# 自动写诗

# 1 实验概述

本实验的目标是训练一个可以自动写中文诗的模型，完成两个功能：

- 给首句能续写
- 给几个字能藏头

# 2 实验环境

## 2.1 硬件环境

- CPU：Intel i5-12500H
- GPU：NVIDIA Geforce RTX 2050（4GB）

## 2.2 软件环境

- Windows 11
- Anaconda

## 2.3 Conda环境

- python=3.9
- pytorch=2.0.0
- pytorch-cuda=11.8
- torchaudio=2.0.0

# 3 实验思路

## 3.1 数据来源

本次实验的数据来自提供的预处理过的数据集 `tang.npz` ，含有57580首唐诗，每首诗限定在125词，不足125词的以 `<s>` 填充。数据集以npz文件形式保存，包含三个部分：

（1）data: 诗词数据，将诗词中的字转化为其在字典中的序号表示。

（2）ix2word: 序号到字的映射

（3）word2ix: 字到序号的映射

### 3.2 数据预处理

该数据集已经被预处理过，无需再次处理。

### 3.3 模型选择与参数设置

模型网络选择 `LSTM`

参数设置如下：

```
# 设置超参数
BATCH_SIZE = 32
NUM_WORKERS = 2
LR = 5e-3       # 学习率
EMBEDDING_DIM = 128         # 嵌入层维度
HIDDEN_DIM = 256            # 隐藏层维度
EPOCH_NUM = 4                 # 训练轮数
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
EPOCH = 30
LR = 0.001
TRAIN_RATE = 0.8
NUM_WORKERS = 14 # cpu cores num
BATCH_SIZE = 1500
```

### 3.4 写诗

对于第一个功能续写来说，其实现方式比较简单，至于要把首句作为 `inputs` 放入模型即可。而对于第二个要实现的功能——藏头，简易的模型无法理解该功能的含义，因此打算通过手动控制生成过程，将要藏的字在恰当的时机填入恰当的位置来完成。

## 4 实验代码

### 4.1 数据载入部分

```
def prepare_data():
    # 读入预处理的数据
    datas = np.load("./data/tang.npz", allow_pickle=True)
    data = datas['data']
    ix2word = datas['ix2word'].item()
    word2ix = datas['word2ix'].item()

    # 转为torch.Tensor
    data = torch.from_numpy(data)
    dataloader = DataLoader(data,
```

```
                            batch_size = BATCH_SIZE,
                            shuffle = True,
                            num_workers = NUM_WORKERS)
    return dataloader, ix2word, word2ix

dataloader, ix2word, word2ix = prepare_data()
```

## 4.2 模型部分

```
class PoetryModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(PoetryModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, self.hidden_dim, num_layers=2)
        self.linear = nn.Linear(self.hidden_dim, vocab_size)

    def forward(self, inputs, hidden = None):
        seq_len, batch_size = inputs.size()

        if hidden is None:
            h_0 = inputs.data.new(2, batch_size,
self.hidden_dim).fill_(0).float()
            c_0 = inputs.data.new(2, batch_size,
self.hidden_dim).fill_(0).float()
        else:
            h_0, c_0 = hidden

        embeds = self.embedding(inputs)
        outputs, hidden = self.lstm(embeds, (h_0, c_0))
        outputs = self.linear(outputs.view(seq_len * batch_size, -1))
        return outputs, hidden


model_path = None           # 预训练模型路径
model = PoetryModel(len(word2ix), EMBEDDING_DIM, HIDDEN_DIM)
if model_path:
    model.load_state_dict(torch.load(model_path))
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = LR)
```

