第十三次实验报告

学号：518030910308
姓名：刘文轩

一、 实验准备

1、 实验环境介绍

操作系统：ubuntu 14.04
语言：Python 2
IDE：Pycharm 2019.2.3

2、 实验目的

2.1 Pytorch 入门
2.2 了解神经网络与机器学习的概念
2.3 了解深度学习的概念
2.4 学会拟合初等函数
2.5 CIFAR-10 图片分类

3、实验思路

3.1 在 exp1.py 中分别调整 NUM_TRAIN_EPOCHS，LEARNING_RATE，函数参数，观察拟合的曲线的变化，获得拟合效果更好的模型

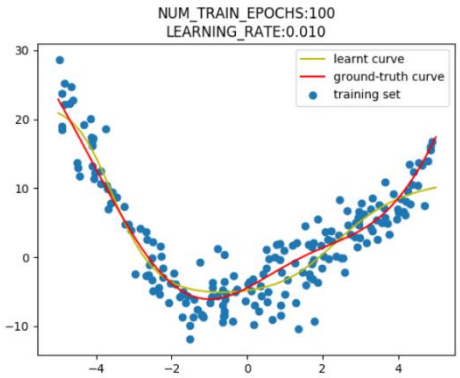
3.2 在 exp2.py 中，首先了解 CIFAR-10，然后在了解了 train 函数后，自己编写 test 函数，并且统计 test_acc 的变化趋势

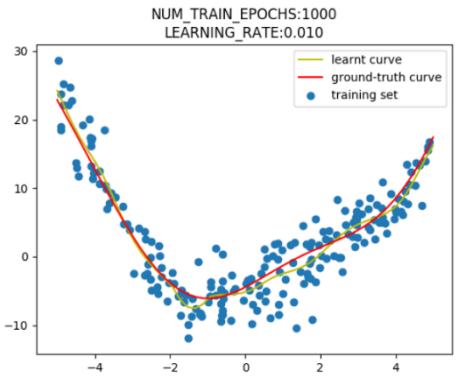
二、 实验过程

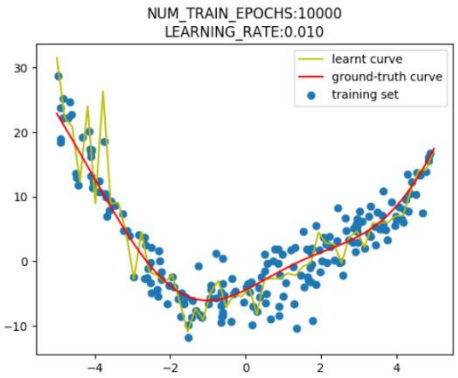
1、 exp1.py

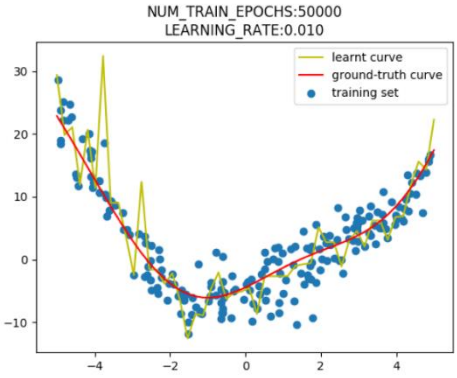
1.1 更改参数 NUM_TRAIN_EPOCHS

在此部分中，在保持其它参数不变的情况下，我们更改参数 NUM_TRAIN_EPOCHS 为 100,1000,10000,50000，得到的实验结果如下所示，相关参数已经在图上标出：









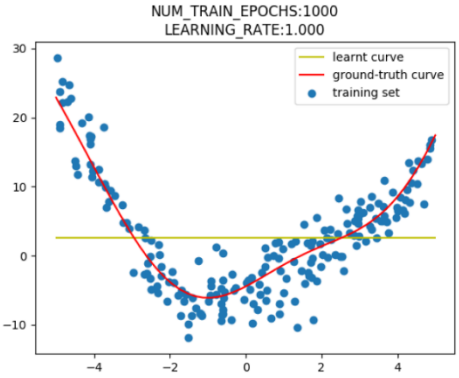
由图中可见，随着 Epoch 次数的增加，所得的曲线先由欠拟合变得拟合，再变得过拟合。

