循环神经网络实验报告

姓名：杨鑫　　学号：2011028

一、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

二、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

三、实验过程

1、原始 RNN

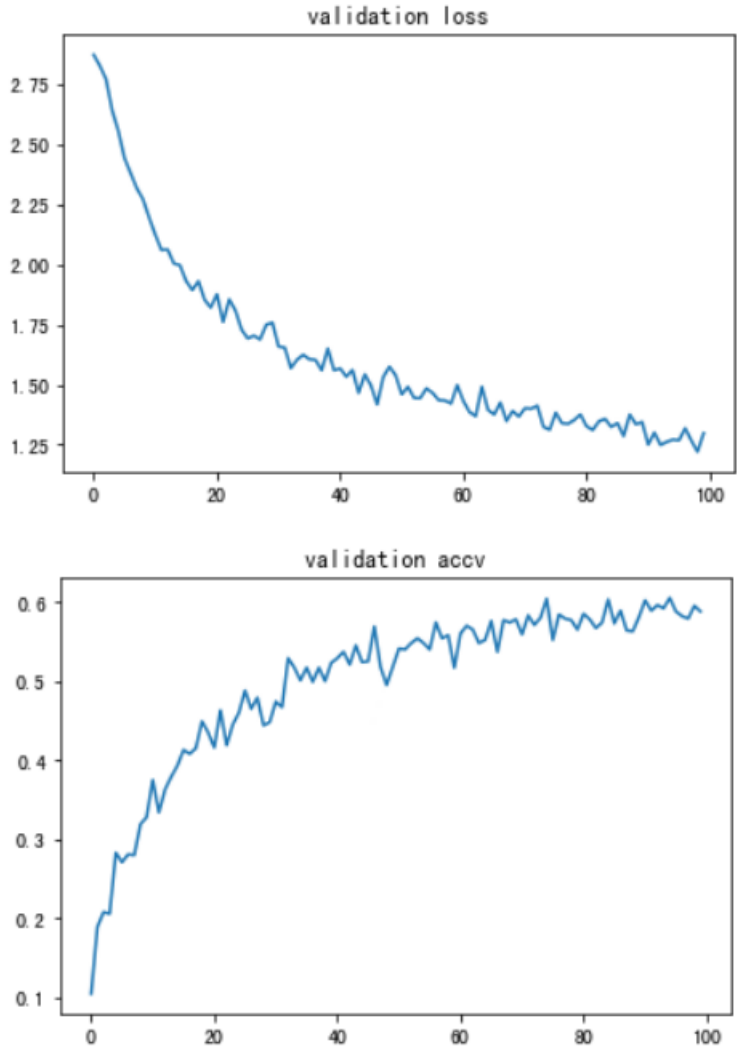
打印原始 RNN 的网络结构：

```
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)
)
```

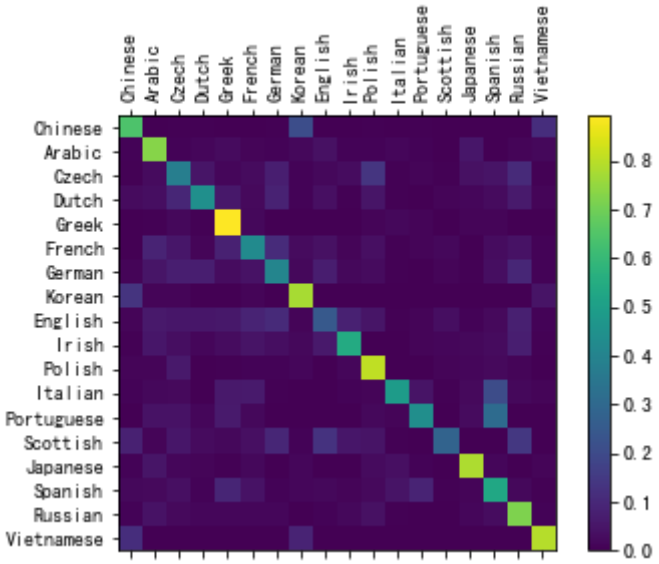
这是一个简单的循环神经网络（RNN）结构，包含三个层：

- 输入层到隐藏层的线性变换层（i2h），输入特征维度为 185，隐藏层维度为 128。
- 输入层到输出层的线性变换层 (i2o)，输入特征维度为 185，输出维度为 18。
- 输出层的 softmax 函数，用于将输出转化为概率分布。

设置学习率为 0.005(经过测试发现 0.005 的时候训练效果最好)，设置迭代次数为 100000，在 CPU 上训练的时间消耗为 2m25s。训练 loss 曲线、准确度曲线图如下：



