## 循环神经网络实验报告

姓名：张铭徐 学号：2113615

实验要求：

* 掌握RNN原理
* 学会使用PyTorch搭建循环神经网络来训练名字识别
* 学会使用PyTorch搭建LSTM网络来训练名字识别

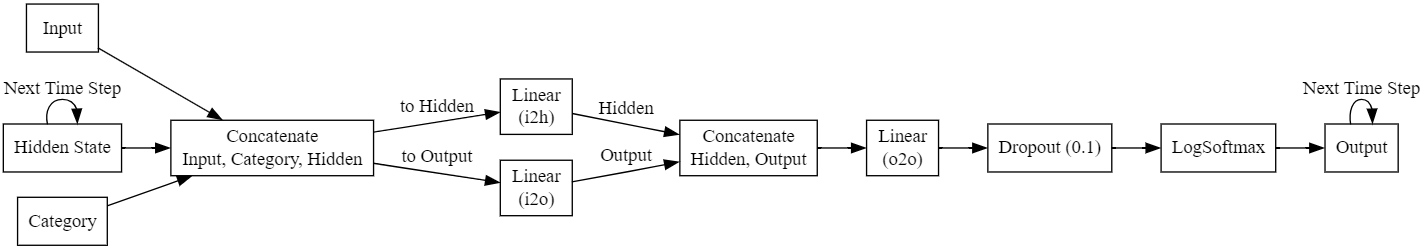
报告内容：

* 老师提供的原始版本RNN网络结构（可用print(net)打印，复制文字或截图皆可）、在名字识别验证集上的训练loss曲线、准确度曲线图以及预测矩阵图
* 个人实现的LSTM网络结构在上述验证集上的训练loss曲线、准确度曲线图以及预测矩阵图
* 解释为什么LSTM网络的性能优于RNN网络（重点部分）
* 格式不限

与前几次实验一样，我们并没有使用老师提供的原始版本的LSTM和RNN，而是自行实现了若干版本。

RNN主要用于处理序列数据，其与原本的深度神经网络相比，增加了循环的结构，原本的网络是前馈的，但是RNN则是循环的，其主要结构包括一个循环单元，这个单元在处理序列的每个时间步时接收两个输入：当前时间步的输入数据（例如，在文本中的当前字或词）以及上一个时间步的隐藏状态（代表了至该时间点为止的序列信息）。基于这两个输入，循环单元更新其隐藏状态，并可产生一个输出。这个输出可以直接用作某些任务的结果，如分类或回归，或者仅用作下一个时间步的输入部分。关键点在于，隐藏状态包含了之前时间步的信息，因此网络能够“记住”并利用过去的信息。

首先我们给出我们实现RNN的网络结构图：



我们的代码如下：

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(output\_size + input\_size + hidden\_size, hidden\_size)

        self.i2o = nn.Linear(output\_size + 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)

    def forward(self, category, input, hidden):

        input\_combined = torch.cat((category, input, hidden), dim=1)

        hidden = self.i2h(input\_combined)

        output = self.i2o(input\_combined)

        output\_combined = torch.cat((hidden, output), dim=1)

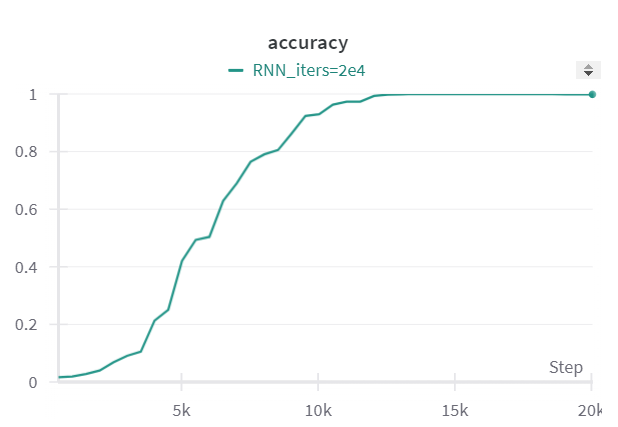
        output = self.o2o(output\_combined)

        output = self.dropout(output)

