深蓝学院
shenlanxueyuan.com

点云作业第五讲——基于深度学习的点云分类

主讲人 陆一帆

数据集类构建

- 深度学习任务四部曲

构建数据集类
- 加载数据
- 实现__len__, __getitem__函数

构建网络模型
- 初始化网络层
- 实现forward函数

编写训练代码
- 初始化
- 训练

编写测试代码

数据集类构建

●数据集类构建

- 关键点：
  1. 继承官方的torch.utils.data.Dataset 类
  2. 特征对应标签
  3. 实现__len__, __getitem__类成员函数
  4. 数据增强

数据集类构建

深蓝学院 shenlanxueyuan.com

- 数据集类初始化

```
class PointNetDataset(Dataset):
  def __init__(self, root_dir, train):
    super(PointNetDataset, self).__init__()

    self._train = train
    self._classes = []

    self._features = []
    self._labels = []

    self.load(root_dir)
```

我们构造一个数据集类，继承官方的torch.utils.data.Dataset。

_train：加载数据集类型（训练、测试、验证）

_classes：所有类别的名称

_features：样本特征

_labels：样本标签

除了定义了一些私有成员外，我们还执行了load函数。load函数就是加载数据集中的数据（下一页）。

数据集类构建

深蓝学院 shenlanxueyuan.com

●数据加载

首先用self._train决定到底读取哪些数据。

```
def load(self, root_dir):
    things = os.listdir(root_dir)
    files = []
    for f in things:
        if self._train == 0:
            if f == 'modelnet40_train.txt':
                files = read_file_names_from_file(root_dir + '/' + f)
        elif self._train == 1:
            if f == 'modelnet40_test.txt':
                files = read_file_names_from_file(root_dir + '/' + f)
        elif self._train == 2:
            if f == 'modelnet40_test.txt':
                files = read_file_names_from_file(root_dir + '/' + f)
        if f == "modelnet40_shape_names.txt":
            self._classes = read_file_names_from_file(root_dir + '/' + f)
```

接着，把样本的特征和标签存到成员变量中：

```
for file in files:
    num = file.split("_")[-1]
    kind = file.split("_" + num)[0]
    if kind not in tmp_classes:
        tmp_classes.append(kind)
    else:
        if self._train == 2:
            continue
    pcd_file = root_dir + '/' + kind + '/' + file + '.txt'
    np_pts = read_pcd_from_file(pcd_file)
    # print(np_pts.shape) # (10000, 3)
    self._features.append(np_pts)
    self._labels.append(kind)
```

数据集类构建

深蓝学院 shenlanxueyuan.com

- __getitem__实现

数据加载完毕后，所有东西就都在self._features和self._labels中了。这时候，我们去实现__getitem__函数。该函数传入一个index，输出对应的特征和标签。

```
def __getitem__(self, idx):
    feature, label = self._features[idx], self._labels[idx]
```

得到feature后，做数据增强，归一化、旋转、高斯噪声，与课堂上讲解的一致：

```
# normalize
center = np.expand_dims(np.mean(feature, axis = 0), 0)
feature = feature - center
dist = np.max(np.sqrt(np.sum(feature ** 2, axis = 1)),0)
feature = feature / dist #scale
# rotation
theta = np.random.uniform(0, np.pi*2)
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
    [np.sin(theta), np.cos(theta)]])
feature[:,[0,2]] = feature[:,[0,2]].dot(rotation_matrix)
# jitter
feature += np.random.normal(0, 0.02, size=feature.shape)
```

数据集类构建

深蓝学院
shenlanxueyuan.com

- __getitem__实现

label需要从数字变成一个one hot的向量

```
l_lable = [0 for _ in range(len(self._classes))]
l_lable[self._classes.index(label)] = 1
```

最后把他们转化为torch中的tensor向量并输出：

```
feature = torch.Tensor(feature.T)
label = torch.Tensor(l_lable)

return feature, label
```

数据集类构建

- __len__实现

超简单，就是样本数量

```
def __len__(self):
    return len(self._features)
```

- 总结

Dataset类中，__len__和__getitem__这两个函数必须实现，而且名字和传参也不能改（因为是override的基类的成员函数），其他内容都是自己定义的，想咋写咋写，只要这两个函数的功能正常实现了，就可以work。

