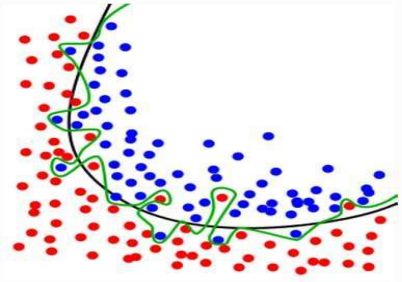
媒体与认知

Media and Cognition



V 1.0 2022.3.3

清华大学电子工程系

薛有泽 助教

Email: xueyz19@mails.tsinghua.edu.cn

第一次作业介绍

- 理论部分：
  - 选择题：5道，涉及基础的pytorch，numpy编程知识
  - 计算题：2道，涉及单个神经元的线性层设计
- 编程部分：
  - 完成线性分类器的代码
  - 训练、测试、可视化线性分类器
  - 撰写作业报告：鼓励同学们使用作业文档的LaTeX模板完成报告
  - 自选课题报告（选择自选课题替代编程作业的同学）

第一次作业介绍

➢ 编程部分：

◆ 完成线性分类器的代码

序号	内容	说明
0	导入模型依赖库	导入 numpy，torch，matplotlib，sys 等必要的依赖库，为保证其余部分代码正确运行，请不要删除该部分的代码，若有必要，可以增加需要导入的库。
1	定义数据类模块	完成数据类的定义。
2	定义模型结构	完成模型类的定义。请不要使用 torch.nn.Linear 类。
3	定义损失函数	完成二元交叉熵损失函数的定义。请不要使用 torch.nn.BCELoss 类。
4	训练-验证代码	完成模型的训练和验证代码。
5	测试代码	模型测试代码。请不要修改或删除该部分代码。
6	结果可视化	损失曲线的绘制代码及分类结果展示。请不要修改或删除该部分代码。
7	运行入口	主程序入口。请不要修改或删除该部分代码。

需要按照要求完成线性分类器的代码，主要包括数据类的定义、模型类的定义、损失函数的定义、模型的训练和验证代码。

注意：

在模型类的定义中，要求同学们不使用nn.Linear类实现线性分类的功能；

在损失函数的定义中，要求同学们不使用nn.BCELoss类实现二元交叉熵。

第一次作业介绍

➢ 编程部分：

- ◆ 训练、测试、可视化

Windows 平台	打开 Anaconda Prompt，用 cd 命令进入作业所附程序解压生成的子目录 hw1。若要换盘符，需用“< 盘符 >:” 命令，比如从 c 盘换至 d 盘，键入 d: 和回车即可。分别运行: python classification.py train python classification.py test python classification.py visual
Linux/Mac 平台	在命令行终端中，用 cd 命令进入作业所附程序解压生成的子目录 hw1。运行: python classification.py train python classification.py test python classification.py visual

需要按照要求运行上一个任务中完成的代码，实现线性分类器的训练、测试和可视化。

要求在作业报告中附上训练过程中的损失函数曲线，以及分类界面的可视化效果图。

第一次作业的考察目标

- 熟悉PyTorch编程
  - PyTorch张量(Tensor)
- 理解神经网络模型训练步骤
  - 训练网络都需要哪些函数
  - 如何计算梯度、更新模型参数
  - 训练和测试有什么区别
- 初探构建神经网络代码的细节
  - 线性层（或全连接层）是怎么写的
  - loss函数是怎么写的

PyTorch基础操作

- PyTorch是一个开源的Python机器学习库，基于Torch深度学习工具包，底层由C++实现
- PyTorch是使用GPU和CPU优化的深度学习张量库，类似于python的科学计算库—numpy
- PyTorch的特点：易于使用的API，python的支持，动态计算图

