科学计算与数学建模实验指导书

内 容 简 介

本课程以数学建模思想、方法为主线，有机融入科学计算的理论与方法，是集科学计算方法、现代数学、计算机技术与实际问题求解于一体的一门新型课程，采用研究性教学与探索型学习相结合的教学模式，主要讲授数学建模思想和科学计算方法。

本课程的学习应注重理论与实践相结合，因此实验教学是教学环节中必不可少的重要内容。在教学实验过程中，以实际问题为背景，渗透数学建模思想，通过课程中所学到数学建模的步骤和方法，编程实现实际问题的数学模型，用模型的求解引入科学计算的基本知识和一般方法。

实验部分包括实验目的、实验内容和实验所需环境等，介绍了每个实验所需的一些基础知识和技巧。在实验中给出的实验题，跟课堂教学的内容都有密切的关系，所以需要将课堂上讲授的例子程序融会贯通，掌握实验所需的一些基本方法和工具，并在吃透例子程序的基础上，积极独立思考设计和编写满足实验要求的程序。

中南大学计算机学院

上机实验要求及规范

深度学习课程具有比较强的实践性。上机实验是一个重要的教学环节。一般情况下学生能够重视实验环节，对于编写程序上机练习具有一定的积极性。但是容易忽略实验的总结，忽略实验报告的撰写。对于一名大学生必须严格训练分析总结能力、书面表达能力。需要逐步培养书写科学实验报告以及科技论文的能力。拿到一个题目，一般不要急于编程。正确的方法是：首先理解问题，明确给定的条件和要求解决的问题，然后按照自顶向下，逐步求精，分而治之的策略，按照面向对象的程序设计思路，逐一地解决子问题。

一、实验报告的基本要求：

一般性、较小规模的上机实验题，必须遵循下列要求。养成良好的习惯。

姓名 班级 学号 日期 题目

i. 问题描述

ii. 设计简要描述

iii. 程序清单（带有必要的注释）

iv. 结果分析（原始图示，测试数据与运行记录，分析正确性；）

v. 调试报告：

实验者必须重视最后这两个环节，否则等同于没有完成实验任务。这里可以体现个人特色、或创造性思维。具体内容包括：测试数据与运行记录；调试中遇到的主要问题，自己是如何解决的；经验和体会等。

二、实验报告的提高要求：

阶段性、较大规模的上机实验题，应该遵循下列要求。养成科学的习惯。

（1）问题描述

（2）需求和规格说明

（3）描述问题，简述题目要解决的问题是什么。规定软件做什么。原题条件不足时补全。

（4）概要设计：功能模块的划分

（5）详细设计：每部分模块的设计，含数据结构的设计，算法的描述（流程图或PDL）

a.设计思想：主要算法基本思想。

b.设计表示：每个函数的头和规格说明；列出每个函数所调用和被调用的函数，也可以通过调用关系图表达。

（6）实现注释：各项功能的实现程度、在完成基本要求的基础上还有什么功能。

（7）用户手册：即使用说明书。

（8）调试报告：调试过程中遇到的主要问题是如何解决的；设计的回顾、讨论和分析；时间复杂度、空间复杂度分析；改进设想；经验和体会等。

基于深度学习的医药诊断评估系统

一、实验目的

① 掌握卷积网络、嵌入层、LSTM、自编码器等方面的内容
② 学会搭建 Python 开发环境
③ 学会使用 PyTorch 和 Keras 框架搭建神经网络
④ 学会使用 pyplot 绘制实验结果

二、实验开发环境和工具

可以在Linux、Windows或Mac操作系统上搭建开发环境，所使用的开发工具包括Anaconda、PyTorch、Tensorflow、Keras等，使用Python语言。因为Keras是基于Tensorflow的高层API,所以需要先安装Tensorflow再安装Keras。

