文本生成（以 bigram 与 LSTM 为方法）

ISBD@RUC

辛家辉

2022-06-18

目录

1 引言

自然语言处理 (natural language processing) 是工业界与学术界共同关心的方向，不论是推荐系统 (recommended system)、聊天机器人 (chat bot) 还是自动翻译 (automatic translation)，都属于这一方向。

文本生成 (text generation) 是自然语言处理中一个重要的研究领域，有着巨大的应用需求。依据输入与输出的维数，文本生成可分为一对多或多对多的任务：一对多是指一个输入对应多个输出，比如生成古诗或文段时先指定一个起始词；多对多可以是输入一段文字、一张图像、一个数据集，输出一段文本，可以是提取的摘要、语义或是描述。

本报告只考虑文本生成，重点考虑五言古诗生成，考虑了 bigram 模型和 LSTM 方法。其中 bigram 模型是经典的概率模型方法，LSTM 是循环神经网络 RNN 的变形。通过使用 LSTM 可以延伸到几乎所有的文本生成领域。

2 方法及其结果

2.1 bigram 模型

bigram 模型 (见Figura 1) 只使用给定当前的文字的条件下，下一个文字的条件概率。通过对语料库中条件概率的经验估计，容易实现 bigram。

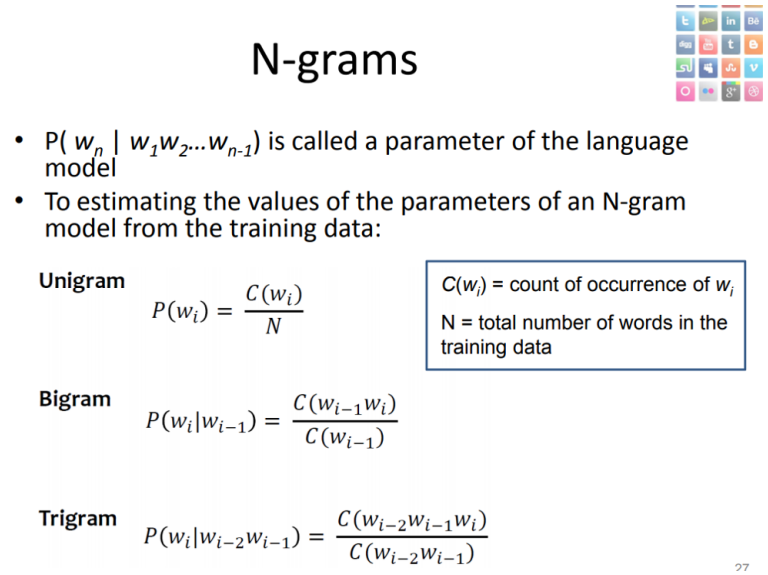


Figure 1: Unigram, bigram and trigram. (from internet)

我在老师的 R code 基础上，使用了唐诗三百首和全唐诗两个语料库实现 bigram，划分

每一半句（以"，" 或"。" 结尾）作为一个独立的训练样本。

但是考虑到这样把前半句与后半句等同视之，可能会损失一些韵律上的要求。所以额外地，把全唐诗分为前半句与后半句两个语料库分别计算 bigram1 (前半句) 和 bigram2 (后半句)，然后生成时分别使用不同的 bigram 生成前半句与后半句。

其中 bigram 模型参数对比见Tabla 1。[1] [2] [3]

Table 1: 两个语料库下的 bigram 模型参数对比

	唐诗三百首	全唐诗	全唐诗（分前后句）
原始行数	2536	187771	-
含诗句的行数	1465	133671	-
诗句数	1465	212075	-
不同汉字的数量	2361	7883	-
bigram 大小	1MB	54.5MB	32.3MB (每个 bigram)
创建 bigram 的时间	$< 2s$	$\approx 20min$	$10min \sim 20min$
生成四句五言诗的时间	0.08 (0.01)s	1.17 (0.12)s	0.74 (0.05)s
生成中是否会出现 bug	是	否	否

使用不同语料库 bigram 生成的诗句如下

- 唐诗三百首:

