《机器学习编程实践》课程——DAY 2

课程内容

从经典 Resnet 案例出发，学习：

- 数据的预处理和加载
- 模型架构的构建
- 整体训练框架的实现

一、 命令行参数接收

```
# 接收命令行输入
parser = argparse.ArgumentParser(description='PyTorch CIFAR10 Training')

# 分别设置各个参数，help使得论文可读性更高
parser.add_argument('--lr', default=0.1, type=float, help='learning rate')
parser.add_argument('--resume', '-r', action='store_true',
help='resume from checkpoint') # bool型数据的默认值为false

args = parser.parse_args() #实例化
```

二、 预处理和加载数据

1 数据预处理

```
  # 调用transforms.Compose库
  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(), # ToTensor操作完成了[0,1]的归一化
  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
  ])
```

2 加载数据

2.1 直接调用库

```
    # dataset通常和dataloader一起用
    #
        训练集，使用torchversion.datasets库,root指文件要保存的路径,transform指数据预处理操作
    trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform_train)
    trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=128, shuffle=True, num_workers=0)

    # 测试集
    testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform_test)
    testloader = torch.utils.data.DataLoader(
    testset, batch_size=100, shuffle=False, num_workers=0)
```

2.2 不用库的数据加载

```
# 构造一个类
class own_dataset(Dataset):
  # 初始化函数
  def __init__(self, root, preprocess):
  super(own_dataset, self).__init__()
  self.preprocess = preprocess
  self.image_paths = []
  self.labels = []

  # 找到root目录下的所有子目录（root为数据下载路径）
  label_list = os.listdir(root)
  for label in label_list:
    image_folder = os.path.join(root, label) # 得到每个子文件夹的绝对路径
```

```
      for file in os.listdir(image_folder): # 遍历每个文件
        if file.endswith(("png", "jpg", "gif")): #
           .endswith表示得到文件的后缀
          self.image_paths.append(os.path.join(image_folder, file)) #
             得到每个文件的绝对路径，加入列表image_paths中
          self.labels.append(label_list.index(label))

    # 得到数据总数
    def __len__(self):
      return len(self.image_paths)

    def __getitem__(self, item):
      image = Image.open(self.image_paths[item]) # 打开第item个图像数据
      image = self.preprocess(image) # 对图像进行预处理
      label = self.labels[item]
      return image, label

# 实例化
trainset = own_dataset(root="./data", preprocess=transform_train)
```

三、 模型架构的构建

```
# 调用ResNet18函数得到ResNet18网络，定义为net
net = ResNet18()

# ResNet18函数，实例化ResNet类
# 其中ResNet函数提供了模型构造的基本框架，通过不同参数构建不同模型
def ResNet18():
  return ResNet(BasicBlock, [2, 2, 2, 2])

# ResNet类:
class ResNet(nn.Module):
  # 初始化函数，block表示不同块（主要有Bottleneck和BasicBlock）
  # num_blocks表示块的数量
  def __init__(self, block, num_blocks, num_classes=10):
    super(ResNet, self).__init__()
    self.in_planes = 64
  self.conv1 = nn.Conv2d(3, 64, kernel_size=3,
  stride=1, padding=1, bias=False)
  self.bn1 = nn.BatchNorm2d(64)
  self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
  self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
  self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
  self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
  self.linear = nn.Linear(512 * block.expansion, num_classes)

  def _make_layer(self, block, planes, num_blocks, stride):
  strides = [stride] + [1] * (num_blocks - 1)
  layers = []
  for stride in strides:
  layers.append(block(self.in_planes, planes, stride))
  self.in_planes = planes * block.expansion
  return nn.Sequential(*layers)

  def forward(self, x):
  out = F.relu(self.bn1(self.conv1(x)))
  # 通过不同层
  out = self.layer1(out)
  out = self.layer2(out)
  out = self.layer3(out)
  out = self.layer4(out)
  # 通过最后的注意力层，得到最终输出
  out = F.avg_pool2d(out, 4)
  out = out.view(out.size(0), -1)
  out = self.linear(out)
  return out
```

四、　整体训练框架的实现

1　异常中断保存训练信息

```
# 下次训练可以直接从异常中断点继续训练
if args.resume:
    print('==> Resuming from checkpoint..')
    assert os.path.isdir('checkpoint'), 'Error: no checkpoint directory
```

```
        found!'
    checkpoint = torch.load('./checkpoint/ckpt.pth')
    net.load_state_dict(checkpoint['net'])
    # weight = net.state_dict()
    # torch.save(weight, "/your/path")
    # weight = torch.load("/your/path")
    # net.load_state_dict(weight)
    best_acc = checkpoint['acc']
    start_epoch = checkpoint['epoch']
```

2 优化器和调度器

```
criterion = nn.CrossEntropyLoss() # 损失

# 优化器，优化目标为net.parameters()，即网络中的参数
optimizer = optim.SGD(net.parameters(), lr=args.lr,
momentum=0.9, weight_decay=5e-4)

scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)
```

3 训练

```
# 训练函数
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train() # train表明可以进行梯度回传
    train_loss = 0
    correct = 0
    total = 0

    # 每次从trainloader中加载一个batch_size的数据对
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad() # 清空梯度
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward() # 后向传播
        optimizer.step() # 更新
        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

        # 进度条，可以直接用tqdm库实现
        progress_bar(batch_idx, len(trainloader), 'Loss: %.3f | Acc: %.3f%%
            (%d/%d)'% (train_loss / (batch_idx + 1), 100. * correct / total,
            correct, total))
```

4 测试

```
def test(epoch):
    global best_acc
    net.eval() # 测试阶段使用eval
    test_loss = 0
    correct = 0
    total = 0

    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(testloader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

            progress_bar(batch_idx, len(testloader), 'Loss: %.3f | Acc: %.3f%%
                (%d/%d)'% (test_loss / (batch_idx + 1), 100. * correct / total,
                correct, total))

    # Save checkpoint.
    acc = 100. * correct / total
    if acc > best_acc:
        print('Saving..')
        state = {
            'net': net.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state, './checkpoint/ckpt.pth')
        best_acc = acc
```

5 最终调用

```
for epoch in range(start_epoch, start_epoch + 200):
    train(epoch)
    test(epoch)
    scheduler.step()
```