深度学习

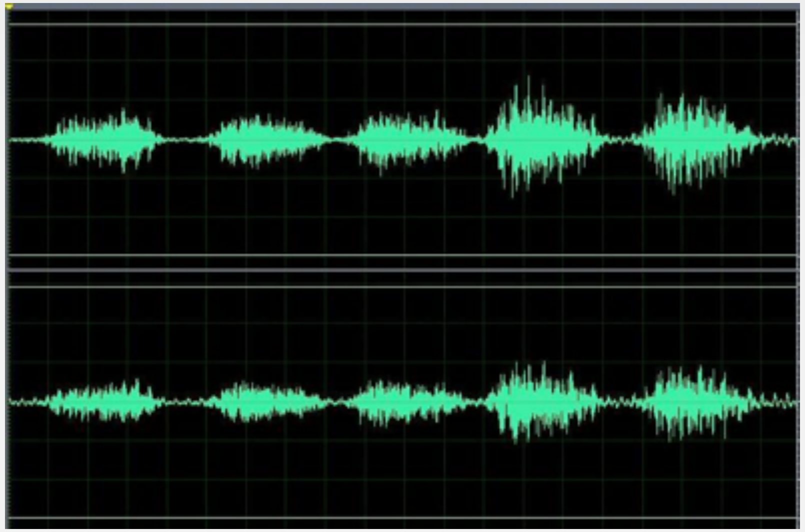
神经莫扎特

运用RNN进行文本生成

张江

集智AI学园
campus.swarma.org

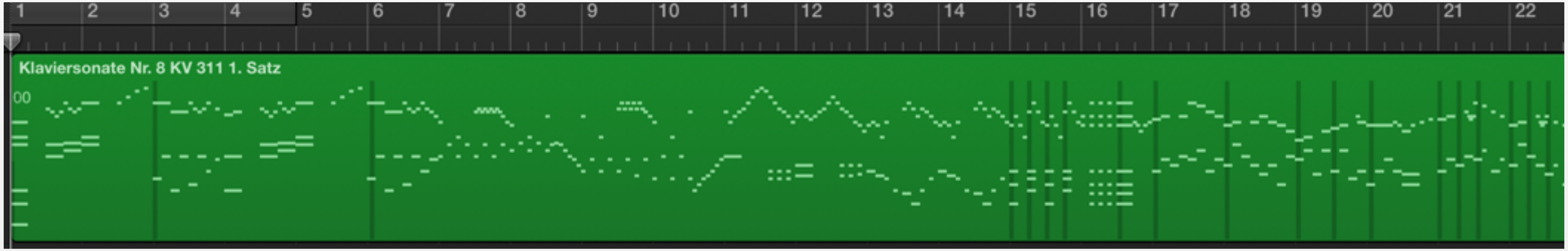
我们生活在一个序列的世界



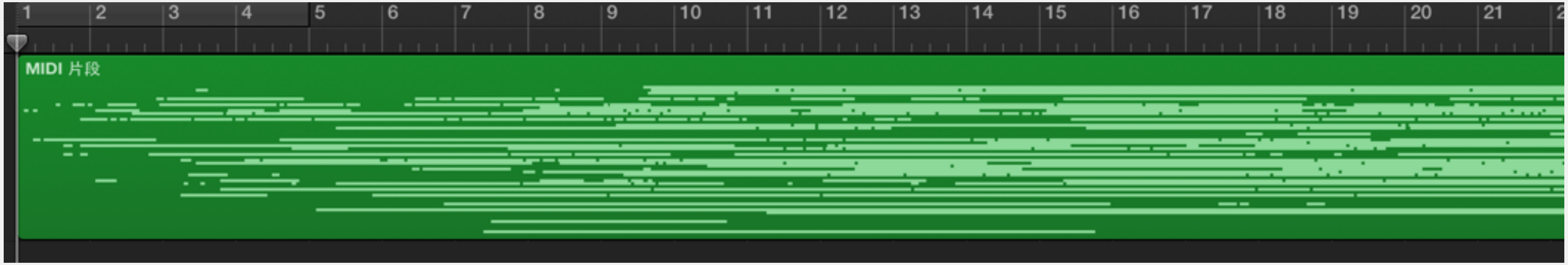
序列生成问题

- A yellow bus driving down a road with green trees and green grass in the background.

神经网络莫扎特



用莫扎特的原曲作为训练数据，让RNN学习其中的Pattern，然后再让它重新创造一段曲子



序列生成问题通用解决方法

基本思想

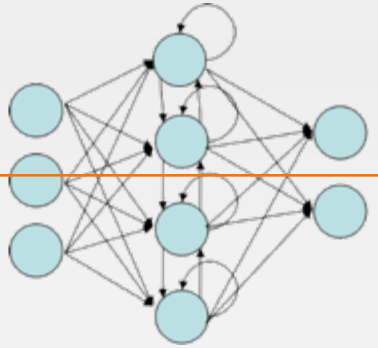
1. 将生成问题转化成一个预测问题;
2. 完成自举过程：给定一个种子，不断用已经生成的数据预测下一个数据

序列生成问题通用解决方法：第一步

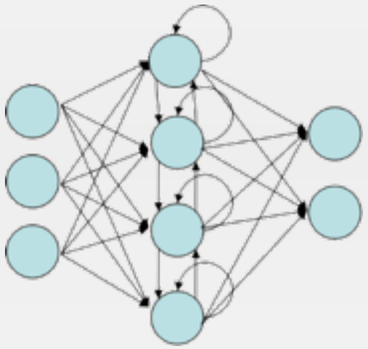
我爱北京天安？ → 门

序列生成问题通用解决方法：第一步

我爱北京天安? ⟶ 门

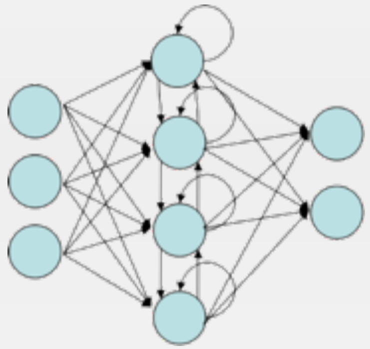
$w_1, w_2, w_3, \ldots$ ⟶  ⟶ w_n

序列生成问题通用解决方法：第二步

w_1, w_2, w_3 →  → w_4

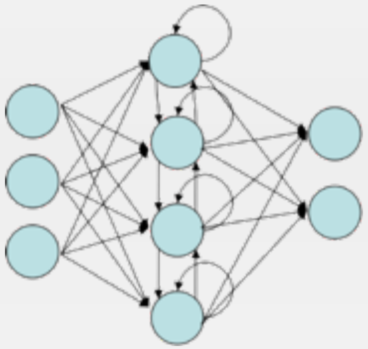
w_1, w_2, w_3, w_3, w_4

序列生成问题通用解决方法：第二步

$w_{1,}w_2,w_3,w_4$ →  → w_5

w_1,w_2,w_3,w_3,w_4,w_5

序列生成问题通用解决方法：第二步

w_1, w_2, w_3, w_4, w_5 →  → w_6

$w_1, w_2, w_3, w_3, w_4, w_5, w_6$

上下文无关语法生成器

让RNN学会一定的语法：

01
0011
0000011111
000111
01
000111
00000000001111111111

训练数据

0^n1^n

为什么要研究这样的语法?

- 它足够简单
- 自然序列中存在着类似的Pattern
    - The evidence was convincing(nv)
    - The evidence the lawyer provided was convincing(nnvv)
    - The evidence the lawyer the gangster retained provided was convincing(nnnvvv)
- 学术价值

下一个数字是几?

0000

下一个数字是几？

00000

下一个数字是几?

000001

下一个数字是几?

0000011

下一个数字是几?

00000111

下一个数字是几?

000001111

下一个数字是几？

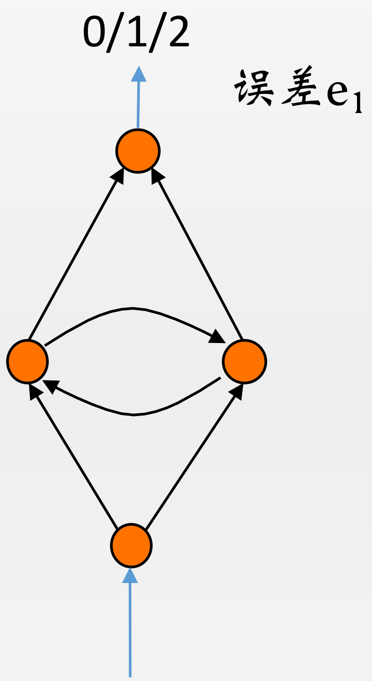
0000011111

问题的关键点和难度

- 问题的关键点在第一个1出现之后
  - 000001……
  - 后续的1必然要与前面的0的个数相匹配
- RNN必须自己学会计数
  - n没有显示地表达在输入数字中

RNN如何工作

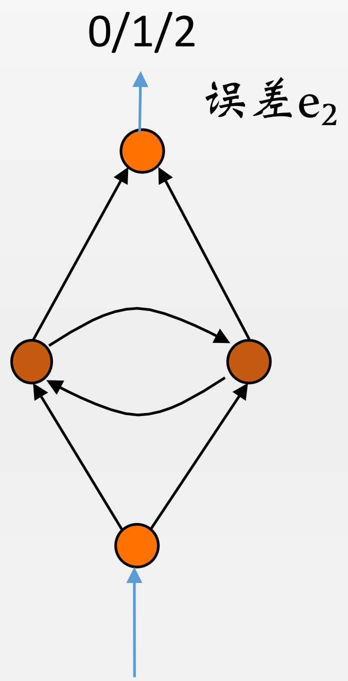
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2



3 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

RNN如何工作

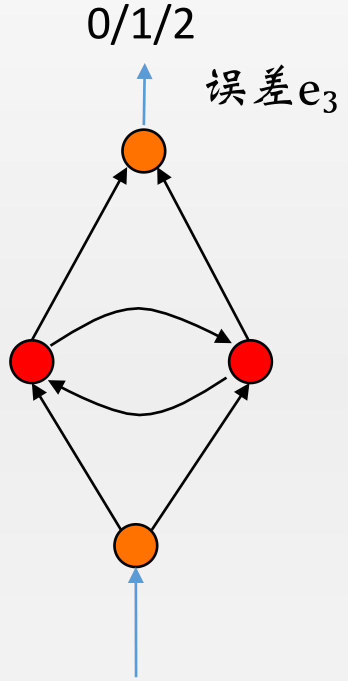
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2



3 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

RNN如何工作

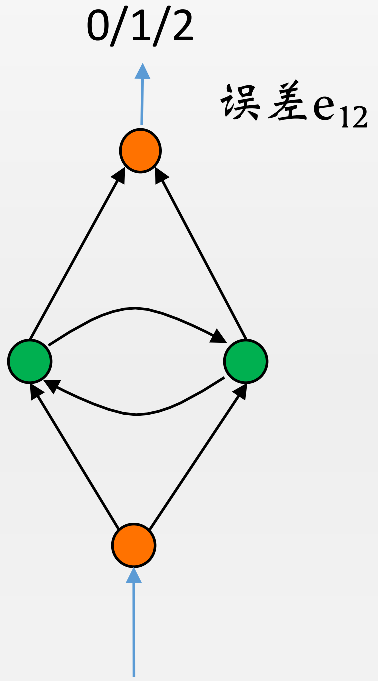
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2



3 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

RNN如何工作

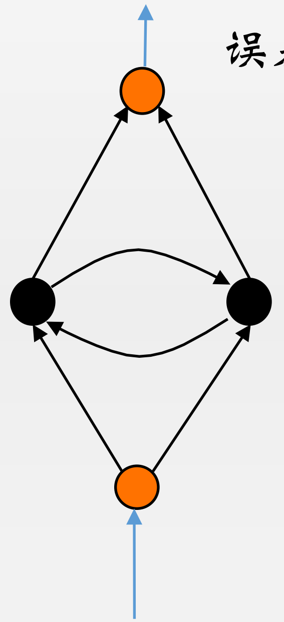
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2



