自动写诗

撰写人：微电子所-杨登天-202028015926089

目 录

自动写诗

杨登天-微电子所-202028015926089

1、实验要求

- 基于 Python 语言和任意一种深度学习框架(实验指导书中使用 Pytorch 框架进行介绍)，完成数据读取、网络设计、网络构建、模型训练和模型测试等过程，最终实现一个可以自动写诗的程序。网络结构设计要有自己的方案，不能与实验指导书完全相同。
- 随意给出首句，如给定“湖光秋月两相和”，输出模型续写的诗句。也可以根据自己的兴趣，进一步实现写藏头诗（不做要求）。要求输出的诗句尽可能地满足汉语语法和表达习惯。实验提供预处理后的唐诗数据集，包含 57580 首唐诗（在课程网站下载），也可以使用其他唐诗数据集。
- 按规定时间在课程网站提交实验报告、代码以及 PPT。

2、环境配置

Anaconda Navigator 1.10.0
Python 3.7.10 64-bit
Pytorch
Visual Studio Code 1.56.2
Colab

3、实验所用数据集

实验提供预处理过的数据集，含有 57580 首唐诗，每首诗限定在 125 词，不足 125 词的以</s>填充。数据集以 npz 文件形式保存，包含三个部分：

（1） data: 诗词数据，将诗词中的字转化为其在字典中的序号表示。
（2） ix2word: 序号到字的映射
（3） word2ix: 字到序号的映射

以上可以通过以下代码来尝试验证

```
---------------------------------------------------------------------------
def prepareData():
    datas = np.load("D://AI computer system Lab/MyLab/DLcourse/Lab3/tang.npz", allow_pickle=True)
    data = datas['data']
    ix2word = datas['ix2word'].item()
    word2ix = datas['word2ix'].item()
    data = torch.from_numpy(data)
```

```
    dataloader = DataLoader(data,
                            batch_size=16,
                            shuffle=True,
                            num_workers=0)
    return dataloader, ix2word, word2ix

poem_loader, ix2word, word2ix = prepareData()
print(ix2word)
print(word2ix['<EOP>'])
```

4、实验分析

本次代码的构建主体是实验代码，但是在网络层结构、训练和预测上自行发展！
其中，修改了实验指导程序的 Config，因为确实一开始没有弄明白，所以就打算直接用以前的思路定义常量和初始化常量的方式替代 Config！
其中，修改了《实验指导文件》中并未提到的完全的 3 个连接层，源程序只有一个连接层，根据这个提示新增 2 个连接层，具体参数设置比较大胆，有参考知乎上的经验！
训练和调用分为两部分，训练部分在 google 的 colab 上实现

4.1 数据处理模块

这一部分完成的目标是对 tang.npz 数据集的情况进行试探，并对 3 个部分分别加以提取。
代码如下：

```
def prepareData():
    datas = np.load("drive/My Drive/Notebooks/tang.npz", allow_pickle=True)
    data = datas['data']
    ix2word = datas['ix2word'].item()
    word2ix = datas['word2ix'].item()
    data = torch.from_numpy(data)
    dataloader = DataLoader(data,batch_size=16,shuffle=True,num_workers=0)
    return dataloader, ix2word, word2ix

#测试
poem_loader, ix2word, word2ix = prepareData()
#print(word2ix['哪'])
#print(ix2word) # {0: '<character>', ... , 8290 : '<EOP>', 8291: '<START>', 8292: '</s>'}
#print(word2ix['<EOP>']) # 8290
num_box = 0
#for li, oth in enumerate(poem_loader):
#    num_box += 1
```

```
#print(num_box)
```

