第十三周报告

1. 实验概览

本实验旨在于学习深度学习的原理（构建多层感知机通过权值迭代和非线性层实现复杂函数拟合），理解卷积神经网络的结构和其提取图像特征的过程。本实验中基于 PyTorch 进行模型训练，实现初等函数的拟合并对 CIFAR-10 进行图片分类。

2. 实验环境

Docker: sjtumic/ee208

3. 解决思路

a) 图像检索

- **构建图像数据库**

本实验基于预训练的 resnet50 模型对于图像进行 encode 操作

```
print('Load model: ResNet50')
#model = torch.hub.load('pytorch/vision', 'resnet50', pretrained=True)
model = torchvision.models.resnet50(pretrained=True)
```

从数据库读入图片后，首先对图片进行压缩裁剪等预处理操作

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
trans = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])
```

这一步中，将原本(512, 512, 3)的图片重整为(1, 3, 224, 224)

构建提取图片特征的网络结构

```python
def features(x):
    x = model.conv1(x)
    x = model.bn1(x)
    x = model.relu(x)
    x = model.maxpool(x)
    x = model.layer1(x)
    x = model.layer2(x)
    x = model.layer3(x)
    x = model.layer4(x)
    x = model.avgpool(x)

    return x
```

该模型调用了预设置的 resnet50 框架，其中利用 conv1, bn1, rel, maxpool 作为网络输入部分，四层卷积层对于图片进行卷积操作，最后调用平均池化层输出，得到图片的特征（对于 Resnet 的具体结构讨论见第五部分）

最后将每张图得到的 feature 存为.npy 文件，构建成数据库

```python
print('Save features!')
np.save('./lab13-CNN_features/features/' + imgPath + '.npy', image_feature)
```

- **图像检索**

读入图片后，同样利用训练好的 resnet50 网络作为 encode 网络对输入图片进行特征提取（函数代码同上）

```python
input = default_loader('./afhq/train/cat/flickr_cat_000002.jpg') # 输入图片，可以更改
input_image = trans(input)
input_image = torch.unsqueeze(input_image, 0)

print('Extract features!')
start = time.time()
image_feature = features(input_image)
image_feature = image_feature.detach().numpy()
```

将特征向量归一化，并与数据库中已有特征向量的归一化向量计算 L2 距离，距离越小则认为相似度越高，选择相似度最高的五个结果输出

```python
def get_top_scores(image_feature, thresh=0.0855, feature_path='./lab13-CNN_features/features'):
    image_feature = image_feature / (np.linalg.norm(image_feature))

    score = []
    final_name = []
    for feature in os.listdir(feature_path):
        name = feature[:-4]
        feature = os.path.join(feature_path, feature)
        feature = np.load(feature)
        feature = feature / (np.linalg.norm(feature))
        sub = feature - image_feature

        score_n = np.linalg.norm(sub)
        if score_n < thresh:
            score.sort(reverse=True)
            if len(score) < 5:
                score.append(score_n)
                final_name.append(name)
            else:
                for i in range(5):
                    if score_n < score[i]:
                        score[i] = score_n
                        final_name[i] = name
                        break

    return score, final_name
```

b) 函数拟合

基于 models 文件中给出的 Naive_NN 进行函数拟合，具体实验结果在第四部分详细分析

Naive_NN 架构：三层全连接层及 Sigmoid 函数作为非线性层

```python
class Naive_NN(torch.nn.Module):

    def __init__(self):
        super(Naive_NN, self).__init__()
        self.fc1 = torch.nn.Linear(1, 64)
        self.fc2 = torch.nn.Linear(64, 64)
        self.fc3 = torch.nn.Linear(64, 1)

    def forward(self, x):
        e1 = torch.sigmoid(self.fc1(x))
        e2 = torch.sigmoid(self.fc2(e1))
        return self.fc3(e2)
```

c) 图像分类

采用 resnet20 和 resnet50 进行训练（将以下两行分别注释进行训练）

```python
# Model
print('==> Building model..')
model = resnet20()
model = torchvision.models.resnet50(pretrained=True)
```

参考 train 函数对 test 函数进行补充计算 test acc