## 4.3 训练部分

```
# train
from tqdm import tqdm
train_losses = []
train_counter = []
```

```
5
6   for epoch in range(0, EPOCH_NUM):
7       model.train()
8
9       loop = tqdm(dataloader, total = len(dataloader))
10      for data in loop:
11          data = data.long().transpose(1, 0).contiguous()
12          data = data.to(device)
13          inputs, target = data[:-1, :], data[1:, :]
14          outputs, _ = model(inputs)
15          loss = criterion(outputs, target.view(-1))
16
17          optimizer.zero_grad()
18          loss.backward()
19          optimizer.step()
20
21          train_losses.append(loss.item())
22          if len(train_counter) == 0:
23              train_counter.append(len(data))
24          else:
25              train_counter.append(train_counter[-1] + len(data))
26
27          loop.set_description(f'Epoch [{epoch + 1}/{EPOCH_NUM}]')
28          loop.set_postfix(loss = loss.item())
29
30      print('[epoch:%d] <Train> Loss: %.03f' % (epoch + 1, train_losses[-1]))
31
32  print('Train has finished, total epoch is %d' % EPOCH_NUM)
33  torch.save(model.state_dict(), './' + str(EPOCH_NUM) +'epoch-model.pth')
34  torch.save(optimizer.state_dict(), './' + str(EPOCH_NUM) +'epoch-
    optimizer.pth')
```

## • 4.4 写诗部分

```
1   def generate(start_words, ix2word, word2ix):
2       model = PoetryModel(len(word2ix), EMBEDDING_DIM, HIDDEN_DIM)
3       model.load_state_dict(torch.load(model_path))
4       model.to(device)
5       results = list(start_words)
6       start_word_len = len(start_words)
7
8       inputs = torch.Tensor([word2ix['<START>']]).view(1, 1).long()
9       inputs = inputs.to(device)
10      hidden = None
11
12      for i in range(max_gen_len):
13          outputs, hidden = model(inputs, hidden)
14          if i < start_word_len:
15              w = results[i]
```

```
            inputs = inputs.data.new([word2ix[w]]).view(1, 1)
        else:
            top_index = outputs.data[0].topk(1)[1][0].item()
            w = ix2word[top_index]
            results.append(w)
            inputs = inputs.data.new([top_index]).view(1, 1)
        if w == '<EOP>':
            del results[-1]
            break

    poetry_str = ""
    for ch in results:
        poetry_str += ch
        if ch == u'。':
            poetry_str += '\n'
    return poetry_str

# load model and poetry
model_path = './4epoch-model.pth'       # 模型路径
start_words = '湖光秋月两相和'             # 唐诗的第一句
max_gen_len = 125                       # 生成唐诗的最长长度
results = generate(start_words, ix2word, word2ix)
print(results)
```

## • 4.5 写藏头诗部分

```
def gen_acrostic(start_words, ix2word, word2ix):
    model = PoetryModel(len(word2ix), EMBEDDING_DIM, HIDDEN_DIM)
    model.load_state_dict(torch.load(model_path))
    model.to(device)
    results = []
    start_word_len = len(start_words)
    inputs = (torch.Tensor([word2ix['<START>']]).view(1, 1).long())
    inputs = inputs.to(device)
    hidden = None

    index = 0            # 指示已生成了多少句
    pre_word = '<START>' # 上一个词

    for i in range(max_gen_len_acrostic):
        outputs, hidden = model(inputs, hidden)
        top_index = outputs.data[0].topk(1)[1][0].item()
        w = ix2word[top_index]
        if (pre_word in {u'。', u'! ', '<START>'}):
            # 如果生成的诗已经包含全部“头”，则结束
            if index == start_word_len:
                break
            # 把“头”作为输入喂入模型
            else:
```

