#### 作业3：用LSTM写出一首唐诗

班级：数科2004 姓名： 刘一 学号：2020005923

**一、作业要求**

根据github上中文诗词爱好者收集的5万首唐诗数据集，用LSTM写出一首唐诗

**二、运行环境：**

操作系统：

Windows11

软件环境：

python 3.7.7

torch 1.12.0

cuda 11.7

cudnn 7.6.5

**三、操作步骤**

1. 导入相关的库

|  |
| --- |
| import numpy as np  import torch  import torch.nn as nn  from torch.utils.data import DataLoader |

1. 加载数据集，返回Dataloader以及编号与汉字的映射

|  |
| --- |
| def prepareData():      # 读入预处理的数据      # 读入npz文件，对于npz文件，需要指定可以序列化      datas = np.load("tang.npz", allow\_pickle=True)      data = datas['data']      data=data[0:20000]      print(data.shape)      print(f"共有{data.shape[0]}首诗")      # 编号与汉字的映射      ix2word = datas['ix2word'].item()      # 汉字与编号的映射      word2ix = datas['word2ix'].item()      # 转为torch.Tensor      data = torch.from\_numpy(data)      dataloader = DataLoader(data,                           batch\_size = 16,                           shuffle = True,                           num\_workers = 2)      return dataloader, ix2word, word2ix  dataloader, ix2word, word2ix = prepareData() |

1. 利用torch.nn定义的LSTM模型定义古诗训练模型，主要设置了embedding层、lstm层以及输出层的参数和前向传播的过程。

|  |
| --- |
| class PoetryModel(nn.Module):      def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_dim):          # 调用父类的初始化方法          super(PoetryModel, self).\_\_init\_\_()          # 传入的隐层维数          self.hidden\_dim = hidden\_dim          # 设置传入的Word的查找表，将单词转化为向量表示          self.embedding = nn.Embedding(vocab\_size, embedding\_dim)          # 设置LSTM的参数，输入的维度为embedding的维度，输出维度为隐层的维度，设置了2层LSTM单元          self.lstm = nn.LSTM(embedding\_dim, self.hidden\_dim, num\_layers=2)          # 定义线性输入与输出的维度转换关系          self.linear = nn.Linear(self.hidden\_dim, vocab\_size)      def forward(self, input, hidden = None):          # 返回样本大小与样本个数          seq\_len, batch\_size = input.size()          # 根据是否有隐层，定义ht和ct，ht为上一个单词隐层的输出，ct为lstm单元之前的记忆          if hidden is None:              h\_0 = input.data.new(2, batch\_size, self.hidden\_dim).fill\_(0).float()              c\_0 = input.data.new(2, batch\_size, self.hidden\_dim).fill\_(0).float()          else:              h\_0, c\_0 = hidden          # 将输入转化为查找表          embeds = self.embedding(input)          # 将转化后的embeds作为输入传入lstm模型          output, hidden = self.lstm(embeds, (h\_0, c\_0))          # 首先将lstm产生的output转化为序列个数\*批量数x单个字向量的维度          output = self.linear(output.view(seq\_len \* batch\_size, -1))          return output, hidden |

1. 设置训练过程中的超参数

|  |
| --- |
| # 设置超参数  learning\_rate = 5e-3       # 学习率  embedding\_dim = 128        # 嵌入层维度  hidden\_dim = 256           # 隐藏层维度  model\_path = None          # 预训练模型路径  epochs = 1                 # 训练轮数  verbose = True             # 打印训练过程  device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu') |

1. 训练模型，包括输入传入模型，设置优化方法、损失函数，反向传播、参数更新，并且最后将其训练好的模型保存。

|  |
| --- |
| def train(dataloader, ix2word, word2ix):      # 配置模型，是否继续上一次的训练      model = PoetryModel(len(word2ix), embedding\_dim, hidden\_dim)      if model\_path:          model.load\_state\_dict(torch.load(model\_path))      model.to(device)      # 设置优化器      optimizer = torch.optim.Adam(model.parameters(), lr = learning\_rate)      # 设置损失函数      criterion = nn.CrossEntropyLoss()      # 定义训练过程      for epoch in range(epochs):          # 解包批处理的序号和对应的数据          for batch\_idx, data in enumerate(dataloader):              data = data.long().transpose(1, 0).contiguous()              data = data.to(device)              input, target = data[:-1, :], data[1:, :]              output, \_ = model(input)              loss = criterion(output, target.view(-1))              if batch\_idx % 900 == 0 & verbose:                  print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(                      epoch+1, batch\_idx \* len(data[1]), len(dataloader.dataset),                      100. \* batch\_idx / len(dataloader), loss.item()))              # 每次累计梯度前需要清空梯度              optimizer.zero\_grad()              # 反向传播              loss.backward()              # 更新参数              optimizer.step()      # 保存模型      torch.save(model.state\_dict(), 'model.pt') |

1. 设置生成古诗模型的超参数

|  |
| --- |
| # 设置超参数  model\_path = 'model.pt'        # 模型路径  start\_words = '落秋阁前念桃花'  # 唐诗的第一句  max\_gen\_len = 125                # 生成唐诗的最长长度 |

