构建用于起名的循环神经网络

神经网络与深度学习课程实验报告

班级：110

姓名：耿翊中

学号：2021213382

2023 年 12 月

目录

1. 实验要求

A5：循环神经网络 (20 points)
任务：构建一个用于起名字的循环神经网络
• 数据：8000 多个英文名字（https://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/）
• 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。
• 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。

附加题（10 分）：
• 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。
• 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。

背景知识：给定大量的文本数据，可训练一个基于循环神经网络的语言模型，该模型可用于计算一个句子的出现概率（P(w1, w2, ..., wT)），或者根据上文中的词推断某个词作为下一个词的出现概率（P(wt | w1, w2, ..., wt-1)）。

2. 实验原理

2.1 模型原理

2.1.1 RNN 原理

循环神经网络（RNN）是一种用于处理序列数据的神经网络结构，其基本原理是网络中存在循环连接，使得网络具有记忆能力，能够捕捉时间序列中的依赖关系。 RNN 的核心结构包括一个隐藏状态向量，该向量随着输入序列的每个时间步进行更新，从而在不同时间步传递信息。下面是一个 RNN 的示意图 1：

RNN 的工作过程可以描述为：

对于输入序列 X=(x1 ,x2 ,...,xT)，RNN 首先初始化一个隐藏状态 h0 ，通常为零向量。

对于每个时间步 t=1,2,...,T，RNN 执行以下操作：

计算当前时间步的隐藏状态 ht =f(Uxt +Wht-1 +b)，其中 U,W,b 是可学习的参数，f 是一个非线性激活函数，如 tanh 或 ReLU。

计算当前时间步的输出 ot =g(Vht +c)，其中 V,c 是可学习的参数，g 是一个输出函数，如 softmax 或 sigmoid。

输出序列 O=(o1 ,o2 ,...,oT) 作为 RNN 的预测结果，可以用于不同的任务，如分类、回归、生成等。

RNN 的优点是它可以处理任意长度的序列，并且可以利用长期的上下文信息。

2.1.2 RNN 用于起名的原理

为了生成名字，我们可以使用 RNN 来学习一个语言模型，也就是给定一个字母或一个词的前缀，预测下一个字母或词的概率分布。例如，给定一个字母"a"，我们希望 RNN 能够输出一个概率向量，表示以"a"开头的名字中，下一个字母可能是什么，以及它们的概率有多大。例如，RNN 可能会输出这样一个向量：

b	c	d	e	f	…	z
0.1	0.05	0.02	0.3	0.01	…	0.01

这个向量表示，以"a"开头的名字中，下一个字母最有可能是"e"，其次是"b"，然后是"c"，依次类推。我们可以根据这个向量，随机地选择一个字母作为下一个字母，比如我们选择了"e"，那么我们就得到了一个新的前缀"ae"，然后我们再把这个前缀输入到 RNN 中，得到下一个字母的概率分布，重复这个过程，直到生成一个完整的名字或者遇到一个结束符。

为了训练 RNN，我们需要准备一些已有的名字作为数据集，在这个实验中我们选择了 https://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/ 中的数据集。我们把每个名字看作一个序列，每个字母或词看作一个时间步，然后把每个序列的输入和输出错开一个时间步，作为 RNN 的训练数据。例如，如果一个名字是"Adam"，那么我们可以把它分成两个序列，一个是输入序列，一个是输出序列，如下所示：

输入序列	输出序列
<start> A	A d
A d	d a

输入序列	输出序列
d a	a m
a m	m <end>

其中，<start>和<end>是特殊的符号，表示序列的开始和结束。我们把每个字母或词转换成一个特征向量，作为 RNN 的输入层，然后把每个输出序列的每个时间步的字母或词转换成一个 one-hot 向量，作为 RNN 的输出层。我们使用一个损失函数，比如交叉熵损失，来衡量 RNN 的输出和真实的输出之间的差异，然后使用反向传播算法和梯度下降法来更新 RNN 的参数，使得损失函数最小化。

RNN 的结构可以用下图来表示，其中 X 是输入层，S 是隐藏层，O 是输出层，U、V、W 是参数矩阵，f 和 g 是激活函数，t 是时间步：

RNN 的计算过程可以用下面的公式来表示：

$$S_t = f(U \cdot X_t + W \cdot S_{t-1}) O_t = g(V \cdot S_t)$$

其中,St 表示第 t 个时间步的隐藏层的值,Xt 表示第 t 个时间步的输入层的值,Ot 表示第 t 个时间步的输出层的值，U、V、W 表示输入层到隐藏层、隐藏层到输出层、隐藏层到隐藏层的参数矩阵，f 和 g 表示激活函数。

