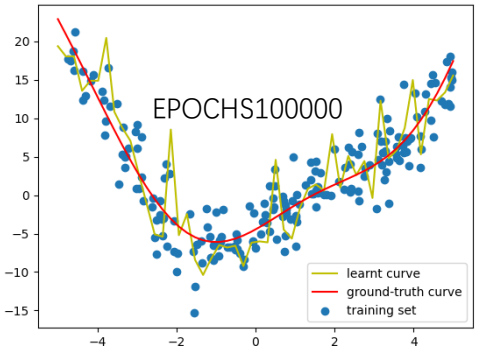

EPOCHS100
learnt curve
ground-truth curve
training set


EPOCHS200
learnt curve
ground-truth curve
training set


EPOCHS1000
learnt curve
ground-truth curve
training set


EPOCHS10000
learnt curve
ground-truth curve
training set


EPOCHS50000
learnt curve
ground-truth curve
training set


EPOCHS100000
learnt curve
ground-truth curve
training set

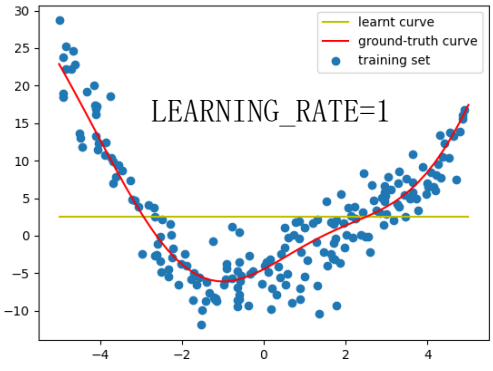
B. 固定 NUM_TRAIN_EPOCHS=1000，更改参数 LEARNING_RATE 为 1, 0.1, 0.01, 0.001，你发现拟合得到的曲线有什么变化？

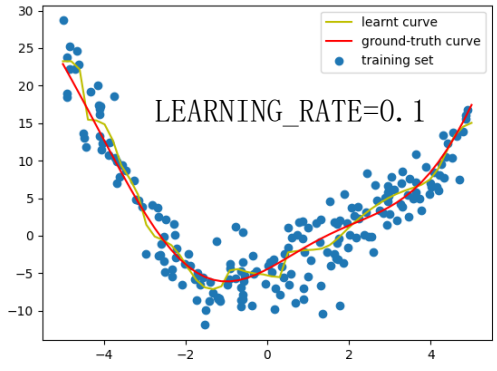
【解】

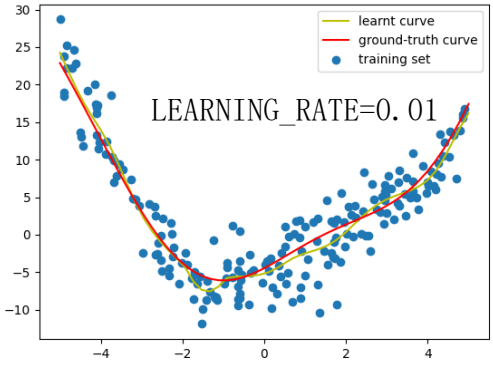
可以发现，随着学习率 LEARNING_RATE 的减小，带来的时间花费更大，例如，同样的状态转移过程$x \to x-1$，当 LEARNING_RATE 是 1 的时候，仅需要一步即可到达，而 LEARNING_RATE 为 0.1 时却需要经过这样的步骤：$x \to x-0.1 \to x-0.2 \to x-0.3 \to \cdots \to x-0.9 \to x-1.0$，即当 LEARNING_RATE 变为原来的十分之一时，耗费的步骤代价可能就是原来的十倍，因此时间代价是接近于指数级别的增长的。