```
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.
    test_loss = 0
    correct = 0
    total = 0
    for _, (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()
    acc = 100. * correct / total
    ########################################
```

根据 epoch 数进行学习率衰减

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

for epoch in range(start_epoch, end_epoch + 2):
    lr = 0.01
    train(epoch + 5)
    test(epoch + 5)
```

4. 代码运行结果

a) 图像检索

输入:

检索结果

```
pixabay_dog_002610.jpg :  0.08891839
flickr_dog_000757.jpg :  0.08884598
pixabay_dog_000278.jpg :  0.08862213
pixabay_dog_000327.jpg :  0.08847078
pixabay_dog_000490.jpg :  0.086958006
```

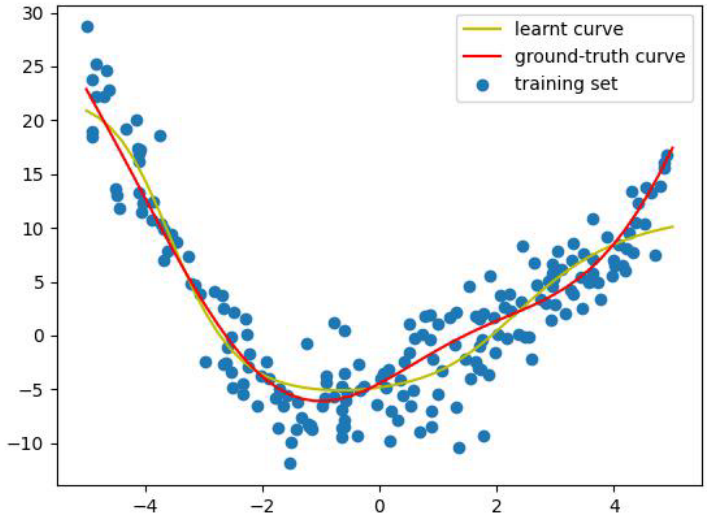
（分数越小说明图片差距越小）

分别对应图片

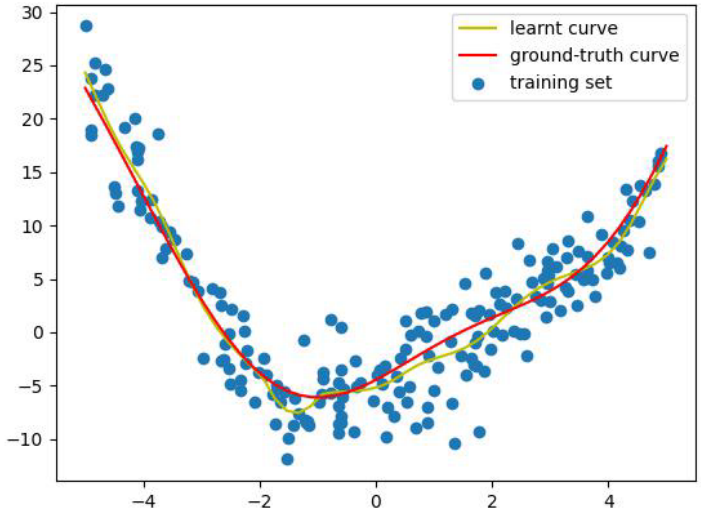
可见模型具有一定检索分类能力，但仍然存在一定分类不准的问题，可以通过进一步对模型训练调整参数获得更好的分类结果