得到的 RNN 预测矩阵图如下：



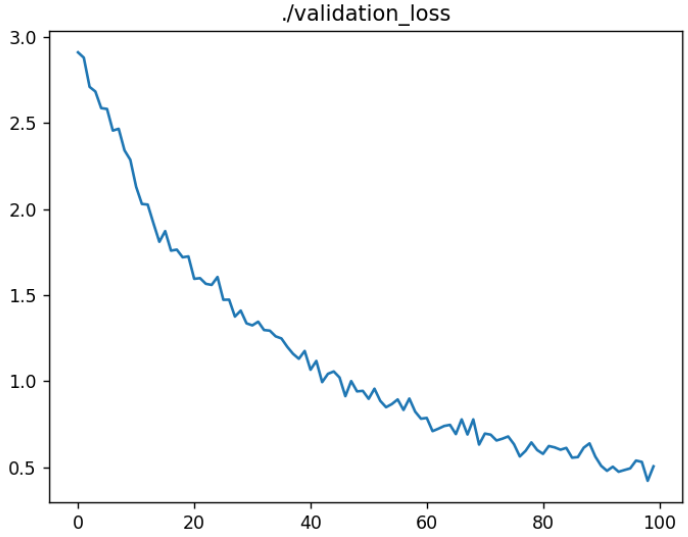
2、利用 Pytorch 搭建 LSTM

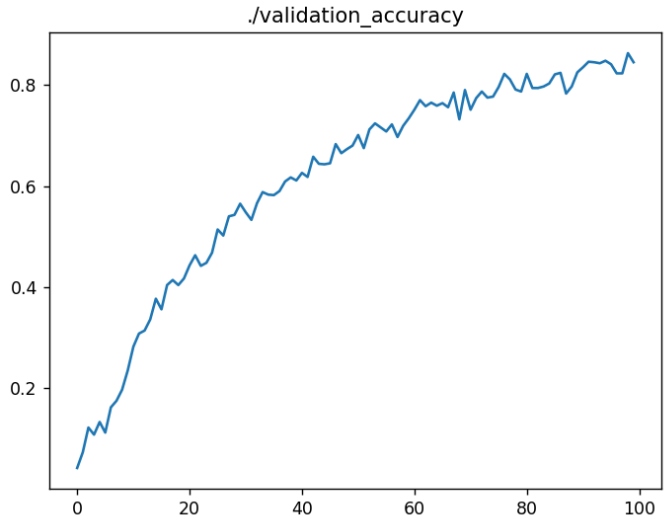
利用 Pytorch 搭建的 LSTM 网络结构如下：

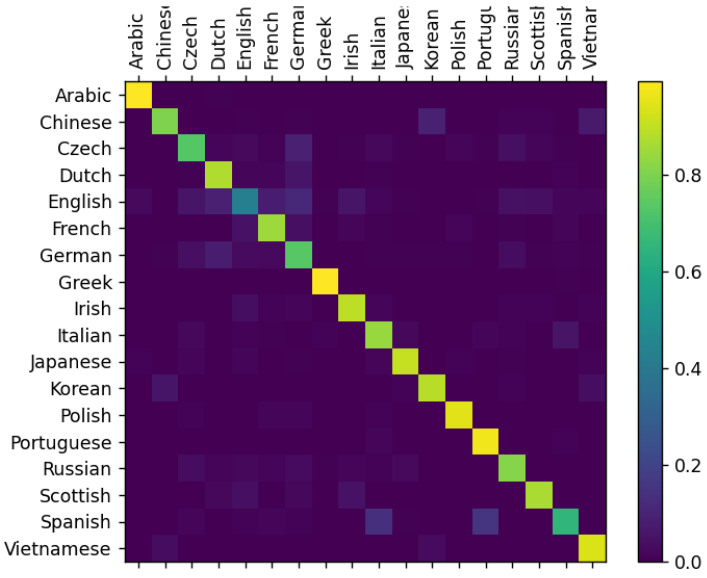
```
LSTM(
  (lstm): LSTM(57, 128, num_layers=2)
  (linear): Linear(in_features=128, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

搭建了一个两层的 LSTM 网络，输入维度为 57，隐藏层维度为 128，输出维度为 18。其中，LSTM 层的输入维度为 57，这个值是根据输入数据的特征维度而定的。在这个例子中，57 是指数据集中所有字符的数量（大小写字母、标点符号、空格等），因此输入向量的长度为 57。LSTM 层的隐藏层维度为 128，意味着 LSTM 层有 128 个隐藏单元。最后，输出层是一个线性层，将 LSTM 层的输出映射为 18 个类别（即 18 个国家）。输出通过 LogSoftmax 函数进行归一化，以便获得每个类别的概率分布。

设置学习率为 0.08(经过测试发现 0.08 的时候训练效果最好)，设置迭代次数为 100000，在 CPU 上训练的时间消耗为 3m34s。训练 loss 曲线、准确度曲线图如下：





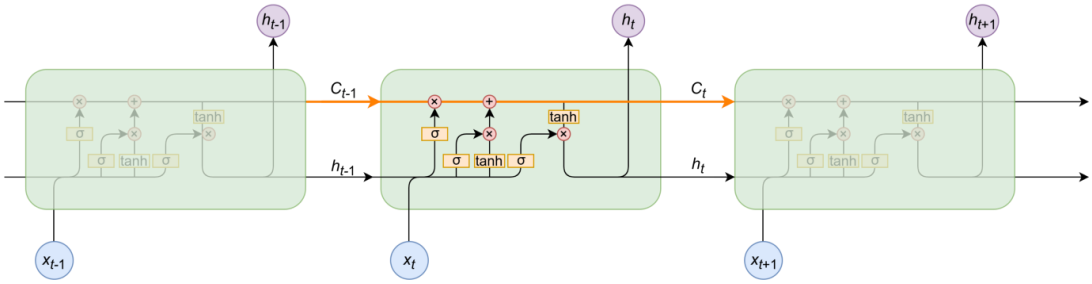


可以看到经过 LSTM 训练后的模型的准确率显著提高。

3、手动实现 LSTM

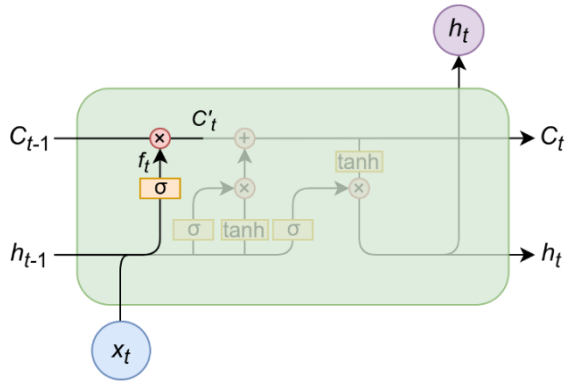
在动手实现 LSTM 之前需要了解一下 LSTM 的工作原理，LSTM 是 RNN 网络中最有趣的结构之一，不仅仅使得模型可以从长序列中学习，还创建了长短期记忆模块，模块中所记忆的数值在需要时可以得到更改。

LSTM 的结构如下：



LSTM 由门控单元组成，一种可由数学运算后控制信息是否传递或保持的计算结构，这因为这种结构，使得模型可以决定是在长期还是短期记忆中进行输出。