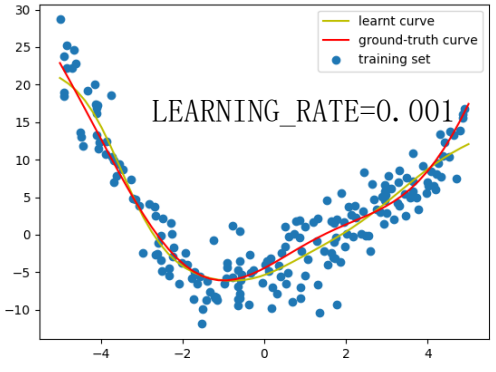
但是，于此同时我也注意到 LEARNING_RATE 的下调是有一定好处的，具体体现在准确率的表现上，简单的来说，对于转移过程$x \to x-0.045$（此处假设$x-0.045$是最佳的状态），当 LEARNING_RATE=1 时，最理想的状态只有$x \to x-1$或$x \to x+1$，始终无法到达$x-0.045$，而且相邻的两个状态$x-1, x+1$与最佳状态$x-0.045$均差距甚远，从而容易导致较大误差，当 LEARNING_RATE 按 10 倍缩小时，每次的步长也按照 10 倍缩小，进而能达到的准确率就会变得更高，当 LEARNING_RATE=0.001 时，显然，$x-0.045$这个状态是可以得到的，进而可以说明，随着 LEARNING_RATE 的降低，准确度将会逐渐升高。

【实验结果】









C. 自定义一个函数 f(x)，调整合适的参数，使得模型拟合效果尽可能好。

【解】

这里我利用的是一个我当年高考数学导数题的函数的改写方式。由于在 exp1.py 中并不支持负数方向的 f(x)值，我通过限定 x 为平方项的形式来处理这个函数，具体表达式为：

$$f(x) = x^2 + 2x + 1 + \sin(x^2 + 1) - \log(x^2 + 1) = (x + 1)^2 + \sin(x^2 + 1) - \log(x^2 + 1)$$

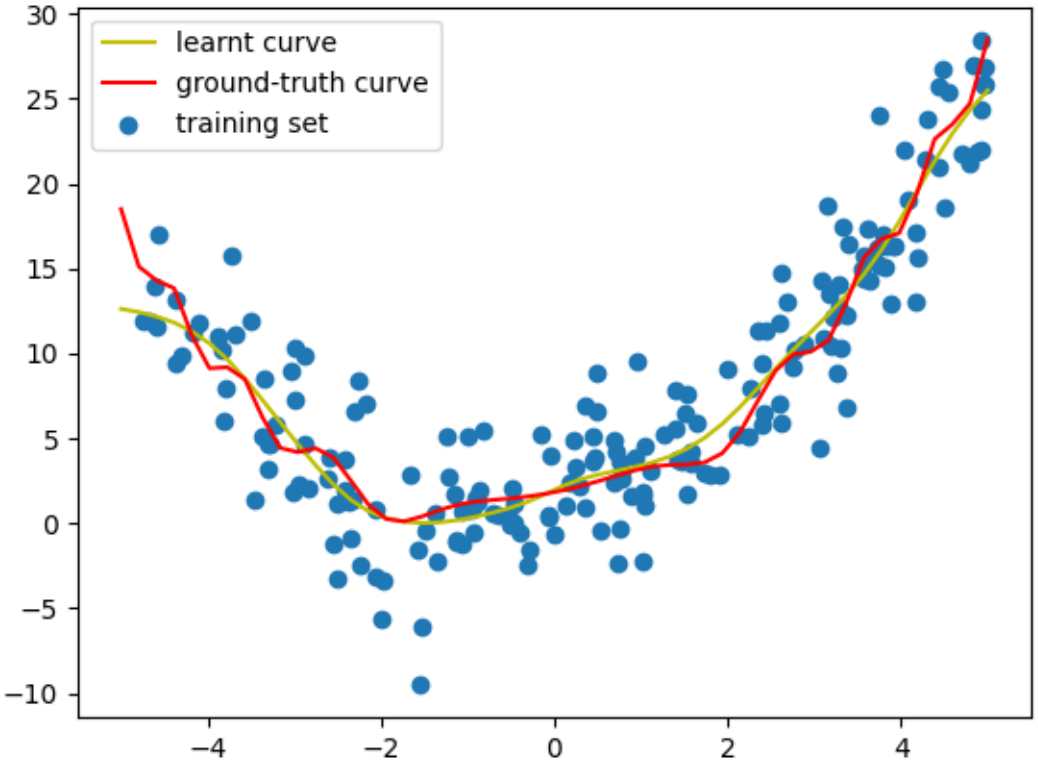
通过前面的分析很容易地发现，当 LEARNING_RATE 尽可能小的时候得到的拟合效果是比较好的，故此处我的 LEARNING_RATE 取 0.001 的值来进行处理。