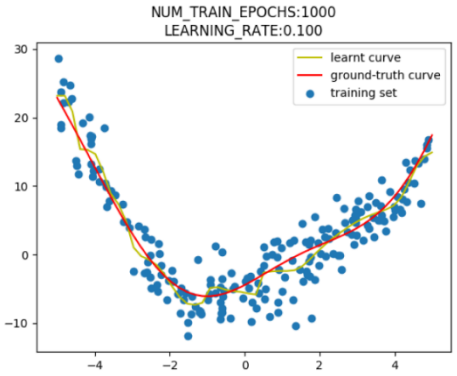
Epoch 的意义在于：在神经网络中传递完整的数据集一次是不够的，而且我们需要将完整的数据集在同样的神经网络中传递多次。但是由于我们使用的是有限的数据集，并且我们在一个迭代过程中是梯度下降的，因此仅仅更新权重一次或者说使用一个 Epoch 是不够的。

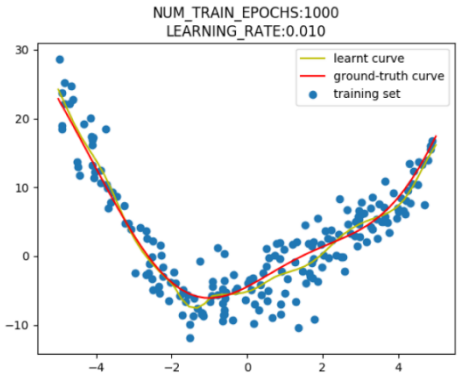
随着 Epoch 数量增加，神经网络中的权重的更新次数也增加，曲线从欠拟合变得过拟合。那么几个 Epoch 才是合适的呢？不幸的是，这个问题并没有正确的答案。对于不同的数据集，答案是不一样的。但是数据的多样性会影响合适的 Epoch 的数量。

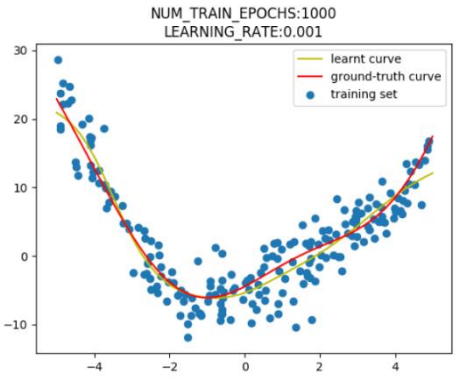
1.2 更改参数 LEARING_RATE

在此部分中，在保持其它参数不变的情况下（固定 NUM_TRAIN_EPOCHS=1000），我们更改参数 LEARING_RATE 为 1,0.1,0.01,0.001，得到的实验结果如下所示，相关参数已经在图上标出：









由图形可见，当学习率设置的过小时，收敛过程将变得十分缓慢。而当学习率设置的过大时，梯度可能会在最小值附近来回震荡，甚至可能无法收敛，如第一张图片，所得的拟合曲线完全就是一条直线，拟合是很不理想的。