2.2 代码实现

2.2.1 RNN 神经网络实现

RNN 神经网络的实现是基于[PyTorch]框架的，它的主要功能是根据输入的类别和字符序列，生成一个输出的字符序列，例如用于生成名字或诗歌等。

首先，定义了一个 RNN 类，继承了 nn.Module 类，这是 PyTorch 中定义自定义神经网络模型的标准方式。

然后，在__init__方法中，初始化了一些参数和层：

self.hidden_size 是隐藏状态向量的维度，也就是 RNN 的记忆单元的大小。

self.i2h 是一个全连接层，用于计算当前时间步的隐藏状态，它的输入是类别向量、当前字符向量和上一个时间步的隐藏状态向量的拼接，它的输出是一个新的隐藏状态向量。

self.i2o 是一个全连接层，用于计算当前时间步的输出，它的输入和 self.i2h 的输入相同，它的输出是一个输出向量，表示当前时间步的预测结果。

self.o2o 是一个全连接层，用于进一步处理当前时间步的输出，它的输入是当前时间步的隐藏状态向量和输出向量的拼接，它的输出是一个新的输出向量，表示当前时间步的最终预测结果。

self.dropout 是一个随机失活层，用于防止过拟合，它的概率为 0.1，表示每个时间步有 10%的输出单元被随机置为 0。

self.softmax 是一个对数 softmax 层,用于将输出向量转换为概率分布,它的维度为 1，表示对第 1 维度的数据进行 logsoftmax 操作，这里第 1 维度表示输出向量的长度，也就是字符集的大小。

接下来，在 forward 方法中，定义了前向传播的逻辑：

首先，将类别向量、当前字符向量和上一个时间步的隐藏状态向量拼接起来，得到一个临时的输入向量 input_tmp。

然后，将 input_tmp 输入到 self.i2h 层，得到当前时间步的隐藏状态向量 hidden。

接着，将 input_tmp 输入到 self.i2o 层，得到当前时间步的输出向量 output。

再然后，将当前时间步的隐藏状态向量和输出向量拼接起来，得到一个临时的输出向量 output_tmp。

再接着，将 output_tmp 输入到 self.o2o 层，得到当前时间步的最终输出向量 output。

再再然后，将 output 输入到 self.dropout 层，进行随机失活操作，得到一个稀疏的输出向量 output。

最后，将 output 输入到 self.softmax 层，进行对数 softmax 操作，得到一个概率分布向量 output，表示当前时间步的预测结果。

返回 output 和 hidden 作为 RNN 的输出，其中 output 用于计算损失函数和评估性能，hidden 用于传递给下一个时间步。

最后，在 init_hidden 方法中，定义了隐藏状态向量的初始化操作：

返回一个全零的向量，其形状为(1, self.hidden_size)，表示一个批次大小为 1，维度为 self.hidden_size 的隐藏状态向量。

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(n_categories + input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(n_categories + input_size + hidden_size, output_size)
        self.o2o = nn.Linear(hidden_size + output_size, output_size)
        self.dropout = nn.Dropout(0.1)
        self.softmax = nn.LogSoftmax(dim=1)  # dim=1 表示对第 1 维度的数据进行 logsoftmax 操作

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

2.2.2 训练 RNN 神经网络

这个函数是用于对 RNN 神经网络进行训练的，它的主要原理是使用随机梯度下降（SGD）算法来优化网络的参数，使得网络能够根据输入的类别和字符序列，生成一个输出的字符序列，例如用于生成名字或诗歌等。

首先，创建一个 RNN 对象，指定输入、隐藏和输出的维度，然后将其转移到设备上，这里的设备可以是 CPU 或 GPU，取决于运行环境。

然后，定义一个负对数似然损失函数（NLLLoss），用于计算网络的输出和目标之间的差异，以及一些变量，用于记录总的损失、每个批次的损失、训练的轮数、学习率和开始时间等。

接下来，进入一个循环，对每个训练轮数（epoch）进行以下操作：

首先，将网络的梯度清零，这是为了避免梯度累积，影响优化过程。

然后，初始化一个隐藏状态向量，用于存储网络的记忆，以及一个训练损失变量，用于累加每个时间步的损失。随机生成一个训练样本，包括类别向量、输入字符向量和目标字符向量，这些向量都是用独热编码（one-hot encoding）表示的，即每个向量只有一个位置为1，其余位置为0，表示对应的类别或字符。

再接着，将目标字符向量在最后一个维度上增加一个维度，这是为了符合损失函数的输入要求，即目标向量的形状应该为(批次大小，类别数)。

再然后，进入一个循环，对输入字符向量的每个时间步进行以下操作：

首先，将类别向量、当前字符向量和上一个时间步的隐藏状态向量输入到网络中，得到当前时间步的输出向量和隐藏状态向量，这些向量都是用实数表示的，表示对应的概率或值。将当前时间步的输出向量和目标向量输入到损失函数中，计算当前时间步的损失，然后将其累加到训练损失变量中，这样就可以得到每个训练样本的总损失。对训练损失变量进行反向传播，即根据损失函数对网络的参数进行求导，得到每个参数的梯度，这些梯度表示了参数对损失的影响程度，用于指导参数的更新方向。

再接着，进入一个循环，对网络的每个参数进行以下操作：

首先，根据参数的梯度和学习率，计算参数的更新量，即梯度乘以学习率的负数，表示参数应该向梯度的反方向移动，以减小损失。然后，将参数的更新量加到参数的数据上，即更新参数的值，这样就完成了一次参数的优化。再然后，将训练损失除以输入字符向量的长度，得到该训练样本的平均损失，然后将其累加到总的损失变量中，这样就可以得到每个批次的平均损失。