3 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

RNN如何工作

0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2



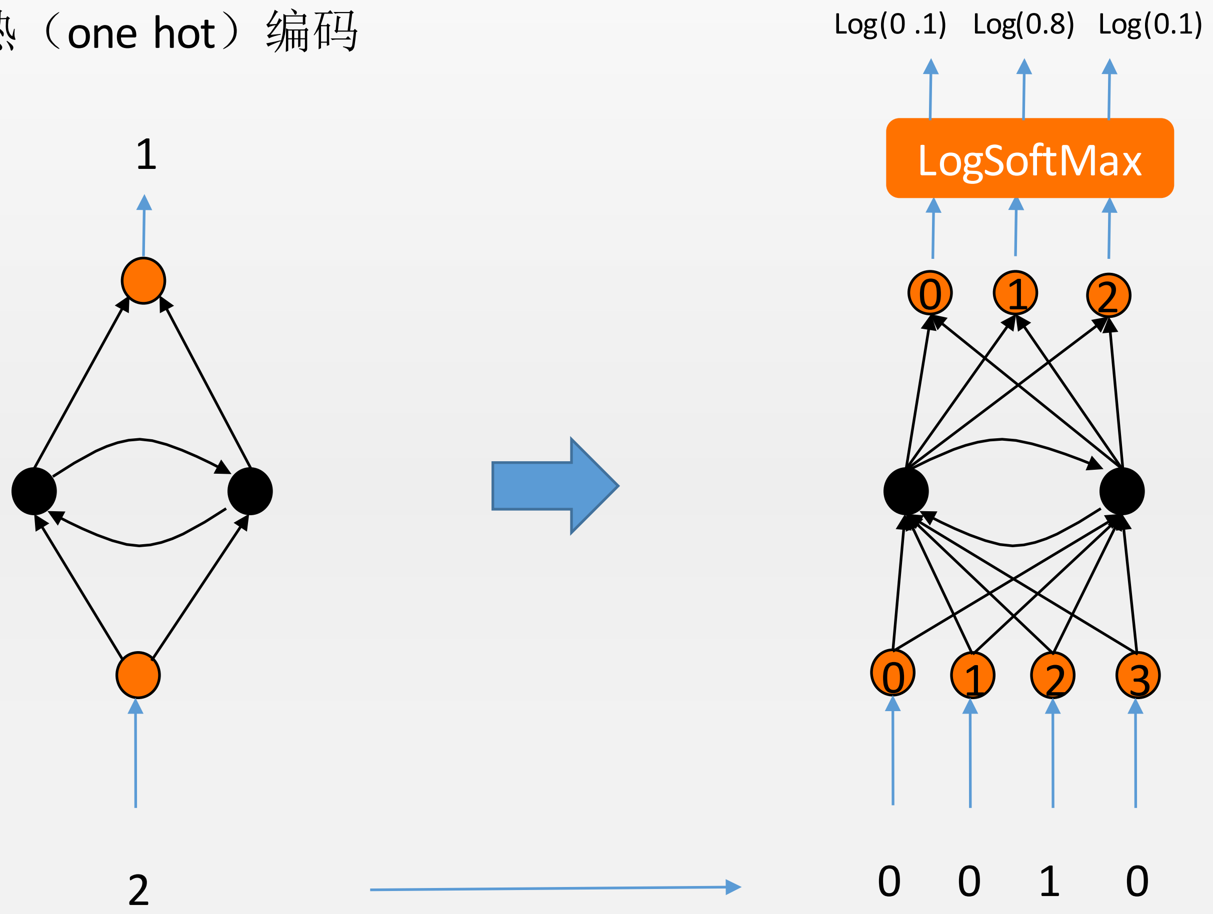
将每一步的误差加总、反传

3 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

真正的神经网络

输出：0，1，2三种可能

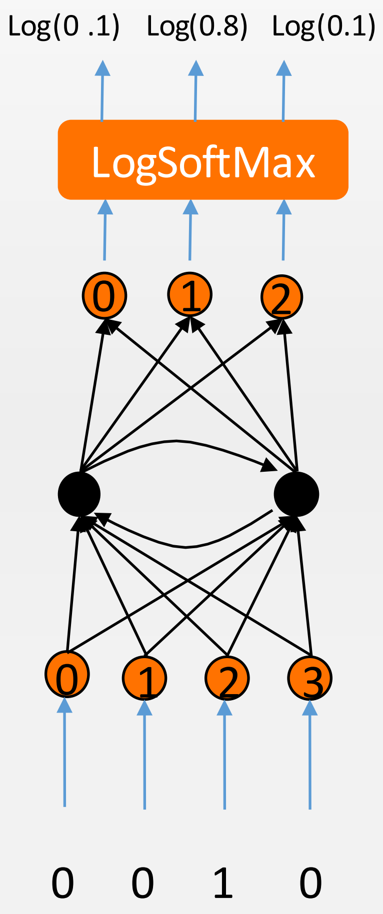
需要将类型变量变成独热（one hot）编码



输入：0，1，2，3四种可能

损失函数

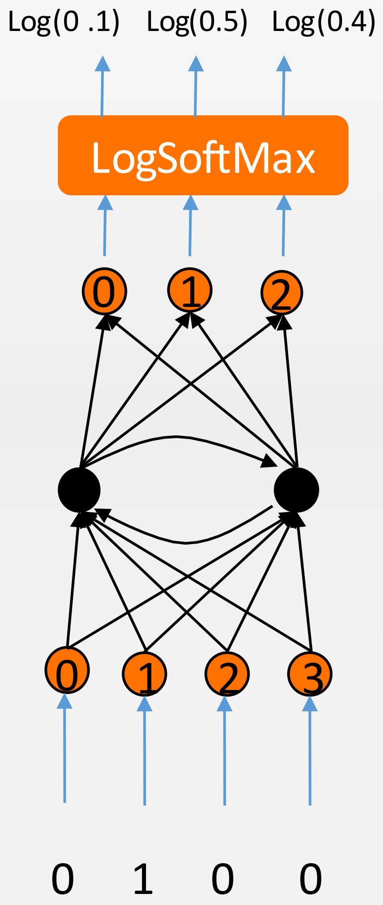
标准答案:　　1



$$L_{t=0} = -\log p_1 = -\log(0.8)$$

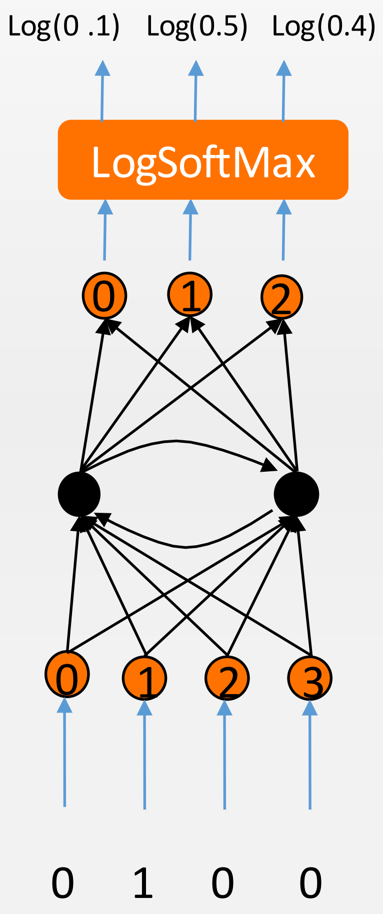
损失函数

标准答案： 0



$$L_{t=1} = -\log p_0 = -\log(0.1)$$

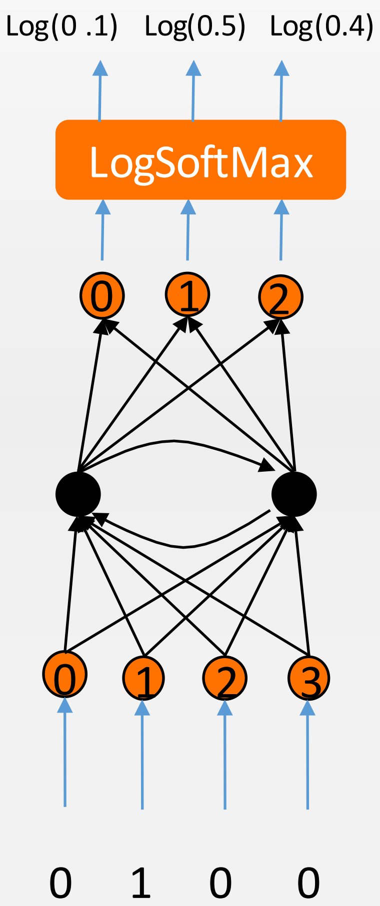
损失函数



假设序列总长度为T，则：

$$L_{总} = -(\log p_0 + \log p_1 + \log p_2 + \cdots \log p_T)$$

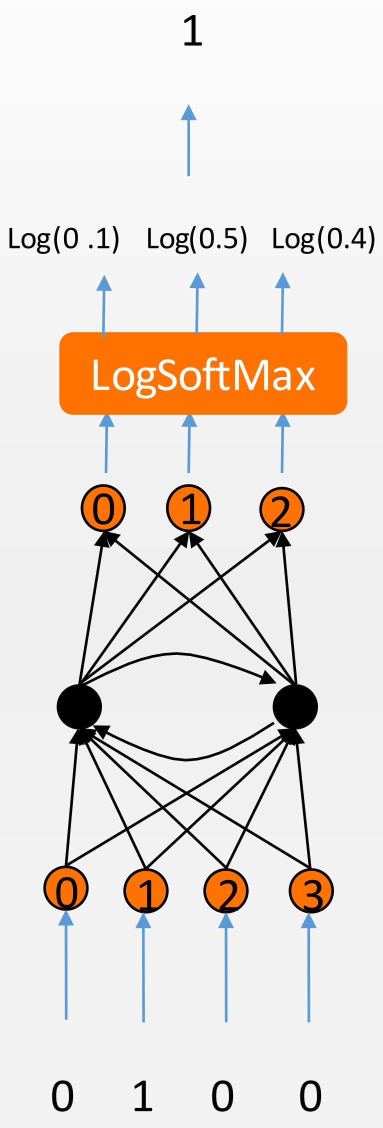
测试效果



n	目标序列	RNN预测	不同的位数量
0	2	0	1
1	012	002	1
2	00112	00012	1
3	0001112	0000112	1
4	000011112	0000011112	1
……	……	……	……
13	000000000000011111111111112	00000000000000111111111112	1
14	00000000000000111111111111112	00000000000000011111111111212	2

N>13以后，开始出现更多的错误，说明2个隐含单元的RNN的记忆容量就是13

第二阶段：序列生成



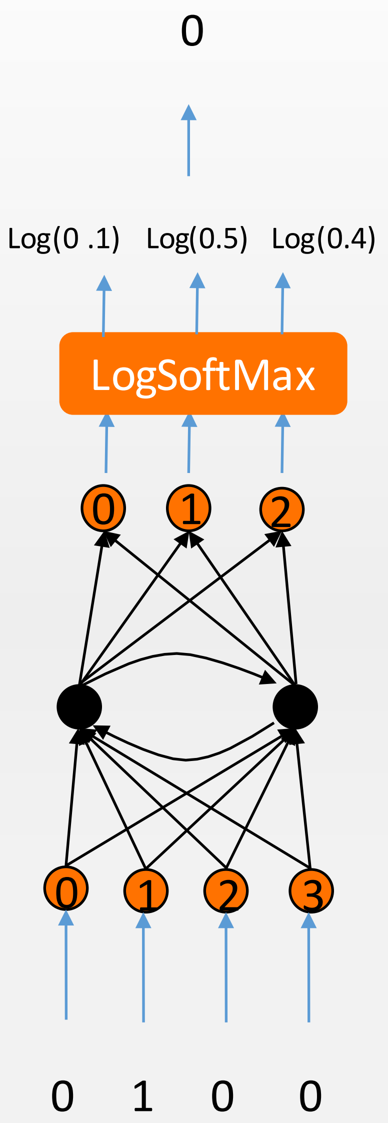
torch.multinomial(output.view(-1).exp())

output.view(-1).exp():

