RNN

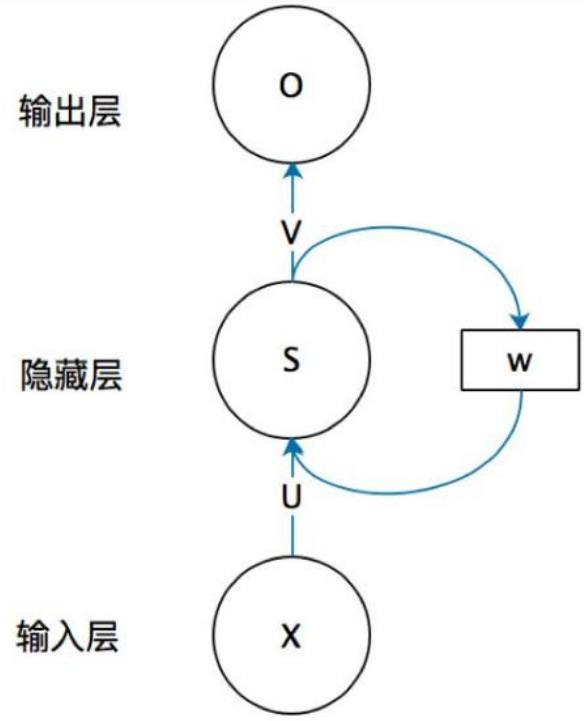
循环神经网络

为什么需要 RNN（循环神经网络）

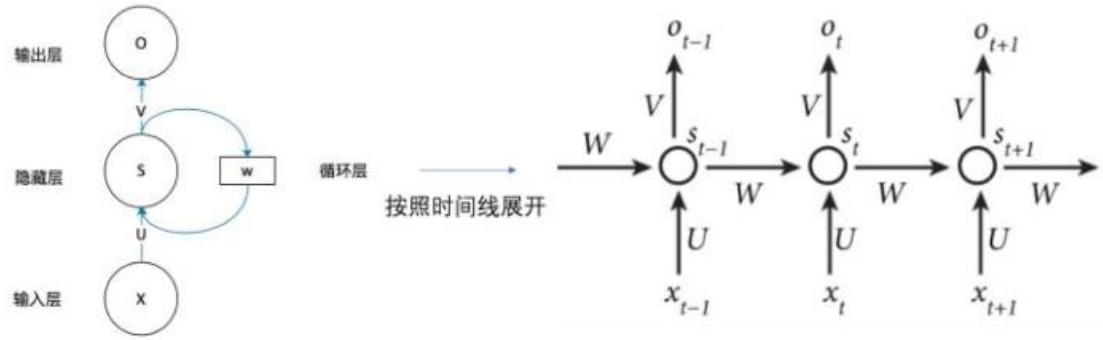
DNN 都只能单独的取处理一个个的输入，前一个输入和后一个输入是完全没有关系的。但是，某些任务需要能够更好的处理序列的信息，即前面的输入和后面的输入是有关系的。

比如，当我们在理解一句话意思时，孤立的理解这句话的每个词是不够的，我们需要处理这些词连接起来的整个序列； 当我们处理视频的时候，我们也不能只单独的去分析每一帧，而要分析这些帧连接起来的整个序列。

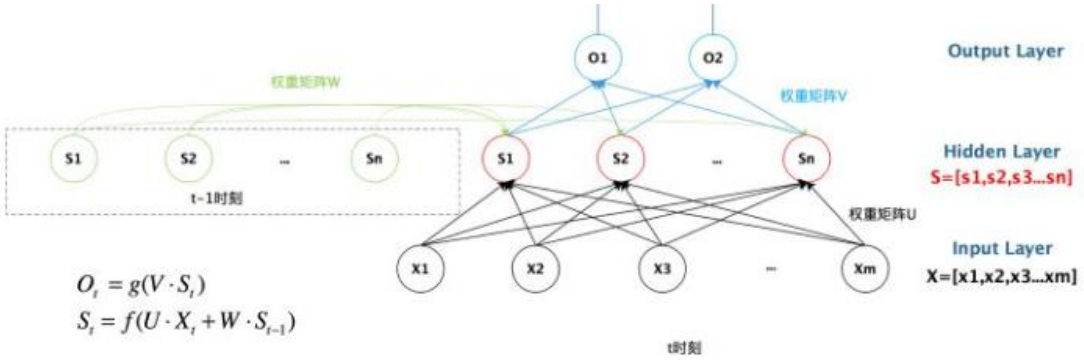
以 nlp 的一个最简单词性标注任务来说，将我 吃 苹果 三个单词标注词性为 我/nn 吃/v 苹果/nn。



尚学堂·瓦战程序员



RNN时间线展开图



$$O_t = g(V \cdot S_t)$$

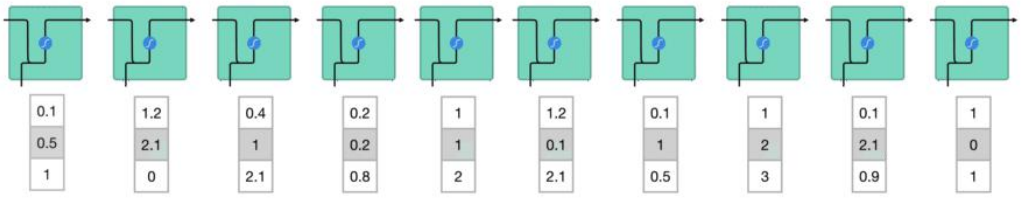
$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

St的值不仅仅取决于Xt，还取决于St-1

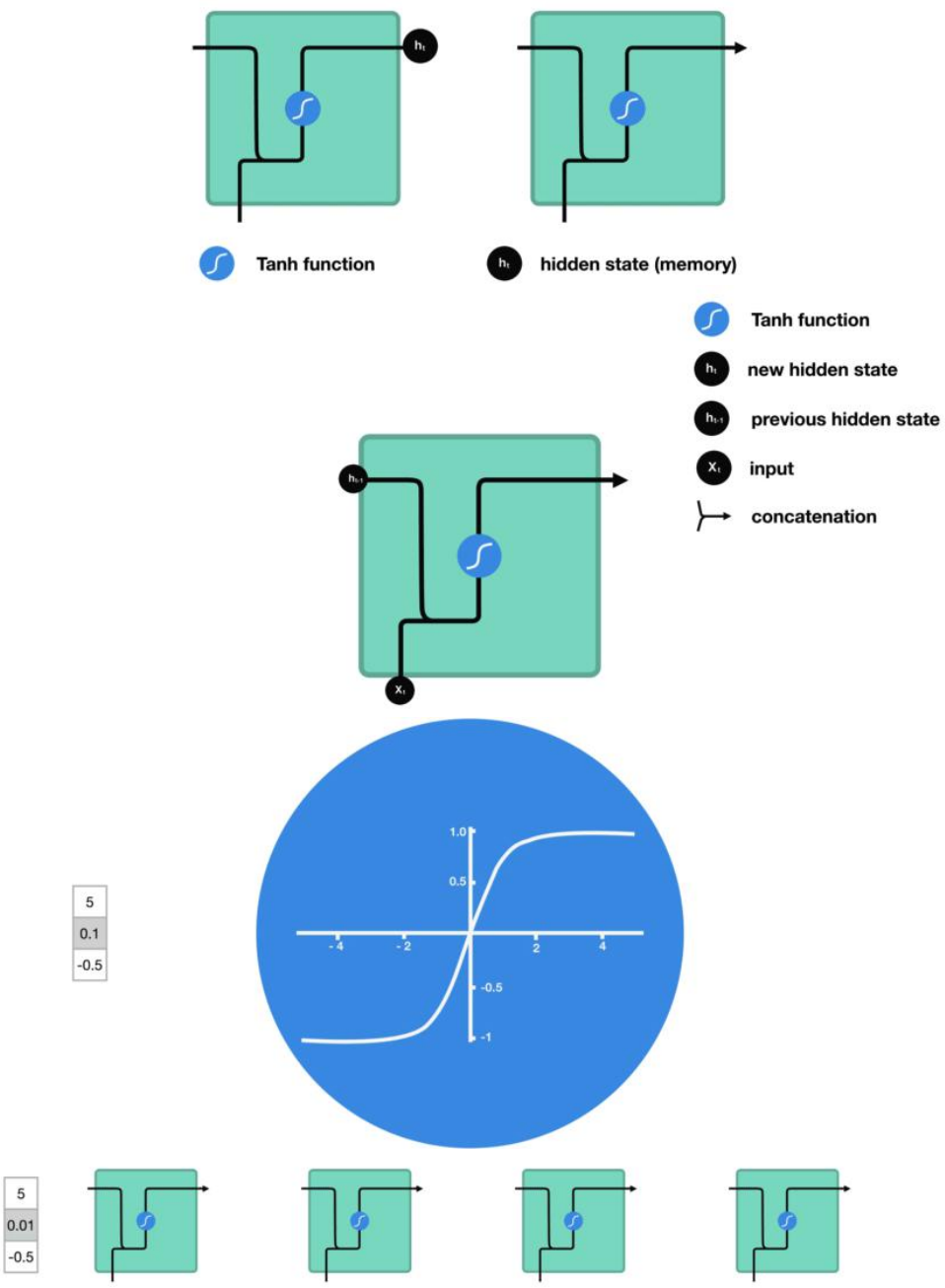
RNN公式

RNN 手写数字识别

LSTM



尚学堂·瓦战程序员



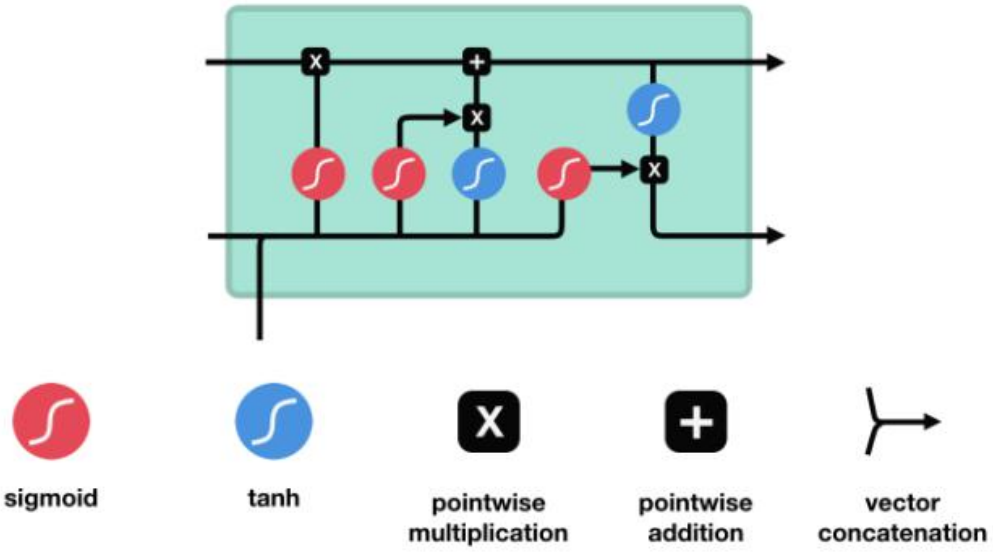
The Problem, Short-term Memory

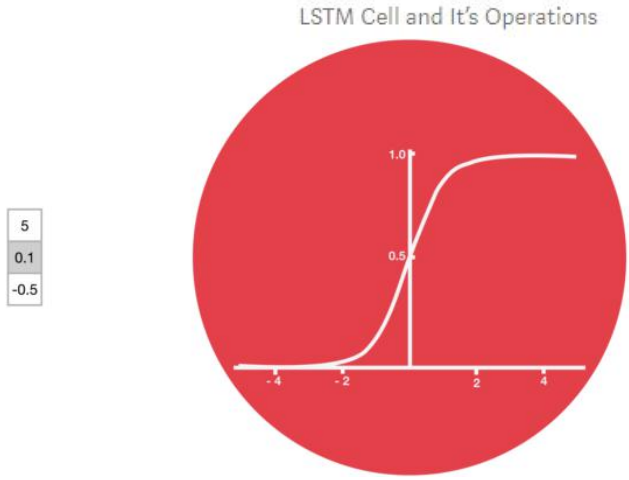
Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they’ ll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN’ s may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn’ t contribute too much learning.

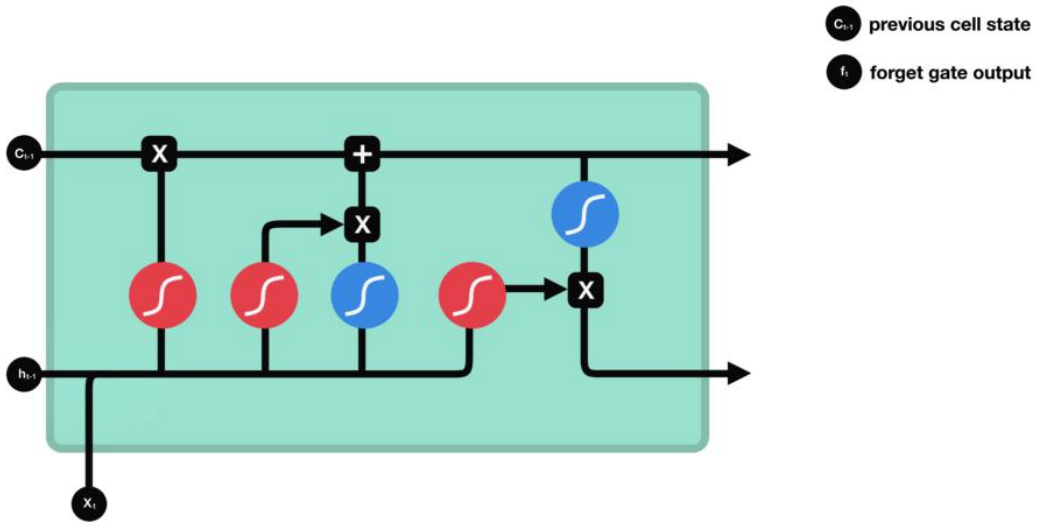
尚学堂·瓦战程序员

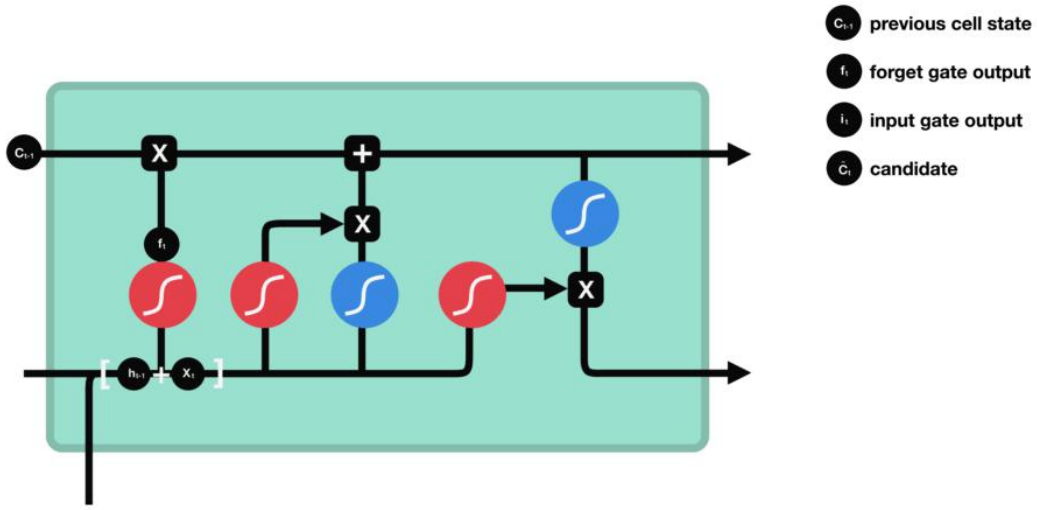
So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don’ t learn, RNN’ s can forget what it seen in longer sequences, thus having a short-term memory.