b) 函数拟合

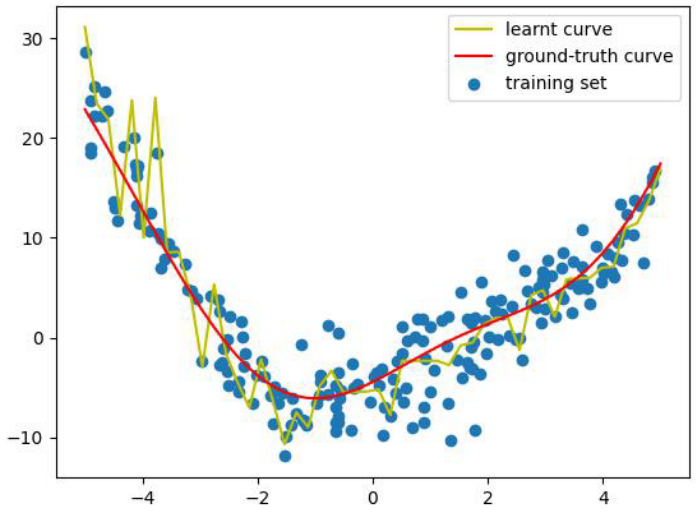
- 改变 training epoch
  Epoch=100



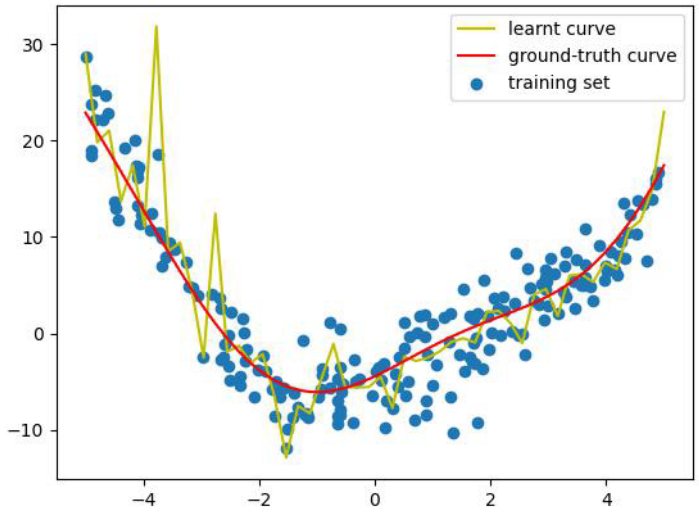
Epoch=1000



Epoch=10000

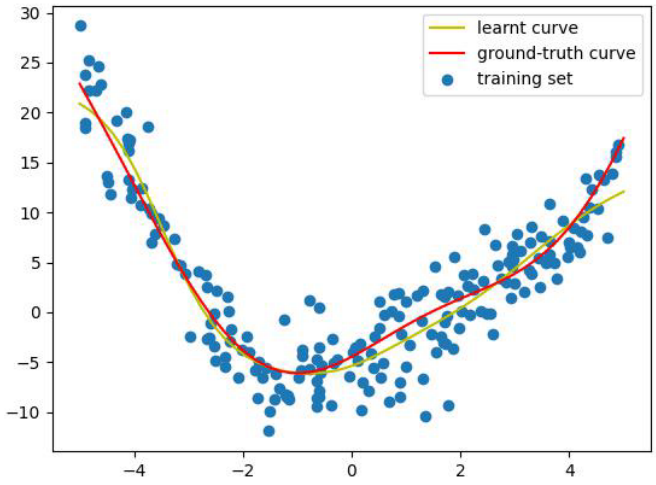


Epoch=50000

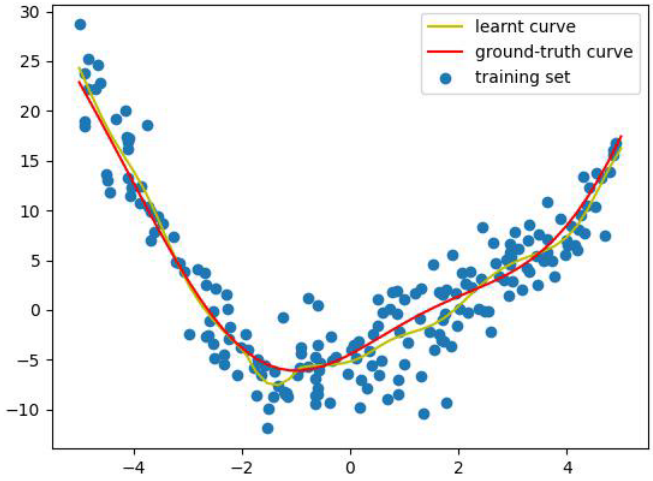


由上述图片可以发现，当训练 epoch 增加时，模型的拟合能力由欠拟合——正确拟合——过拟合变化。当 epoch 较小，模型对于函数的拟合能力欠佳；而当 epoch 过大，由于神经网络对于函数的表达能力较强，模型出现对于数据集过拟合的情况，出现过分波动和噪声过大的现象，故在实际训练中需要调整训练的 epoch 或增加其他特殊 layer 以取得更好的训练效果。

