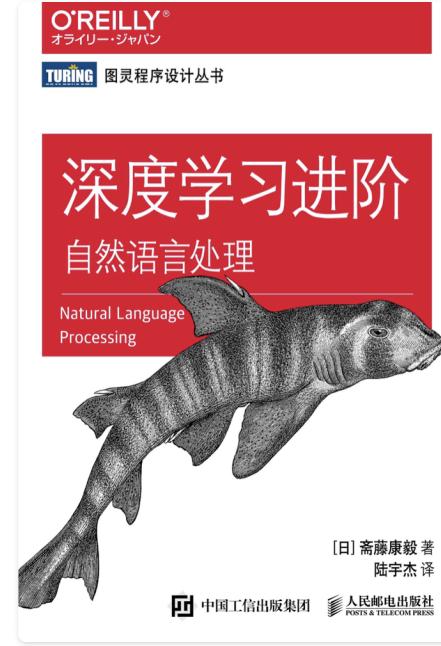


Deep Learning: Natural Language Processing

bifnudozhao@tencent.com

Outline

- 自然语言与单词的分布式表示
 - 基于计数的方法
- word2vec
 - 基于推理的方法
 - 神经网络中单词的处理方法
 - CBOW 模型
 - Embedding 层
- RNN
 - 语言模型
 - RNN 的实现
- Gated RNN
 - RNN 的问题
 - LSTM 的结构
- 基于 RNN 生成文本

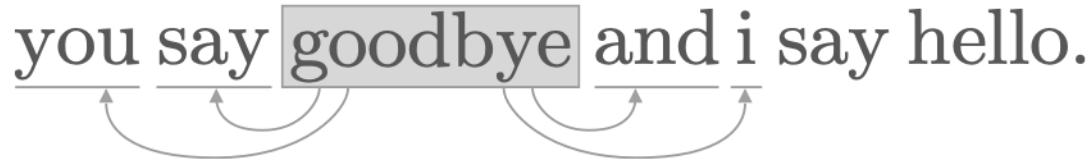


深度学习进阶-自然语言处理

自然语言与单词的分布式表示

基于计数的方法

分布式假设（distributional hypothesis）是指，某个单词的含义是由它周围的单词形成。单词本身没含义，是由它所在的上下文形成。



上面是一个窗口大小为 2 的上下文例子，其中 `you`，`say`，`and`，`i` 是 `goodbye` 的上下文。对于计算机来说，`goodbye` 本身是没有任何意思的，这个词的“意义”由其上下文进行定义。

自然语言与单词的分布式表示

共现矩阵

使用向量来表示单词最直接的方式是对周围的词的数量进行计数。这种方法称为“基于计数的方法”。比如上面的例子，假设窗口为 1，则可以得到如下的共现矩阵 (*co-occurrence matrix*)。

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

自然语言与单词的分布式表示

向量间的相似度

余弦相似度 (cosine similarity) 是比较常用的相似度算法。设有两个向量 \mathbf{x}, \mathbf{y} ，其相似度计算如下

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}| |\mathbf{y}|} = \frac{\sum_{i=0}^n x_i y_i}{\sqrt{\sum_{i=0}^n x_i^2} \sqrt{\sum_{i=0}^n y_i^2}}$$

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

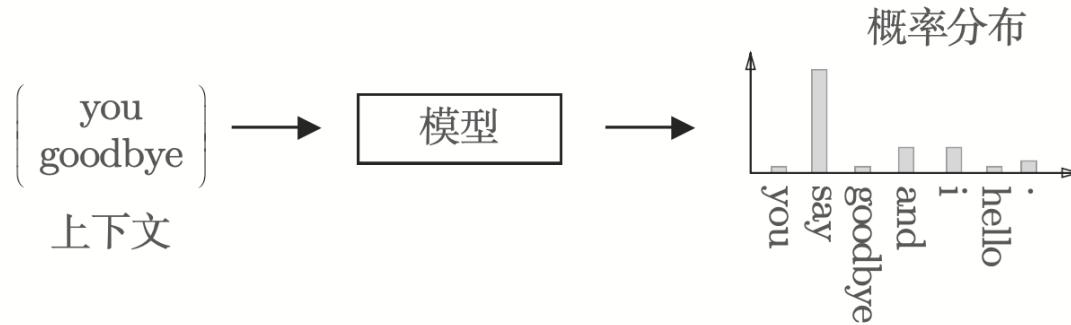
word2vec

基于推理的方法

基于推理的方法的主要操作是“推理”。

you ? goodbye and i say hello.

问题转变为“给出单词`you`以及`goodbye`，预测问号处是什么单词”。



基于推理的方法:输入上下文，模型输出各个单词的出现概率。

word2vec

神经网络中单词的处理方法

要用神经网络来处理单词，就需要将单词转化为 one-hot 的表示。回忆一下，神经网络内部运算是矩阵运算，而矩阵就是一组向量，向量的元素是实数。如果想用神经网络处理单词，就必须将单词转化为向量。

单词	单词ID	one-hot表示
$\begin{pmatrix} \text{you} \\ \text{goodbye} \end{pmatrix}$	$\begin{pmatrix} 0 \\ 2 \end{pmatrix}$	$\begin{pmatrix} (1, 0, 0, 0, 0, 0, 0) \\ (0, 0, 1, 0, 0, 0, 0) \end{pmatrix}$

单词、单词 ID 以及它们的 one-hot 表示如上图所示。

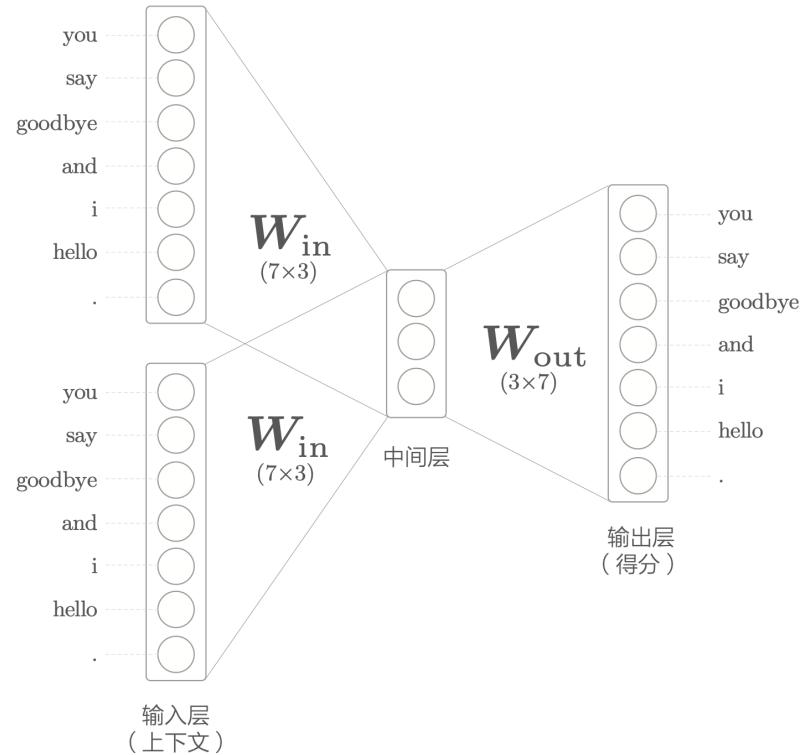
word2vec

CBOW 模型的推理

原版的 word2vec 提出了名为 “continuous bag-of-words (CBOW) ” 的模型作为神经网络。CBOW 的输入是上下文，假设上下文用 ['you', 'goodbye'] 两个单词，将其转换为 one-hot 表示，如果有 N 个单词，则输入层有 N 个。

其中，中间层是输入层各个层做完变换之后得到的平均值。输出层是各个单词的得分，它的值越大，则说明对应单词的出现概率就越高。

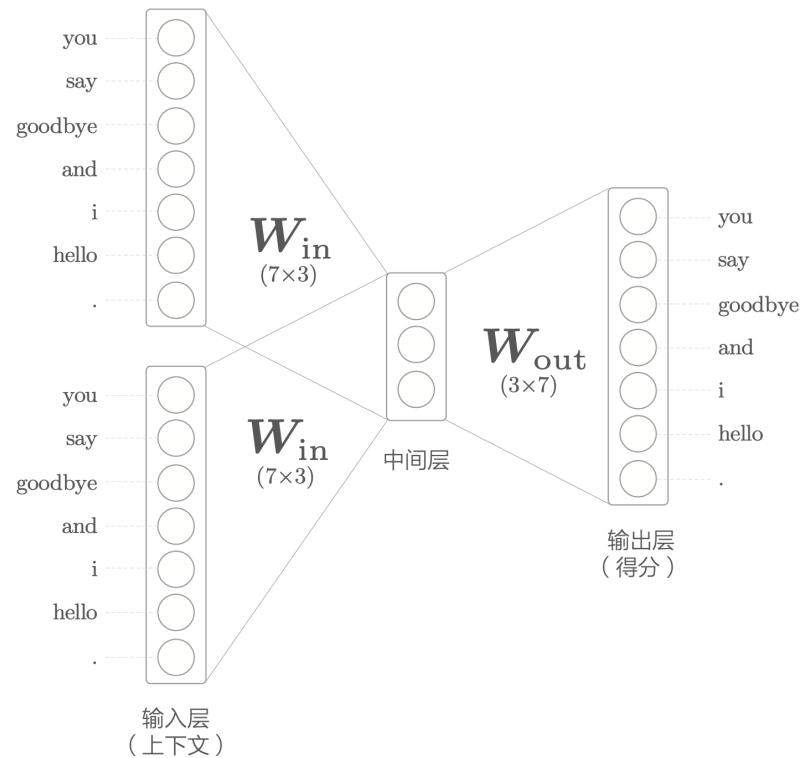
中间层的神经元数量比输入层少这一点很重要。中间层需要将预测单词所需的信息压缩保存，从而产生密集的向量表示。这时，中间层被写入了我们人类无法解读的代码，这相当于“编码”工作。而从中间层的信息获得期望结果的过程则称为“解码”。这一过程将被编码的信息复原为我们可以理解的形式。



word2vec

CBOW 模型的推理

```
1  you = [1, 0, 0, 0, 0, 0, 0]
2  goodbye = [0, 0, 1, 0, 0, 0, 0]
3
4  W_in_1 = np.random.rand(7, 3)
5  W_in_2 = np.random.rand(7, 3)
6
7  hidden = (you * W_in_1 + goodbye * W_in_2) / 2
8
9  W_out = np.random.rand(3, 7)
10
11 output = hidden * W_out
12 # output = [
13 #   0.2343, -> you
14 #   0.92343, -> say
15 #   0.1873, -> goodbye
16 #   0.023, -> and
17 #   0.3271, -> i
18 #   0.1937, -> hello
19 #   0.073 -> .
20 # ]
```



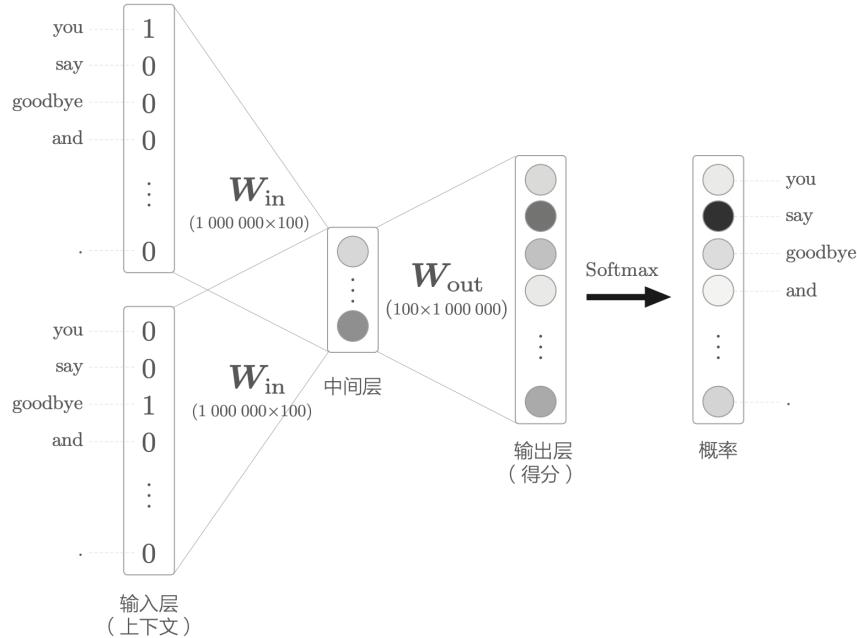
word2vec

CBOW 模型的瓶颈

当词汇量数量非常庞大，比如 100 万个，而神经元有 100 个的时候，word2vec 的处理过程如右图所示。

计算瓶颈会出现在

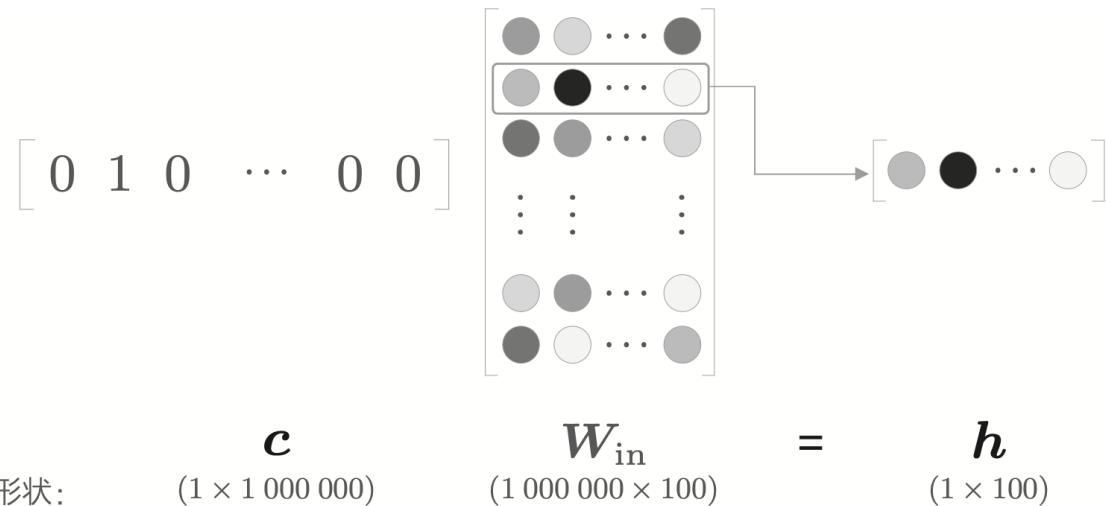
- 输入层的 one-hot 表示和权重矩阵 \mathbf{W}_{in} 的乘积。
- 中间层和权重矩阵 \mathbf{W}_{out} 的乘积。
- Softmax 层的计算。



word2vec

Embedding 层

由于单词转化为 one-hot 表示，并且输入了 MatMul 层，在计算乘积的时候，实际上只是从 \mathbf{W}_{in} 里提取出了 one-hot 对应的那一行权重而已，所以这个不需要真的进行矩阵乘积，而只需要创建一个从权重参数中抽取“单词 ID 对应行”的层即可，这里成为 Embedding 层。



word2vec

Embedding 层

```
1  you = [1, 0, 0, 0, 0, 0, 0]
2  goodbye = [0, 0, 1, 0, 0, 0, 0]
3
4  # W_in_1 = np.random.rand(7, 3)
5  # W_in_2 = np.random.rand(7, 3)
6  #
7  # hidden = (you * W_in_1 + goodbye * W_in_2) / 2
8  W_in = np.random.rand(7, 3)
9  you_idx = embedding.get_index(you)
10 goodbye_idx = embedding.get_index(goodbye)
11 hidden[you_idx, goodbye_idx] = W_in[you_idx, goodbye_idx]
12
13 W_out = np.random.rand(3, 7)
14
15 output = hidden * W_out
```

如前所述，通过 embedding 层，我们可以将巨大的矩阵乘法变为简单的行向量提取。上面的代码例子中，我们只需要初始化一个输入权重矩阵 `W_in`，然后通过 embedding 层获取 `you` 以及 `goodbye` 的下标，直接从 `W_in` 里提取出对应行，直接放入 `hidden` 的对应行中。从而大大减少了计算。

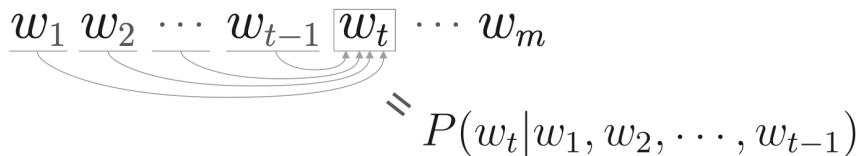
RNN

语言模型

语言模型 (language model) 给出了单词序列发生的概率。考虑由 m 个单词 w_1, \dots, w_m 构成的句子，将单词按 w_1, \dots, w_m 的顺序出现的概率记为 $P(w_1, \dots, w_m)$ 。使用后验概率可以将这个联合概率分解为

$$p(w_1, \dots, w_m) = \prod_{t=1}^m P(w_t | w_1, \dots, w_{t-1})$$

这个模型有时候也称为条件语言模型 (*conditional language model*)。



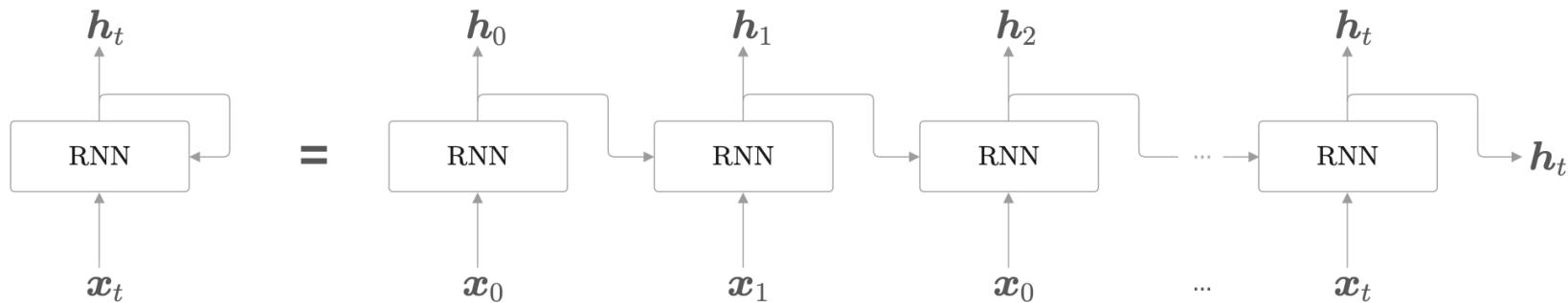
语言模型中的后验概率:若以第 t 个单词为目标词，则第 t 个单词左侧的全部单词构成上下文(条件)。

RNN

循环神经网络

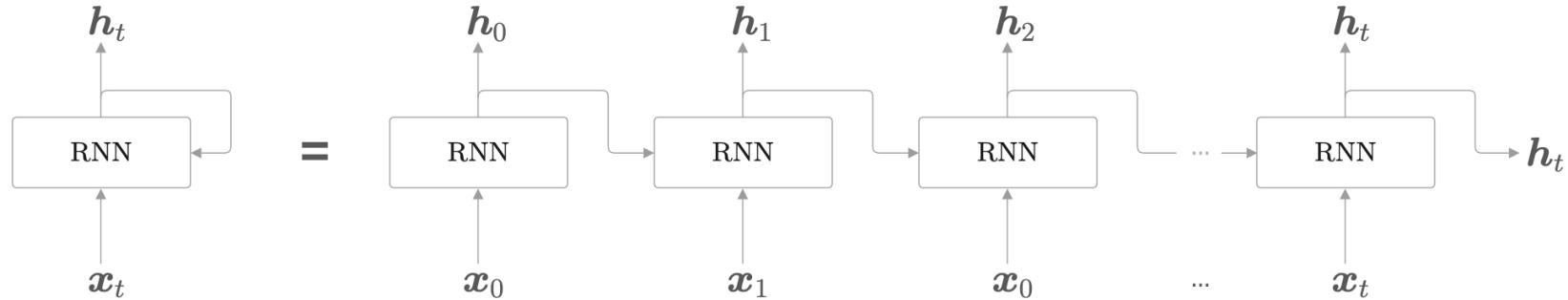
RNN (Recurrent Neural Network) 为循环神经网络。与 CBOW 的最大区别在于，CBOW 会受到上下文大小的限制，如果窗口大小无法覆盖上下文，将无法进行正确的推理。但是通过 RNN 可以做到这点。

CBOW 的窗口是双侧的，并且它的每次预测都是独立的，也就是上下文无关的，当上下文比较长的时候，很难一次过喂给 CBOW。而 RNN 将上下文改为单侧的，并且 RNN 内部有记忆能力，所以理论上可以持续地为 RNN 输入上下文。



RNN

循环神经网络



将循环层展开。虽然展开为多个独立的单元，但是这些单元组成一个整体的层。上面的计算可以表示为

$$\mathbf{h}_t = \tanh(\mathbf{h}_{t-1}\mathbf{W}_h + \mathbf{x}_t\mathbf{W}_x + \mathbf{b})$$

其中， \mathbf{W}_x 将输入 \mathbf{x} 转化为输出 \mathbf{h} ，而 \mathbf{W}_h 将前一个 RNN 层的输出转化为当前时刻的输出。RNN 的 \mathbf{h} 存储状态，时间每前进一步，它就以上式被更新。有时候这个状态被称为隐藏状态（hidden state）。

RNN

```
1  class RNN:  
2      def __init__(self, hidden_size, dimension):  
3          self.W_h = np.random.rand(hidden_size, dimension)  
4          self.W_x = np.random.rand(dimension, hidden_size)  
5          self.prev_h = np.ones(hidden_size)  
6  
7      def forward(self, x_t):  
8          h_t = np.tanh(self.prev_h * self.W_h + x_t * self.W_x)  
9          self.prev_h = h_t  
10         return h_t
```

$$\mathbf{h}_t = \tanh(\mathbf{h}_{t-1} \mathbf{W}_h + \mathbf{x}_t \mathbf{W}_x + \mathbf{b})$$

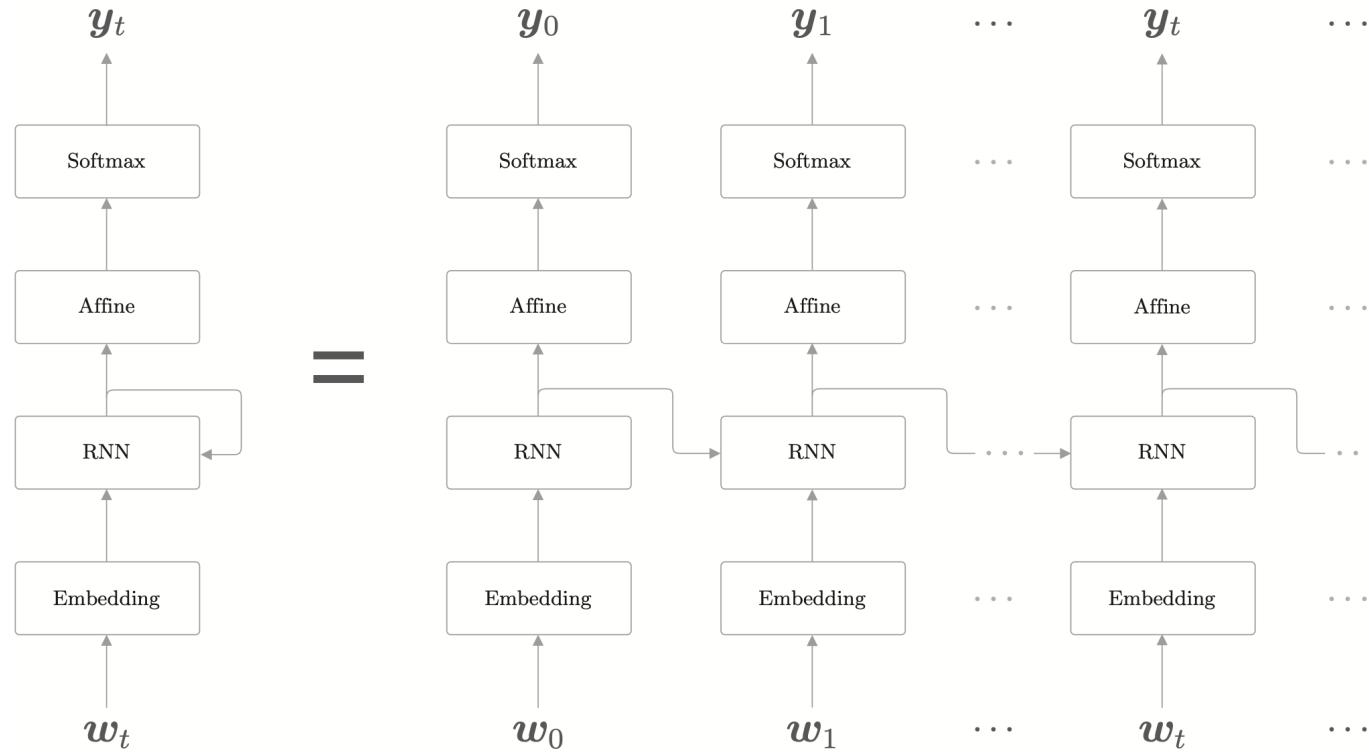
初始化的时候随机两个参数矩阵。前馈网络计算的时候，直接翻译 RNN 的计算公式，以及将当前的 `h_t` 保存下来，作为下一次前馈计算的隐藏参数。

```
1  h_t = np.ones(hidden_size)  
2  for x_t in sequence_data:  
3      h_t = RNN.forward(x_t, h_t)
```

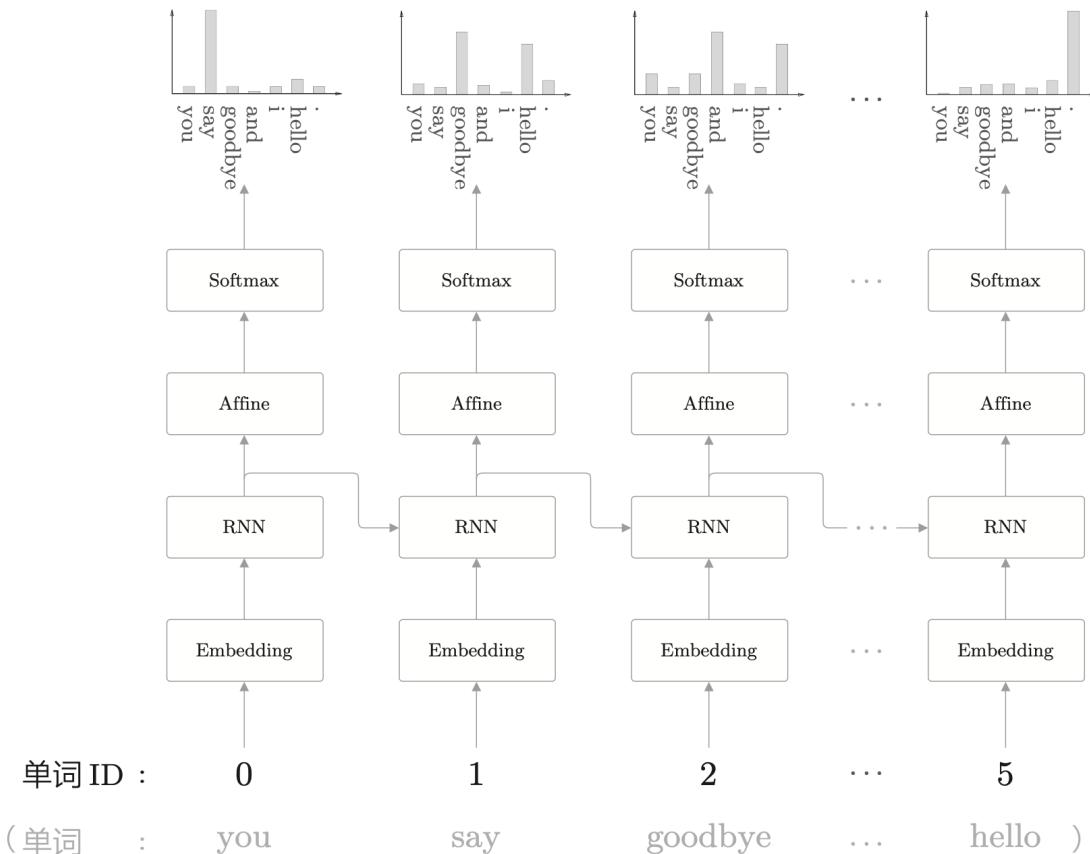
使用 RNN 层的时候，只需要一直输入数据即可。

RNN

RNN 语言模型全貌



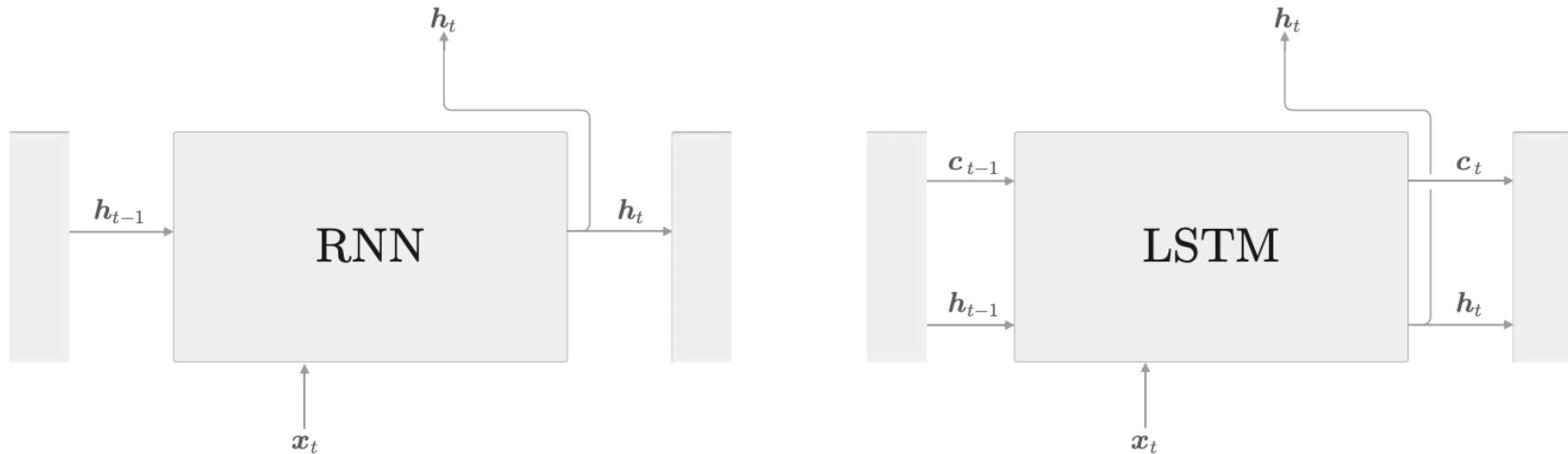
RNN



Gated RNN

LSTM

LSTM (Long short-term memory) 在 RNN 的基础上进行了改进，下面是 LSTM 与 RNN 的接口比较图。



LSTM 增加了路径 c 。这个 c 称为记忆单元，它只在 LSTM 内部发挥作用，并不向外层输出。

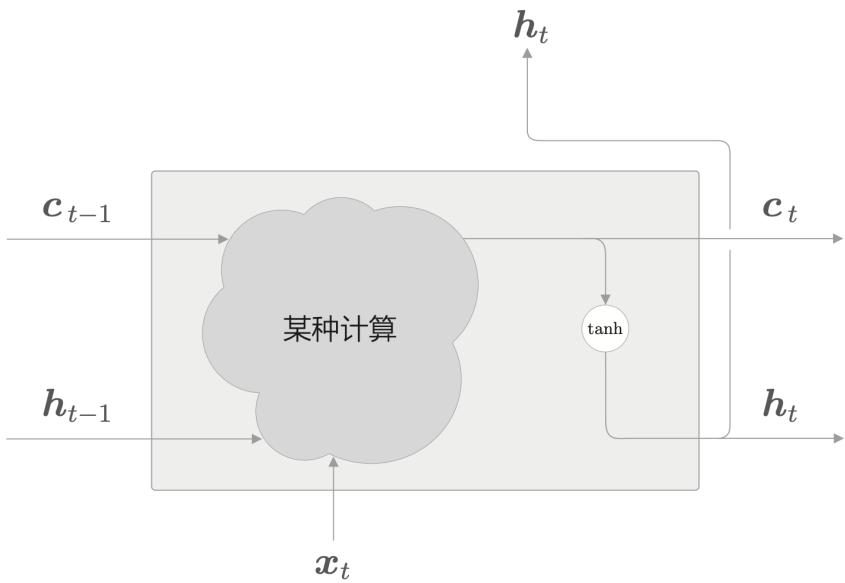
Gated RNN

LSTM 的结构

LSTM 有记忆单元 c_t 。这个 c_t 存储了时刻 t 时的 LSTM 的记忆。然后，基于这个充满必要信息的记忆，向外部的层（和下一时刻的 LSTM）输出隐藏状态 h_t 。

LSTM 有记忆单元 c_t 。这个 c_t 存储了时刻 t 时的 LSTM 的记忆。然后，基于这个充满必要信息的记忆，向外部的层（和下一时刻的 LSTM）输出隐藏状态 h_t 。

- 输出门：控制输出量
- 遗忘门：控制之前的记忆需要忘记的量
- 新的记忆单元：控制新输入需要记忆的量
- 输入门：控制输入的影响程度



```
1 h_t, c_t = np.tanh(  
2     some_calculation(x_t, h_prev, c_prev))
```

Gated RNN

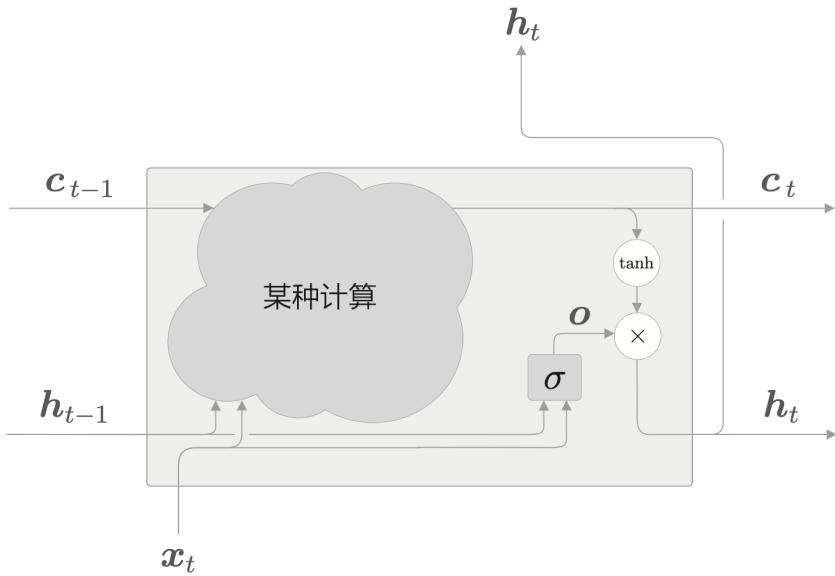
LSTM 的结构 - 输出门

下面考虑对 $\tanh(c_t)$ 施加门。这作用在于针对 $\tanh(c_t)$ 的各个元素，调整它们作为下一时刻的隐藏状态的重要程度。由于这个门管理下一个隐藏状态 h_t 的输出，所以称为输出门（output gate）。

输出门的开合程度（流出比例）根据输入 x_t 和上一个状态 h_{t-1} 求出。sigmoid 函数用 σ 表示。下面是输出门的输出计算方式。

$$o = \sigma(x_t W_x^{(o)} + h_{t-1} W_h^{(o)} + b^{(o)})$$

最后，得到输出 $h_t = o \odot \tanh(c_t)$ 。这里的乘积 \odot 是对应元素的乘积。



```
1  output = np.sigmoid(  
2      x_t * W_x_output +  
3      h_prev * W_h_output +  
4      b_output  
5  )
```

Gated RNN

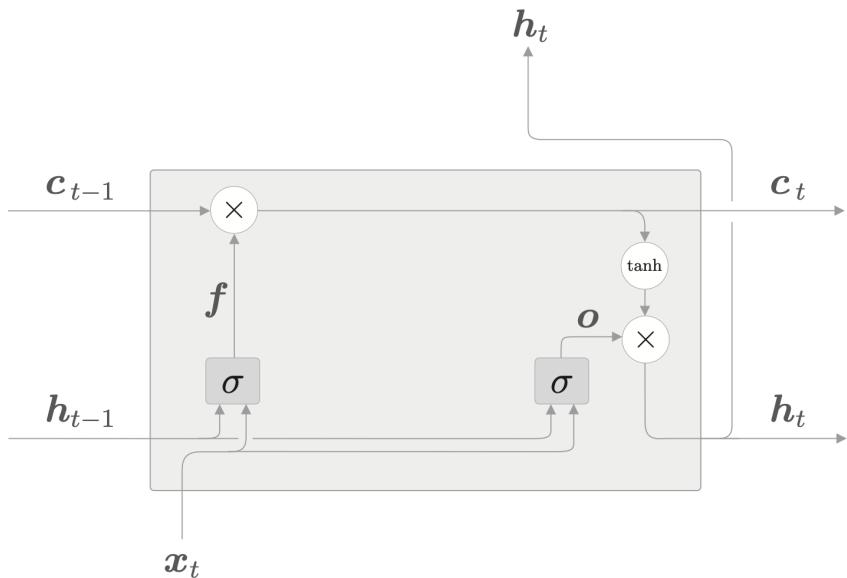
LSTM 的结构 - 遗忘门

现在，我们在 c_{t-1} 上添加一个忘记不必要记忆的门，称为遗忘门（forget gate）。

遗忘门的计算方式与输出门的计算方式类似，最终也是算出一个权重，来决定 c_{t-1} 的重要程度。

$$f = \sigma(x_t W_x^{(f)} + h_{t-1} W_h^{(f)} + b^{(f)})$$

最后，得到输出 $c_t = f \odot c_{t-1}$ 。



```
1  forget = np.sigmoid(
2      x_t * W_x_forget +
3      h_prev * W_h_forget +
4      b_forget
5  )
6
7  c_t = forget * c_prev
```

Gated RNN

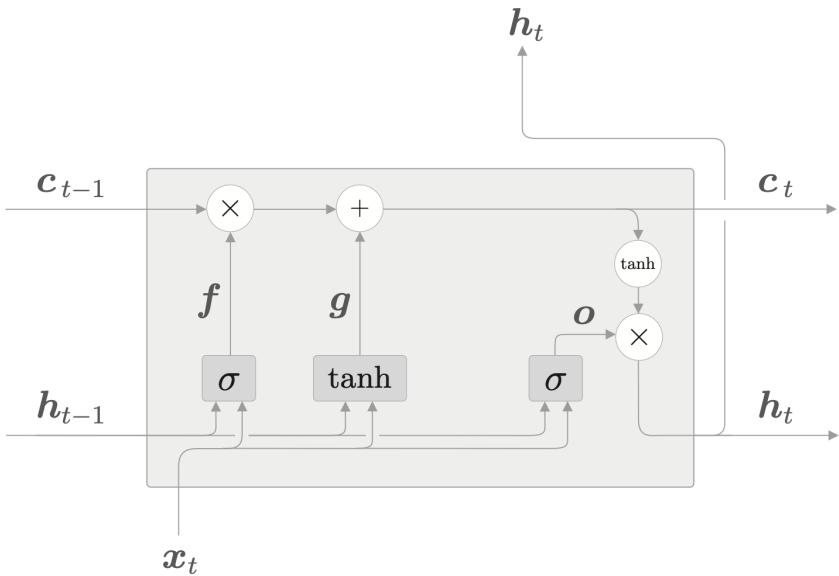
LSTM 的结构 - 新的记忆单元

遗忘门从上一时刻的记忆单元删除了应该忘记的东西，下面我们往这个记忆单元增加一些应当记住的新信息。

如上图所示，基于 \tanh 节点计算出的结果被加到上一时刻的记忆单元 c_{t-1} 上，这样一来，新的信息就被加到记忆单元中。这个节点的作用不是门，所以不用 sigmoid 函数。

$$g = \tanh(x_t W_x^{(g)} + h_{t-1} W_h^{(g)} + b^{(g)})$$

通过将这个 g 加到 c_{t-1} 上，从而形成新的记忆。



```
1 gain = np.tanh(  
2     x_t * W_x_gain +  
3     h_prev * W_h_gain +  
4     b_gain  
5 )
```

Gated RNN

LSTM 的结构 - 输入门

输入门用于控制新增信息的权重，是用来控制 g 的。

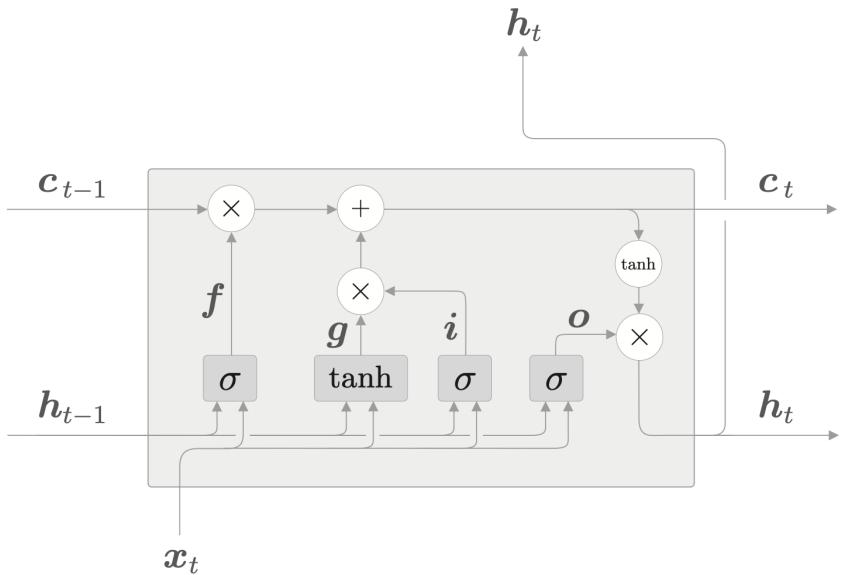
由于这是一个控制门，所以使用 σ 函数。

$$i = \sigma(x_t W_x^{(i)} + h_{t-1} W_h^{(i)} + b^{(i)})$$

由此，最终得到的 c_t, h_t 可以表示为

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

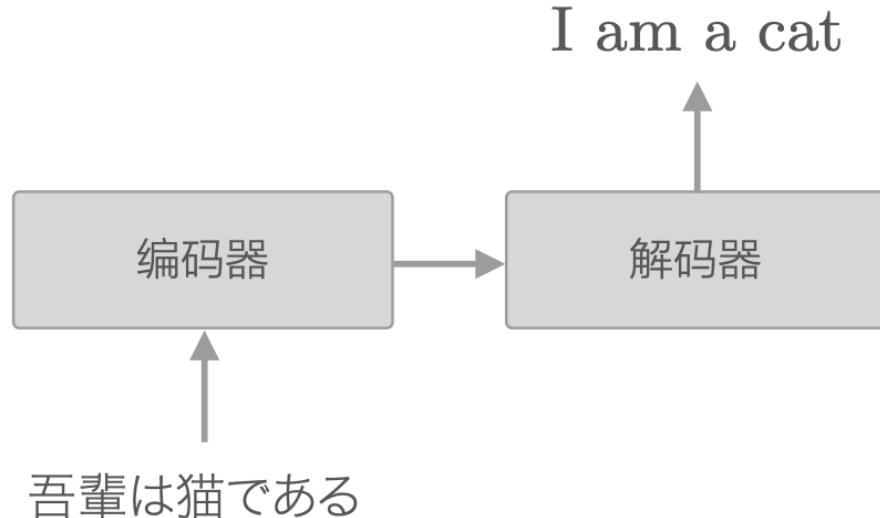


```
1  input = np.tanh(
2      x_t * W_x_input +
3      h_prev * W_h_input +
4      b_input
5  )
6
7  c_t = forget * c_prev + input * gain
8  h_t = output * np.tanh(c_t)
```

基于 RNN 生成文本

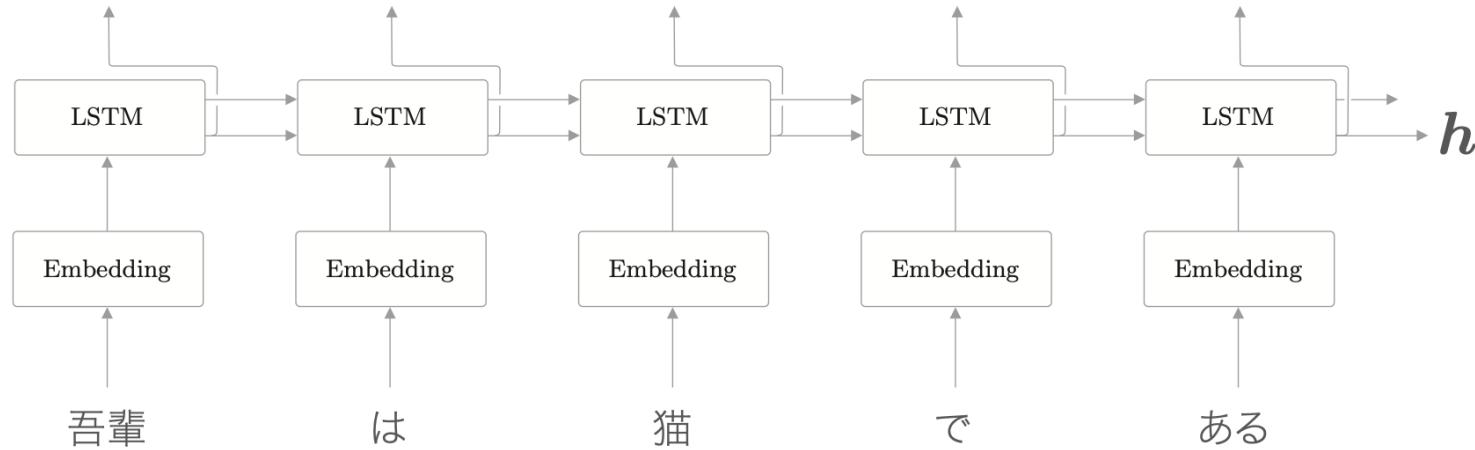
seq2seq 模型

seq2seq 模型也称为 Encoder-Decoder 模型。编码器对输入数据进行编码，解码器对被编码的数据进行解码。



基于 RNN 生成文本

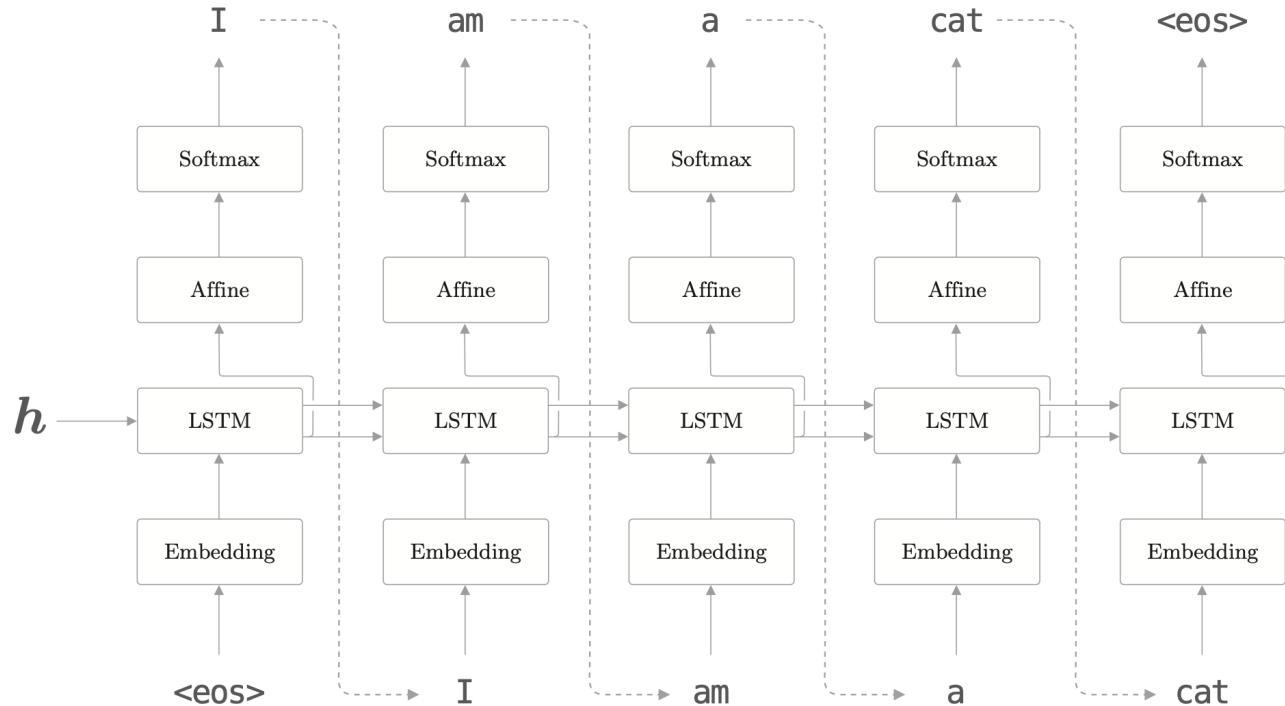
seq2seq 模型 - 编码器



编码器结构如上图所示，它将时序数据转换为隐藏状态 h 。由于 LSTM 的隐藏状态是一个固定长度的向量。说到底，编码就是将任意长度的文本转换为一个固定长度的向量。

基于 RNN 生成文本

seq2seq 模型 - 解码器



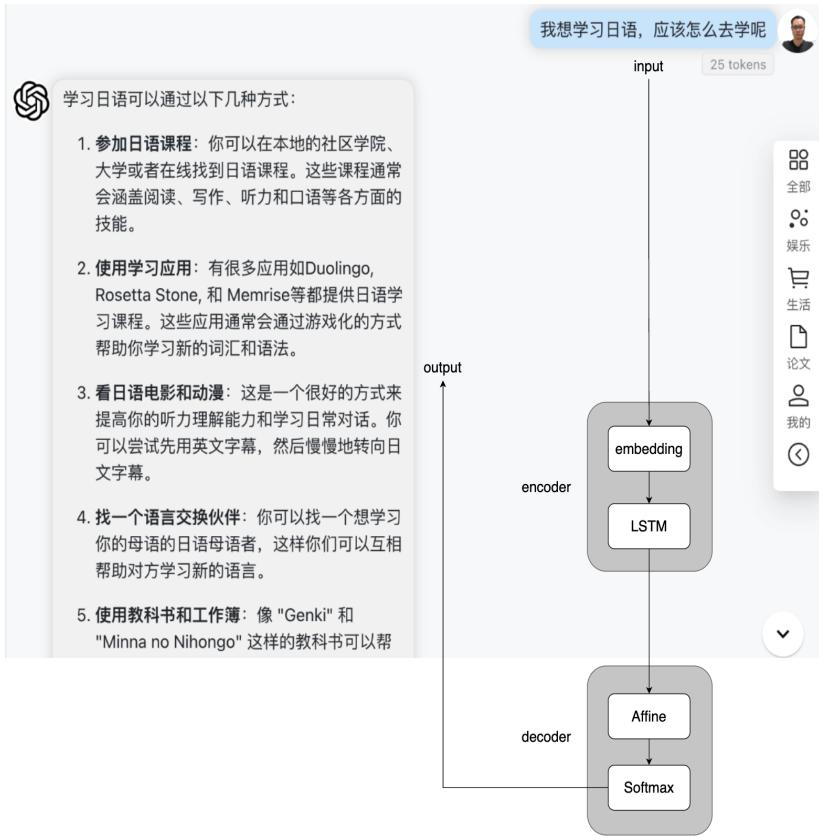
解码器结构如上图所示。其实就是一个生成器的结构，通过输入得到输出的概率分布，然后基于概率分布生成下一个单词（直接取概率最大的单词为确定性生成，使用概率生成的为非确定性生成）。

GPT

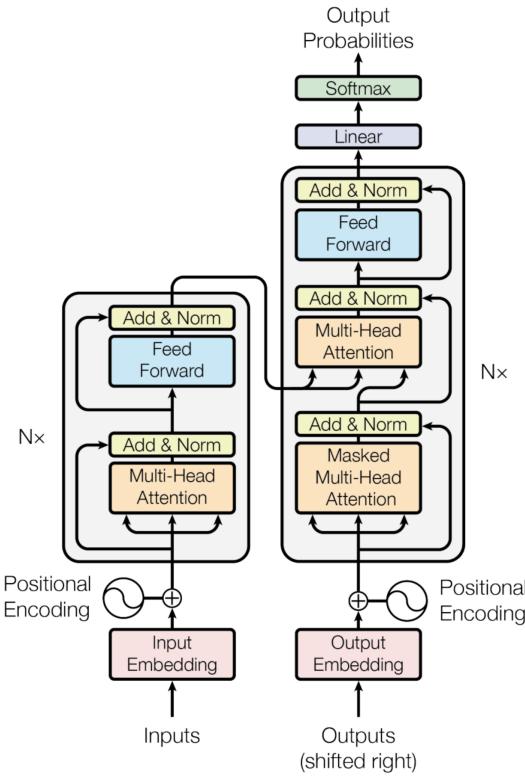
一个例子

当与 GPT 进行对话的时候：

1. 向 GPT 说一句话
2. 这句话进入到 GPT 内部
3. 进入编码器
4. 通过 embedding 转化为一个向量序列（seq）
5. 经过 RNN 结构转化为内部编码
6. 进入解码器
7. 通过向前网络计算输出
8. 将输出转化为单词概率分布
9. 通过单词概率分布生成输出
10. 整理输出并返回回答



总结



GPT 的 transformer 架构

要点

- 单词本身没意义，单词的意义是由其上下文决定的
- 大语言模型就是一个概率模型，输入是上文，输出是下文预测
- RNN 是大语言模型里的一个基础结构
- RNN 适合于处理时序数据
- LSTM 是更精细的 RNN 结构，可以分别控制输入，输出，遗忘，新记忆的权重
- 基于 RNN 的文本生成一般使用编码器-解码器架构
- 编码器将任意长度的输入编码为固定长度的内部表示
- 解码器基于固定长度内部表示生成输出的单词分布
- 根据输出的单词分布可以生成回答