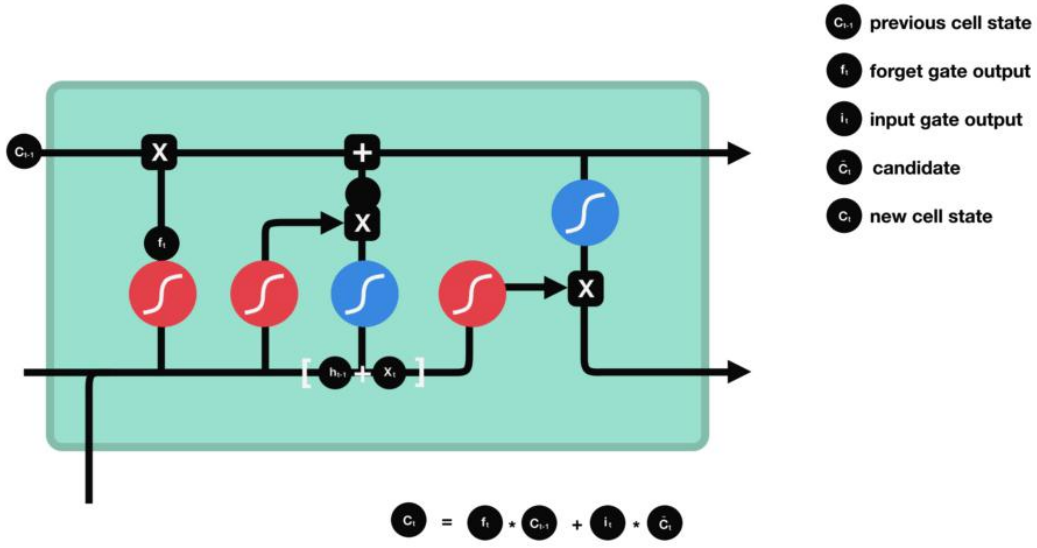




尚学堂·瓦战程序员

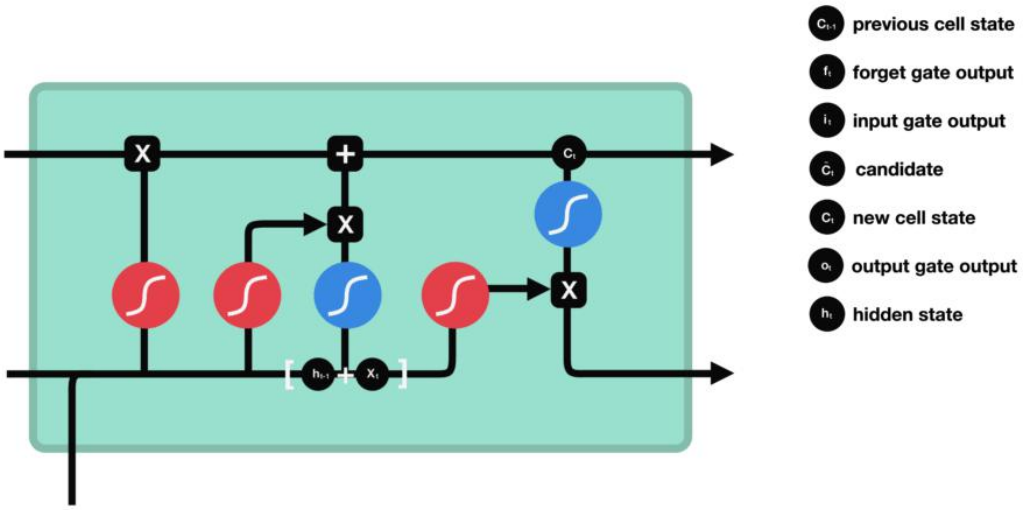




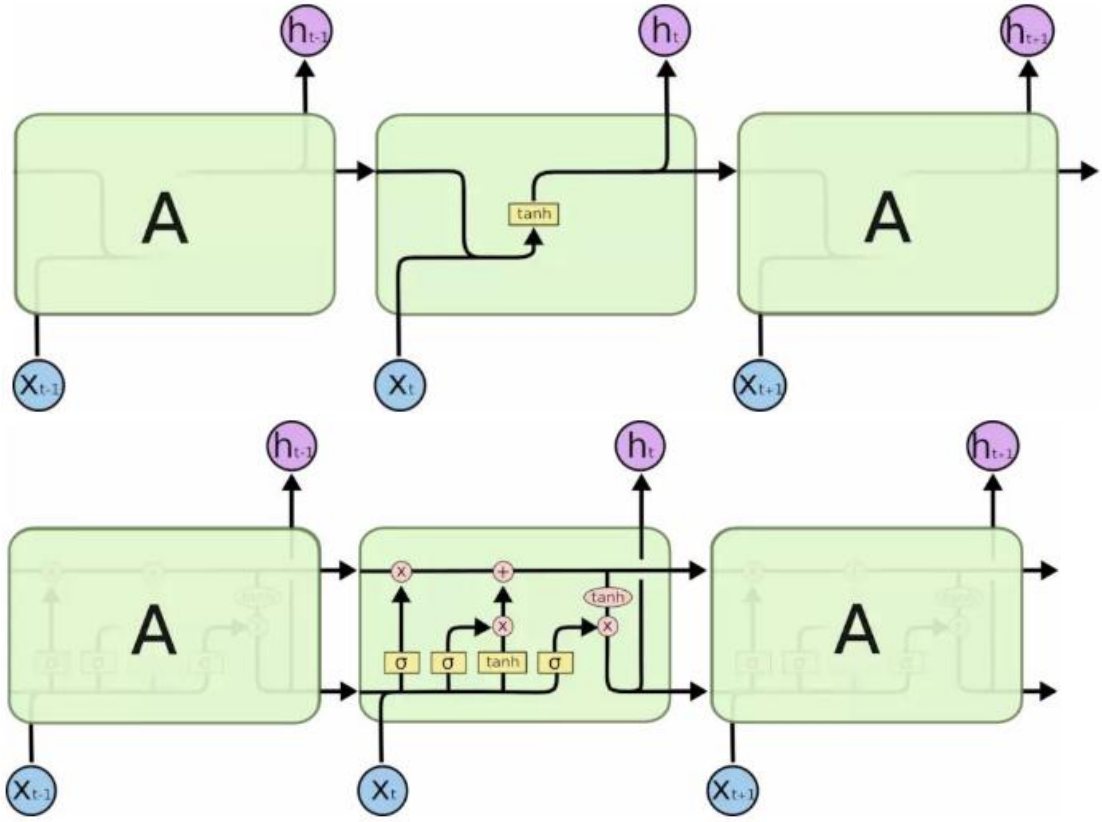


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

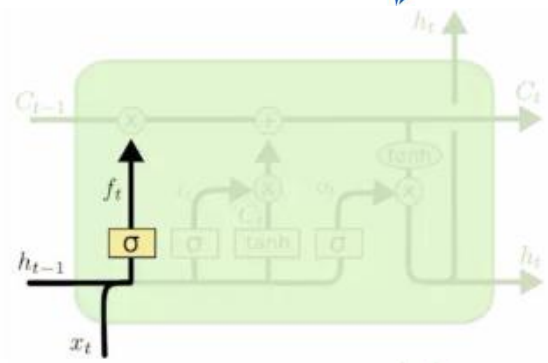
尚学堂·瓦战程序员



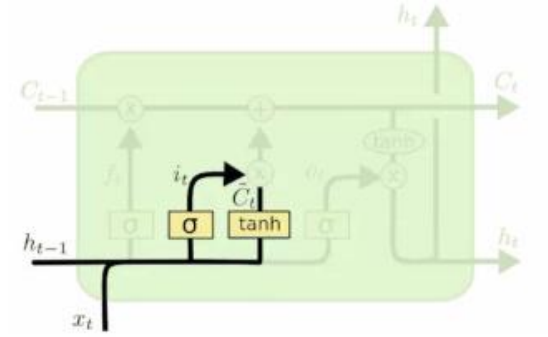
To review, the Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be.



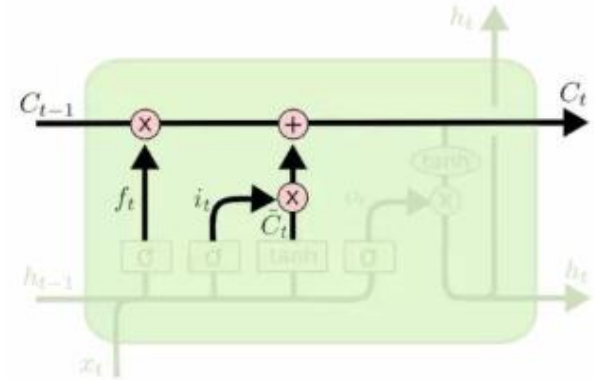
尚学堂·瓦战程序员



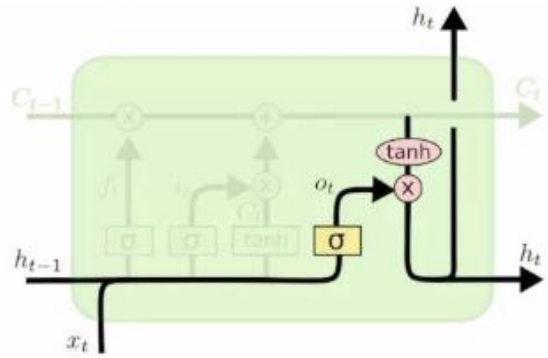
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



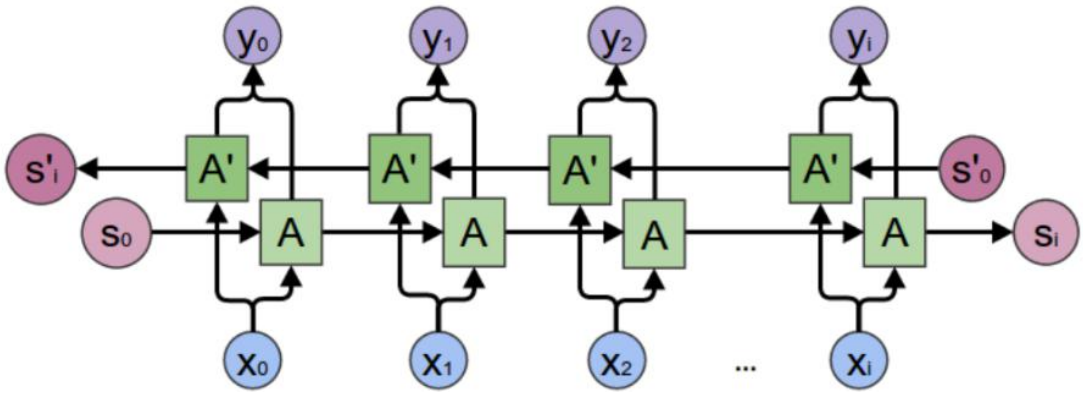
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

NLP 情感分析

Keras

尚学堂·瓦战程序员

双向 LSTM



LSTM 应用到双向 RNN 中

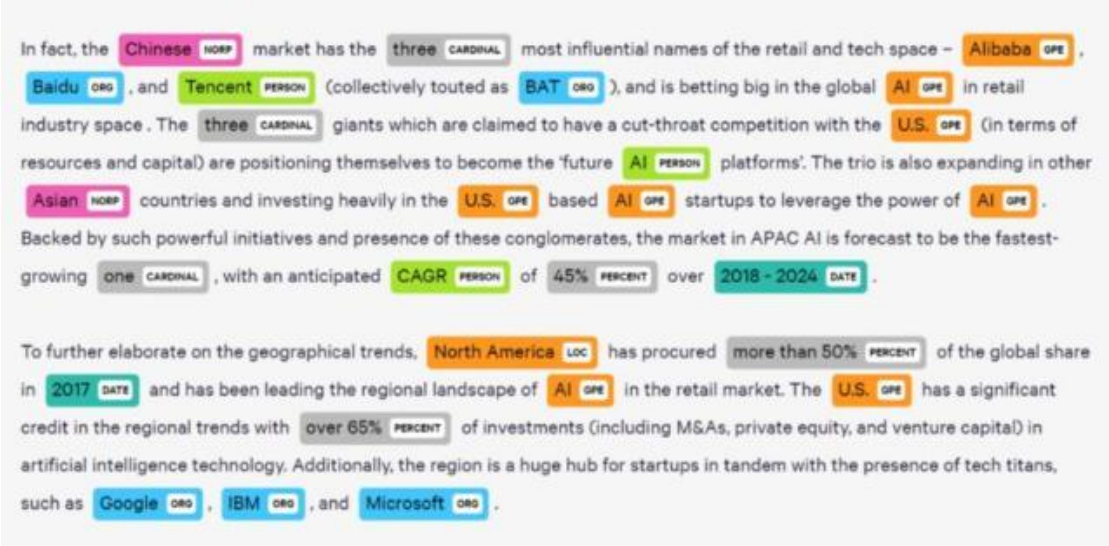
```
# lstm 模型正方向传播的 RNN
lstm_fw_cell = tf.nn.rnn_cell.BasicLSTMCell(embedding_size, forget_bias=1.0)
# 反方向传播的 RNN
lstm_bw_cell = tf.nn.rnn_cell.BasicLSTMCell(embedding_size, forget_bias=1.0)
```

但是看来看去，输入两个 cell 都是相同的啊？

其实在 bidirectional_dynamic_rnn 函数的内部，会把反向传播的 cell 使用 array_ops.reverse_sequence 的函数将输入的序列逆序排列，使其可以达到反向传播的效果。

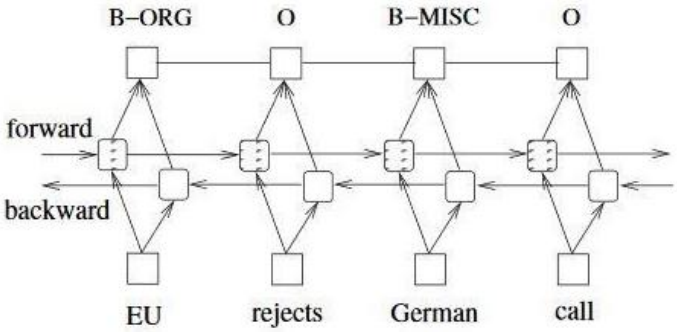
在实现的时候，我们是需要传入两个 cell 作为参数就可以了

NER 命名实体识别

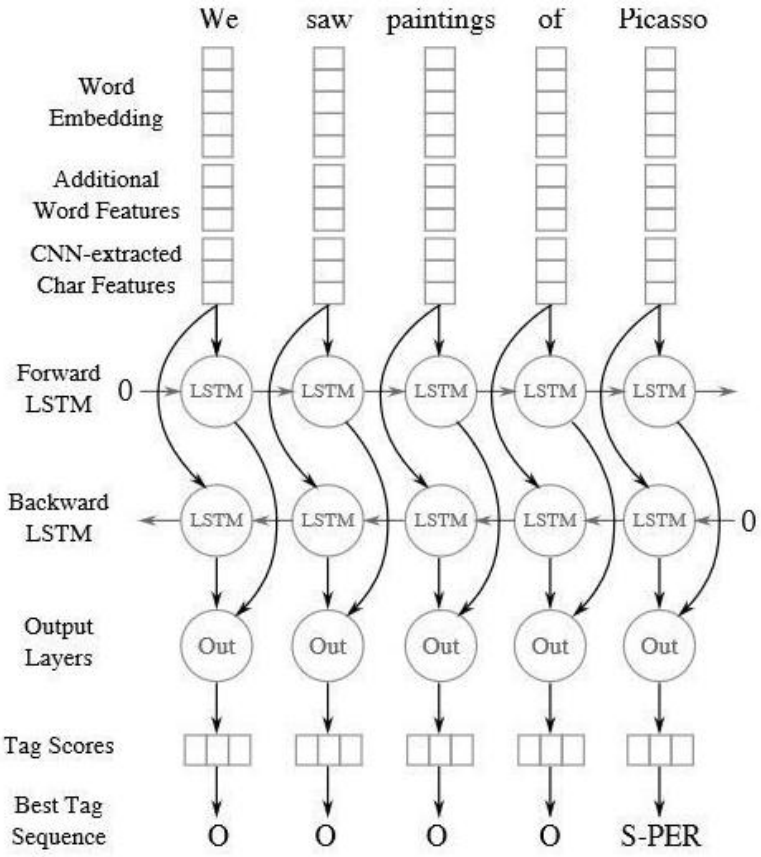


尚学堂·瓦战程序员

双向 LSTM-CRF

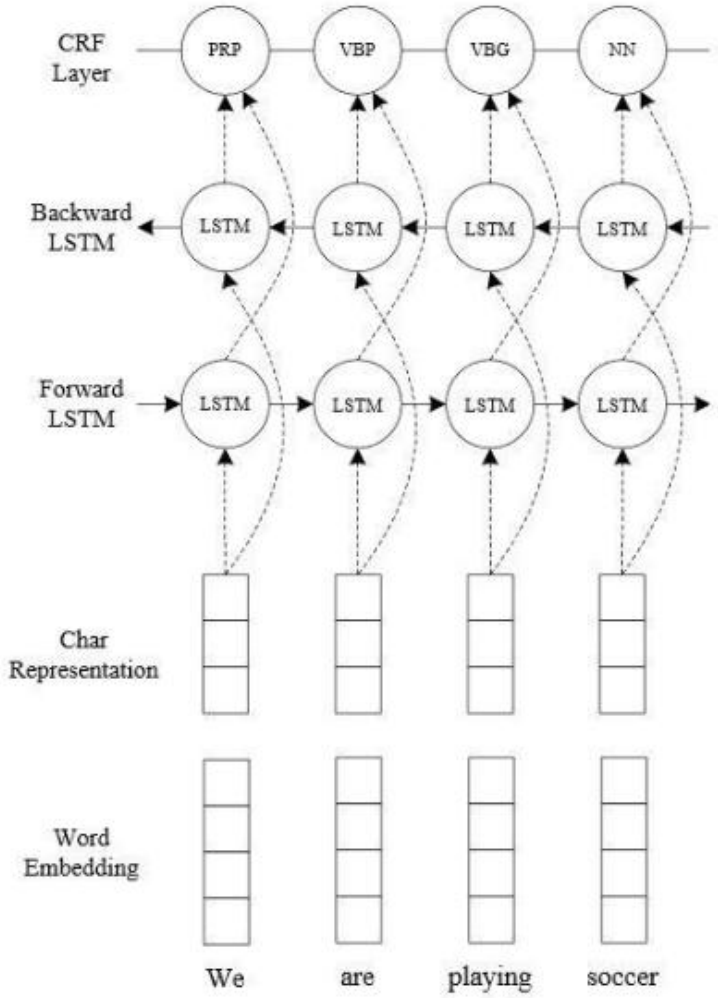


双向 LSTM-CNNs



双向 LSTM-CNNS-CRF

尚学堂·瓦战程序员


CRF Layer
PRP
VBP
VBG
NN
Backward LSTM
LSTM
LSTM
LSTM
LSTM
Forward LSTM
LSTM
LSTM
LSTM
LSTM
Char Representation
Word Embedding
We
are
playing
soccer

尚学堂·瓦战程序员

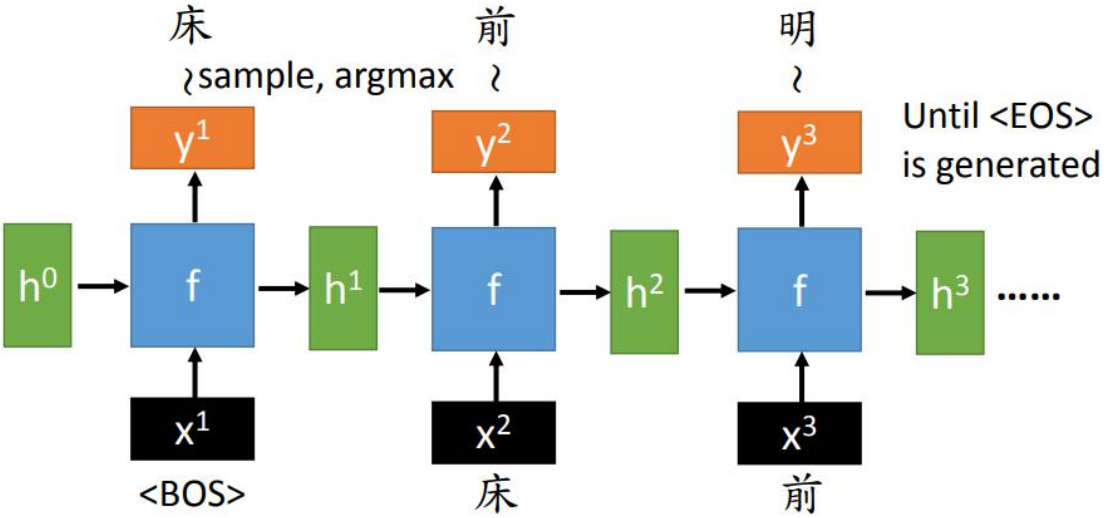
写唐诗

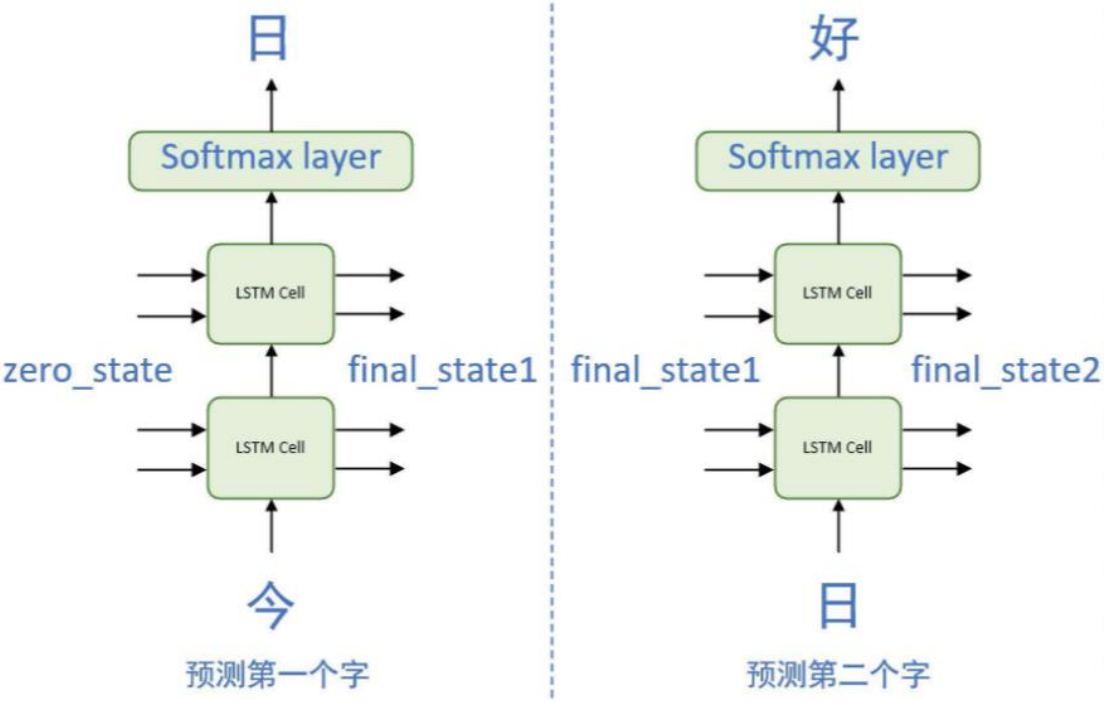
Generation

y^2: P(w|<BOS>,床)

y^3: P(w|<BOS>,床,前)

- Sentences are composed of characters/words
- Generating a character/word at each time by RNN