4.2 模型构建模块

这一部分完成的目标是实现 Embedding、单层 LSTM、全连接层的三个网络层，而事实上实验指导代码并未提供完整的三个全连接层，所以这里对三个全连接层进行补充完善！

```
class PoetryModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, drop_prob, lstm_layers):
        super(PoetryModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.lstm_layers = lstm_layers
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, self.hidden_dim, num_layers=self.lstm_layers,
batch_first=True)
        #self.linear = nn.Linear(self.hidden_dim, vocab_size)
        self.linear1 = nn.Linear(self.hidden_dim,2048)
        self.linear2 = nn.Linear(2048,4096)
        self.linear3 = nn.Linear(4096,vocab_size)
        #增加 dropout
        self.dropout = nn.Dropout(drop_prob)

    def forward(self, input, hidden=None):
        embeds = self.embeddings(input)
        batch_size, seq_len = input.size()
        if hidden is None:
            h_0 = input.data.new(self.lstm_layers, batch_size, self.hidden_dim).fill_(0).float()
            c_0 = input.data.new(self.lstm_layers, batch_size, self.hidden_dim).fill_(0).float()
        else:
            h_0, c_0 = hidden
        sample_pre, hidden = self.lstm(embeds, (h_0, c_0))
        #  多加两层
        sample_pre = torch.tanh(self.linear1(sample_pre))
        #sample_pre = self.dropout(sample_pre)
        sample_pre = torch.tanh(self.linear2(sample_pre))
        sample_pre = self.linear3(sample_pre)
        sample_pre = sample_pre.reshape(batch_size * seq_len, -1)
        return sample_pre, hidden
```

4.3 训练并保存模型模块

本模块还包括对未训练完的模型接着调用训练，其中对调用模块那句话进行了注释，需要时解除注释，但在运行之前还需要对最近更新的模型文件进行改名字！训练部分主要是通过词向量和 LSTM 网络来预测接下来会出现的词语或者诗句，和 target 的监督向量进行对比，从而根据这样的监督信号来不断改进整个网络的参数结构！

```
---------------------------------------------------------------------------
    def train(epochs, poem_loader, word2ix):
        # 定义模型、设置优化器和损失函数、获取模型输出、计算误差、误差反向传播等步骤
        model = PoetryModel(len(word2ix),embedding_dim=EMBEDDING_DIM, hidden_dim=HIDDEN_DIM, drop_prob=DROP_PROB,lstm_layers=LSTM_LAYER)
        model.train()
        model.to(device) # 移动模型到 cuda
        optimizer = optim.Adam(model.parameters(), lr=LR)
        scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size = 10,gamma=0.1)
        criterion = nn.CrossEntropyLoss()
        #model.load_state_dict(torch.load("drive/My Drive/Notebooks/tang.pth"))
        loss_meter = meter.AverageValueMeter()

        loss_box = []
        iter_box = []

        for epoch in np.arange(epochs):
            loss_meter.reset()
            for chara_index, character in enumerate(poem_loader):
                print("Epoch ",epoch," : It has trained for ",chara_index," iterations!")
                character = character.long().transpose(1,0).contiguous()
                character = character.to(device)
                #character.contiguous()
                #sample = character[0].to(device)
                #target = character[1].to(device)
                #target = target.view(-1)
                optimizer.zero_grad()
                sample = character[:-1, :]
                target = character[1:, :]
                sample_pre, _ = model(sample)
                loss = criterion(sample_pre,target.view(-1))
                loss.backward()
                optimizer.step()
                loss_meter.add(loss.item())
            # output model
            torch.save(model.state_dict(),'drive/My Drive/Notebooks/tang.pth')
```

```
                print("It has saved .pth file for ",epoch," times!")
                model.load_state_dict(torch.load("drive/My Drive/Notebooks/tang.pth"))
                loss_box.append(loss)
                iter_box.append(epoch)
                scheduler.step()
        print("Finish all!")
```

4.4 预测模块

本模块沿用实验指导文件的代码，其程序如下

```
def generate(model, start_words, ix2word, word2ix):
    results = list(start_words)
    start_words_len = len(start_words)
    #  第一个词语是<START>
    input = torch.Tensor([word2ix['<START>']]).view(1, 1).long()
    hidden = None
    model.eval()
    with torch.no_grad():
        for i in range(MAX_GEN_LEN):
            sample_pre, hidden = model(input, hidden)
        #  如果在给定的句首中，input 为句首中的下一个字
            if i < start_words_len:
                w = results[i]
                input = input.data.new([word2ix[w]]).view(1, 1)
                #  否则将 output 作为下一个 input 进行
            else:
                top_index = sample_pre.data[0].topk(1)[1][0].item()
                w = ix2word[top_index]
                results.append(w)
                input = input.data.new([top_index]).view(1, 1)
            if w == '<EOP>':
                del results[-1]
                break
    return results
```

