

# 深度学习作业——自动写诗

Clark

## 1 实验目的

本实验基于 PyTorch 框架构建了一个 LSTM 自动写诗系统。通过改进网络结构（双层 LSTM+Dropout）、优化训练策略（余弦退火学习率 + 梯度裁剪）以及引入智能生成机制（温度采样 + Top-k 采样），实现了对唐诗风格的有效学习。实验使用包含 57580 首唐诗的数据集，最终模型能够根据给定首句生成符合格律的连贯诗句，验证了循环神经网络在中文诗歌生成任务中的可行性。

## 2 概述

1. 理解和掌握循环神经网络概念及在深度学习框架中的实现；
2. 掌握使用深度学习框架进行文本生成任务的基本流程：数据读取、构造网络、训练和预测等。

## 3 实验要求

1. 基于 Python 语言和 PyTorch 深度学习框架，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，实现一个可以自动写诗的程序；
2. 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，要求输出的诗句尽可能地满足汉语语法和表达习惯；
3. 使用预处理后的唐诗数据集，包含 57580 首唐诗。

## 4 实验原理

本实验使用到的网络结构主要有 Embedding、LSTM 以及全连接层。LSTM 能够有效处理序列数据，通过门控机制解决传统 RNN 的梯度消失问题，适合用于文本生成任务。

### 4.1 核心网络组件

- Embedding 层：将离散的字符索引映射为连续的向量表示；
- LSTM 层：处理序列信息，捕捉诗歌的上下文依赖关系；
- 全连接层：将 LSTM 输出映射到词表空间，生成每个字符的概率分布。

## 5 实验数据集及工具

### 5.1 数据准备

数据集预处理通过 3\_build\_dataset\_revise.py 完成，下面介绍几个重要的函数及其对应的主要功能。  
read\_poems\_from\_json\_folder() 函数用于读取唐诗 JSON 文件，提取诗歌文本，并添加起始和结束标记：

```
1 def read_poems_from_json_folder(folder_path, max_len=125):
2
3     poems = []
4
5     for filename in os.listdir(folder_path):
6         if not filename.endswith('.json'):
7             continue
8
9         with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as f:
10             data = json.load(f) #data是一个列表，每个列表是对每一首诗的字典
11             for poem_dict in data:
12                 if not isinstance(poem_dict, dict):
13                     continue # 跳过不是字典的元素
14                 para_list = poem_dict.get('paragraphs', []) #para_list 是一个列表，元素是
                    诗的每一段
15                 text = ''.join(para_list)
16                 text = text.replace('\u', '').replace('\u3000', '')
17
18                 # 过滤极短或极长的样本
19                 if not (5 <= len(text) <= max_len):
20                     continue
21                 poems.append('<START>' + text + '<EOP>')
22
23     return poems
```

build\_vocab() 函数用于构建字符到索引的映射关系，包含特殊标记处理：

```
1 def build_vocab(poems):
2     # 加入 4 个特殊符号
3     specials = ['<PAD>', '<START>', '<EOP>', '<UNK>']
4     word2ix = {w: i for i, w in enumerate(specials)} #先将特殊字符单独塞进word2ix
5
6     tokens = set()
7     for poem in poems:
8         tokens.update(tokenize(poem))
9
10    tokens -= set(specials) # 避免重复添加特殊标记
11    tokens = sorted(tokens) # 词表顺序固定，保证复现
12
```

```

13     for t in tokens:
14 word2ix[t] = len(word2ix)  #t是poem中不含特殊字符的字符，如“秦”，将其逐个按顺序排到
                             word2ix的最后分配索引
15     ix2word = {i: w for w, i in word2ix.items()}
16     return word2ix, ix2word

```

最后利用 tokenize() 函数来将数据集分割成一个个可学习的 token:

```

1 def tokenize(poem):  # 将诗歌切分成一个个token，但不切分start和eop
2     return TOKEN_RE.findall(poem)