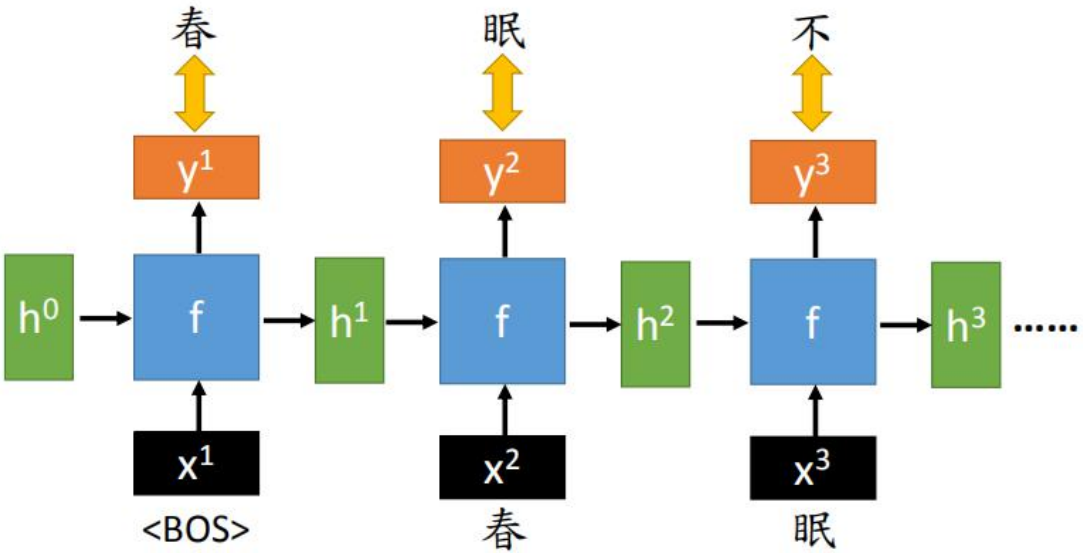
尚学堂·瓦战程序员

Generation

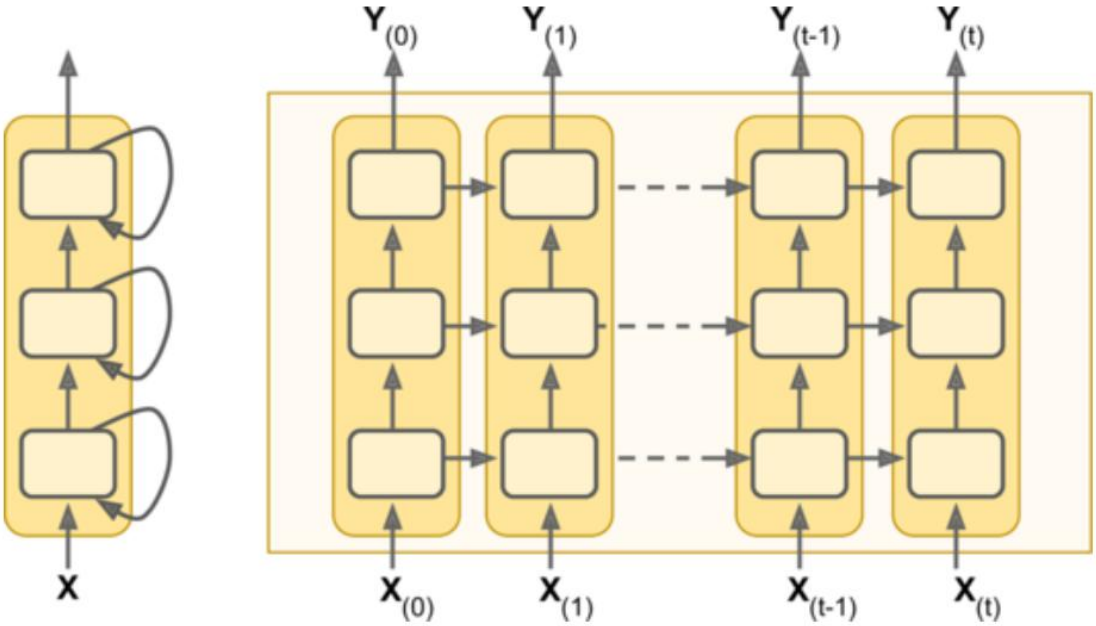
: minimizing cross-entropy

- ***Training***

Training data: 春 眠 不 覺 曉

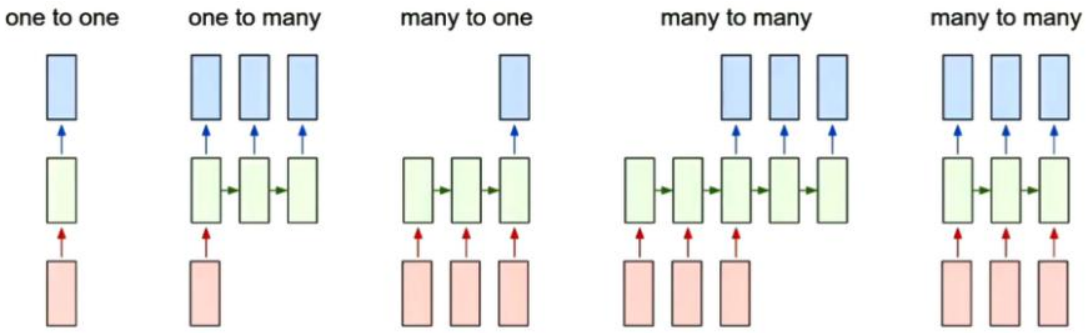


MultiRNNCell



尚学堂·瓦战程序员

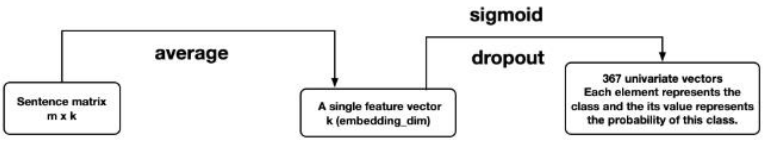
Seq2Seq

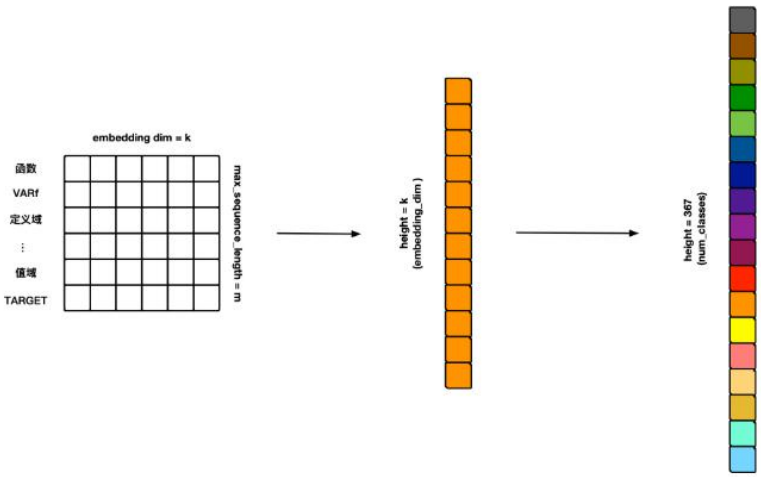


聊天机器人

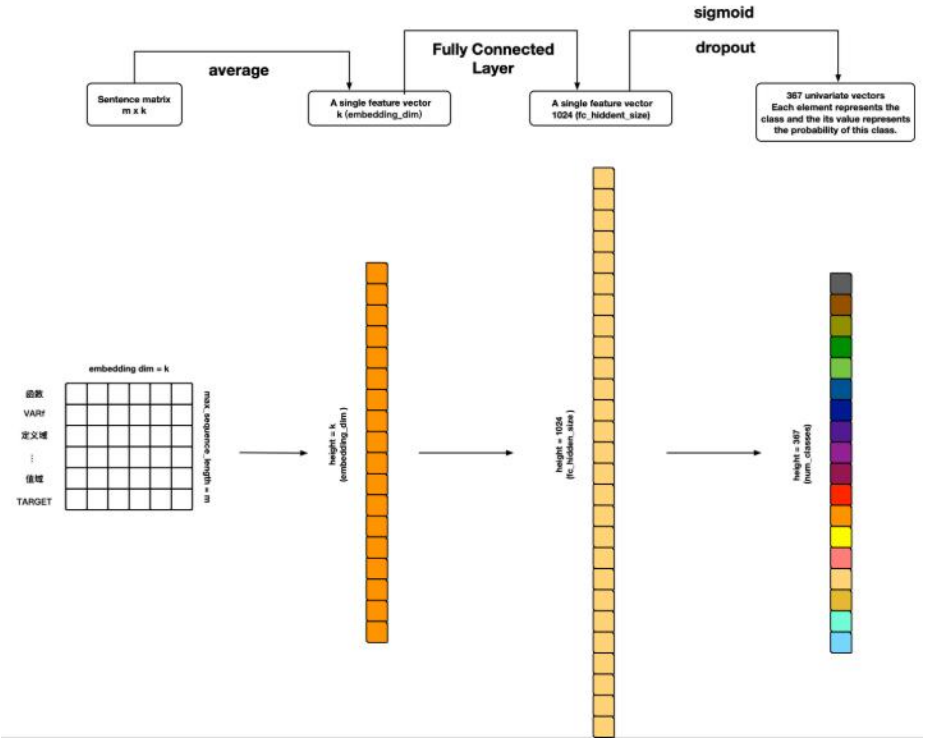
多标签网络结构

FastText

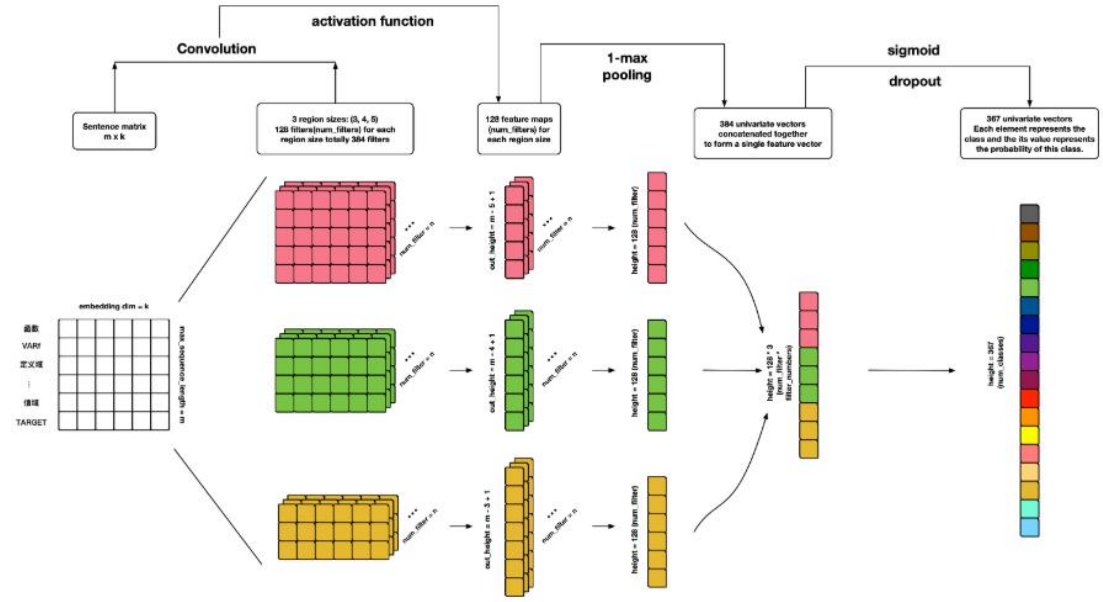




TextANN

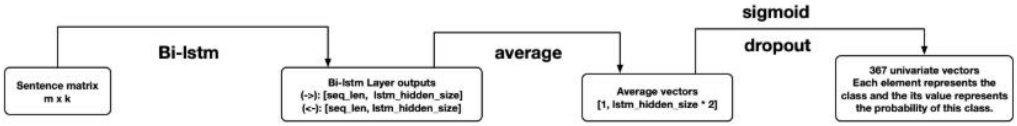


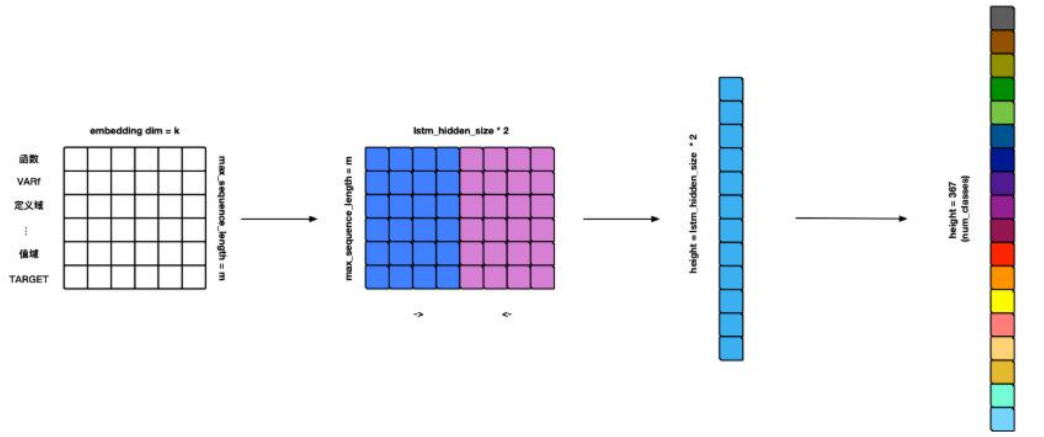
TextCNN



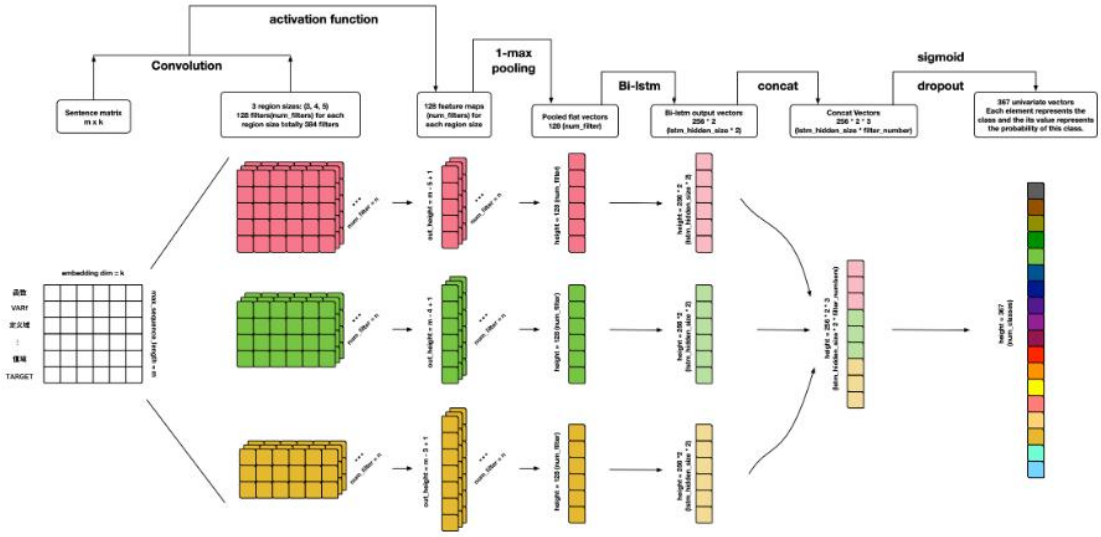
TextRNN

尚学堂·瓦战程序员

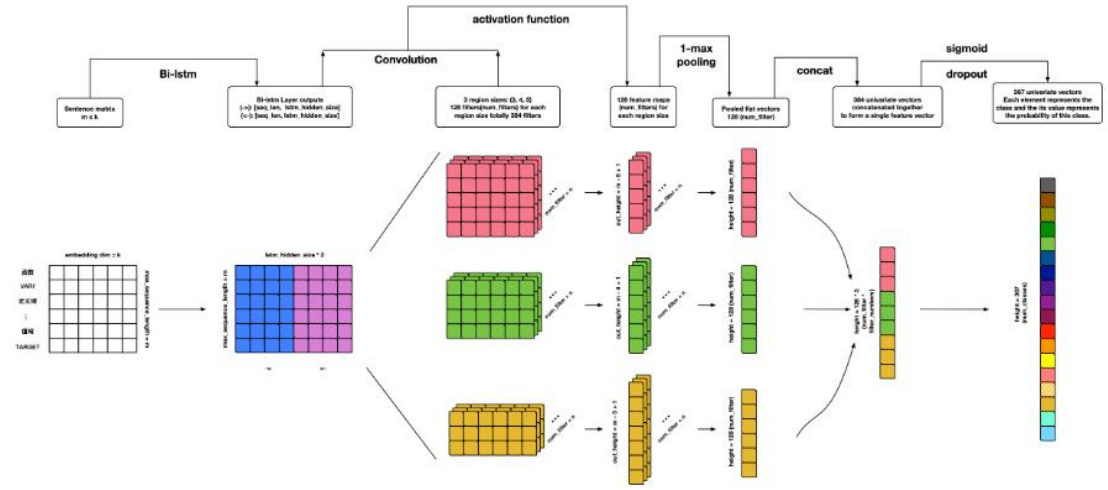




TextCRNN



TextRCNN



Attention

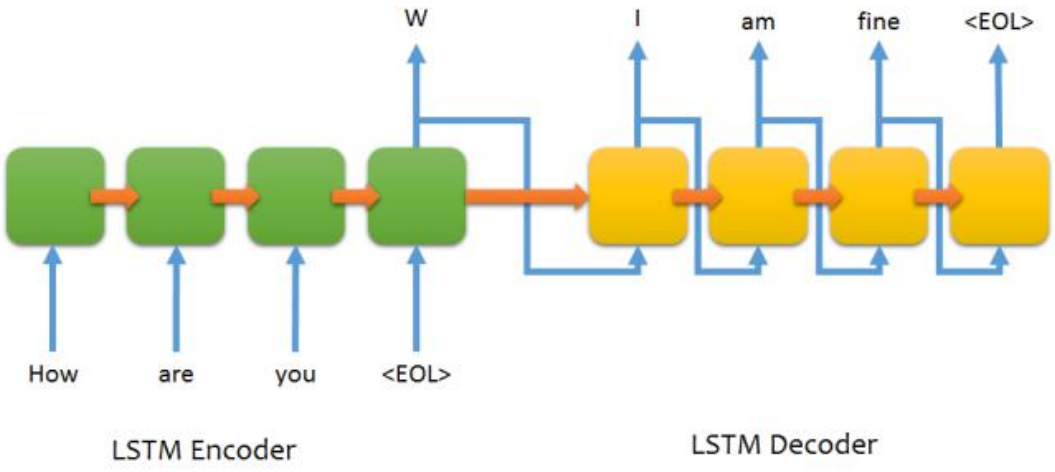
Encoder-Decoder 框架

要了解深度学习中的注意力模型，就不得不先谈 Encoder-Decoder 框架，因为目前大多数注意力模型附着在 Encoder-Decoder 框架下，当然，其实注意力模型可以看作一种通用的思想，本身并不依赖于特定框架，这点需要注意。

Encoder-Decoder 框架可以看作是一种深度学习领域的研究模式 ,应用场景异常广泛。下图是文本处理领域里常用的 Encoder-Decoder 框架最抽象的一种表示。