4.5 预测诗句模块

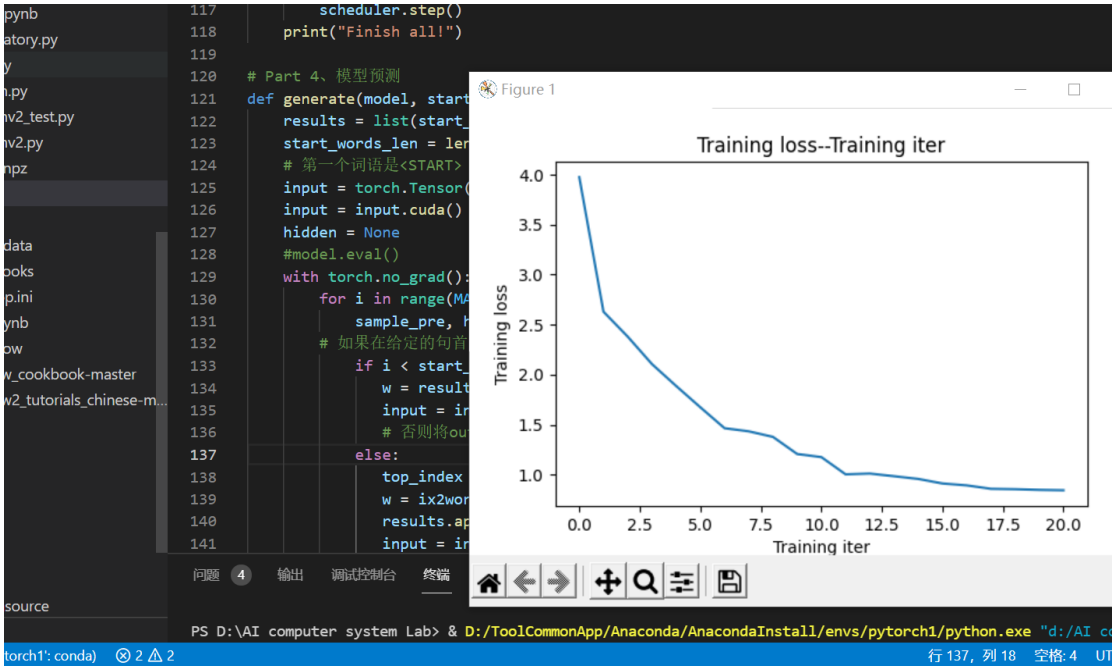
这一模块调用已经训练好的模型进行诗句预测，对输入首句进行预测。从 colab 转移到本地。

```
#预处理部分
```

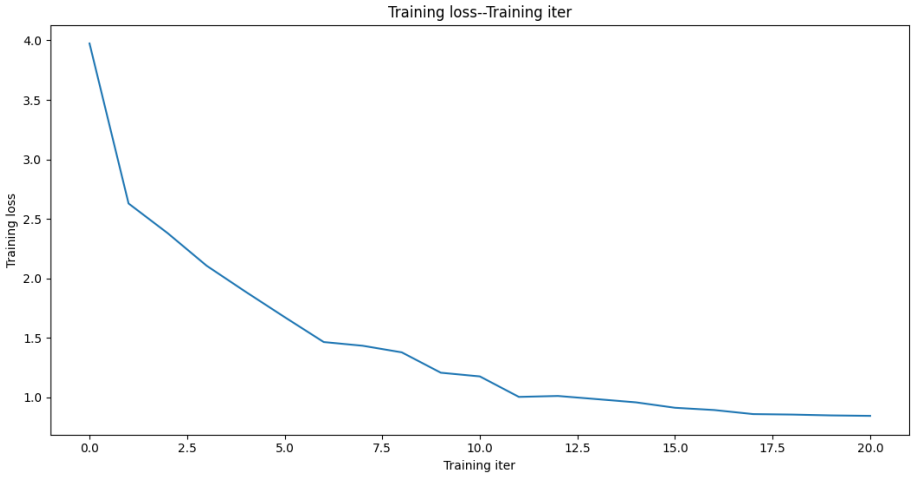
```
sample = np.load("D://AI computer system Lab/MyLab/DLcourse/Lab3/tang.npz",
                 allow_pickle=True)
ix2word = sample['ix2word'].item()
word2ix = sample['word2ix'].item()
# 类定义模型
model = poemv2.PoetryModel(len(ix2word), EMBEDDING_DIM, HIDDEN_DIM)
# 载入参数
model.load_state_dict(torch.load("D://AI computer system Lab/MyLab/DLcourse/Lab3/tang.pth"))
print("Please input the first half sentence of poem!")
# 输入诗句
first = str(input())
#生成诗句
ge_po = poemv2.generate(model, first, ix2word, word2ix)
gen_poetry = ''.join(ge_po)
print("The predicting poem is as follows ： " ,gen_poetry)
print("Thanks for your using and Pls give 5-star appreciation")
```