然而，在训练轮次数的选择上面由于取数范围广，导致了在选取具体数值的时候本身是比较困难，且不容易进行的。

这里我进行了比较粗略的比较方式，即人工二分的方式，假定可以认为本次实验的拟合效果服从正态分布，则可以用 NUM_TRAIN_EPOCHS=200、500、1000、2000、3000、3500、4000 等数值的查找，经过多次比较，初步认为当 NUM_TRAIN_EPOCHS=3500 时拟合的效果相对比较好。

【实验结果】

LEARNING_RATE=0.001　NUM_TRAIN_EPOCHS=3500



练习二：补充 exp2.py 的代码，使得程序可以完整运行。使用 resnet20 模型，训练一个 cifar-10 的分类器。（推荐训练策略：以 0.1 的学习率(learning rate, lr)训练 5 个 epoch，再以 0.01 的 lr 训练 5 个 epoch。）。

【解】

首先需要补充 exp2.py 的代码。

通过对 train(epoch)函数的参考，可以发现 train(epoch)本质上就是对每一个批次进行训练，得到一个模型 model，然后在训练的过程中需要每次都将过程中产生的优化器的步长 optimizer.step()进行更新调整，并且可以将训练好的相应参数输出（例如准确率 acc 等）。

由此，我认为，test(epoch)就是利用 train(epoch)内训练好的模型来进行测试，测试过程中将测试得到的准确率 acc 等参数输出来比较说明模型训练的情况，从而理清实验原理后可以较为简单地直接参考 train(epoch)来完善 test(epoch)的主要内容。

由于直接输出在终端的方式并不美观直接，故我通过 python 文件读写的方式存储。

综上所述，可以得到相应的代码为：

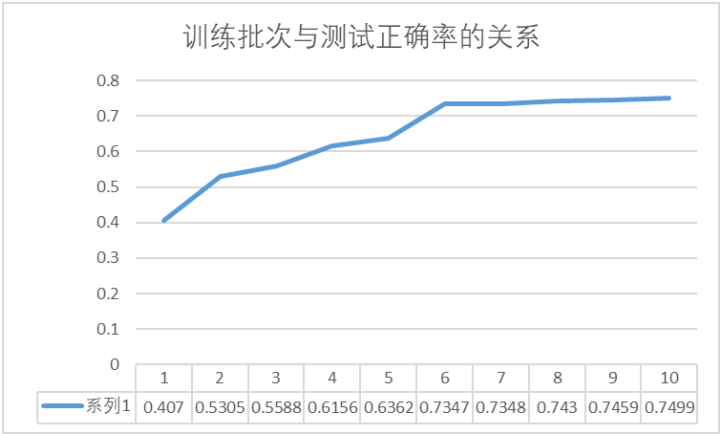
```
train_loss = 0
correct = 0
total = 0
for batch_idx, (inputs, targets) in enumerate(testloader):
    # optimizer.zero_grad(): 优化器进行初始化
    optimizer.zero_grad()
    # inputs 是 x, model 是 function, outputs 是 f(x)
    outputs = model(inputs)
    # 损失（loss）：神经网络输出和目标之间的距离
    loss = criterion(outputs, targets)
    # train_loss 是每次测试的损失
    train_loss += loss.item()
    # 记录输出的最大值（最好的匹配结果）
    _, predicted = outputs.max(1)
    # 记录目标总数
    total += targets.size(0)
    # 记录目标正确匹配数
    correct += predicted.eq(targets).sum().item()
    with open('result.txt', 'a') as f:
        f.write('TEST::Epoch [%d] Batch [%d/%d] Loss: %.3f | Traininig Acc: %.3f%% (%d/%d)
    \n'% (epoch, batch_idx + 1, len(testloader), train_loss / (batch_idx + 1), 100. * correct
    / total, correct, total))
acc = 1.0 * correct / total
```

