A5 循环神经网络实验报告

一、任务目标

1、基本实验要求完成：

a. **任务：构建一个用于起名字的循环神经网络**

b. **数据：8000多个英文名字（https://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/）**

c. **采用已有的英文名字，训练一个RNN，实现一个起名字的计算机程序，当输入名字的第一个或前几个字母时，程序自动生成后续的字母，直到生成一个名字的结束符。**

d. **采用可视化技术，绘制出模型为每个时刻预测的前5个最可能的候选字母。**

2、进阶要求：

a. **事实上，你也可以给定结尾的若干个字母，或者随意给出中间的若干个字母，让RNN补全其它字母，从而得到一个完整的名字。请尝试设计并实现一个这样的RNN模型。**

b. **从模型生成的名字中，挑选你最喜欢的一个，并采用可视化技术，绘制生成这个名字的过程。**

二、实验实现过程：

1、实验平台概述：

本次实验采用 Python 完成。

依赖库有：

依赖库	版本
numpy	1.24.3
torch	1.13.0+cu116
matplotlib	3.6.0
adjustText	0.8.0

2、任务代码实现及实现原理：

① 输入名字第一个或前几个字母进行生成

针对第一个基本任务需求，采用两种模型结构进行实现，

第一是用基本的全连接网络搭建基本RNN网络方案，

第二是采用LSTM网络结合全连接层进行实现。

1. 数据集的预处理

根据所提供的数据集地址，将包含宠物名，女性，男性名称的三个txt文件下载，并将无关字符和空行进行去除。之后读取每个文件每行的名字，并根据种类分类，保留种类信息。在通过所有的asc字符以及特殊字符总串构建完整词表，将名字转换为对应的张量格式:

$(L,\ N,\ B)$, L是名字长度， N是批次大小， B指词表大小

同时我们根据训练方式：**已知当前字符，生成下一个字符**这样的训练模式，构建目标字符串，即去除第一个字符，添加最后末尾的结束符号: **'-'**

具体代码如下:

```
origin_datasets = glob.glob('./datasets/*.txt')

all_kinds = []
all_kinds_names = {}
for data in origin_datasets:
    kind = os.path.splitext(os.path.basename(data))[0]
    all_kinds.append(kind)
    one_kind_names = open(data, encoding='utf-8').read().strip().split('\n')
    all_kinds_names[kind] = one_kind_names

num_of_all_kinds = len(all_kinds)
all_letters = string.ascii_letters + " .,;'-"
num_of_all_letters = len(all_letters)

def train_datasets_make():
    kind = random.choice(all_kinds)
    name = random.choice(all_kinds_names[kind])

    kind_tensor = torch.zeros(1, num_of_all_kinds)
    kind_tensor[0][all_kinds.index(kind)] = 1

    input_name_tensor = torch.zeros(len(name), 1, num_of_all_letters)
    for i in range(len(name)):
        letter = name[i]
        input_name_tensor[i][0][all_letters.find(letter)] = 1

    letter_indexes = [all_letters.find(name[j]) for j in range(1, len(name))]
    letter_indexes.append(num_of_all_letters - 1)
```

```
    target_name_tensor = torch.LongTensor(letter_indexes)

    return kind_tensor.to('cuda'), input_name_tensor.to('cuda'), target_name_ten
```

2. 基本RNN模型结构搭建

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_of_all_kinds):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(num_of_all_kinds + input_size + hidden_size, hidden
        self.i2o = nn.Linear(num_of_all_kinds + input_size + hidden_size, output
        self.o2o = nn.Linear(hidden_size + output_size, output_size)
        self.dropout = nn.Dropout(0.1)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, category, input, hidden):
        input_combined = torch.cat((category, input, hidden), 1)
        hidden = self.i2h(input_combined)
        output = self.i2o(input_combined)
        output_combined = torch.cat((hidden, output), 1)
        output = self.o2o(output_combined)
        output = self.dropout(output)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)
```

我们使用基本的全连接层分别定义RNN的输入层，隐藏层，输出层，在最后使用LogSoftmax作为概率输出（即对softmax取log），针对一开始并无序列隐藏信息储存问题，进行了隐藏信息的初始化（都取0）。

模型的输入使用三个信息：名称种类， 当前时刻字符， 隐藏层信息，并进行拼接作为最终输入。最后输出结合RNN提取信息以及其隐藏层信息输入一全连接层得到最后概率输出。

3. LSTM模型搭建

```
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_of_all_kinds, L
        super(LSTM, self).__init__()
        self.lstm = nn.LSTM(input_size+num_of_all_kinds, hidden_size, LSTM_nums,
        self.hidden_size = hidden_size
        self.lstm_layers = LSTM_nums
```

```
        self.bidirectional = bidirectional
        if bidirectional:
            self.__class__.__name__ = 'BiLSTM'
            self.liner_layers = nn.Linear(hidden_size * 2, output_size)
        else:
            self.liner_layers = nn.Linear(hidden_size, output_size)

        self.softmax = nn.LogSoftmax(dim=1)

    def initHidden(self):
        if self.bidirectional:
            return (torch.zeros(self.lstm_layers*2,  self.hidden_size).to('cuda'
        else:
            return (torch.zeros(self.lstm_layers,  self.hidden_size).to('cuda'),

    def forward(self, category, input, hidden):
        input_combined = torch.cat((category, input), 1)
        output, (h_n, c_n) = self.lstm(input_combined, hidden)
        output = self.liner_layers(output)
        output = self.softmax(output)

        return output, (h_n, c_n)
```

在定义LSTM模型过程中利用了torch.nn.LSTM接口，在进行自身LSTM的结构定义过程中要针对是否为双向LSTM进行特殊处理。

LSTM为双向时，LSTM层输出维度特征为**（词长度，批次大小，隐藏层大小*2）**，不为双向时为**（词长度，批次大小，隐藏层大小）**。

LSTM为双向时，其隐藏、记忆信息维度大小为：**（lstm层数*2， 隐藏层大小）**

LSTM为单向时，其隐藏、记忆信息维度大小为：**（lstm层数， 隐藏层大小）**

针对其行为的不同，网络结构要进行适当的调整。

4. 训练步骤实现

在该生成名字的基础任务中，采用的训练方式是根据当前给定名字字符输入生成下一个名字字符的方式，实现起来较为简单：