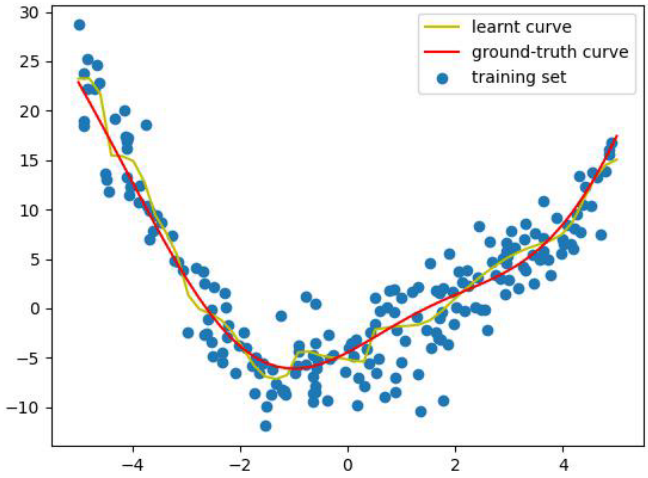
- 改变学习率
  lr = 0.001


learnt curve
ground-truth curve
training set
30
25
20
15
10
5
0
−5
−10
−4
−2
0
2
4

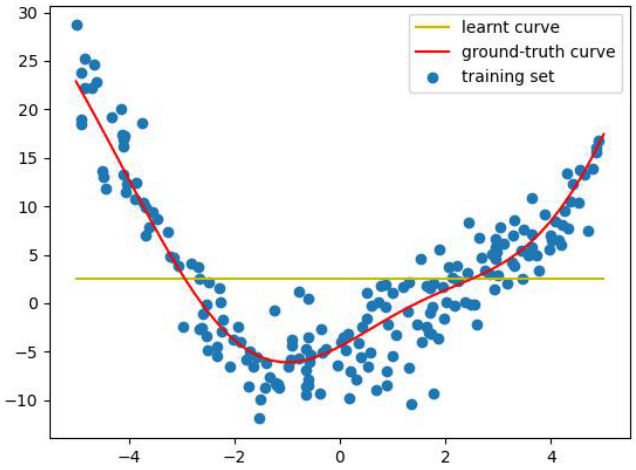
lr = 0.01


learnt curve
ground-truth curve
training set
30
25
20
15
10
5
0
−5
−10
−4
−2
0
2
4

lr = 0.1



lr = 1



由上图所示，当学习率过大时，易出现迭代不收敛的情况（根据凸优化知识可以知道，对于凸函数梯度下降收敛需要有收敛步长小于 L/2）；而当学习率较大（满足收敛条件）或较小时，易出现过拟合或收敛速度较慢的问题（如图中 lr=0.01 的时候有最好的效果）；故在实际应用中应多次实验获得最好的学习率

