智能计算系统技术文档

1952395沈韬 | Tao Shen(ShaoCHi)

背景介绍

```
开发环境
Python 3.9
Tensorflow 2.0
```

当前古诗句生成任务大多基于单一的循环神经网络（RNN）结构，在生成时需事先给定一个起始字，然后以该起始字为基础进行古诗句的生成，生成过程的可控性较差，往往达不到预期效果。同时，对于NLP（自然语言处理）、情感分析等一般采用RNN结构进行处理，所以这里采用LSTM模型进行实现

基于深度学习的古诗自动生成系统是通过神经网络对数据集进行学习和语义分析后训练出模型，在该模型上对于用户的输入进行响应从而生成对应的古诗。模型可以根据用户的输入生成古诗，例如藏头诗、补全古诗等，生成的古诗格式是保证正确的。

该模型主要分为服务于LSTM神经网络的数据预处理模块、LSTM神经网络模块和GUI模块。数据预处理模块中对于4万多首古诗进行预处理，转化为One-Hot编码，神经网络才能进行矩阵激素爱你、学习，LSTM神经网络模块是最核心的模块，需对激活函数、损失函数进行选取，参数优化等操作。

网络结构分析

- 传统RNN

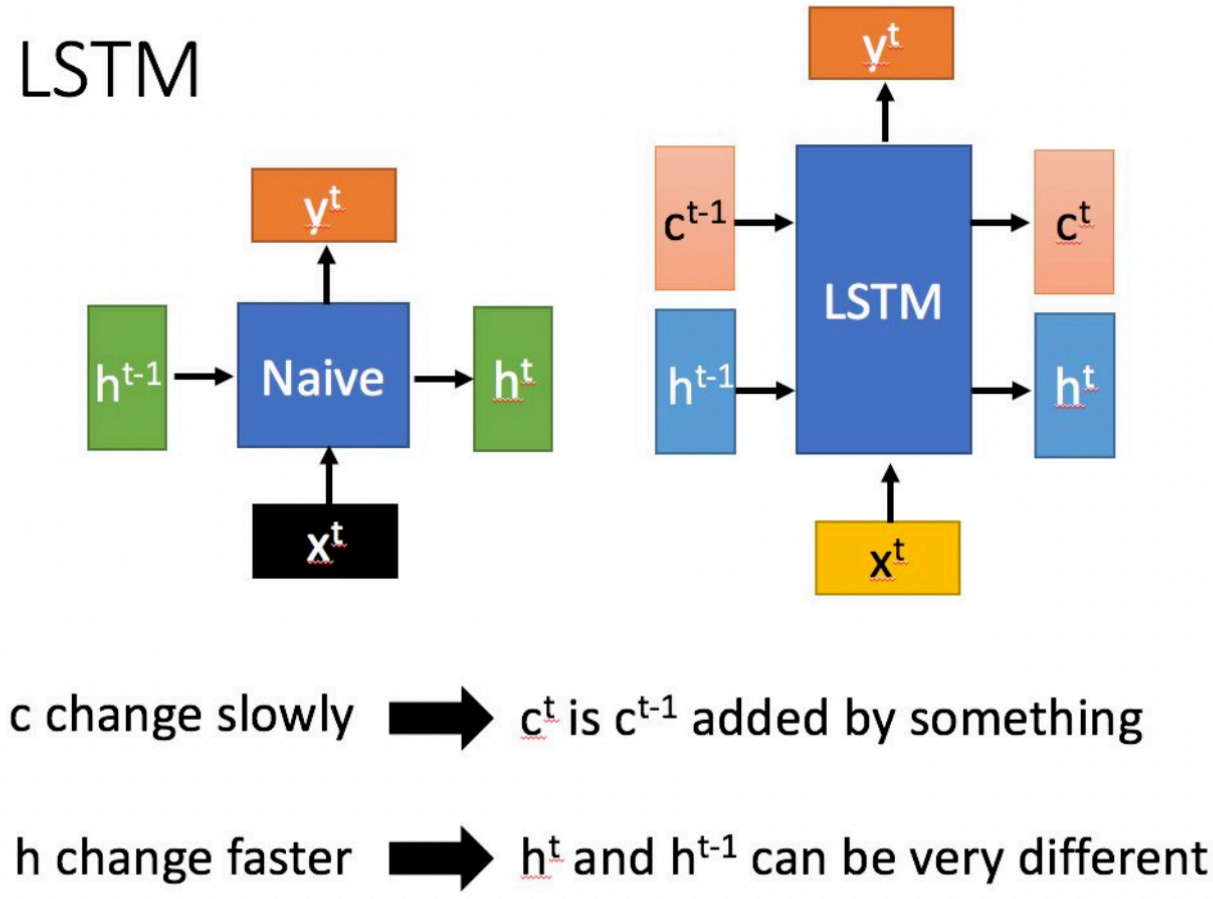
  循环神经网络（Recurrent Neural Network，RNN）是一种用于处理序列数据的神经网络。相比一般的神经网络来说，他能够处理序列变化的数据。比如某个单词的意思会因为上文提到的内容不同而有不同的含义，RNN就能够很好地解决这类问题。

