南開大學
Nankai University

计算机学院

深度学习实验报告

循环神经网络

姓名：杨馨仪

学号：2011440

专业：计算机科学与技术

2024 年 6 月 23 日

目录

1 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

2 原始 RNN 网络

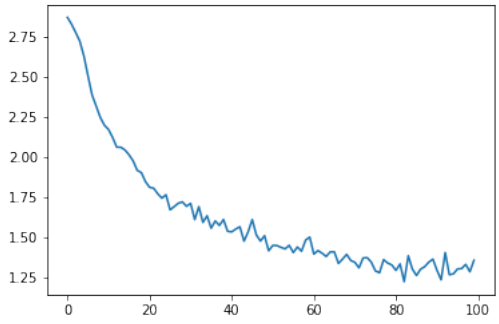
通过打印网络，可知原始网络结构如下图2.1所示。

```
RNN(
  (i2h): Linear(in_features=185, out_features=128, bias=True)
  (i2o): Linear(in_features=185, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

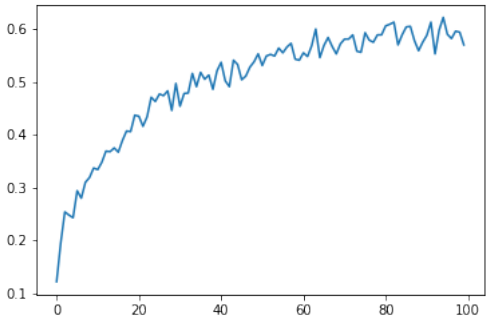
图 2.1: 原始 RNN 网络结构

该网络有两个线性层，以及一个 softmax 层。网络会根据当前输入与历史隐层态，将两者拼接到一起，去计算新的隐层态以及输出。输出会接着经过 softmax 层进行归一化操作，最后得到每个 category 的概率预测。

通过名字识别任务进行训练与测试，训练 loss 曲线如图 2.2(a)所示，RNN 的准确度曲线如图 2.2(b)所示，RNN 预测矩阵图如2.3所示。



(a) 原始 RNN 的损失曲线



(b) 原始 RNN 的准确度曲线

图 2.2: 原始 RNN 的损失曲线与准确度曲线

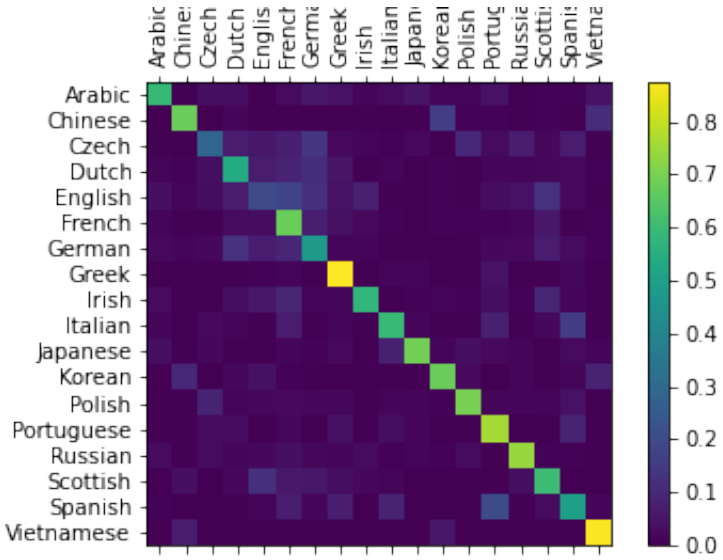


图 2.3: 原始 RNN 预测矩阵图

RNN 的预测准确度可以达到 60% 左右。在一些语言如 Czech，English，German，Spanish 上效果明显较差。

3 调用 Pytorch 内置的 LSTM 实现网络

在这一部分，通过调用 Pytorch 内置 nn.LSTM，并加入线性层和 softmax 层简单的构建了一个 LSTM 网络。打印网络结构如下：

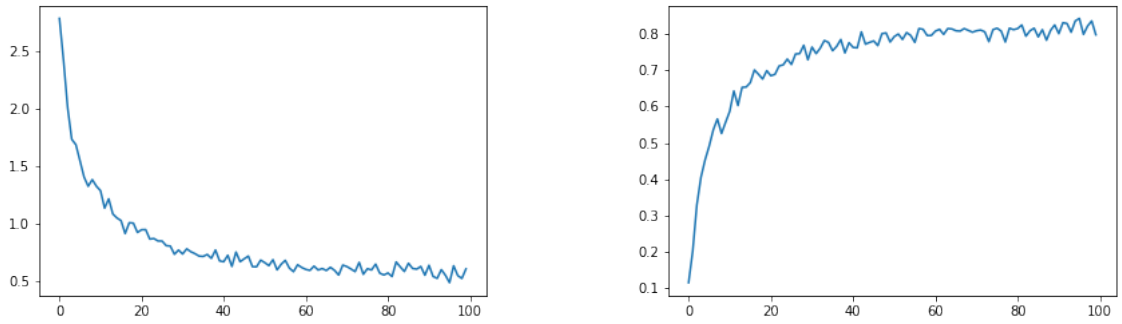
```
LSTM(
  (rnn): LSTM(57, 128, num_layers=2)
  (out): Linear(in_features=128, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=-1)
)
```

图 3.4: 调用 Pytorch 内置的 LSTM 实现的网络结构

该网络包含一个两层的 LSTM 模型和一个全连接层，最后通过 softmax 层将输出转化为各个 category 的概率值：

- LSTM 模型输入维度为 57，输出维度为 128，有两层。
- 全连接层输入维度为 128，输出维度为 18，对应 18 个语言类别。
- softmax 层将输出转化为各个 category 的概率值。

通过名字识别任务进行训练与测试，训练 loss 曲线如图 3.5(a)所示，该 LSTM 网络的准确度曲线如图 3.5(b)所示，该 LSTM 网络的预测矩阵图如3.6所示。



(a) LSTM 的损失曲线 (b) LSTM 的准确度曲线

图 3.5: LSTM 的损失曲线与准确度曲线

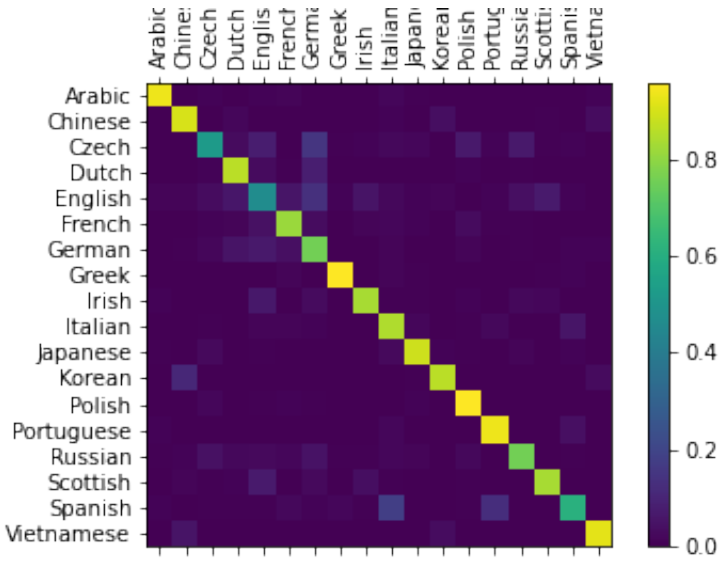


图 3.6: LSTM 预测矩阵图

该 LSTM 模型的预测准确度可以达到 80% 左右，效果相比于 RNN 有明显提升。

4 自己实现的 LSTM 网络

在这一部分，自己通过利用线性层，激活函数层去构建 LSTM 的遗忘门，输入门，细胞更新，与输出门，并最终实现网络结构，代码如下：

借助 Pytorch 内置 LSTM 接口实现的网络