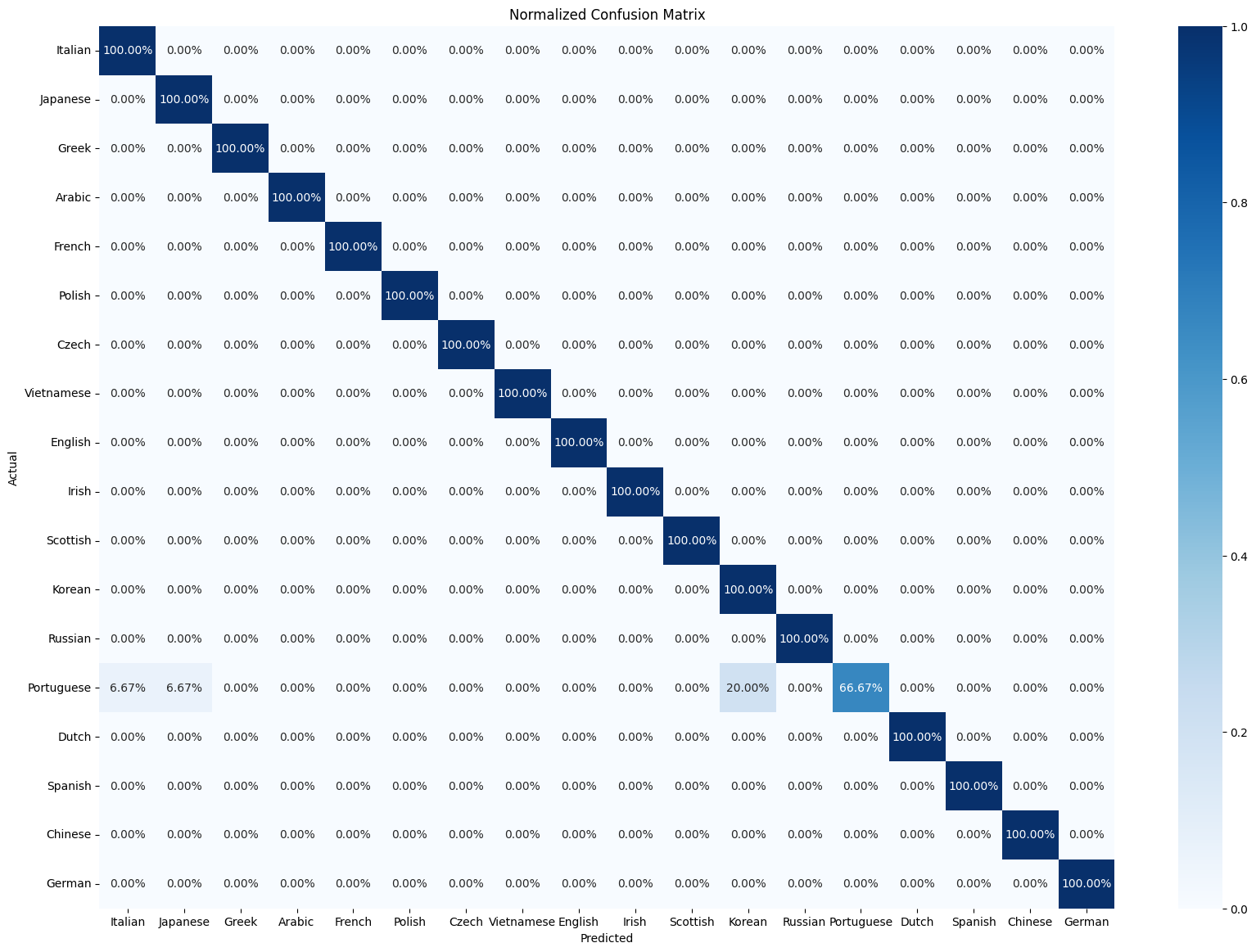
        output = self.softmax(output)

        return output, hidden

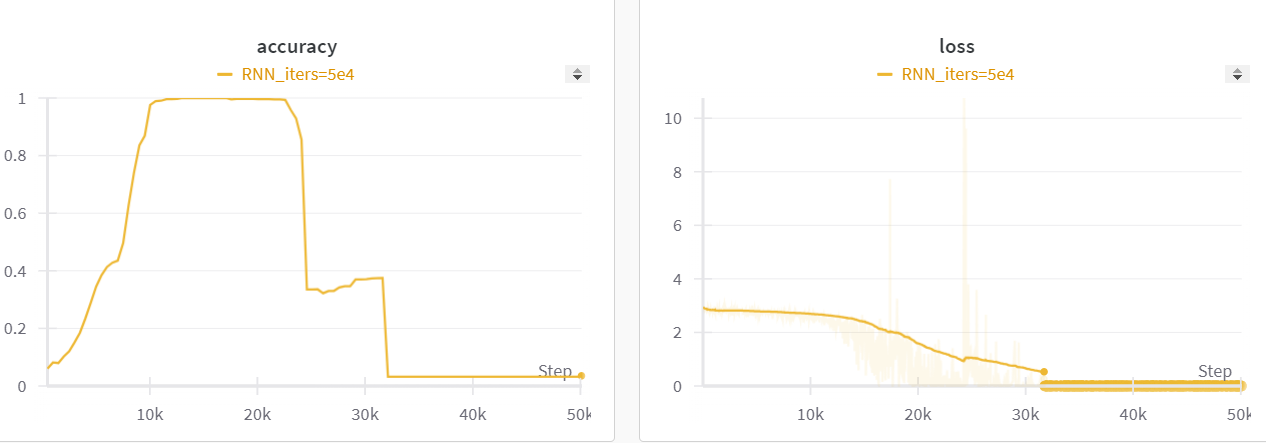
在实验中，我们进行了多次实验，分别迭代2e5次，2e4次，1e4次，取准确率最高效果最好的作为实验结论，即迭代2e4次，其准确率和训练loss如下图所示：