实验指导书中的内容以在Windows系统下使用PyCharm进行开发为例。

三、实验内容

本次实验需要完成医药诊断系统中的两个模块：基于自编码器和卷积网络的肺炎图像识别模块和基于嵌入层和LSTM的药物评价情感分析模块。

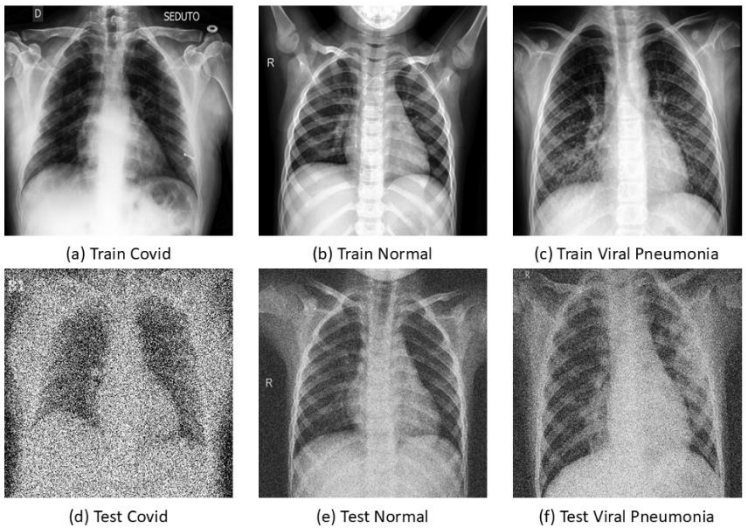
1. 基于自编码器和卷积网络的肺炎图像识别模块

1.1 数据介绍

实验使用肺部X-光片作为数据集，标签分为三类：新冠肺炎、正常和普通肺炎。数据分布如下表所示：

	Covid	Normal	Viral Pneumonia
Train	111	70	70
Test	26	20	20

对于测试集的图片，我们使用高斯噪声进行了扰动。对于训练集，我们使用原始的干净图片。



(a) Train Covid　(b) Train Normal　(c) Train Viral Pneumonia

(d) Test Covid　(e) Test Normal　(f) Test Viral Pneumonia

1.2 实验准备

实验开始前，需要先安装PyTorch，可以通过命令行的方式安装CPU版本：

```
>pip3 install torch torchvision torchaudio
```

或

```
>conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

更推荐安装GPU版本并使用显卡进行训练。安装GPU版本前需要先安装CUDA，可以参考CSDN的教程。安装完CUDA后可以在PyTorch上找到对应CUDA版本PyTorch的安装命令。

1.3 实验步骤

1.3.1 导入相关库

```
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import transforms, datasets
from tqdm import tqdm
```

plt用于绘图，nn中包含了用于构建神经网络的隐藏层（全连接层、卷积层等），F中包含了各种激活函数（ReLu、Sigmoid等），DataLoader用于在训练时加载数据，datasets和transforms用于读取和处理数据集，tqdm用于进度条可视化。

1.3.2 定义超参数

```
TRAIN_BATCH_SIZE = 32
TEST_BATCH_SIZE = 66
EPOCHS = 50
LR = 0.01
device = "cuda" if torch.cuda.is_available() else "cpu"
print('[INFO] Running on ' + device)
```

TRAIN_BATCH_SIZE: 训练时批次大小，即一次更新使用多少样本进行梯度计算。设置的越大，占用显存越多，但计算次数越少。一般设置为2的n次方。

TEST_BATCH_SIZE: 与上类似，此处66为测试集总样本数，即一次性将所有测试数据放入模型计算。

EPOCHS: 总共训练多少轮。每一轮训练会遍历一遍训练数据。

LR: 学习率，也叫做步长，$w_{t+1} = w_t - \eta g$中的η。

device: 使用何种设备进行训练，如果安装的cpu版本则只能使用cpu。

1.3.3 读取数据

torch对于一些常用的数据集做了封装，可以直接调用，例如datasets.MNIST()。但此处我们使用的是本地的图片数据，可以使用ImageFolder将一个文件夹下的图片读取成数据集。

```
train_dataset = datasets.ImageFolder('../data/covid19/train',
                                     transforms.Compose([
                                         transforms.Grayscale(),
                                         transforms.Resize((256, 256)),
                                         transforms.ToTensor()])
                                     )
```

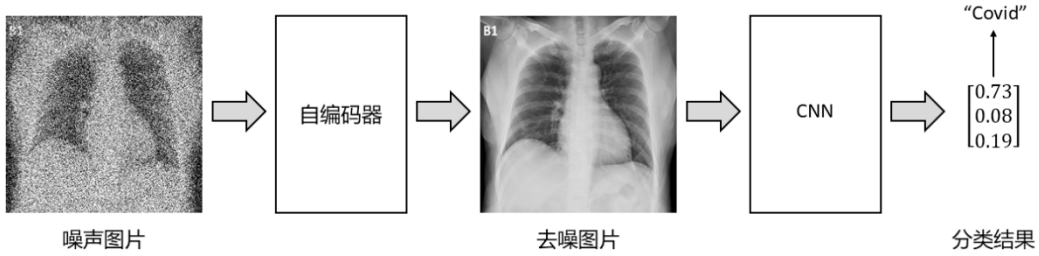
transforms指在读取数据时对数据进行的处理。
Grayscale: 指以灰度图的形式读取。
Resize: 由于图片文件尺寸不一致，在训练前需要将它们重塑成相同尺寸。
ToTensor: 将图片格式转换成张量形式，torch的计算以张量的形式进行。
常用的还有Normalize进行标准化等。
在读取完数据集后还需要定义DataLoader用于加载数据。

