Chatbot

Syllabus

- 概述
- 内容检索式Chatbot
- 生成式Chatbot

Chatbot Types

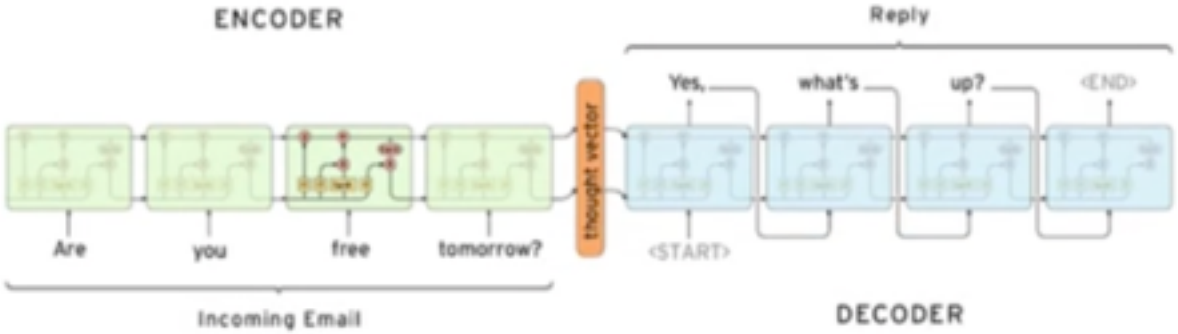
聊天机器人 (Chatbot)，也被称为对话引擎或者对话系统，大家在智能客服和语音智能助手等场景下可以看到它的身影，它是目前的热点之一。在以下内容当中我们将重温一些被用于聊天机器人中的深度学习技术，披露出目前技术能够解决或者可能解决的问题以及几乎难以解决的问题。

模型分类

基于检索技术的模型 VS 生成式模型

基于检索技术的模型较为简单，主要是根据用户的输入和上下文内容，使用了知识库（存储了事先定义好的回复内容）和一些启发式方法来得到一个合适的回复。启发式方法简单的有基于规则的表达式匹配，复杂的有一些机器学习里的分类器。这些系统不能够生成任何新的内容，只是从一个固定的数据集中找到合适的内容作为回复。

生成式模型则更加复杂，它不依赖于预定义好的回复内容，而是利用生成式的方法逐词(字)生成新的回复内容。生成式模型典型的有**基于机器翻译模型**的，与传统机器翻译模型不同的是，生成式模型的任务不是将一句话翻译成其他语言的一句话，而是将**用户的输入[翻译]为一个回答(response)**



Chatbot Types

总结

以上两种模型均有优缺点。对于基于检索技术的模型，由于使用了知识库且数据为预先定义好的，因此进行回复的内容语法上较为通顺，较少出现语法错误。是基于检索技术的模型中没有会话概念，不能结合上下文给出更加[智能]的回复。而生成式模型则更加[智能]一些，它能够更加有效地利用上下文信息从而知在讨论的东西是什么；然而生成式模型比较难以训练，并且输出的内容经常存在一些语法错误（尤其对于长句子而言），以及模型训练需要大规模的数据。

深度学习技术都能够用于基于检索技术的模型和生成式模型中，但是目前的研究热点在生成式模型上。深度学习框架例如**Sequence to Sequence**非常适合生成文本，非常多的研究者希望能够在这个领域取得成功。然而目前这一块的研究还在初期阶段，工业界的产品更多的还是使用基于检索计算的模型。

短对话 VS 长对话

直观上处理长对话内容将更加困难，这是因为你需要在当前对话的情境下知道之前的对话说过什么。如果是一问一答的形式，技术上这将简单的多。通常对于客服对话而言，长对话更加常见，一次对话中往往会伴随着多个关联问题。

开放域 VS 特定领域

面向开放域的聊天机器人技术面临更多困难，这是因为会话可能涉及的面太广，没有一个清晰的目标和意图。在一些社交网站例如Twitter和Reddit上的会话是属于开放域的，会话涉及的主题多种多样，需要的知识量也将非常巨大。

面向特定领域的相关技术则相对简单一些，这是因为特定领域给会话的主题进行了限制，目标和意图也更加清晰，典型的例子有客服系统助手和购物助手。这些系统通常是为了完成某些特定任务，尽管用户在该系统中也能够问些其他方面的东西，但是系统并不会给出相应的回复。

Challenges

如何结合上下文信息

为了产生质量更高的回复，聊天机器人系统通常需要利用一些上下文信息(Context)，这里的上下文信息包括了**对话过程中的语言上下文信息和用户的身份信息**等。在长对话中人们关注的是之前说了什么内容以及产生了什么内容的交换，这是语言上下文信息的典型。常见的方法是将一个会话转化为向量形式，但这对长会话而言是困难的。论文Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models和Attention with Intention for a Neural Network Conversation Model中的实验结果表明了这一点。另外，可以结合的上下文信息还包括会话进行时的日期地点信息、用户信息等。

语义一致性

理论上来说，机器人面对相同语义而不同形式的问题应该给予一致的回复，例如这两个问题[How old are you?]和[What's your age?]。这理解起来是简单的，但是是学术界目前的难题之一（如下图）。许多系统都试图对相同语义而不同形式的问题给予语义上合理的回复，但却没有考虑一致性，最大的原因在于训练模型的数据来源于大量不同的用户，这导致机器人失去了固定统一的人格。论文A Persona-Based Neural Conversation Model中提及的模型旨在创建具有固定统一人格的机器人。

message	Where do you live now?
response	I live in Los Angeles.
message	In which city do you live now?
response	I live in Madrid.
message	In which country do you live now?
response	England, you?