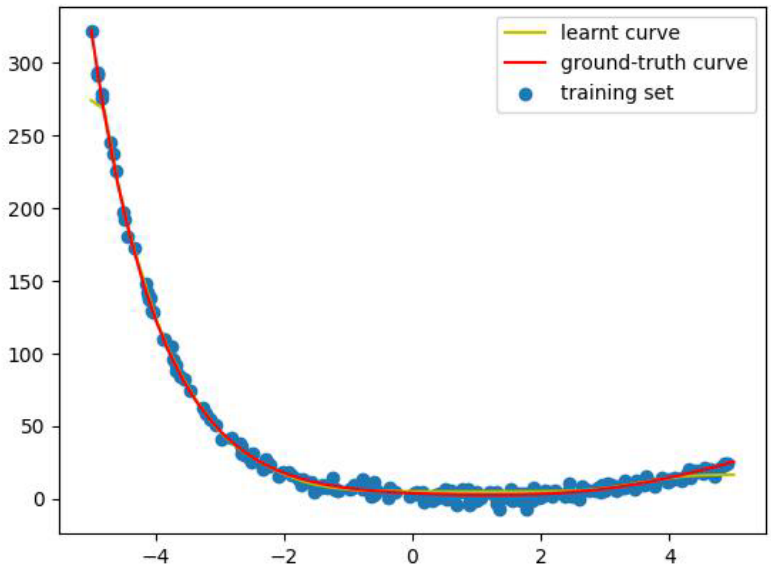
- 自定义函数拟合

取函数如下

```
def f2(x):
    """Actual function (ground truth)."""
    return (x**2) + 2 * np.cos(x) + 2 / np.exp(x)
```

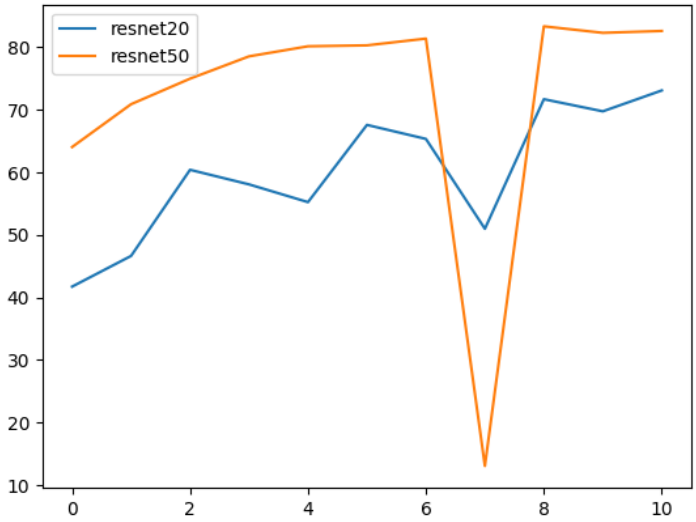
由上述实验可以得到，当 NUM_TRAIN_EPOCHS=1000 和 LEARNING_RATE=0.01 时具有较好的拟合效果，故在这一条件下进行训

练，结果图像为：



c) 图像分类

分别利用 resnet20 和 resnet50 训练 10 个 epoch 得到的 test acc 结果，得到的 checkpoint 在文件夹中附上



5. 分析与思考

a) Resnet 残差网络

在神经网络过深时，会出现梯度消失或梯度爆炸的问题（以梯度消失为例，反向传播过程中，每向前传播一层，都要乘以一个小于 1 的误差梯度，造成梯度消失），且对于恒等函数拟合不佳，残差网络能够有效解决上述问题，其具体提出的网络结构为：

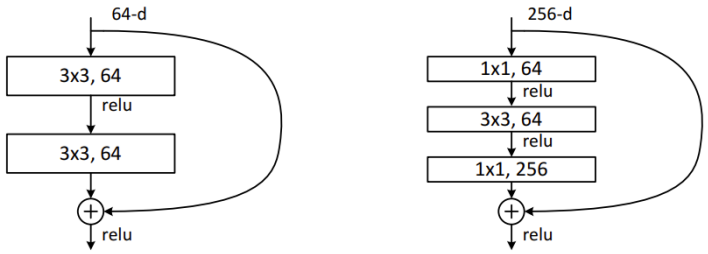


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

即在卷积层/全连接层之外混入运算前的 feature，使得网络能够构建较深同时不产生梯度消失或梯度爆炸问题

b) 解决过拟合问题

- 数据增广：对于已有的数据进行明度改变/色相改变/反转/裁剪等办法增加数据数据量
- 正则化：L1/L2 正则化，相当于对于损失函数中的某些参数增加惩罚项，防止权重过大造成过拟合，其中 L1 正则化产生稀疏解，L2 正则化产生平滑解
- Dropout：在训练过程中，按照一定的概率将神经网络单元暂时从网络中丢弃，有助于模型泛化