网络模型类构建

深蓝学院
shenlanxueyuan.com

- 网络模型类构建

  - 关键点：
    1. 继承官方的nn.Module 类
    2. 网络层实现
    3. forward函数实现

网络模型类构建

深蓝学院 shenlanxueyuan.com

- 网络层实现

```
class PointNet(nn.Module):
  def __init__(self):
    super(PointNet, self).__init__()
    self.conv1 = nn.Conv1d(3, 64, 1)
    self.conv2 = nn.Conv1d(64, 128, 1)
    self.conv3 = nn.Conv1d(128, 1024, 1)
    self.fc1 = nn.Linear(1024, 512)
    self.fc2 = nn.Linear(512, 256)
    self.fc3 = nn.Linear(256, 40)

    self.bn1 = nn.BatchNorm1d(64)
    self.bn2 = nn.BatchNorm1d(128)
    self.bn3 = nn.BatchNorm1d(1024)
    self.bn4 = nn.BatchNorm1d(512)
    self.bn5 = nn.BatchNorm1d(256)

    self.relu = nn.ReLU(inplace=True)
    self.dropout = nn.Dropout(p=0.3)
```

Init函数里主要定义需要用到的网络层的类型和尺寸。Conv1d、BatchNormal、Linear、Relu都是课上老师给的开源代码中用到的常用网络层。Dropout层是我自己加的，也是基础网络层。

对这些基础网络层不熟悉的同学可以去看torch官网给出的60min入门教程：
https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

网络模型类构建

深蓝学院 shenlanxueyuan.com

- 网络层实现

```
def forward(self, x):
    x = self.relu(self.bn1(self.conv1(x)))
    x = self.relu(self.bn2(self.conv2(x)))
    x = self.relu(self.bn3(self.conv3(x)))
    x = torch.max(x, 2, keepdim=True)[0]
    # print(x.shape)
    x = x.view(-1, 1024)

    x = self.relu(self.bn4(self.dropout(self.fc1(x))))
    x = self.relu(self.bn5(self.dropout(self.fc2(x))))
    x = self.fc3(x)

    return x
```

forward函数负责将输入的feature经过一系列网络层的处理后输出。

这里输入的feature就是x。函数的第一行表示将输入的x，经过conv1，bn1，relu三个层的处理后再赋值给x自己。前三行的操作就是将每个点(一共N个点)从xyz三个维度的信息(N*3)，变成1024维信息(N*1024)。第四行，取每一列的最大值，就是求global feature(1*1024)。最后，将global feature通过几个全连接层得到预测结果(1*40)。

注意！返回的预测结果还需要经过softmax操作才能用于计算loss，参考右侧函数的实现。这个函数实现在模型训练代码中，在模型搭建部分不需要。

```
def softXEnt(input, target):
    logprobs = torch.nn.functional.log_softmax(input, dim=1)
    return -(target * logprobs).sum() / input.shape[0]
```

训练代码编写

深蓝学院 shenlanxueyuan.com

- 超参数设置

```
SEED = 13
batch_size = 32
epochs = 10000
decay_lr_factor = 0.95
decay_lr_every = 2
lr = 0.01
gpus = [0]
show_every = 1
val_every = 3
```

模型搭建好，数据集处理好，接下来开始训练。**训练第一步**，设置训练中的超参数。

其中，SEED是生成随机数的种子，seed设定之后，每一次跑代码生成的随机数是固定的。

batch_size是一次训练使用的样本数量。batch size的设定一般取8，16，32，64，128等。batch size怎么设置参考这个回答：
https://www.zhihu.com/question/61607442/answer/204525634

epoch是遍历dataset的次数，不一定要跑完，模型准确率不涨了就可以提前结束；gpus是gpu序号；

show every是每隔2次训练，打印一次loss和accuarcy；

val every是每隔3次epoch，计算一次模型在验证集的表现，并决定是否要保存当前参数。

lr，decay_lr_factor，decay_lr_every分别表示学习率，学习率衰减因子以及每隔2次epoch衰减一次学习率

训练代码编写

深蓝学院 shenlanxueyuan.com

●数据集加载

主函数刚开始时，设置随机数种子和训练所用的机器(CPU或GPU)。

```
if __name__ == "__main__":
    torch.manual_seed(SEED)
    device = torch.device(f'cuda:{gpus[0]}' if torch.cuda.is_available() else 'cpu')
```