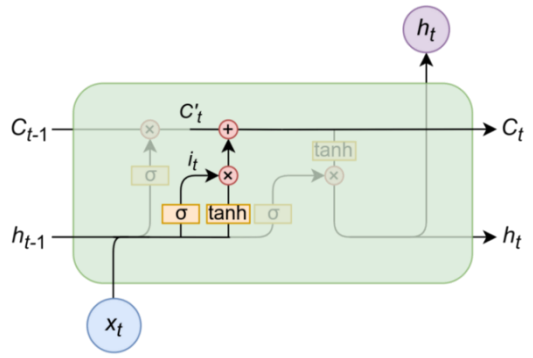
遗忘门：



$$f_t = \sigma\left(U_f x_t + V_f h_{t-1} + b_f\right)$$
$$C'_t = f_t \cdot C_{t-1}$$

遗忘单元可以将**输入信息和隐藏信息进行信息整合**，并进行信息更替，更替步骤如右图公式，其中**与乘上权重矩阵后，加上偏置项后，经过激活函数**，此时输出值为位于[0,1]之间,并将上一个时间步的与激活函数输出值相乘，更新为

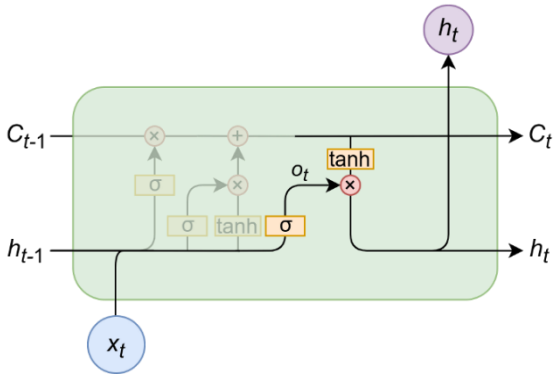
输入门：



$$i_t = \sigma(U_i\, x_t + V_i\, h_{t-1} + b_i)$$
$$C_t^+ = \tanh(U_c\, x_t + V_c\, h_{t-1} + b_c)$$
$$C_t = C'_t + i_t \cdot C_t^+$$

当有输入进入时，**输入门会结合输入信息与隐藏信息进行整合**，并对信息进行更替过程与过程类似，中间公式使用了 tanh 函数，可以将输出缩放到[-1,1]之间，再更新

输出门：



$$o_t = \sigma(U_o\, x_t + V_o\, h_{t-1} + b_o)$$
$$h_t = o_t \cdot \tanh(C_t)$$

输出门也会对输出过程进行控制，与输入门不同的是，输出门使用 tannh 激活函数。

因此最后 LSTM 的公式如下：

$$f_t = \sigma(U_f\, x_t + V_f\, h_{t-1} + b_f)$$
$$i_t = \sigma(U_i\, x_t + V_i\, h_{t-1} + b_i)$$
$$o_t = \sigma(U_o\, x_t + V_o\, h_{t-1} + b_o)$$
$$g_t = \tanh\,(U_g\, x_t + V_g\, h_{t-1} + b_g) \text{ a.k.a. } \tilde{c}_t$$
$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$
$$h_t = o_t \circ \tanh\,(c_t)$$

因此使用 pytorch 实现的代码如下：