  鹤自惭见此，白头已合千。去何劳歌吹，人与云外见。

  开晚日满林，片孤城阙在。荒园花木叶，每忆旧业本。

  微微星斗升，湘竹林期来。归故乡心逐，上帝乡关身。

  今若要自高，浮云中相似。静处处尽老，近臣衣已矣。

- 全唐诗:

  西极浦月上，蜀国旧时情。南亭不敢住，寒雁一言自。

  官高挂帆归，荣枯树色黄。梦中不相伴，馀清风远征。

  何劳劳问津，云间岁暮钟。与故园中事，已去水声如。

  道春风吹毛，塞色正相随。何曾对青云，一为君住得。

[1] 运行时间在本人电脑计算，存在差异

[2] 如果样本量太小则容易出现其后面出现任何字的条件概率均为 0，使 bigram 生成程序崩溃

[3] 生成四句五言诗的时间：运行十次，括号外是均值，括号内是标准差

- 全唐诗（分前后句）:

  犹似雪满空，忽闻铃肠断。未识君歌声，海为远连天。

  来途无端嫁，落家营共徘。故人收拾理，不成败由来。

  海上书初见，十年事且未。风尘心云山，欲曙色诏才。

  孤云霄多病，忽然离别寻。坐看上林叟，遥夜雨晚见。

唐诗三百首生成的不通顺之处甚多，无法勾连意象。

全唐诗生成的意象可以勾连，除某些字莫名其妙以外基本通顺，如“西极浦月上，蜀国旧时情。”“梦中不相伴，馀清风远征。”但是有些句子前后顺序有些古怪，如“南亭不敢住，寒雁一言自。”可以换为“寒雁一言自，南亭不敢住。”

全唐诗（分前后句）虽然生成的前后顺序保持，但仍然语义不通。

所以我考虑使用 LSTM 进行改良。

2.2 LSTM

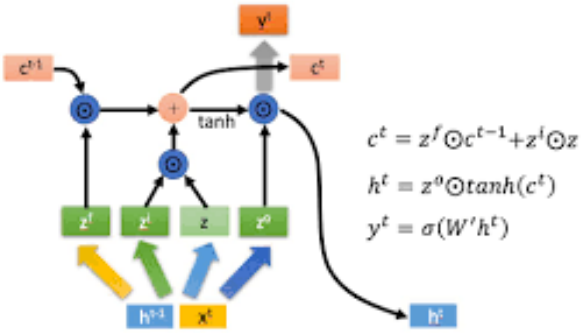


Figure 2: lstm illusration from zhihu

LSTM

LSTM 是 RNN 的变形，使用状态 c_t 来记住过去的信息，额外输出一个隐状态 h_t。输入一个序列 $x_1, x_2 \cdots, x_T$ 后初始化 h_0，可以由 (x_1, h_0) 计算 (y_1, h_1)，并递推得到 $\{y_i, h_i\}_{i=1}^{T}$。在这个递推过程中，均使用同一个神经网络的参数。其在许多 NLP 任务上取得了巨大的成功。

我基于DRSY@github的 python code，主要做了如下几点改变，在 google colab (可以使用免费的 GPU) 运行。

- 使用全唐诗收录的李世民共 481 句五言诗作为语料库，共 1362 个不同的字。[4]
- 使用 GPU 训练 100 个 epoch（比使用 CPU 快十倍左右）
- 使用 Adam 作为优化方法

[4]由于 LSTM 训练速度偏慢，原 code 只使用了 50 句七言古诗。

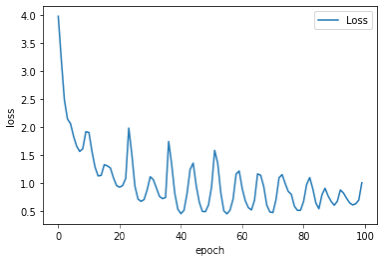
- 调整步长 (learning rate)，惩罚系数 (weight decay)，词向量维数 (embedding size)，LSTM 的隐藏层维数 (hidden seize)，也就是调参
- 每十个 epoch 保存一次模型，方便比较