```
def train(kind_tensor, input_name_tensor, target_name_tensor):
    hidden = rnn.initHidden()
    optimizer.zero_grad()
    target_name_tensor.unsqueeze_(-1)
    loss = torch.tensor(0.0, requires_grad=True).to('cuda')

```

```
    for i in range(input_name_tensor.size(0)):
        output, hidden = rnn(kind_tensor, input_name_tensor[i], hidden)
        loss = loss + criterion(output, target_name_tensor[i])
    loss.backward()
    optimizer.step()
    return output, loss.item()/input_name_tensor.size(0)
```

在定义模型的过程中，特地采用了统一的接口设计处理，可以良好的进行代码复用，其中criterion损失函数采用了CrossEntropyLoss的设计，求得分类概率损失，并进行累计。

5. **预测步骤实现**

由于预测步骤需要考虑前面给定字符，故应定义预测逻辑如下：

a. 先根据给定字符进行网络的前向传播

b. 得到最后一个给定字符后的隐藏层信息

c. 结合当前最后一个字符和隐藏层信息进行生成

```
def predict(kind, first='A'):
    with torch.no_grad():
        kind_tensor = torch.zeros(1, num_of_all_kinds).to('cuda')
        kind_tensor[0][all_kinds.index(kind)] = 1
        input = torch.zeros(len(first), 1, num_of_all_letters).to('cuda')
        for i in range(len(first)):
            input[i][0][all_letters.find(first[i])] = 1
        hidden = rnn.initHidden()
        predict_name = first

        for i in range(7):
            if i==0:
                for j in range(len(first)):
                    output, hidden = rnn(kind_tensor, input[j], hidden)
            output, hidden = rnn(kind_tensor, input[0], hidden)
            tv, ti = output.topk(5)
            words_prob.append(torch.exp(tv)[0].tolist())
            print(torch.exp(tv)[0])
            word_5 = []
            for k in range(5):
                print(all_letters[ti[0][k]], end = ' ')
                word_5.append(all_letters[ti[0][k]])
            words.append(word_5)
            print('\n')

            tv, ti = output.topk(1)
            t = ti[0][0]
```

```
            if(t == num_of_all_letters - 1):
                break
            else:
                predict_name += all_letters[t]

            input = torch.zeros(len(first), 1, num_of_all_letters).to('cuda')
            input[0][0][t] = 1
        return predict_name
```

在实现预测任务的过程中，也进行了为使用matplotlib可视化技术的预备处理，储存每步前五个字符以及其预测概率进行可视化。

② 随意给出名字的任意位置进行补全生成

针对该需求，若简单采用之前的方案会有问题如下：

a. 如何提取并利用名字的位置信息，和将要预测的名字位置信息？

b. 训练方式的确定，是先从第一个可见字符开始向前/向后预测还是其他的方式？

针对上述难点，决定采用**Seq2Seq模型**的方案，先用一个**双向LSTM作为编码器**，提取全局的待补全名字信息，再结合**该提取的全局位置信息与当前待预测字符进行拼接预测出当前位置的名字字符**。

故有实现如下：

1. **数据的掩码式预处理**

对于该模型的最终效果，希望能达成：**对待预测字符进行补全，对给定字符给出原本的字符不做更改**。

故在处理数据时，随机以50%的概率对采样名字进行掩码不超过名字长度一半（降低一些训练难度）的特殊处理，掩码位置根据不重复随机采样获得。

而目标名字的处理则将从之前“下一个字符”转变为“当前字符”：

```
def completion_train_datasets_make():
    kind = random.choice(all_kinds)
    name = random.choice(all_kinds_names[kind])
    target_name = copy.deepcopy(name)

    name_length = len(name)
    if_comple = int(random.uniform(0, 2))

    if if_comple == 1:
        get_times = name_length / 2
        get_locations = random.sample(range(name_length), int(get_times))
```

```
        for i in get_locations:
            s = list(name)
            s[i] = '.'
            name = ''.join(s)

    kind_tensor = torch.zeros(1, num_of_all_kinds)
    kind_tensor[0][all_kinds.index(kind)] = 1

    input_name_tensor = torch.zeros(len(name), 1, num_of_all_letters)
    for i in range(len(name)):
        letter = name[i]
        input_name_tensor[i][0][all_letters.find(letter)] = 1

    letter_indexes = [all_letters.find(target_name[j]) for j in range(0, len(tar
    letter_indexes.append(num_of_all_letters - 1)
    target_name_tensor = torch.LongTensor(letter_indexes)

    return kind_tensor.to('cuda'), input_name_tensor.to('cuda'), target_name_ten
```

对于掩码，使用字符''作为待补全位置标记（正常名字中不含该字符）。

2. Seq2Seq模型构建

```
class Seq2Seq(nn.Module):
    def __init__(self, input_size, encoder_hidden_size, decoder_hidden_size,
                 output_size, num_of_all_kinds, LSTM_nums = 2) -> None:
        super().__init__()

        self.encoder_hidden_size = encoder_hidden_size
        self.decoder_hidden_size = decoder_hidden_size
        self.lstm_layers = LSTM_nums
        self.encoder = nn.LSTM(input_size+num_of_all_kinds, encoder_hidden_size,
                               LSTM_nums, bidirectional=True)
        self.decoder = nn.LSTM(encoder_hidden_size*2 + input_size + num_of_all_k
                               decoder_hidden_size, LSTM_nums)
        self.linear_layer = nn.Linear(decoder_hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def encoder_initHidden(self):
        return (torch.zeros(self.lstm_layers * 2,  self.encoder_hidden_size).to(
            torch.zeros(self.lstm_layers * 2, self.encoder_hidden_size).to('cuda

    def decoder_initHidden(self):
        return (torch.zeros(self.lstm_layers,  self.decoder_hidden_size).to('cud
             torch.zeros(self.lstm_layers, self.decoder_hidden_size).to('cuda'))

```

