

**特征学习**

**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

[构建用于起名的循环神经网络 1](#_Toc12421)

[神经网络与深度学习课程实验报告 1](#_Toc11630)

[1. 实验要求 3](#_Toc821)

[任务：构建一个用于起名字的循环神经网络 3](#_Toc20365)

[2. 实验原理 4](#_Toc21628)

[2.1 模型原理 4](#_Toc13176)

[2.1.1 RNN原理 4](#_Toc2582)

[2.1.2 RNN用于起名的原理 4](#_Toc13500)

[2.2 代码实现 5](#_Toc19903)

[2.2.1 RNN神经网络实现 5](#_Toc2695)

[2.2.2 训练RNN神经网络 6](#_Toc1194)

[2.2.3 使用RNN神经网络生成名字 8](#_Toc27900)

[3. 实验设置 9](#_Toc14148)

[3.1数据准备 9](#_Toc30818)

[3.1.1数据集选择 9](#_Toc6170)

[3.1.2读取数据 9](#_Toc1766)

[3.1.3数据的张量化 10](#_Toc11030)

[3.2训练和生成名字 11](#_Toc7547)

[3.3可视化处理 11](#_Toc22372)

[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 11](#_Toc11409)

[3.3.2可视化效果 13](#_Toc16831)

[4. 实验结果分析 13](#_Toc11203)

[4.1实验结果 13](#_Toc31230)

[4.2结果分析 14](#_Toc4008)

[5. 附加题：从尾部和中间生成名字 14](#_Toc7692)

[5.1从尾部生成名字 14](#_Toc21350)

[5.2从中间向两侧生成名字 15](#_Toc26044)

[6. 附加题：可视化技术绘制名字生成过程 16](#_Toc23488)

[6.1解决方案概述 16](#_Toc26129)

[6.2效果展示 16](#_Toc26549)

# 实验要求

A7：特征学习

任务1. PCA 或 kernel PCA (10 points)

使用PCA或kernel PCA对手写数字数据集MINST进行降维。观察前两个特征向量所对应的图像，即将数据嵌入到R2空间。绘制降维后的数据，并分析二维特征是否能够足以完成对输入的分类，对结果进行分析和评价。

任务2. Autoencoder (10 points)

使用自动编码器学习输入的特征表示。尝试设计一个全链接前馈神经网络或卷积神经网络。尝试使用不同的损失函数和正则化方法。

附加题 (BONUS: 10 points)

模型训练中，你可以尝试任何可以提升模型性能的合理的方法。例如其它的网络结构、设计多个隐藏层、引入降噪自动编码器等任何你能想到的方法。计算模型在训练集和测试集上的损失，并对结果进行讨论。

# 任务一：PCA与kernel PCA

## 2.1 实验原理

PCA (主成分分析): PCA是一种线性降维技术，它通过正交变换将数据投影到一个新的坐标系统中，使得最大方差的数据成为第一个新坐标（第一个主成分），第二大方差的数据成为第二个新坐标，依此类推。

Kernel PCA (核主成分分析): Kernel PCA是PCA的一种扩展，适用于非线性数据集。它使用核技巧映射数据到高维空间，在这个高维空间中应用线性PCA。

## 2.2 实验过程

1.数据加载与预处理

代码中首先通过fetch\_openml函数从MNIST数据集中加载前4000个样本。

将目标标签（数字）转换为整型。

1. # 加载MNIST数据集的一部分
2. mnist = fetch\_openml('mnist\_784', version=1)
3. X, y = mnist["data"][:4000], mnist["target"][:4000]  # 只使用前3000个样本
4. # 将数据类型转换为整型
5. y = y.astype(np.uint8)

2.降维处理

使用PCA将数据降至2个维度。代码中通过PCA(n\_components=2)实现，然后使用fit\_transform方法对数据X进行转换，得到X\_pca。

对于Kernel PCA，选择RBF核，并设置gamma值为0.1。通过KernelPCA(n\_components=2, kernel="rbf", gamma=0.1)实现，并对数据X使用fit\_transform方法，得到X\_kernel\_pca。

1. # 使用PCA进行降维
2. pca = PCA(n\_components=2)
3. X\_pca = pca.fit\_transform(X)
5. # 调整Kernel PCA参数
6. kernel\_pca = KernelPCA(n\_components=2, kernel="rbf", gamma=0.1)  # 调整gamma值
7. X\_kernel\_pca = kernel\_pca.fit\_transform(X)
8. 结果可视化