所使用的神经网络模型比较简单，从输入到输出经过如下过程

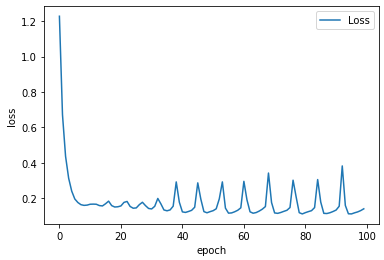
- 把字转换成 one-hot 向量输入
- embed(嵌入层)：1363 维 one-hot 向量映射到 128 维词向量
- lstm：x_t 词向量为 128 维，h_t 隐向量为 256 维
- fc1(全连接层)：256 维（隐向量）映射到 1363 维（one-hot 向量）
- 最后做 ReLu + log_softmax 得到输出

我使用了四组 (learning rate, weight decay) 组合，分别是 (1e-2,1e-4), (1e-3,1e-4), (1e-2,1e-5), (1e-3,1e-5)。其中 learning rate 是每次更新的步长，weight decay 是为了防止过拟合的正则化系数。其 train loss 曲线如Figura 3。

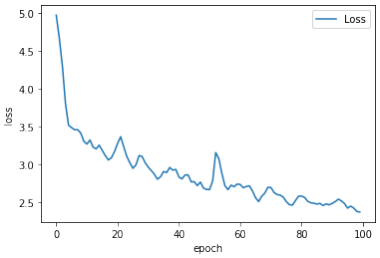
步长 learning rate=1e-3 可以在几十个 epoch 下把 loss 稳定在 0.1 左右 (Figura 3b,Figura 3d)，而 learning rate=1e-2(Figura 3a,Figura 3c) 则只能维持在 2.5 左右难以进一步下降。weight decay 影响过拟合能力，如同样令 learning rate=1e-3(Figura 3b,Figura 3d)，更小的 weight decay 下 (Figura 3d)loss 下降更快、但震荡更大，意味着过拟合程度更明显。



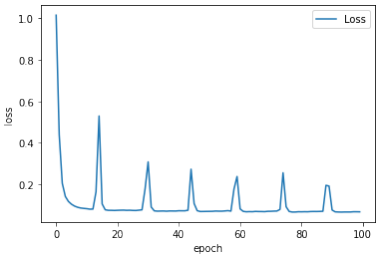
(a) (learning rate, weight decay)=(1e-2,1e-4)



(b) (learning rate, weight decay)=(1e-3,1e-4)



(c) (learning rate, weight decay)=(1e-2,1e-5)



(d) (learning rate, weight decay)=(1e-3,1e-5)

Figure 3: 不同参数下的损失曲线

训练得到的模型存储为".pkl" 文件，大小为 3.5MB。GPU 版本的模型四次训练的时间均在 110s 与 120s 之间，而 CPU 版本大于 1000s。每组参数下，载入 11 个模型（训练 $1, 11, \cdots, 100$ 个 epoch 的模型）并生成八行十六句五言诗，花费总时间均小于 1s。

展示不同参数不同个数 epoch 下生成的诗句，碍于篇幅只展示 (learning rate, weight decay)=(1e-2,1e-4) 与 (learning rate, weight decay)=(1e-3,1e-5) 两组。[5]