尚学堂·瓦战程序员



文本处理领域的 Encoder-Decoder 框架可以这么直观地去理解：可以把它看作适合处理由一个句子（或篇章）生成另外一个句子（或篇章）的通用处理模型。对于句子对<Source,Target>，我们的目标是给定输入句子 Source，期待通过 Encoder-Decoder 框架来生成目标句子 Target。Source 和 Target 可以是同一种语言，也可以是两种不同的语言。而 Source 和 Target 分别由各自的单词序列构成：

$$\mathbf{Source} = \langle x_1, x_2 \ldots x_m \rangle$$

$$\mathbf{Target} = \langle y_1, y_2 \ldots y_n \rangle$$

Encoder 顾名思义就是对输入句子 Source 进行编码，将输入句子通过非线性变换转化为中间语义表示 C：

$$C = \mathcal{F}(x_1, x_2 \ldots x_m)$$

对于解码器 Decoder 来说，其任务是根据句子 Source 的中间语义表示 C 和之前已经生成的历史信息

$y_1, y_2 \ldots\ldots y_{i-1}$

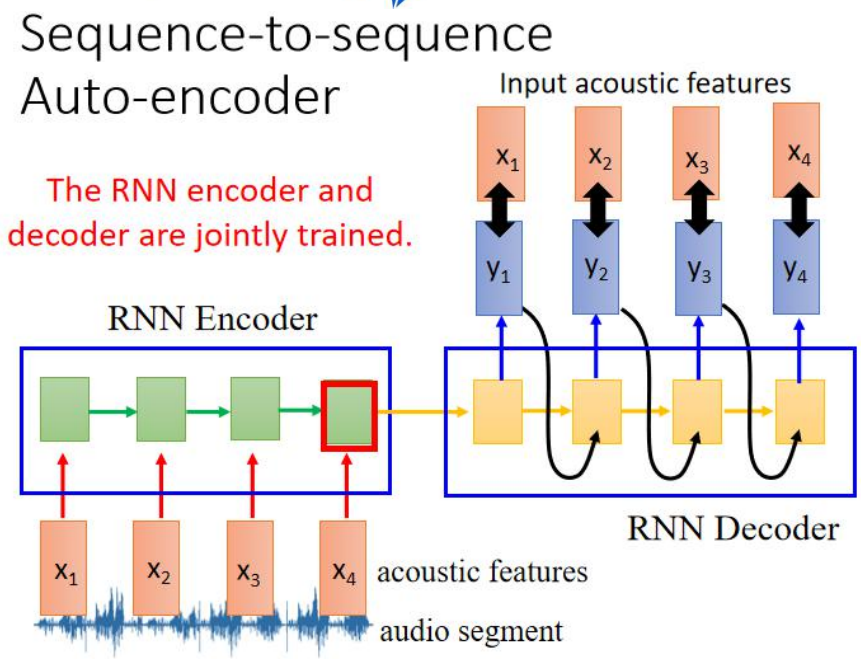
来生成 i 时刻要生成的单词

$$y_i = \mathcal{G}(C, y_1, y_2 \ldots y_{i-1})$$

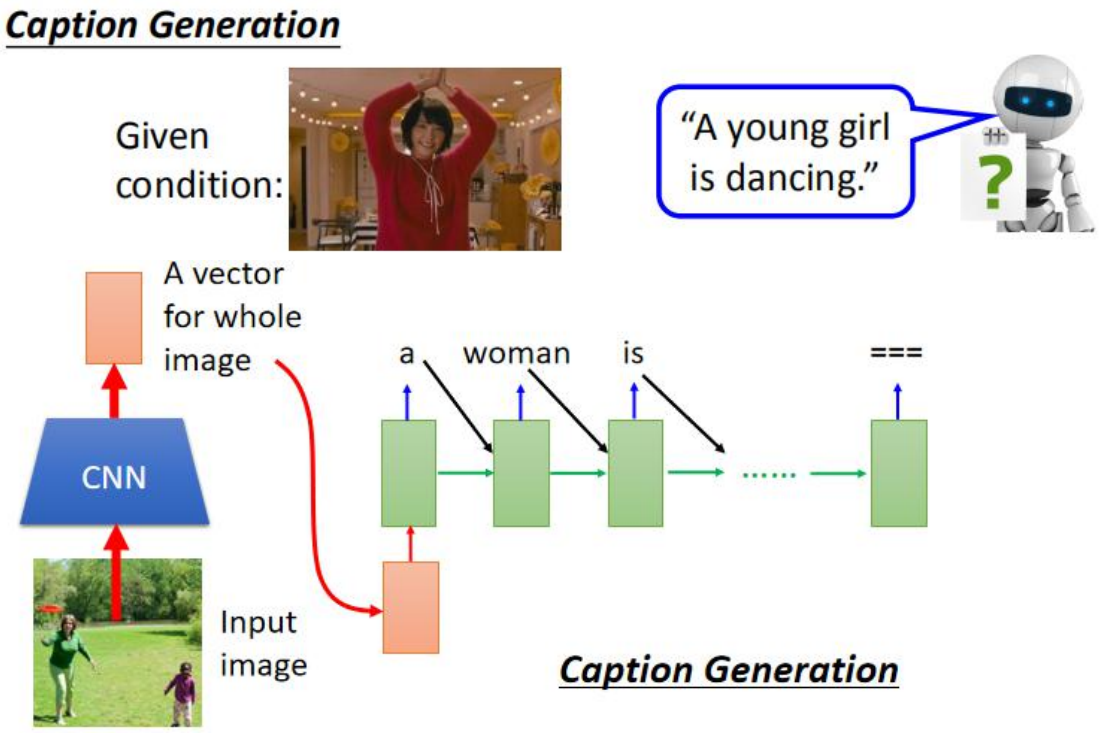
每个 yi 都依次这么产生，那么看起来就是整个系统根据输入句子 Source 生成了目标句子 Target。如果 Source 是中文句子，Target 是英文句子，那么这就是解决机器翻译问题的 Encoder-Decoder 框架；如果 Source 是一篇文章，Target 是概括性的几句描述语句，那么这是文本摘要的 Encoder-Decoder 框架；如果 Source 是一句问句，Target 是一句回答，那么这时问答系统或者对话机器人的 Encoder-Decoder 框架。由此可见，在文本处理领域，Encoder-Decoder 的应用领域相当广泛。

Encoder-Decoder 框架不仅仅在文本领域广泛使用，在语音识别、图像处理等领域也经常使用。比如对于语音识别来说，上图所示的框架完全适用，区别无非是 Encoder 部分的输入是语音流，输出是对应的文本信息；

尚学堂·瓦战程序员

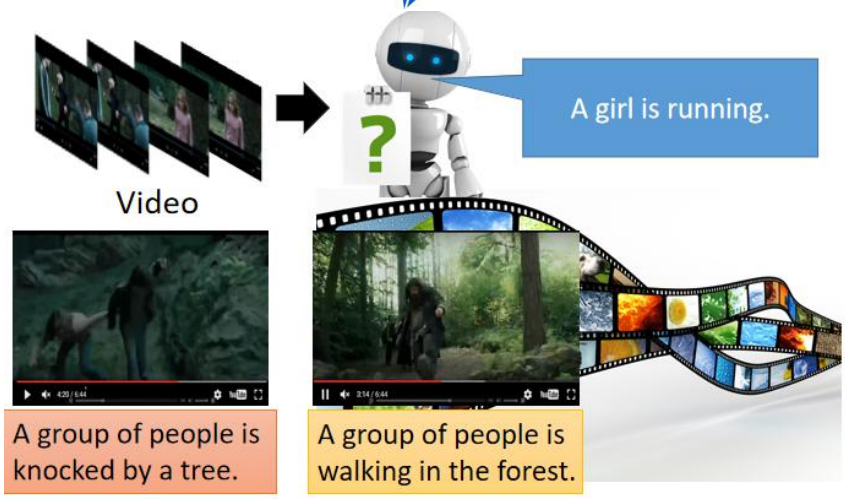


而对于“图像描述”任务来说，Encoder 部分的输入是一副图片，Decoder 的输出是能够描述图片语义内容的一句描述语。



一般而言，文本处理和语音识别的 Encoder 部分通常采用 RNN 模型，图像处理的 Encoder 一般采用 CNN 模型。

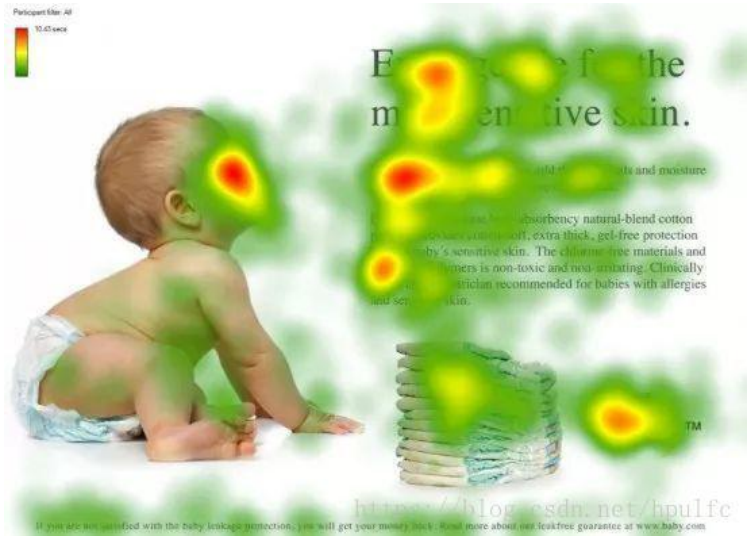
尚学堂·瓦战程序员



Attention 注意力机制

注意力模型最近几年在深度学习各个领域被广泛使用，无论是图像处理、语音识别还是自然语言处理的各种不同类型的任务中，都很容易遇到注意力模型的身影。所以，了解注意力机制的工作原理对于关注深度学习技术发展的技术人员来说有很大的必要。

从注意力模型的命名方式看，很明显其借鉴了人类的注意力机制，因此，我们首先简单介绍人类视觉的选择性注意力机制。



视觉注意力机制是人类视觉所特有的大脑信号处理机制。人类视觉通过快速扫描全局图像，获得需要重点关注的目标区域，也就是一般所说的注意力焦点，而后对这一区域投入更多注意力资源，以获取更多所需要关注目标的细节信息，而抑制其他无用信息。

这是人类利用有限的注意力资源从大量信息中快速筛选出高价值信息的手段，是人类在长期进化中形成的一种生存机制，人类视觉注意力机制极大地提高了视觉信息处理的效率与准确性。

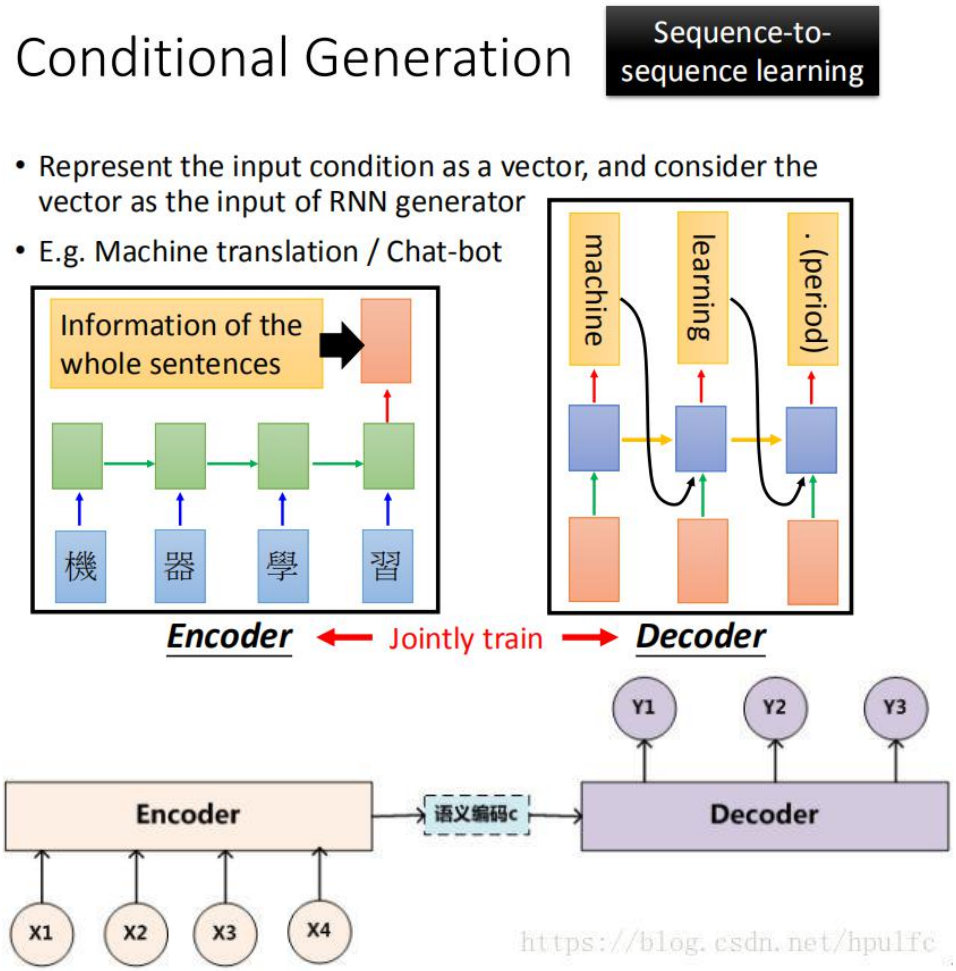
上图形象化展示了人类在看到一副图像时是如何高效分配有限的注意力资源的，其中红

尚学堂·瓦战程序员

色区域表明视觉系统更关注的目标，很明显对于图中所示的场景，人们会把注意力更多投入到人的脸部，文本的标题以及文章首句等位置。

深度学习中的注意力机制从本质上讲和人类的选择性视觉注意力机制类似，核心目标也是从众多信息中选择出对当前任务目标更关键的信息。

引入 Attention 机制



图中展示的 Encoder-Decoder 框架是没有体现出“注意力模型”的，所以可以把它看作是注意力不集中的分心模型。为什么说它注意力不集中呢？请观察下目标句子 Target 中每个单词的生成过程如下：

$$y_1 = f(C)$$
$$y_2 = f(C, y_1)$$
$$y_3 = f(C, y_1, y_2)$$

其中 f 是 Decoder 的非线性变换函数。从这里可以看出，在生成目标句子的单词时，不论生成哪个单词，它们使用的输入句子 Source 的语义编码 C 都是一样的，没有任何区别。而语义编码 C 是由句子 Source 的每个单词经过 Encoder 编码产生的，这意味着不论是生成哪个单词，y1,y2 还是 y3，其实句子 Source 中任意单词对生成某个目标单词 yi 来说影

尚学堂·瓦战程序员

响力都是相同的，这是为何说这个模型没有体现出注意力的缘由。这类似于人类看到眼前的画面，但是眼中却没有注意焦点一样。

如果拿机器翻译来解释这个分心模型的 Encoder-Decoder 框架更好理解，比如输入的是英文句子：Tom chase Jerry，Encoder-Decoder 框架逐步生成中文单词：“汤姆”，“追逐”，“杰瑞”。

在翻译“杰瑞”这个中文单词的时候，分心模型里面的每个英文单词对于翻译目标单词“杰瑞”贡献是相同的，很明显这里不太合理，显然“Jerry”对于翻译成“杰瑞”更重要，但是分心模型是无法体现这一点的，这就是为何说它没有引入注意力的原因。

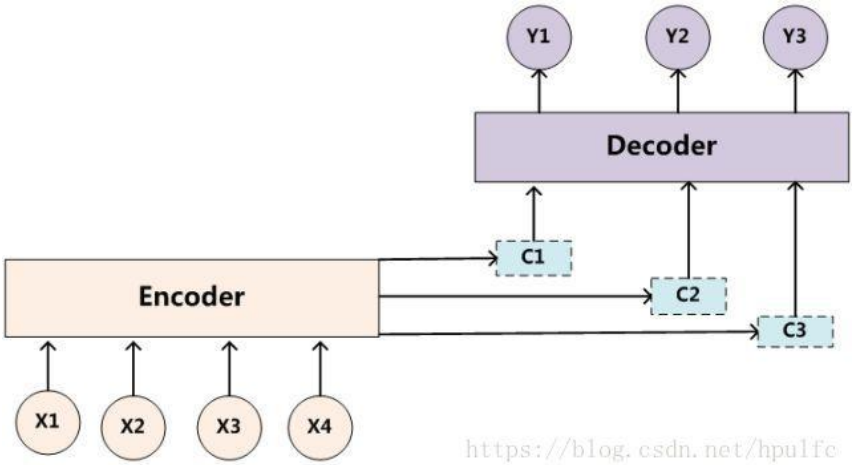
没有引入注意力的模型在输入句子比较短的时候问题不大，但是如果输入句子比较长，此时所有语义完全通过一个中间语义向量来表示，单词自身的信息已经消失，可想而知会丢失很多细节信息，这也是为何要引入注意力模型的重要原因。

上面的例子中，如果引入 Attention 模型的话，应该在翻译“杰瑞”的时候，体现出英文单词对于翻译当前中文单词不同的影响程度，比如给出类似下面一个概率分布值：

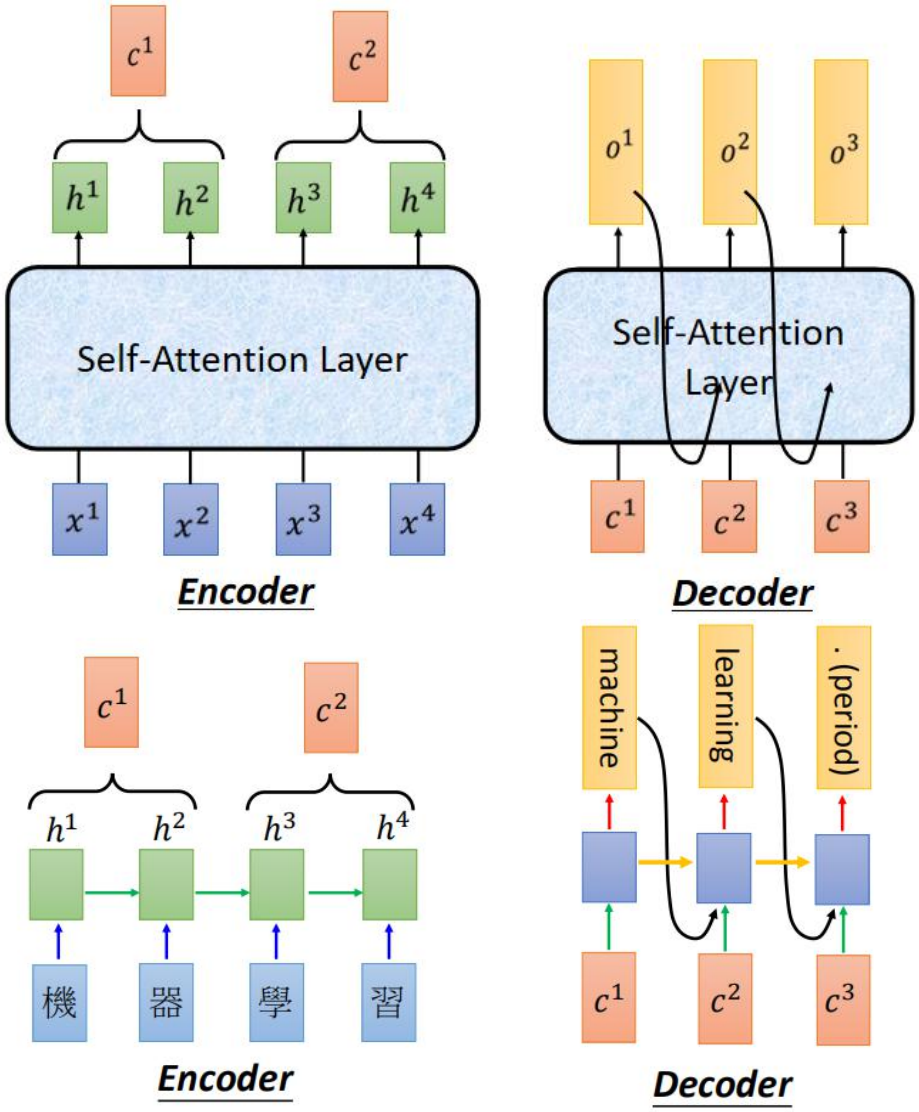
（Tom,0.3）(Chase,0.2) (Jerry,0.5)

每个英文单词的概率代表了翻译当前单词“杰瑞”时，注意力分配模型分配给不同英文单词的注意力大小。这对于正确翻译目标语单词肯定是有帮助的，因为引入了新的信息。

同理，目标句子中的每个单词都应该学会其对应的源语句子中单词的注意力分配概率信息。这意味着在生成每个单词 yi 的时候，原先都是相同的中间语义表示 C 会被替换成根据当前生成单词而不断变化的 Ci。理解 Attention 模型的关键就是这里，即由固定的中间语义表示 C 换成了根据当前输出单词来调整成加入注意力模型的变化的 Ci。增加了注意力模型的 Encoder-Decoder 框架理解起来如下图所示。



https://blog.csdn.net/hpulfc



即生成目标句子单词的过程成了下面的形式：

$$y_1 = f1(C_1)$$
$$y_2 = f1(C_2, y_1)$$
$$y_3 = f1(C_3, y_1, y_2)$$

而每个 Ci 可能对应着不同的源语句子单词的注意力分配概率分布，比如对于上面的英汉翻译来说，其对应的信息可能如下：

$$C_{汤姆} = g(0.6 * f2("Tom"), 0.2 * f2(Chase), 0.2 * f2("Jerry"))$$

$$C_{追逐} = g(0.2 * f2("Tom"), 0.7 * f2(Chase), 0.1 * f2("Jerry"))$$

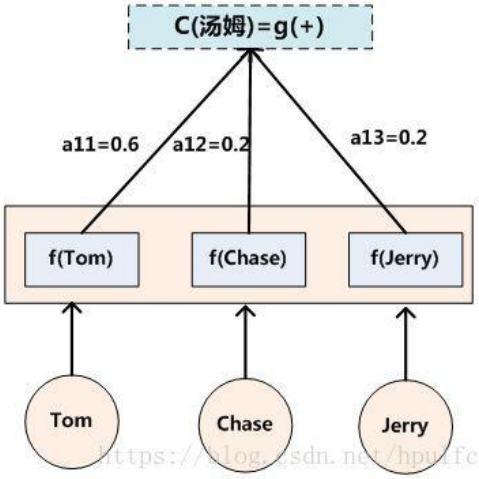
$$C_{杰瑞} = g(0.3 * f2("Tom"), 0.2 * f2(Chase), 0.5 * f2("Jerry"))$$