```
    def encoder_forward(self, category, input, hidden):
        input_combined = torch.cat((category, input), 1)
        output, (h_n, c_n) = self.encoder(input_combined, hidden)
        return output, (h_n, c_n)

    def decoder_forward(self, category, input, hidden, encoder_output):
        input_combined = torch.cat((category, input, encoder_output), 1)
        output, (h_n, c_n) = self.decoder(input_combined, hidden)
        output = self.linear_layer(output)
        output = self.softmax(output)
        return output, (h_n, c_n)
```

对于编码器部分，使用一个双向的LSTM层进行全局位置信息提取。

对于解码器部分，结合了一个LSTM层及全连接层和logsoftmax函数进行对给予全局信息和当前字符的预测。

因为使用了LSTM的原因，分别对编解码器的LSTM层进行隐藏层初始化，也分别对其前向传播进行了定义。

编码器：

$$feats = BiLSTM(concat(category, inputchars, encoder\_hidden))$$

解码器：

$$context\_output = LSTM(concat(feats, givenchar, decoder\_hidden))$$
$$output = LogSoftmax(Linear(context\_output))$$

3. 训练步骤

前文已给出当前训练思路：

> 结合编码器提取的名字全文待补全信息和当前给定字符，进行当前字符预测。

```
def completion_train(kind_tensor, input_name_tensor, target_name_tensor):
    optimizer.zero_grad()
    target_name_tensor.unsqueeze_(-1)
    loss = torch.tensor(0.0, requires_grad=True).to('cuda')

    encoder_hidden = rnn.encoder_initHidden()
    decoder_hidden = rnn.decoder_initHidden()

    # 编码器先获取全局信息
    # 判断是否为补全任务
    iscompletion = False
    for i in range(input_name_tensor.size(0)):
        if input_name_tensor[i][0][all_letters.find('.')] == 1:
            iscompletion = True # 存在缺项则为补全任务，需要提取全局信息
```

```
15
16      encoder_output = torch.zeros(1, rnn.encoder_hidden_size * 2).to('cuda')
17      if iscompletion:
18          for i in range(input_name_tensor.size(0)):
19              encoder_output, encoder_hidden = rnn.encoder_forward(kind_tensor,
20                                               input_name_tensor[i], encoder_hidden)
21
22      # 解码器根据编码器编码信息和输入信息进行补全训练或进行普通训练
23      for i in range(input_name_tensor.size(0)):
24          output, decoder_hidden = rnn.decoder_forward(kind_tensor, input_name_ten
25                                                       decoder_hidden, encoder_output)
26          loss = loss + criterion(output, target_name_tensor[i])
27
28      loss.backward()
29      optimizer.step()
30      return output, loss.item()/input_name_tensor.size(0)
```

4. 预测步骤

```
1  def completion_predict(kind, first='A'):
2      with torch.no_grad():
3          encoder_hidden = rnn.encoder_initHidden()
4          decoder_hidden = rnn.decoder_initHidden()
5
6          kind_tensor = torch.zeros(1, num_of_all_kinds).to('cuda')
7          kind_tensor[0][all_kinds.index(kind)] = 1
8          input = torch.zeros(len(first), 1, num_of_all_letters).to('cuda')
9          predict_name = first
10
11         iscompletion = False
12         completion_locations = []
13         encoder_output = torch.zeros(1, rnn.encoder_hidden_size * 2).to('cuda')
14         for i in range(len(first)):
15             input[i][0][all_letters.find(first[i])] = 1
16             if first[i] == '.':
17                 iscompletion = True
18                 completion_locations.append(i)
19
20         if iscompletion:
21             for i in range(len(first)):
22                 encoder_output, encoder_hidden = rnn.encoder_forward(kind_tensor
23                 input[i], encoder_hidden)
24
25         words_prob = []
26         words = []
```