- (learning rate, weight decay)=(1e-2,1e-4)
  - 1 epoch

    冻云时驻道，隔岫有云。前王云四时，无景鸟花生。

    楚云惊玉阙，菊散一丛金。散影玉阶柳，梅上朝春空。

    循气冰初镜，岩菊方丛欢。继文遥天地，眺逐散一丛。

    阶柳一花生，梅柳散碧空。历文文阴白，玉溜。
  - 31 epochs

    崇文时低飞，骋低飞还。。远岫飘云叶，迷路飞云浮。

    乔柯嗸娇鸟，低枝映美人。疏黄一鸟弄，树冷间渐尘。

    馀花攒旧涧，宿雾足朝烟。兹俯高层，有岸隐平畴。

    金鞍移上苑，玉勒骋平畴。绮峰含翠雾，照日蕊红林。
  - 61 epochs

    径细无全磴，松小未含烟。二仪初创象，三才柳位分。

    怀卑运深广，持满守灵长。连甍岂一拱，飞魄岁中舒。

    太液仙舟迥，西园隐遥芳。作异甘泉日，泉石且娱心。

    罗绮昭阳殿，芬芳玳瑁筵。条风飘献节，灰风知芳春。
  - 91 epochs

    讨论穷义府，看核披经笥。岭烟深明月，夕影轻重金。

    兽忙投密树，鸿惊起高罗。碧林青旧竹，绿沼翠新苔。

[5] 🄵属于汉字生僻字无法显示

色洒妆台粉，花飘绮席衣。九龙蟠焰动，绣柱奚光浮。

带岫凝全碧，障霞隐半红。提壶菊花岸，高蓉芙蓉池。

- (learning rate, weight decay)=(1e-3,1e-5)
  - 1 epoch

    蕊间飞禁苑，鹤处舞伊川。石鲸分玉溜，劫烬隐平沙。

    抽思滋泉侧，飞想傅岩中。雕宫静龙漏，绮阁宴公侯。

    惨日映峰沉，愁云随盖转。水花翻照树，堤兰倒插波。

    华林满芳景，洛阳遍阳春。回銮游福地，极目玩芳晨。
  - 31 epochs

    披襟欢眺望，极目畅春情。岩廊罢机务，崇文聊驻辇。

    叠松朝若夜，复岫阙疑全。上弦明月半，激箭流星远。

    高轩暖春色，邃阁媚朝光。广待淳化敷，方嗣云亭响。

    瀑流还响谷，猿啼自应虚。石鲸分玉溜，劫烬隐平沙。
  - 61 epochs

    心随朗日高，志与秋霜洁。以兹游观极，悠然独长想。

    积善忻馀庆，畅武悦成功。镇下千行泪，非是为思人。

    蝉啼觉树冷，萤火不温风。浇俗庶反淳，替文聊就质。

    架海波澄镜，韬戈器反农。遍野屯万骑，临原驻五营。
  - 91 epochs

    未佩兰犹小，无丝柳尚新。观仪不失序，遵礼方由事。

    代马依朔吹，惊禽愁昔丛。紫庭文🄵满，丹墀衮绂连。

    🄵莺犹响殿，横丝正网天。砌冷兰凋佩，闺寒树陨桐。

    六五诚难继，四三非易仰。黄莺弄渐变，翠林花落馀。

在学习率与正则化参数均较大的情况下，loss 降低较慢，所以在训练较少 epoch 的时候，会出现生成的"，" 与"。" 错位。但在训练较多 epoch 时，非常通顺，可以学会前后句的对仗，比如训练 31 epochs 的“绮峰含翠雾，照日蕊红林。”，训练 61 epochs 的“罗绮昭阳殿，芬芳玳瑁筵。”，训练 91 epochs 的“带岫凝全碧，障霞隐半红。”

在学习率与正则化参数均较小的情况下，loss 降低较快，过拟合明显。就算只训练 1 epoch，也不会出现生成的"，" 与"。" 错位，还有对仗“惨日映峰沉，愁云随盖转。水花翻照树，绮阁宴公侯。”。训练更多的 epoch 虽然也有对仗，如 61 epochs 的“遍野屯万骑，临原驻五营。”；但对仗往往不再工整，更像人（李世民：正是本人）所写，如 31 epochs 的“上弦明月半，激箭流星远。”和 61 epochs 的“镇下千行泪，非是为思人。蝉啼觉树冷，萤火不温风。”都可以说是佳句。

模仿写诗不需要太重视过拟合的问题，太像李世民未尝不是一件好事。

3 总结

本报告给出了使用 bigram 和 LSTM 两种模型的文本生成实践，用于自动生成五言诗句。其中 bigram 是经典的强模型方法，或者说是模型驱动 (model-driven) 的方法；LSTM 是近十年来流行的神经网络方法，是弱模型或者说数据驱动 (data-driven) 的方法。

bigram 对于大量文本训练速度较慢，模型较大，但效果仍然较差（由于只考虑前文一个字难以做到通顺的语句）。LSTM 在极少的文本（481 行）下训练速度尚可，模型较小，可以发掘出诗句的模式，比如对仗、词与词之间的意象依赖、标点的位置，效果非常好。

神经网络作为计算框架，以梯度下降为核心的优化算法，有大量的参数可以拟合复杂的数据集，效果在某些任务上表现极好。如今的算法包几乎不需要数学上的推导和领域知识 (domain knowledge)，直接调包可以直接迁移到别的任务上，也是被广泛使用的原因。