```
train_dataloader = DataLoader(train_dataset, batch_size=TRAIN_BATCH_SIZE, shuffle=True)
```

对于测试集的处理类似，请同学们自行完成。

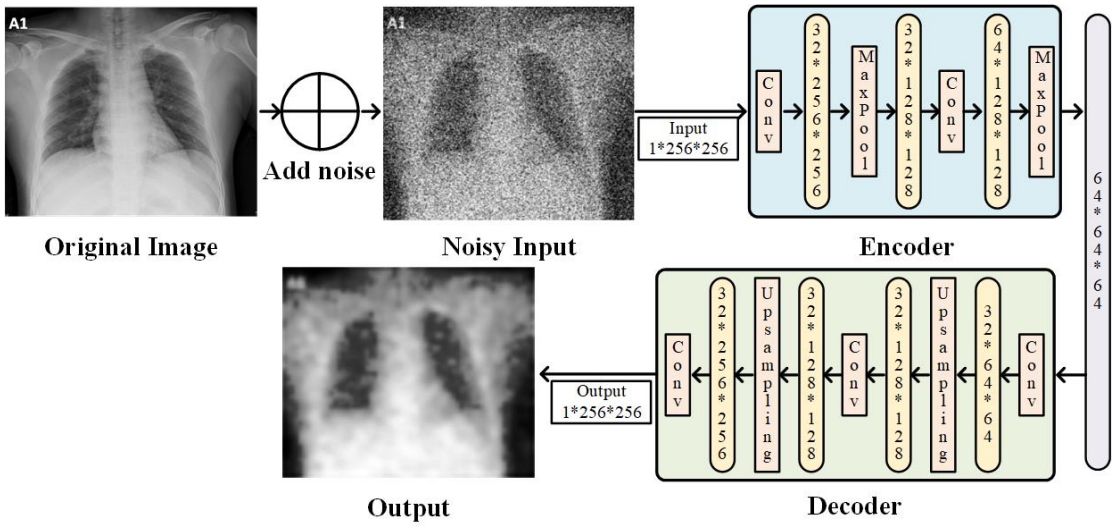
1.3.4 定义模型

本模块由两个模型组成，分别是用于去噪的自编码器和用于分类的CNN。



(1) 自编码器

自编码器的模型结构如下图所示：



模型定义代码如下：

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2)
        )
        self.decoder = nn.Sequential(
            nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.UpsamplingNearest2d(scale_factor=2),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.UpsamplingNearest2d(scale_factor=2),
            nn.Conv2d(32, 1, kernel_size=3, stride=1, padding=1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

模型继承自nn.Module类，在__init__()函数中定义模型的结构，在forward()函数中定义模型的前向传播过程。

(2) CNN

以下分别是CNN的模型结构和前向传播过程：

```
self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)
self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
self.fc1 = nn.Linear(in_features=32 * 64 * 64, out_features=128)
self.fc2 = nn.Linear(in_features=128, out_features=32)
self.fc3 = nn.Linear(in_features=32, out_features=3)
```

```
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.view(-1, 32 * 64 * 64)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

请同学们仿照自编码器结构图绘制出CNN的网络结构图，包括数据经过每一个隐藏层后的尺寸变化（不必绘制激活函数）。

1.3.5 模型训练

(1) 初始化模型

```
autoencoder = Autoencoder().to(device)
```

通过to()来指定在CPU或GPU上计算。

(2) 指定损失函数与优化器算法

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=LR)
```

自编码器选用均方差作为损失函数，Adam作为优化器。

(3) 开始训练

基本的训练流程如下所示：