其中，f2 函数代表 Encoder 对输入英文单词的某种变换函数，比如如果 Encoder 是用的 RNN 模型的话，这个 f2 函数的结果往往是某个时刻输入 xi 后隐层节点的状态值；g 代表 Encoder 根据单词的中间表示合成整个句子中间语义表示的变换函数，一般的做法中，g 函数就是对构成元素加权求和，即下列公式：

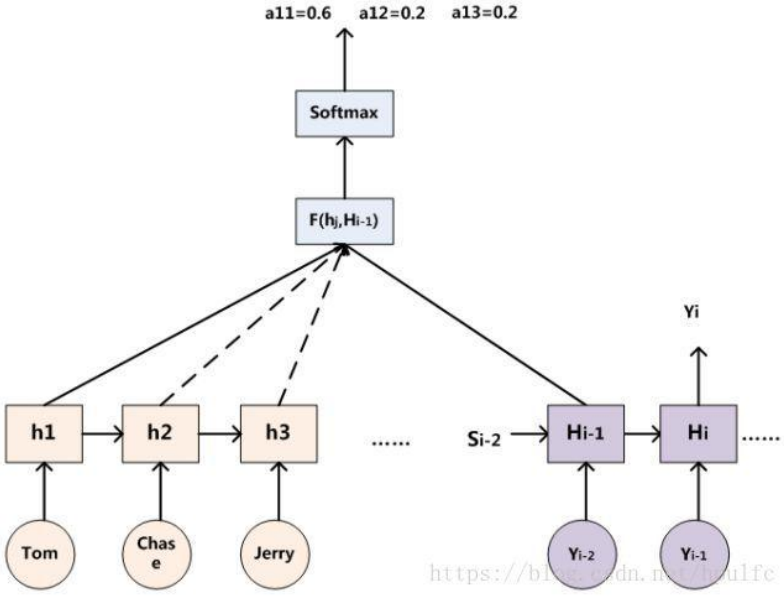
尚学堂·瓦战程序员

$$C_i = \sum_{j=1}^{L_x} a_{ij} h_j$$

其中，Lx 代表输入句子 Source 的长度，aij 代表在 Target 输出第 i 个单词时 Source 输入句子中第 j 个单词的注意力分配系数，而 hj 则是 Source 输入句子中第 j 个单词的语义编码。假设下标 i 就是上面例子所说的“ 汤姆” ，那么 Lx 就是 3，h1=f(“Tom”)，h2=f(“Chase”),h3=f(“Jerry”)分别是输入句子每个单词的语义编码，对应的注意力模型权值则分别是 0.6,0.2,0.2，所以 g 函数本质上就是个加权求和函数。如果形象表示的话，翻译中文单词“汤姆”的时候，数学公式对应的中间语义表示 Ci 的形成过程类似下图。

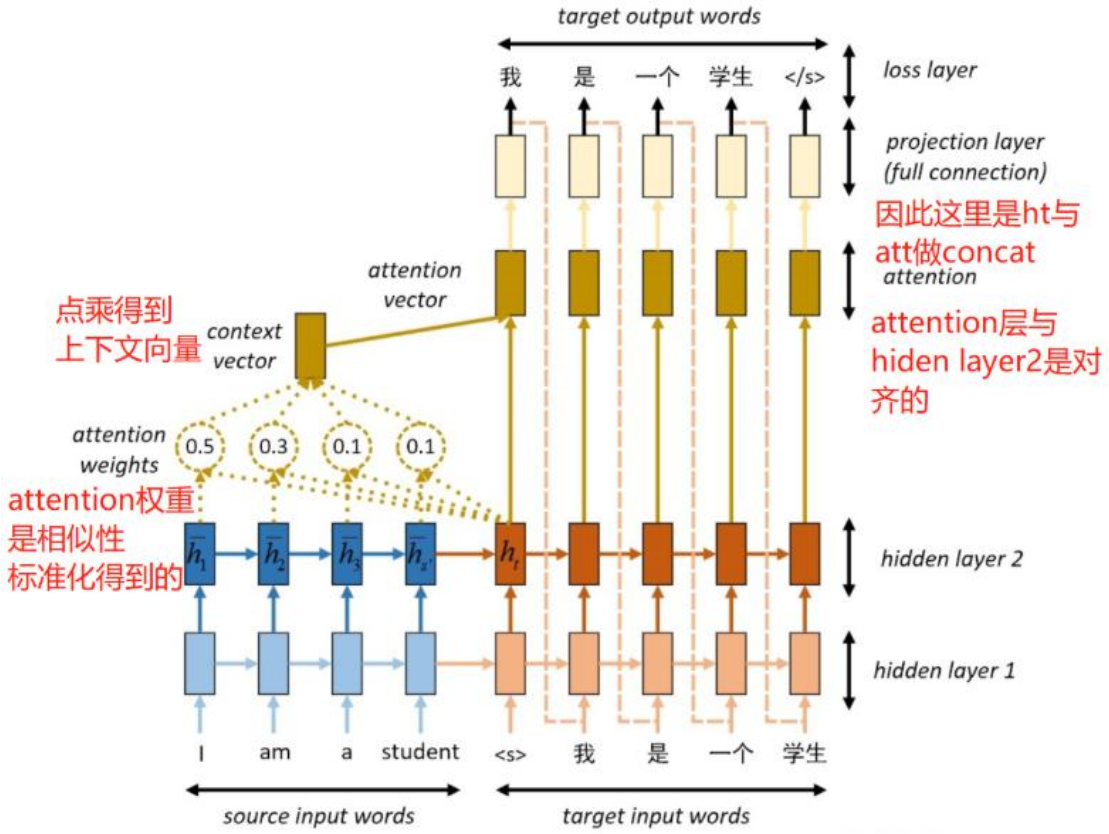


这里还有一个问题：生成目标句子某个单词，比如“汤姆”的时候，如何知道 Attention 模型所需要的输入句子单词注意力分配概率分布值呢？就是说“汤姆”对应的输入句子 Source 中各个单词的概率分布：(Tom,0.6)(Chase,0.2) (Jerry,0.2) 是如何得到的呢？
那么用下图可以较为便捷地说明注意力分配概率分布值的通用计算过程。



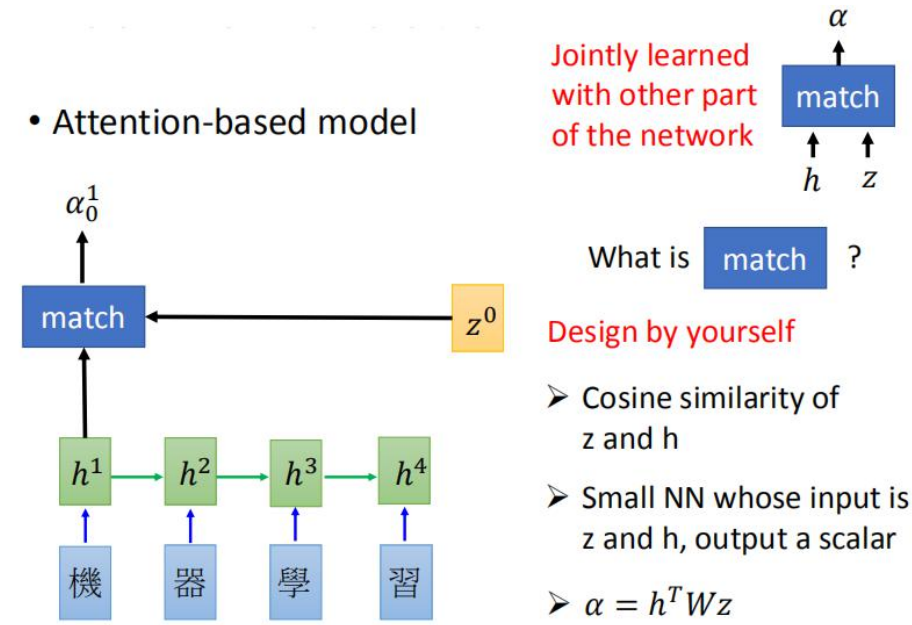
对于采用 RNN 的 Decoder 来说，在时刻 i，如果要生成 yi 单词，我们是可以知道 Target 在生成 Yi 之前的时刻 i-1 时，隐层节点 i-1 时刻的输出值 Hi-1 的，而我们的目的是要计算生成 Yi 时输入句子中的单词“Tom”、“Chase”、“Jerry”对 Yi 来说的注意力分配概率

分布 ,那么可以用 Target 输出句子 i-1 时刻的隐层节点状态 Hi-1 去一一和输入句子 Source 中每个单词对应的 RNN 隐层节点状态 hj 进行对比 ,即通过函数 F(hj,Hi-1)来获得目标单词 yi 和每个输入单词对应的对齐可能性，这个 F 函数在不同论文里可能会采取不同的方法，然后函数 F 的输出经过 Softmax 进行归一化就得到了符合概率分布取值区间的注意力分配概率分布数值。

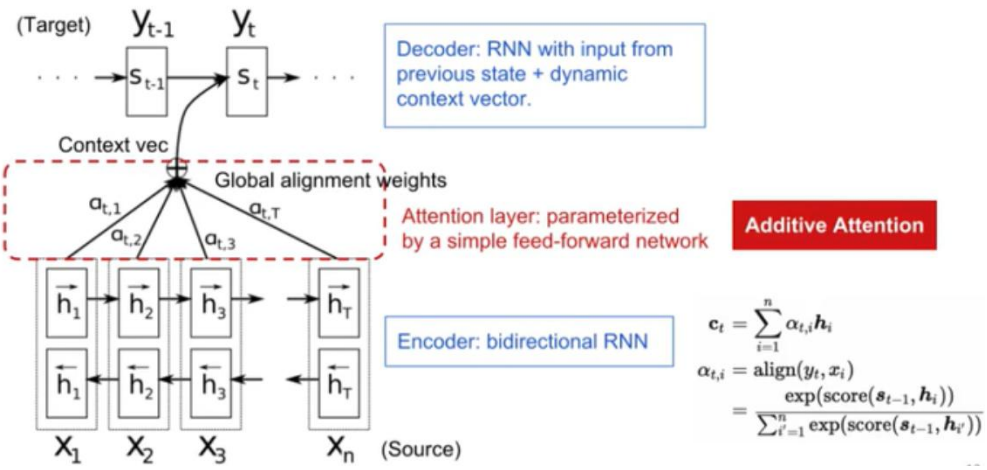


尚学堂·瓦战程序员

Attention 算法流程



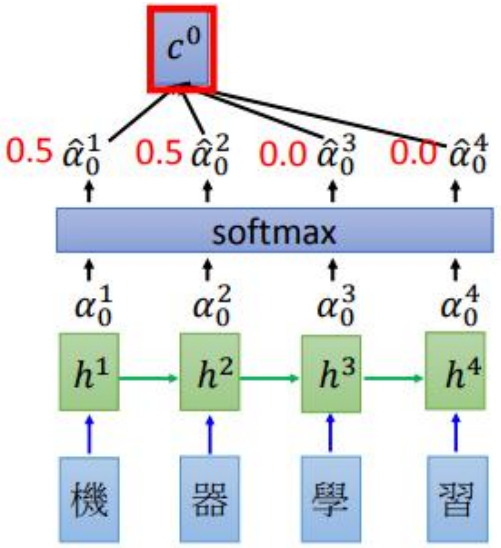
Bahdanau et al., 2015. Neural Machine Translation by Jointly Learning to Align and Translate



1) The current target hidden state is compared with all source states to derive attention weights

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \qquad \text{[Attention weights]} \qquad (1)$$

尚学堂·瓦战程序员



2) Based on the attention weights we compute a context vector as the weighted average of the source states.

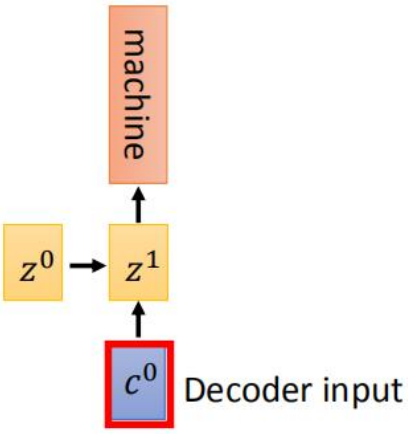
$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \qquad \text{[Context vector]} \qquad (2)$$

$$c^0 = \sum \hat{\alpha}_0^i h^i$$

$$= 0.5h^1 + 0.5h^2$$

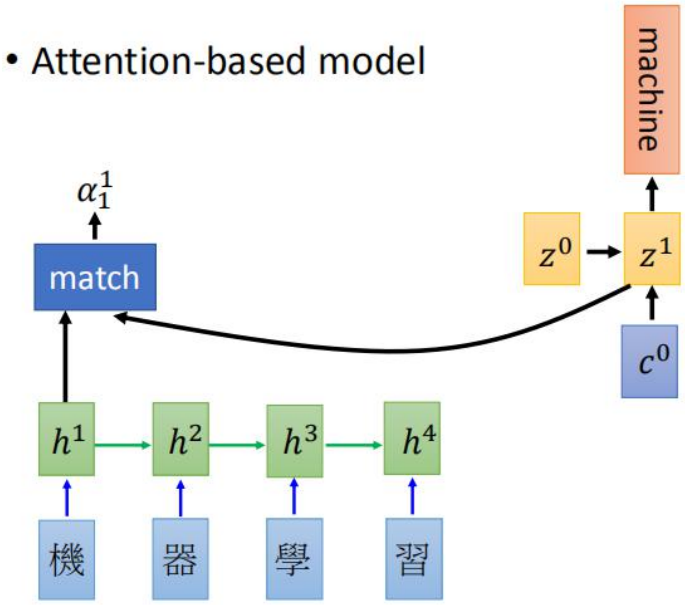
3) Combine the context vector with the current target hidden state to yield the final attention vector , 这里 at 就是 zt

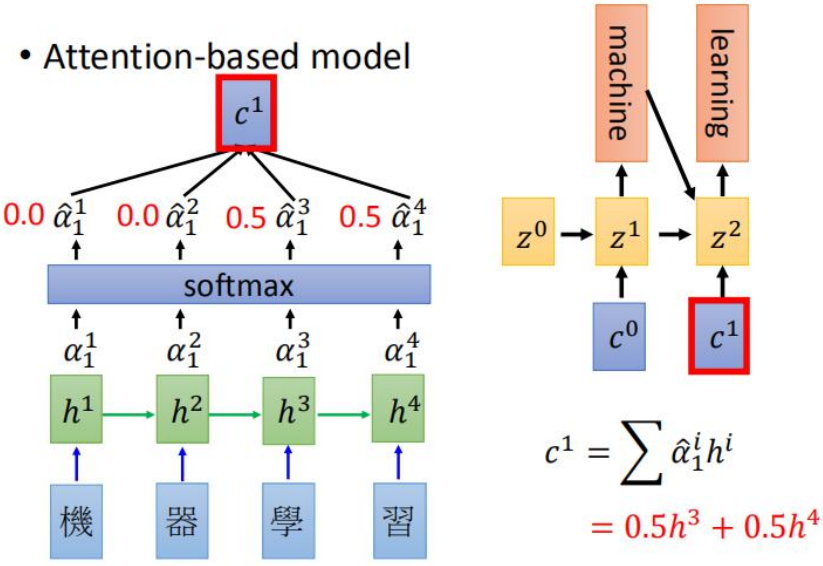
$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \qquad \text{[Attention vector]} \qquad (3)$$

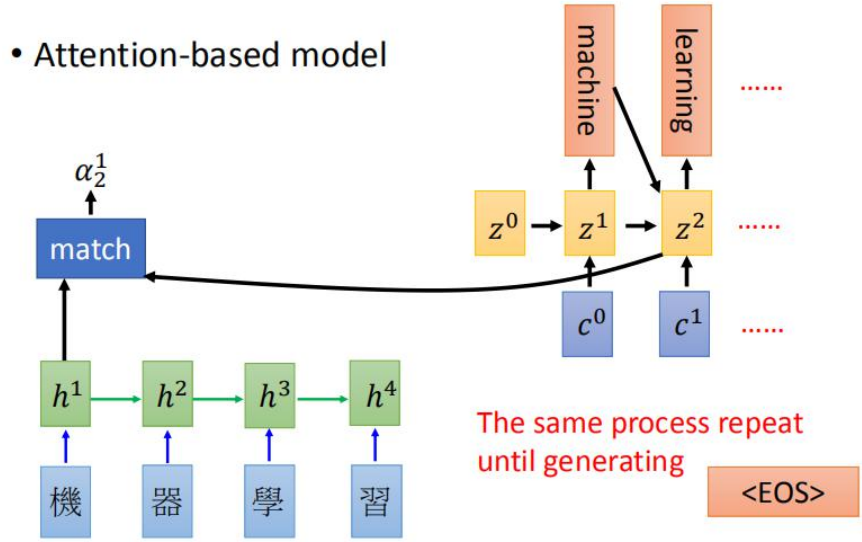


4) The attention vector is fed as an input to the next time step (input feeding).

尚学堂·瓦战程序员







其中第 1 步不同的 Attention 的方法就是 match 不同

尚学堂·瓦战程序员

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_t + \boldsymbol{W}_2 \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} \quad (4)$$

这样就可以把每个时间步重要的信息传给 decode 中 ,以上就是 Attention 机制的处理过程。其中 match 操作一般是求两个向量的相似度，通常有如下方法：

1）余弦相似度

2）一个简单的神经网络，输入为 h 和 w，输出为α

3）或者矩阵变换α=hTWz (Multiplicative attention，Luong et al., 2015)

在 tensorflow1.0 版本以后的 api seq2seq 库中，包含了两种 Attention 算法，他们的区别就是 match 操作的不同，因此也有人称他们为加法 Attention 和乘法 Attention，具体内容下：

1）BahdanauAttention：

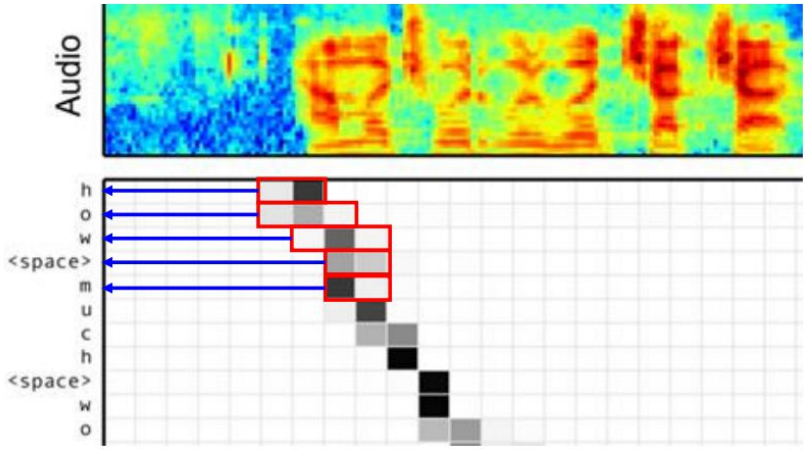
$$u^t = v^T \tanh(W_1 h + W_2 d_t)$$
$$a^t = softmax(u^t)$$
$$c^t = \sum_l^L a_l^t h_l$$

2）LuongAttention：

$$u^t = d_t W_1 h$$
$$a^t = softmax(u^t)$$
$$c^t = \sum_l^L a_l^t h_l$$

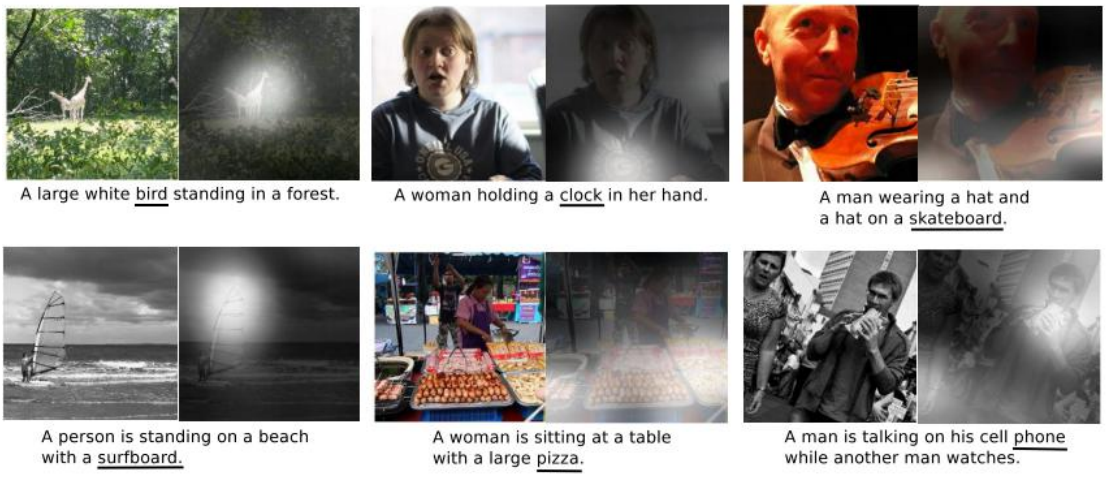
有上面两个公式可以看出，BahdanauAttention 和 LuongAttention 的区别就是在 match 过程中的计算方式不同，一个是将 decode 的状态与 encode 的输出求和，一个是求乘，所以才有了加法 Attention 和乘法 Attention 的叫法。