但是，神经网络的可解释性问题没有被深入理解，庞大的参数模型和人脑的关系难以理清。古典的科学和哲学不都是尽可能用少的参数对世界的建模吗？更深、更大的神经网络与此相反，于是难以解释神经网络的巨大成功。

4 代码

4.1 R code for bigram

```
#fileName1 <- "唐诗三百首.txt"
fileName1 <- "全唐诗.txt"
#fileName1 <- "宋词三百首.txt"
#setwd("C:/Users/Sai Li/Downloads")
TS <- readChar(fileName1, file.info(fileName1)$size, useBytes = T)
nchar(TS)
#head(TS)
#fileName2 <- "宋词三百首.txt"
#SC <- readChar(fileName2, file.info(fileName2)$size)
#nchar(SC)
#SC.corpus<-strsplit(SC, split='\r\n')[[1]]
TS.corpus<-strsplit(TS, split='\r\n')[[1]]
len.ts <- length(TS.corpus) #2536
headlines<-grep("[0-9]", TS.corpus)
TS.corpus1<-list()
j=1
for(i in 1:(len.ts-3)){
  #全唐诗 nchar(TS.corpus[i])>=16; 唐诗三百首>=12
  if(!(i %in% headlines) & nchar(TS.corpus[i])>=16 &
  nchar(TS.corpus[i])%%2==0){#not headline or empty line
    TS.corpus1[[j]]<-TS.corpus[i]
    j=j+1
  }
}
length(TS.corpus1) #1465
head(TS.corpus1)
filter_func<-function(x){
  strsplit(x,"\\s+|, |。")[[1]]
}
```

```
str.dic<-lapply(TS.corpus1,filter_func)



str.dic<-unique(unlist(str.dic)) #2362 unique chars

str.dic<-str.dic[-1]

str.dic<-c("s",str.dic)


p<-length(str.dic)

char.dic<-paste0(str.dic,collapse="")
char.dic<-(strsplit(char.dic,"")[[1]])
char.dic<-unique(char.dic)
q<-length(char.dic)

bigram <- vector(mode = "list", length = q)

for(i in seq(2,p,2)){#every sentence contains two str.dic
  char.cur1=strsplit(str.dic[i],split="")[[1]]
  char.cur<-c("s",char.cur1)
  char.cur2=strsplit(str.dic[i+1],split="")[[1]]
  char.cur<-c(char.cur,"s",char.cur2)
  len.cur=length(char.cur)
  for(k in 1:(len.cur-1)){
    if(char.cur[k+1]=='s'){next} #the end of the sentence
    loc<-which(char.dic==char.cur[k])
    bigram[[loc]]<-c(bigram[[loc]],char.cur[k+1])
  }
}


#set.seed(1234)
```

```
write.poem<-function(bigram, char.dic,n.char=5, n.rows=4){
  word.gen<-function(char.vec, rand=T){
    tab<-table(char.vec)
    df<-data.frame(name=names(tab), prob=as.matrix(tab)/length(char.vec))
    #add
    df$prob=df$prob+1e-6
    #end add
    df$prob<-df$prob/sum(df$prob)
    if(rand){
      sub1<-which(df$prob>=quantile(df$prob,0.9))
      prob1<-df$prob[sub1]/sum(df$prob[sub1])
      sel.char<-sub1[which.max(rmultinom(1,1,prob1))]
    }else{
      sel.char<-which.max(df$prob)
    }
    df$name[sel.char]
  }
  poet<-vector(mode = "list", length = n.rows)
  s.loc<-which(char.dic=='s') #start
  for(i in 1: n.rows){
