南開大學
Nankai University

南　开　大　学

计 算 机 学 院

深度学习及应用实验作业

作业三 循环神经网络实践

姓名：徐宇昂

学号：2011742

年级：2020 级

专业：计算机科学与技术

指导教师：侯淇彬

2023 年 6 月

摘要

本次实验基于 rnn 代码，个人实现 lstm 并在 names 数据集上进行训练。

关键字：前馈神经网络，pytorch，RNN

目录

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、 平凡 RNN

(一) RNN 的网络结构

原始 RNN 仅有一个隐藏层构成，它的网络结构如下图所示：

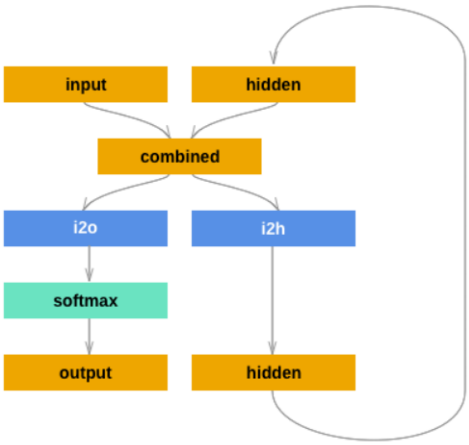
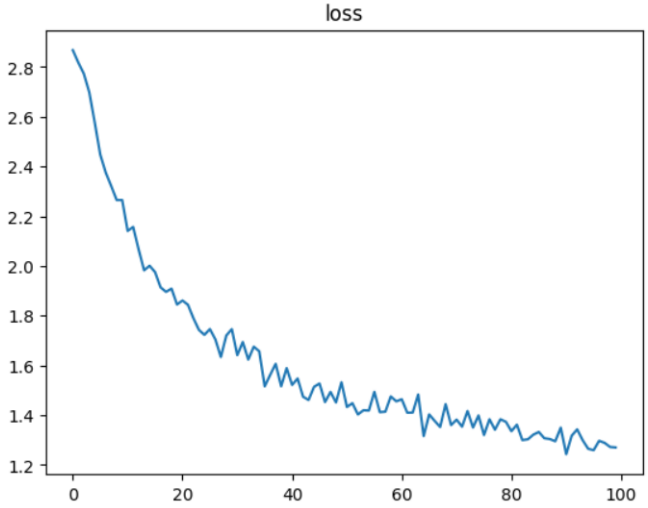


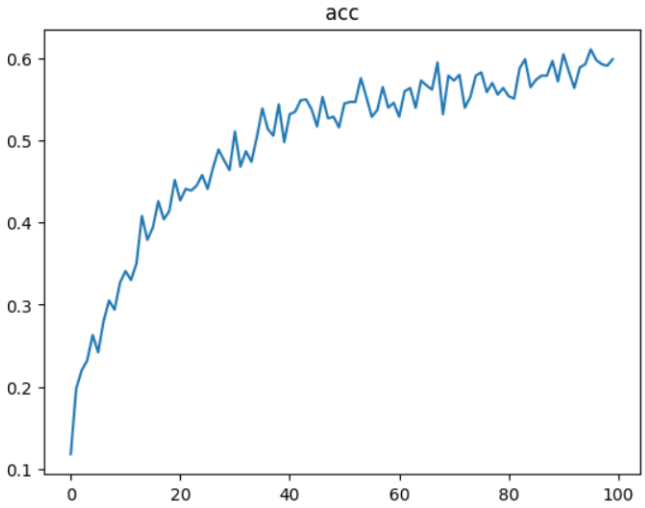
图 1: 传统 CNN 结构示意图

```
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)
)
```