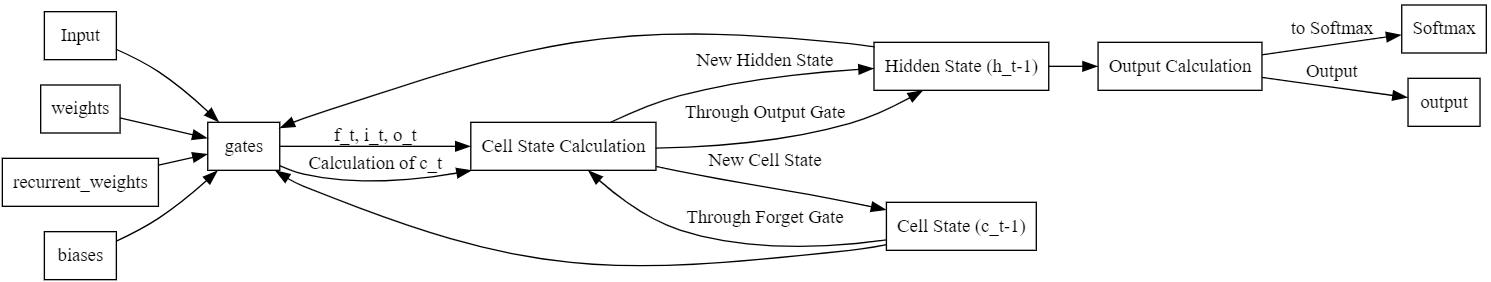
我们需要做类似于分类问题，所以我们可以绘制出混淆矩阵，如下所示：



最终在RNN中，我们获得了99.8%的准确率，迭代次数为2e4次。我们可以发现，在迭代次数过多的情况下，很容易出现梯度消失和梯度爆炸的现象，如下图所示：



在上面图中，我们设定迭代次数为5e4，随着训练的进行，我们可以看到在约3e4时出现了问题，loss变为了NaN，也就是出现了梯度消失的现象，随之而来的是整体准确率极度下跌。这是由于在反向传播过程中，梯度必须通过每个时间步传递，梯度可能会随着时间的推移而变得非常小（消失）或非常大（爆炸）。所以其中一个解决方案就是实现LSTM，在本次实验中，我们也自行实现了LSTM结构：



相比于RNN的结构，LSTM增加了输入门、遗忘门和输出门，这些门控制信息是被记住还是遗忘，以及将什么信息传递到隐藏状态。这样我们控制过去的信息是否往下继续传递，就避免了某些信息经过较长时间的稀释之下梯度消失的问题。我们实现的代码如下：

class LSTM(nn.Module):

    def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

        super(LSTM, self).\_\_init\_\_()

        self.hidden\_size = hidden\_size

        # 直接实现 LSTM 而不调用 nn.LSTM

        self.input\_size = input\_size

        self.hidden\_size = hidden\_size

        self.output\_size = output\_size

        # Gates' weights and biases

        self.w\_i = nn.Parameter(torch.Tensor(input\_size, hidden\_size))

        self.w\_f = nn.Parameter(torch.Tensor(input\_size, hidden\_size))

        self.w\_c = nn.Parameter(torch.Tensor(input\_size, hidden\_size))

        self.w\_o = nn.Parameter(torch.Tensor(input\_size, hidden\_size))

        self.u\_i = nn.Parameter(torch.Tensor(hidden\_size, hidden\_size))

        self.u\_f = nn.Parameter(torch.Tensor(hidden\_size, hidden\_size))

        self.u\_c = nn.Parameter(torch.Tensor(hidden\_size, hidden\_size))

        self.u\_o = nn.Parameter(torch.Tensor(hidden\_size, hidden\_size))

        self.b\_i = nn.Parameter(torch.Tensor(hidden\_size))

        self.b\_f = nn.Parameter(torch.Tensor(hidden\_size))

        self.b\_c = nn.Parameter(torch.Tensor(hidden\_size))

        self.b\_o = nn.Parameter(torch.Tensor(hidden\_size))

        self.hidden2out = nn.Linear(hidden\_size, output\_size)

        self.softmax = nn.LogSoftmax(dim=1)  # Ensure that 'dim=1' is correct

    def forward(self, input, hidden):