Performance

对话模型的评测

评价一个对话模型的好坏在于它是否很好地完成了某项任务，例如在对话中解决了客户的问题。这样的训练数据需要人工标注和评测，所以获取上需要一定人力代价。有时在开放域中的对话系统也没有一个清晰的优化目标。用于机器翻译的评测指标BLEU不能适用于此，是因为它的计算基础是语言表面上的匹配程度，而对话中的回答可以是完全不同词型但语义通顺的语句。论文How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation中给出结论，目前常用的评测指标均与人工评测无关。

意图和回复多样性

生成式模型中的一个普遍问题是，它们都想要生成一些通用的回答，例如[That's great!]和[I don't know]这样的可以应付许多的用户询问。早期的Google智能回复基本上以[I love you]回复所有的东西链接，这是一些模型最终训练出来的结果，原因在于训练数据和训练的优化目标。因此，有些研究学者开始关注如何提升机器人的回复的多样性，然而人们在对话过程中的回复与询问有一定特定关系，是有一定意图的，而许多面向开放域的机器人不具备特定的意图。

实际效果

以目前的研究水平所制造的机器人能够取得的效果如何？使用基于检索技术的显然无法制作出面向开放域的机器人，这是因为你不能编写覆盖所有领域的语料；而生成式的面向开放域的机器人还属于**通用人工智能(Artifical General Intelligence, AGI)**水平，距离理想状态还相距甚远，但相关研究学者还在致力于此。

对于特定领域的机器人，基于检索的技术和生成式模型都能够利用。但是对于长对话的情境，也面临许多困难。

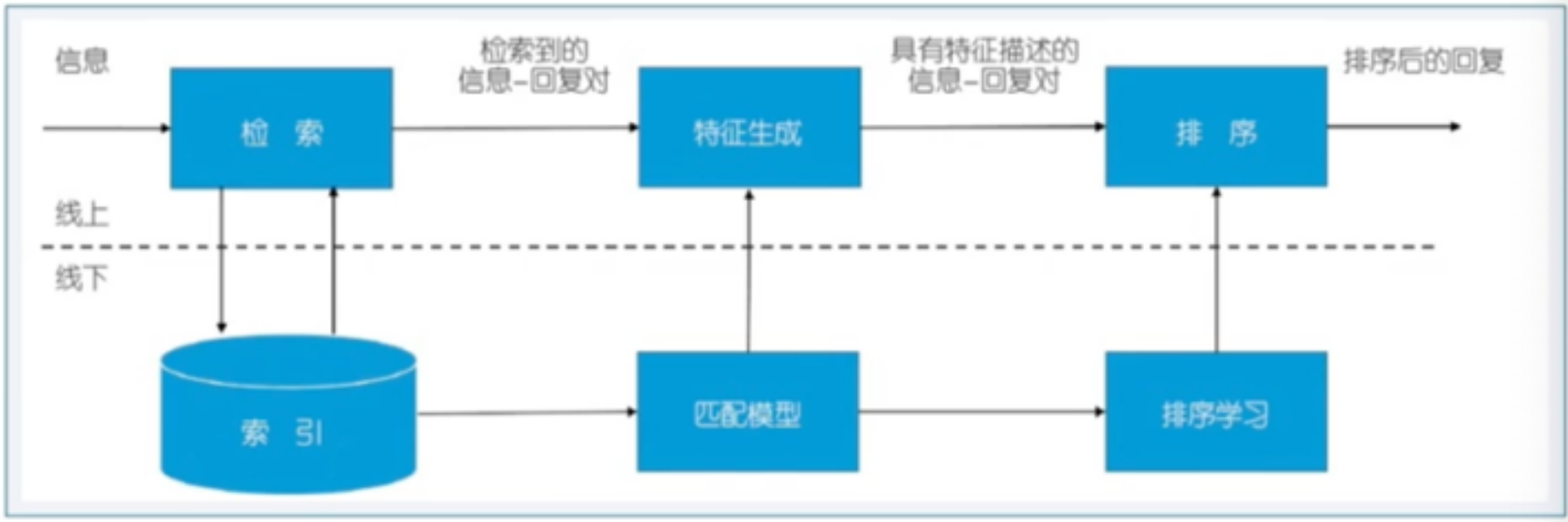
在最近对Andrew NG的采访中，NG提到：

> 目前深度学习的价值主要体现在能够获取大量数据的特定领域。目前一个无法做的事情是产生一个有意义的对话。

基于检索模型的聊天机器人

本文我们将介绍和实现一个基于检索模型的聊天机器人。检索模型所使用的回复数据通常是**预先存储且知道（或定义）的数据**，而不像生成式模型那样可以创造出崭新的、未知的回复内容（模型没有见过）。准确来讲，检索式模型的输入是**一段上下文内容 C (会话到目前未知的内容信息)** 和**一个可能作为回复的候选答案**；模型的输出是对这个候选答案的打分。寻找最合适的回复内容的过程是：先对一堆候选答案进行打分及排序，最后选出分值最高的那个最为回复。

也许你会质疑为什么不直接使用生成式模型，生成式模型不需要预先存储且定义好的数据，比起检索模型更加的灵活多变。原因在于目前生成式模型的效果并不佳，由于生成式模型的约束条件少，过于多变的模型导致生成的response中出现一些语法错误和语义无关的内容。生成式模型需要海量的训练数据，且难以优化。目前工业界常用的模型还是基于检索的模型，或者以生成式模型作为补充的两者结合，谷歌的Smart Reply就是一个例子。尽管目前生成式模型是学术界的研究热点，但在实践中使用检索式模型是更加合适的选择。



Ubuntu对话数据集

这篇博客我们将使用Ubuntu对话数据集（论文来源 github地址）。这个数据集（Ubuntu Dialog Corpus, UDC）是目前最大的公开对话数据集之一，它是来自Ubuntu的IRC网络上的对话日志。这篇论文介紹了该数据集生成的具体细节。下面简单介紹一下数据的格式。

训练数据有1,000,000条实例，其中一半是正例（label为1），一半是负例（label为0，负例为随机生成）。每条实例包括一段上下文信息（context），即Query；和一段可能的回复内容，即Response；Label为1表示该Response确实是Query的回复，Label为0则表示不是。下面是数据示例：

```
In [62]: pd.options.display.max_colwidth = 500
         train_df.head()
```

Out[62]:

	Context	Utterance	Label
0	i think we could import the old comment via rsync , but from there we need to go via email . i think it be easier than cach the status on each bug and than import bite here and there __eou__ __eot__ it would be veri easi to keep a hash db of message-id __eou__ sound good __eou__ __eot__ ok __eou__ perhap we can ship an ad-hoc apt_preferec __eou__ __eot__ version ? __eou__ __eot__ thank __eou__ __eot__ not yet __eou__ it be cover by your insur ? __eou__ __eot__ yes __eou__ but it 's reali no...	basic each xfree86 upload will not forc user to upgrad 100mb of font for noth __eou__ no someth i do in my spare time . __eou__	1
1	i 'm not suggest all - onli the one you modifi . __eou__ __eot__ ok , it sound like you re agre with me , then __eou__ though rather than " the one we modifi " , my idea be " the one we need to merg " __eou__ __eot__	sorri __eou__ i think it be ubuntu relat . __eou__	0
2	afternoon all __eou__ not entir relat to warti , but if grub-instal take 5 minut to instal , be this a sign that i should just retri the instal :) __eou__ __eot__ here __eou__ __eot__ you might want to know that thinic in warti be buggi compar to that in sid __eou__ __eot__ and appar gnome be sudden almost perfect (out of the thinic problem) , nobodi report bug : -p __eou__ i do n't get your question , where do you want to past ? __eou__ __eot__ can i file the panel not link to ed ? :) ...	yep . __eou__ oh , okay . i wonder what happen to you __eou__ what distro do you need ? __eou__ yes __eou__	0
3	interest __eou__ grub-instal work with / be ext3 , fail when it be xfs __eou__ i think d-i instal the relev kernel for your machin . i have a p4 and it instal the 386 kernel __eou__ holi crap a lot of stuff get instal by default :) __eou__ you be instal vim on a box of mine __eou__ ;) __eou__ __eot__ more like osx than debian ;) __eou__ we have a select of python modul avail for great justic (and python develop) __eou__ __eot__ 2.8 be fix them iirc __eou__ __eot__ pong __eou__ vino will...	that the one __eou__	1
4	and becaus python give mark a woodi __eou__ __eot__ i 'm not sure if we re mean to talk about that public yet . __eou__ __eot__ and i think we be a " pant off " kind of compani ... : p __eou__ you need new glass __eou__ __eot__ mono 1.0 ? dude , that 's go to be a barrel of laugh for total non-releas relat reason dure hoari __eou__ read bryan clark 's entri about networkmanag ? __eou__ __eot__ there be an accompani irc convers to that one < g > __eou__ explain ? __eou__ i guess you could s...	(i think someon be go to make a joke about .au bandwidth ...) __eou__ especi not if you re use screen ;) __eou__	1

内容检索式Chatbot

数据集的生成使用了NLTK工具，包括分词、stemmed、lemmatized等文本预处理步骤；同时还使用了NER技术，将文本中的实体，如姓名、地点、组织、URL等替换成特殊字符。这些文本预处理并不是必须的，但是能够提升一些模型的性能。据统计，query的平均长度为86个word，而response的平均长度为17个word，更多的数据统计信息见Jupyter notebook。

数据集也包括了测试和验证集，但这两部分的数据和训练数据在格式上不太一样。在测试集和验证集中，对于每一条实例，有一个正例和九个负例数据（也称为干扰数据）。模型的目标在于给正例的得分尽可能的高，而给负例的得分尽可能的低。下面是数据示例：

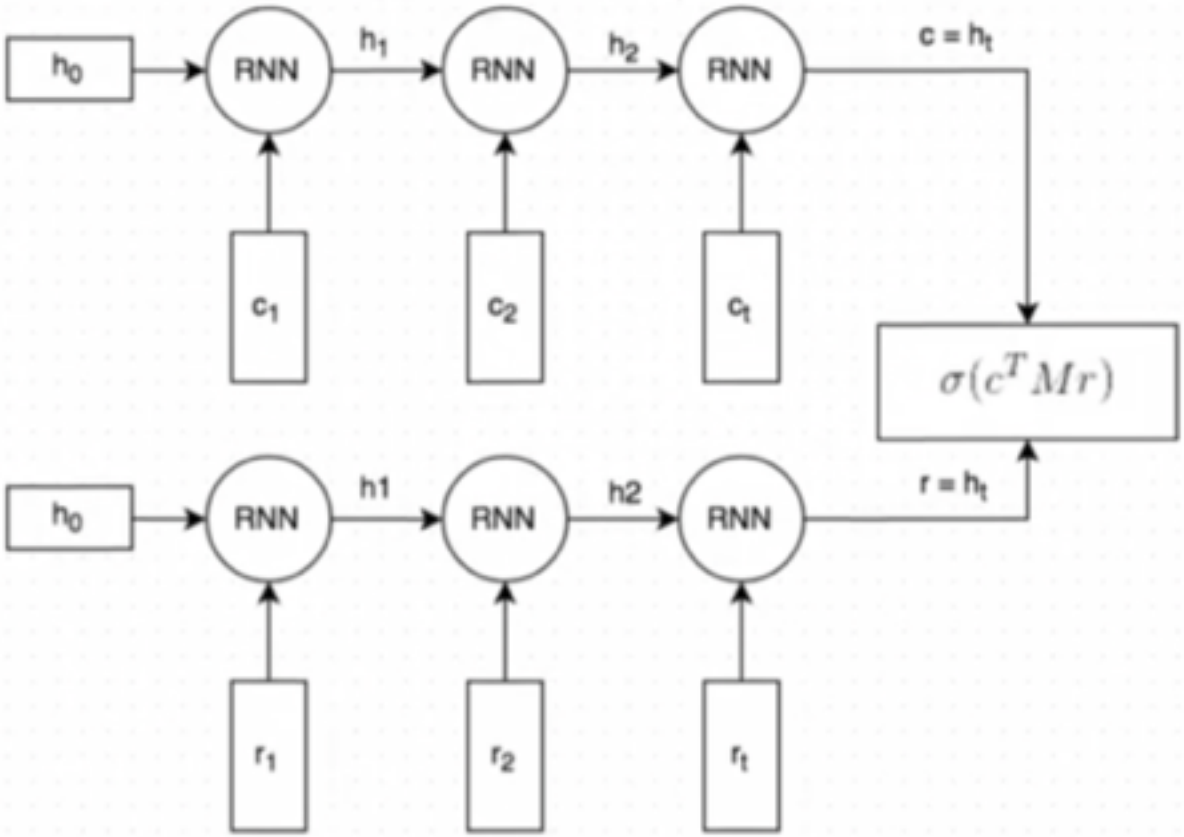
模型的评测方式有很多种。其中最常用到的是**recall@k**，即经模型对候选的response排序后，前k个候选中存在正例数据（正确的那个）的占比；显然k值越大，该指标会越高，因为这对模型性能的要求越松。

在Ubuntu数据集中，负例数据都是随机生成的；然而在现实中，想要从全部的数据中随机生成负例是不可能的。谷歌的Smart Reply则使用了聚类技术，然后将每个类的中取一些作为负例，这样生成负例的方式显得更加合理（考虑了负例数据的多样性，同时减少时间开销）。