Speech Recognition



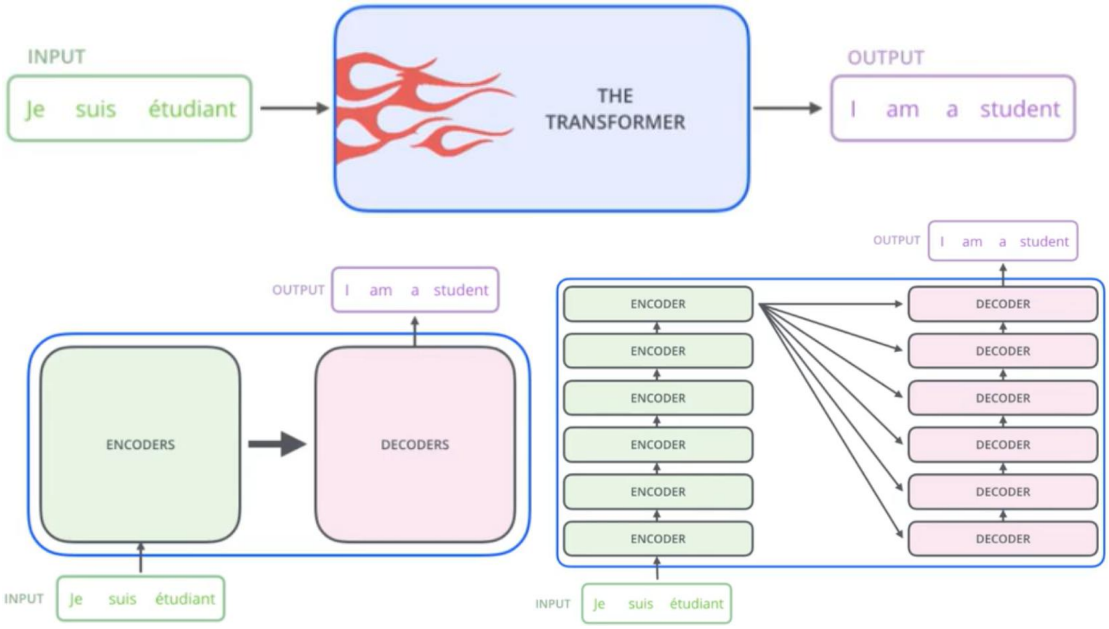
尚学堂·瓦战程序员

Image Caption Generation

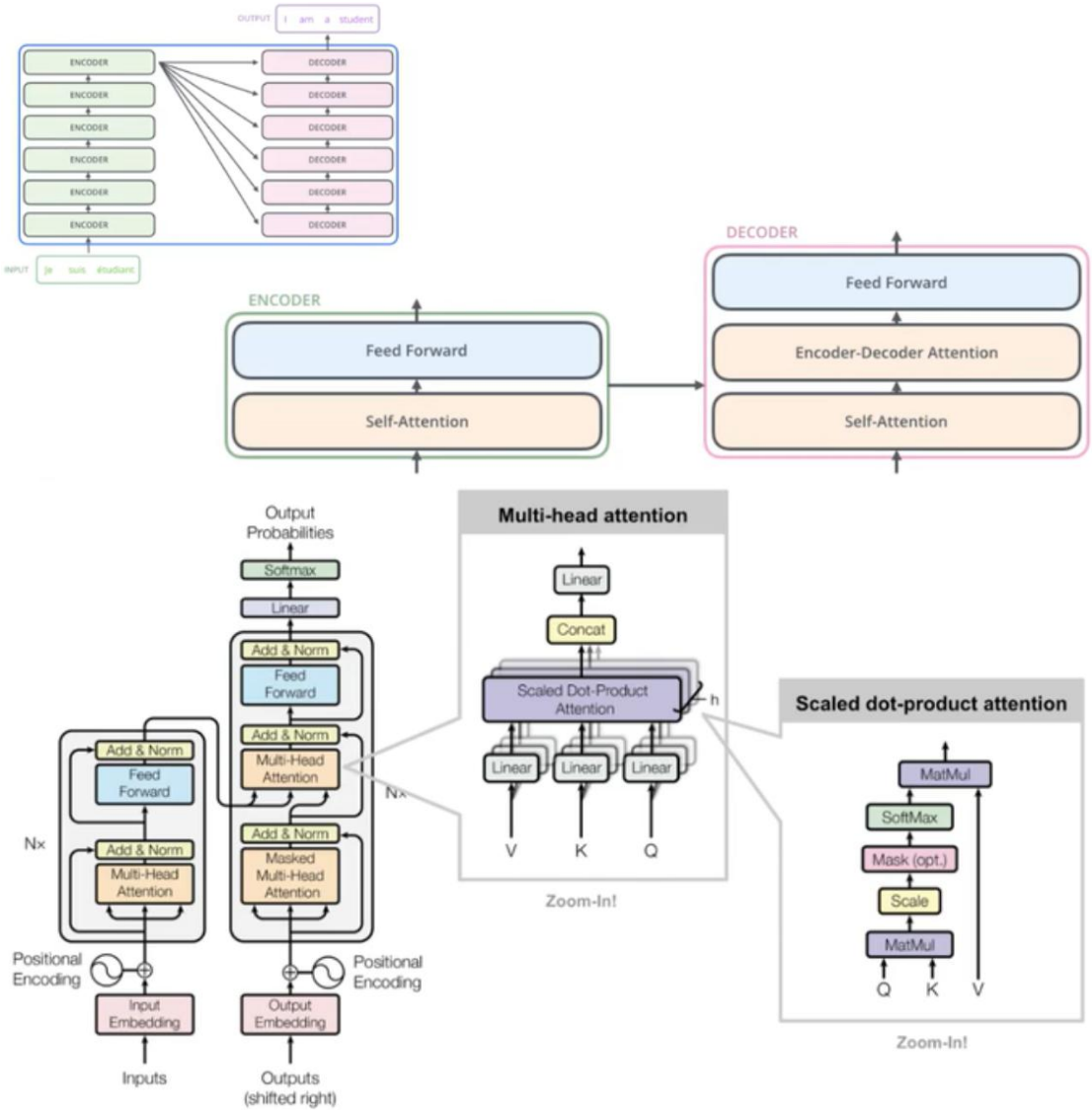


Transformer

没有用到 RNN 和 CNN 什么的！！！



尚学堂·瓦战程序员



Scaled dot product Attention 就是我们的 self-attention

Multi head Attention 就是 Scaled dot product Attention 的堆积

Transformer 就是 Multi head Attention 的堆积

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

$$\alpha_{t,i} = \mathbf{align}(y_t, x_i) = \frac{\exp(\mathbf{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\mathbf{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

抽象 attention

Q 就是 decoder 中 St-1 这个上一时刻的输出

K 就是 encoder 中多个隐变量 ht

V 就是 K

attention 中不就是计算的 Q 和 K 之间的相似度，QK，然后除以一个常量 dk，然后做一个 softmax，不就是我们之前的α，然后乘上 V 不就是得到 context vector Ct

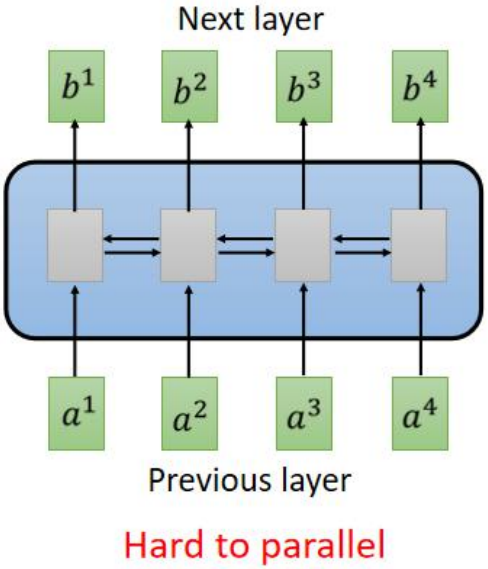
尚学堂·瓦战程序员

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

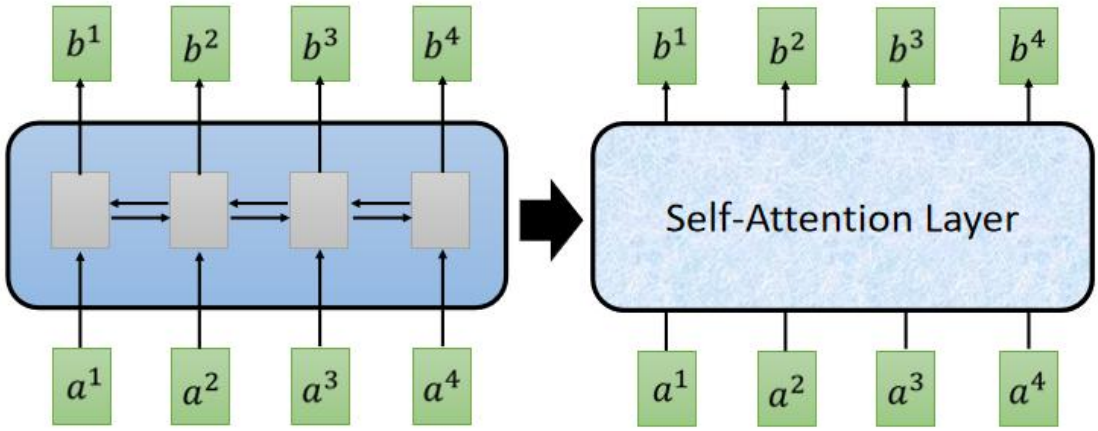
Self-Attention

attention 的特例



bi is obtained based on the whole input sequence.

b1 b2 b3 b4 can be parallelly computed.



You can try to replace any thing that has been done by RNN with self-attention.

尚学堂·瓦战程序员



q: query (to match others)

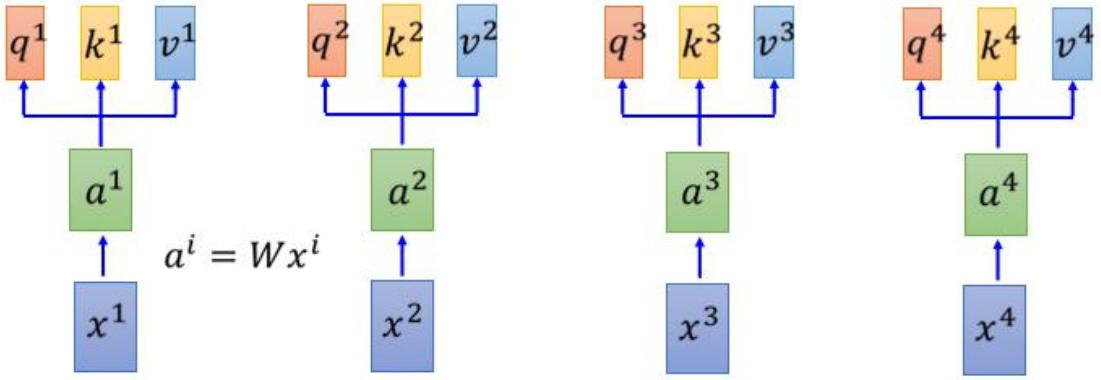
$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

$$v^i = W^v a^i$$

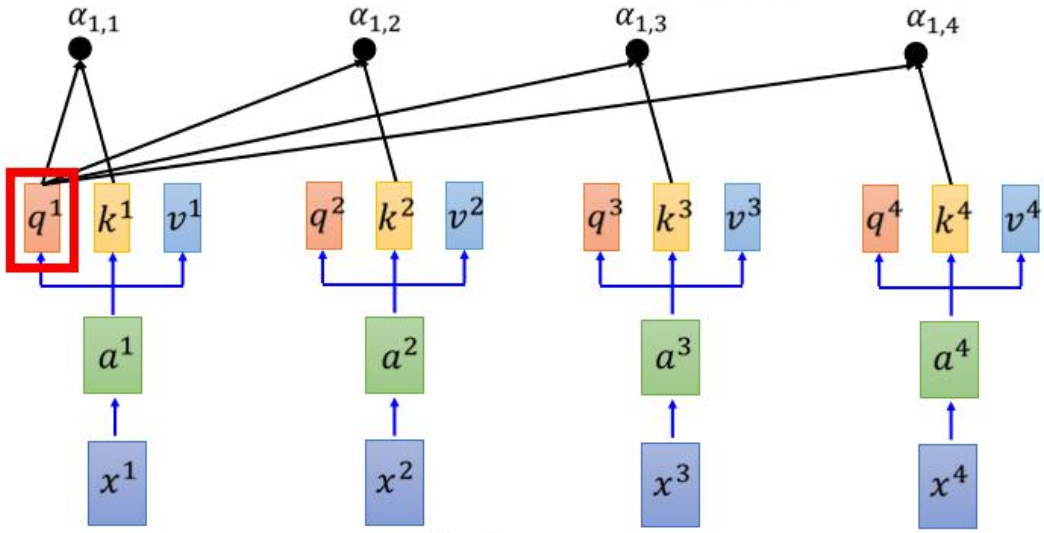


拿每個 query q 去對每個 key k 做 attention

d is the dim of q and k

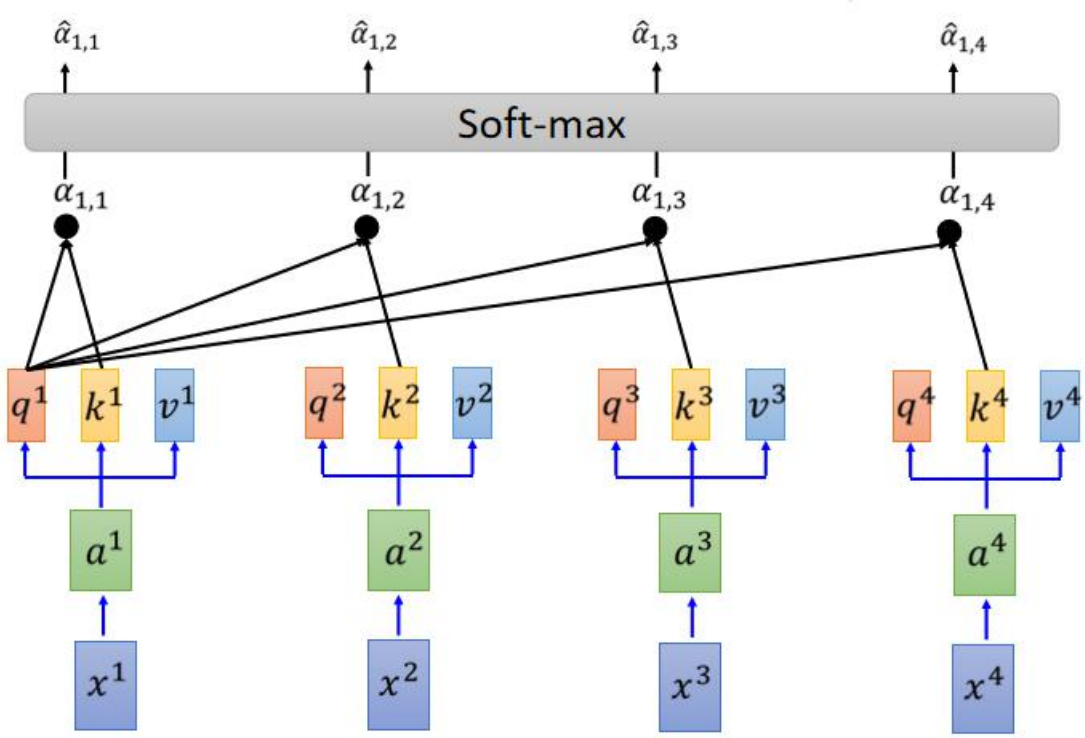
Scaled Dot-Product Attention: $\alpha_{1,i} = q^1 \cdot k^i / \sqrt{d}$

dot product



$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$

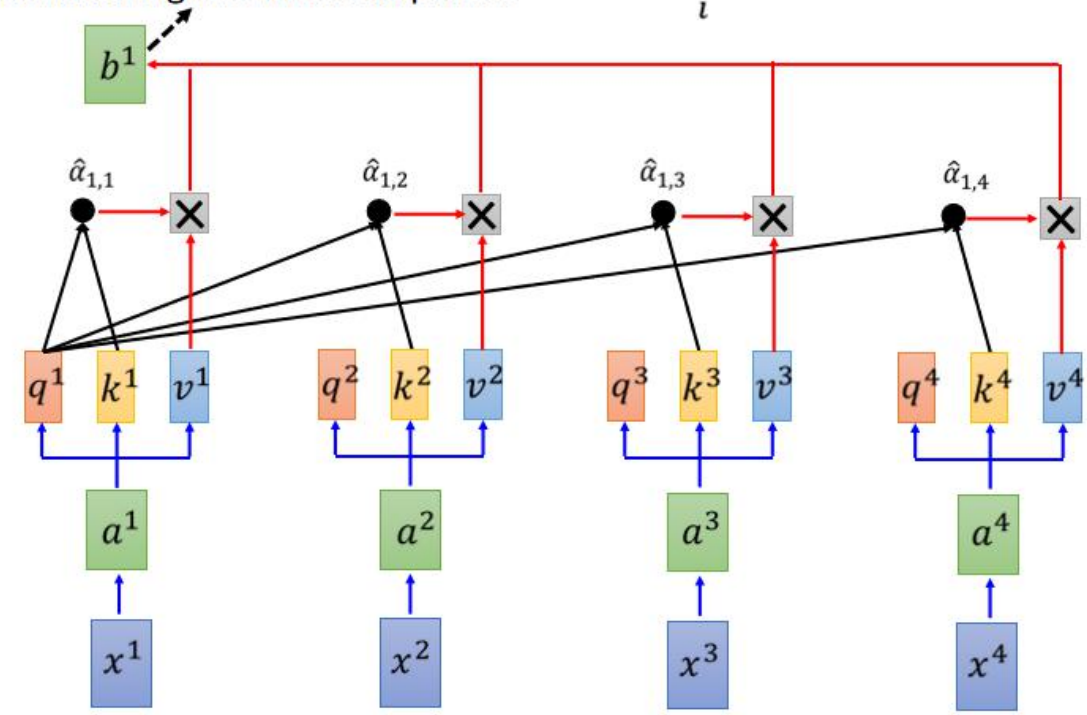
尚学堂·瓦战程序员



Self-attention

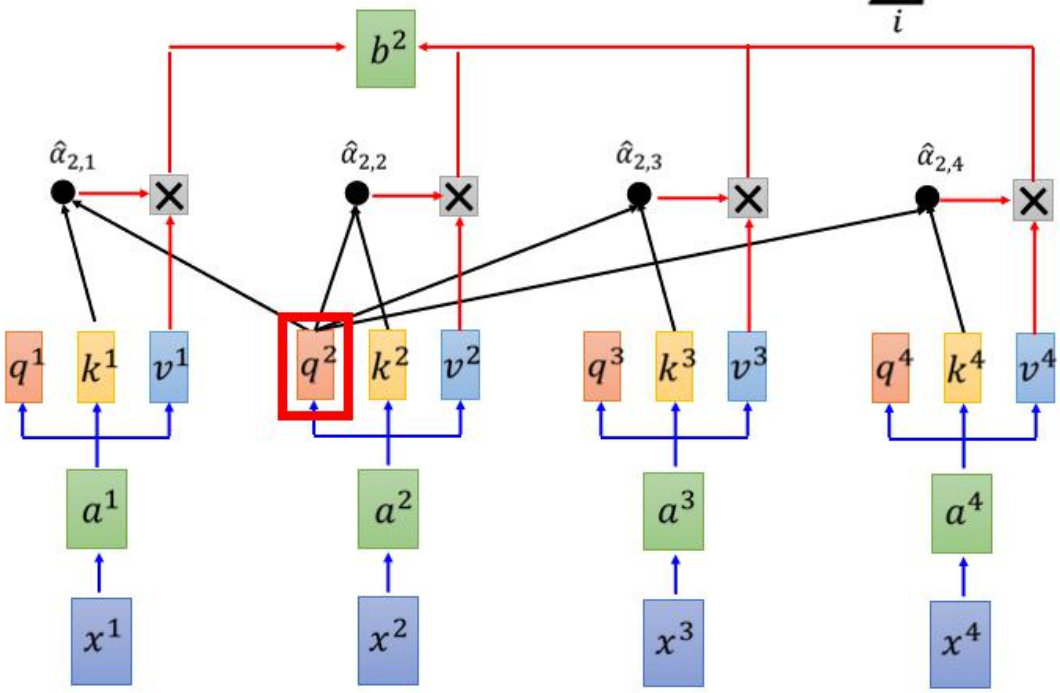
Considering the whole sequence

$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

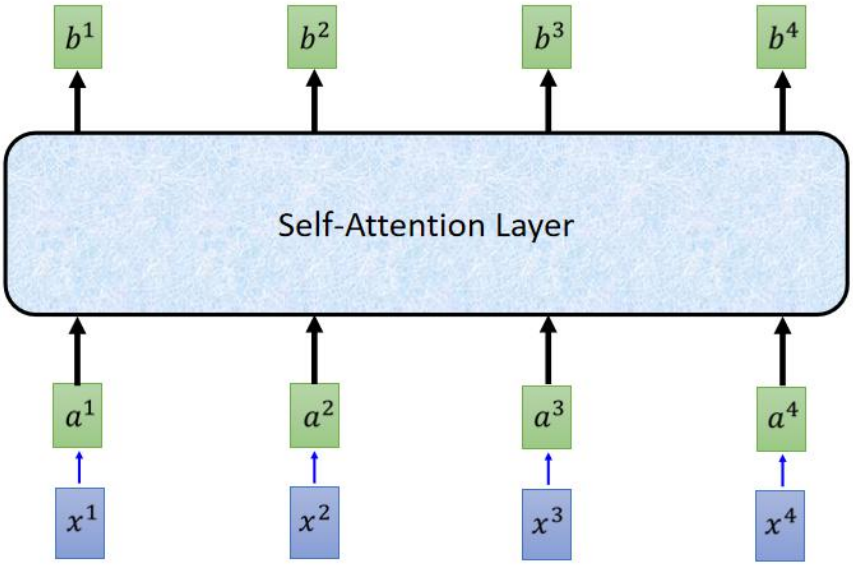


尚学堂·瓦战程序员

拿每個 query q 去對每個 key k 做 attention $b^2 = \sum_i \hat{\alpha}_{2,i} v^i$



b^1, b^2, b^3, b^4 can be parallelly computed.