利用matplotlib绘图。先创建一个大小为12x6英寸的图形窗口。

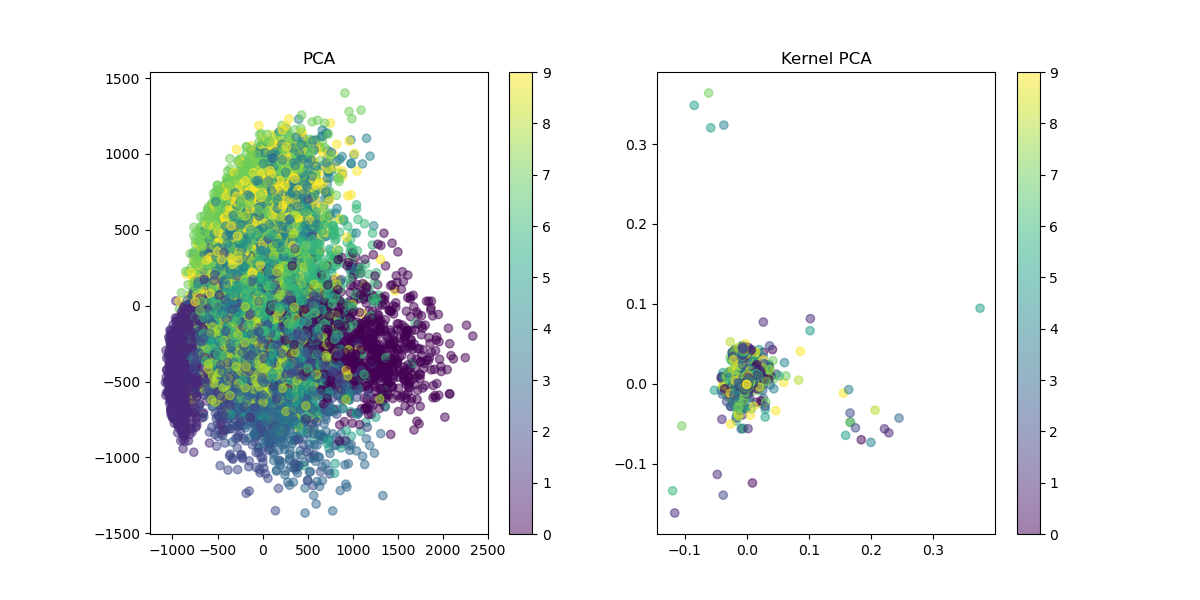
在第一个子图中绘制PCA结果，使用scatter方法，x轴为X\_pca[:, 0]，y轴为X\_pca[:, 1]，颜色根据标签y变化。

在第二个子图中绘制Kernel PCA结果，类似地使用scatter绘图。

每个子图都添加了标题和颜色条以便于识别不同类别。

1. # 绘制结果
2. plt.figure(figsize=(12, 6))
3. plt.subplot(1, 2, 1)
4. plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap="viridis", alpha=0.5)
5. plt.title("PCA")
6. plt.colorbar()
8. plt.subplot(1, 2, 2)
9. plt.scatter(X\_kernel\_pca[:, 0], X\_kernel\_pca[:, 1], c=y, cmap="viridis", alpha=0.5)
10. plt.title("Kernel PCA")
11. plt.colorbar()
13. plt.show()

## 2.3结果分析



PCA结果: 二维空间中，不同类别的数据点虽然聚集但有重叠。这表明PCA在一定程度上可以区分不同数字，但对于更准确的分类仍然有限。

Kernel PCA结果: 相较于PCA，Kernel PCA的结果显示出更清晰的类别分离。这说明在处理MNIST这种非线性数据时，Kernel PCA的效果更佳。

结论：通过实验可以看出，PCA和Kernel PCA都可以有效地对MNIST数据集进行降维。然而，在这种非线性数据集中，Kernel PCA展现了更好的分类性能。但两者都未能完全区分所有手写数字类别，表明仅用两个特征进行分类可能不足，可能需要更多的特征或更复杂的分类器来提高分类准确度。