内容检索式Chatbot

这篇博文将建立的NN模型为两层Encoder的LSTM模型（Dual Encoder LSTM Network），这种形式的网络被广泛应用在chatbot中（尽管可能效果并不是最佳的那个，你可以尽可能地尝试其他的NN模型）。seq2seq模型常用于机器翻译领域，并取得了较大的效果。使用Dual LSTM模型的原因在于这个模型被证明在这个数据集有较好的效果（详情见这里），这可以作为我们后续模型效果的验证。

两层Encoder的LSTM模型的结构图如下（论文来源）：



内容检索式Chatbot

大致的流程如下：

(1) Query和Response都是经过分词的，分词后每个词embedded为向量形式。初始的词向量使用GloVe vectors，之后词向量随着模型的训练会进行fine-tuned（实验发现，初始的词向量使用GloVe并没有在性能上带来显著的提升）。

(2) 分词且向量化的Query和Response经过相同的RNN（word by word）。RNN最终生成一个向量表示，捕捉了Query和Response之间的[语义联系]（图中的c和r）；这个向量的维度是可以指定的，这里指定为256维。

(3) 将向量c与一个矩阵M相乘，来预测一个可能的回复r'。如果c为一个256维的向量，M维256*256的矩阵，两者相乘的结果为另一个256维的向量，我们可以将其解释为[一个生成式的回复向量]。矩阵M是需要训练的参数。

(4) 通过点乘的方式来预测生成的回复r'和候选的回复r之间的相似程度，点乘结果越大表示候选回复作为回复的可信度越高；之后通过sigmoid函数归一化，转成概率形式。图中把第(3)步和第(4)步结合在一起了。

为了训练模型，我们还需要一个损失函数（loss function）。这里使用二元的交叉熵（binary cross-entropy）作为损失函数。我们已知实例的真实label `y`，值0或1；通过上面的第(4)步可以得到一个概率值 `y'`；因此，交叉熵损失值为 `L = -y * ln(y') - (1 - y) * ln(1 - y')`。这个公式的意义是直观的，即当 `y=1` 时，`L = -ln(y')`，我们期望 `y'` 尽量地接近1使得损失函数的值越小；反之亦然。

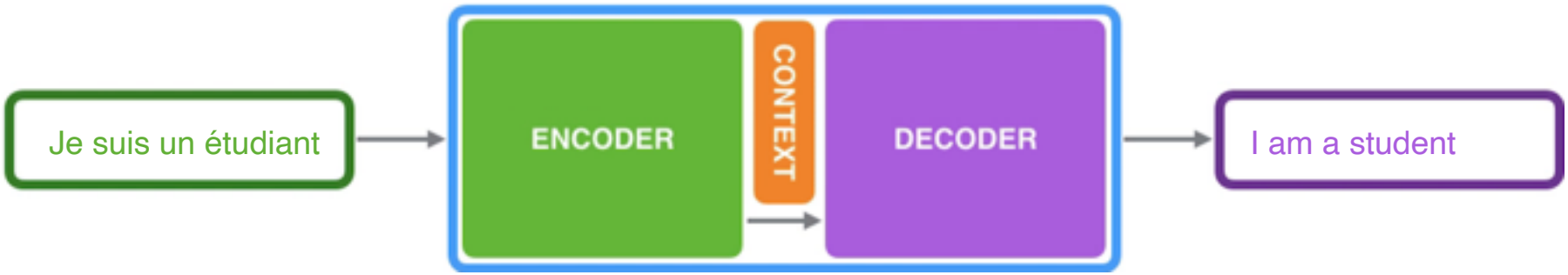
实现过程中使用了numpy、pandas、TensorFlow和TF Learn等工具。

生成式Chatbot

序列到序列的模型是非常有意思的NLP模型，我们的很多NLP任务，是文本到文本的映射(对应)，这个过程就像是下面图里展示的过程。当然seq2seq模型不仅仅是用在NLP中的模型，它的输入也可以是语音信号或者图像表示。

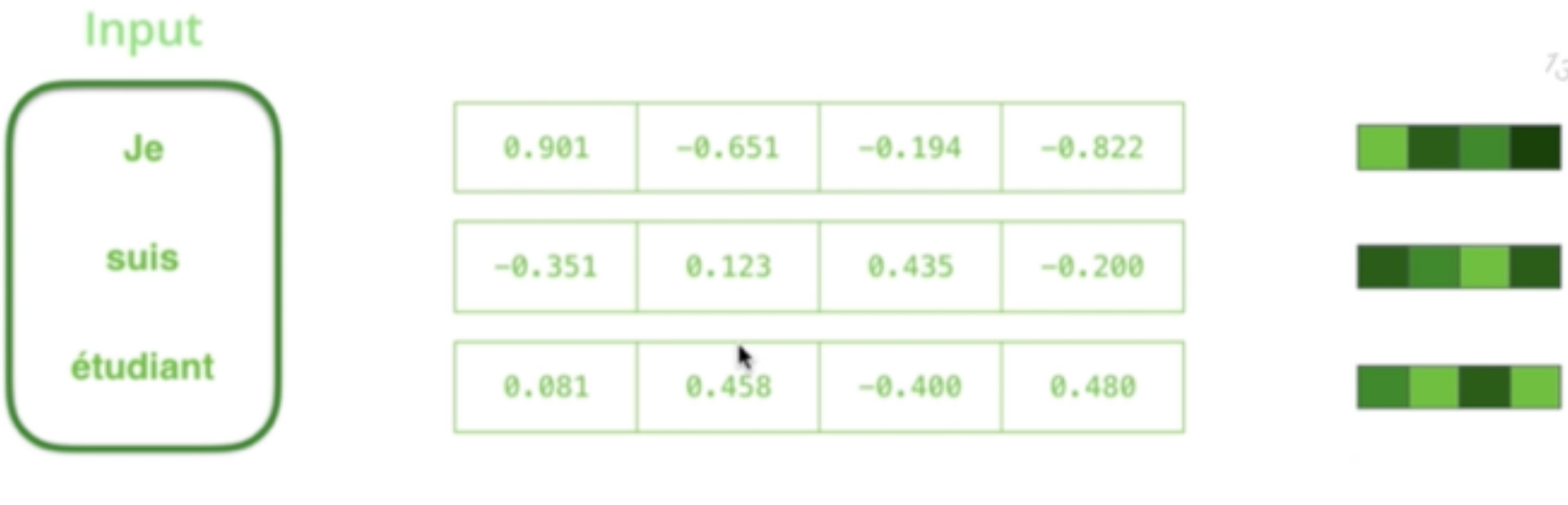
更具体一点，在NLP的任务中，其实输入的是文本序列，输出的很多时候也是文本序列，下图所示的是一个典型的机器翻译任务中，输入的文本序列(源语言表述)到输出的文本序列(目标语言表述)之间的变换。