    poet[[i]]<-word.gen(char.vec=bigram[[s.loc]])
    for(j in 2:n.char){
      pre.loc<-which(char.dic==poet[[i]][j-1])
      poet[[i]]<-c(poet[[i]],word.gen(bigram[[pre.loc]]))
    }
  }
  poet
}
write.poem(bigram, char.dic, n.char=5)


# Two bigram, bigram1 for the former sentence and bigram2 for the latter

bigram1 <- vector(mode = "list", length = q)
bigram2 <- vector(mode = "list", length = q)
```

```
for(i in seq(2,p,2)){#every sentence contains two str.dic
  char.cur1=strsplit(str.dic[i],split="")[[1]]
  char.cur1<-c("s",char.cur1)
  char.cur2=strsplit(str.dic[i+1],split="")[[1]]
  char.cur2<-c("s",char.cur2)
  len.cur1=length(char.cur1)
  len.cur2=length(char.cur2)
  for(k in 1:(len.cur1-1)){
    loc<-which(char.dic==char.cur1[k])
    bigram1[[loc]]<-c(bigram1[[loc]],char.cur1[k+1])
  }
  for(k in 1:(len.cur2-1)){
    loc<-which(char.dic==char.cur2[k])
    bigram2[[loc]]<-c(bigram2[[loc]],char.cur2[k+1])
  }
  if(i%%10000==0)print(i)#loop print
}


#set.seed(1234)
my.write.poem<-function(bigram1,bigram2, char.dic,n.char=5, n.rows=4){
  word.gen<-function(char.vec, rand=T){
    tab<-table(char.vec)
    df<-data.frame(name=names(tab), prob=as.matrix(tab)/length(char.vec))
    #add
    df$prob=df$prob+1e-6
    #end add
    df$prob<-df$prob/sum(df$prob)
    if(rand){
      sub1<-which(df$prob>=quantile(df$prob,0.9))
      prob1<-df$prob[sub1]/sum(df$prob[sub1])
      sel.char<-sub1[which.max(rmultinom(1,1,prob1))]
    }else{
      sel.char<-which.max(df$prob)
```

```
    }
    df$name[sel.char]
  }
  poet<-vector(mode = "list", length = n.rows)
  s.loc<-which(char.dic=='s') #start
  for(i in seq(1, n.rows,2)){
    poet[[i]]<-word.gen(char.vec=bigram1[[s.loc]])
    for(j in 2:n.char){
      pre.loc<-which(char.dic==poet[[i]][j-1])
      poet[[i]]<-c(poet[[i]],word.gen(bigram1[[pre.loc]]))
    }
  }
  s.loc<-which(char.dic=='s') #start
  for(i in seq(2, n.rows,2)){
    poet[[i]]<-word.gen(char.vec=bigram2[[s.loc]])
    for(j in 2:n.char){
      pre.loc<-which(char.dic==poet[[i]][j-1])
      poet[[i]]<-c(poet[[i]],word.gen(bigram2[[pre.loc]]))
    }
  }
  poet
}
my.write.poem(bigram1,bigram2, char.dic, n.char=5)

#select 李世民 poems but also need addtional comparison

which(str.dic[1:1e4]=="隔岫断猿吟")#964
lstm<-c()
for(i in seq(1,964,2)){
  lstm<-c(lstm,paste0(str.dic[i],", ",str.dic[i+1],"。"))
}
writeLines(lstm,"lstm.txt",useByte=T)
```