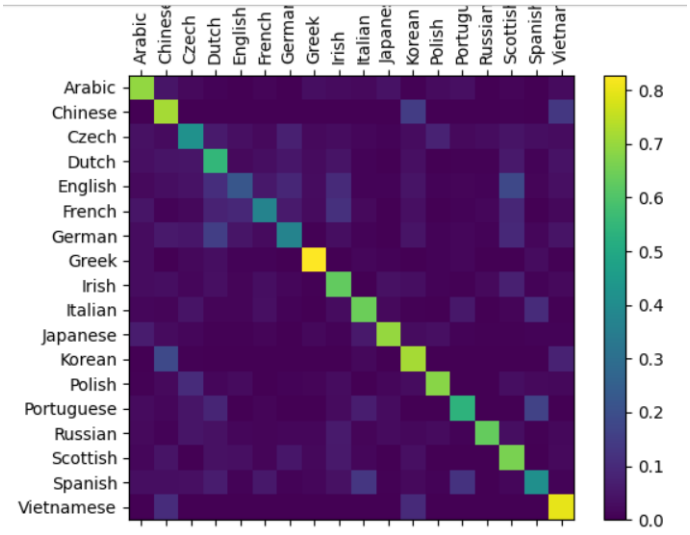
(二) RNN 的训练结果

在数据集上训练过程中的 loss 曲线和 acc 曲线如下图所示：





预测矩阵图如下图所示：



可以看出预测结果准确率还是非常不错的。

三、 LSTM 实现

(一) LSTM 网络结构

原始 RNN 的隐藏层只有一个状态，即 h，它对于短期的输入非常敏感。再增加一个状态，即 c，让它来保存长期的状态，称为单元状态 (cell state)。

如图4所示

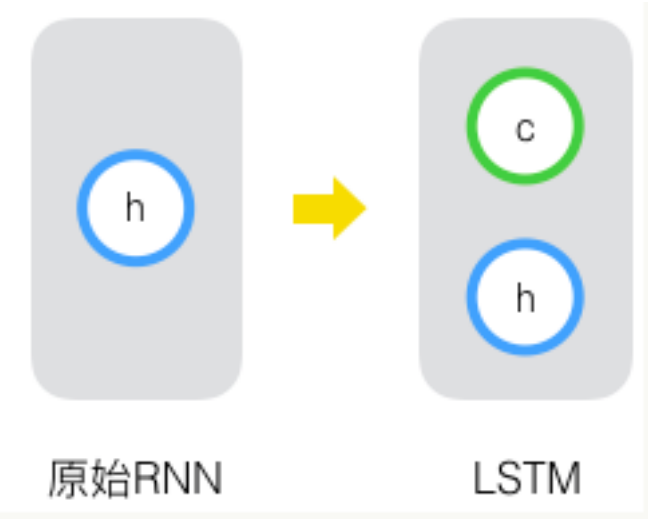


图 2: cell

把上图按照时间维度展开：

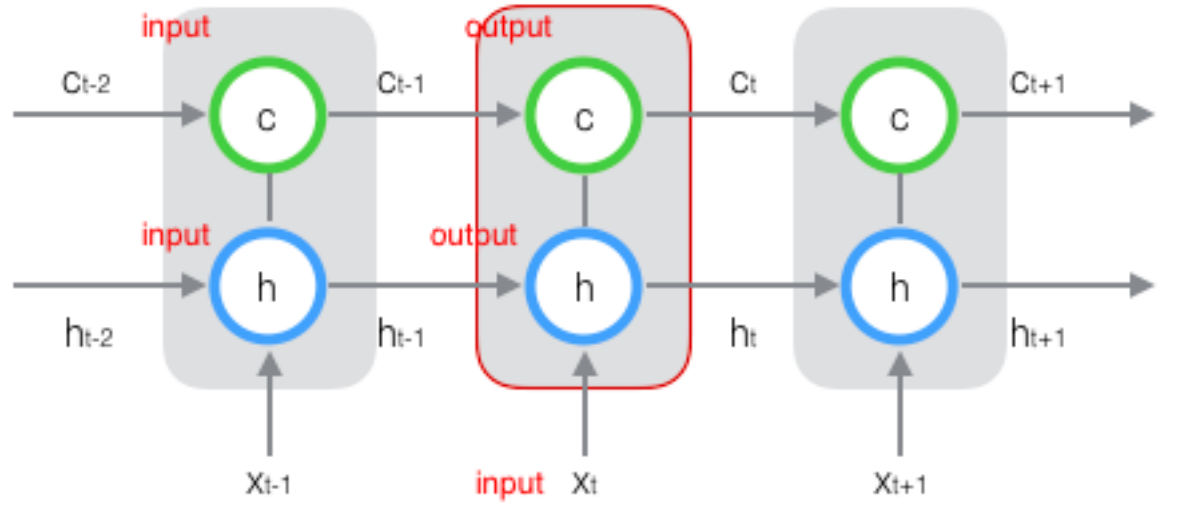


图 3: cell

在 t 时刻，LSTM 的输入有三个：当前时刻网络的输入值 x_t、上一时刻 LSTM 的输出值 h_t-1、以及上一时刻的单元状态 c_t-1；LSTM 的输出有两个：当前时刻 LSTM 输出值 h_t、和当前时刻的单元状态 c_t.