```
autoencoder.train()
for epoch in range(EPOCHS):
    for img, label in tqdm(train_dataloader):
        img = img.to(device)
        noisy_img = add_noise(img)
        optimizer.zero_grad()
        reconstructed = autoencoder(noisy_img)
        loss = criterion(reconstructed, img)
        loss.backward()
        optimizer.step()
```

每次更新执行如下操作：

① 从加载器中获取输入数据
② 清空梯度
③ 计算模型输出
④ 根据模型输出和标签计算loss
⑤ 反向传播
⑥ 更新模型

由于训练集是干净图片，我们在将其输入到模型之前需要手动添加噪声，代码如下：

```
def add_noise(img, noise_factor=0.5):
    noisy_img = img + noise_factor * torch.randn_like(img)
    noisy_img = torch.clamp(noisy_img, 0., 1.)
    return noisy_img
```

请同学们自行完成：在训练过程中打印每个epoch的平均loss变化情况，如下图所示：

```
100%|██████████| 8/8 [00:05<00:00,  1.53it/s]
[Autoencoder]  Epoch 45/50  Loss: 0.0044
100%|██████████| 8/8 [00:05<00:00,  1.55it/s]
[Autoencoder]  Epoch 46/50  Loss: 0.0043
100%|██████████| 8/8 [00:05<00:00,  1.53it/s]
[Autoencoder]  Epoch 47/50  Loss: 0.0043
100%|██████████| 8/8 [00:04<00:00,  1.61it/s]
[Autoencoder]  Epoch 48/50  Loss: 0.0043
100%|██████████| 8/8 [00:05<00:00,  1.57it/s]
[Autoencoder]  Epoch 49/50  Loss: 0.0043
100%|██████████| 8/8 [00:04<00:00,  1.61it/s]
[Autoencoder]  Epoch 50/50  Loss: 0.0043
```

提示：loss变量的类型是Tensor，通过loss.item()可以获取其值。

对于CNN，损失函数选用交叉熵，训练过程与上类似，需要注意的是，在将数据输入模型前，除了需要手动添加噪声，还需要先通过autoencoder，为了不在更新CNN的同时更新自编码器，这一步不需要产生梯度：

```
for data, label in train_dataloader:
    data, label = data.to(device), label.to(device)
    noisy_data = add_noise(data)
    with torch.no_grad():
        input = autoencoder(noisy_data)
    output = model(input)
```

请同学们自行完成CNN的训练，并打印训练过程中loss和正确率的变化。

提示：下述代码可以计算出当前批次中被正确判断的样本数：

```
pred = output.argmax(
    dim=1, keepdim=True
)
correct += pred.eq(target.view_as(pred)).sum().item()
```

模型的保存与加载：

```
torch.save(model.state_dict(), '../model/cnn.pth')

model.load_state_dict(torch.load('../model/cnn.pth', weights_only=True))
```

1.3.6 模型测试

可以在每个epoch结束时对模型进行测试，测试代码与训练代码类似，只是不会进行更新，代码如下：

```
model.eval()
with torch.no_grad():
    for data, label in test_dataloader:
        data, label = data.to(device), label.to(device)
        data = autoencoder(data)
        output = model(data)
        # 计算测试loss
        # 计算测试正确率
```

由于测试集已经是加噪的图片，我们不必手动添加噪声。

请同学们自行完成：打印CNN在不同epoch训练和测试的loss及正确率，如下图所示：