合适的学习率能够使目标函数在合适的时间内收敛到局部最小值。

1.3 自定义函数

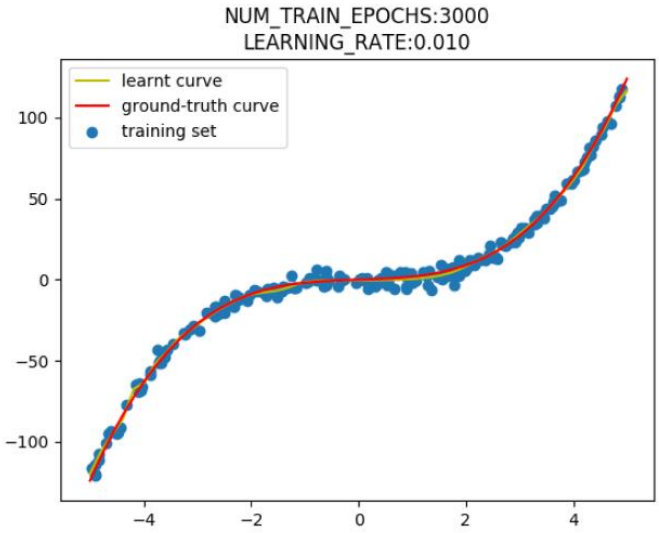
在这里我们设计的自定义函数为：

$$f(x) = x^3 + sinx$$

在经过多次调整后，设置的参数为：

$$NUM\_TRAIN\_SAMPLES = 200$$
$$NUM\_TRAIN\_EPOCHS = 3000$$
$$LEARNING\_RATE = 0.01$$

所得的拟合图像为：

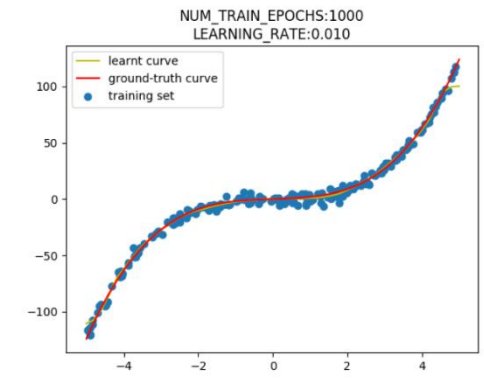


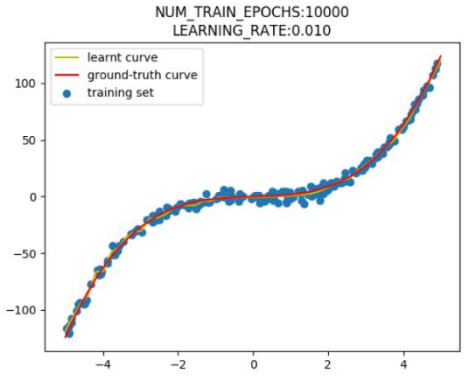
由图可见，实验所得的拟合曲线与原曲线的重合程度很高，拟合结果很好。这主要是由于我结合之前几个小问的经验选择了合适的参数。

首先，学习率就设置为 0.01，如果设置为 0.1 则图像变化太突兀，如果设置为 0.001 则图像收敛太缓慢，设置为 0.01 刚刚好。

其次训练轮数设置为 3000 次，这是因为如果设置为 1000 次或者 2000 次，在拟合曲线的两端仍有较明显的不重合，在提高轮数到 3000 次后则比较重合，然而如果设置过高（如 10000 次），则一方面时间很长，另一方面所得的结果

也出现了过拟合的趋势。以下展示轮数为 1000 次和 10000 次时的相对不够完善的结果。





2、 exp2.py

2.1 补充代码

我们首先加载的 pytorch 自带 cifar 数据集：

```
print('==> Preparing data..')
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False)
```

数据预处理 torchvision.transforms 这一部分主要是进行数据的中心化(torchvision.transforms.CenterCrop)、随机剪切

(torchvision.transforms.RandomCrop)、正则化、图片变为 Tensor、tensor 变为图片等，这些是为了进行数据增强。

其中，数据增强(Data Augmentation)：是指对图片进行随机的旋转、翻转、裁剪、随机设置图片的亮度和对比度以及对数据进行标准化(数据的均值为 0，方差为 1)。通过这些操作，我们可以获得更多的图片样本，原来的一张图片可以变为多张图片，扩大了样本容量，对于提高模型的准确率和提升模型的泛化能力非常有帮助。

接下来，我们完善代码，其中 train()部分的代码已经很完善，我们参考其编写 test()部分的代码：

```
def test(epoch):
    print('==> Testing...')
    global best_acc
    model.eval()
    test_loss = 0
    correct = 0
    total = 0
    result[epoch] = []
    with torch.no_grad():
        ##### TODO: calc the test accuracy #####
        # Hint: You do not have to update model parameters.
        #       Just get the outputs and count the correct predictions.
        #       You can turn to `train` function for help.
        for batch_idx, (inputs, targets) in enumerate(testloader):
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
            print('Epoch [%d] Batch [%d/%d] Loss: %.3f | Testing Acc: %.3f%% (%d/%d)'
                  % (epoch, batch_idx + 1, len(testloader), test_loss / (batch_idx + 1),
                     100. * correct / total, correct, total))
            if batch_idx + 1 == len(testloader):
                result[epoch] = 100. * correct / total
        acc = 100. * correct / total
        ########################################
    # Save checkpoint.
    print('Test Acc: %f' % acc)
    print('Saving..')
    state = {
        'net': model.state_dict(),
```

```
        'acc': acc,
        'epoch': epoch,
    }
    if not os.path.isdir('checkpoint'):
        os.mkdir('checkpoint')
    torch.save(state, './checkpoint/ckpt_%d_acc_%f.pth' % (epoch, acc))
```

对于每一个 batch，我们统计它的 loss 以及 correct，并且累加到总的数据记录中，然后依据之前得到的数据计算 test 过程中的 acc，其公式为：

$$acc = 100. * correct / total$$

接着保存下每一个 epoch 之后得到的模型，以便在之后接着使用。

test 部分的代码已经补充完整，我们接下来要做的就是以 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。这一部分的代码很简单，我们不多赘述。