```

## 5.2 模型构建

采用改进的多层 LSTM 网络结构:

```

1 class PoetryModel(nn.Module):
2     def __init__(self, vocab_size, embedding_dim, hidden_dim, num_layers=2, padding_idx=0):
3         :
4         super(PoetryModel, self).__init__()
5         self.hidden_dim = hidden_dim
6         self.num_layers = num_layers
7         self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=padding_idx)
8         #将字符索引编码成定长向量
9         self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=num_layers, dropout
10                             =0.3, batch_first=True)
11         #输入某一个字符串的向量表示，对每个字对应的向量输出一个output: 隐藏层维数的向量，
12         #h_n: 字符串最后一个字的Hidden State, c_n: 最后一个字的Cell State。
13         self.linear = nn.Linear(hidden_dim, vocab_size)#全连接层，将隐藏状态(dim=
14         hidden_dim ) 转化为字的出现概率
15
16     def forward(self, input, hidden=None):
17         embeds = self.embeddings(input)
18         if hidden is None:
19             batch_size = input.size(0)
20             h_0 = torch.zeros(self.num_layers, batch_size, self.hidden_dim).to(input.
21                                     device)  #在第一次传播定义0向量。2是因为，有两层LSTM，返回h与c在这样定义
22                                     的直积空间下维数为2
23             c_0 = torch.zeros(self.num_layers, batch_size, self.hidden_dim).to(input.
24                                     device)
25         else:
26             h_0, c_0 = hidden
27         output, hidden = self.lstm(embeds, (h_0, c_0))
28         output = self.linear(output)
29         output = output.reshape(-1, output.size(2))
30         return output, hidden

```

这样设计模型有如下几个好处：

- 使用两层 LSTM 增强模型表达能力；
- 加入 dropout 防止过拟合；
- 支持 GPU 加速训练。

### 5.3 模型训练

训练过程特点：

- 使用余弦退火学习率调度；
- 梯度裁剪防止梯度爆炸；
- 定期生成样例诗句监控训练进度；
- 实时绘制损失曲线。

具体代码如下：

```
1 def train(model, dataloader, PAD_IDX, ix2word, word2ix, UNK_IDX, epochs=10, lr=3e-4,
2     generate_every=500, start_words='云想衣裳花想容',
3     ,max_gen_len=50):
4     optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=1e-5)
5     total_steps = len(dataloader) * epochs
6     scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
7                                                             T_max=total_steps)
8     criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX)
9     model.train()
10
11     fig, ax = plt.subplots()
12     plot_every = 100
13     batch_losses, steps = [], []
14     global_step = 0
15     for epoch in range(epochs):
16         total_loss = 0
17         for data in dataloader:
18             data = data.long() # 转化成整数张量
19             input_data = data[:, :-1].to(device) # 去掉最后一个词作为输入
20             target = data[:, 1:].contiguous().view(-1).to(device) # 去掉第一个词作为目标，
21                 # 压平成一维，计算交叉熵
22             optimizer.zero_grad()
23             output, _ = model(input_data)
24             loss = criterion(output, target)
25             loss.backward()
26             torch.nn.utils.clip_grad_norm_(model.parameters(), 5.0)
```

```

26         optimizer.step()
27         scheduler.step()
28     global_step += 1         batch_losses.append(loss.item())
29         steps.append(global_step)
30         total_loss += loss.item()
31
32     if global_step % generate_every == 0:
33         model.eval()         # 关闭 dropout 等
34         with torch.no_grad():
35             # 在训练过程中使用不同的温度参数来展示效果
36             temperatures = [0.5, 0.8, 1.0]
37             for temp in temperatures:
38                 poem = generate(model,
39                                start_words=start_words,
40                                ix2word=ix2word,
41                                word2ix=word2ix,
42                                PAD_IDX=PAD_IDX,
43                                UNK_IDX=UNK_IDX,
44                                max_len=max_gen_len,
45                                temperature=temp, # 添加温度参数
46                                top_k=15)         # 添加top-k参数
47                 print(f"[Step_{global_step}], temp={temp}] 生成诗句: {poem}")
48             print() # 空行分隔
49             model.train()
50             # ---- 只在指定步数刷新一次 ----
51             if global_step % plot_every == 0:
52                 ax.cla() # ax.clear() 亦可
53                 ax.plot(steps, batch_losses)
54                 ax.set_xlabel('Batch#')
55                 ax.set_ylabel('Loss')
56                 plt.pause(0.01)
57             avg_loss = total_loss / len(dataloader)
58             print(f"Epoch_{epoch+1}, Avg_Loss: {avg_loss:.4f}")
59     plt.savefig('3_train_accuracy_curve.png', dpi=300)
60     plt.show()