```
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1):
        super(LSTM, self).__init__()

        self.hidden_size = hidden_size
```

```
        self.num_layers = num_layers

        self.forget_gate = nn.Linear(input_size+hidden_size, hidden_size)
        self.input_gate = nn.Linear(input_size+hidden_size, hidden_size)
        self.cell_update = nn.Linear(input_size+hidden_size, hidden_size)
        self.output_gate = nn.Linear(input_size+hidden_size, hidden_size)

        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=-1)

    def forward(self, input):
        if input.dim() == 2 and input.shape[0] == 1:
            input = input.unsqueeze(1)
        # combined = torch.cat((input, hidden), 1)
        h_t = torch.zeros(self.num_layers, input.shape[1], self.hidden_size,
            requires_grad=False)
        c_t = torch.zeros(self.num_layers, input.shape[1], self.hidden_size,
            requires_grad=False)
        for i in range(input.shape[0]):
            state_t = torch.concat((input[i], h_t[-1]), dim=-1)
            i_t = torch.sigmoid(self.forget_gate(state_t))
            f_t = torch.sigmoid(self.input_gate(state_t))
            g_t = torch.tanh(self.cell_update(state_t))
            o_t = torch.sigmoid(self.output_gate(state_t))

            c_t = f_t * c_t + i_t * g_t
            h_t = o_t * torch.tanh(c_t)

        output = self.out(h_t[-1])
        output = self.softmax(output)
        return output