最后，根据一些条件，打印当前的训练进度、时间和损失，或者将总的损失添加到一个列表中，用于绘制损失曲线，或者保存网络的参数到一个文件中，用于后续的测试或使用。

```
def train():
    rnn = RNN(n_letters, n_hidden, n_letters).to(device)

    loss = nn.NLLLoss()
    total_loss = 0
    all_losses = []
    epoch_num = 100000
    lr = 0.0005
    epoch_start_time = time.time()

    for epoch in range(epoch_num):
        rnn.zero_grad()
        train_loss = 0
        hidden = rnn.init_hidden()
        category_tensor, input_tensor, target_tensor = random_training_example()
        target_tensor.unsqueeze_(-1)

        for i in range(input_tensor.size()[0]):
```

```
            output, hidden = rnn(category_tensor, input_tensor[i], hidden)
            train_loss += loss(output, target_tensor[i])  # 每次的 loss 都需要计算
        train_loss.backward()

        for p in rnn.parameters():
            p.data.add_(p.grad.data, alpha=-lr)

        total_loss += train_loss.item() / input_tensor.size()[0]  # 该名字的平均损失

        if epoch % 5000 == 0:
            print('[%05d/%03d%%] %2.2f sec(s) Loss: %.4f' %
                  (epoch, epoch / epoch_num * 100, time.time() - epoch_start_time,
                   train_loss.item() / input_tensor.size()[0]))
            epoch_start_time = time.time()

        if (epoch + 1) % 500 == 0:
            all_losses.append(total_loss / 500)
            total_loss = 0

    torch.save(rnn.state_dict(), '../model/rnn_params.pkl')  # 保存模型的数据

    plt.figure()
    plt.plot(all_losses)
    plt.show()
```

2.2.3 使用 RNN 神经网络生成名字

使用 predict()函数，接收类别和字母，而后生成名字

这个函数是用于对 RNN 神经网络进行预测的，它的主要功能是根据输入的类别和起始字母，生成一个输出的字符序列，例如用于生成名字或诗歌等。

首先，创建一个 RNN 对象，指定输入、隐藏和输出的维度，然后将其转移到设备上，这里的设备可以是 CPU 或 GPU，取决于运行环境。加载一个文件中保存的网络的参数，这些参数是之前训练网络时得到的，用于初始化网络的权重和偏置。接下来，定义一个最大长度变量，表示输出字符序列的最大长度，这里设置为 20，表示最多生成 20 个字符。

接下来，将输入的类别和起始字母转换为独热编码的向量，分别赋值给类别向量和输入向量，这些向量都是用 0 和 1 表示的，表示对应的类别或字符。然后，初始化一个隐藏状态向量，用于存储网络的记忆，以及一个输出名字变量，用于拼接生成的字符，初始值为起始字母，还有一个列表变量，用于存储每个时间步的前 5 个字符及其概率，初始值为空。

接下来，进入一个循环，对每个时间步进行以下操作，直到达到最大长度或者生成结束符为止：

首先，将类别向量、当前字符向量和上一个时间步的隐藏状态向量输入到网络中，得到当前时间步的输出向量和隐藏状态向量，这些向量都是用实数表示的，表示对应的概率或值。然后，对当前时间步的输出向量进行排序，选出前 5 个最大的值和对应的索引，这些值和索引表示当前时间步的预测结果的前 5 个候选字符及其概率。

最后，返回输出名字变量和列表变量，作为预测函数的输出，其中输出名字变量表示生成的字符序列，列表变量表示每个时间步的预测结果的前 5 个候选字符及其概率。

```
    def predict(category, start_letter):
        rnn = RNN(n_letters, n_hidden, n_letters).to(device)
        rnn.load_state_dict(torch.load('../model/rnn_params.pkl'))  # 加载模型训练所得到的参数
        max_length = 20  # 名字的最大长度
        with torch.no_grad():
            category_tensor = category_to_tensor(category)
            input = input_to_tensor(start_letter)
            hidden = rnn.init_hidden()

            output_name = start_letter
            top5_each_step = []  # 用于存储每步的前 5 个字符及其概率

            for i in range(max_length):
                output, hidden = rnn(category_tensor, input[0], hidden)
                top_v, top_i = output.topk(5)  # 选出最大的值，返回其 value 和 index

                top5_each_step.append((top_i, top_v))

                top_i = top_i[0][0].item()
                if top_i == n_letters - 1:  # n-letters-1 是 EOS
                    break
                else:
                    letter = all_letters[top_i]
                    output_name += letter
                input = input_to_tensor(letter)  # 更新 input，继续循环迭代
        return output_name, top5_each_step
```

3. 实验设置

3.1 数据准备

3.1.1 数据集选择

从 https://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/）下载了 8000 多个英文名字

并根据类别分为 3 个 txt 文件，分别为 male.txt,female.txt,pet.txt。这样就可以分类别训练生成名字。

3.1.2 读取数据

使用函数对数据进行读取。

find_files(path):