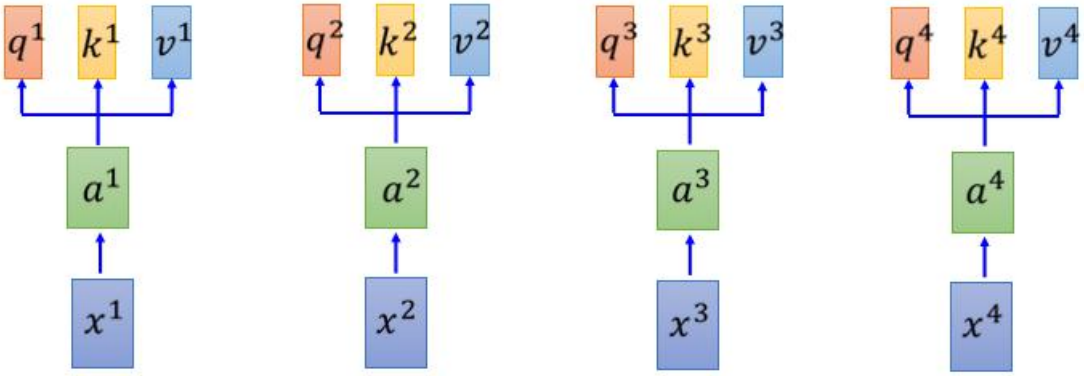


尚学堂·瓦战程序员

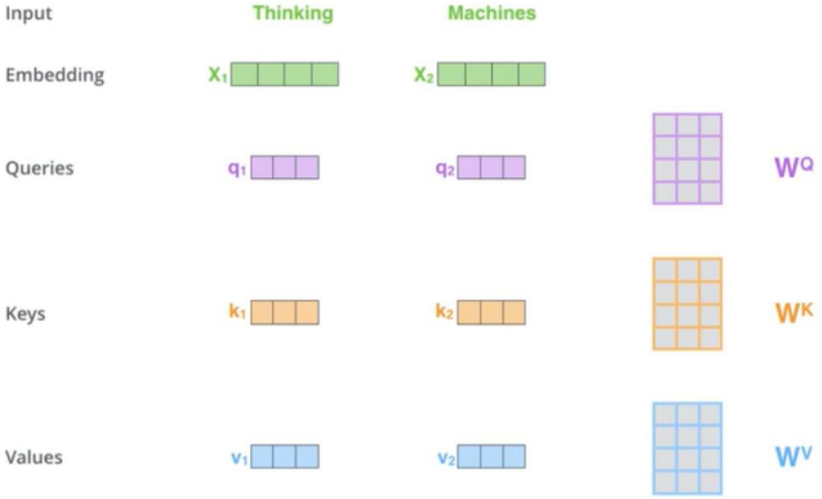
$$q^i = W^q a^i \qquad \begin{matrix} q^1 & q^2 & q^3 & q^4 \\ Q \end{matrix} = W^q \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ I \end{matrix}$$

$$k^i = W^k a^i \qquad \begin{matrix} k^1 & k^2 & k^3 & k^4 \\ K \end{matrix} = W^k \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ I \end{matrix}$$

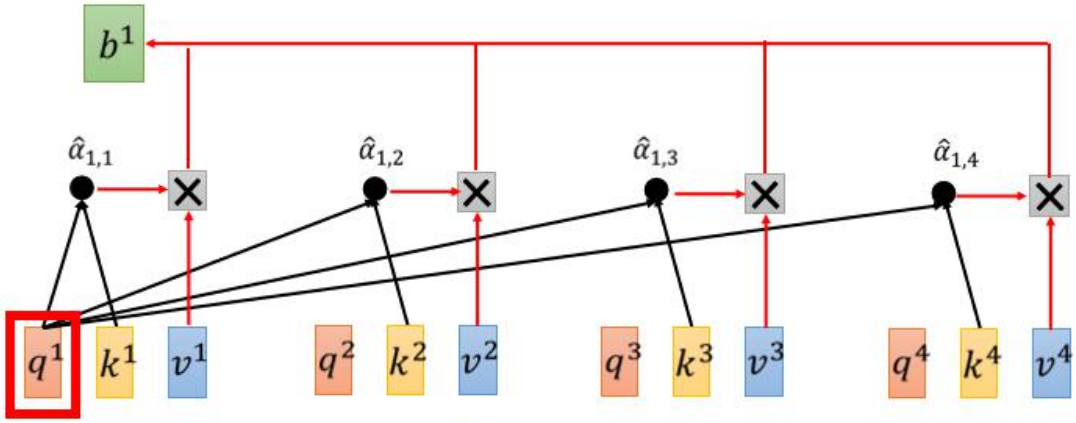
$$v^i = W^v a^i \qquad \begin{matrix} v^1 & v^2 & v^3 & v^4 \\ V \end{matrix} = W^v \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ I \end{matrix}$$



下图 X1 就是上面的 a1，乘 Wq 得以到 q1，以此类推



尚学堂·瓦战程序员



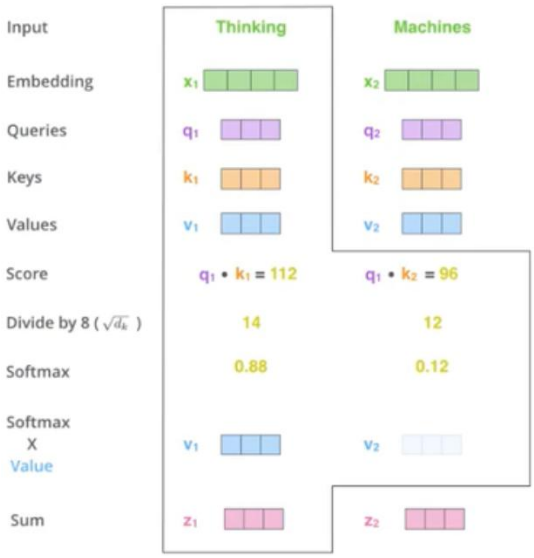
$$\alpha_{1,1} = k^1 q^1 \quad \alpha_{1,2} = k^2 q^1$$

$$\alpha_{1,3} = k^3 q^1 \quad \alpha_{1,4} = k^4 q^1$$

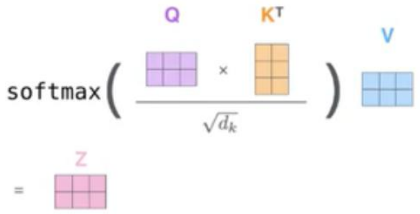
(ignore $\sqrt{d}$ for simplicity)

$$\begin{bmatrix}\alpha_{1,1}\\ \alpha_{1,2}\\ \alpha_{1,3}\\ \alpha_{1,4}\end{bmatrix} = \begin{bmatrix}k^1\\ k^2\\ k^3\\ k^4\end{bmatrix} q^1$$

下图的 z1 就是上图的 b1，依此类推



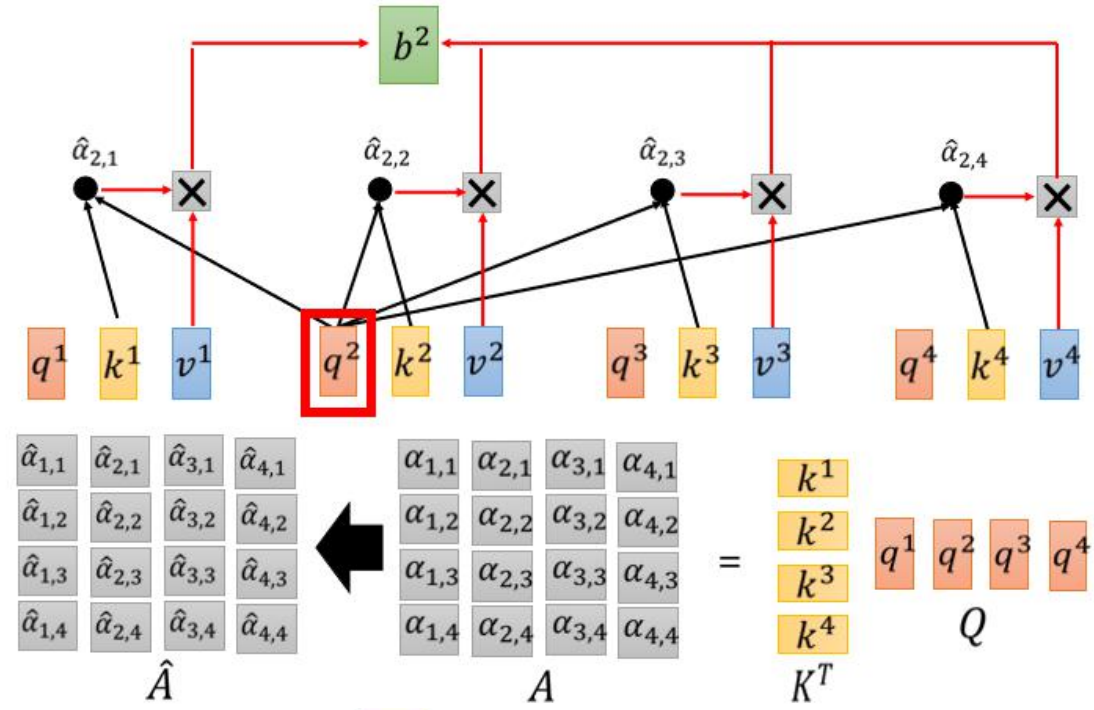
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

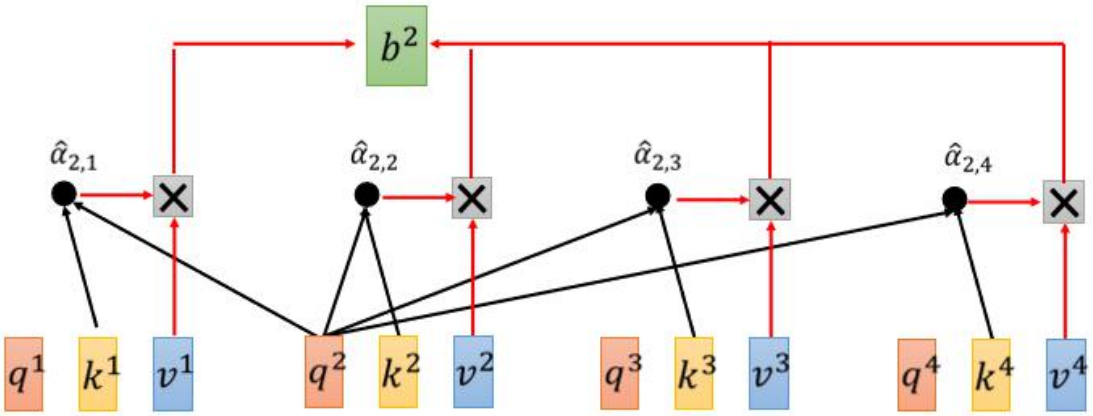


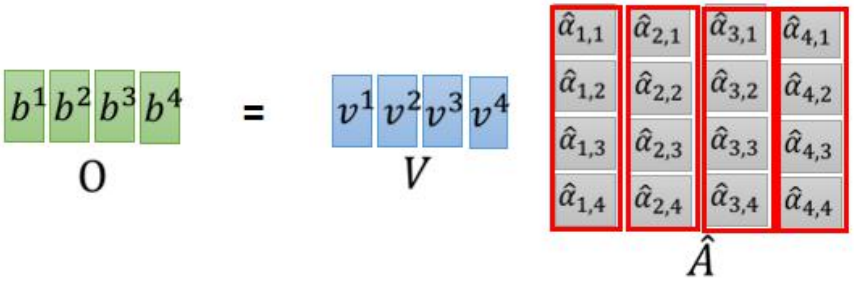
尚学堂·瓦战程序员

Self-attention

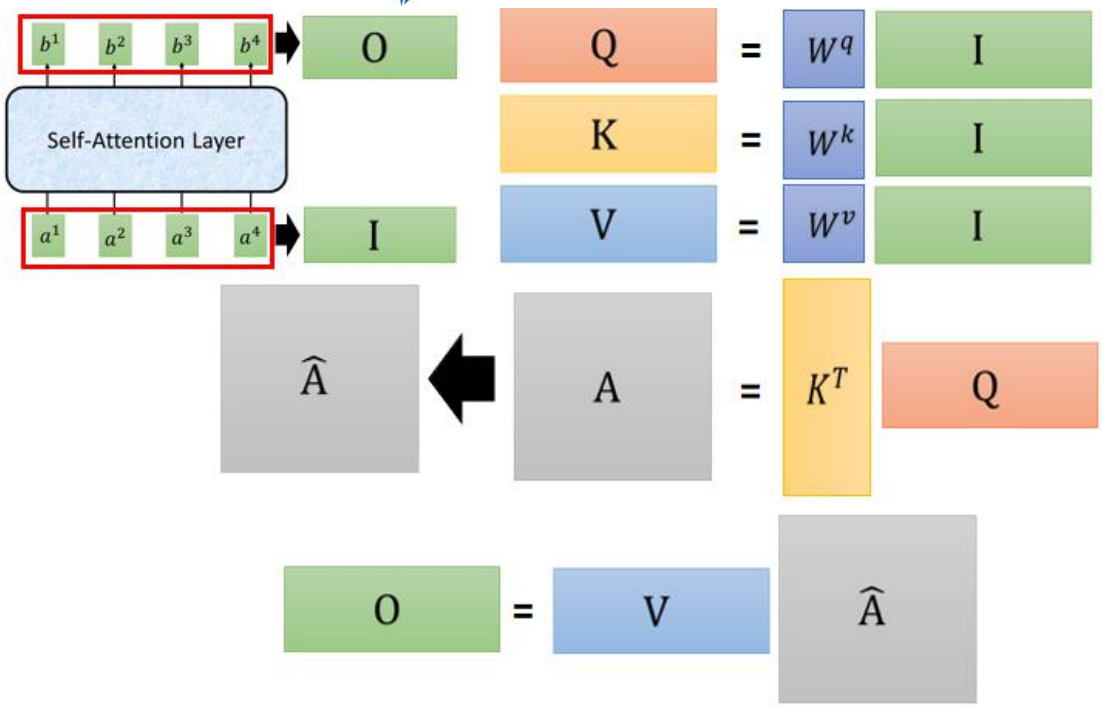
$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



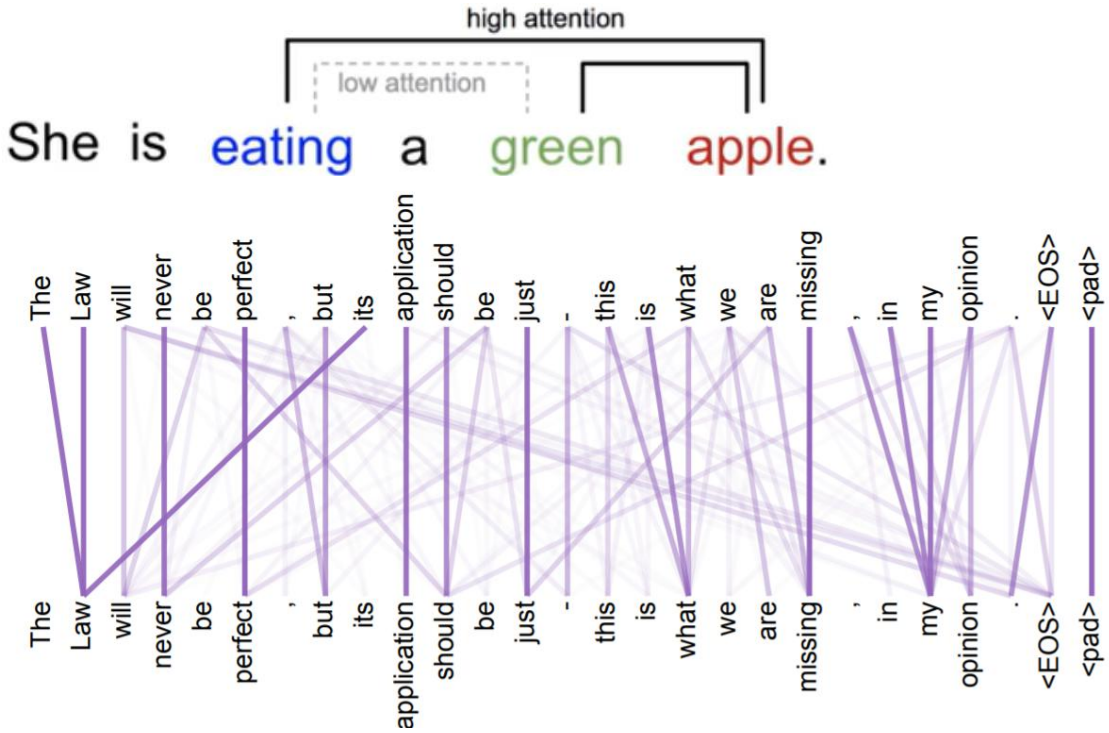




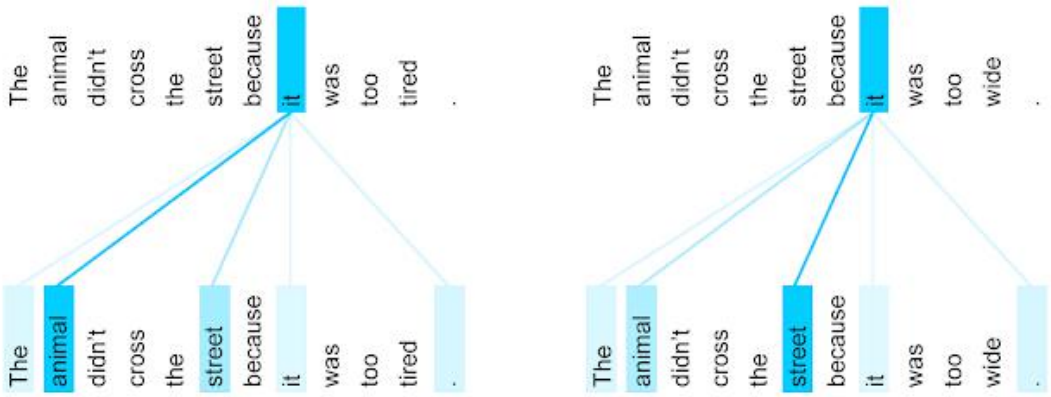
尚学堂·瓦战程序员



反正就是一堆矩陣乘法，用 GPU 可以加速！！！



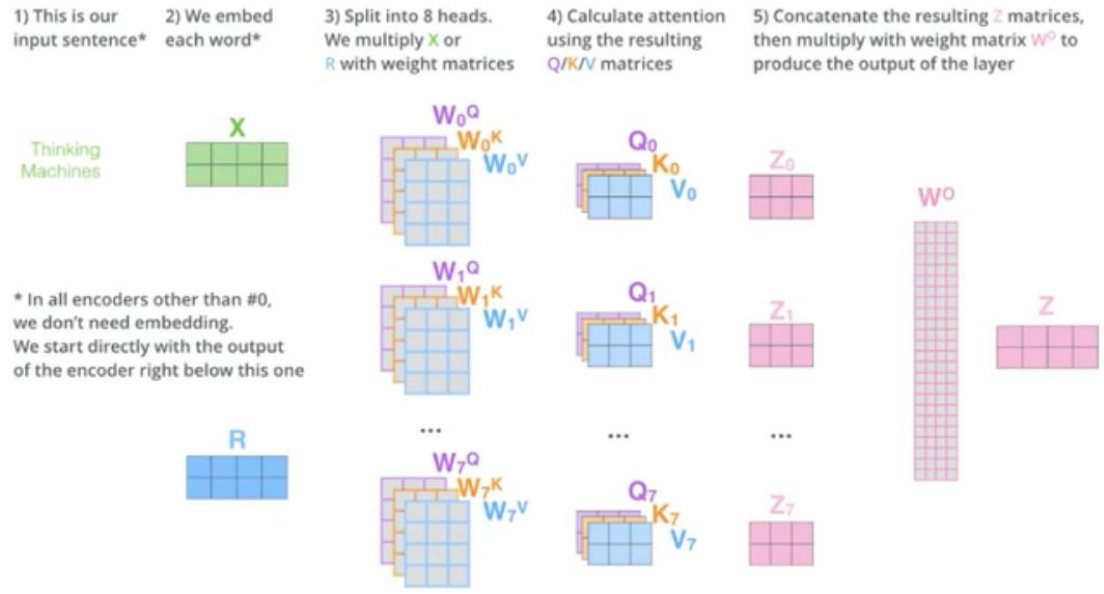
尚学堂·瓦战程序员



The encoder self-attention distribution for the word “it” from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).