```
100%|██████████| 8/8 [00:04<00:00,  1.88it/s]
[CNN]  Epoch 46/50 Train  Loss: 0.000020  Accuracy: 100.00
[CNN]  Epoch 46/50 Test  Loss: 1.013535  Accuracy: 87.88
100%|██████████| 8/8 [00:04<00:00,  1.90it/s]
[CNN]  Epoch 47/50 Train  Loss: 0.000015  Accuracy: 100.00
[CNN]  Epoch 47/50 Test  Loss: 1.017201  Accuracy: 87.88
100%|██████████| 8/8 [00:04<00:00,  1.87it/s]
[CNN]  Epoch 48/50 Train  Loss: 0.000030  Accuracy: 100.00
[CNN]  Epoch 48/50 Test  Loss: 1.021951  Accuracy: 87.88
100%|██████████| 8/8 [00:04<00:00,  1.87it/s]
[CNN]  Epoch 49/50 Train  Loss: 0.000019  Accuracy: 100.00
[CNN]  Epoch 49/50 Test  Loss: 1.025834  Accuracy: 87.88
100%|██████████| 8/8 [00:04<00:00,  1.88it/s]
[CNN]  Epoch 50/50 Train  Loss: 0.000016  Accuracy: 100.00
[CNN]  Epoch 50/50 Test  Loss: 1.028288  Accuracy: 87.88
```

提示：可以把训练和测试的过程封装成函数train(epoch)和test(epoch)，在函数的最后打印对应信息，然后通过下方代码来执行训练和测试。

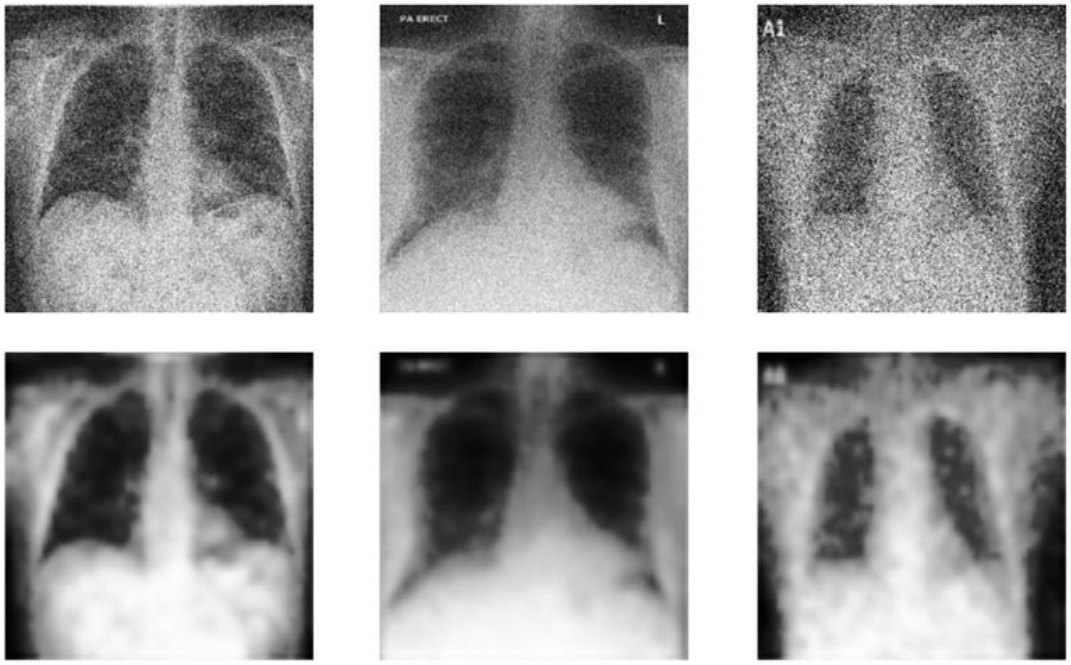
```
for i in range(EPOCHS):
    train(i)
    test(i)
```

1.3.7 绘制结果

(1) 自编码器结果绘制

我们在模型计算时使用的变量类型都是Tensor，我们需要先将其转换成图片形式，可以通过如下代码展示autoencoder的输入和输出。

```
for data, _ in test_dataloader:
    data = data.to(device)
    output = autoencoder(data)
    tp = transforms.ToPILImage()
    noisy_img = tp(data[0].cpu())
    reconstructed = tp(output[0].cpu())
    plt.imshow(noisy_img, cmap='gray')
    plt.axis('off')
    plt.show()
    plt.imshow(reconstructed, cmap='gray')
    plt.axis('off')
    plt.show()
```



(2) CNN训练过程正确率及loss可视化

可以定义两个列表用于存储训练过程中每个epoch的loss和正确率。