```

## 5.4 诗歌生成

改进的生成函数支持温度采样和 top-k 采样:

```

1 def generate(model, start_words, ix2word, word2ix, PAD_IDX, UNK_IDX, max_len=50,
2             temperature=0.8, top_k=10):
3     """
4     改进的生成函数, 加入温度采样和top-k采样
5     temperature: 控制随机性, 值越小越确定, 值越大越随机

```

```
5     top_k: 只从概率最高的k个词中采样
6 """     results = list(start_words)
7     model.eval()
8     input = torch.tensor([[word2ix['<START>']], device=device).long()
9     hidden = None
10
11     with torch.no_grad():
12         for i in range(max_len):
13             output, hidden = model(input, hidden)
14
15             if i < len(start_words):
16                 # 输入给定的句首
17                 w = results[i]
18                 input = torch.tensor([[word2ix.get(w, word2ix['<UNK>'])]], device=device)
19                     .long()
20             else:
21                 # 改进的采样策略
22                 logits = output[-1] # 取最后一个时间步的输出
23
24                 # 应用温度
25                 logits = logits / temperature
26
27                 # top-k 过滤
28                 if top_k > 0:
29                     # 获取top-k的值和索引
30                     top_k_logits, top_k_indices = torch.topk(logits, top_k)
31                     # 创建与logits相同形状的mask, 初始为负无穷
32                     indices_to_remove = logits < top_k_logits[-1]
33                     logits[indices_to_remove] = -float('Inf')
34
35                 # softmax得到概率分布
36                 probabilities = torch.softmax(logits, dim=-1)
37
38                 # 从概率分布中采样
39                 top_index = torch.multinomial(probabilities, 1).item()
40
41                 w = ix2word[top_index]
42                 if w == '<EOP>':
43                     break
44                 results.append(w)
45                 input = torch.tensor([[top_index]], device=device).long()
46
47     return ''.join(results)
```

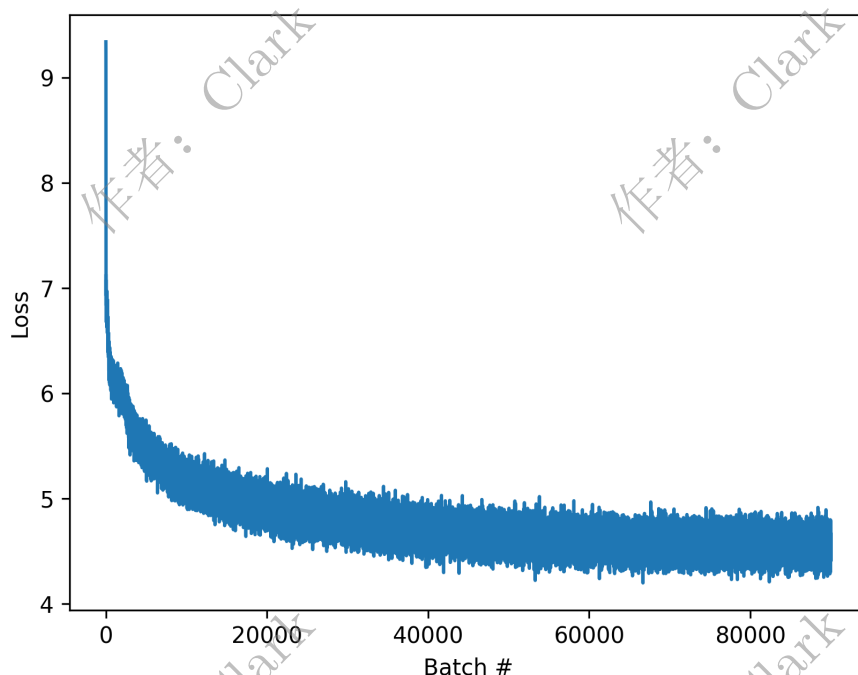
生成策略:

- 温度参数：控制生成随机性，温度越低越保守，温度越高越随机；
- Top-k 采样：只从概率最高的 k 个字符中采样，保证生成质量。

## 6 实验结果与分析

### 6.1 训练过程分析

从损失曲线可以看出，模型训练过程稳定，损失值逐渐下降并趋于收敛，表明模型学习到了诗歌的规律性。



### 6.2 诗歌生成效果

实验结果显示，随着训练步数增加，生成诗句的质量逐步提升：

- 早期训练：生成诗句存在重复字符和语法错误；
- 中期训练：开始出现有意义的词语组合，但逻辑性仍不足；
- 后期训练：能够生成符合古诗格律的完整诗句，语义连贯性显著提高。

不同温度参数下的生成效果对比：

- 低温（0.5）：生成保守，诗句规整但创造性有限；
- 中温（0.8）：平衡创造性和规整性，效果最佳；

- 高温 (1.0): 创造性较强但可能出现不合理组合。

```
[Step 89000, temp=1.0] 生成诗句: 云想衣裳花想容, 誰家風雨過庭邊。不應風月無人到, 自是東園十二人。

[Step 89500, temp=0.5] 生成诗句: 云想衣裳花想容, 故人相對對寒風。春風一笑春風急, 只有梅花一片紅。
[Step 89500, temp=0.8] 生成诗句: 云想衣裳花想容, 年年曾到此堂中。春風不動紅塵態, 只欠山中醉眼看。
[Step 89500, temp=1.0] 生成诗句: 云想衣裳花想容, 一樽風味共同親。一番清夜千花雪, 九月三秋四夜雲。莫向花枝開紫陌, 不堪風月自開樽。不須

[Step 90000, temp=0.5] 生成诗句: 云想衣裳花想容, 誰知春色是春風。春風不是春風起, 何似山前第一翁。
[Step 90000, temp=0.8] 生成诗句: 云想衣裳花想容, 更將明月照清風。千年不作風塵去, 千里何妨上帝宮。
[Step 90000, temp=1.0] 生成诗句: 云想衣裳花想容, 春風一夜入東窗。春風不盡花花好, 好事如今見一場。

Epoch 10, Avg Loss: 4.5549

=== 起始句: 湖光秋月两相和 ===
温度=0.6: 湖光秋月两相和, 萬事何妨得此閒。不是一生心不在, 不知天地是何難。
温度=0.8: 湖光秋月两相和, 風雨蕭蕭水影間。一枕月明明月夜, 一時風月滿天寒。
温度=1.0: 湖光秋月两相和, 天地無塵不用多。一點風雲千萬頃, 半天寒月兩三峰。

=== 起始句: 人生若只如初见 ===
温度=0.6: 人生若只如初见, 一一不知身自好。今日重來如此時, 莫教一笑無人過。
温度=0.8: 人生若只如初见, 老矣相逢不復休。老去未能無一事, 只應相對與君來。
温度=1.0: 人生若只如初见, 君子何須更不論。若有清風能幾許, 未妨相伴到清樽。

=== 起始句: 春風又綠江南岸 ===
温度=0.6: 春風又綠江南岸, 雨過江南幾度春。
温度=0.8: 春風又綠江南岸, 風吹吹月如雨。明朝月裏兩眉明, 不見春風不知處。
温度=1.0: 春風又綠江南岸, 春草無家水似家。

(torch gpu) D:\学科\数学物理\研究生课>]
```