接着实例化数据集的类，并加载到DataLoader中。
DataLoader是torch官方的工具，用于从数据集类中抽取数据。Shuffle=true表示将原始数据打乱后再取出。

```
print("Loading train dataset...")
train_data = PointNetDataset("../../../dataset/modelnet40_normal_resampled", train=0)
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
print("Loading valid dataset...")
val_data = PointNetDataset("../../../dataset/modelnet40_normal_resampled/", train=1)
val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=True)
```

训练代码编写

●初始化

接下来是一系列的初始化。
初始化模型，to(device)表示将模型加载到GPU(CPU)中。

```
model = PointNet().to(device=device)
```

初始化优化器，这里用的Adam优化器(还有其他优化器可以选择，Adam的性能相对来说很好)。优化器初始化的时候需要传入模型的参数(因为优化器的工作就是根据梯度更新这些参数嘛)，和学习率。

```
optimizer = optim.Adam(model.parameters(), lr=lr)
```

初始化scheduler，用于训练过程中自动完成学习率的衰减。

```
scheduler = optim.lr_scheduler.StepLR(
        optimizer, step_size=decay_lr_every, gamma=decay_lr_factor)
```

为了在训练过程中了解训练情况，我们往往要绘制loss，acc曲线，这里采用tensorborad进行数据可视化。初始化如下，传入的参数为程序运行过程中tensorboard产生的可视化数据输出的地址。

```
writer = SummaryWriter('../output/runs/tersorboard')
```

训练代码编写

深蓝学院 shenlanxueyuan.com

●开始训练

训练前需要将模型设置为train模式，网络将启用BatchNormalization和 Dropout。

```
model.train()
```

接下来开始训练：

```
for epoch in range(epochs):
  acc_loss = 0.0
  num_samples = 0
  start_tic = time.time()
  for x, y in train_loader:
    x = x.to(device)
    y = y.to(device)
    optimizer.zero_grad()
    out = model(x)
    loss = softXEnt(out, y)
    # print('acc: ', acc)
    loss.backward()
    optimizer.step()
```

每个epoch都会遍历一次dataset中的样本(从train loader中不断取出数据(x)和标签(y)，直到取完一遍为止)。

在进行模型的inference之前，一定要执行optimizer.zero_grad()，将网络中每个参数的梯度清零，这样更新参数的时候才是朝着正确的方向更新。

x经过模型inference之后得到预测结果(out)，预测结果和标签(y)用于计算loss。

得到loss后，执行backward后向传播，这个过程就把网络中每个参数对于cost的梯度算出来了。

最后执行optimzer.step，优化器就会按照梯度方向更新网络的参数了。至此，一步训练结束。

训练代码编写

- 准确率计算&学习率衰减

每次训练结束后，别忘了计算一下准确率(acc)，并用tensorboard可视化出来：

```
acc = np.sum(np.argmax(out.cpu().detach().numpy(), axis=1) ==
    np.argmax(y.cpu().detach().numpy(), axis=1)) / len(y)      You, secon
acc_loss += batch_size * loss.item()
num_samples += y.shape[0]
global_step += 1
if (global_step + 1) % show_every == 0:
  # ...log the running loss
  writer.add_scalar('training loss', acc_loss / num_samples, global_step)
  writer.add_scalar('training acc', acc, global_step)
```

最后每次epoch结束后，衰减一下lr：

```
scheduler.step()
```

训练代码编写

深蓝学院 shenlanxueyuan.com

- 模型验证&参数保存

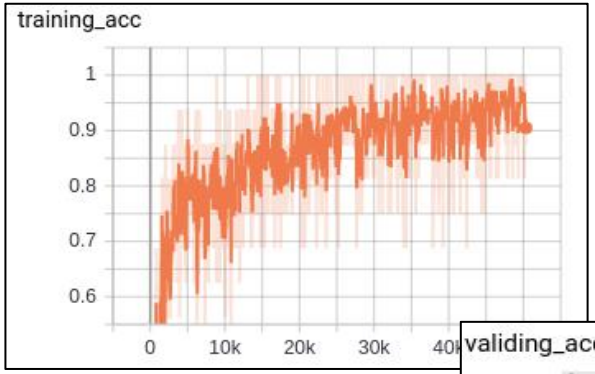
训练的过程中，我们要及时保存当前最好的网络参数，这样过拟合后，我们可以及时停止，选择最好的一次模型参数作为最终结果。因此，每隔3次epoch，我们计算一下当前参数在验证集上的表现，如果比之前好，就保存下来。顺便把val acc可视化出来。