4.2 Python code for LSTM

```
# -*- coding: utf-8 -*-
"""poetry.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1DcSVQ0USXRrhsX10KqFoXzBGduIKdc
"""

# Commented out IPython magic to ensure Python compatibility.
from google.colab import drive
drive.mount('/content/drive/')
# %cd /content/drive/MyDrive/poetry

'''
    poetry-gen
'''
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import random
import matplotlib.pyplot as plt
import time
import os
import sys

raw_data=[]
with open('lstm.txt','r') as f:
        for line in f:
                raw_data.append(line.strip('\n'))
print(raw_data)

first_char = set()
```

```
word_dict = set()

for sentence in raw_data:
    first_char.add(sentence[0])
    for char in sentence:
        if char not in word_dict:
            word_dict.add(char)
word_dict.add('eos')#end of sentence
word2indx = {word:idx for idx,word in enumerate(sorted(word_dict))}#diction
indx2word = {idx:word for idx,word in enumerate(sorted(word_dict))}
print('vocabulary size:', len(word2indx))
print('poem lines:', len(raw_data))

def make_one_case(sentence) -> (torch.tensor, torch.tensor):#create input a
    sentence = list(sentence)
    sentence.append('eos')
    length = len(sentence)
    inputs = []
    targets = []
    for i in range(1, length):
        pre = sentence[i-1]
        nex = sentence[i]
        inputs.append(word2indx[pre])
        #construct input
        targets.append(word2indx[nex])
        #construct output
    return torch.tensor(inputs, dtype=torch.long),
    torch.tensor(targets, dtype=torch.long)

class Net(nn.Module):

    def __init__(self, vocab_size, embedding_size, hidden_size):
        super().__init__()
        self.hidden_size = hidden_size
```

```
        self.embed = nn.Embedding(vocab_size, embedding_size)
        self.lstm = nn.LSTM(input_size=embedding_size,
        hidden_size=self.hidden_size)
        self.fc1 = nn.Linear(self.hidden_size, vocab_size)

    def forward(self, sentence, hidden):
        seq_length = sentence.size()[0]#length per sentence (sequence)
        embeds = self.embed(sentence).view(seq_length, 1, -1)
        # after embbeding reshape as (time_step, batch, embedding_size)
        lstm_out, hidden = self.lstm(embeds, hidden)
        output = F.relu(self.fc1(lstm_out.view(seq_length, -1)))
        #relu activation
        output = F.log_softmax(output, dim=1)#softmax to get prob
        return output, hidden

    def init_hidden(self):
        return (torch.zeros(1,1,self.hidden_size).cuda(),
                torch.zeros(1,1,self.hidden_size).cuda())

# hyperparameters
VOCAB_SIZE = len(word2indx)
HIDDEN_SIZE = 256
EMBEDDING_SIZE = 128

model = Net(VOCAB_SIZE, EMBEDDING_SIZE, HIDDEN_SIZE).cuda()
optimizer = optim.Adam(model.parameters(), lr=0.01, weight_decay=0.00001)
loss_function = nn.NLLLoss()

print(model)
model

train = True
if os.path.exists('params_last.pkl'):
    model.load_state_dict(torch.load('params_last.pkl'))
    train = False
```

```
if train == False:
    ans = input(r'Trained model already exists, still training? (Y/N):')
    if ans == 'Y' or ans == 'y':
        train = True


# SGD batch_size=1
if train:
    print('train start')
    epoch = 100
    batch_size = len(raw_data)
    Loss = []
    start_time = time.time()
    for i in range(epoch):
        _loss = 0
        indxs = list(range(batch_size))
        random.shuffle(indxs)
        for j in indxs:
            inputs, targets = make_one_case(raw_data[j])
            inputs=inputs.cuda()
            targets=targets.cuda()
            hidden = model.init_hidden()

            outputs, hidden = model(inputs, hidden)
            loss = loss_function(outputs, targets)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            _loss += loss.item()
        _loss /= batch_size
        print('epoch:{}, loss:{:.2f}'.format(i, _loss))
        if i % 10 == 0:
            torch.save(model.state_dict(), 'params_%depoch.pkl'%i)
        Loss.append(_loss)
    end_time = time.time()
```

```
    print('train done, cost{}s'.format(end_time-start_time))
    torch.save(model.state_dict(), 'params_last.pkl')
    plt.plot(range(len(Loss)), Loss, label='Loss')
    plt.legend()
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.show()

def test():
  def sample(startword, max_len=11) -> str:
    if startword not in word_dict:
        return 'null'
    inputs = torch.tensor([word2indx[startword]], dtype=torch.long).cuda()
    #inpts=inputs.cuda()
    output_poetry = startword
    hidden = model.init_hidden()
    for i in range(max_len):
        outputs, hidden = model(inputs, hidden)
        topv, topi = outputs.data.topk(1)
        w = topi[0][0].item()
        word = indx2word[w]
        if word == 'eos':
            break
        else:
            output_poetry += word
        inputs = torch.tensor([w], dtype=torch.long).cuda()
    return output_poetry

  nums = 8
  for i in range(nums):
      word = random.sample(list(first_char), 1)[0]
      print(sample(startword=word, max_len=11))

ans = input(r'Input id of epoch? (-1 for last):')
if ans == '-1':
```

```
    model.load_state_dict(torch.load('params_last.pkl'))
if int(ans)%20==0:
    model.load_state_dict(torch.load('params_{}epoch.pkl'.format(ans)))

start=time.time()
for i in range(10):
    print(i)
    model.load_state_dict(torch.load('params_{}epoch.pkl'.format(i*10)))
    test()
print("last")
test()
end=time.time()
print(end-start)

print(model)
```