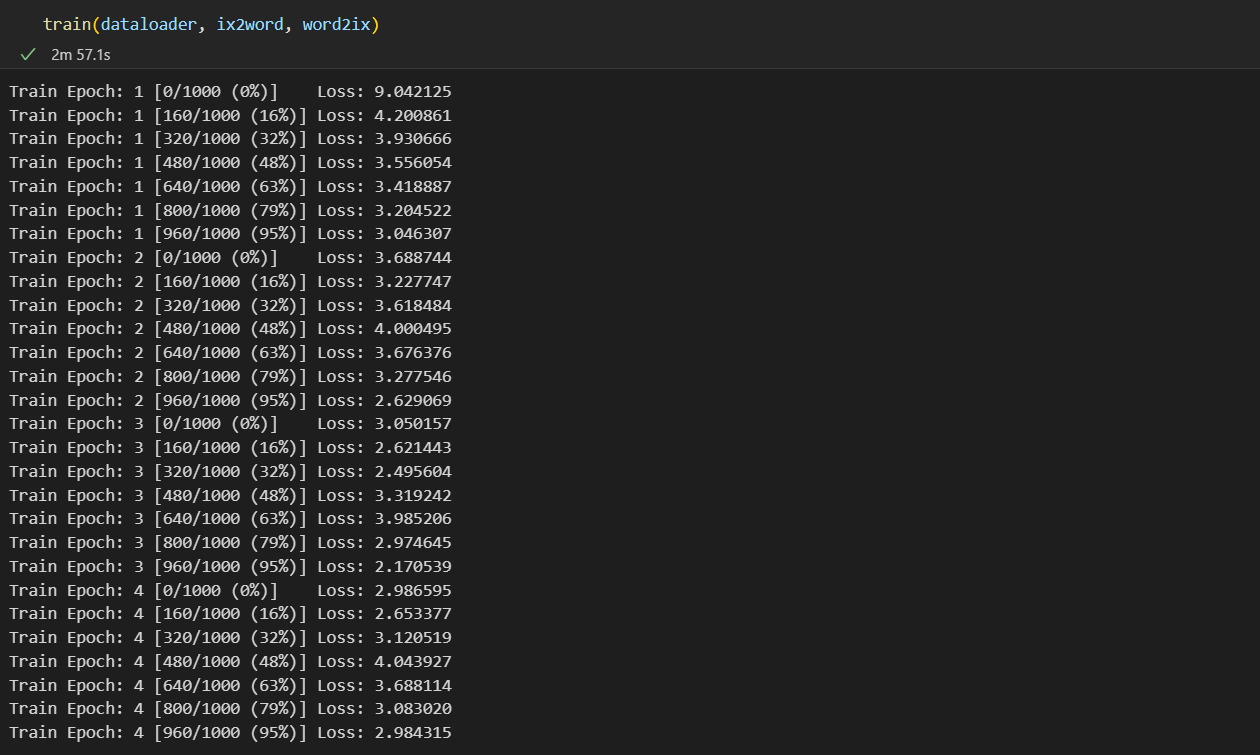
1. 利用之前训练好的模型生成古诗

|  |
| --- |
| def generate(start\_words, ix2word, word2ix):      # 读取模型      model = PoetryModel(len(word2ix), embedding\_dim, hidden\_dim)      model.load\_state\_dict(torch.load(model\_path))      model.to(device)        # 读取唐诗的第一句      results = list(start\_words)      start\_word\_len = len(start\_words)        # 设置第一个词为<START>      input = torch.Tensor([word2ix['<START>']]).view(1, 1).long()      input = input.to(device)      hidden = None      # 生成唐诗      for i in range(max\_gen\_len):          output, hidden = model(input, hidden)          # 读取第一句          if i < start\_word\_len:              w = results[i]              input = input.data.new([word2ix[w]]).view(1, 1)          # 生成后面的句子          else:              top\_index = output.data[0].topk(1)[1][0].item()              w = ix2word[top\_index]              results.append(w)              input = input.data.new([top\_index]).view(1, 1)          # 结束标志          if w == '<EOP>':              del results[-1]              break        return results |

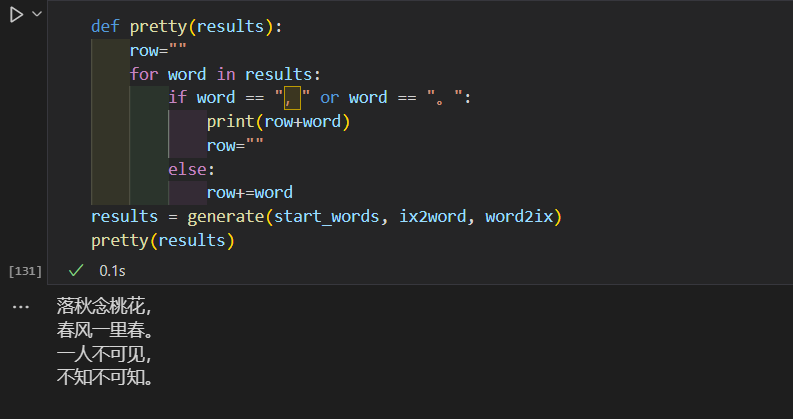
1. 生成古诗

|  |
| --- |
| def pretty(results):      row=""      for word in results:          if word == "，" or word == "。":              print(row+word)              row=""          else:              row+=word  results = generate(start\_words, ix2word, word2ix)  pretty(results) |

**四、结果分析**



上图为训练过程中的迭代次数以及更新次数，并且计算得到每次的损失值。这里只取了1000首诗词。



上图展现的是生成的是给定第一句后，生成的古诗。