功能: 此函数用于获取指定路径下的所有文件。

参数: path - 要搜索的文件路径，可以包含通配符（如*.txt）。

返回值: 返回匹配指定路径模式的文件列表。
实现细节: 使用 glob.glob(path)从目录通配符搜索中生成文件列表。glob 模块能够找到匹配特定模式的所有路径名（类似于 Unix 的 shell 规则）。

unicode_to_ascii(str):
功能: 此函数将 Unicode 字符串转换为 ASCII，通常用于处理包含特殊字符或重音符号的国际化文本数据。
参数: str - 需要转换的字符串。
返回值: 转换为 ASCII 的字符串。
实现细节: 使用 unicodedata.normalize('NFD', str)将字符串标准化。这里的'NFD'表示字符应该被分解为多个组合字符表示。接着，通过列表推导式过滤掉所有的组合字符（类别为'Mn'的 Unicode 字符）并检查每个字符是否在 all_letters（未在代码中定义，可能是一个包含所有允许字符的集合）中。

read_lines(files_list):
功能: 读取给定文件列表中每个文件的内容。
参数: files_list - 文件路径列表。
返回值: 一个包含类别和相应行（名字列表）的字典。
实现细节:
遍历文件列表。
使用 os.path.splitext(os.path.basename(file))[0]从文件名中提取类别（假设文件名即为类别名）。
读取每个文件的行，并对每行应用 unicode_to_ascii 函数进行处理。
将处理后的行存储在 category_lines 字典中，键为类别，值为名字列表。
all_categories 列表存储所有不同的类别。

3.1.3 数据的张量化

使用函数是将数据转换为张量（Tensor）的形式，这是在使用循环神经网络（RNN）进行名字生成等机器学习任务时的常见步骤。

category_to_tensor(category):
功能: 将类别转换为 one-hot 编码的张量。
参数: category - 类别名称。
返回值: One-hot 编码的张量。
实现细节:
找到类别在 all_categories 列表中的索引。
创建一个全为 0 的张量，其形状为(1, n_categories)，其中 n_categories 是类别总数。
将对应类别索引位置的元素设置为 1，实现 one-hot 编码。
将张量转移到指定的设备（如 GPU）上。
在 RNN 名字生成模型中，这种 one-hot 编码允许模型根据类别生成特定风格的名字。

input_to_tensor(input):
功能: 将单个名字的每个字母转换为 one-hot 编码的张量。
参数: input - 名字字符串。

返回值: 代表名字的 one-hot 编码张量。
实现细节:
创建一个形状为(len(input), 1, n_letters)的全 0 张量，n_letters 是字母表中字母的总数。
遍历名字中的每个字母，找到其在字母表 all_letters 中的索引，并在张量中相应位置设置为 1。
将张量转移到指定设备。
在 RNN 中，这种表示允许模型逐字母地处理输入名字，并学习字母序列的模式。

target_to_tensor(input):
功能: 创建一个代表目标输出的张量，即名字中从第二个字母开始的字母索引，再加上一个特殊的结束符（EOS）。
参数: input - 名字字符串。
返回值: 目标输出的张量。
实现细节:
对于输入字符串中的每个字母（除了第一个字母），找到其在字母表中的索引。
将这些索引加入列表，并在末尾添加一个表示结束（EOS）的特殊索引。
创建一个长整型张量，包含这些索引。
将张量转移到指定设备。

在 RNN 训练中，目标张量用于指导模型学习在给定当前字母时预测下一个字母。EOS 符号表示名字结束，对于模型来说是一个重要的信号。

3.2 训练和生成名字

训练和生成名字的具体函数已经在之前模型原理章节中提到过了，这里不再赘述，只做简单说明。

训练模型后将模型的参数以.pkl 文件形式，存储下来，在生成名字时载入，并进行生成即可。并且生成名字时，将每一时刻前五可能的候选字母进行记录，传递下来，供可视化模块使用。

3.3 可视化处理

3.3.1 绘制模型每个时刻预测的前 5 个最可能的候选字母

使用函数 plot_predictions(top5_each_step) 用于可视化一个模型在每个时刻预测的前 5 个最可能的候选字母。函数的工作流程和细节如下：

设置图表:

使用 plt.figure(figsize=(22, 12)) 设置图表的大小。

使用 plt.title("Top 5 Predictions at Each Step") 给图表添加标题。

遍历每个时间步:

函数通过 for step, (topi, topv) in enumerate(top5_each_step) 遍历每个时间步。这里，top5_each_step 应该是一个包含每个时间步的前 5 个预测和对应概率的列表或元组。

处理和转换数据:

对于每个时间步，函数提取前 5 个预测的字符索引（topi）和相应的对数概率值（topv）。

字符索引被转换为字符。如果索引对应特殊的结束符（EOF），则使用 'EOF' 字符串替换；否则，使用 all_letters 列表（应包含所有可能的字符）转换索引为字符。

对数概率值通过 topv[0].exp() 转换回实际概率值。

创建条形图:

使用 plt.subplot(len(top5_each_step), 1, step + 1) 创建每个时间步的条形图。这样为每个时间步创建了一个单独的子图。

plt.bar(characters, values.cpu(), align='center', alpha=0.7) 生成具有字符标签和对应概率值的条形图。

设置 y 轴的范围为 0 到 1（plt.ylim(0, 1)），因为概率值的范围是 0 到 1。

为条形图添加数值标签:

在每个条形图上方添加标签，显示每个条的实际概率值。这是通过 plt.text() 函数实现的。

调整布局并展示图表:

使用 plt.tight_layout() 调整子图的布局，确保它们不会重叠。

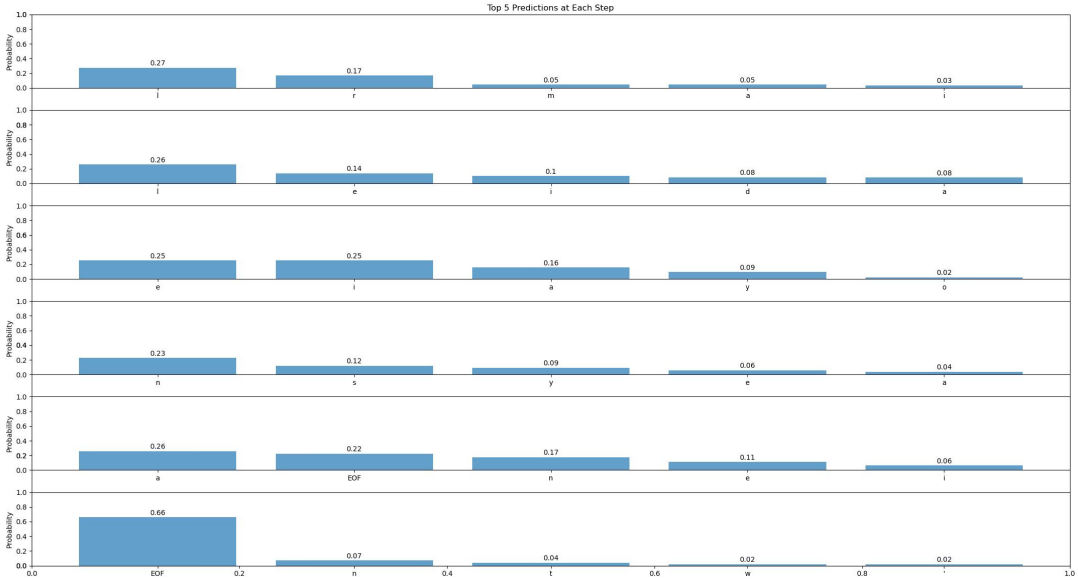
使用 plt.show() 展示最终的图表。

```
    def plot_predictions(top5_each_step):
        # 设置图表大小和标题
        plt.figure(figsize=(22, 12))
        plt.title("Top 5 Predictions at Each Step")

        # 遍历每个时间步
        for step, (topi, topv) in enumerate(top5_each_step):
            # 获取前 5 个字符及其概率
            characters = []
            for i in topi[0]:
                if i.item() == 58:
                    characters.append('EOF')
                else:
                    characters.append(all_letters[i])
            # characters = [all_letters[i] for i in topi[0]]
            values = topv[0].exp()  # 将 LogSoftmax 转换回概率

            # 创建条形图
            plt.subplot(len(top5_each_step), 1, step + 1)
            bars = plt.bar(characters, values.cpu(), align='center', alpha=0.7)
            plt.ylim(0, 1)  # 设置 y 轴范围
            plt.xticks(characters)
            plt.ylabel('Probability')

            # 为每个条形图添加数值标签
            for bar in bars:
                yval = bar.get_height()
                plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.01, round(yval, 2), ha='center', va='bottom')

        # 调整布局
```

```
31.     plt.tight_layout()
32.     plt.show()
```

3.3.2 可视化效果

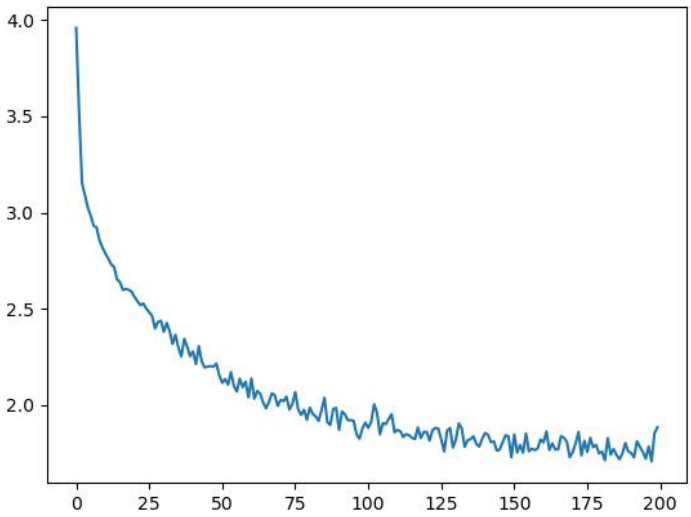
如图为给出首字母 A，生成名字 Allena 的效果图



4. 实验结果分析

4.1 实验结果

进行训练后，得到训练的损失函数变化图为：



对首字母从 A-Z 分别预测生成的女生名字为：

'Arine', 'Barie', 'Carie', 'Dannyl', 'Elle', 'Farley', 'Garia', 'Hobbastie', 'Ianna', 'Jone', 'Keley', 'Larie', 'Marie', 'Nelie', 'Odie', 'Pinel', 'Qone', 'Rando', 'Shanne', 'Tobera', 'Uanne', 'Varie', 'Willie', 'Xanne', 'Yanne', 'Zanner'