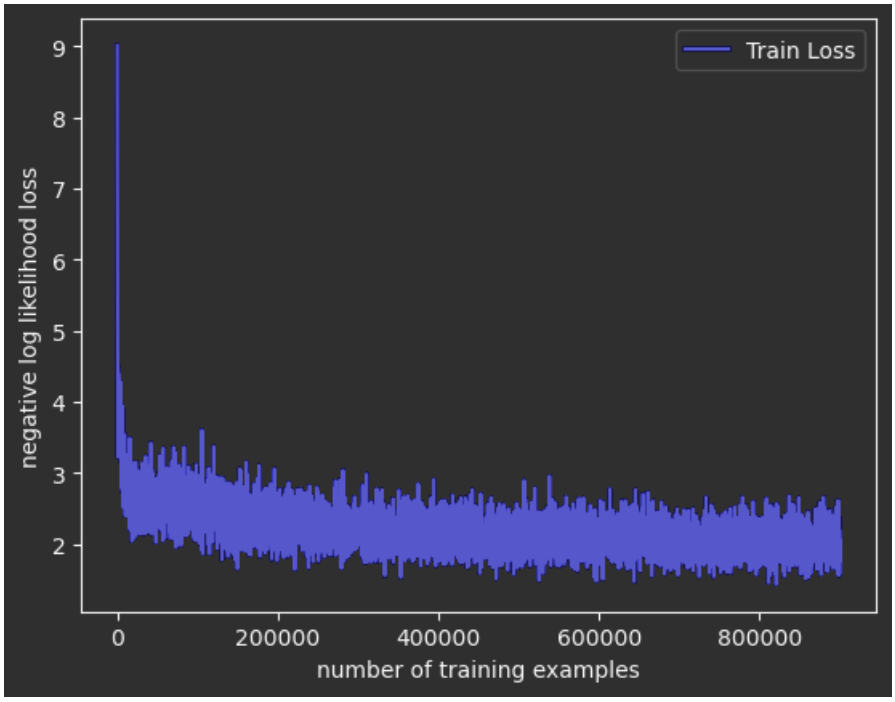
```
                    w = start_words[index]
                    index += 1
                    inputs = (inputs.data.new([word2ix[w]])).view(1, 1)
            else:
                inputs = (inputs.data.new([word2ix[w]])).view(1, 1)
            results.append(w)
            pre_word = w
        poetry_str = ""
        for ch in results:
            poetry_str += ch
            if ch == u'。':
                poetry_str += '\n'
        return poetry_str

# load model and poetry
model_path = './4epoch-model.pth'        # 模型路径
start_words_acrostic  = '湖光秋月两相和'    # 藏头的内容
max_gen_len_acrostic = 125               # 生成唐诗的最长长度
results_acrostic = gen_acrostic(start_words_acrostic, ix2word, word2ix)
print(results_acrostic)
```

# 5 实验结果

## 5.1 训练部分输出展示

```
Epoch [1/4]: 100%|██████████| 1800/1800 [02:17<00:00, 13.07it/s, loss=2.61]
[epoch:1] <Train> Loss: 2.610
Epoch [2/4]: 100%|██████████| 1800/1800 [02:17<00:00, 13.12it/s, loss=2.07]
[epoch:2] <Train> Loss: 2.072
Epoch [3/4]: 100%|██████████| 1800/1800 [02:16<00:00, 13.20it/s, loss=2]
[epoch:3] <Train> Loss: 2.000
Epoch [4/4]: 100%|██████████| 1800/1800 [02:16<00:00, 13.19it/s, loss=1.81]
[epoch:4] <Train> Loss: 1.813
Train has finished, total epoch is 4
```

## 5.2 Train Loss图像




## • 5.3 根据首句写诗

```
results = generate(start_words, ix2word, word2ix)
print(results)
```

湖光秋月两相和，水石无人见山色。
山中日月照春风，春风吹落花前开。
春风吹落花落时，春风吹落花前开。
春风吹落花落时，春风吹落花前开。
春风吹落花落时，春风吹落花前开。
春风吹落花落时，春风吹落花前开。
春风吹，春色长，不见春风吹。
不知此曲不相见，不见春风吹不定

## • 5.4 写藏头诗

```
results_acrostic = gen_acrostic(start_words_acrostic, ix2word, word2ix)
print(results_acrostic)
```

湖上春风起，江南春草深。
光辉日月好，独坐夜深深。
秋色生春色，清风入竹扉。
月明花下月，花落夜灯明。
两岸花初发，新晴月满林。
相思无限客，春色满林阴。
和气随风起，清风入夜深。

# 6 总结

本次实验学到了很多很多，首先是对文本数据的处理，由于实验附件提供了 `word2vec` 相关内容，因此难度有所下降。另外是对网络生成的掌控。由于要实现藏头诗的生成，而简易的 `LSTM` 网络本身无法理解**藏头**的含义，所以选择了通过手动的方式来控制生成。最后由于训练速度的变慢，前两个实验所采用的训练过程不再合适，学习了如何在训练过程中使用 `tqdm` 来加上进度条获取训练速度、时间等相关信息。

从结果上看，模型生成诗句的效果一般。由于训练内容单一，模型并不知道何时停止生成，且过长的生成会造成内容的重复。至于藏头部分，通过手动藏头的方式可以很好的将字放入句首，但也带来了整个诗可能不再连贯和谐的问题，所以实验仍有不小的提升空间。