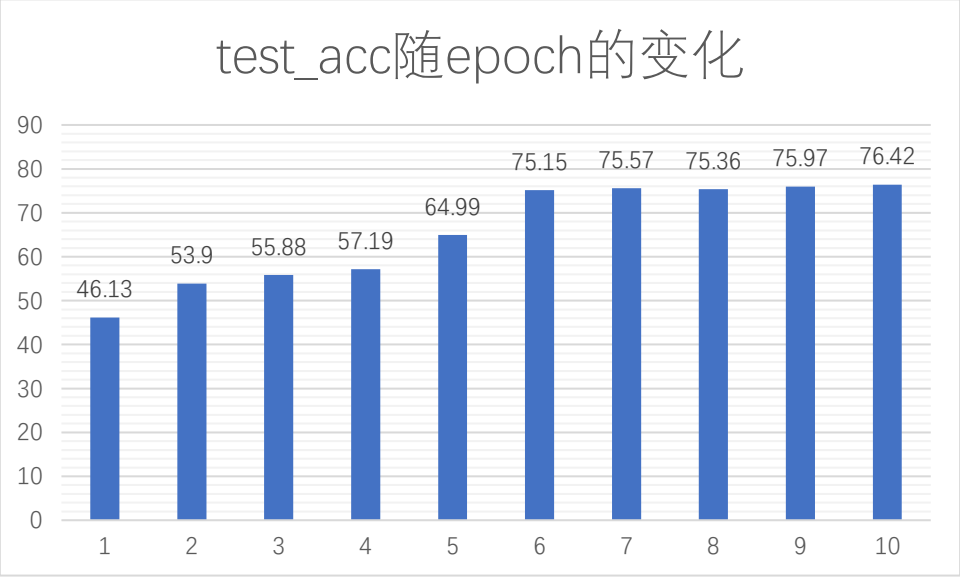
```
for epoch in range(start_epoch, end_epoch + 1):
    train(epoch)
    test(epoch)

optimizer = optim.SGD(model.parameters(), lr=0.01, weight_decay=5e-4)
for epoch in range(end_epoch+1, end_epoch + 6):
    train(epoch)
    test(epoch)

print result
```

2.2 test_acc 变化趋势

十个 epoch 中的 test_acc 变化趋势如图所示：



由图可见，随着 epoch 次数的增多，test_acc 的数值整体呈现上升趋势。并且在学习率为 0.1 时，test_acc 变化较大，波动较大。当学习率变为 0.01 后，test_acc 变化逐渐平稳，稳定在百分之七十几。

2.3 Train acc 和 Test acc 有什么关联和不同？在 lr 从 0.1 变到 0.01 后，acc 发生了什么变化？为什么？

在本实验中，总体上 Train acc 越高，模型匹配程度越高，因而 Test acc 越高。但是 Training set 和 Test set 的样本是完全不相交的。Training set 是用来训练我们的 model 的。Test set 是作为实际的数据来检验模型的，它是对模型在实际场景中的检验。两个数据集没有交集，Training acc 和 Test acc 并不是反映同一事物的。他们本质上反映的是在各自数据集中匹配的结果。

在 lr 为 0.1 时，acc 变化很快，当 lr 从 0.1 变为 0.01 后，acc 变化急剧变慢，acc 开始缓慢上升。这是因为学习率越大，输出误差对参数的影响就越大，参数更新的就越快，但同时受到异常数据的影响也就越大，很容易发散。因而在训练 epoch 次数增多后，我们选择将 lr 变小。

三、 实验总结

1、 实验概述

本次实验的主要任务，可以总结为 Pytorch 入门，并且了解深度学习和神经网络的概念，并且自己实现一个图片分类的实验，并观察实验的结果。

2、 感想总结

在这次的实验中，学会了的东西有很多，其中最重要的就是提高了自己处理问题、收集相关资料、解决问题的能力，这在我们将来的学习和工作生活中都是很重要的。而具体细化开来，在本次实验中：

2.1 了解了机器学习的概念

2.2 了解了神经网络和深度学习的概念

2.3 学会了 Pytorch 的初步使用，并且完成了一次图片分类

3、 创新点与难点

本次实验中，一大难点就是如何找到合适的 epoch 以及 lr，这一方面需要不断试错，另一方面也需要了解它们背后的原理，了解了它们之后仔细处理，得到合适的结果。