- Tensor（张量）是pytorch的基础数据类型，pytorch对于Tensor的操作很多和numpy对于array的操作类似。

导入相关函数库

```
import numpy as np
import torch
# 导入 pytorch 内置的 mnist 数据
from torchvision.datasets import mnist
#导入预处理模块
import torchvision
import torchvision.transforms as transforms
#导入nn及优化器
import torch.nn.functional as F
import torch.optim as optim
from torch import nn
# 导入数据加载器
from torch.utils.data import DataLoader
```

PyTorch张量基础操作

➢ 张量的创建有四种基本方式

1. 从数据直接创建：

```
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
```

2. 从numpy数组创建：

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

3. 仿照已有张量创建：

```
x_ones = torch.ones_like(x_data)
x_rand = torch.rand_like(x_data)
```

4. 创建特定形式的张量：

```
shape = (2, 3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)
```

PyTorch张量基础操作

- 张量属性：包括尺寸、数据类型、所在设备等

```
tensor = torch.rand(3, 4)
print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
```

```
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

- PyTorch 张量/Numpy数组转化：

```
t = torch.ones(5)
n = t.numpy()

n = np.ones(5)
t = torch.from_numpy(n)
```

PyTorch张量基础操作

- 张量操作：张量有上百种可以执行的操作，最基本的包括属性调整、数学运算、拼接等。
  - 属性调整：

尺寸调整

```
t = torch.ones((2,3,4))
t = t.view(6, 4)
t = t.reshape((2, 12))
```

数据类型

```
t = t.long()
t = t.type(torch.float)
```

计算所用设备

```
t = t.to('cuda')
t = t.cuda()
t = t.cpu()
```

  - 数学运算：

```
a = torch.rand(3, 4)
aT = a.transpose(0, 1)
b = torch.rand(3, 4)
c = a + b
d = a - b
e = torch.matmul(aT, b)
f = a*b
```

  - 拼接：

```
a = torch.rand(3, 4)
b = torch.rand(3, 4)
c = torch.cat([a, b], 0)
d = torch.stack([a, b], 0)
```

用PyTorch搭建神经网络

- 搭建网络需要定义若干函数，不同的网络也只是在函数细节中有差别，在整体架构上差别不大。
  - 数据集相关函数
  - 模型构建相关函数
  - 训练相关函数
  - 验证/测试相关函数
  - 主函数

数据集相关函数

➢ 自定义数据集(Dataset)

- 用于控制数据读入的过程
- 先定义类的初始化函数
- 再定义基本数据处理函数

```python
class MyDataset(Dataset):
    def __init__(self, file_path):
        '''
        :param file_path: npy file containing the features and labels of data
        '''
        data = np.load(file_path)
        self.feat = data[:, :2].astype(np.float32)
        self.labels = data[:, 2].astype(np.long)

    def __len__(self):
        return self.feat.shape[0]

    def __getitem__(self, index):
        assert index <= len(self), 'index range error'
        feat = self.feat[index]
        label = self.labels[index]
        return feat, label
```

本示例中，npy 文件中的数据尺寸为(N, 3), 总共有 N 组数据，每组数据前两个元素为特征数据，最后一个元素为类别标签

➢ 数据加载器(DataLoader)

- 用于在训练过程中控制批量送入数据
- 先初始化数据加载器
- 训练过程每次迭代时送入一批数据

```python
dataset = MyDataset(file_path)
trainloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```python
for step, (feats, labels) in enumerate(trainloader):  # get a batch of data
```

模型构建

- 定义模型 (model)
  - 继承**nn.Module**
  - 定义模型中的变量

- 前向计算 (forward)
  - 模型相当于一个函数， 对输入数据进行处理， 得到输出结果
  - **pytorch**自带的 **autograd**不需要用户编 写误差反向传播 **backward** 的代码

