



南开大学  
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：王泳鑫

学号：1911479

年级：2019级

专业：计算机科学与技术

指导教师：侯淇彬

2022 年 5 月 21 日

## 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，FNN

## 目录

一、 实验要求	1
二、 LSTM实现	1
(一) LSTM网络结构 . . . . .	1
1. LSTM的前向计算 . . . . .	2
(二) 代码实现 . . . . .	2
(三) 训练结果 . . . . .	3
三、 解答	4

## 一、 实验要求

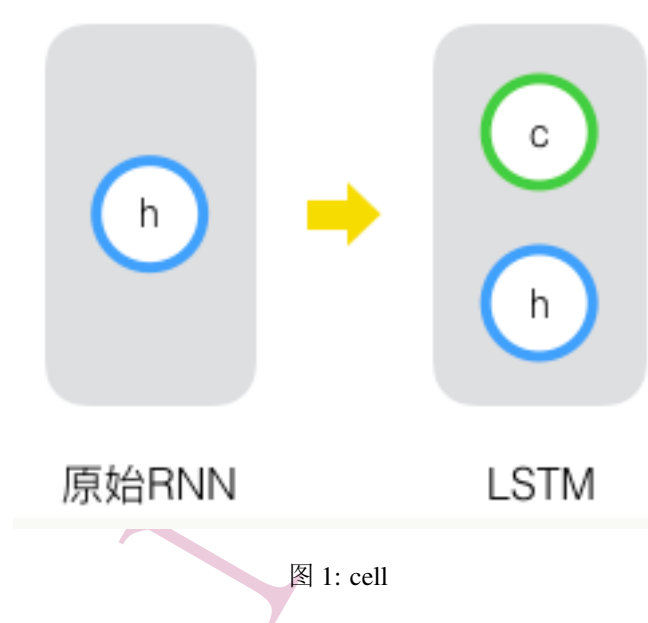
- 掌握RNN原理
- 学会使用PyTorch搭建循环神经网络来训练名字识别
- 学会使用PyTorch搭建LSTM网络来训练名字识别

## 二、 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3所示



把上图按照时间维度展开：

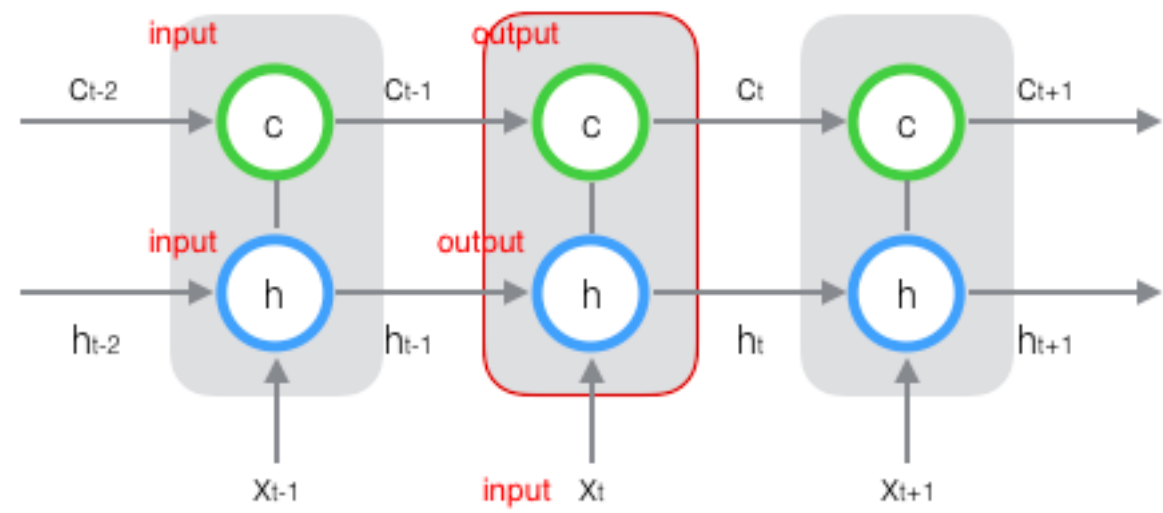


图 2: cell

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

```

1 class LSTM(nn.Module):
2     def __init__(self, input_size, hidden_size, num_layers, output_size, dropout_prob,
3                     directions = 1):
4
5         super(LSTM, self).__init__()
6
7         self.num_layers = num_layers
8         self.hidden_size = hidden_size
9         self.directions = directions
10
11        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout
12                             = dropout_prob)
13
14        self.dropout = nn.Dropout(dropout_prob)
15        self.linear = nn.Linear(hidden_size, output_size)
16
17    def init_hidden_states(self, batch_size):
18        state_dim = (self.num_layers * self.directions, batch_size, self.hidden_size)
19        return (torch.zeros(state_dim).to(device), torch.zeros(state_dim).to(device))
20
21    def forward(self, x, states):
22        x = x.view(len(x), 1, -1)

```

```
19     x, (h, c) = self.lstm(x, states)
20     out = self.linear(x)
21     return out, (h, c)
22
23
24 n_hidden = 128
25 #rnn = RNN(n_letters, n_hidden, n_categories)
26
27
28 INPUT_SIZE = n_letters
29 DROPOUT = 0.2
30 DIRECTIONS = 1
31 NUM_LAYERS = 2
32 BATCH_SIZE = 5
33 OUTPUT_SIZE = n_categories
34 HIDDEN_SIZE = n_hidden
35 LEARNING_RATE = 0.0001
36 STATE_DIM = NUM_LAYERS * DIRECTIONS, BATCH_SIZE, HIDDEN_SIZE
37
38 lstm = LSTM(INPUT_SIZE,
39             HIDDEN_SIZE,
40             NUM_LAYERS,
41             OUTPUT_SIZE,
42             DROPOUT).to(device)
```

### (三) 训练结果

训练结果如图3所示

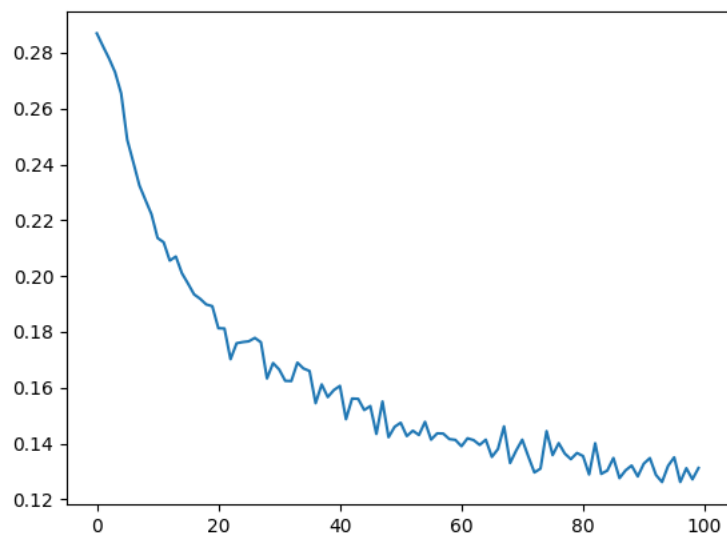


图 3: 训练结果

### 三、 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，本质原因就是矩阵高次幂导致的。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

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

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衰减程度，与它 Forget Gate 的功能一致。