5、实验结果

结合 colab 上的每次的训练误差，训练了 20 个 epoch，最终得到的 loss 是 0.8452



具体情况如下



诗集的预测情况：
本次实验最大字数设置为了 200!
输入：这门课真得很难
输出：这门课真得很难，近东四十六回节。八千仞中歌一声，重报天师不知定。眉如锦水文系击，令辨一阵电如生。金为寒星夜磨响，白银满城如晓日。报君般狂，今古好。或有宫人，紫苞青乌，相并不如吾。云是君子，相见自吹。秦人赵女口，鞭骨麻女，直为人人心忘，今古应，一穗霏。金舆直御罢且风，汉帝渡头今日暮。邑建二千年，韩家七十三。鴈来过故国，梁属属蕃歌。除去取皆殊士，不知何处相防杀，化负山。

分析：总的来看，这首诗其实还不太行，比如字数还不一致，并非是严格的七言！诗所烘托得氛围也前后不搭调！不过字数前后差不多，说明光字数还是比较满意的!

另外：尝试了输入“人生尽头哪有你”
输出显示：报错

KeyError: '哪'

因为联想到“问渠哪得清如许，为有源头活水来”中有“哪”字，但是显示报错
后来查明原因，使用如下代码

```
def prepareData():
    datas = np.load("D://AI computer system Lab/MyLab/DLcourse/Lab3/tang.npz", allow_pickle=True)
    data = datas['data']
    ix2word = datas['ix2word'].item()
    word2ix = datas['word2ix'].item()
    data = torch.from_numpy(data)
    dataloader = DataLoader(data,
                            batch_size=16,
                            shuffle=True,
                            num_workers=0)
```

```
    return dataloader, ix2word, word2ix

poem_loader, ix2word, word2ix = prepareData()
print(word2ix['哪'])
```

显示结果依旧是 KeyError: '哪'，因此可以断定是 word2ix 内部的问题!

6、实验心得

这次自动写诗实验可以说感触很多，其实一开始需要修改哪些参数才能让情况变得更好，真得一头雾水，尽管老师在课程网站上一再强调诚信原则，但是在时间紧张的情况下（这段时间复习、考试比较密集，也包括了一门片上芯片设计的大作业和包含 51 条指令的 RISC-V 设计及其一整套从前端综合到后端设计的脚本编写），我也只能去 CSDN、知乎、StackOverflow 上寻找自动写诗参数选取的经验。

另外一整个训练过程其实学到很多东西，首先是在 colab 上尝试了运行 gpu 代码，速度显然比我这台电脑运行起来快很多，而且最最关键的部分是下列的代码：

```
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size = 10,gamma=0.1)
```

参考来源：Pytorch 中的学习率衰减及其用法 - 简书 (jianshu.com)

添加这一部分助力 loss 不会反复，否则训练 12 个小时都不可能收敛（Colab 一次最多使用 12 个小时，如果没有充值人民币的话），事实上，一开始代码运行起来实在是令人惊心动魄，因为 loss 始终在 2 左右下不去，后来想到找方法，于是才有了改动学习率的方法。

自动写诗的程序，总的来说，老师给的实验指导代码已经很完整了，其中有文字和程序出入的地方也是一个很好的提示! 比如事实上代码只有一层连接层，但是文字有三层连接层! 另外比如设置了 Config.xxx 变量，后来我才意识到这是一个可以当作配置文件的类变量，但是一开始一头雾水，觉得这个和平时定义一个变量并没有多大区别!

根据诗的结果，已经做到的地方就是能够基本实现写诗了，尽管漏洞百出。

根据诗的结果，没做到的地方还有

1、诗的氛围前后没有控制好；
2、诗的每一句并没有做到都是 7 个字；
3、诗的押韵还存在很大问题，并不符合常见几类诗的平仄；
4、诗的前后句衔接上存在很大的逻辑漏洞；
5、诗整体言之无物；
6、诗整体的风格不够确定，如果可以模仿某一位诗人的风格作诗应该会更好。