```python
class Model(nn.Module):
    def __init__(self, input_size, output_size):
        '''
        :param input_size: dimension of input features
        :param output_size: number of classes, 1 for classification task of two class categories
        '''
        super(Model, self).__init__()
        # the linear layer
        self.linear = nn.Linear(input_size, output_size)
        # the sigmoid activation function
        self.act = nn.Sigmoid()

    def forward(self, x):
        '''
        Linear classifier forward
        --------------------------
        :param x: input features with size [batch_size, feature_dimension]
        :return: probabilities of characters for each input image, with size [batch_size]
        '''
        # forward: y = sigmoid(xW^T + b)
        # use the linear layer to perform xW^T + b, and use self.act() for activation
        out = self.act(self.linear(x))
        # as "out" is the output of the shape [batch_size, output_size] and output_size is 1
        # the matrix form of out [batch_size, 1] should be squeezed into a vector for final output
        return out.squeeze(1)  # [batch_size, 1] ==> [batch_size]
model = Model(input_size=2, output_size=1)
pred = model(feats)
```

模型训练

- 定义损失函数 (loss function)
  - 计算output和label的差异
  - loss通常是一个batch的均值

```python
def bce_loss(pred, label):
    '''
    Binary cross entropy loss function
    ----------------------------
    :param pred: predictions with size [batch_size, *], * refers to the dimension of data
    :param label: labels with size [batch_size, *]
    :return: loss value, divided by the number of elements in the output
    '''
    b_loss = nn.BCELoss()
    loss = b_loss(pred, label)
    return loss
```

- 初始化各模块
  - 装载数据集
  - 初始化模型
  - 初始化优化器

```python
# training and validation data loader
trainset = MyDataset(train_file_path)
valset = MyDataset(val_file_path)
trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
valloader = DataLoader(valset, batch_size=batch_size, shuffle=False)

# initialize the model
model = Model(input_size=2, output_size=1)
model = model.to(device)

# optimizer
optimizer = optim.SGD(model.parameters(), lr, momentum)
```

模型训练

➢ 训练过程

- 设置训练模式
- 装载数据
- 优化器清空原有梯度
- 前向计算
- 计算loss
- 误差反向传播
- 更新模型参数

```
for epoch in range(n_epochs):
    # set the model in training mode
    model.train()

    # to save total loss in one epoch
    total_loss = 0.
    for step, (feats, labels) in enumerate(trainloader):  # get a batch of data

        # set data type and device
        feats, labels = feats.to(device), labels.type(torch.float).to(device)

        # call a function to clear gradients in the optimizer
        optimizer.zero_grad()

        # run the model which is the forward process
        out = model(feats)

        # compute the binary cross entropy loss, and call backward propagation function
        loss = bce_loss(out, labels)
        loss.backward()

        # sum up of total loss, loss.item() return the value of the tensor as a standard python number
        # this operation is not differentiable
        total_loss += loss.item()

        # call a function to update the parameters of the models
        optimizer.step()
```

模型测试

- 训练过程中在验证集上验证、训练结束时在测试集上进行测试时，模型参数均不参与训练，所以无需在验证/测试过程中计算梯度、更新模型参数，只需要进行前向计算(forward)即可
  - 设置验证模式
  - 关闭梯度计算
  - 进行前向计算
  - 评价指标

```python
# validation
if (epoch + 1) % valInterval == 0:
    # set the model in evaluation mode
    # call a function to enter evaluation mode
    model.eval()

    n_correct = 0.  # number of images that are correctly classified
    n_ims = 0.  # number of total images

    with torch.no_grad():  # we do not need to compute gradients during validation

        for feats, labels in valloader:
            feats, labels = feats.to(device), labels.type(torch.float).to(device)
            # input images "ims" into the model for the forward process to generate output
            # similar to the forward process in training
            out = model(feats)

            # if out > 0.5, assign the prediction result as a character image, otherwise as a background image
            predictions = torch.round(out)

            # sum up the number of images correctly recognized
            n_correct += torch.sum(predictions == labels)

            # sum up the total image number
            n_ims += feats.size(0)

    # show prediction accuracy
    print('Epoch {:02d}: validation accuracy = {:.1f}%'.format(epoch + 1, 100 * n_correct / n_ims))
```

主函数

- 与C++的main函数类似，python也需要一个主函数
  - 定义主函数
  - 设定随机种子
  - 设计实现处理流程
  - 根据需要，处理输入的命令行参数