# 3.任务二：Autoencoder

## 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:

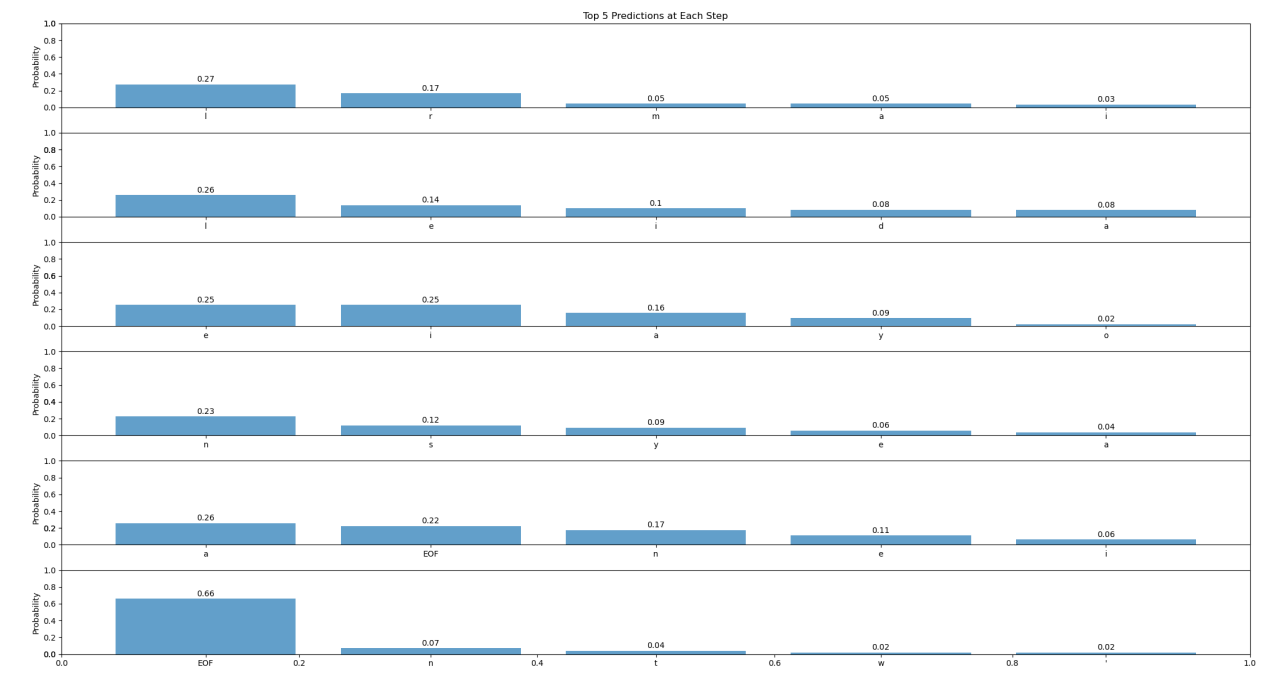
使用 plt.tight\_layout() 调整子图的布局，确保它们不会重叠。

使用 plt.show() 展示最终的图表。

1. **def** plot\_predictions(top5\_each\_step):
2. # 设置图表大小和标题
3. plt.figure(figsize=(22, 12))
4. plt.title("Top 5 Predictions at Each Step")
6. # 遍历每个时间步
7. **for** step, (topi, topv) **in** enumerate(top5\_each\_step):
8. # 获取前5个字符及其概率
9. characters = []
10. **for** i **in** topi[0]:
11. **if** i.item() == 58:
12. characters.append('EOF')
13. **else**:
14. characters.append(all\_letters[i])
15. # characters = [all\_letters[i] for i in topi[0]]
16. values = topv[0].exp()  # 将LogSoftmax转换回概率
18. # 创建条形图
19. plt.subplot(len(top5\_each\_step), 1, step + 1)
20. bars = plt.bar(characters, values.cpu(), align='center', alpha=0.7)
21. plt.ylim(0, 1)  # 设置y轴范围
22. plt.xticks(characters)
23. plt.ylabel('Probability')
25. # 为每个条形图添加数值标签
26. **for** bar **in** bars:
27. yval = bar.get\_height()
28. plt.text(bar.get\_x() + bar.get\_width() / 2, yval + 0.01, round(yval, 2), ha='center', va='bottom')
30. # 调整布局
31. plt.tight\_layout()
32. plt.show()