  RNN存在梯度爆炸和梯度消失的问题，所以RNN只能适应于短期神经网络记忆

- LSTM

  长短期记忆（Long short-term memory, LSTM）是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

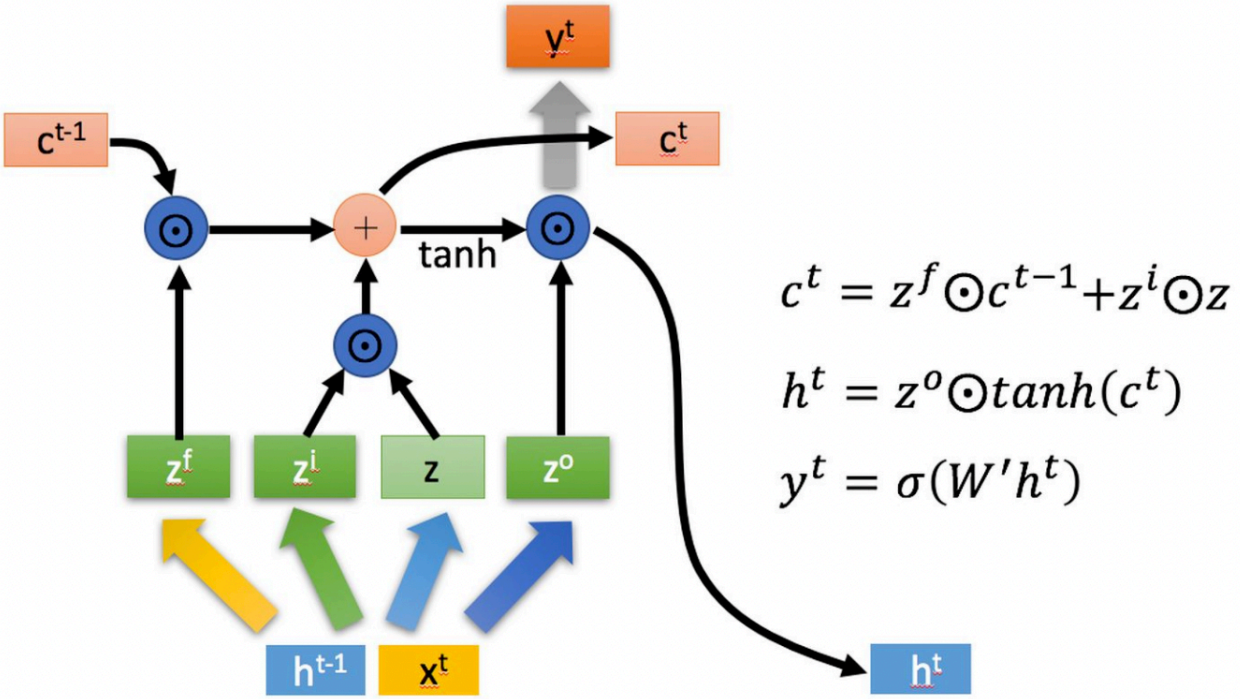
- RNN与LSTM对比



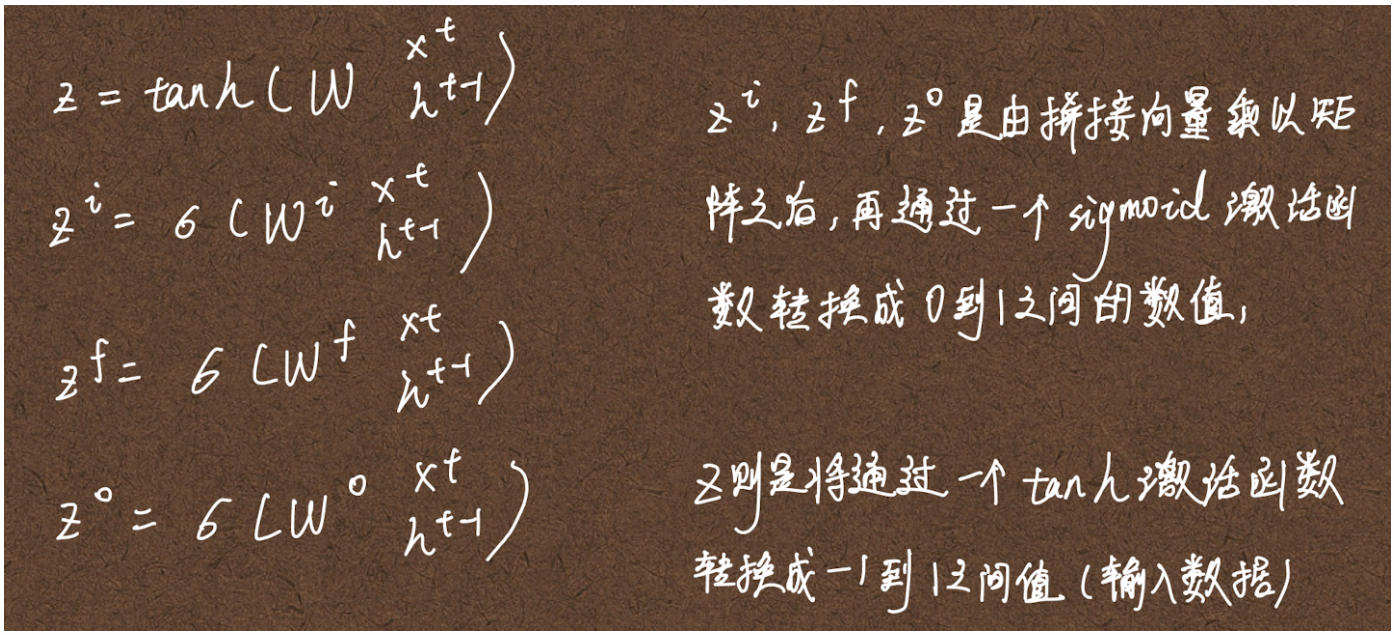
左边的网络结构为普通RNN网络结构，右边即为LSTM论文结构

```
可以看出RNN网络结构仅有一个状态的输入，即上一节点的h^(t-1),其f函数一般采用tanh和Relu函数
```

```
在LSTM网络结构中，存在两种传输状态，C^(t-1)=>单元状态（cell state）
                                    h^(t-1)=>隐蔽状态（hidden state）
```



其中h^(t-1)和X^t将拼接得到四个状态，其含义分别如下



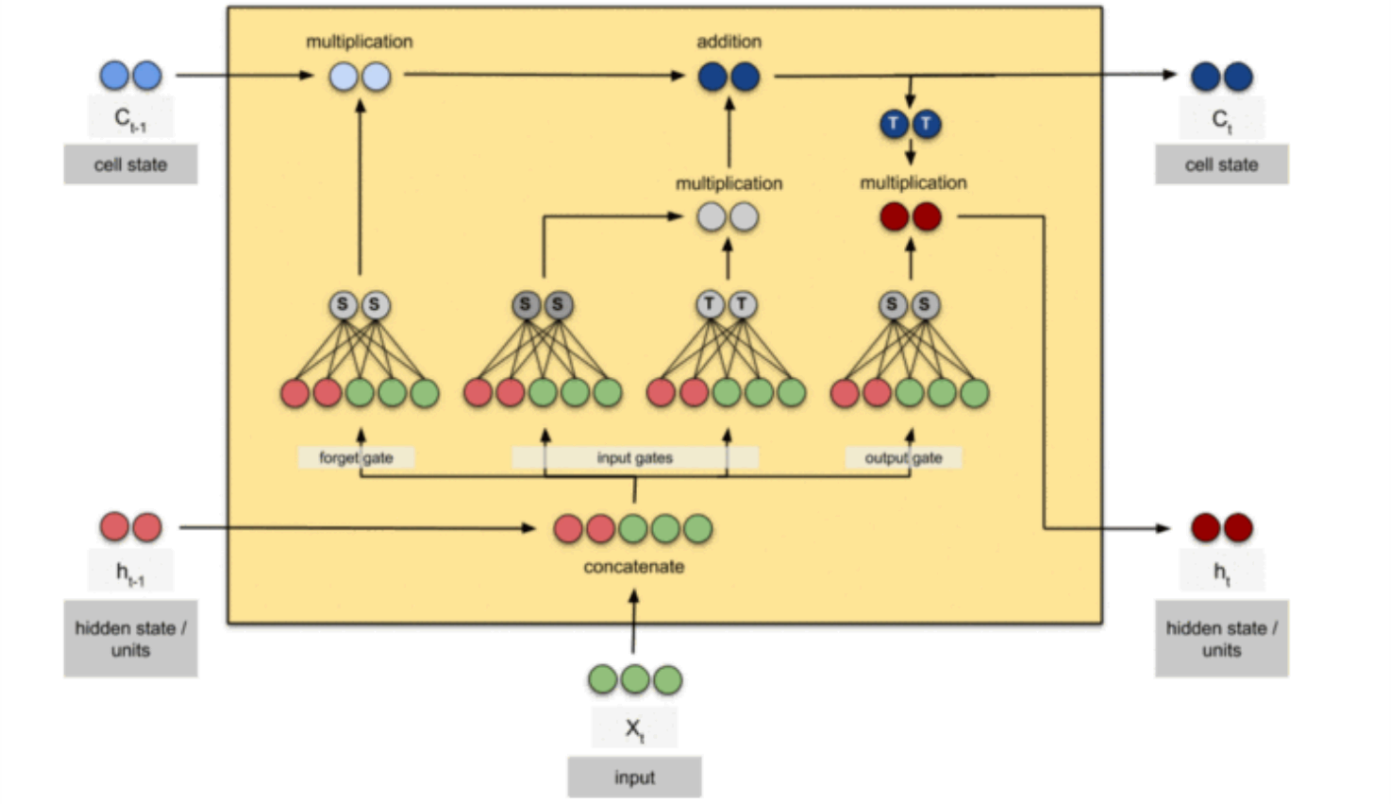
其使用如下（LSTM内部的三个阶段）

- 忘记阶段。这个阶段主要是对上一个节点传进来的输入进行选择性忘记。简单来说就是会 “忘记不重要的，记住重要的”。 具体来说是通过计算得到的 z^f （f表示forget）来作为忘记门控，来控制上一个状态的 c^{t-1} 哪些需要留哪些需要忘。
- 选择记忆阶段。这个阶段将这个阶段的输入有选择性地进行“记忆”。主要是会对输入 x^t 进行选择记忆。哪些重要则着重记录下来哪些不重要，则少记一些。当前的输入内容由前面计算得到的 z 表示。而选择的门控信号则是由 z^i （i代表information)来进行控制。

将上面两步得到的结果相加，即可得到传输给下一个状态的C^t。也就是上图中的第一个公式。

- 输出阶段。这个阶段将决定哪些将会被当成当前状态的输出。主要是通过 z^o 来进行控制的。并且还对上一阶段得到的 c^o 进行了放缩（通过一个tanh激活函数进行变化）。

其节点状网络结构如图



其网络结构隐层互相连接，以此来实现长期记忆

激活函数

激活函数选取LSTM常用的tanh函数对数据进行计算，将其数据变为-1到1之间的一个值，然后将其转化为其预测的下一个词的概率

损失函数

因为古诗的生成可以概括为多分类问题，其损失函数选取categorical_crossentropy loss（交叉熵损失函数）

-函数表达式
$C=-1N\sum i\sum k[yikln(pik)]$
其中，yik——样本i的label，与正类类别相同为1，否则为0；pik——样本ii的预测值为第k类的概率。

优点
使用逻辑函数得到概率，并结合交叉熵当损失函数时，在模型效果差的时候学习速度比较快，在模型效果好的时候学习速度变慢
缺点
sigmoid(softmax)+cross-entropy loss 擅长于学习类间的信息，因为它采用了类间竞争机制，它只关心对于正确标签预测概率的准确性，忽略了其他非正确标签的差异，导致学习到的特征比较散。基于这个问题的优化有很多，比如对softmax进行改进，如L-Softmax、SM-Softmax、AM-Softmax等。

优化器

优化器选择keras.optimizers.Adam()

在监督学习中我们使用梯度下降法时，学习率是一个很重要的指标，因为学习率决定了学习进程的快慢（也可以看作步幅的大小）。如果学习率过大，很可能会越过最优值，反而如果学习率过小，优化的效率可能很低，导致过长的运算时间，所以学习率对于算法性能的表现十分重要。而优化器keras.optimizers.Adam()是解决这个问题的一个方案。其大概的思想是开始的学习率设置为一个较大的值，然后根据次数的增多，动态的减小学习率，以实现效率和效果的兼得。

参数分析

数据预处理(对数据集进行处理，同时进行One-Hot编码:

```
# 禁用词，包含如下字符的唐诗将被忽略
DISALLOWED_WORDS = [' (', ') ', '(', ')', '__', '《', '》', '【', '】', '[', ']']
# 诗句最大长度
MAX_LEN = 64
# 最小词频（用于过滤低频词）
MIN_WORD_FREQUENCY = 8
```

```
# 每个epoch训练完成后，随机生成SHOW_NUM首古诗作为展示
SHOW_NUM = 3
```

模型采用随机梯度下降SGD来训练，： 将训练集分成多个mini_batch（即常说的batch）,一次迭代训练一个minibatch（即batchsize个样本），根据该batch数据的loss更新权值。这相比于全数据集训练，相当于是在寻找最优时人为增加了一些随机噪声，来修正由局部数据得到的梯度，尽量避免因batchsize过大陷入局部最优。

```
# 训练
TRAIN_EPOCHS = 20
BATCH_SIZE = 4
# LSTM每层神经元个数
HIDDEN_NUM = 128
```

论文研读心得

基于深度学习的歌词和古诗自动生成系统设计

基于Seq2Seq模型的自定义古诗词生成

对于论文中所提到的系统进行了初步的实现，即该模型；

使用（训练和测试）方法

训练模型

- 在setting.py中配置相关路径（现在的路径为相对路径）
- 运行train.py进行训练
- 运行过程中的时间及关键步骤信息等存于 logs/poetry.logs 中
- 所有配置信息都存放于 settings.py

```
现有参数为
epochs: 20
batch-size: 16
```

测试模型

由于该模型是对于古诗的生成，对于古诗的评价好坏指标没有具体

- 运行eval.py进行模型的查看
- 同时可以进行用户的交互

```
有三种功能
1、随机生成一首诗
2、根据用户给定的首句，生成剩下的部分
3、根据用户的输入，生成一首藏头诗
```

效果演示

- 模型概览

```
model = tf.keras.Sequential([
    # 不定长度的输入
    tf.keras.layers.Input((None, )),

    # 词嵌入层
    tf.keras.layers.Embedding(input_dim=tokenizer.vocab_size,
output_dim=settings.HIDDEN_NUM),
```

```
    # 第一个LSTM层，返回序列作为下一层的输入
    tf.keras.layers.LSTM(settings.HIDDEN_NUM, dropout=0.5, return_sequences=True),

    # 第二个LSTM层，返回序列作为下一层的输入
    tf.keras.layers.LSTM(settings.HIDDEN_NUM, dropout=0.5, return_sequences=True),

    # 对每一个时间点的输出都做softmax，预测下一个词的概率
    tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(tokenizer.vocab_size,
                                                          activation='softmax')),
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 128)         439552
_________________________________________________________________
lstm (LSTM)                  (None, None, 128)         131584
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 128)         131584
_________________________________________________________________
time_distributed (TimeDistri (None, None, 3434)        442986
=================================================================
Total params: 1,145,706
Trainable params: 1,145,706
Non-trainable params: 0
_________________________________________________________________
```

第一层embedding：将一个特征转换为一个向量。比如最容易理解的one-hot编码。但在实际应用当中，将特征转换为one-hot编码后维度会十分高。所以我们会将one-hot这种稀疏特征转化为稠密特征，通常做法也就是转化为我们常用的embedding。在NLP领域中，我们需要将文本转化为电脑所能读懂的语言，也就是把文本语句转化为向量，也就是embedding，其产生的参数个数为439552
第二层LSTM：其节点个数为128，产生的参数个数为131584，并将其输出作为输入传入下一层的LSTM中
第三层LSTM：其节点个数为128，产生的参数个数为131584
第四层TimeDistri：对每个时间点的输出进行softmax，产生的参数个数为442986