接下来，利用推荐训练策略，可以得到我们的代码为：

```
start_epoch = 0
end_epoch = 4
```

```
lr = 0.1
for epoch in range(start_epoch, end_epoch + 1):
    train(epoch)
    test(epoch)
start_epoch = 5
end_epoch = 9
lr = 0.01
optimizer = optim.SGD(model.parameters(), lr=lr, weight_decay=5e-4)
for epoch in range(start_epoch, end_epoch + 1):
    train(epoch)
    test(epoch)
```

本质上就是先进行 0~4 批次的 lr=0.1 的训练，再进行 5~9 批次的 lr=0.01 的训练，最后对于每一个批次（10000 张图片），我们用我们的模型进行测试，得到相应的结果如下：



【思考】在 lr 从 0.1 变到 0.01 后，acc 发生了什么变化？为什么？

可以发现，当 lr 从 0.1 转变至 0.01 时（图中的 5、6 点）会突然出现测试匹配率 acc 突然急剧升高的情况，我认为出现这个情况的原因主要是因为由于 lr 变小，能够修正的最小步长变大了，虽然时间上可能耗费的代价更大，但是更长的时间换来的是更优的准确率匹配，可以理解为，由于学习率 LEARNING_RATE 的减小，带来的准确度的区间变大了。如前文的例子中对于转移过程$x \to x - 0.045$（此处假设$x - 0.045$是最佳的状态）的分析，由于步长减小，带来的精度是增加的，故当突然设置了一个 lr 值的减小并且在模型训练还不够充分的情况下，可以使得准确率突然升高。并且由于这种情况的出现，在第六次训练之后实际上模型已经基本稳定了，导致后面的曲线相对平坦许多。

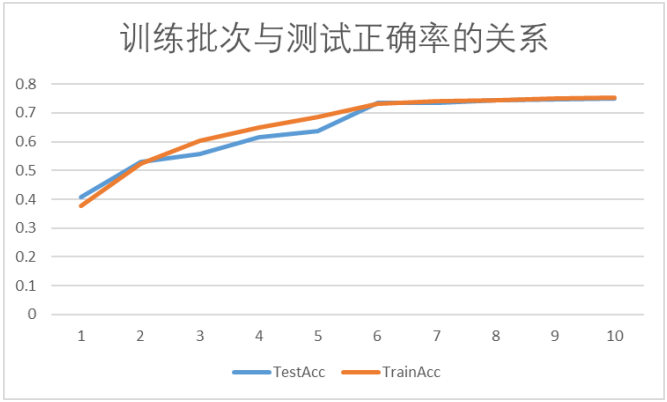
【思考】Train acc 和 Test acc 有什么关联和不同?

在计算公式上面，Train acc 和 Test acc 是一样的，均为$acc = \frac{Correct}{tot}$。

在原理上二者是不同的，前者是在对模型进行训练的过程中产生的参数值，在实现过程中优化器 optimizer 的步长等参数仍在进行修改，而后者由于进行了 model.value()的值锁定，故其优化器的步长始终为定值，因而二者的值是不一样的。

此外，二者在生成时所采取的对象是不同的，前者采用的是 trainset 数据集，后者采用的是 testset 数据集，每一个 batch 的大小虽然都是 128，但是前者有 391batches，后者仅有 79batches。

最后，在正确率的趋势上，二者都呈现了“随着训练/测试批次的增加，正确率也在单调增长”的稳定趋势，且增长幅度是相似的。



四、实验体会

这是本课程的倒数第二次实验，将近学期末，我的收获非常丰富。

在本次实验中，需要尝试全新的知识“机器学习”，实验中能够通过对 pytorch 的难度比较不大的应用来初步接触机器学习的概念及工具，这对于像我一样的初学者来说是比较容易吸收接受的一个概念、工具、手段。

当然，本次实验也首次让我意识到，高端配置的设备环境有多么的重要。同样的一个 exp2.py 的程序，同学利用商务本跑可能需要一个半小时，而我利用游戏本却只需要不到 1 小时，这是相当惊人的效率所在，进而，我也不难理解为什么实验室的学长、老师对芯片是非常看重的，尤其是机器学习方面的。

希望最后一次实验，我能依旧保持最饱满的热情去完成！