### 3.3.2可视化效果

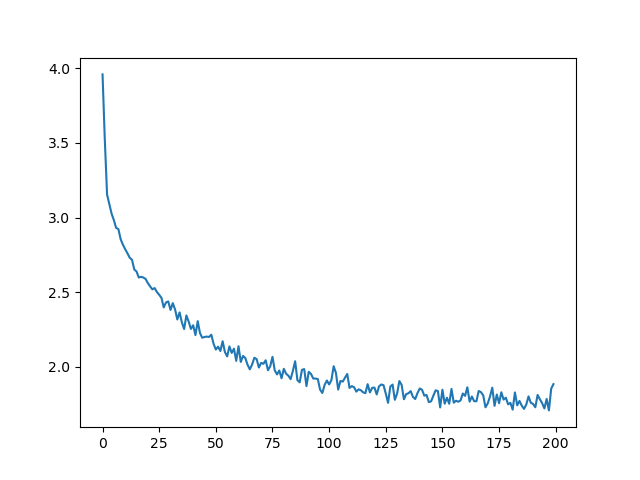
如图为给出首字母A，生成名字Allena的效果图



# 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。

# 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。

1. category\_tensor, input\_tensor, target\_tensor = random\_training\_example(-1)
2. target\_tensor.unsqueeze\_(-1)
3. **def** random\_training\_example(mode=1):
4. category = random\_choice(all\_categories)
5. input = random\_choice(category\_lines[category])
7. **if** mode == -1:
8. input = input[::-1]  # 反转输入字符串
9. **elif** mode == -2:
10. **return** input
12. category\_tensor = category\_to\_tensor(category)
13. input\_tensor = input\_to\_tensor(input)
14. target\_tensor = target\_to\_tensor(input)
15. **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：