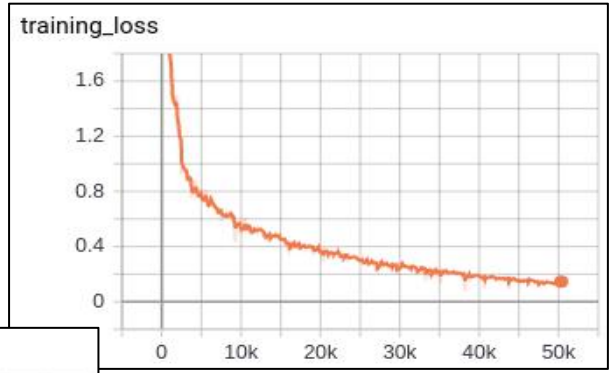
```
if (epoch + 1) % val_every == 0:
    acc = get_eval_acc_results(model, val_loader, device)
    print("eval at epoch[" + str(epoch) + f"] acc[{acc:3f}]")
    writer.add_scalar('valiging acc', acc, global_step)
    if acc > best_acc:
        best_acc = acc
        save_ckp(save_dir, model, optimizer, epoch, best_acc, date)
```

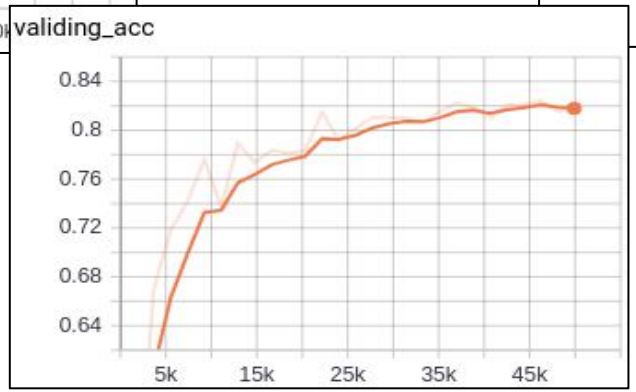
训练代码编写

深蓝学院 shenlanxueyuan.com

- 训练结果







Tensorboard启动方式：
> cd ${代码根目录}
> tensorboard --logdir output/runs/tersorboard
> 打开浏览器，输入网址：http://localhost:6006/

测试代码编写

深蓝学院 shenlanxueyuan.com

●初始化

训练结束后，我们需要在测试集上测试模型的性能，因此需要编写测试代码。

测试代码非常简单，大部分内容和训练代码一样。比如最开始的随机数种子设置，CPU或GPU设置，测试数据加载以及模型加载，和训练代码一样 。

```
if __name__ == "__main__":
    torch.manual_seed(SEED)        You, a week ago • update
    device = torch.device(f'cuda:{gpus[0]}' if torch.cuda.is_available() else 'cpu')
    print("Loading test dataset...")
    test_data = PointNetDataset("../../../dataset/modelnet40_normal_resampled", train=1)
    test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
    model = PointNet().to(device=device)
```

测试代码编写

深蓝学院
shenlanxueyuan.com

- 模型加载

既然是要测试训练好的模型，就需要将之前训练得到的参数结果加载进来。

```
if ckp_path:
    load_ckp(ckp_path, model)
    model = model.to(device)
```

模型加载完之后，调整为eval模式。在这个模式下，网络将不启用BatchNormalization和 Dropout。

```
model.eval()
```