```
loss_per_epoch = []
acc_per_epoch = []
```

在训练和测试过程中，我们可以向这两个列表添加数据：

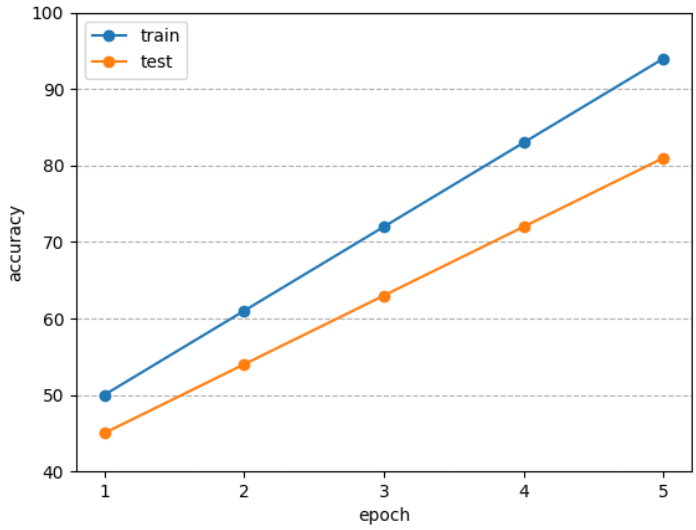
```
# 训练过程
# 计算loss
# 计算acc
loss_per_epoch.append(loss)
acc_per_epoch.append(acc)
```

有了数据之后，我们可以绘制正确率和loss的变化情况，下面是绘制折线图的代码示例（并非真实数据）：

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import MultipleLocator

epoch = [1, 2, 3, 4, 5]
acc_train = [50, 61, 72, 83, 94]
acc_test = [45, 54, 63, 72, 81]

plt.plot(epoch, acc_train, label='train', marker='o')
plt.plot(epoch, acc_test, label='test', marker='o')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend()
# y轴设置范围为40~100
plt.ylim(40, 100)
# x轴刻度设为1的倍数
ax = plt.gca()
ax.xaxis.set_major_locator(MultipleLocator(1))
# 背景绘制虚线
plt.grid(axis="y", linestyle='--')
plt.show()
```



1.3.8 （选做）部署模型

把模型部署到实际应用中，以Web或GUI为用户提供操作接口以上传图片。Web可以考虑使用Flask作为服务端，GUI可以考虑使用PyQT开发。

2. 基于嵌入层和LSTM的药物评价情感分析模块

2.1 数据介绍

本实验使用的数据集是国外对药物的评论及评分数据，数据集的属性如下表所示：

uniqueID	drugName	condition	review	rating	date	usefulCount
唯一标识符	药物名称	病人症状	病人对药物的评论	病人对药物的评分	评价日期	觉得此评价有用的人数

评分取值为1到10的整数，在实验中我们只使用到review和rating属性。

2.2 实验准备

在实验开始前，需要安装Tensorflow和Keras包，可以通过如下指令安装：

```
>pip install tensorflow
```

```
>pip install keras
```

提示：tensorlow和keras对numpy、pandas等包有着较为严格的版本依赖，如果担心破坏已有的python环境，建议创建一个新的环境。

2.3 实验步骤

2.3.1 导入相关包

```
import pandas as pd
from keras._tf_keras.keras.utils import to_categorical
from keras._tf_keras.keras.utils import pad_sequences
from keras._tf_keras.keras.preprocessing.text import Tokenizer
from keras._tf_keras.keras.models import Sequential
from keras._tf_keras.keras.layers import Embedding, Dense, LSTM
import matplotlib.pyplot as plt
```

pd用于读取csv格式的文件，to_categorical、pad_sequences和Tokenizer用于处理文本数据，Sequential、Embedding、Dense和LSTM用于搭建模型，plt用于绘图。

2.3.2 读取数据

(1) 读取原始数据

```
train_data = pd.read_csv('../data/review/drugsComTrain_raw.csv')
```

	uniqueID	drugName	condition	review	rating	date	usefulCount
0	206461	Valsartan	Left Ventricular...	"It has no si...	9	20-May-12	27
1	95260	Guanfacine	ADHD	"My son is h...	8	27-Apr-10	192
2	92703	Lybrel	Birth Control	"I used to ta...	5	14-Dec-09	17
3	138000	Ortho Evra	Birth Control	"This is my fi...	8	3-Nov-15	10
4	35696	Buprenorphine / ...	Opiate Depend...	"Suboxone ...	9	27-Nov-16	37
5	155963	Cialis	Benign Prostati...	"2nd day on...	2	28-Nov-15	43
6	165907	Levonorgestrel	Emergency Con...	"He pulled ...	1	7-Mar-17	5
7	102654	Aripiprazole	Bipolar Disorde	"Abilify chan...	10	14-Mar-15	32
8	74811	Keppra	Epilepsy	" I Ve had n...	1	9-Aug-16	11
9	48928	Ethinyl estradiol ...	Birth Control	"I had been ...	8	8-Dec-16	1
10	29607	Topiramate	Migraine Preve...	"I have been...	9	1-Jan-15	19