0.1, 0.5, 0.4

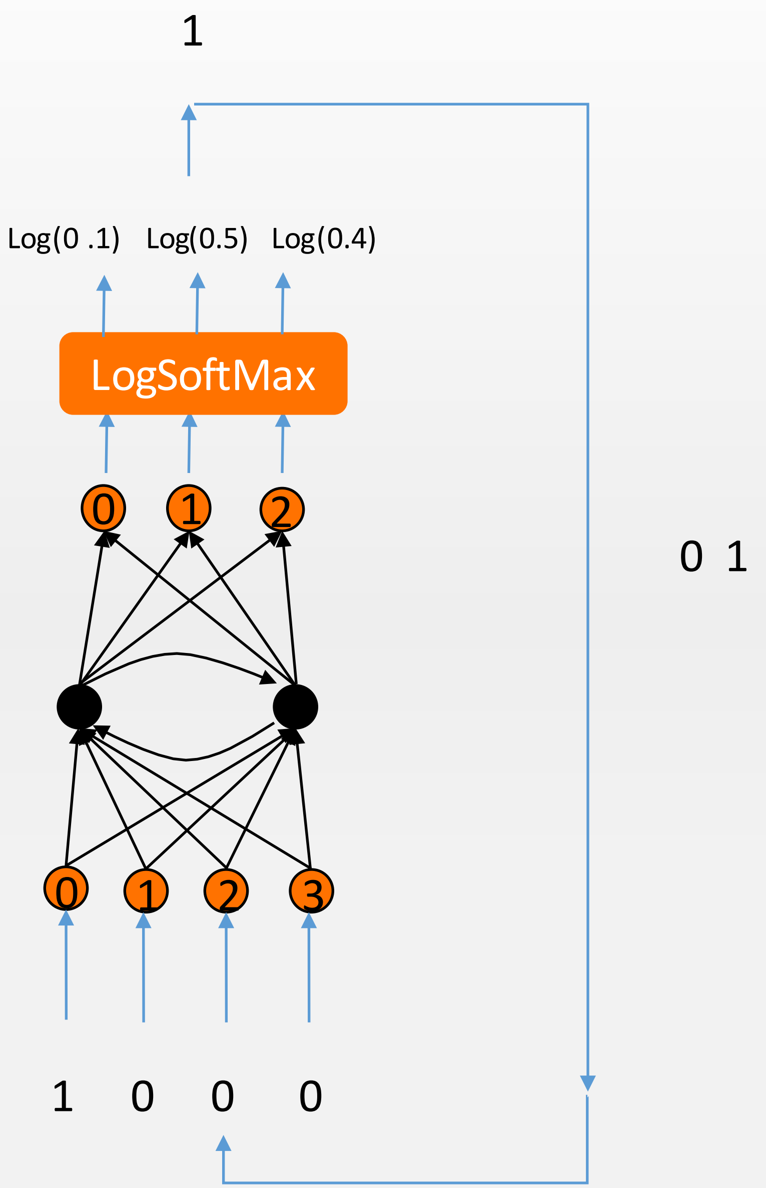
multinomial依概率（0.1,0.5,0.4）投掷骰子，并选择其中一个

第二阶段：序列生成

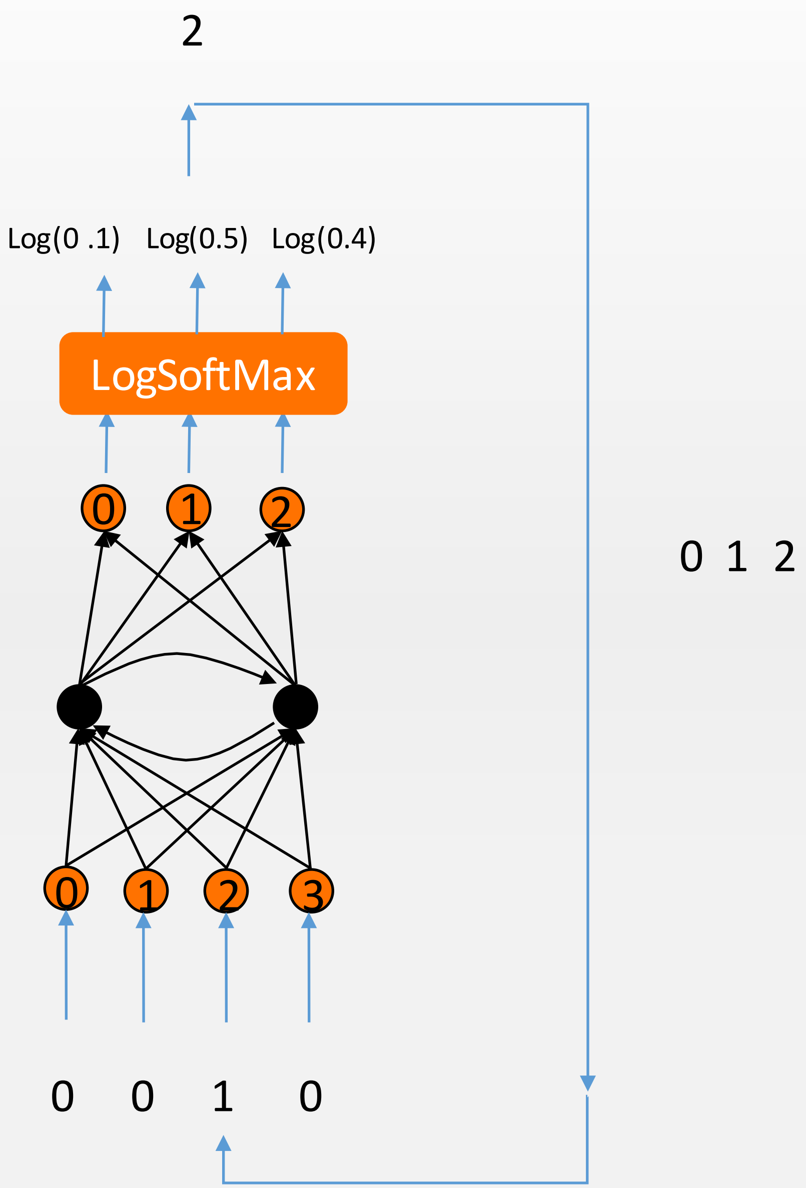


0

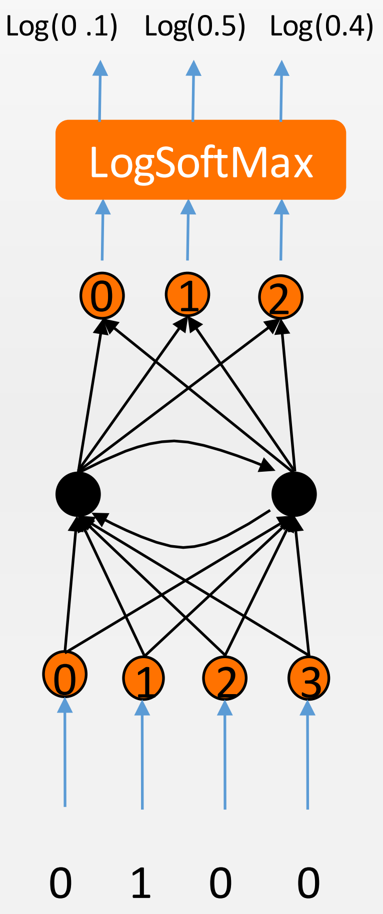
第二阶段：序列生成



第二阶段：序列生成



RNN的PyTorch实现



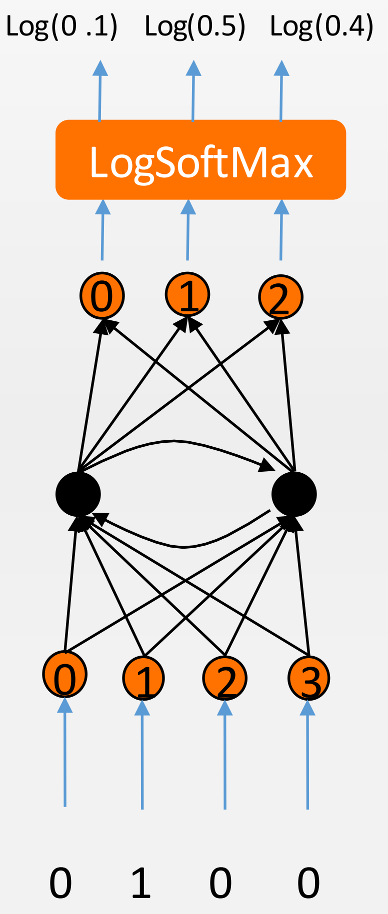

```python
class SimpleRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers = 1):
        super(SimpleRNN, self).__init__()

        self.hidden_size = hidden_size
        self.num_layers = num_layers
        # 一个embedding层
        self.embedding = nn.Embedding(input_size, hidden_size)
        # 隐含层内部的相互链接
        self.rnn = nn.RNN(hidden_size, hidden_size, num_layers, batch_first = True)
        # 输出的全链接层
        self.fc = nn.Linear(hidden_size, output_size)
        # 最后的logsoftmax层
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):

        # 先进行embedding层的计算，它可以把一个
        x = self.embedding(input)
        # 从输入到隐含层的计算
        output, hidden = self.rnn(x, hidden)
        # 从输出output中取出最后一个时间步的数值
        output = output[:,-1,:]
        # 喂入最后一层全链接网络
        output = self.fc(output)
        # softmax函数
        output = self.softmax(output)
        return output, hidden
```

RNN函数的坑



self.rnn = nn.RNN(hidden_size, hidden_size, num_layers, **batch_first = True**)

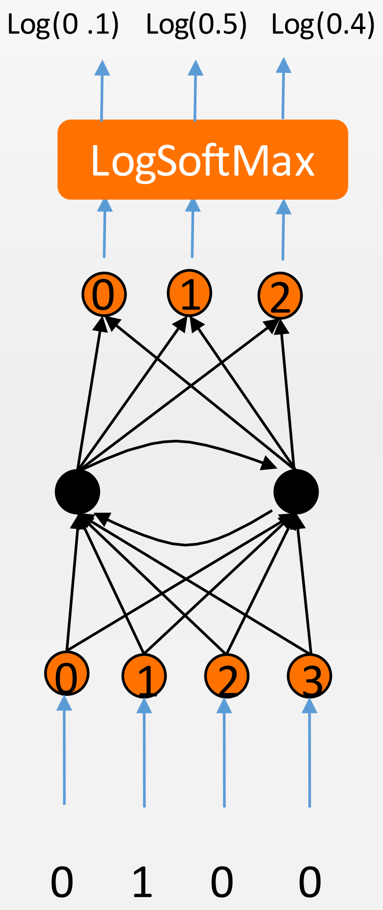
output, hidden = self.rnn(x, hidden)

- x: batch_size * time_step * input_size
- hidden : num_layers * batch_size * hidden_size
- output: time_step * batch_size * hidden_size

例子：

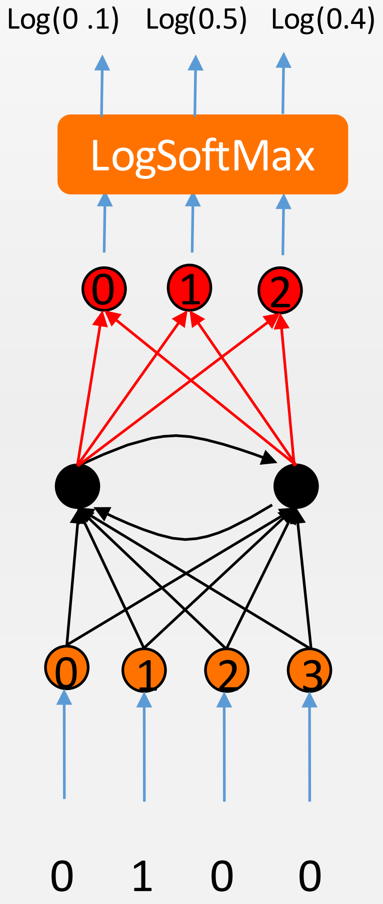
```
>>> rnn = nn.RNN(10, 20, 2)    ： 输入单元10， 隐含单元20， 2层
>>> input = Variable(torch.randn(5, 3, 10)) ： 5个时间步， batch_size =3, 10维的输入向量
>>> h0 = Variable(torch.randn(2, 3, 20)) ： 2层， batch： 3， 隐含单元20
>>> output, hn = rnn(input, h0)： output为一个5*3*20的张量
```