1. LSTM **的前向计算**

   1. 遗忘门：它决定了上一时刻的单元状态 c_t-1 有多少保留到当前时刻 c_t

   2. 输入门：它决定了当前时刻网络的输入 x_t 有多少保存到单元状态 c_t

   3. 输出门：控制单元状态 c_t 有多少输出到 LSTM 的当前输出值 h_t

(二) 代码实现

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

        self.num_layers = num_layers
        self.hidden_size = hidden_size
        self.directions = directions

        self.lstm = nn.LSTM(input_size,hidden_size,num_layers,batch_first=True,dropout
                                                        =dropout_prob)
        self.dropout = nn.Dropout(dropout_prob)
        self.linear = nn.Linear(hidden_size,output_size)

    def init_hidden_states(self, batch_size):
        state_dim = (self.num_layers * self.directions, batch_size, self.hidden_size)
        return (torch.zeros(state_dim).to(device), torch.zeros(state_dim).to(device))

    def forward(self, x, states):
        x = x.view(len(x),1,-1)
        x, (h, c) = self.lstm(x, states)
        out = self.linear(x)
        return out, (h, c)


n_hidden = 128
#rnn = RNN(n_letters, n_hidden, n_categories)


INPUT_SZIE = n_letters
DROPOUT = 0.2
DIRECTIONS = 1
NUM_LAYERS = 2
BATCH_SIZE = 5
OUTPUT_SIZE = n_categories
HIDDEN_SIZE = n_hidden
LEARNING_RATE = 0.0001
STATE_DIM = NUM_LAYERS * DIRECTIONS, BATCH_SIZE, HIDDEN_SIZE

lstm = LSTM(INPUT_SZIE,
    HIDDEN_SIZE,
    NUM_LAYERS,
    OUTPUT_SIZE,
    DROPOUT).to(device)
```

(三) 训练结果

训练结果如图4所示

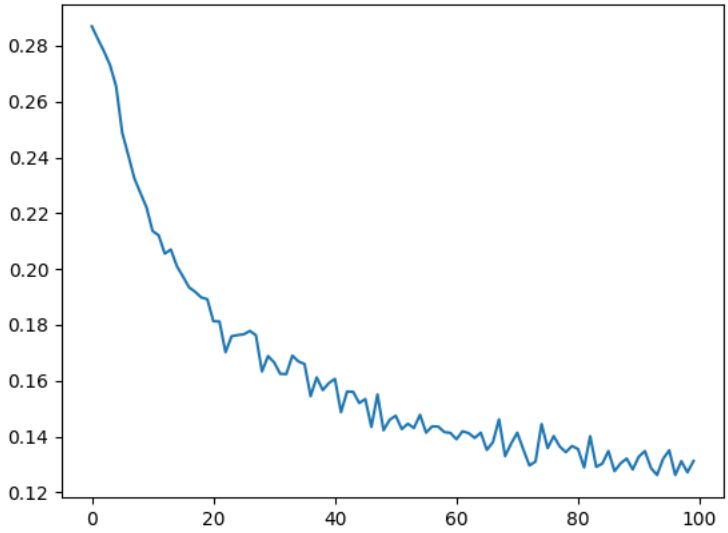


图 4: 训练结果

四、 解答

LSTM 是为了解决 RNN 的 Gradient Vanish 的问题所提出的。关于 RNN 为什么会出现 Gradient Vanish，上面已经介绍的比较清楚了，本质原因就是因为矩阵高次募导致的。下面简要解释一下为什么 LSTM 能有效避免 Gradient Vanish。对于 LSTM，有如下公式

$$c^t = f^t \odot c^{t-1} + i^t \odot g^t$$

模仿 RNN，我们来计算 $\delta^{k-1} = \partial C^t / \partial c^{k-1}$ ，有

$$\begin{aligned}\delta^{k-1} &= \frac{\partial C^t}{\partial c^{k-1}} \\ &= \frac{\partial C^t}{\partial c^k}\frac{\partial c^k}{\partial c^k - 1} \\ &= \delta^k \frac{\partial c^k}{\partial c^{k-1}} \\ &= \delta^k \left(f^t + \ldots\right)\end{aligned}$$

公式里其余的项不重要，这里就用省略号代替了。可以看出当 $f^t = 1$ 时，就算其余项很小，梯度仍然可以很好导到上一个时刻，此时即使层数较深也不会发生 Gradient Vanish 的问题; 当 $f^t = 0$ 时，即上-时刻的信号不影响到当前时刻，则梯度也不会回传回去; f^t 在这里也控制着梯度传导的🄵减程度，与它 Forget Gate 的功能一致。