```python
if __name__ == '__main__':
    # set random seed for reproducibility
    seed = 3
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True

    args = sys.argv

    if len(args) < 2 or args[1] not in ['train', 'test', 'visual']:
        print('Usage: $python classification.py [mode]')
        print('       mode should be one of [train, test, visual]')
        print('Example: python classification.py train')
        raise AssertionError

    mode = args[1]

    # -- run the code for training and validation
    if mode == 'train':
        train_val(train_file_path='data/character_classification/train_feat.npy',
                  val_file_path='data/character_classification/val_feat.npy', n_epochs=20, batch_size=8,
                  lr=0.01, momentum=0.9, valInterval=5, device='cpu')

    # -- test the saved model
    elif mode == 'test':
        test(model_path='saved_models/model_epoch20.pth',
             test_file_path='data/character_classification/train_feat.npy',
             batch_size=8, device='cpu')

    # -- visualization
    else:
        visual(model_path='saved_models/model_epoch20.pth',
               file_path='data/character_classification/test_feat.npy')
```

搭建网络的细节

- 除了使用nn.Linear类以外，线性层还可以怎么实现？
  - 提示：使用nn.Parameter

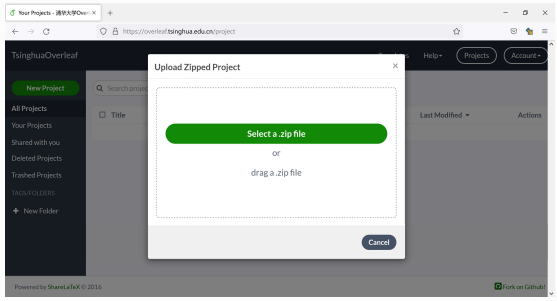
- torch.matmul()包含哪些必要参数？

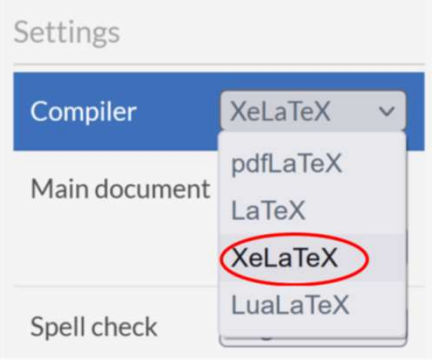
搭建网络的细节

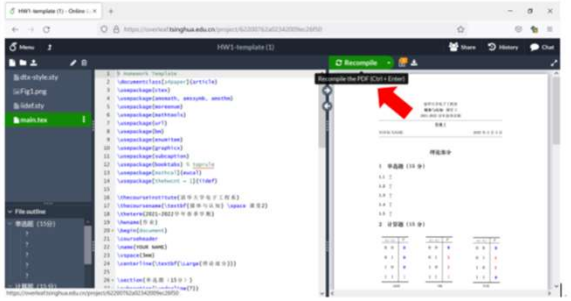
- 为什么计算loss一般需要取均值？
- 除了使用nn.BCELoss类以外，bce_loss还可以如何实现？

关于作业的LaTex模板

- 随作业发布的LaTex模板打包文件为HW1-template.zip
- 用校园网账号登录Tsinghua Overleaf 在线LaTeX文字排版网站：https://overleaf.tsinghua.edu.cn/project
  - 点击左上角 “New Project”→“**Upload Project**”，上传HW1-template.zip
  - 点击左上角“Menu”，将其中的“Compiller”设为“**XeLaTex**”，以便处理中文LaTeX文档
  - 选中“main.tex”文件，点击右边窗口上方的“**Recompile**”，即可编译生成PDF文件







- LaTex使用方法可参考 https://www.overleaf.com/learn

参考资料

- Pytorch官方文档：PyTorch documentation — PyTorch 1.10 documentation
- 关于Pytorch中tensor的官方讲解：Tensors — PyTorch Tutorials 1.10.1+cu102 documentation

谢谢大家！