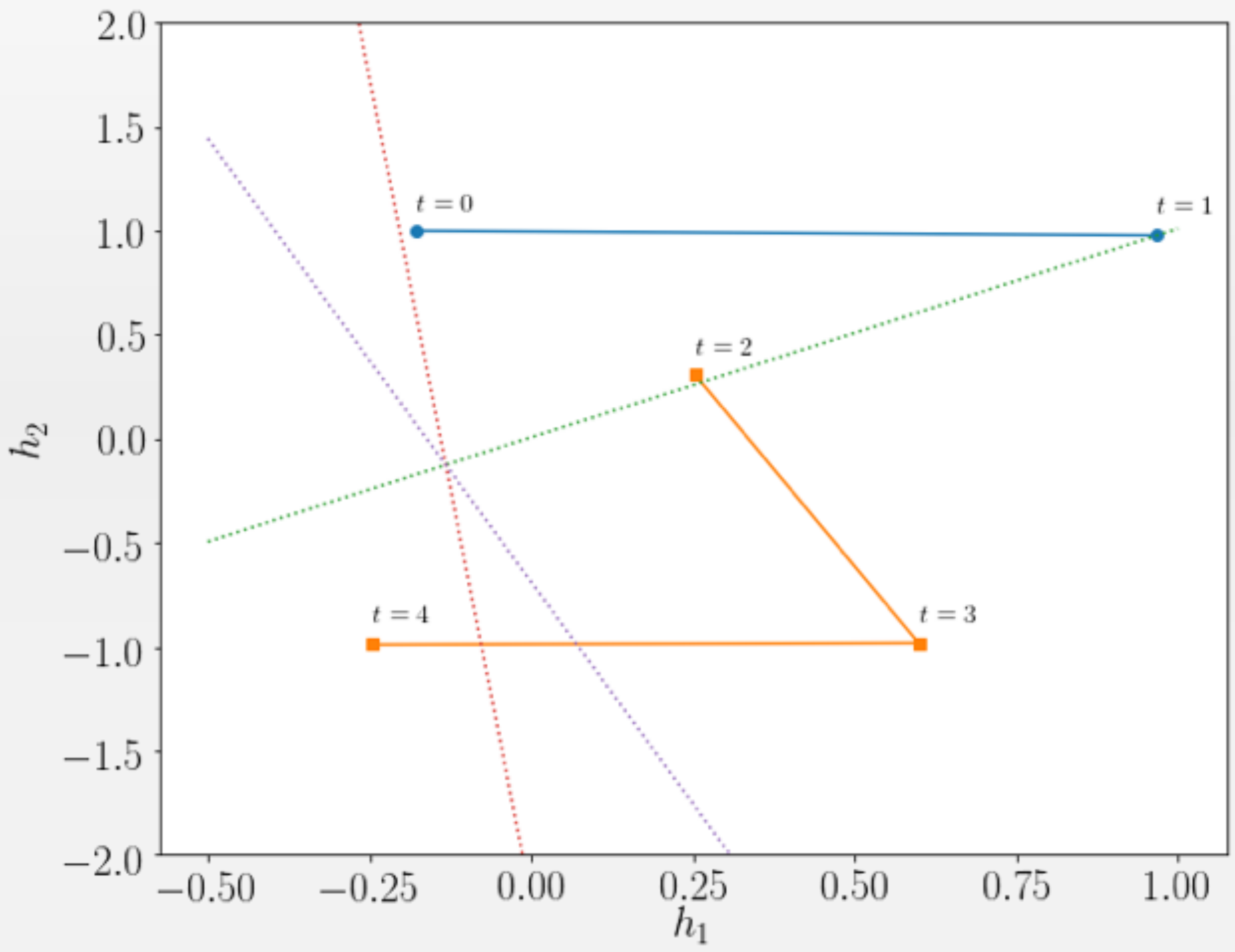
解剖这个RNN



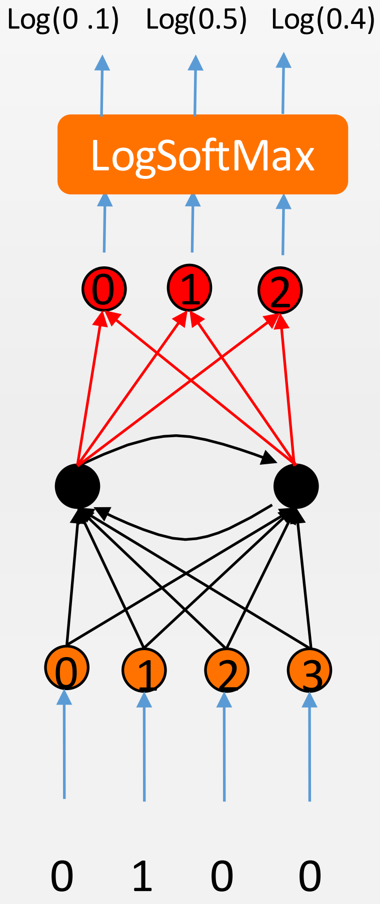
隐含单元分析

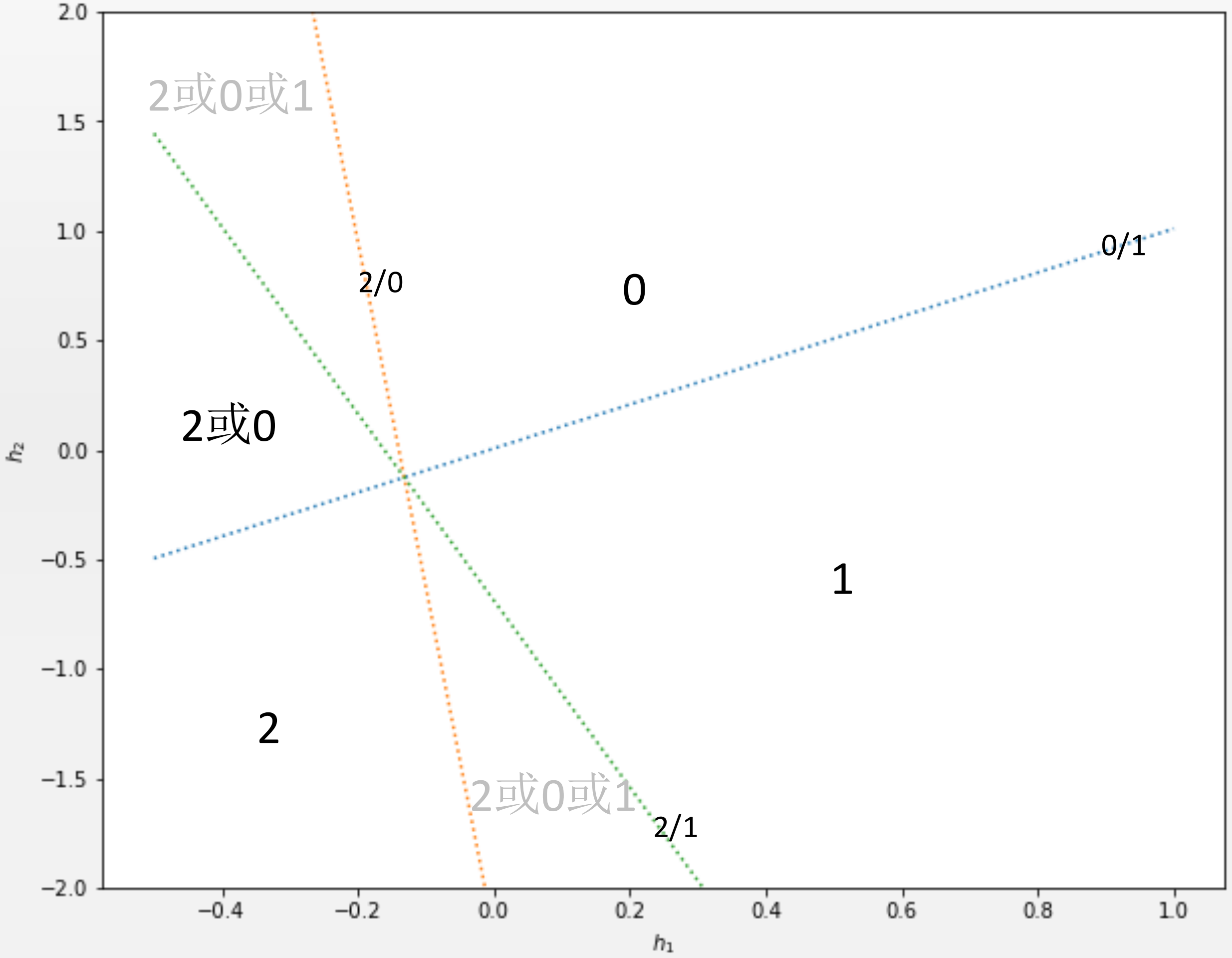
Input: 00112, output: 00012





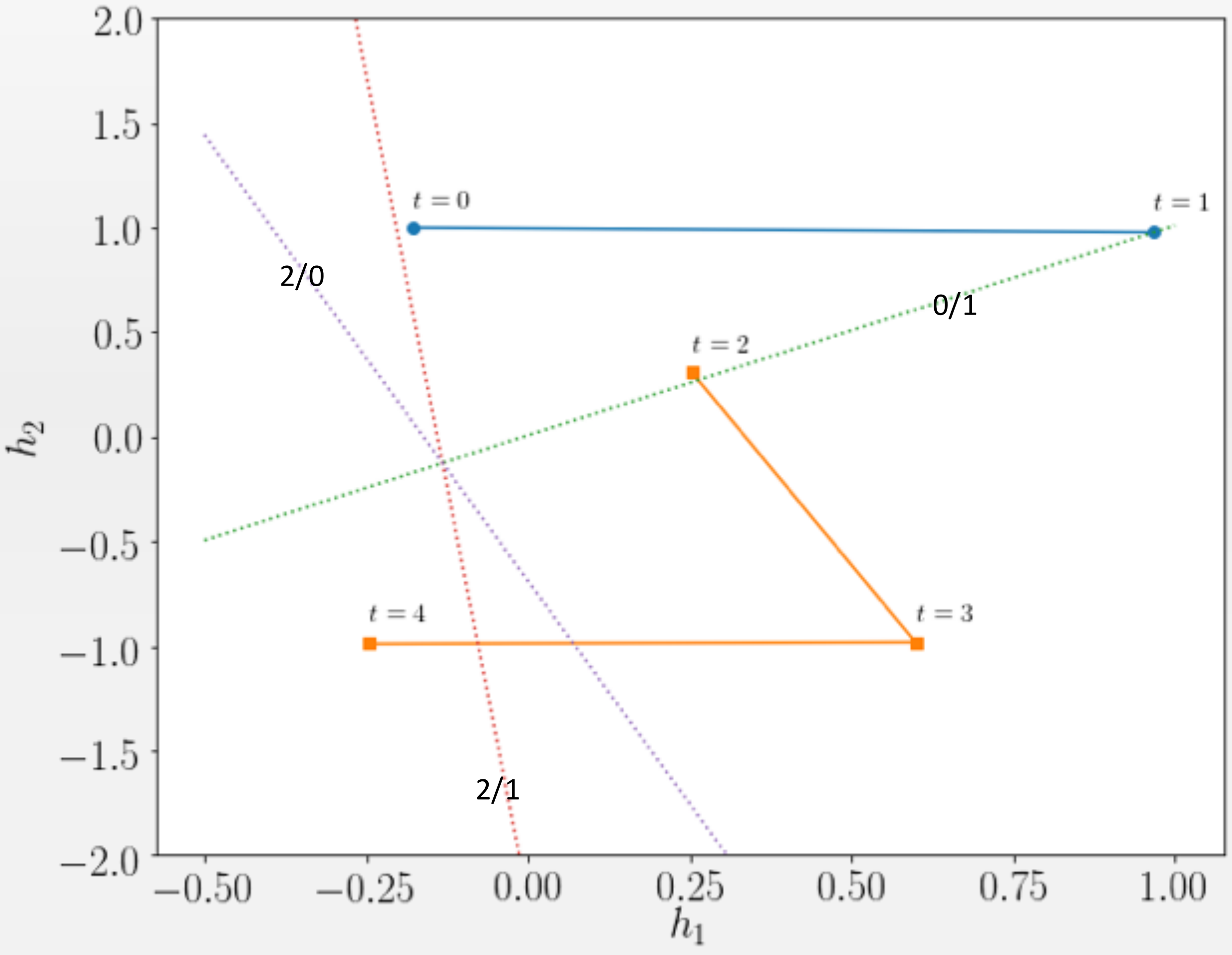
输出层分类器


Log(0 .1)
Log(0.5)
Log(0.4)
LogSoftMax
0
1
2
0
1
2
3
0 1 0 0


2或0或1
0/1
2/0
0
2或0
1
2
2或0或1
2/1
h_1
h_2

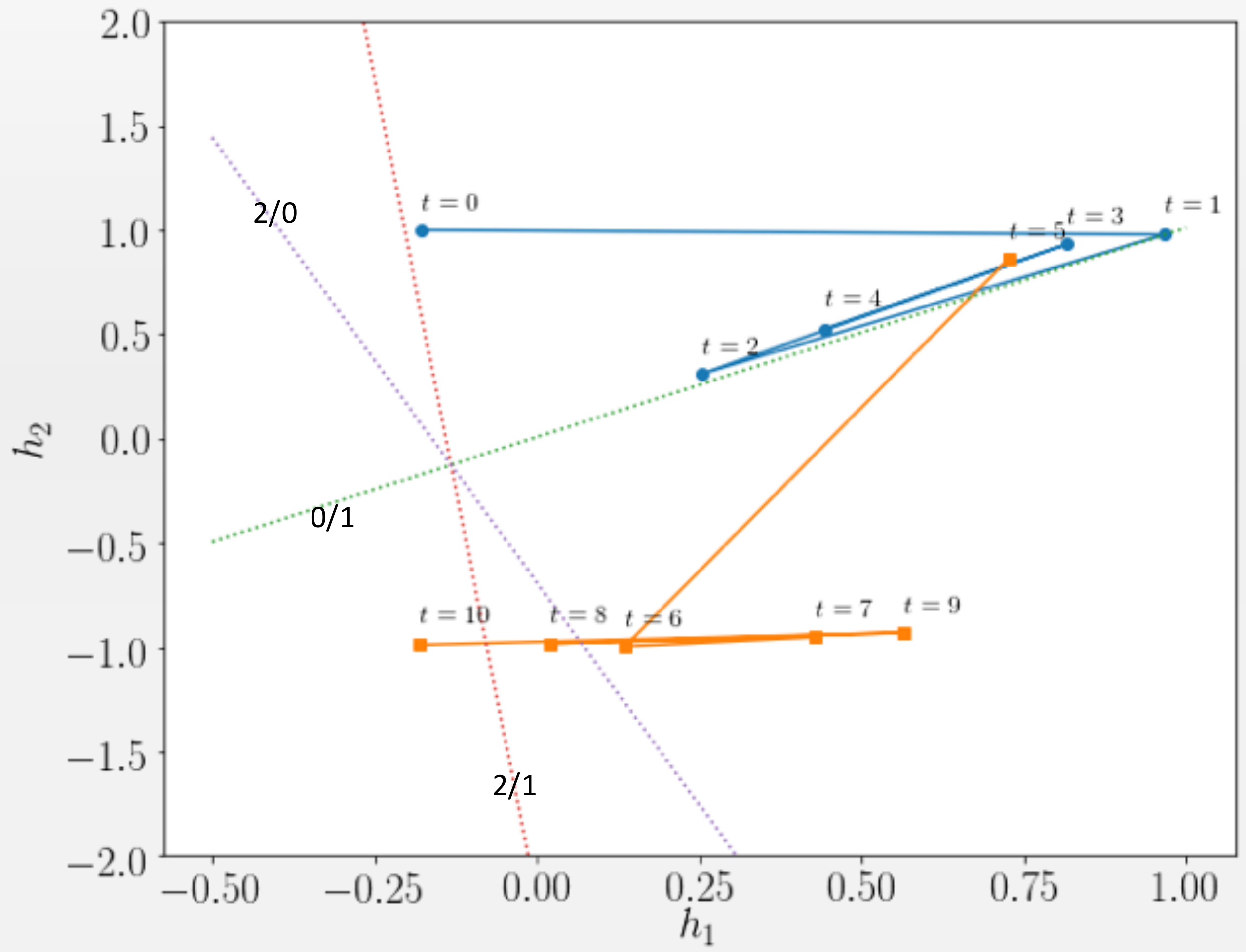
隐含单元分析

Input: 00112, output: 00012



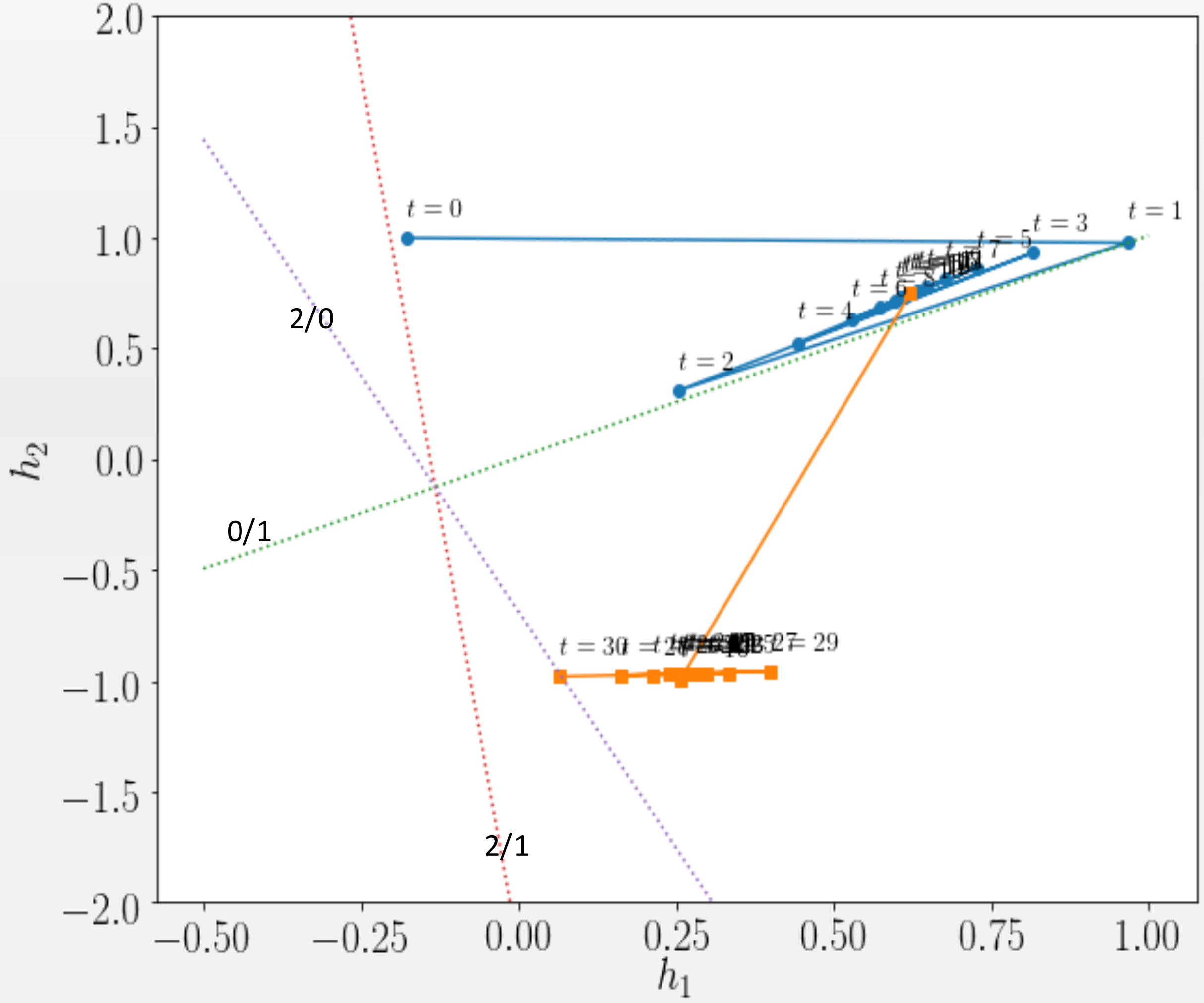
隐含单元分析

Input: 00000111112, output:00000011112

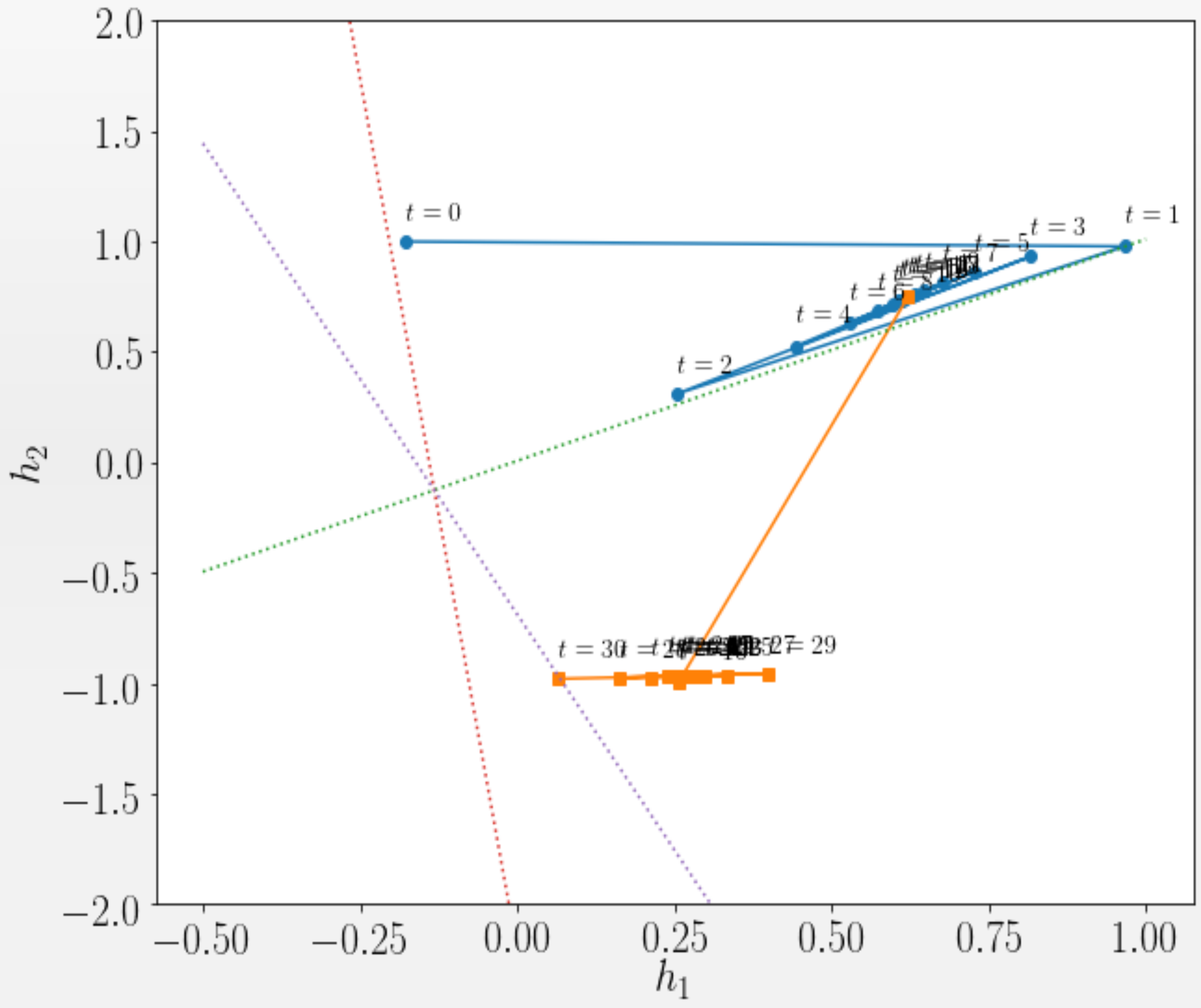


隐含单元分析

Input: 000000000011111111112, output:000000000001111111111



事实上……



RNN实现了两个动力系统:

$$h_{t+1} = f(h_t) = RNN(0, h_t)$$

$$h_{t+1} = g(h_t) = RNN(1, h_t)$$

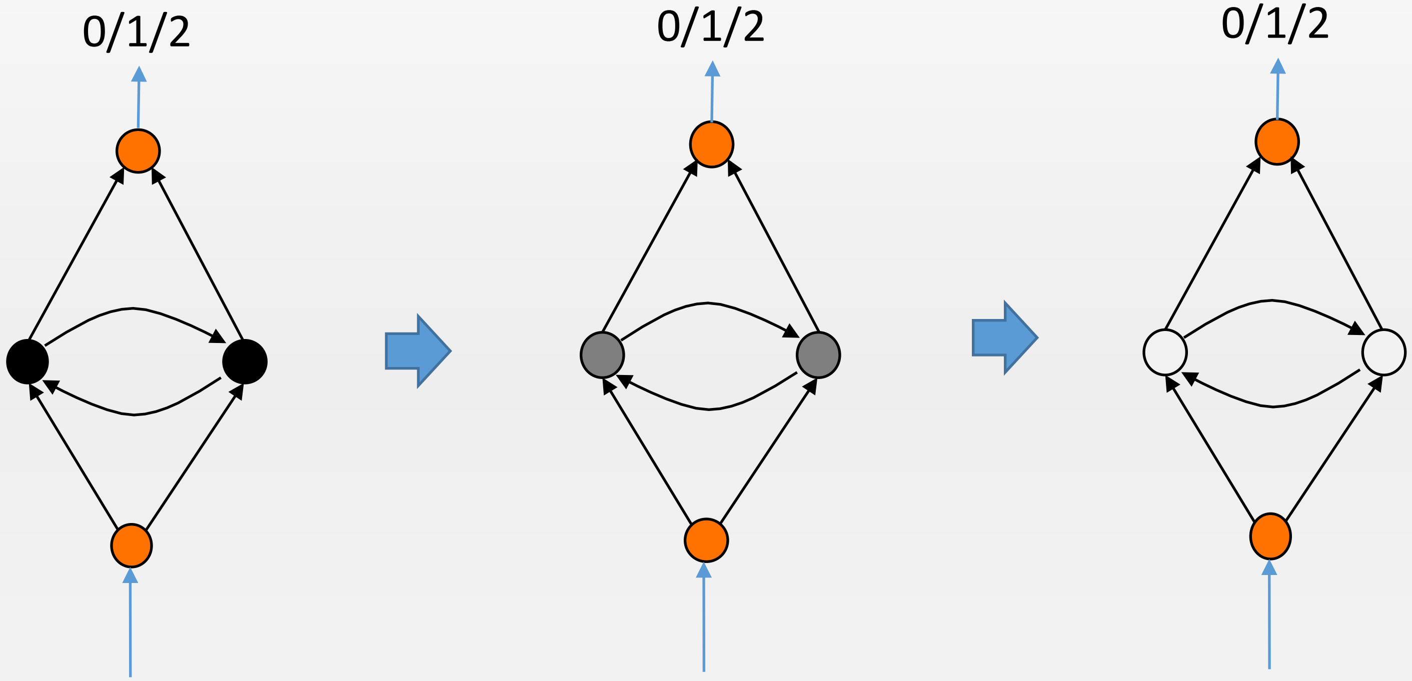
这两个动力系统要求:

- f和g的不动点要能被绿色线分开
- 运动的直线为系统的特征向量方向
- f与g的特征值应该互为倒数

P. Rodriguez et al., A Recurrent Neural Network that Learns to Count, Connection Science, Vol. 11, No1, 1999: 5-40

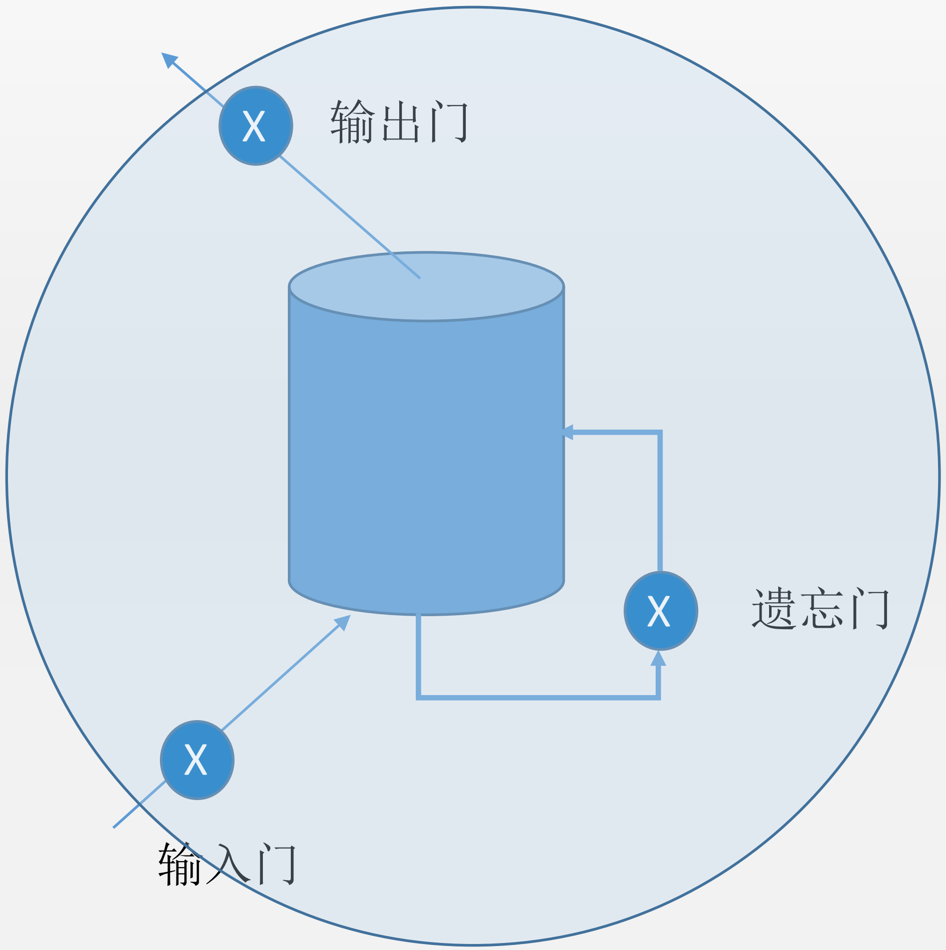
RNN的痛点：无法完成长程记忆

0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

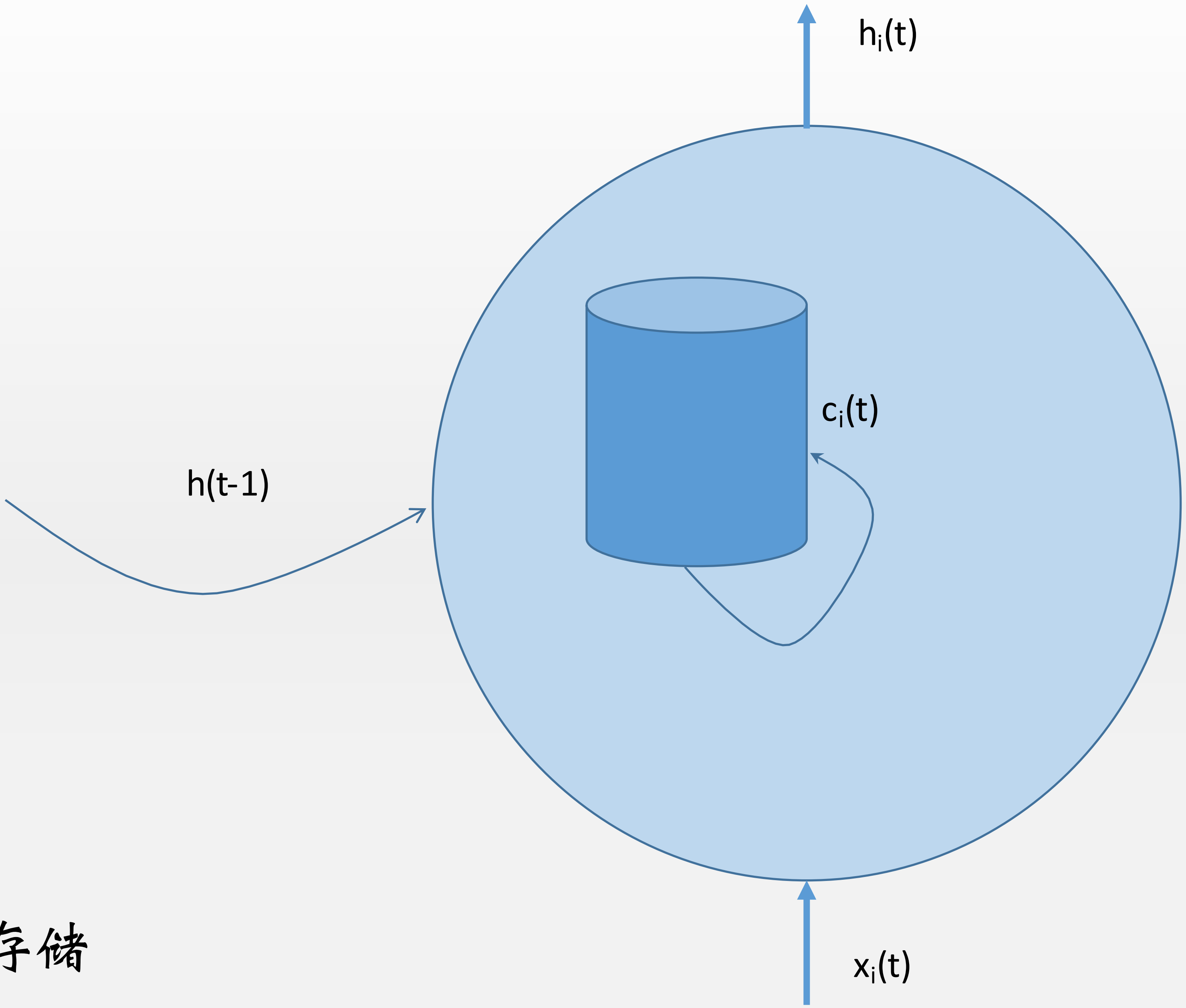


3 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2

让我们尝试用LSTM来做

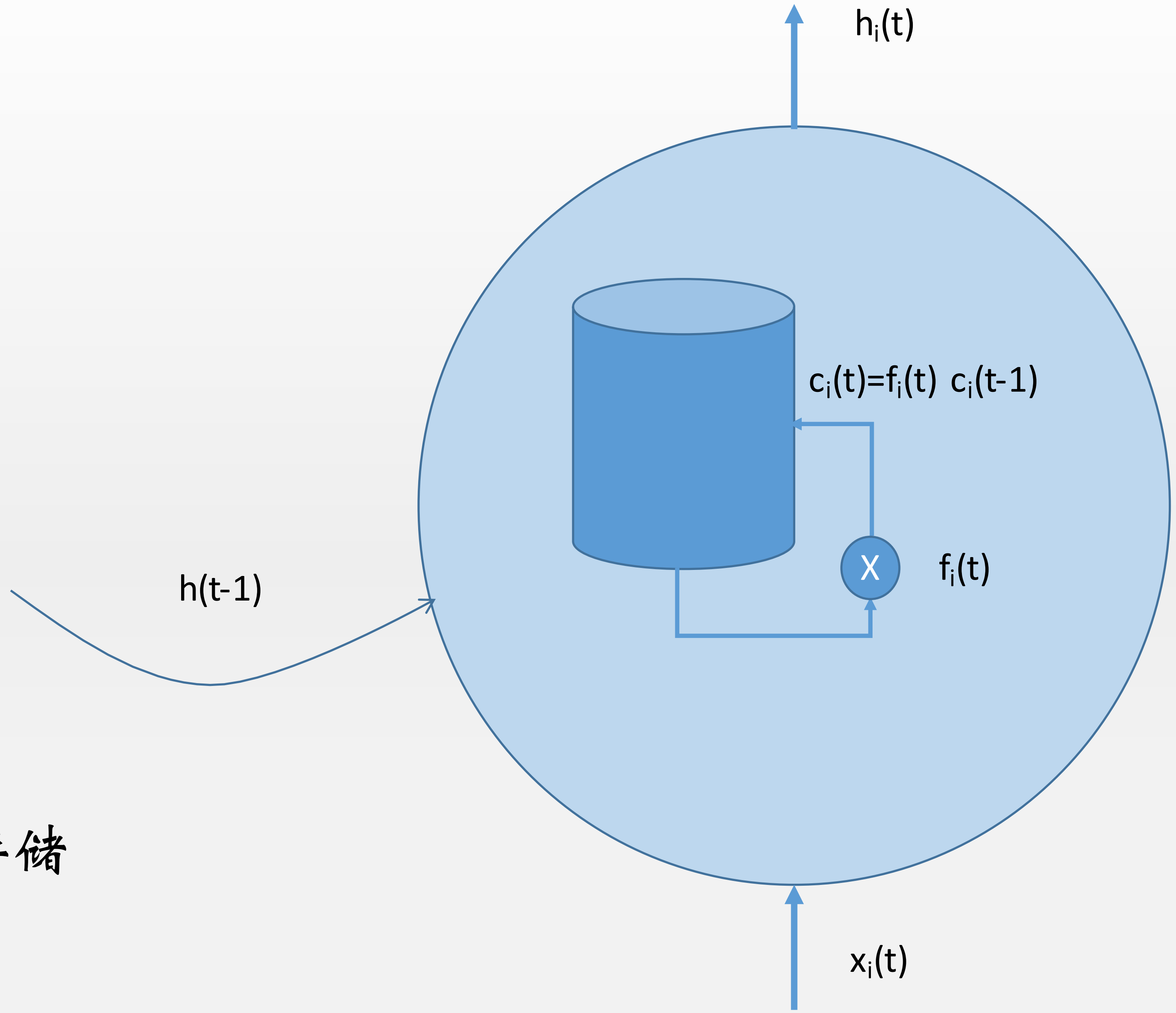


解决方案



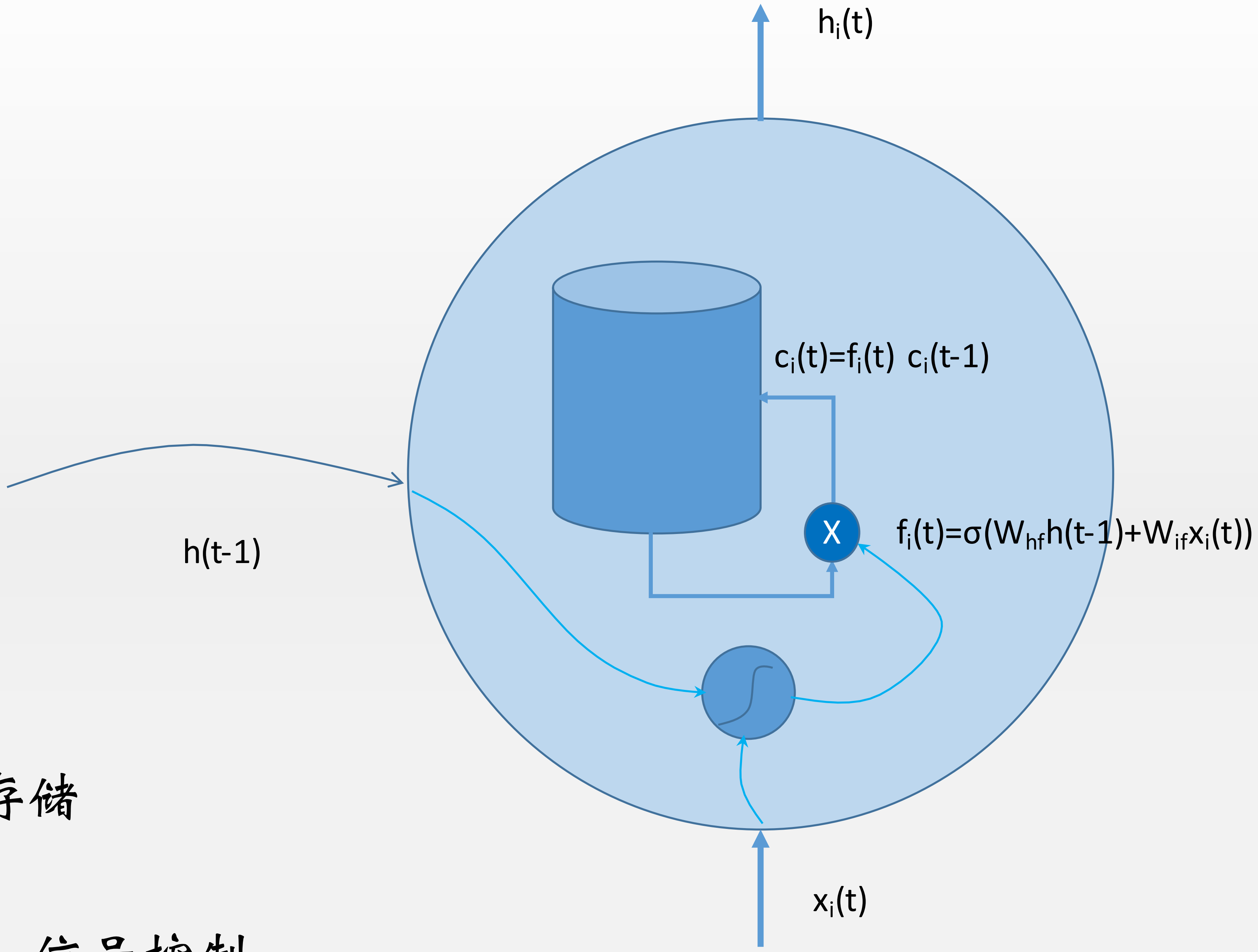
- 加入内部的存储

解决方案



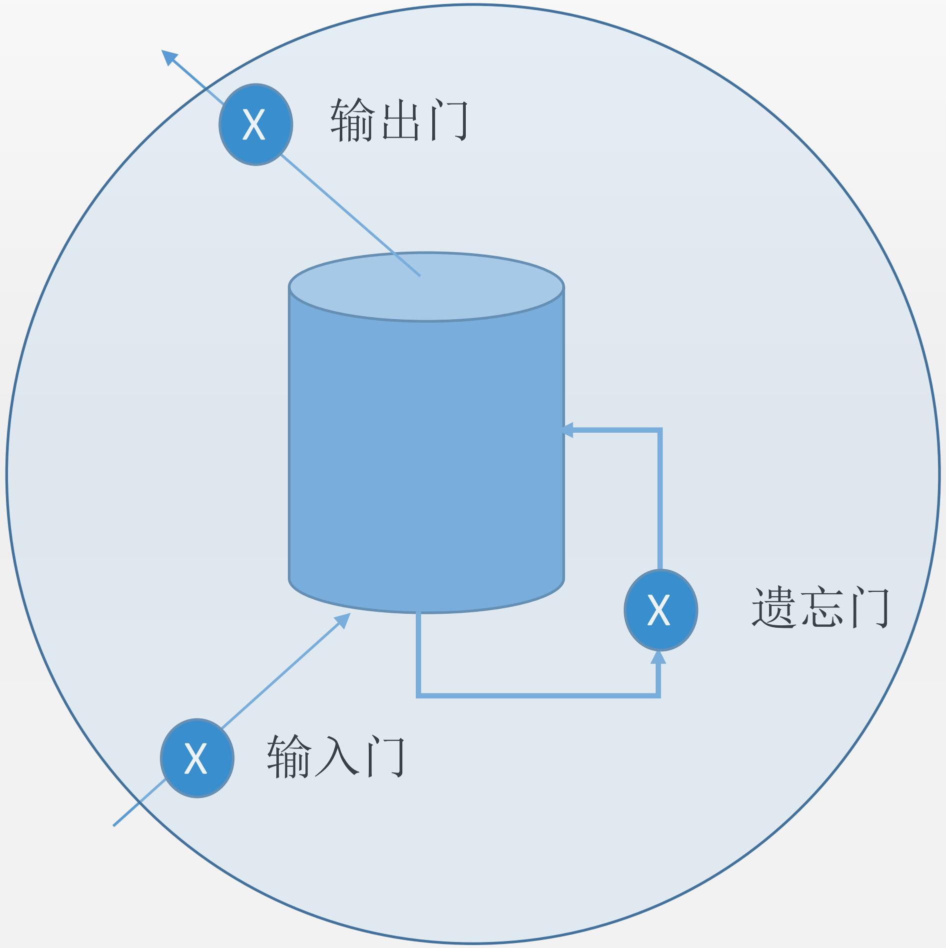
- 加入内部的存储
- 加入遗忘门

解决方案



- 加入内部的存储
- 加入遗忘门
- 遗忘门由输入信号控制

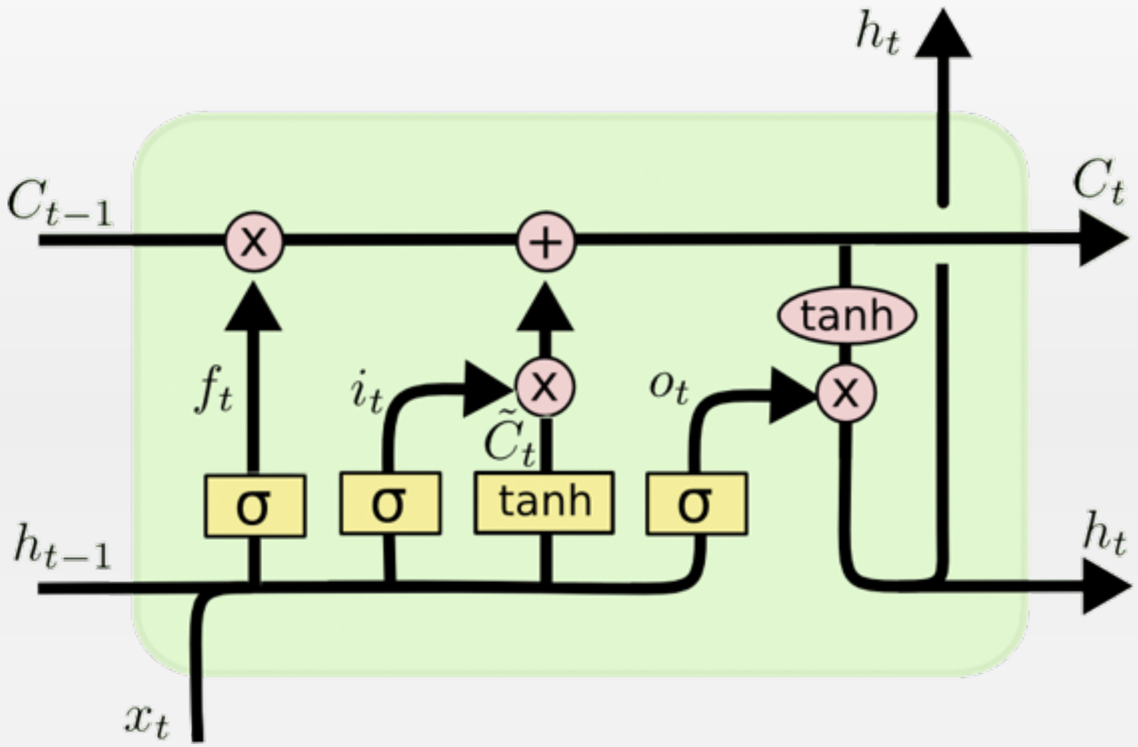
LSTM一个细胞的结构



$$h_{t+1} = o_t * \tanh(c_t)$$
$$c_{t+1} = f_t * c_t + i_t * g_t$$
$$g_t = \tanh(W_{ig}x_t + W_{hc}h_t + b_g)$$

$$i_{t+1} = \sigma(W_{ii}x_t + W_{hi}h_t + b_i)$$
$$f_{t+1} = \sigma(W_{if}x_t + W_{hf}h_t + b_f)$$
$$o_{t+1} = \sigma(W_{io}x_t + W_{ho}h_t + b_o)$$

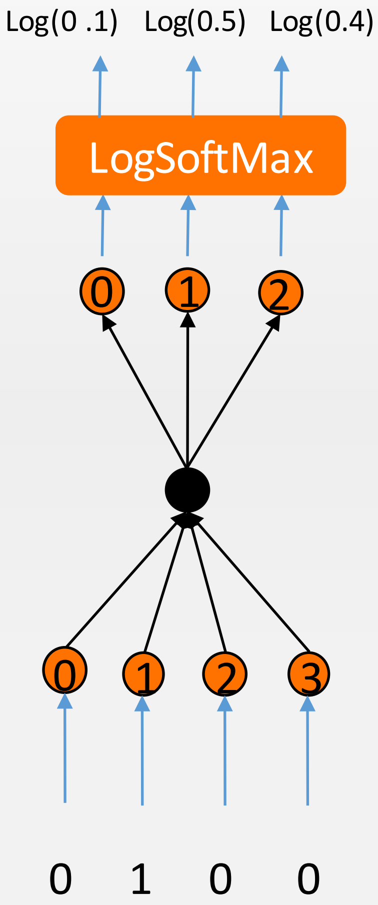
LSTM一个细胞的结构



$$h_{t+1} = o_t * \tanh(c_t)$$
$$c_{t+1} = f_t * c_t + i_t * g_t$$
$$g_t = \tanh(W_{ig}x_t + W_{hc}h_t + b_g)$$

$$i_{t+1} = \sigma(W_{ii}x_t + W_{hi}h_t + b_i)$$
$$f_{t+1} = \sigma(W_{if}x_t + W_{hf}h_t + b_f)$$
$$o_{t+1} = \sigma(W_{io}x_t + W_{ho}h_t + b_o)$$

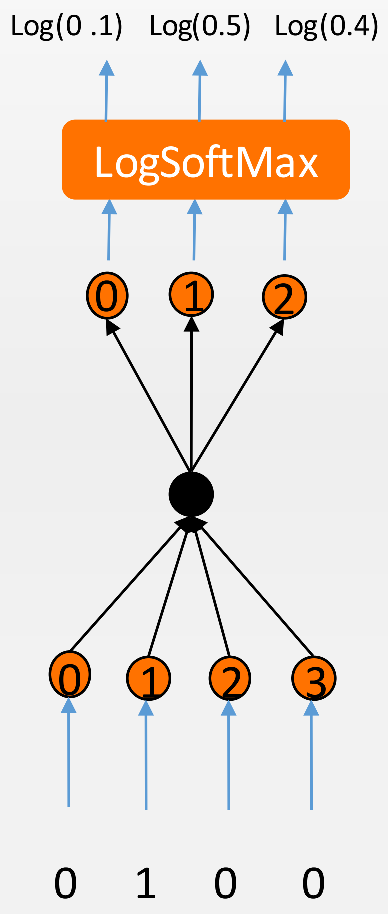
测试效果



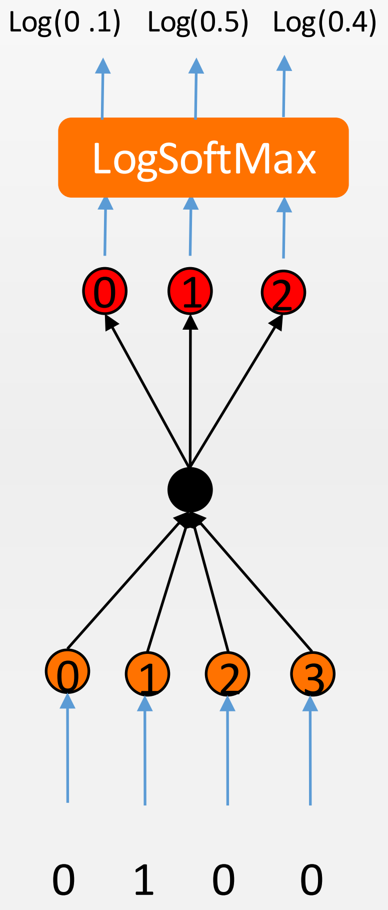
n	目标序列	RNN预测	不同的位数量
0	2	0	1
1	012	002	1
2	00112	00012	1
3	0001112	0000112	1
4	000011112	0000011112	1
……	……	……	……
20	……	……	1
21	……	……	2

N>20以后，开始出现更多的错误，说明1个LSTM单元的记忆容量就已经达到了20

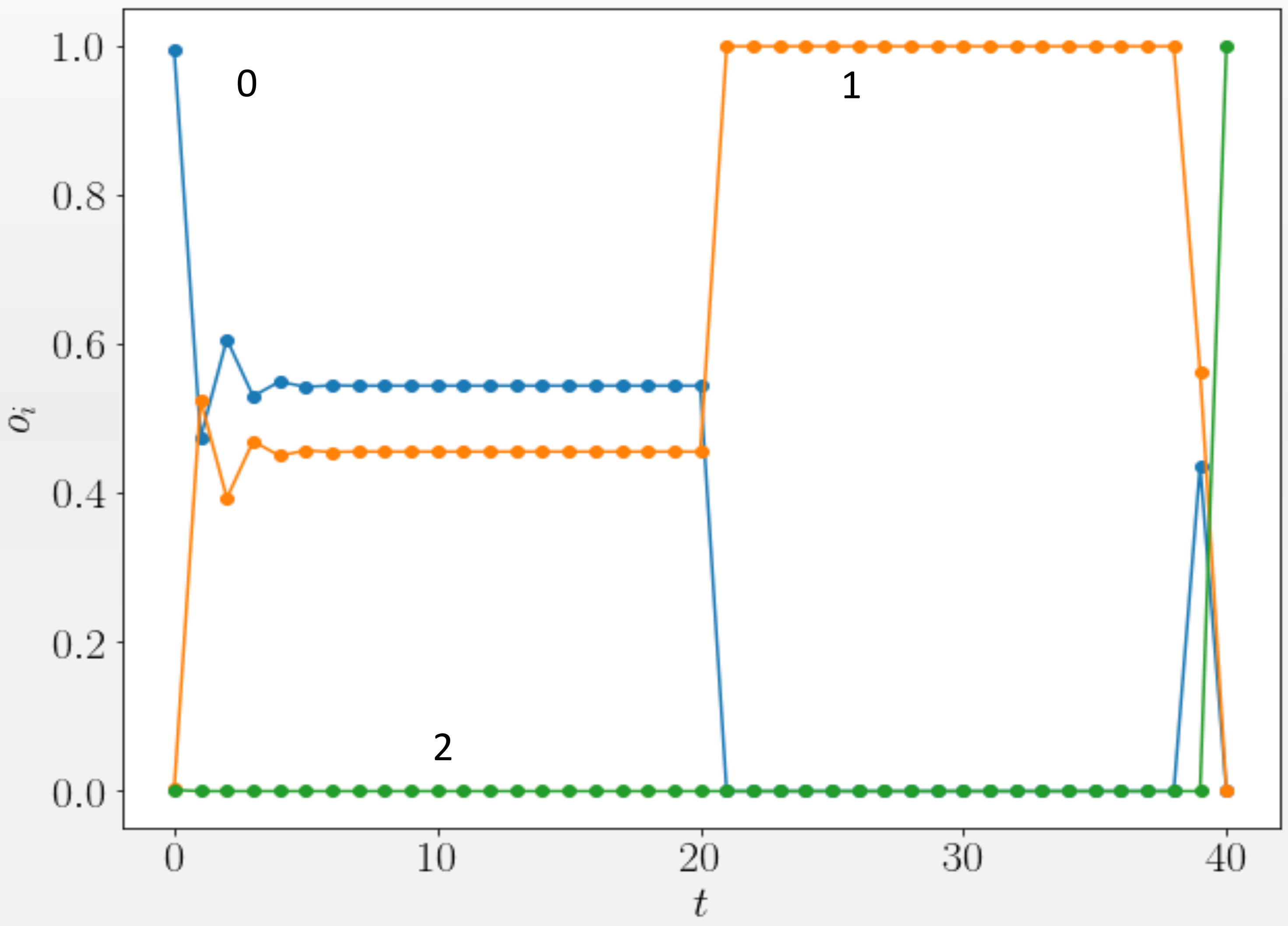
测试效果



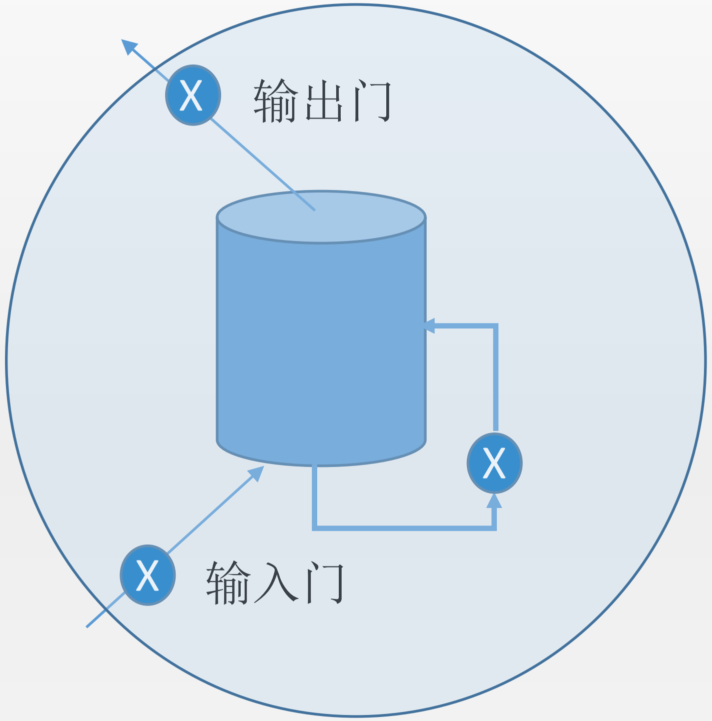
输出层



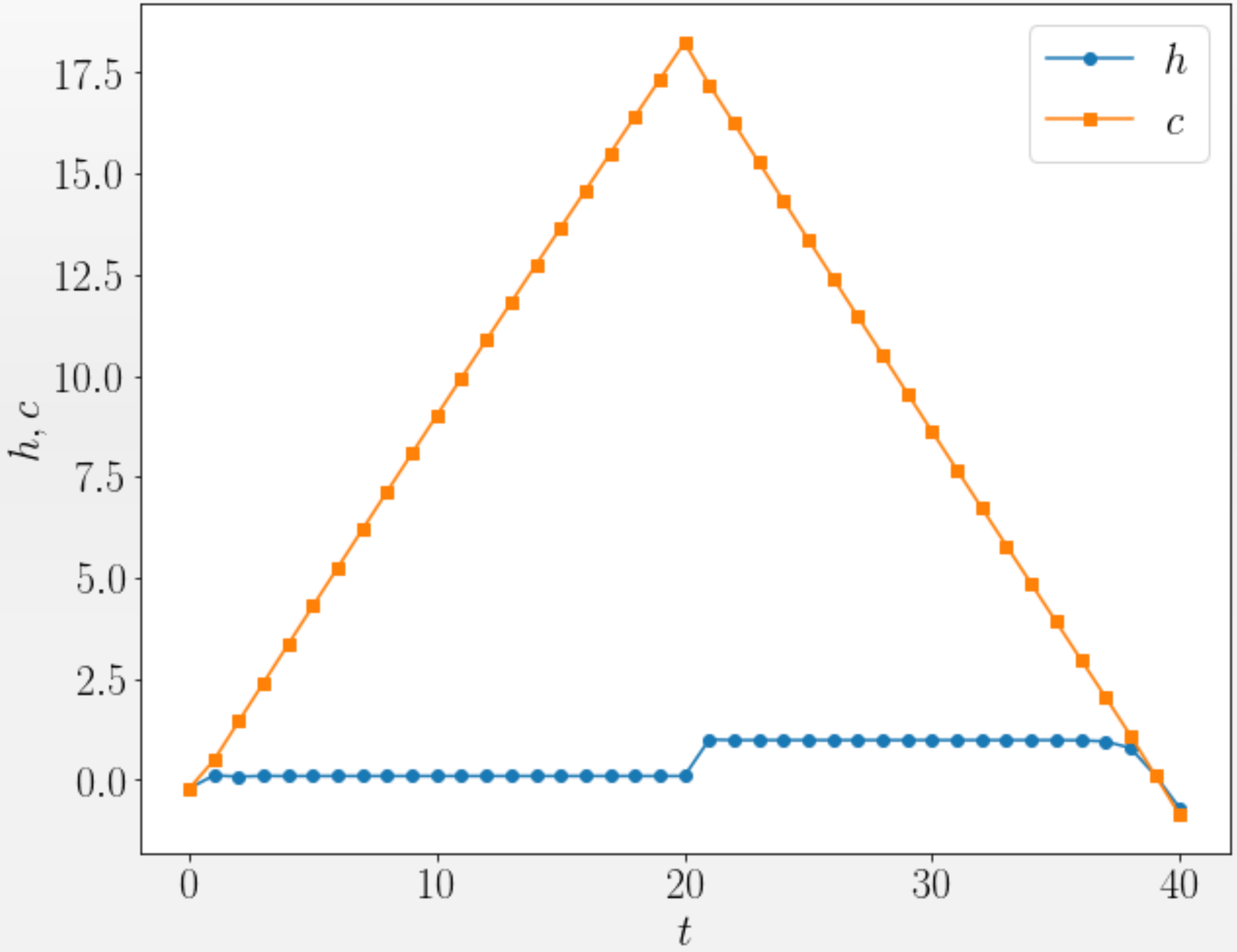
输入： 0^n1^n n＝20



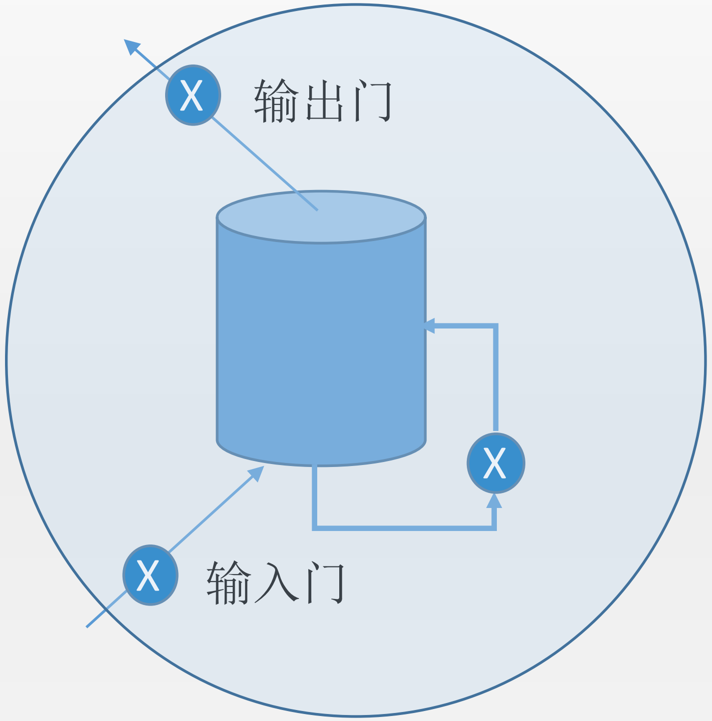
内部状态



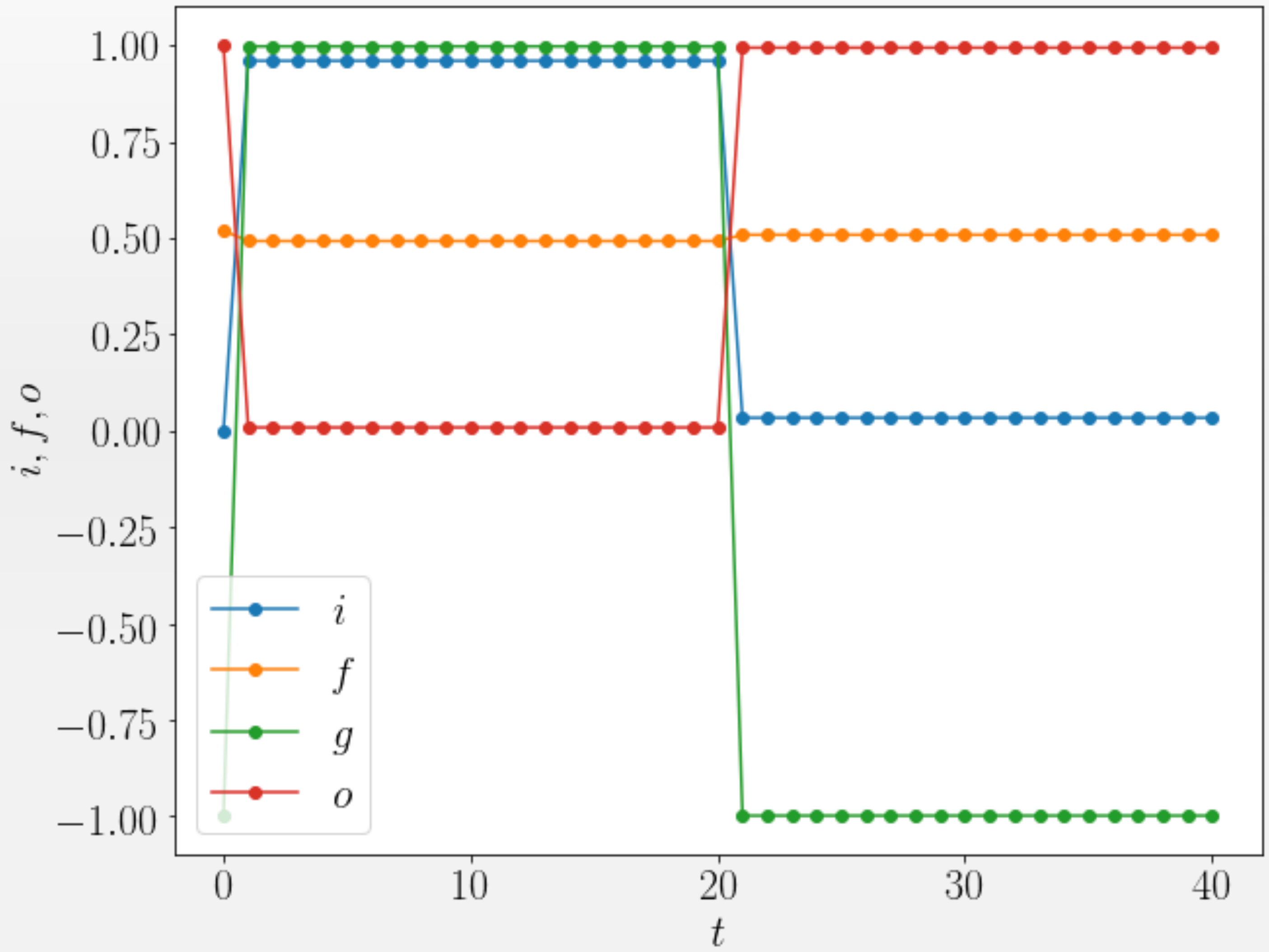
$$h_{t+1} = o_t * \tanh(c_t)$$
$$c_{t+1} = f_t * c_t + i_t * g_t$$
$$g_t = \tanh(W_{ig}x_t + W_{hc}h_t + b_g)$$



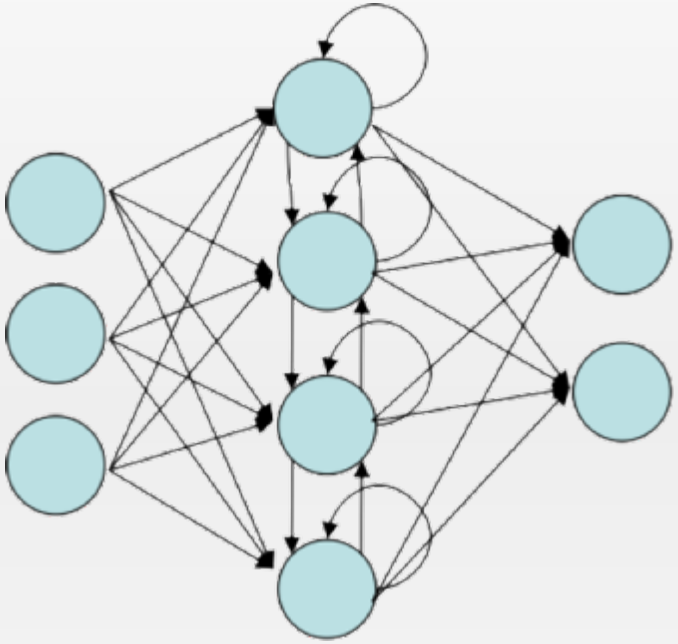
LSTM的各个门



$$i_{t+1} = \sigma(W_{ii}x_t + W_{hi}h_t + b_i)$$
$$f_{t+1} = \sigma(W_{if}x_t + W_{hf}h_t + b_f)$$
$$o_{t+1} = \sigma(W_{io}x_t + W_{ho}h_t + b_o)$$
$$g_{t+1} = \tanh(W_{ig}x_t + W_{hc}h_t + b_g)$$
$$c_{t+1} = f_t * c_t + i_t * g_t$$



神经网络莫扎特



基本步骤

MIDI音乐与MIDO

- 乐器数字接口(Musical Instrument Digital Interface,MIDI) 是20 世纪80 年代初为解决电声乐器之间的通信问题而提出的。MIDI 传输的不是声音信号, 而是音符、控制参数等指令, 它指示MIDI 设备要做什么,怎么做, 如演奏哪个音符、多大音量等。
- MIDO是一个非常方便好用的Python包，可以直接读入midi音乐，也可以输出midi音乐

- 关键的数据结构是一个Message列表：
- [msg1, msg2, msg3,…]
- 每个msg包括：音符（note~[24,102]）、强度（速度[0,128]）与相对上一个音符的时间（time[0,1]）这三组信息

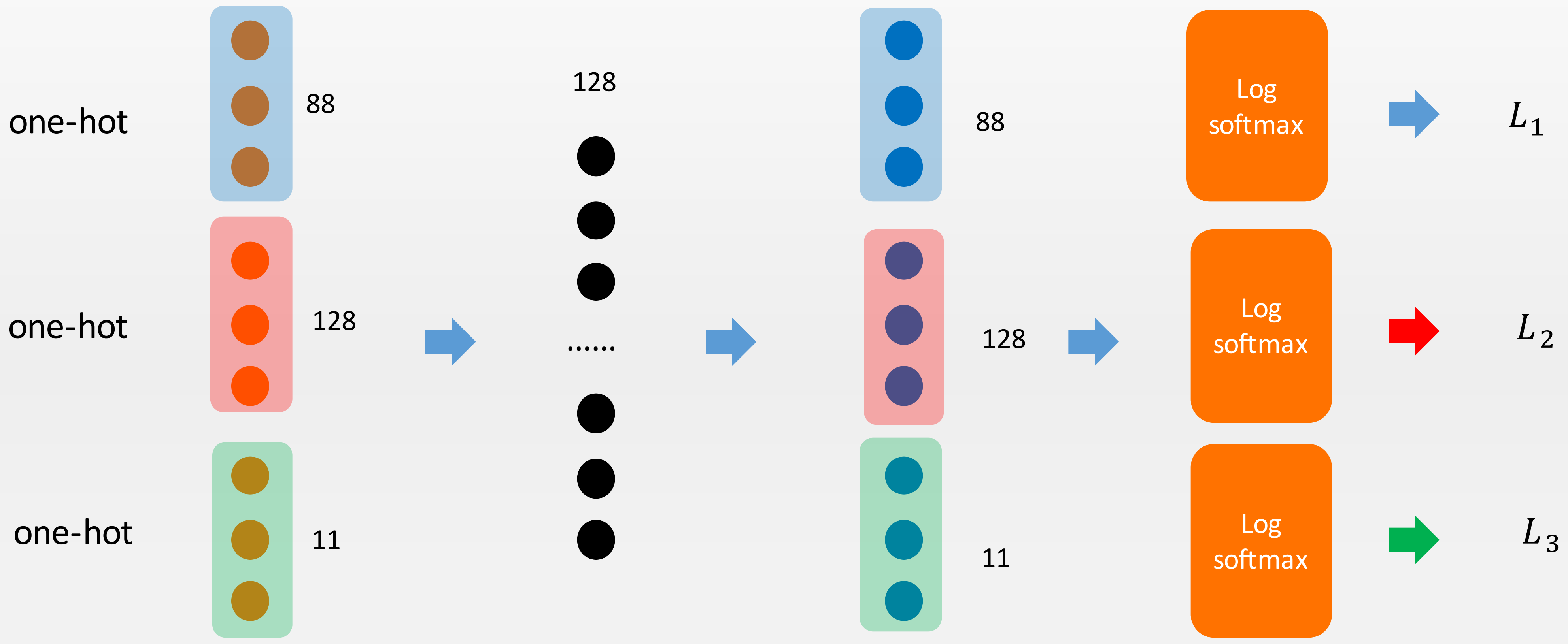
预测问题

78 82 30 …… 72 71 54 43 43 54 56 78 30

120 11 30 …… 33 22 14 126 1 0 0 12 23 0

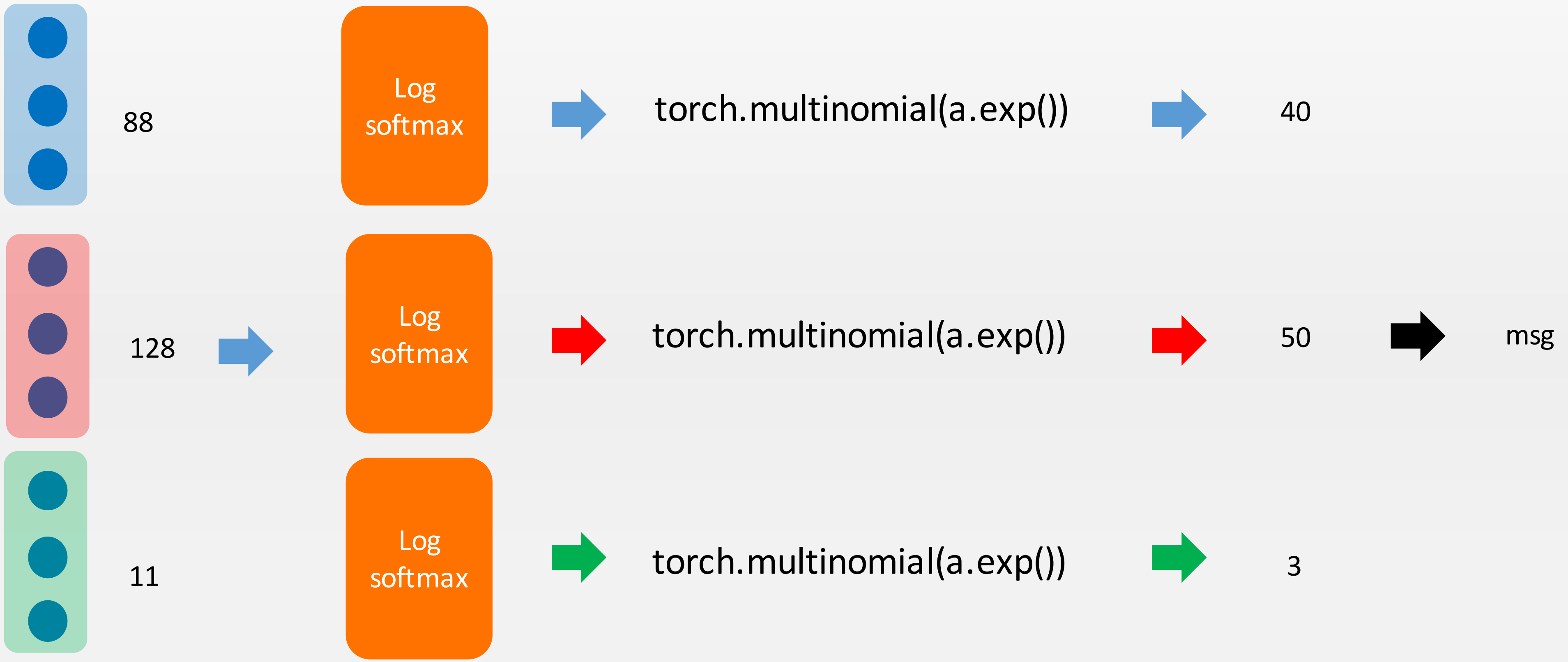
0 1 2 …… 0 3 5 0 1 0 0 3 0 1

阶段I：神经网络架构



$$L = L_1 + L_2 + L_3$$

阶段II: 序列生成


88
128
11
Log softmax
Log softmax
Log softmax
torch.multinomial(a.exp())
torch.multinomial(a.exp())
torch.multinomial(a.exp())
40
50
3
msg

今日重点

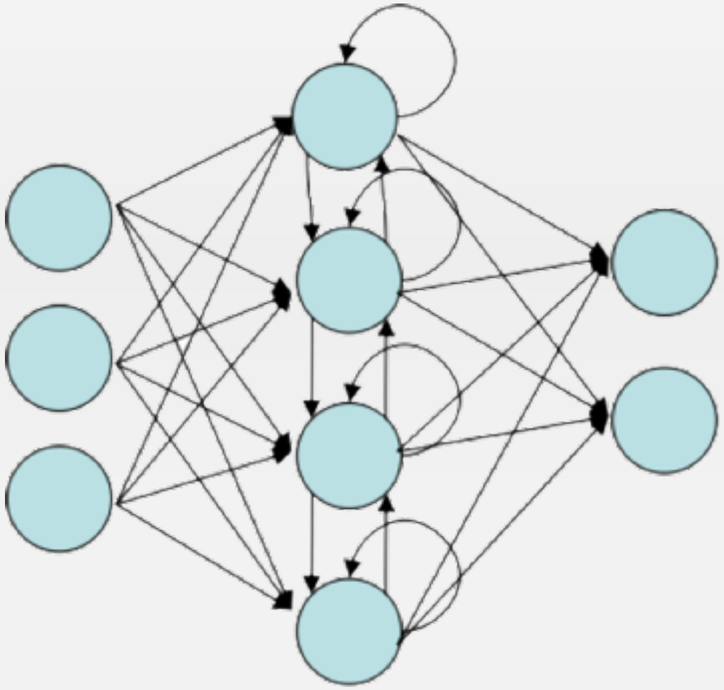
- 如何用神经网络做序列生成？
- RNN与LSTM的工作原理
- RNN是如何记忆Pattern的？
- MIDI音乐的原理
- 如何用LSTM作曲

作业： 名称生成器

- 请设计一个LSTM网络，要求输入一个国别，输出一个这个国别的名称

中国

美国



Zhang Jiang

Jake

敬请期待

彩云小译

- 张江

集智AI学园
campus.swarma.org