对首字母从 A-Z 分别预测生成的男生名字为：

'Allen', 'Bambired', 'Caris', 'Dustyn', 'Elle', 'Fluncy', 'Gricell', 'Hobbestiberca', 'Iando', 'Jeris', 'Kermit', 'Lan', 'Mertin', 'Nerin', 'Odie', 'Pinercat', 'Qanno', 'Radog', 'Shardog', 'Toris', 'Uannoo', 'Varrin', 'Wano', 'Xondo', 'Yannoo', 'Zanne'

4.2 结果分析

模型能够根据不同的首字母生成不同的名字，这表明它学到了如何根据输入的首字母变化其输出。然而，一些名字（如'Elle'）在男女名字中都出现了，这可能表明模型对性别特征的学习不够区分。同时，某些名字（如'Hobbestiberca'，'Pinercat'）看起来在结构上不太像典型的名字，这可能表明模型在生成过程中过度创新或者没有很好地捕捉到人名的结构规律。

5. 附加题：从尾部和中间生成名字

题目要求：

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。

5.1 从尾部生成名字

思路：从尾部生成名字，与正向生成名字仅换了一个方向，故我想，只需在训练时将所有名字反过来进行训练，得到一个 rnn_reverse.pkl 的反向模型。在生成名字时，调用这个模型，进行生成，再对得到的名字进行翻转，就得到了反向生成的名字。

这需要重写 train 和 predict 函数：

train_reverse 函数的工作原理如下：

模型初始化：

创建一个 RNN 模型实例，并准备损失函数、优化器和其他训练参数。

训练循环：

对于每个训练周期（epoch），随机生成一个训练样本。但这里的样本是反向的，即从名字的末尾开始。

```
    category_tensor, input_tensor, target_tensor = random_training_example(-1)
          target_tensor.unsqueeze_(-1)
```

```
def random_training_example(mode=1):
    category = random_choice(all_categories)
    input = random_choice(category_lines[category])

    if mode == -1:
        input = input[::-1]  # 反转输入字符串
    elif mode == -2:
        return input

    category_tensor = category_to_tensor(category)
    input_tensor = input_to_tensor(input)
    target_tensor = target_to_tensor(input)
    return category_tensor, input_tensor, target_tensor
```

逐个字符地进行训练，计算输出与目标的损失，并通过反向传播更新模型权重。
记录和打印损失，以便监控训练进度。
模型保存：
在训练结束后，保存模型的权重。

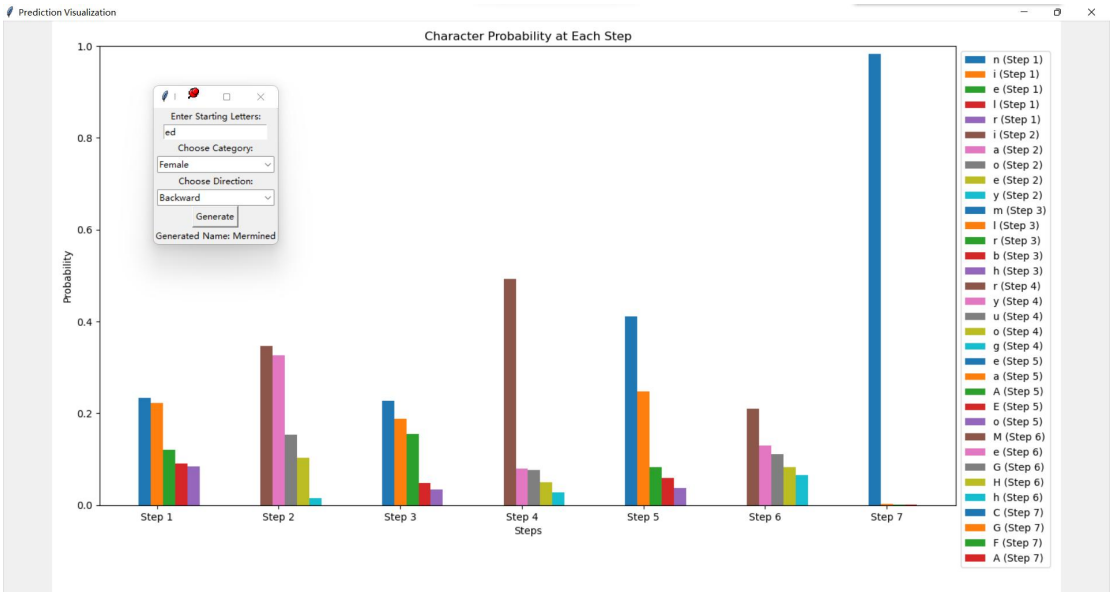
predict_reverse 函数的工作原理如下：
加载训练好的模型：
加载先前保存的反向 RNN 模型。
预测循环：
使用输入的尾部字母作为起始点，通过 RNN 模型逐步预测出名字的前一个字母。
每一步都记录下当前步骤的前五个最可能的字符及其概率（用于可视化或分析）。
如果生成了特殊的结束符或达到最大长度，则停止生成。
结果处理：
由于生成是从尾部开始的，所以最后需要将结果字符串反转，得到正常顺序的名字。

```
    def predict_reverse(category, end_letters):
        rnn = RNN(n_letters, n_hidden, n_letters).to(device)
        rnn.load_state_dict(torch.load('../model/rnn_params_reverse.pkl'))  # 加载反向训练的模型
        max_length = 20

        with torch.no_grad():
            category_tensor = category_to_tensor(category)
            input = input_to_tensor_reverse(end_letters)
            hidden = rnn.init_hidden()

            output_name = end_letters[::-1]
            top5_each_step = []  # 用于存储每步的前 5 个字符及其概率

            for i in range(max_length - len(end_letters)):
                output, hidden = rnn(category_tensor, input[0], hidden)
                topv, topi = output.topk(5)
                top5_each_step.append((topi, topv))

                topi = topi[0][0]
                if topi == n_letters - 1:
                    break
                else:
                    letter = all_letters[topi]
                    output_name += letter
                input = input_to_tensor_reverse(letter)

            return output_name[::-1], top5_each_step
```

生成效果：

输入：ed

生成：Mermined



对末尾字母从 A-Z 分别预测生成的女生名字为：

'CCeresta', 'Carlestab', 'Nieberic', 'Marmand', 'Marie', 'Marliaf', 'Arlaydog', 'Kermath', 'Delini', 'Cherij', 'Datyrick', 'Mariel', 'Kercateriam', 'MaTMarielin', 'Alaino', 'Stanoop', 'Cheriq', 'Duster', 'Cheris', 'Kercat', 'Odielaniu', 'Mariellev', 'Marielew', 'HoShenieeerix', 'Marieley', 'Marieliz'

对末尾字母从 A-Z 分别预测生成的男生名字为：

'Cherica', 'Ferlcatob', 'Theric', 'Merobid', 'Garchalle', 'Styberf', 'Kermitfrog', 'FKermath', 'Kercatari', 'Sharlej', 'Ranayduck', 'Dariel', 'Maram', 'Marielen', 'Rolando', 'Ranoop', 'Mariq', 'Bamber', 'Sheris', 'Kerust', 'Cerustanou', 'Sharlev', 'Marielew', 'Kescaterix', 'Sheley', 'Marlandoz'

5. 2 从中间向两侧生成名字

本来是想使用双向 RNN 对名字前后的特征都进行学习来进行生成，但由于时间原因没有做完，半成品模型可以在代码中的 BiRnn 类中找到。最终选择如果你有一个名字的中间部分，你可以独立地生成它的前缀（名字的开始部分）和后缀（名字的结束部分），然后将它们组合在一起形成完整的名字的方式进行实现。

这种方法通常需要两个模型或同一个模型的两种运行模式：

反向模型 (model_reverse)：这个模型负责生成名字的前缀。它是根据从尾部到头部的顺序训练的，所以它从中间部分开始生成，并向前工作，生成名字的开始部分。

```
def predict_prefix(model_reverse, category, middle_chars):
    max_length = 4
    with torch.no_grad():
        category_tensor = category_to_tensor(category)
        input = input_to_tensor_reverse(middle_chars)  # 注意：使用了反转的输入
        hidden = model_reverse.init_hidden()

        output_prefix = ''
        for i in range(max_length):
```

```
            output, hidden = model_reverse(category_tensor, input[0], hidden)
            topi = output.topk(1)[1].item()
            if topi == n_letters - 1:  # or i == len(middle_chars) - 1:  # EOS 或达到中间字符长度
                break
            else:
                letter = all_letters[topi]
                output_prefix += letter
            input = input_to_tensor_reverse(letter)

        return output_prefix[::-1]  # 反转前缀以恢复正确的顺序
```

前向模型 (model)：这个模型负责生成名字的后缀。它是按照从头部到尾部的顺序训练的，从中间部分开始生成，并向后工作，生成名字的剩余部分。

```
def predict_suffix(model, category, middle_chars):
    max_length = 4
    with torch.no_grad():
        category_tensor = category_to_tensor(category)
        input = input_to_tensor(middle_chars)
        hidden = model.init_hidden()

        output_suffix = middle_chars
        for i in range(max_length):
            output, hidden = model(category_tensor, input[0], hidden)
            topi = output.topk(1)[1].item()
            if topi == n_letters - 1:  # EOS
                break
            else:
                letter = all_letters[topi]
                output_suffix += letter
            input = input_to_tensor(output_suffix)

        return output_suffix[len(middle_chars):]  # 只返回后缀部分
```

在 predict_full_name 函数中，以下步骤被执行：

使用反向模型生成前缀。这通过将中间字符反转并馈送给反向模型来完成，模型一步一步地预测前一个字符，直到达到最大长度或生成结束符为止。生成的前缀在返回前被反转回正常顺序。

使用前向模型生成后缀。这通过直接将中间字符馈送给前向模型来完成，模型一步一步地预测下一个字符，直到达到最大长度或生成结束符为止。

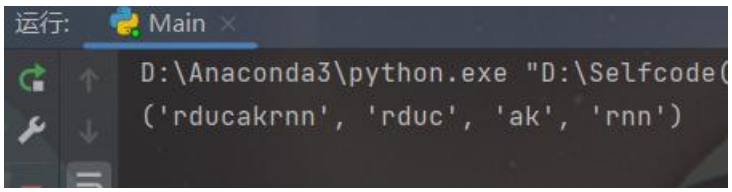
将生成的前缀、中间部分和后缀组合在一起，形成完整的名字。

```
def predict_full_name(category, middle_chars):
    model = RNN(n_letters, n_hidden, n_letters).to(device)
    model.load_state_dict(torch.load('../model/rnn_params.pkl'))  # 加载前向模型
    model_reverse = RNN(n_letters, n_hidden, n_letters).to(device)
    model_reverse.load_state_dict(torch.load('../model/rnn_params_reverse.pkl'))  # 加载反向模型
```

```
6.
7.         prefix = predict_prefix(model_reverse, category, middle_chars)
8.         suffix = predict_suffix(model, category, middle_chars)
9.
10.        return prefix + middle_chars + suffix, prefix, middle_chars, suffix
```

这种生成名字的方法特别适用于情境中你想要围绕某个特定的中间音节或字符组合构建名字。例如，在某些文化中，名字的中间部分可能有特定的含义或声韵，而前后缀则可以更自由地变化。通过这种方法，模型能够围绕这个固定的中间部分创造性地生成符合语言习惯的名字。

生成效果：
输入：ak
生成：Rducakrnn



6. 附加题：可视化技术绘制名字生成过程

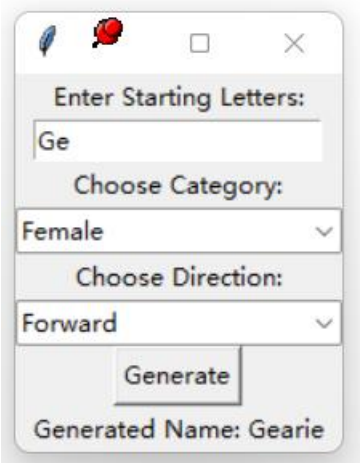
题目：从模型生成的名字中，挑选你最喜欢的一个，并采用可视化技术，绘制生成这个名字的过程。

6.1 解决方案概述

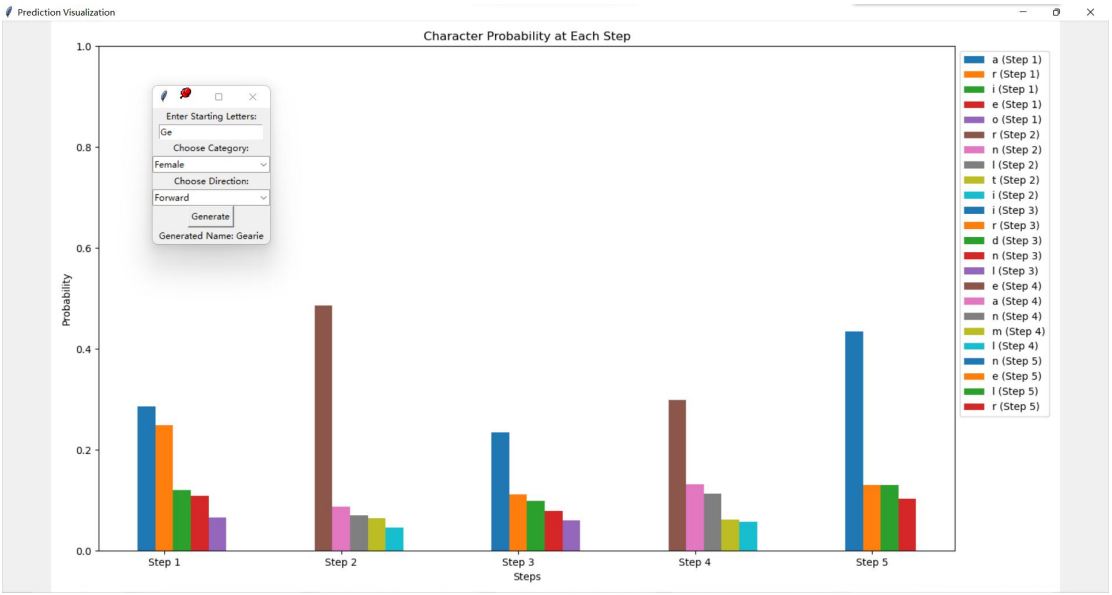
使用 Tkinter 库制作了一个前端界面，再界面中，用户可以输入字符，选择生成名字类别，并选择从前向，后向，还是双向进行生成名字。而后绘制一幅生成名字的图，展示生成过程中每一次生成所选取字符的原因，即前五概率的字符。

6.2 效果展示

1. 输入窗口：



2. 生成效果窗口：


Prediction Visualization
Character Probability at Each Step
Enter Starting Letters:
Ge
Choose Category:
Female
Choose Direction:
Forward
Generate
Generated Name: Gearie
Probability
1.0
0.8
0.6
0.4
0.2
0.0
Step 1
Step 2
Step 3
Step 4
Step 5
Steps
a (Step 1)
r (Step 1)
i (Step 1)
e (Step 1)
o (Step 1)
r (Step 2)
n (Step 2)
l (Step 2)
t (Step 2)
i (Step 2)
i (Step 3)
r (Step 3)
d (Step 3)
n (Step 3)
l (Step 3)
e (Step 4)
a (Step 4)
n (Step 4)
m (Step 4)
l (Step 4)
n (Step 5)
e (Step 5)
l (Step 5)
r (Step 5)