```
27
28          for j in range(len(first)):
29              output, decoder_hidden = rnn.decoder_forward(kind_tensor, input[j],
30                                                           decoder_hidden, encoder_output)
31              tv, ti = output.topk(5)
32              words_prob.append(torch.exp(tv)[0].tolist())
33              print(torch.exp(tv)[0])
34              word_5 = []
35              for k in range(5):
36                  print(all_letters[ti[0][k]], end = ' ')
37                  word_5.append(all_letters[ti[0][k]])
38              words.append(word_5)
39              print('\n')
40              if j in completion_locations:
41                  tv, ti = output.topk(1)
42                  t = ti[0][0]
43                  if(t == num_of_all_letters - 1):
44                      is_end = True
45                      s1 = list(predict_name)
46                      s1[j] = ''
47                      predict_name = ''.join(s1)
48                      break
49                  s1 = list(predict_name)
50                  s1[j] = all_letters[t]
51                  predict_name = ''.join(s1)
52
53          plt.figure()
54          plt.plot(list(range(1, len(words_prob)+1)), words_prob,linestyle='', mar
55          texts = []
56          for x, y, word in zip(list(range(1, len(words_prob)+1)), words_prob, wor
57              for w, prob in zip(word, y):
58                  texts.append(plt.text(x, prob, w))
59
60          adjust_text(texts, )
61
62          plt.show()
63          plt.savefig('./completion_words_map.png')
64
65          return predict_name