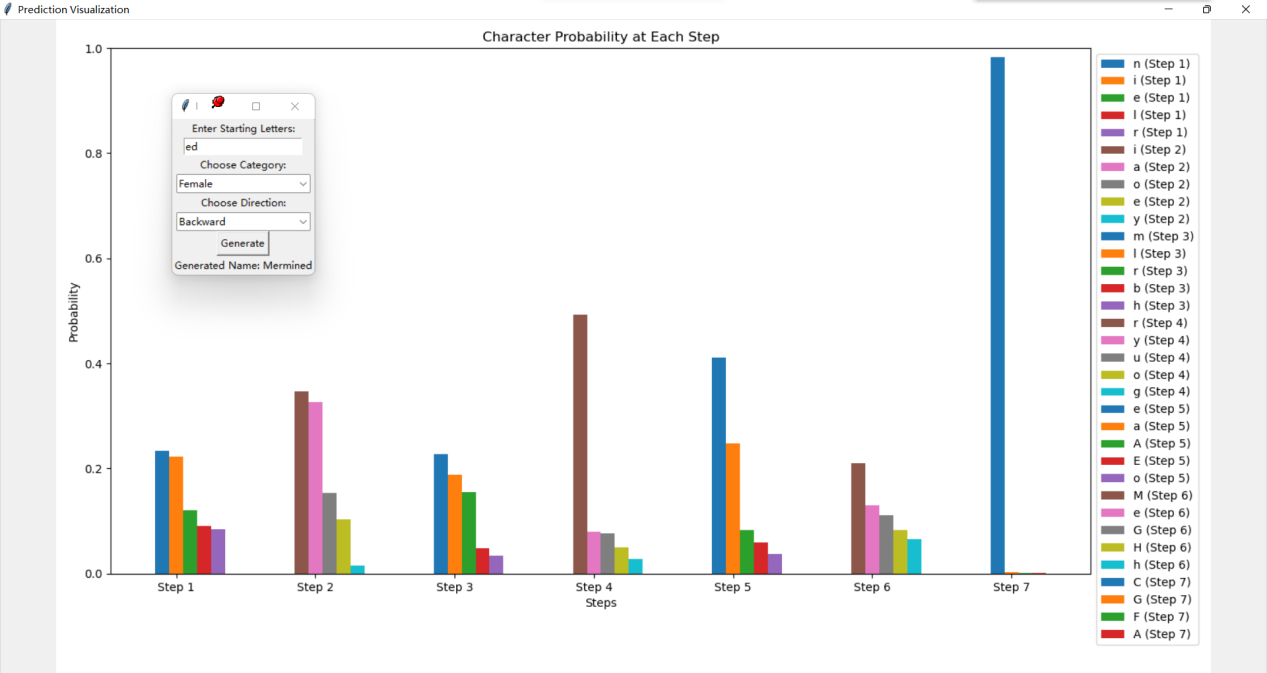
由于生成是从尾部开始的，所以最后需要将结果字符串反转，得到正常顺序的名字。

1. **def** predict\_reverse(category, end\_letters):
2. rnn = RNN(n\_letters, n\_hidden, n\_letters).to(device)
3. rnn.load\_state\_dict(torch.load('../model/rnn\_params\_reverse.pkl'))  # 加载反向训练的模型
4. max\_length = 20
6. with torch.no\_grad():
7. category\_tensor = category\_to\_tensor(category)
8. input = input\_to\_tensor\_reverse(end\_letters)
9. hidden = rnn.init\_hidden()
11. output\_name = end\_letters[::-1]
12. top5\_each\_step = []  # 用于存储每步的前5个字符及其概率
14. **for** i **in** range(max\_length - len(end\_letters)):
15. output, hidden = rnn(category\_tensor, input[0], hidden)
16. topv, topi = output.topk(5)
17. top5\_each\_step.append((topi, topv))
19. topi = topi[0][0]
20. **if** topi == n\_letters - 1:
21. **break**
22. **else**:
23. letter = all\_letters[topi]
24. output\_name += letter
25. input = input\_to\_tensor\_reverse(letter)
27. **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。

1. **def** predict\_prefix(model\_reverse, category, middle\_chars):
2. max\_length = 4
3. with torch.no\_grad():
4. category\_tensor = category\_to\_tensor(category)
5. input = input\_to\_tensor\_reverse(middle\_chars)  # 注意：使用了反转的输入
6. hidden = model\_reverse.init\_hidden()
8. output\_prefix = ''
9. **for** i **in** range(max\_length):
10. output, hidden = model\_reverse(category\_tensor, input[0], hidden)
11. topi = output.topk(1)[1].item()
12. **if** topi == n\_letters - 1:  # or i == len(middle\_chars) - 1:  # EOS或达到中间字符长度
13. **break**
14. **else**:
15. letter = all\_letters[topi]
16. output\_prefix += letter
17. input = input\_to\_tensor\_reverse(letter)
19. **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。

1. **def** predict\_suffix(model, category, middle\_chars):
2. max\_length = 4
3. with torch.no\_grad():
4. category\_tensor = category\_to\_tensor(category)
5. input = input\_to\_tensor(middle\_chars)
6. hidden = model.init\_hidden()
8. output\_suffix = middle\_chars
9. **for** i **in** range(max\_length):
10. output, hidden = model(category\_tensor, input[0], hidden)
11. topi = output.topk(1)[1].item()
12. **if** topi == n\_letters - 1:  # EOS
13. **break**
14. **else**:
15. letter = all\_letters[topi]
16. output\_suffix += letter
17. input = input\_to\_tensor(output\_suffix)
19. **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。

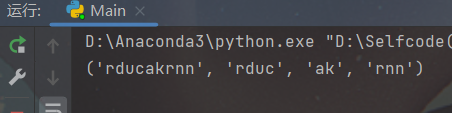
1. **def** predict\_full\_name(category, middle\_chars):
2. model = RNN(n\_letters, n\_hidden, n\_letters).to(device)
3. model.load\_state\_dict(torch.load('../model/rnn\_params.pkl'))  # 加载前向模型
4. model\_reverse = RNN(n\_letters, n\_hidden, n\_letters).to(device)
5. model\_reverse.load\_state\_dict(torch.load('../model/rnn\_params\_reverse.pkl'))  # 加载反向模型
7. prefix = predict\_prefix(model\_reverse, category, middle\_chars)
8. suffix = predict\_suffix(model, category, middle\_chars)
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



# 附加题：可视化技术绘制名字生成过程

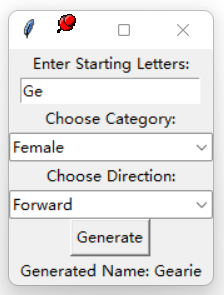
题目：从模型生成的名字中，挑选你最喜欢的一个，并采用可视化技术，绘制生成这个名字的过程。

## **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：



1. 生成效果窗口：