        # Assume 'input' is (seq\_len, batch, input\_dim)

        hx, cx = hidden

        for i in range(input.size(0)):  # Process each time step

            x = input[i]

            i\_t = torch.sigmoid(x @ self.w\_i + hx @ self.u\_i + self.b\_i)

            f\_t = torch.sigmoid(x @ self.w\_f + hx @ self.u\_f + self.b\_f)

            c\_t = f\_t \* cx + i\_t \* torch.tanh(x @ self.w\_c + hx @ self.u\_c + self.b\_c)

            o\_t = torch.sigmoid(x @ self.w\_o + hx @ self.u\_o + self.b\_o)

            hx = o\_t \* torch.tanh(c\_t)

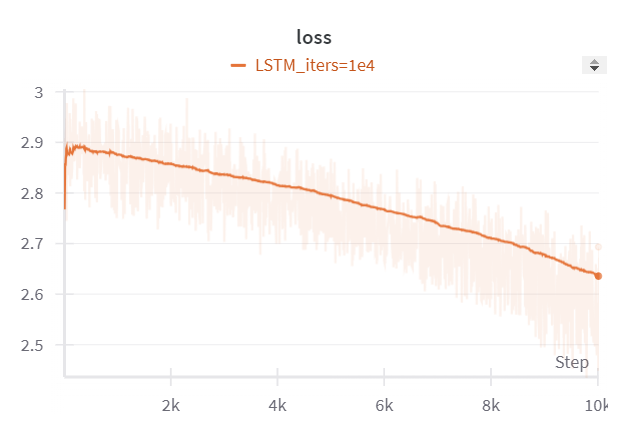
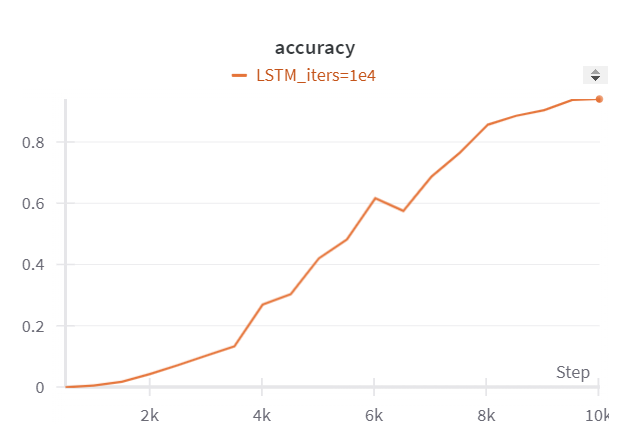
            cx = c\_t

        output = self.hidden2out(hx)

        output = self.softmax(output)

        return output, (hx, cx)

同样的，我们进行训练，我们选取迭代次数为1e4进行实验，训练曲线如下所示：



我们可以看到在迭代次数为1e4之后，模型准确率达到了最优，准确率为100%，与RNN相比，在同样迭代次数下，性能显著性的得到了提升，由于模型准确率为100%，所以我们无需借助混淆矩阵进行分析。

接下来我们考虑为什么LSTM性能优于RNN，首先第一点，正如前面分析的那样，对于长期依赖的数据而言，LSTM设计了一个细胞状态，这是一条贯穿整个链的信息“高速公路”，只有少数线性交互，梯度可以较为完整地在其中流动。通过门控调节哪些信息加入细胞状态，细胞状态可以跨越长时间序列携带信息，使LSTM能够有效学习并利用长期依赖信息；而朴素的RNN由于梯度消失的问题，很难保持对早期输入的记忆，这限制了其处理长序列数据的能力。此外，LSTM在梯度消失和爆炸的问题上也得到了解决，这样我们可以迭代更多的次数增强其能力；最后，LSTM的收敛性也会更优，门控机制提供了更多控制信息流的方法，这提高了模型对输入数据中噪声的鲁棒性，增强的鲁棒性和稳定性进一步改善了模型的训练效率和收敛速度。