(2) 提取“评分”一列

```
train_rating = train_data['rating']
```

	0
0	9
1	8
2	5
3	8
4	9
5	2

(3) 根据评分确定情感标签：1-4分：消极、5-6分：中性、7-10分：积极

```
train_label = -1 * (train_rating <= 4) + 1 * (train_rating >= 7)
```

	0
0	1
1	1
2	0
3	1
4	1
5	-1

测试集处理类似，自行完成。

2.3.3 处理数据

(1) 提取“评论”一列

```
train_review = train_data['review']
```

	0
0	"It has no side effect, I take it in combination of Bystolic...
1	"My son is halfway through his fourth week of Intuniv. ...
2	"I used to take another oral contraceptive, which had 2...
3	"This is my first time using any form of birth control. I&...
4	"Suboxone has completely turned my life around. I fee...
5	"2nd day on 5mg started to work with rock hard erecti...

(2) 将文本序列化

```
tokenizer = Tokenizer(num_words=WORDS)
tokenizer.fit_on_texts(train_review)
train_sequence = tokenizer.texts_to_sequences(train_review)
```

```
train_sequence = {list: 161297}
  000000 = {list: 17} [5, 38, 28, 35, 197, 1, 45, 5, 15, 848, 12, 2922, 99, 150, 2, 3806, 1551]
  000001 = {list: 141} [6, 638, 19, 3380, 225, 508, 1652, 76, 12, 3254, 262, 404, 1226, 62,
  000002 = {list: 138} [1, 153, 4, 45, 277, 1184, 1840, 90, 18, 912, 70, 542, 2, 10, 57, 266,
  000003 = {list: 93} [14, 19, 6, 44, 58, 182, 100, 729, 12, 140, 118, 1, 8, 34, 594, 1, 108, 21,
  000004 = {list: 127} [899, 38, 273, 788, 6, 92, 224, 1, 65, 2318, 1, 8, 34, 26, 6, 452, 2, 1,
  000005 = {list: 69} [625, 29, 13, 523, 49, 4, 113, 21, 2544, 368, 2248, 200, 346, 436, 720,
```

训练样本一共由161297句话，第一句话有17个单词，train_sequence[i][j]中的值代表第i句话中第j个单词在字典中的下标。

(3) 将序列化后的文本填充并形成等长序列

```
x_train = pad_sequences(train_sequence, maxlen=LENGTH)
```

	481	482	483	484	485	486	487	488	489	490	491
0	0	0	5	38	28	35	197	1	45	5	15
1	508	1951	19	107	73	262	11	134	221	248	417
2	1	121	8	22	11	100	115	35	43	3	973
3	12	6	80	1	105	18	184	83	182	140	118
4	19	32	5	9	17	5	785	19	383	3	650
5	60	4931	9	48	1693	1	8	34	480	166	113

(4) 将标签形成独热编码

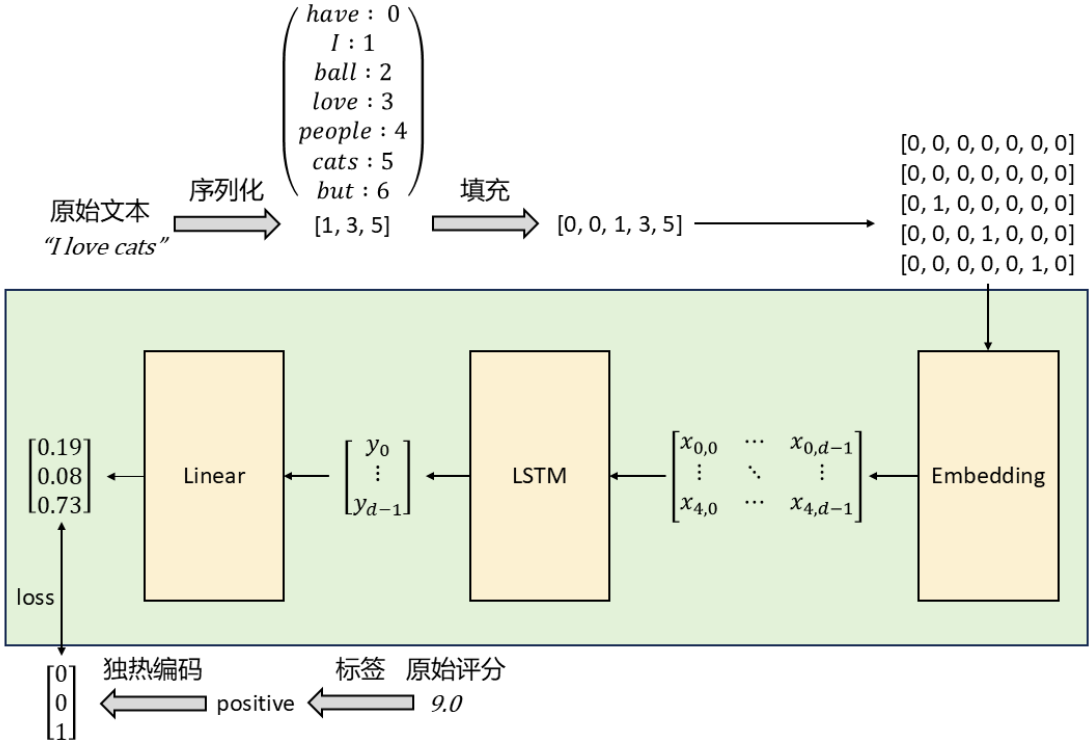
```
y_train = to_categorical(train_label, num_classes=3)
```