测试代码编写

深蓝学院 shenlanxueyuan.com

●测试过程

接下来就是读取feature，进行网络推理，并将推理结果和label比较，看是否一致。

```
with torch.no_grad():
  accs = []
  for x, y in test_loader:
    x = x.to(device)
    y = y.to(device)
    out = model(x)

    pred_y = np.argmax(out.cpu().numpy(), axis=1)
    gt = np.argmax(y.cpu().numpy(), axis=1)
    print("pred[" + str(pred_y)+"] gt[" + str(gt) + "]")

    acc = np.sum(pred_y == gt) / len(pred_y)
    accs.append(acc)

  print("final acc is: " + str(np.mean(accs)))
```

需要注意的是，在训练阶段，不需要进行计算图的生成（因为不需要记录梯度来更新网络参数）。而对于tensor的计算操作，默认是进行计算图的构建的，在这种情况下，可以使用 with torch.no_grad():，强制之后的内容不进行计算图构建。这会加速网络的inference。

应用代码编写

深蓝学院 shenlanxueyuan.com

- 应用例子

测试结束后，我们可以拿我们的模型做一些小开发，比如可视化点云的同时给出模型的预测结果。这一部分代码和test非常像，只有两个地方要修改。

一个是读取dataset的时候，batch_size设置为1

```
batch_size = 1
```

```
app_data = PointNetDataset("../../../dataset/modelnet40_normal_resampled", train=2)
app_loader = DataLoader(app_data, batch_size=batch_size, shuffle=True)
```

第二个是，每做一次inference，把点云画出来(第一节课内容)，再打印出预测结果对应的类别。

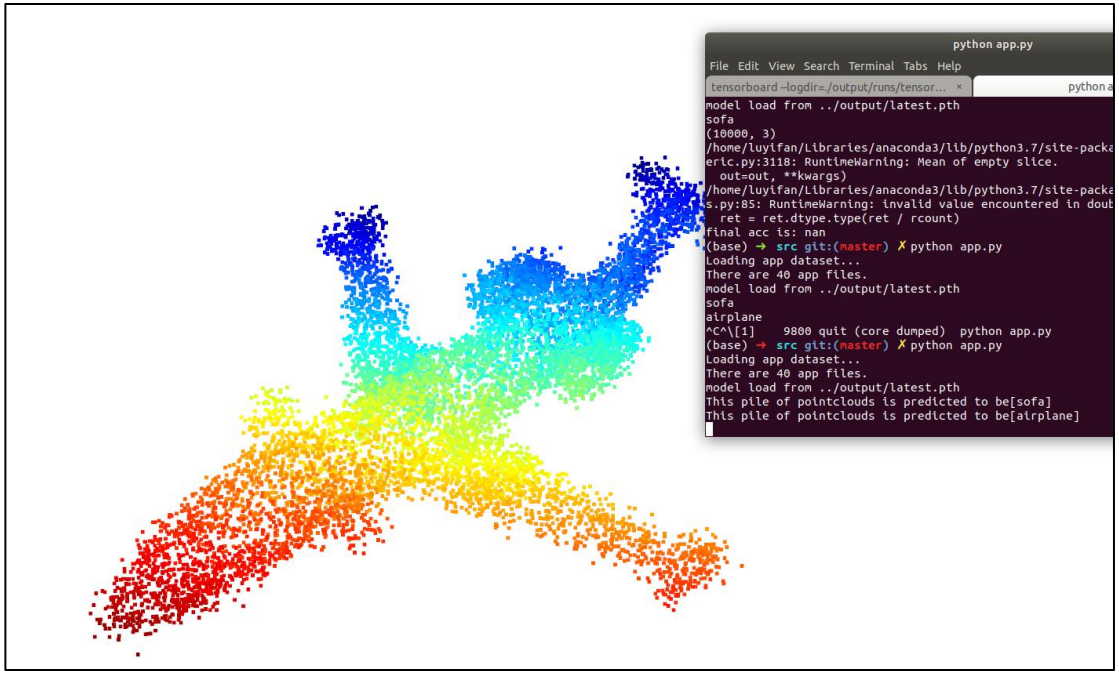
```
pred_y = np.argmax(out.cpu().numpy(), axis=1)
gt = np.argmax(y.cpu().numpy(), axis=1)
print("This pile of pointclouds is predicted to be[" + app_data.classes()[gt[0]] + "]")

pts = x.cpu().numpy()[0].T
show_np_pts3d(pts)
```

应用代码编写

深蓝学院
shenlanxueyuan.com

- 应用效果



在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening