建模背景

在这个快节奏的时代，我们总会在闲暇之余拿出手机，打开抖音等短视频平台来放松自己。有时候，我们会刷到一些让人心旷神怡的视频：它们配有十分治愈的音乐，展现着一副意境深远的画面；画面中或许有一句诗，或许没有。这种视频通常还会附上一句引人深思的文案，如“想看看当代网友的绝世文采”。观看这些视频，我们往往会被美景所吸引，心生向往，仿佛有一股冲动想要挥毫泼墨，写下一首诗来表达内心的感慨。然而，由于文化水平的限制，我们往往无法找到恰如其分的词句来形容这些令人陶醉的美景，只能用那两个众所周知的字来勉强描述。当我们好奇地打开评论区，来自网友的文采仿佛又是一次冲击，令人叹为观止。在美景与文采的双重震撼之下，我们不禁产生了一种望洋兴叹的感觉，觉得自己的才华实在是有限。随着对人工智能的了解逐渐加深，特别是最近大语言模型的火热，我们小组突发奇想：虽然我们或许无法亲手写出一首诗来，但是能否通过训练一个模型来实现这个目标呢？我们开始研究如何构建并训练这样一个模型，使其能够生成富有诗意的文字，表达出人类内心深处的情感。我们希望这个模型不仅能够为那些渴望展现文采的人们提供一种新的方式，还能让更多人了解并欣赏诗歌的美妙之处。经过一番努力，我们终于成功地训练出一个能够生成诗歌的模型。在它的帮助下，那些曾经感叹自己文采不足的人们也能够轻松地挥洒抒怀，尽情地享受诗歌的魅力。而这一切，都源于我们对美景与文采的热爱，以及对技术的探索与创新。

建模

通过对网上资料的查找，我们小组决定采用已有的GRU(Gated Recurrent Unit)模型作为我们的训练模型。

GRU模型是循环神经网络（Recurrent Neural Network，RNN）的一种变体，它的主要目的是解决传统RNN模型中遇到的梯度消失和梯度爆炸问题。在传统RNN中，当序列长度较长时，梯度会不断地通过时间反向传播，导致梯度会变得非常小或非常大，从而使得模型的训练变得困难。而GRU模型通过引入门控机制来有效解决这个问题。

GRU模型的核心是一个门控单元，它可以决定哪些信息需要被记忆、哪些信息需要被遗忘，以及在更新状态时需要考虑哪些信息。GRU模型的门控单元由两个门控组成：重置门（Reset Gate）和更新门（Update Gate）。重置门用于控制选择前一个时间步的隐藏状态和当前输入的哪些部分进行组合，以便更新当前时间步的隐藏状态。更新门用于控制选择哪些前一个时间步的隐藏状态和当前输入进行组合，以及控制当前时间步的隐藏状态相对于前一个时间步隐藏状态的权重。

以下是GRU模型的计算过程和数学公式：

1. **更新门**：计算当前时间步的输入和前一个时间步的隐藏状态的加权和，并通过sigmoid函数将其压缩到[0,1]之间，作为更新门的值。

   $z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$

   其中，x_t为当前时间步的输入，h_{t-1}为前一个时间步的隐藏状态，W_z、U_z和b_z为更新门的权重参数，σ表示sigmoid函数。

2. **重置门**：计算当前时间步的输入和前一个时间步的隐藏状态的加权和，并通过sigmoid函数将其压缩到[0,1]之间，作为重置门的值。

   $r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$

   其中，W_r、U_r和b_r为重置门的权重参数。

3. **重置后的隐藏状态**：将重置门和前一个时间步的隐藏状态进行元素相乘，得到重置后的前一个时间步的隐藏状态。

   $\tilde{h} * t - 1 = r_t \odot h * t - 1$

   其中，⊙表示元素相乘（Hadamard积）。

4. **新的候选隐藏状态**：将当前时间步的输入和重置后的前一个时间步的隐藏状态进行拼接，并通过tanh函数进行非线性变换，得到新的候选隐藏状态。

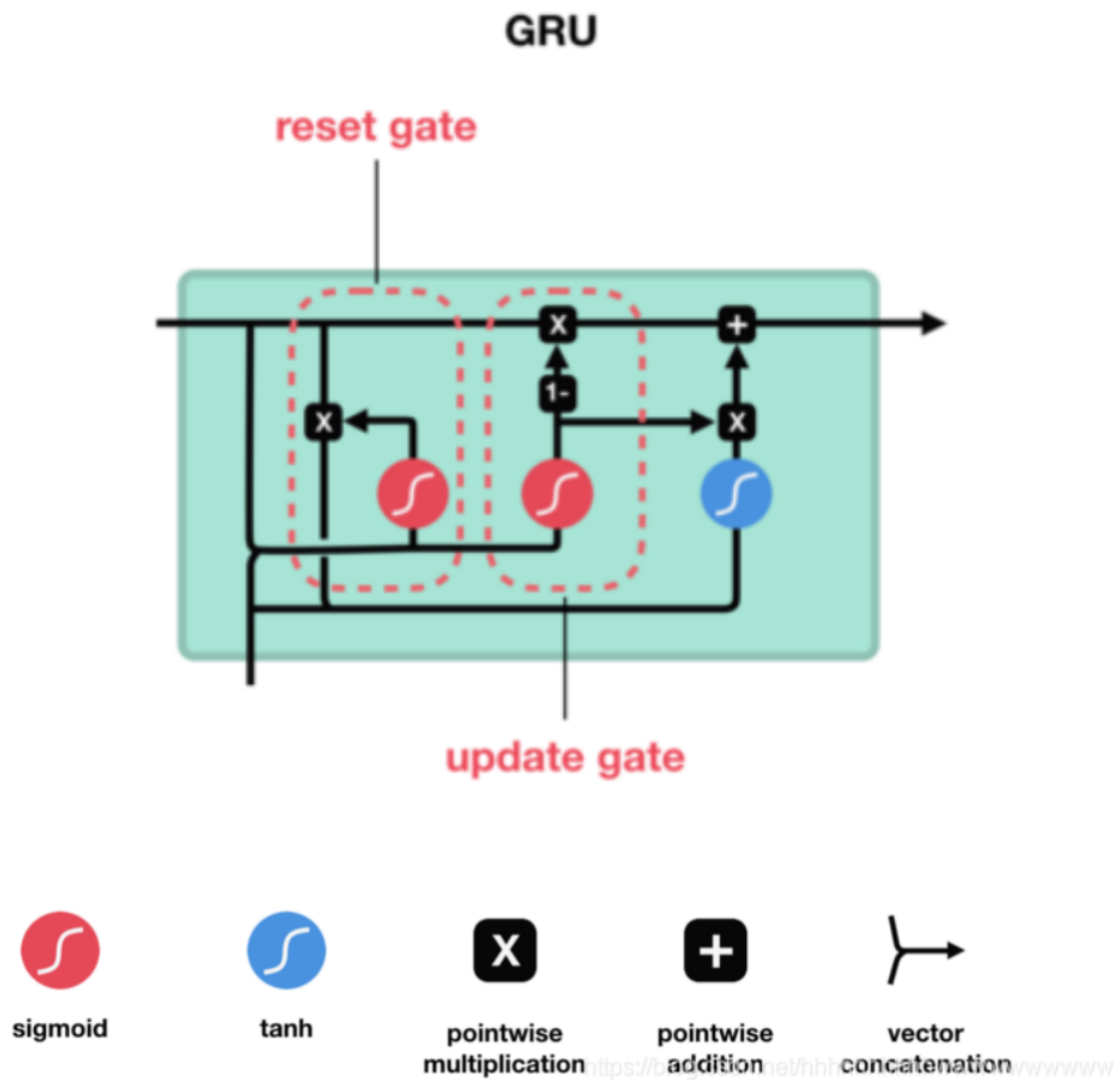
   $\tilde{h} * t = \tanh(W_h x_t + U_h \tilde{h} * t - 1 + b_h)$

   其中，W_h、U_h和b_h为候选隐藏状态的权重参数。

5. **更新当前时间步的隐藏状态**：将更新门和前一个时间步的隐藏状态进行加权平均，再加上候选隐藏状态的加权平均，得到当前时间步的隐藏状态。

   $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

流程图如下：



模型参考：https://github.com/Stevengz/Poem_compose

分析求解

我们预想的建模过程如下：

1. 构造数据集：为了降低难度，我们小组打算构造一个只包含五言诗的数据集，将每一句诗而不是每一首诗作为训练对象，使我们的模型能够学习到诗句中词语组合的规律。
2. 读取数据集：从文件中读取诗歌数据集。
3. 构建词汇表：由于计算机所能识别到的是二进制代码，为了神经网络对文本进行处理和学习，我们将所有诗歌中出现的字符作为词汇表，并给每个字符分配一个唯一的整数索引。
4. 将诗句转换为索引序列：将每一首诗歌中的字符转换为对应的整数索引，得到一个整数序列。将诗句转换为索引序列的作用是将文本数据转换为数字序列，方便神经网络对文本进行处理和学习。在

神经网络中，输入数据通常是数字序列，因此需要将文本数据转换为数字序列。将诗句转换为索引序列后，可以将其作为神经网络的输入，让神经网络学习诗句中的语言规律和特征，进而生成符合语法和韵律规律的诗歌。同时，将诗句转换为索引序列还可以减小数据集的大小，提高训练效率，因为数字序列通常比文本数据占用更少的存储空间。

5. 构建模型的函数：这里我们采用tensorflow提供的现有资源，定义一个 GRU 模型的函数，该模型包括一个嵌入层、一个 GRU 层和一个全连接层，其中嵌入层用于将输入的整数序列编码为密集向量表示。该层将输入的整数序列转换为嵌入矩阵，其中每个整数都被映射为一个固定大小的向量。这些向量可以捕捉到单词之间的语义关系和上下文信息。GRU 层的主要作用是对输入序列进行建模，捕捉序列中的长期依赖关系。GRU 层有三个门控单元（reset gate、update gate 和 new memory cell），这些门控单元可以控制信息的流动，从而防止梯度消失和梯度爆炸。将 GRU 层的输出转换为词汇表上的概率分布。该层将 GRU 层的输出进行线性变换，并通过 softmax 函数将输出转化为概率分布，表示每个词汇在当前上下文中的出现概率。该模型的输入是一个整数序列，输出是一个概率分布，表示下一个字符出现的概率。
6. 将诗句切分成输入和输出：将每一首诗歌的整数序列切分成输入和输出，其中输入是整数序列的前面一部分，输出是整数序列的后面一部分。通过这种方式，可以让神经网络学习到序列中的时间依赖关系，从而更好地生成符合语法和韵律规律的诗歌。
7. 创建训练样本：将所有诗歌的整数序列合并成一个大的整数序列，并将其转换为 TensorFlow 数据集格式。
8. 分批并随机打乱：将数据集分为多个批次，并在每个批次中随机打乱数据，以增加模型的泛化能力。
9. 创建模型：使用构建模型的函数创建一个 GRU 模型，并编译该模型，设置优化器和损失函数。
10. 训练模型：使用数据集训练模型，并在每个训练周期结束时保存模型的权重。可以通过调整超参数和模型结构来优化模型的性能。

仿真

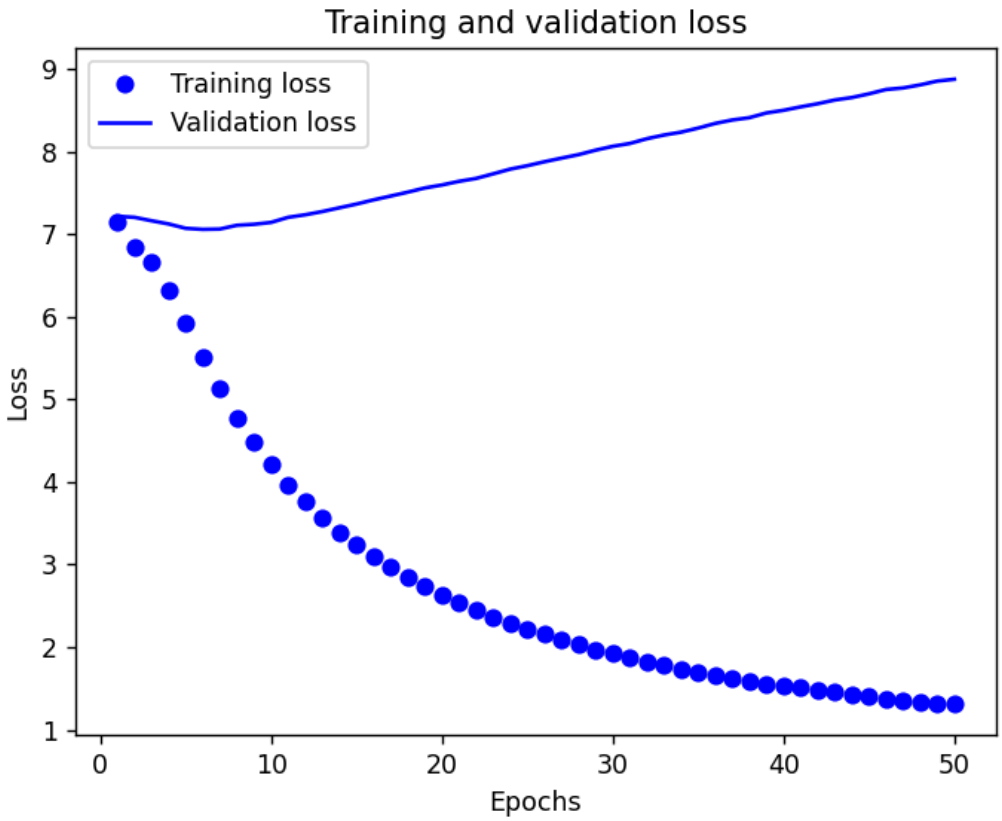
以下是我们的代码：

首先我们收集数据集，这是个体力活，在花了两个小时的时间一首一首复制粘贴过后，我们小组深感手动的效率低下。于是我们写了一个python代码，从https://so.gushiwen.cn收集到了一万多句五言诗。为了防止其中有重复的诗句对我们训练模型造成干扰，于是我们又写了一个去重程序，在完成去重操作后，可用的诗句有7402句。

```
# 构建模型的函数
def GRU_model(vocab_size, embedding_dim, units, batch_size):
    model = keras.Sequential([
        layers.Embedding(vocab_size,
                         embedding_dim,
                         batch_input_shape=[batch_size, None]),
        layers.GRU(units,
                   return_sequences=True,
                   stateful=True,
                   recurrent_initializer='glorot_uniform'),
        layers.Dense(vocab_size)
    ])
    return model
```

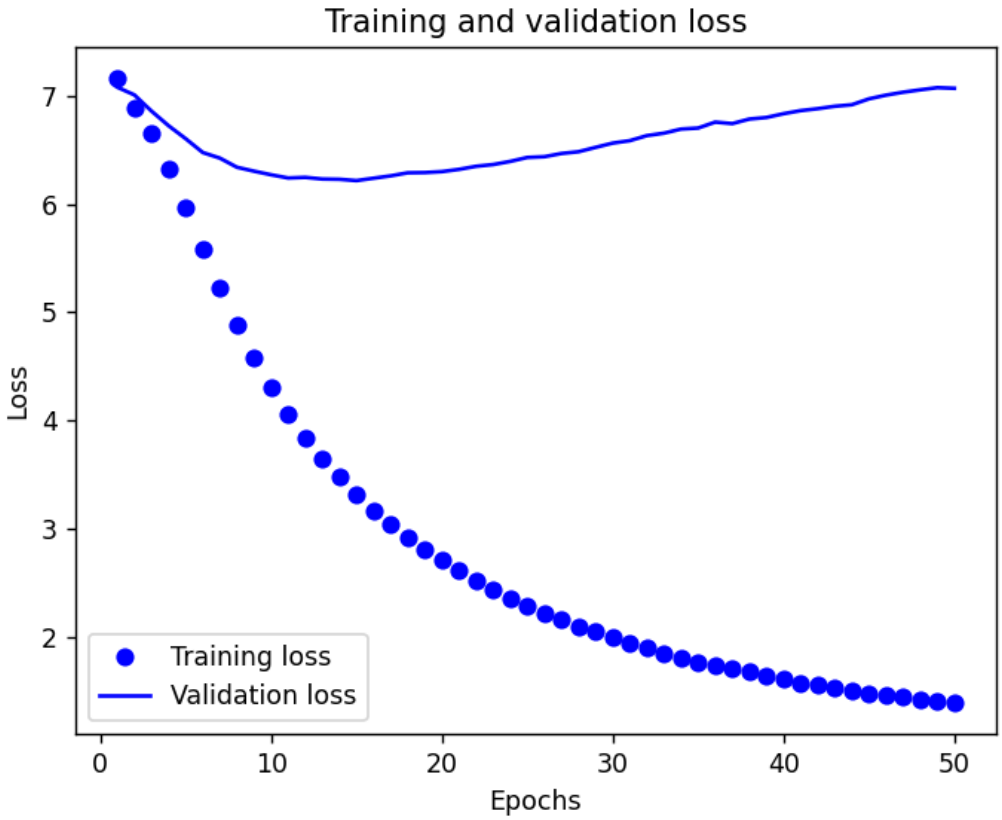
这是我们用于构造GRU模型的代码。在这个函数中，嵌入层的输入维度为 vocab_size，输出维度为 embedding_dim，batch_input_shape 设置为 [batch_size, None]，表示输入序列的长度可以是任意值。GRU 层的输出为一个序列，因为设置了return_sequences=True。同时，由于需要在序列之间保持状态，因此设置了 stateful=True。GRU 单元数由参数 units 指定，而 recurrent_initializer='glorot_uniform' 则表示使用 Glorot 均匀分布来初始化 GRU 的权重。全连接层的输出维度为 vocab_size，表示输出的是一个长度为 vocab_size 的向量，其中每个元素表示一个词汇在当前上下文中的出现概率。

为了检验模型的拟合效果，我们将数据集中前80%的数据作为训练集，后20%的数据作为测试集，并使用 Matplotlib 库绘制了它们的损失函数值随着训练轮数的变化情况。



如图所示，训练集的损失函数值在训练过程中持续下降，并且没有出现明显的波动，但是测试集的损失函数值在训练一定轮数后开始上升，说明我们的模型过拟合了。

我们将数据集中前95%的数据作为训练集，后5%的数据作为测试集后，效果有所好转，但是仍然存在过拟合的情况。



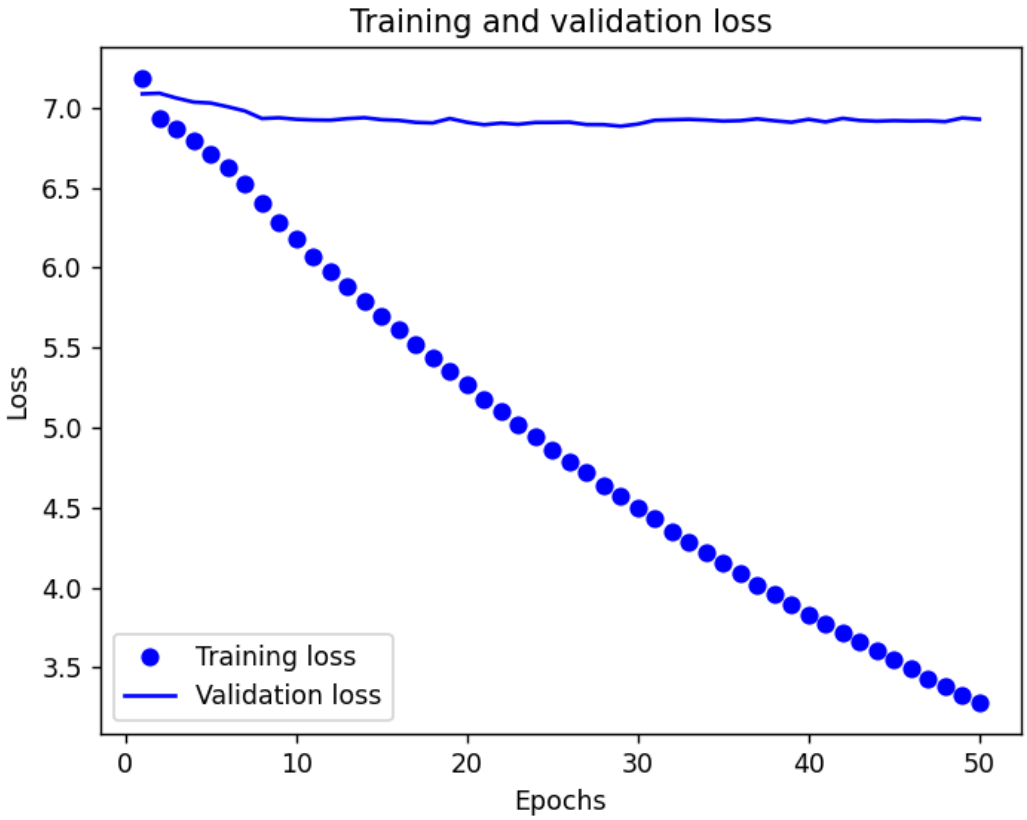
说明数据集过小可能是导致过拟合的原因之一。

如何我们又给模型添加正则化：修改后的代码为：

```
# 构建模型的函数
def GRU_model(vocab_size, embedding_dim, units, batch_size, l2_reg=0.001):
    model = keras.Sequential([
        layers.Embedding(vocab_size,
                         embedding_dim,
                         batch_input_shape=[batch_size, None]),
        layers.GRU(units,
                   return_sequences=True,
                   stateful=True,
                   recurrent_initializer='glorot_uniform',
                   kernel_regularizer=regularizers.l2(l2_reg)),
        layers.Dense(vocab_size)
    ])
    return model
```

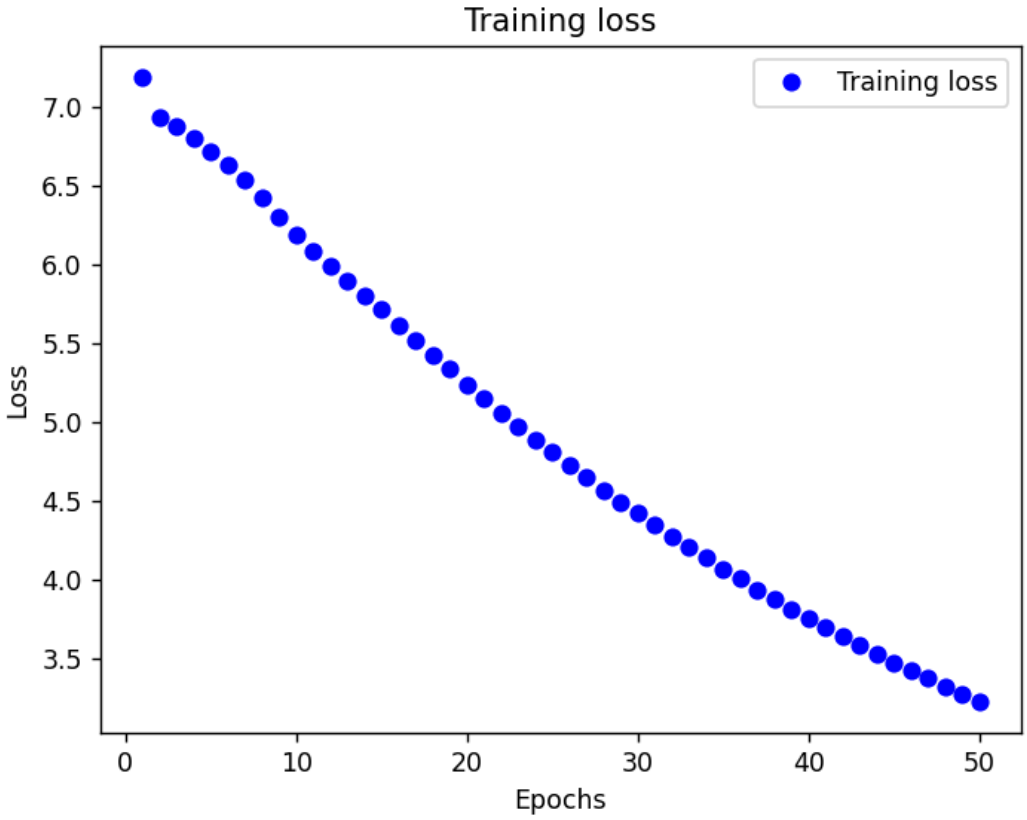
在上述代码中，我们向 GRU 层添加了一个 L2 正则化项，使用 kernel_regularizer 参数来实现。我们使用 regularizers.l2 方法创建一个 L2 正则化器，并将其传递给 kernel_regularizer 参数。这将导致 GRU 层的权重矩阵在训练过程中被 L2 正则化。

得到的结果如下图：



过拟合的问题再度减轻。

最后我们把数据集中所有的数据都作为训练集，并使用 Matplotlib 库绘制了其的损失函数值随着训练轮数的变化情况：



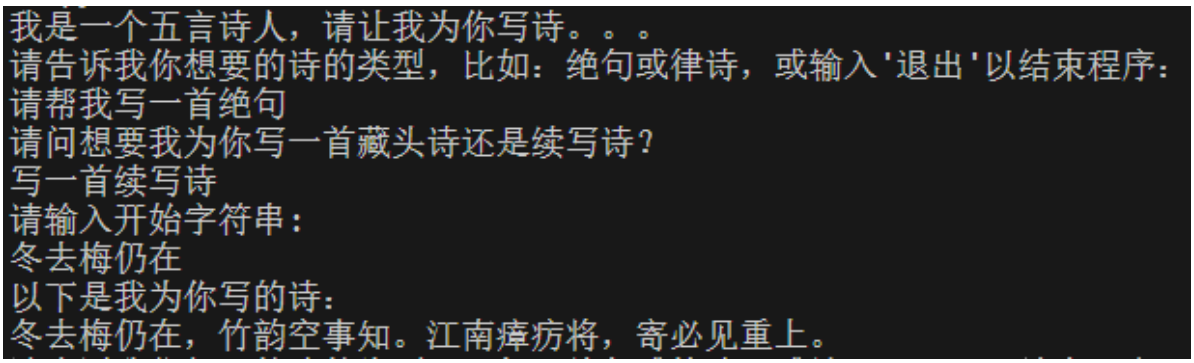
如图所示，训练集的损失函数值在训练过程中持续下降，并且没有出现明显的波动，我们认为拟合结果是比较好的。

现在来看看通过这个模型生成的诗吧：

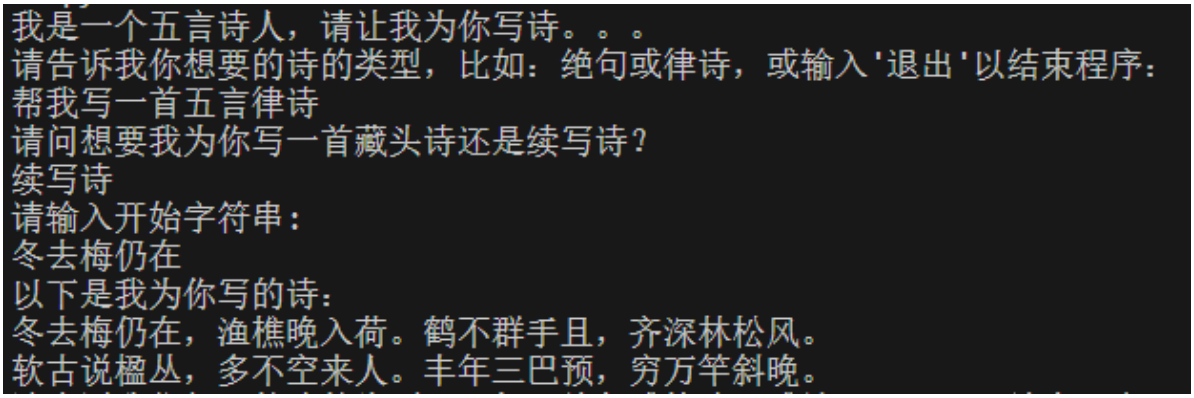
在我们训练完模型生成诗歌的代码中，我们添加了一些if-else逻辑，使其能够模拟出一些简单的跟用户对话的效果。

续写诗：

五言绝句：

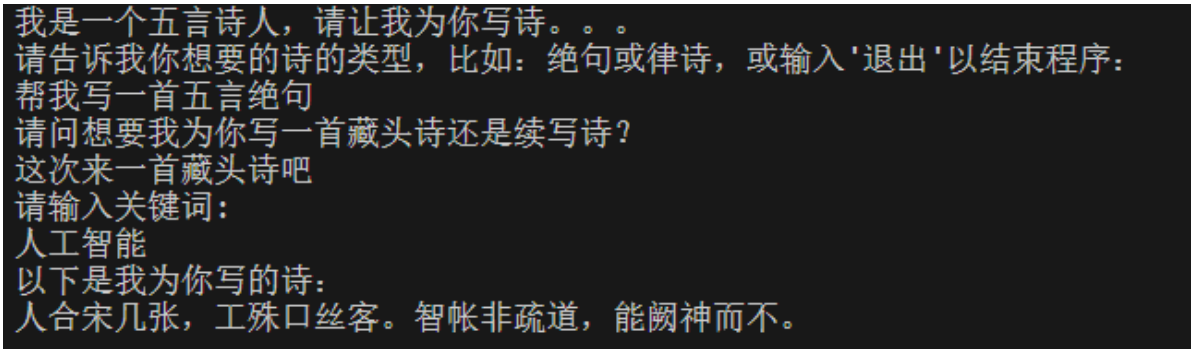


五言律诗：

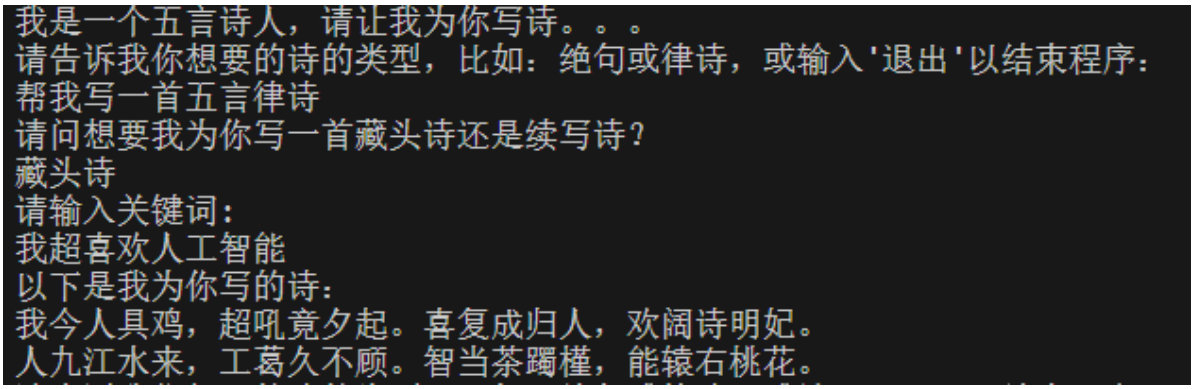


藏头诗：

五言绝句：



五言律诗：



结论

总的来说这次的模型还是达到了我们小组预期的效果的，它能够按照我们的要求生成诗。但是我们的模型仍然存在很多缺陷：

1. 我们的训练集很小并且十分单一，只是将五言诗拆分成句，而缺乏七言诗、古体诗或者词等文体的数据，同时我们也没用整理出古诗中常用的字词组合。
2. 我们这个模型生成的诗的类型也受到很大的限制，只能生成还能看的五言诗，但是对于七言诗，或者古体诗而言，我们这个模型的效果是极差的。
3. 生成的诗是毫无逻辑的。