```

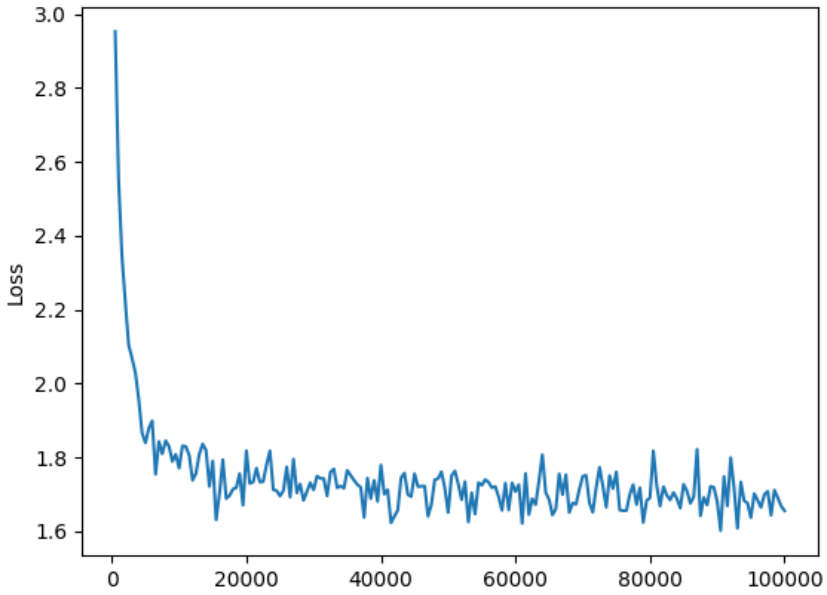
预测实现除要使用编码器提取全局信息这一不同外，还需要对''字符进行替换，以给出最终结果。

3、具体实验结果以及结果分析：

在处理中，我们都保持了网络的隐藏层大小一致，使用了LSTM的模型的LSTM层数仅设为1.

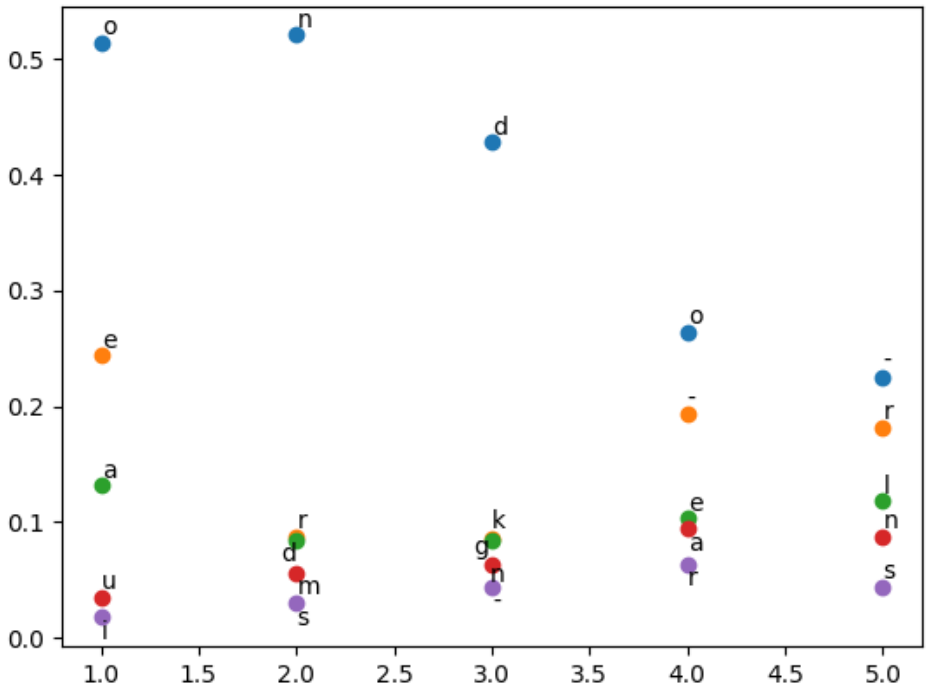
① 输入名字第一个或前几个字母进行生成（Basic RNN):

1. **损失下降曲线：**



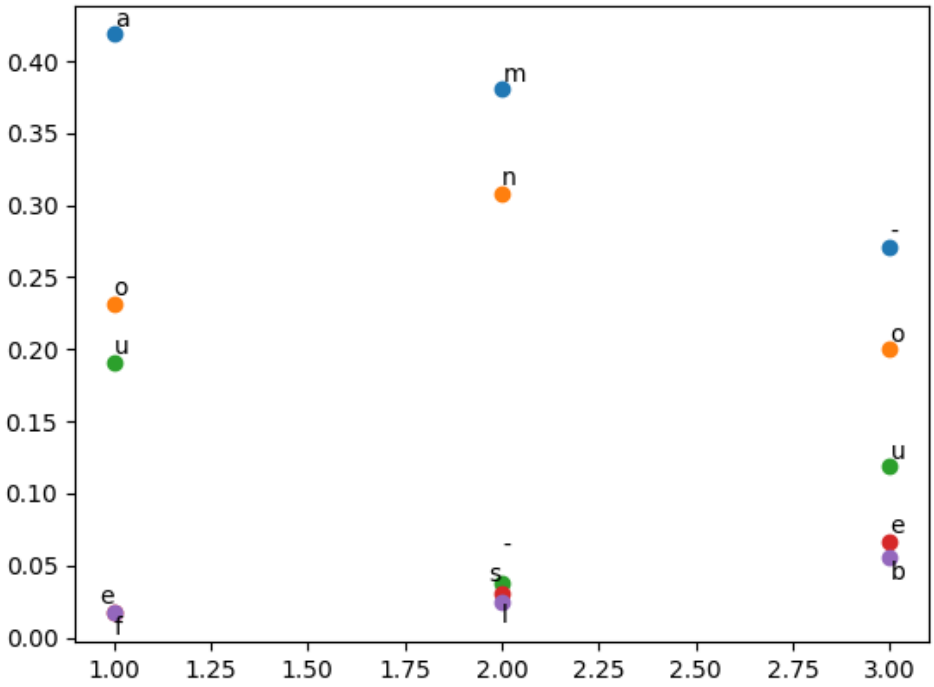
2. **可视化名字生成**

   a. 给定Har -- python main.py Test RNN rnn_name_100000_.pth male Har



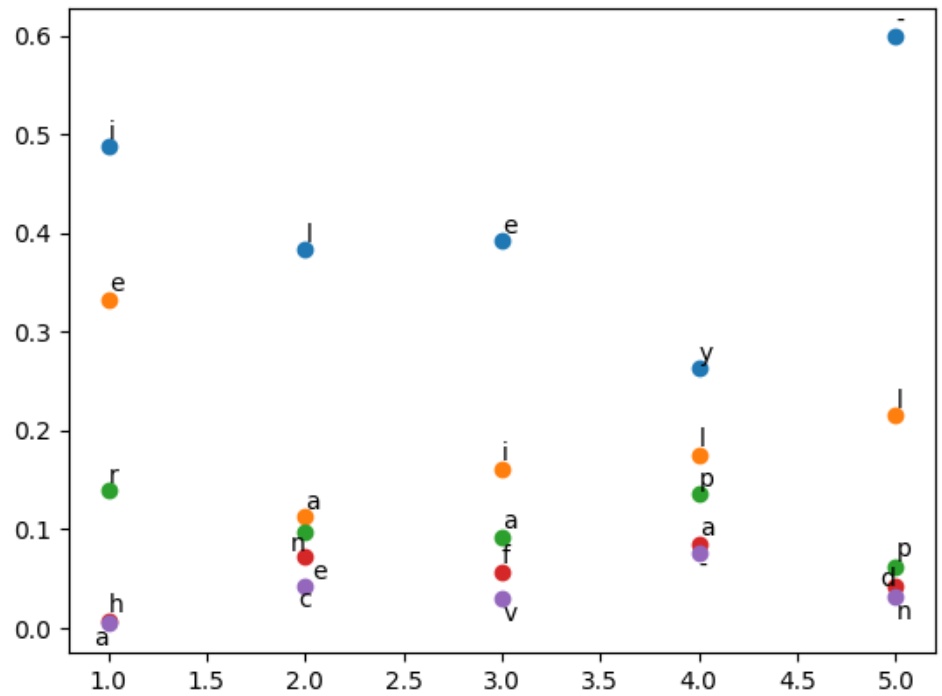
该图中给出了生成名：**Harondo(哈伦多)**的生成示例，纵坐标代表前五个字符对应的概率大小，横坐标为步数。

   b. 给定Joh -- python main.py Test RNN rnn_name_100000_.pth male Joh



该图中给出了生成名：**Joham(约翰姆)**的生成示例。

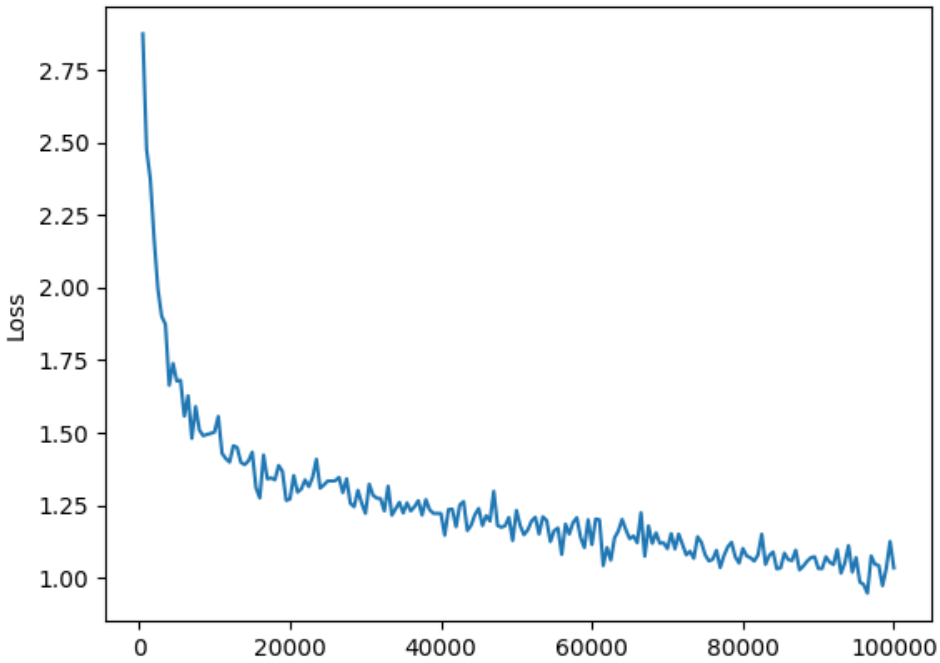
c. 给定Ph -- python main.py Test RNN rnn_name_100000_.pth female Ph



该图中给出了生成名：**Philey(菲莉)**的生成示例。

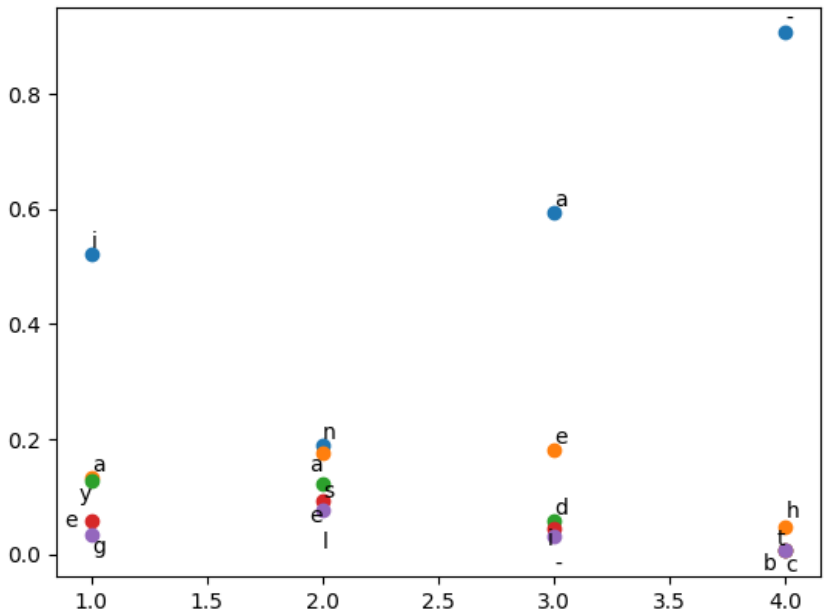
② 输入名字第一个或前几个字母进行生成（LSTM):

1. **损失下降曲线**