更细节一点的结构是一个"编码解码器"结构，编码器处理输入序列中的每个元素(在这里可能是1个词)，将捕获的信息编译成向量（称为上下文内容向量）。在处理整个输入序列之后，编码器将上下文发送到解码器，解码器逐项开始产生输出序列。

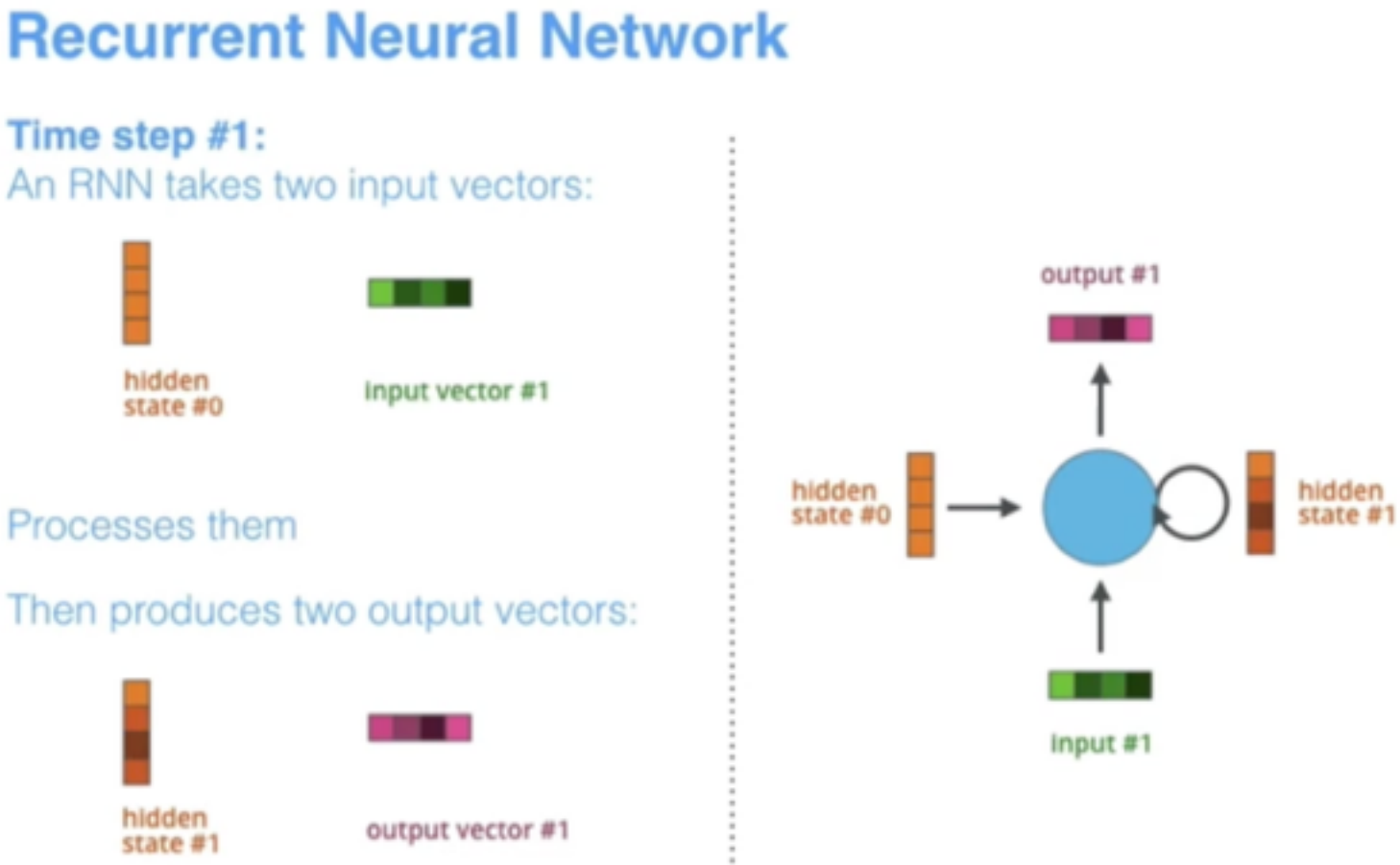


生成式Chatbot

输入的数据(文本序列)中的每个元素(词)通常会被编码成一个稠密的向量，这个过程叫做word embedding，如下图所示



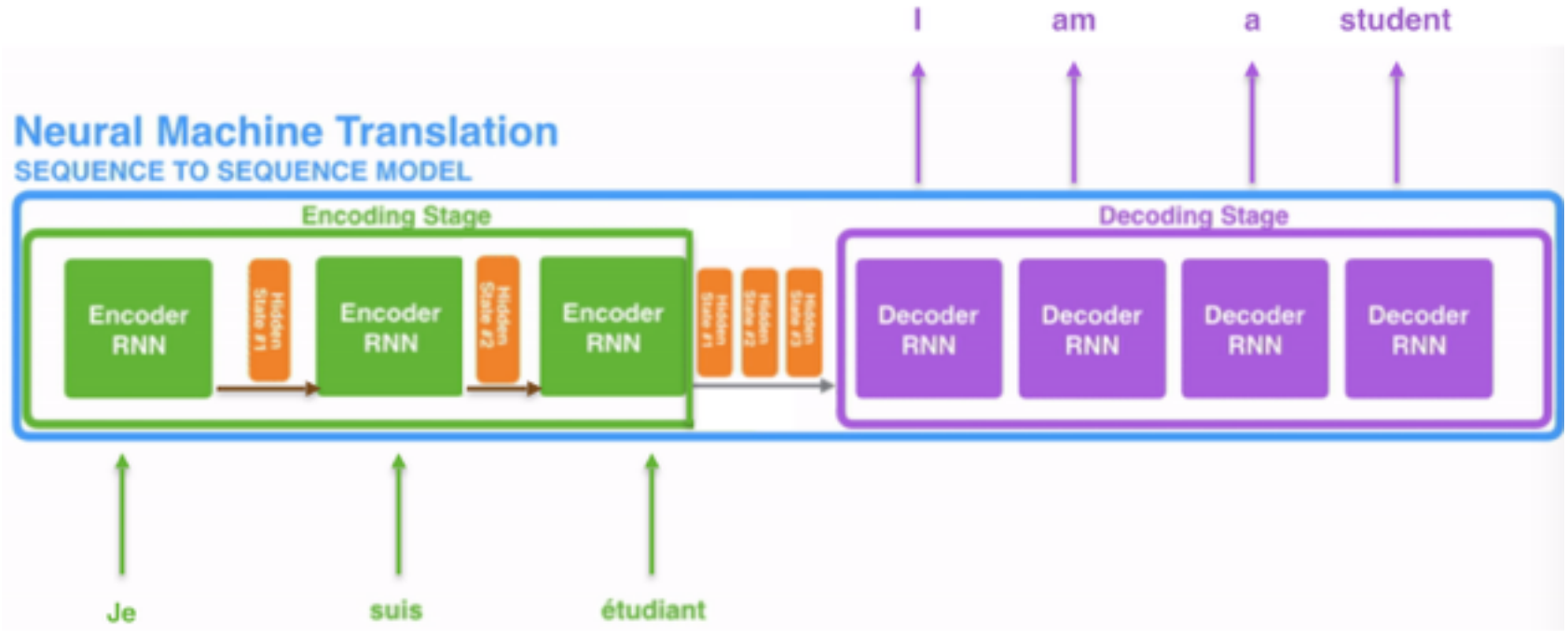
我们的encoder和decoder都会借助于循环神经网络(RNN)这类特殊的神经网络完成，循环神经网络会接受每个位置(时间点)上的输入，同时经过处理进行信息合，并可能会在某些位置(时间点)上输出。如下图所示。



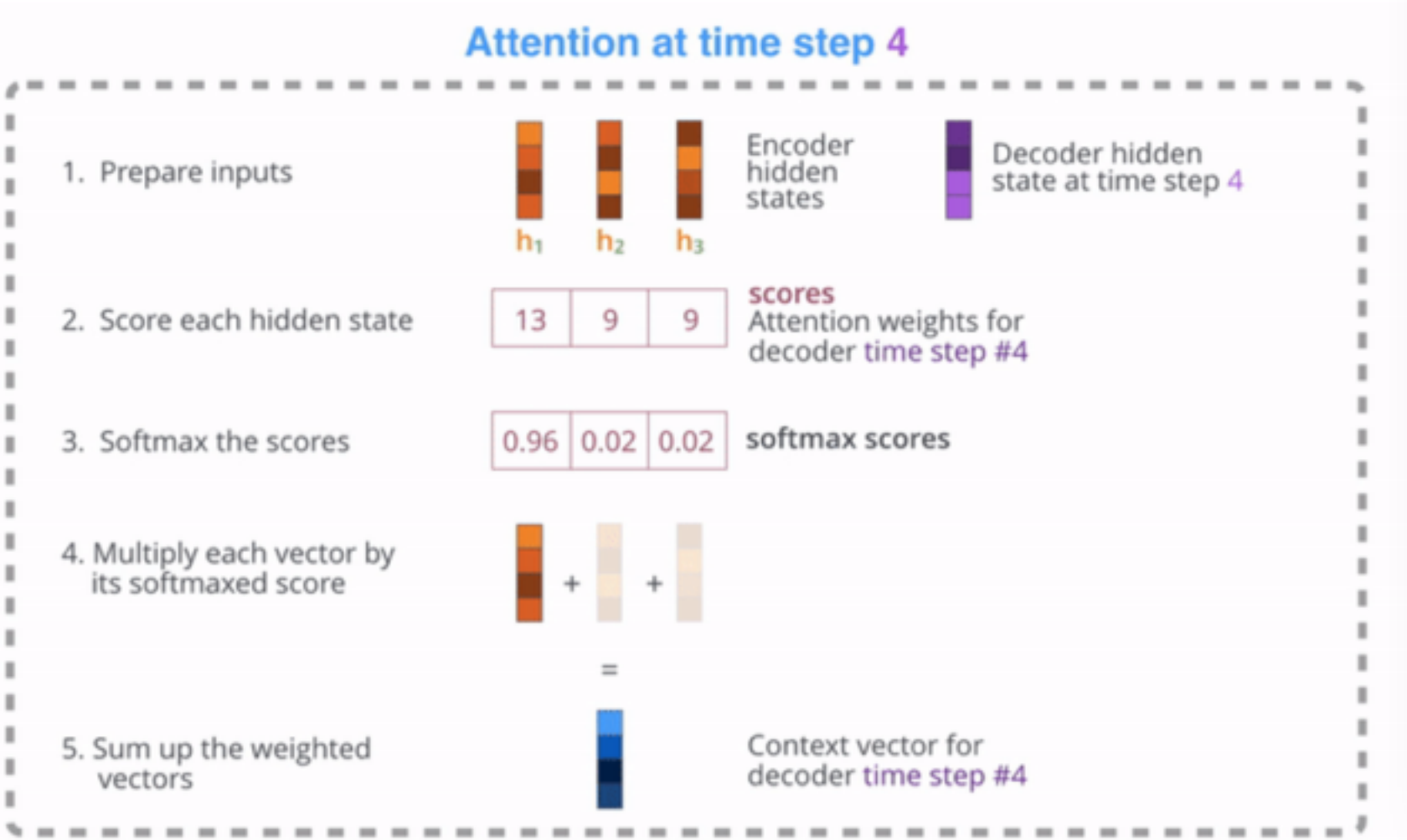
生成式Chatbot

在更多的时候，我们考虑到提升效果，不会寄希望于把所有的内容都放到一个上下文向量(context vector)中，而是会采用一个叫做**注意力模型**的模型来动态处
和解码，动态的图如下所示。

所谓的注意力机制，可以粗略地理解为是一种对于输入的信息，根据重要程度进行不同权重的加权处理(通常加权的权重来源于softmax后的结果)的机制，如下图
所示，是一个在解码阶段，简单地对编码器中的hidden states进行不同权重的加权处理的过程。

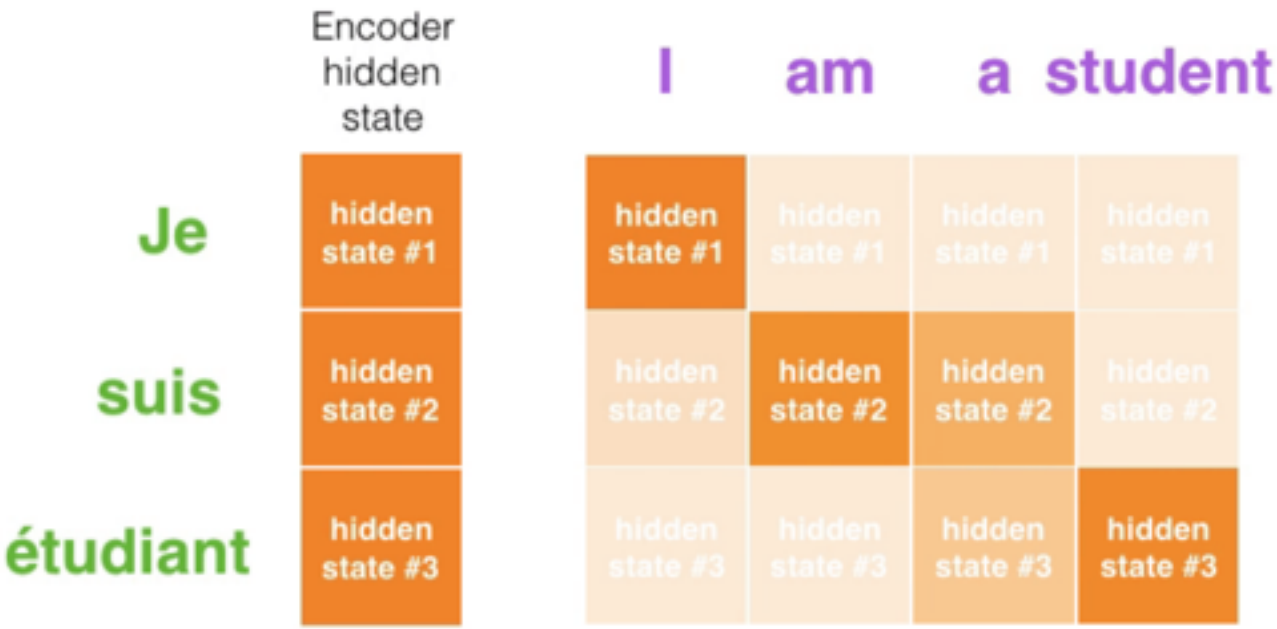


所示，是一个在解码阶段，简单地对编码器中的hidden states进行不同权重的加权处理的过程。



在luong中提到了三种score的计算方法。这里图解前两种：

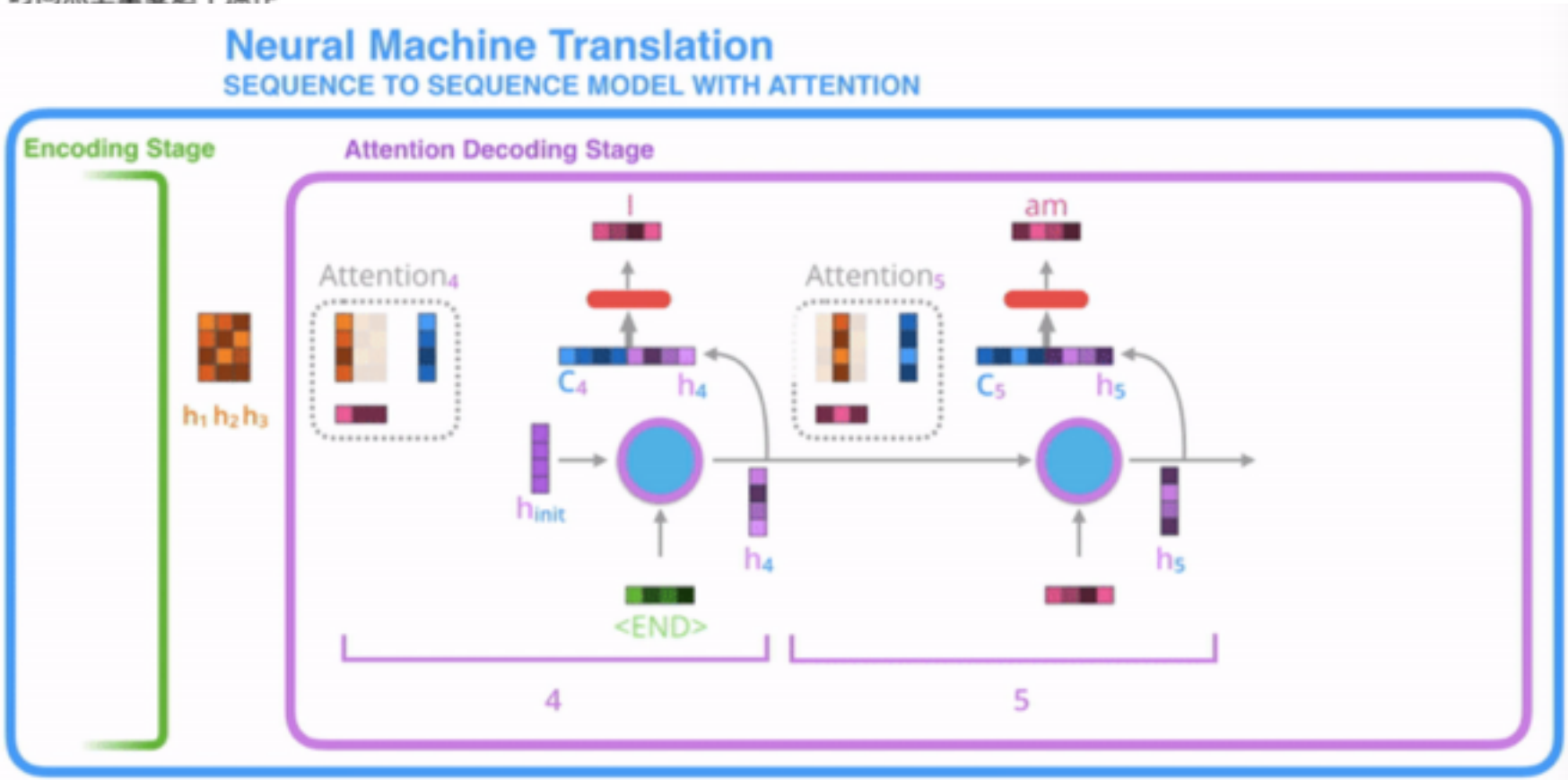
$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{W}_a [\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s] & \textit{concat} \end{cases}$$

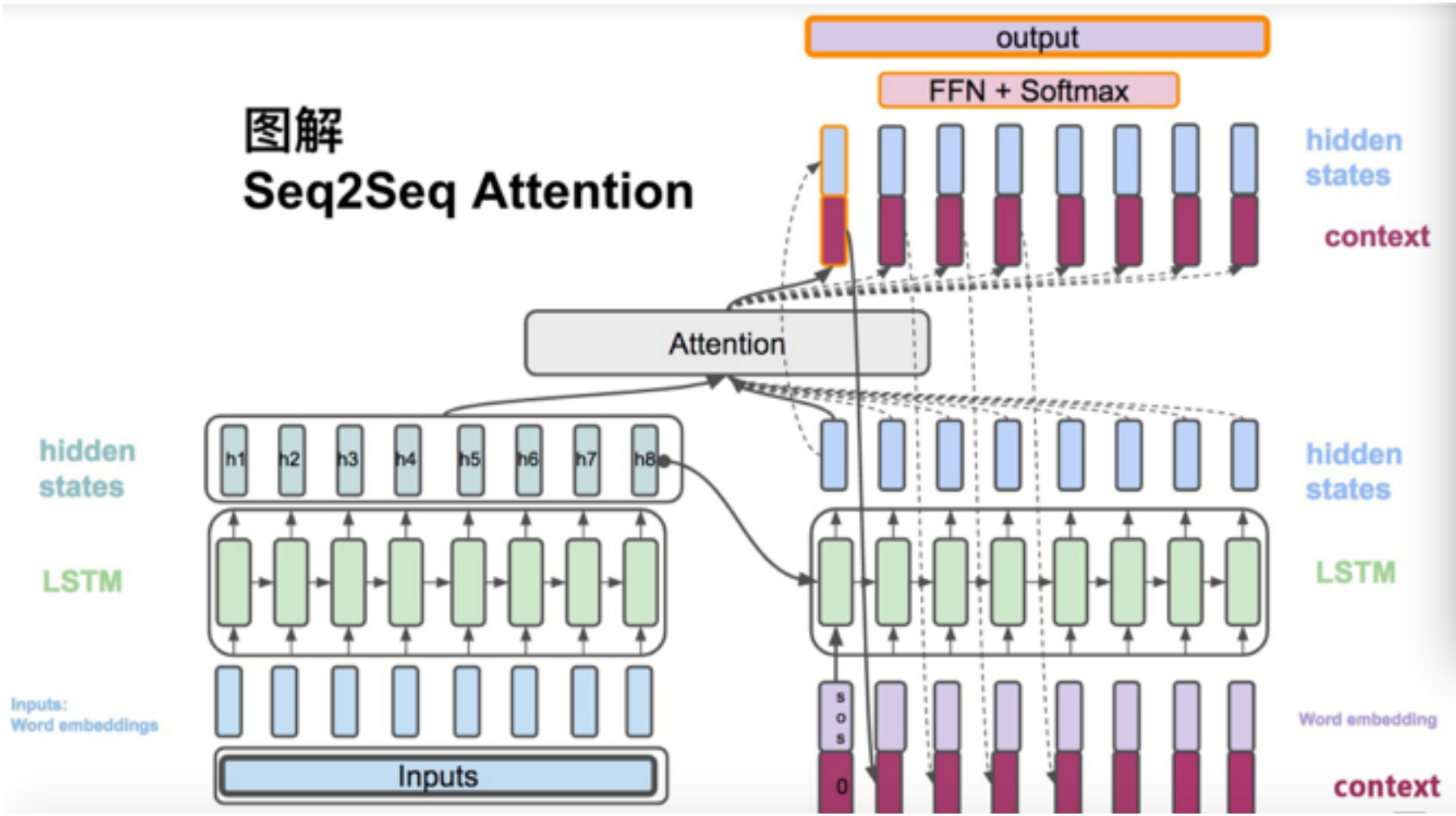


生成式Chatbot

更详细一点的**注意力解码**过程如下图所示。

- 带注意力的解码器RNN接收的嵌入(embedding)和一个初始的解码器隐藏状态(hidden state)。
- RNN处理输入，产生输出和新的隐藏状态向量（h4），输出被摒弃不用。
- attention的步骤：使用编码器隐藏状态(hidden state)和h4向量来计算该时间步长的上下文向量（C4）。
- 把h4和C4拼接成一个向量。
- 把拼接后的向量连接全连接层和softmax完成解码
- 每个时间点上重复这个操作




output
FFN + Softmax
图解
Seq2Seq Attention
hidden states
context
Attention
hidden states
h1
h2
h3
h4
h5
h6
h7
h8
hidden states
LSTM
LSTM
Inputs:
Word embeddings
Word embedding
Inputs
s
o
s
0
context

生成式Chatbot

输入： $x = (x_1, \ldots, x_{T_x})$

输出： $y = (y_1, \ldots, y_{T_y})$

(1) $h_t = RNN_{enc}(x_t, h_{t-1})$, Encoder方面接受的是每一个单词word embedding，和上一个时间点的hidden state。输出的是这个时间点的hid

(2) $s_t = RNN_{dec}(\hat{y}_{t-1}, s_{t-1})$ ， Decoder方面接受的是目标句子里单词的word embedding，和上一个时间点的hidden state。

(3) $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$, context vector是一个对于encoder输出的hidden states的一个加权平均。

(4) $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$, 每一个encoder的hidden states对应的权重。

(5) $e_{ij} = score(s_i, h_j)$, 通过decoder的hidden states加上encoder的hidden states来计算一个分数，用于计算权重(4)

(6) $\hat{s}_t = tanh(W_c[c_t; s_t])$, 将context vector 和 decoder的hidden states 串起来。

(7) $p(y_t | y_{<t}, x) = softmax(W_s \hat{s}_t)$ ， 计算最后的输出概率。