```
class myLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(myLSTM, self).__init__()
        self.lstm = CustomLSTM(input_size, hidden_size)
        self.linear = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):
        out, (h_n, c_n) = self.lstm(x)
        out = (self.linear(out[-1]))
        out = self.softmax(out)
        return out


class CustomLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(CustomLSTM, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim

        """input gate"""
        self.linear_i_x = nn.Linear(self.input_dim, self.hidden_dim)
        self.linear_i_h = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.linear_i_c = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.i_sigmod = nn.Sigmoid()

        """forget gate"""
        self.linear_f_x = nn.Linear(self.input_dim, self.hidden_dim)
        self.linear_f_h = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.linear_f_c = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.f_sigmod = nn.Sigmoid()

        """cell memeory"""
        self.linear_c_x = nn.Linear(self.input_dim, self.hidden_dim)
        self.linear_c_h = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.c_tanh = nn.Tanh()

        """output gate"""
        self.linear_o_x = nn.Linear(self.input_dim, self.hidden_dim)
        self.linear_o_h = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.linear_o_c = nn.Linear(self.hidden_dim, self.hidden_dim)
        self.o_sigmod = nn.Sigmoid()
```

```
        """hidden memory"""
        self.h_tanh = nn.Tanh()

    def input_gate(self, x, h, c):
        return self.i_sigmod(self.linear_i_x(x) + self.linear_i_h(h) +
self.linear_i_c(c))

    def forget_gate(self, x, h, c):
        return self.f_sigmod(self.linear_f_x(x) + self.linear_f_h(h) +
self.linear_f_c(c))

    def cell_memory(self, i, f, x, h, c):
        return f * c + i * self.c_tanh(self.linear_c_x(x) +
self.linear_c_h(h))

    def output_gate(self, x, h, c_next):
        o = self.o_sigmod(self.linear_o_x(
            x) + self.linear_o_h(h) + self.linear_o_c(c_next))
        return o * self.h_tanh(c_next)

    def hidden_memory(self, c_next, o):
        return o * self.h_tanh(c_next)

    def init_hidden_cell(self, x):
        """initial hidden and cell"""
        h_0 = x.data.new(x.size(0), self.hidden_dim).zero_()
        c_0 = x.data.new(x.size(0), self.hidden_dim).zero_()
        return (h_0, c_0)

    def forward(self, x, memory=None):
        _, seq_sz, _ = x.size()
        hidden_seq = []
        if memory is not None:
            h, c = memory
        else:
            h, c = self.init_hidden_cell(x)
        for t in range(seq_sz):
            x_t = x[:, t, :]
            i = self.input_gate(x_t, h, c)  # (x.size(0), hidden_dim)
            f = self.forget_gate(x_t, h, c)
            c = self.cell_memory(i, f, x_t, h, c)
            o = self.output_gate(x_t, h, c)
            h = self.hidden_memory(c, o)
```

```
            hidden_seq.append(h.unsqueeze(0))
        hidden_seq = torch.cat(hidden_seq, dim=0)
        hidden_seq = hidden_seq.transpose(0, 1).contiguous()

        return hidden_seq, (h, c)
```

经过调参发现在隐藏层层数为 32、学习率为 0.035 的时候效果最好，其中 validation loss 和 validation accv 以及预测矩阵图如下：