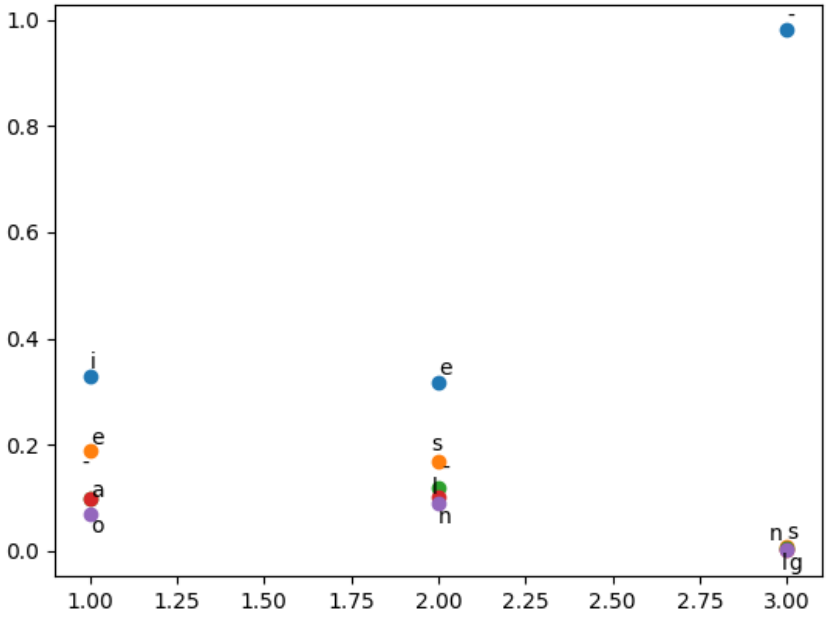
2. **可视化名字生成**

   a. 给定Ma -- python main.py Test LSTM LSTM100000_.pth female Ma



该图中给出了生成名：**Maina(麦娜)**的生成示例。

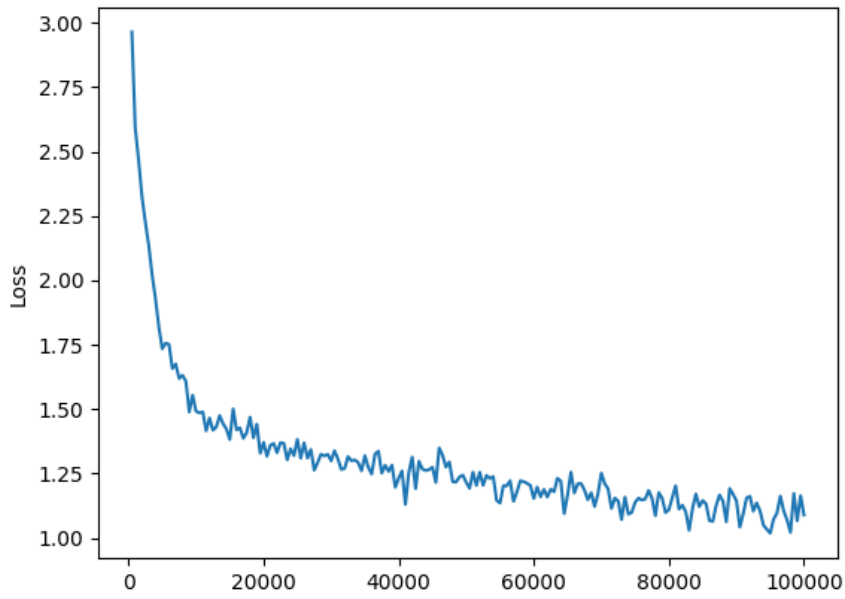
   b. 给定Lo -- python main.py Test LSTM LSTM100000_.pth male Lo



该图中给出了生成名：**Loie(洛伊)**的生成示例。

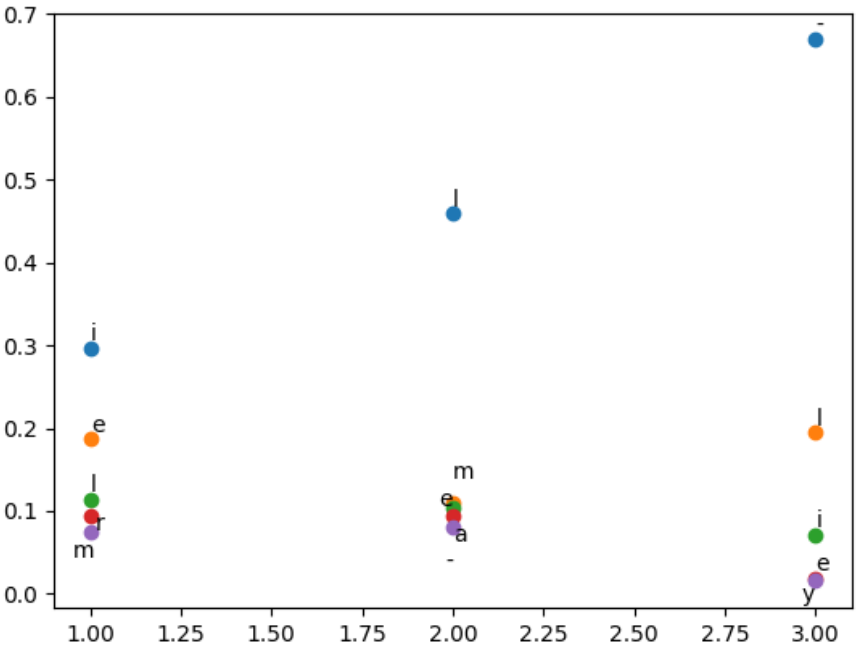
③ 输入名字第一个或前几个字母进行生成（BiLSTM):

1. **损失下降曲线**



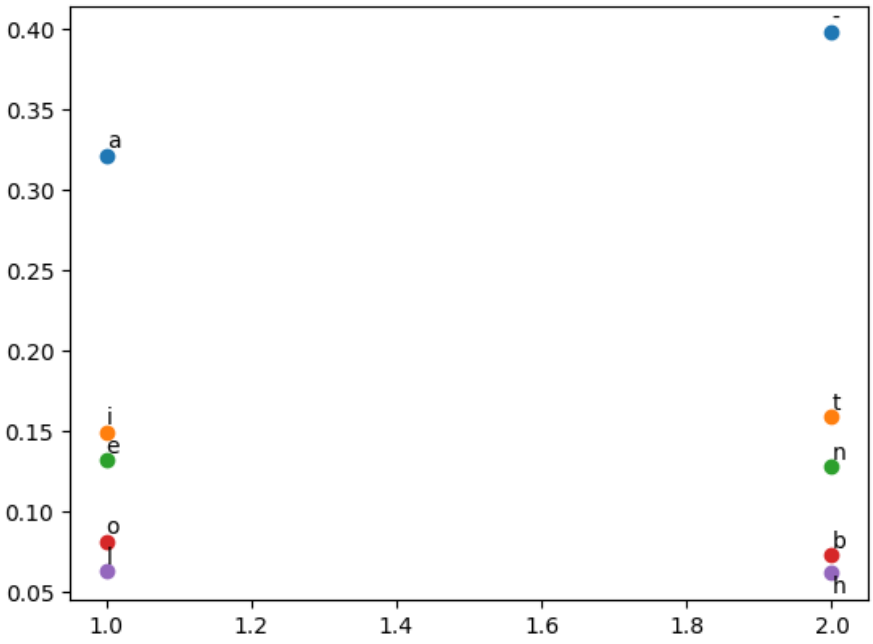
2. **可视化名字生成**

   a. 给定Ka -- python main.py Test BiLSTM BiLSTM100000_.pth male Ka



该图中给出了生成名：**Kail(凯尔)**的生成示例。

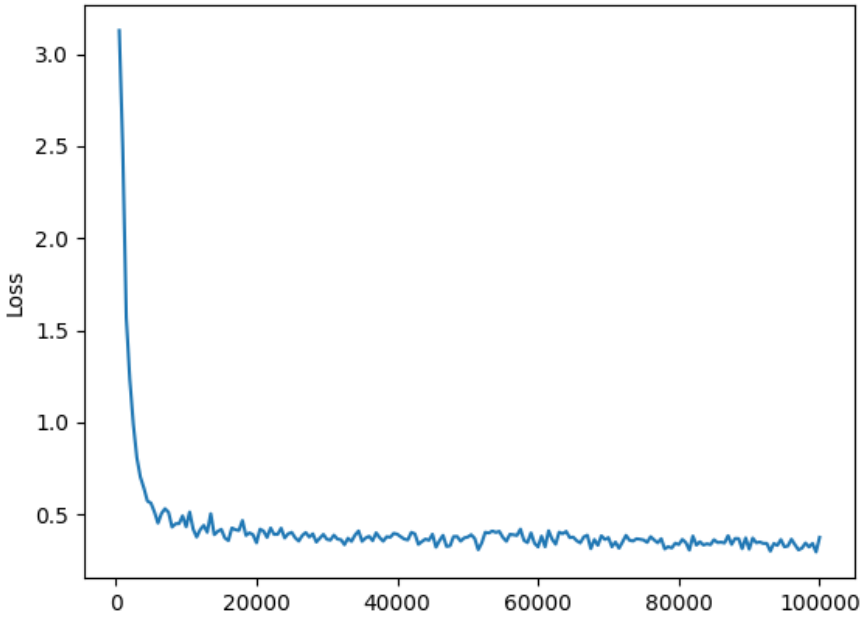
b. 给定Li -- python main.py Test LSTM LSTM100000_.pth female Li



该图中给出了生成名：**Lia(莉娅)**的生成示例。

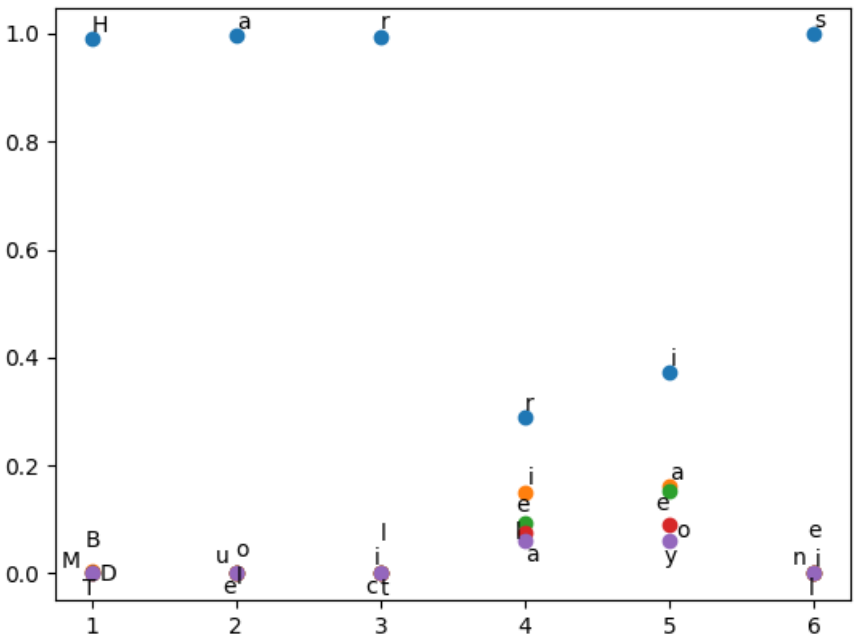
③ 随意给出名字的任意位置进行补全生成（Seq2Seq):

1. **损失下降曲线**



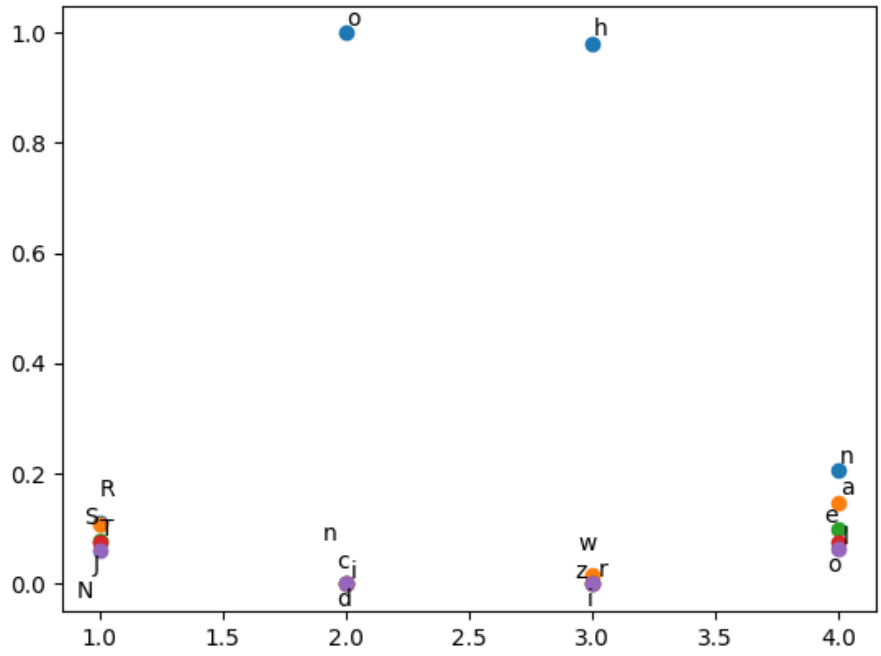