n_hidden = 128
rnn = LSTM(n_letters, n_hidden, n_categories, 2)
print(rnn)
```

打印网络结构如下：

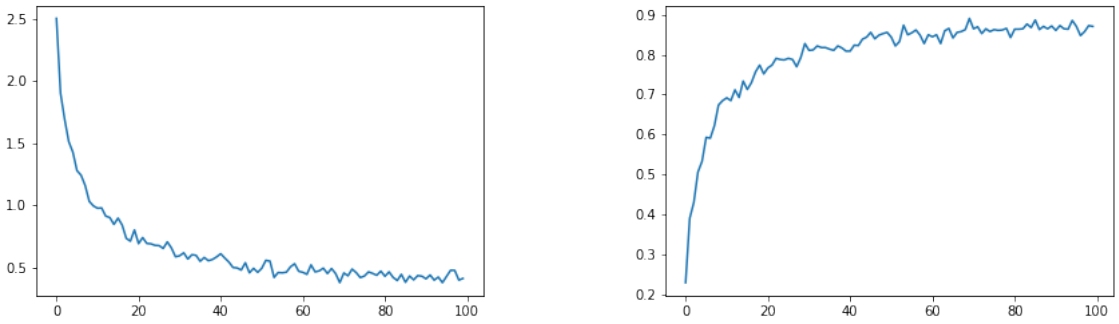
```
LSTM(
  (forget_gate): Linear(in_features=185, out_features=128, bias=True)
  (input_gate): Linear(in_features=185, out_features=128, bias=True)
  (cell_update): Linear(in_features=185, out_features=128, bias=True)
  (output_gate): Linear(in_features=185, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=-1)
)
```

图 4.7: 自己实现的 LSTM 网络结构

该网络包含一个两层的 LSTM 模型和一个全连接层，最后通过 softmax 层将输出转化为各个 category 的概率值：

- LSTM 模型输入维度为 57，输出维度为 128，有两层。
- 对于遗忘门，输入门，细胞更新，输出门，其输入维度都为 185 (57+128)，输出维度都为 128。
- 全连接层输入维度为 128，输出维度为 18，对应 18 个语言类别。
- softmax 层将输出转化为各个 category 的概率值。

通过名字识别任务进行训练与测试，训练 loss 曲线如图 4.8(a)所示，该 LSTM 网络的准确度曲线如图 4.8(a)所示，该 LSTM 网络的预测矩阵图如4.9所示。



(a) 自己实现的 LSTM 的损失曲线 (b) 自己实现的 LSTM 的准确度曲线

图 4.8: 自己实现的 LSTM 的损失曲线与准确度曲线

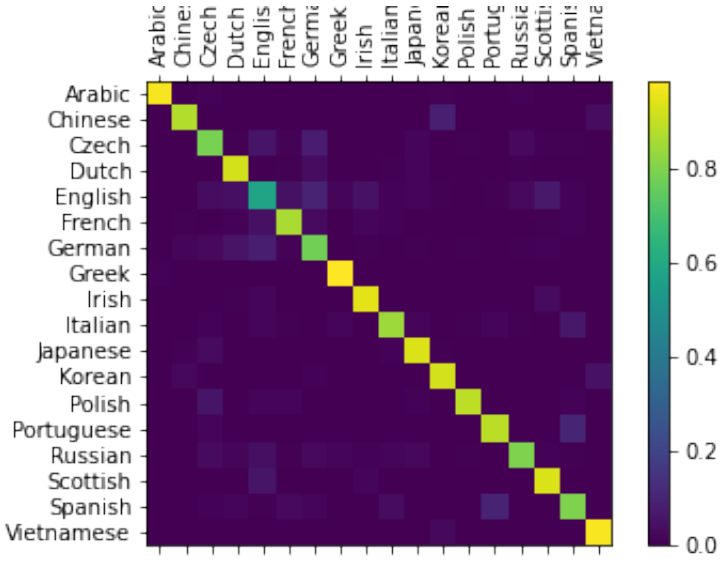


图 4.9: 自己实现的 LSTM 预测矩阵图

该 LSTM 模型的预测准确度可以超过 85%，效果相比于 RNN 有明显提升。

5 RNN 与 LSTM 的对比

LSTM 比 RNN 效果更好，主要是因为 LSTM 在处理序列数据时能够更有效地捕捉和保持长期依赖关系。

RNN 在理论上可以处理长期依赖问题，但在实践中效果不佳。随着时间步长的增加，RNN 中的梯度会以指数方式消失或爆炸，这使得它们难以学习和保持长时间间隔的信息。LSTM 通过引入记忆单元和三个门控机制（输入门、遗忘门和输出门）来解决这个问题。

LSTM 的门控机制允许网络选择性地保留或丢弃信息。具体来说：

- 输入门：控制输入的信息有多少被更新到记忆单元。
- 遗忘门：控制记忆单元中的信息有多少被保留。
- 输出门：控制记忆单元的信息有多少被用于输出。

这些门控机制使得 LSTM 在处理长时间序列数据时可以有选择性地更新和保持重要的信息，而不是被所有历史数据所干扰。记忆单元可以保存重要的信息，门控机制则可以控制信息的流入和流出，使得 LSTM 可以有效地学习和保持长时间间隔的信息。LSTM 通过其特殊的结构确保梯度可以更好地流过多个时间步长。这在反向传播过程中极大地缓解了梯度消失和爆炸问题，使得模型可以更稳定地训练。

而对于 RNN，由于其没有门控机制，所有输入和过去的状态都会对当前状态产生影响，这可能会导致模型在面对复杂的动态变化时表现不佳。