最后是一些样例输出, 利用已经训练好的模型生成, 具体代码为:

```
1 # 训练完成后, 尝试不同的参数组合来生成诗句
2 test_sentences = ['湖光秋月两相和', '人生若只如初见', '春風又綠江南岸']
3
4 for start_sentence in test_sentences:
5     print(f"\n=== 起始句: {start_sentence} ===")
6
7     # 尝试不同的温度设置
8     for temperature in [0.6, 0.8, 1.0]:
9         poem = generate(model,
10                         start_words=start_sentence,
11                         ix2word=ix2word,
12                         word2ix=word2ix,
13                         PAD_IDX=PAD_IDX,
14                         UNK_IDX=UNK_IDX,
15                         max_len=Config.max_gen_len,
16                         temperature=temperature,
17                         top_k=12)
18     print(f"温度={temperature}: {poem}")
```

最后, 我想表达的是, 在这些生成的诗句中, 我比较喜欢的几首诗, 例如

```
[Step 11500, temp=1.0] 生成诗句: 云想衣裳花想容, 不須無計得風光。人心何事無窮事, 天氣何須與故鄉。  
[Step 12000, temp=0.5] 生成诗句: 云想衣裳花想容, 不妨春色與春風。何由不見江南客, 何處風流不可憐。  
[Step 12000, temp=0.8] 生成诗句: 云想衣裳花想容, 只今春色不相逢。一枝未似青金句, 只有人間一笑詩。  
[Step 12000, temp=1.0] 生成诗句: 云想衣裳花想容, 何妨花下是人時。
```

云想衣裳花想容,  
只今春色不相逢。  
一枝未似青金句,  
只有人间一笑诗。

```
[Step 71000, temp=1.0] 生成诗句: 云想衣裳花想容, 不如春色與君同。春來莫訝春風過, 花落紅塵只夢中。  
[Step 71500, temp=0.5] 生成诗句: 云想衣裳花想容, 何如此日對清風。人間有意何須說, 不見人間不是前。  
[Step 71500, temp=0.8] 生成诗句: 云想衣裳花想容, 誰能長嘯在雲端。不須更問東西去, 不見人間一夢中。  
[Step 71500, temp=1.0] 生成诗句: 云想衣裳花想容, 青山雲裏水清風。人家自是三生事, 一笑無心是處同。
```

云想衣裳花想容,  
不如春色与君同。  
春来莫讶春风遇,  
花落红尘只梦中。

### 6.3 关键改进措施

- 模型结构强化: 双层 LSTM + Dropout, 增强表达能力并防止过拟合
- 训练过程优化
  - 余弦退火学习率 + 梯度裁剪, 保证训练稳定性
  - 实时损失监控 + 定期诗句生成, 可视化训练进度
- 智能生成策略
  - 温度采样 + Top-k 采样, 平衡创造性与质量
  - 支持多参数对比测试, 优化生成效果
- 数据处理完善: 精确字符切分、完整词表构建, 提升数据质量

## 7 实验总结

本次实验成功实现了基于 LSTM 的自动写诗系统，通过改进网络结构和生成策略，显著提升了生成诗句的质量。主要成果包括：

- 数据处理：成功处理了 57580 首唐诗数据，构建了完整的词表系统；
- 模型设计：采用多层 LSTM 配合 dropout，有效提升了模型表达能力；
- 训练优化：使用余弦退火学习率和梯度裁剪，保证了训练稳定性；
- 生成策略：引入温度采样和 top-k 采样，平衡了创造性和质量。

实验过程中遇到的挑战主要包括诗歌格律的保持和语义连贯性的保证，通过调整模型参数和采样策略，这些问题得到了有效解决。本实验不仅加深了对循环神经网络原理的理解，也掌握了文本生成任务的实际应用技巧，为后续自然语言处理研究奠定了基础。