总的参数个数为1M左右，是一个比较小的模型

- 参数分析

```
2021-12-09 15:54:22 INFO: ========== 构建模型 ==========
2021-12-09 15:54:22.793576: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the f
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-12-09 15:54:23 INFO: ========== 创建数据集 ==========
2021-12-09 15:54:23 INFO: batch_size: 16
2021-12-09 15:54:23 INFO: data num: 24551
2021-12-09 15:54:23 INFO: ========== 开始训练 ==========
2021-12-09 15:54:23 INFO: train epochs: 20
```

该模型训练时采用的参数为

```
batch-size:16
data num:24551
train epochs:20
```

- 训练分析

```
Epoch 1/20
2021-12-09 15:54:27.688832: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
1534/1534 [==============================] - 224s 145ms/step - loss: 4.8821
2021-12-09 15:58:11 INFO: ========== 保存最优模型 (datasets/best_model.h5) ==========
2021-12-09 15:58:11 INFO: loss: 4.8820881843566895

2021-12-09 15:58:12 INFO: print example: 无城王日年，香客飞云回。
2021-12-09 15:58:12 INFO: print example: 秋道开天春，西开自日还。。，石不前日阳。
2021-12-09 15:58:13 INFO: print example: 年春出何，万雨江未。且同门叶，天长竹行。客道去中，归在多无。
```

```
Epoch 6/20
2021-12-09 16:12:14 INFO: print example: 风行清竹岸，月月乱湖行。路后心归处，新高北晚东。一风春水急，松雪近人稀。岁岁重人此，因时自送然。
1534/1534 [==============================] - 207s 135ms/step - loss: 3.8731
2021-12-09 16:15:41 INFO: ========== 保存最优模型 (datasets/best_model.h5) ==========
2021-12-09 16:15:41 INFO: loss: 3.8731274604797363
```

```
Epoch 20/20
1534/1534 [==============================] - 215s 140ms/step - loss: 3.5781
2021-12-09 17:03:52 INFO: ========== 保存最优模型 (datasets/best_model.h5) ==========
2021-12-09 17:03:52 INFO: loss: 3.5780889987945557

2021-12-09 17:03:54 INFO: print example: 无限江上春，出身如此时。青萝常易意，万月别归魂。风急心愁静，猿声独绕烟。清空听酒泪，谁在岁心清。
2021-12-09 17:03:55 INFO: print example: 高馆中来日，游人早晚清。山明唯道少，时想半初同。孤鸟时垂火，清蝉满月秋。莫怜千载句，应可赋吾贤。
2021-12-09 17:03:57 INFO: print example: 清水一声宿，幽人相隔分。秋风独去起，愁雁上吟来。秋水离惊乱，秋声雁暂迟。未知双史里，乡事梦相迟。
2021-12-09 17:03:57 INFO: ========== 训练结束 ==========
```

通过对训练过程的部分截图，我们可以看出：
每一轮迭代的步数为1534

在进行前几轮训练时，产生的古诗格式存在不正确的现象；一句古诗中穿插着一些标点符号，且loss值还比较大
在经过几轮epoch之后，我们可以看出其
随着训练的进行，我们可以发现loss逐渐下降，最终趋于稳定3.57左右，产生的古诗格式较为工整，且其意境也大有提高

训练时长：
在batch-size选择4时，一轮epoch耗时320s左右，整个模型训练花费大概2h
在batch-size选择16时，一轮epoch耗时207s左右，整个模型训练花费大概1.3h

- 测试分析

运行eval.py即可对模型进行测试

```
# 加载训练好的模型
logging.info('start test ...')
model = tf.keras.models.load_model(settings.BEST_MODEL_PATH)
```

对模型进行测试

```
1、随机生成一首
2、给出第一句，生成余下部分
3、生成藏头诗
4、退出
请输入您的选择：1
2021-12-09 21:51:23 INFO: random create poetry: 高桥自微雪，归坐忽何闻。日日分空去，人将此又稀。
```

```
1、随机生成一首
2、给出第一句，生成余下部分
3、生成藏头诗
4、退出
请输入您的选择：2
请输入第一句：同济大学
2021-12-09 21:51:34 INFO: have_pre create poetry: 同济大学，为尊初候。时其若德，歌舞惟光。登乐明序，天极其疆。永景承隆，时仰玉平。
```

```
1、随机生成一首
2、给出第一句，生成余下部分
3、生成藏头诗
4、退出
请输入您的选择：3
请输入藏头：同济大学
2021-12-09 21:51:41 INFO: create acrostic poetry: 同师本古峰僧客，济树中宵竹叶床。大内强来新酿得，学书须说老生诗。
```