	0	1	2
0	0.00000	1.00000	0.00000
1	0.00000	1.00000	0.00000
2	1.00000	0.00000	0.00000
3	0.00000	1.00000	0.00000
4	0.00000	1.00000	0.00000
5	0.00000	0.00000	1.00000

对于测试集采用同样的处理方式，注意不需要一个新的tokenizer，如果专门再为测试集构建一个tokenizer相当于泄露了测试集的信息。

2.3.4 搭建模型

模型结构如下所示：



通过下述代码定义模型结构以及优化器、损失函数和性能指标：

```
model = Sequential()
model.add(Embedding(WORDS, DEPTH, input_length=LENGTH))
model.add(LSTM(DEPTH))
model.add(Dense(3, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
```

2.3.5 训练及评估模型

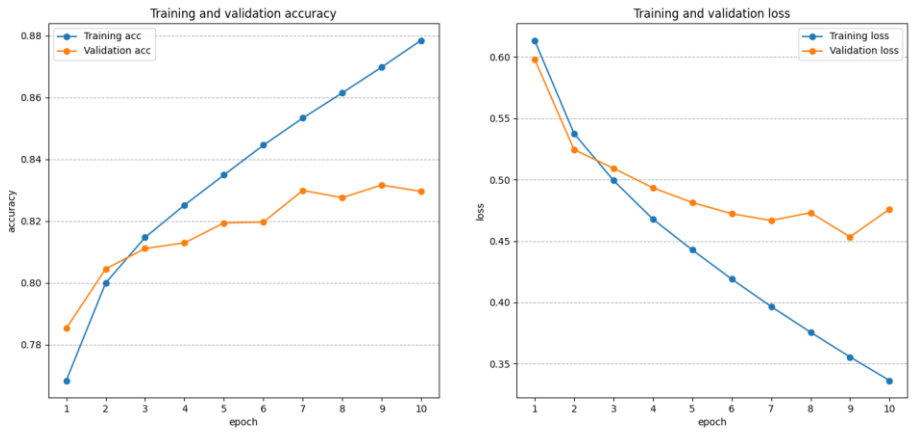
通过下述代码进行训练和测试：

```
history = model.fit(x_train, y_train, epochs=10, batch_size=32, verbose=1, validation_split=0.25)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)
```

定义一个函数来绘制训练过程中的指标变化：

```python
def plot_history(history):
    f, ax = plt.subplots(1, 2, figsize=(16, 7))
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)
    plt.sca(ax[0])
    ax[0].xaxis.set_major_locator(MultipleLocator(1))
    plt.plot(epochs, acc, marker='o', label='Training acc')
    plt.plot(epochs, val_acc, marker='o', label='Validation acc')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.title('Training and validation accuracy')
    plt.grid(axis="y", linestyle='--')
    plt.legend()
    plt.sca(ax[1])
    ax[1].xaxis.set_major_locator(MultipleLocator(1))
    plt.plot(epochs, loss, marker='o', label='Training loss')
    plt.plot(epochs, val_loss, marker='o', label='Validation loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.title('Training and validation loss')
    plt.grid(axis="y", linestyle='--')
    plt.legend()
    plt.show()
```



2.3.6 比较keras和torch两个框架各自的特点

2.3.7 （选做）用torch复现本模块