2. **可视化名字生成**

   a. 给定 Har..s -- python main.py CompletionTest Seq2Seq Seq2Seq100000_.pth male Har..s



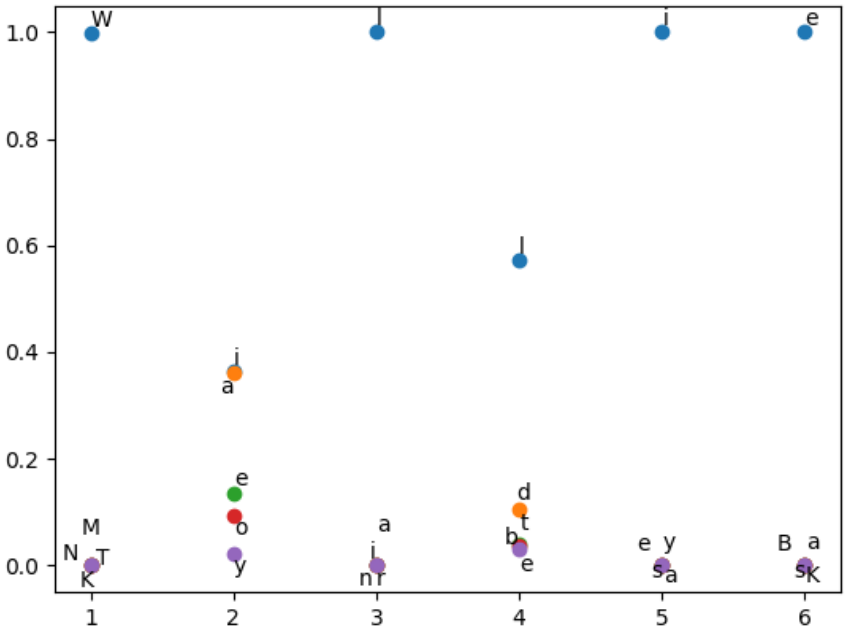
该图中给出了补全名：**Harris(哈里斯)**的补全示例，**可观测模型对于给定确定字符一定将其收敛到原本值，对于缺失项进行正常预测**

   b. 给定.oh. -- python main.py CompletionTest Seq2Seq Seq2Seq100000_.pth male .oh.



该图中给出了补全名：**Rohn(罗恩)**的补全示例。

c. 给定W.l.ie -- python main.py CompletionTest Seq2Seq Seq2Seq100000_.pth female W.l.ie



该图中给出了补全名：**Willie(威莉娅)**的补全示例。

三、结语：

🎉 在本次实验中，先进行了RNN，LSTM的基本实现，后又根据补全任务需要，尝试搭建了Seq2Seq架构的NLP模型。

从损失下降上来看，在根据前面字符进行名字生成的任务上，BiLSTM损失小于LSTM，LSTM损失小于基础的RNN网络。从生成结果上看都能取得良好的效果。

在最后一个补全任务中，我更改了训练方式与模型结构，先用编码器提取待补全的全文信息，再结合当前给定字符进行推理，同时不断传递隐藏层信息，使模型能够良好收敛，进行补全。

最后用Matplotlib库实现了概率，步长上的可视化，并利用adjustText库消除了文字重叠问题。

感谢本次实验机会，使我更加熟悉NLP相关模型设计、训练技术的应用，也体悟到NLP的乐趣！