self attention 就是 Q,K,V 来自于同一个序列

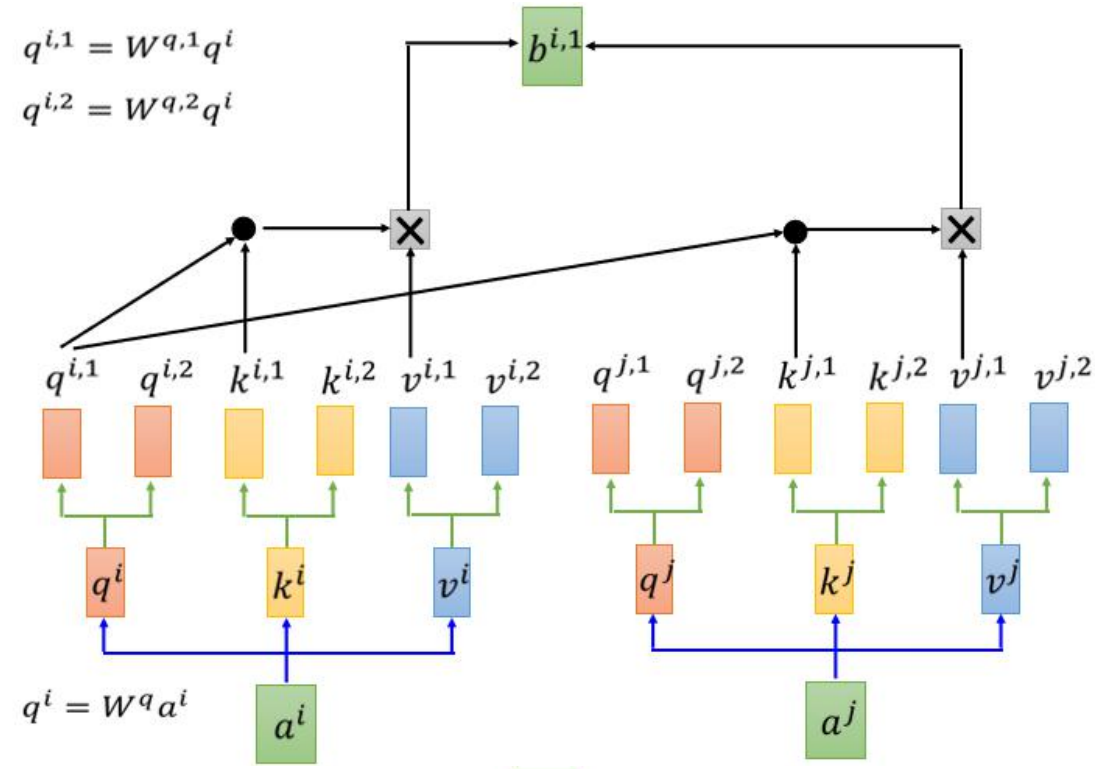
Multi head

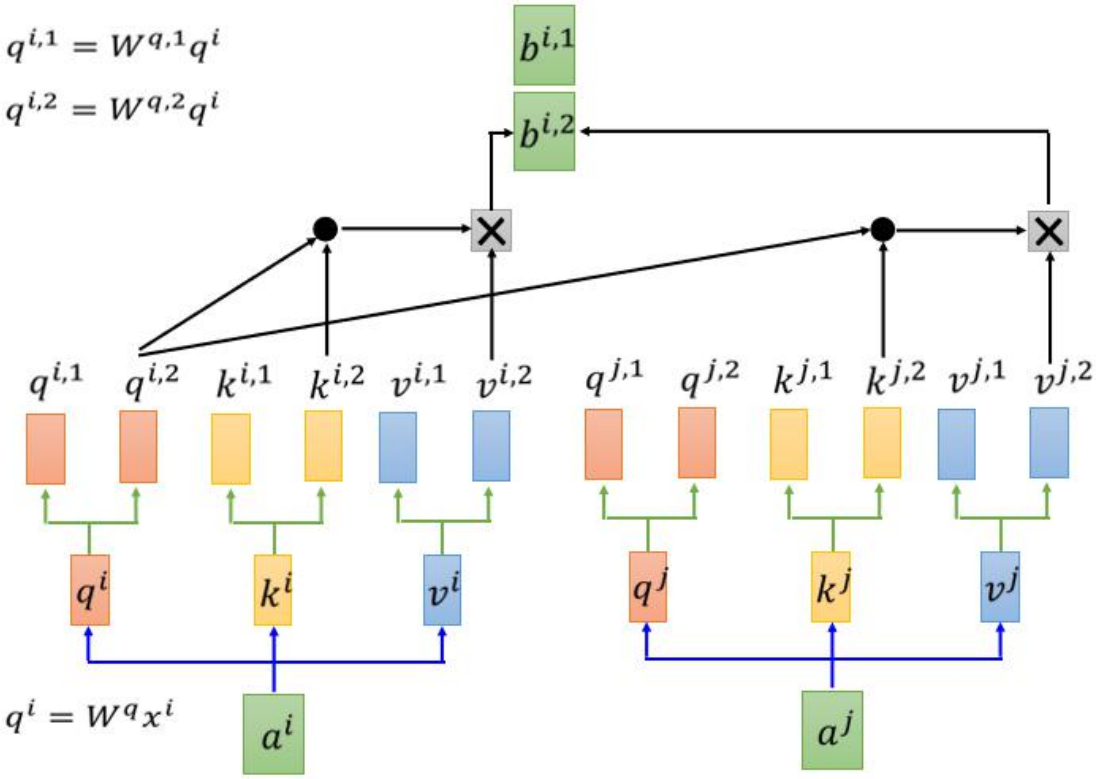


多个头的好处就是可以每个时刻同时关注多个地方，比如上图 8 个 head

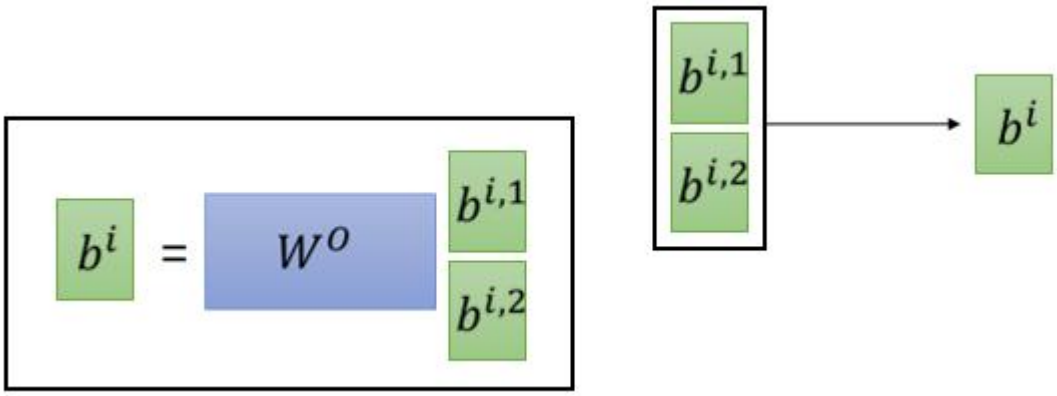
下图 2 个 head

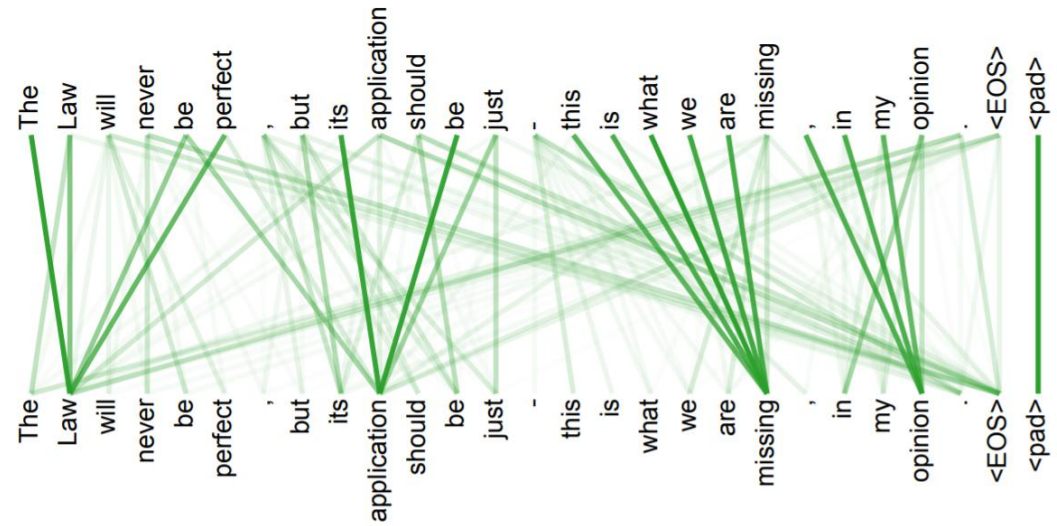
尚学堂·瓦战程序员

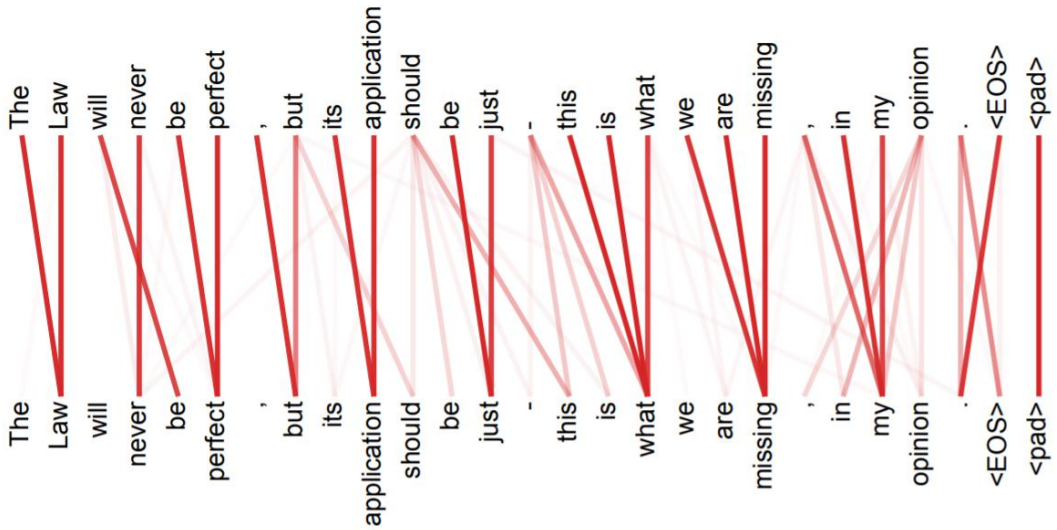

$q^{i,1} = W^{q,1}q^i$
$q^{i,2} = W^{q,2}q^i$
$b^{i,1}$
$q^{i,1}$ $q^{i,2}$ $k^{i,1}$ $k^{i,2}$ $v^{i,1}$ $v^{i,2}$ $q^{j,1}$ $q^{j,2}$ $k^{j,1}$ $k^{j,2}$ $v^{j,1}$ $v^{j,2}$
q^i k^i v^i q^j k^j v^j
$q^i = W^q a^i$
a^i a^j


$q^{i,1} = W^{q,1}q^i$
$q^{i,2} = W^{q,2}q^i$
$b^{i,1}$ $b^{i,2}$
$q^{i,1}$ $q^{i,2}$ $k^{i,1}$ $k^{i,2}$ $v^{i,1}$ $v^{i,2}$ $q^{j,1}$ $q^{j,2}$ $k^{j,1}$ $k^{j,2}$ $v^{j,1}$ $v^{j,2}$
q^i k^i v^i q^j k^j v^j
$q^i = W^q x^i$
a^i a^j

尚学堂·瓦战程序员

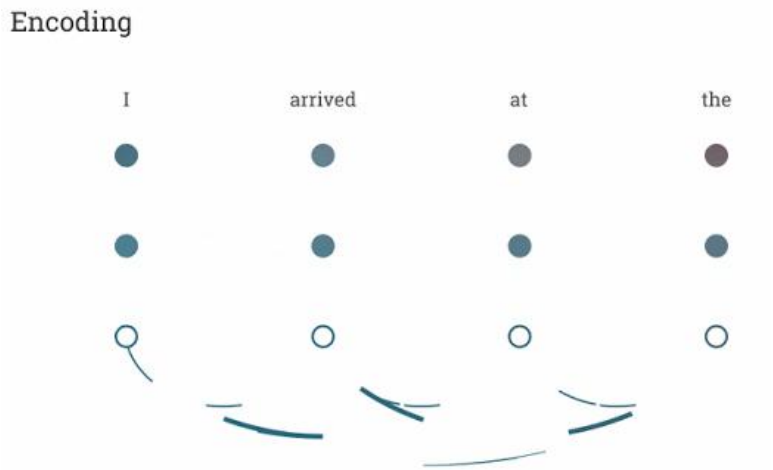

b^i
=
W^o
$b^{i,1}$
$b^{i,2}$
$b^{i,1}$
$b^{i,2}$
b^i


The Law will never be perfect , but its application should be just - this is what we are missing , in my opinion . <EOS> <pad>


The Law will never be perfect , but its application should be just - this is what we are missing , in my opinion . <EOS> <pad>

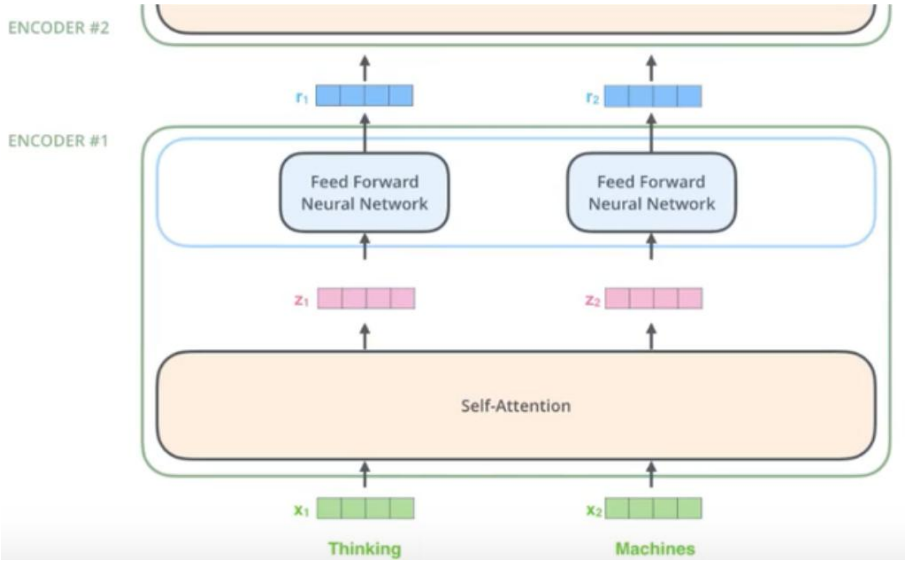
尚学堂·瓦战程序员

Encoder



前向神经网络就是一个全连接网络，每一个时刻拿到的 W 是一样的

encoder 中是双向的！

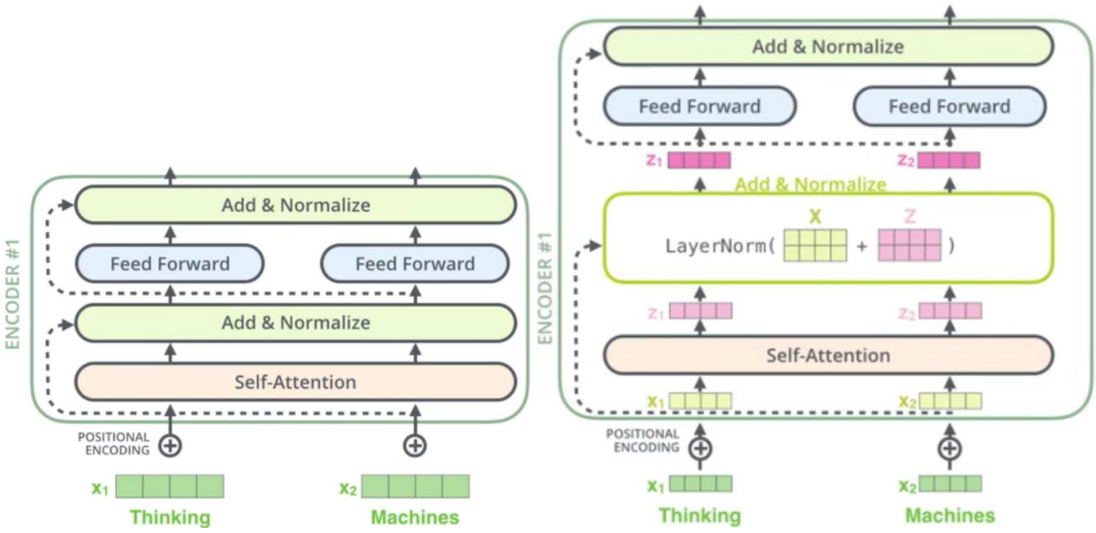


引入残差思想

batch normalization 是在不同样本间去计算的，layer normalization 是在同一个样本上计算的。

如果要用 add 相加必须保证 X 和 Z 维度是一样的，这就得让 self-attention 层处理后的 Z 维度和 X 一样。

尚学堂·瓦战程序员



位置信息

BoW 是词袋模型，不考虑词语在句子中的先后顺序。有些任务对词序不敏感，有些任务词序对结果影响很大。

例如机器翻译，“我爱你”和“你爱我”的翻译结果是不相同。不考虑位置信息的话，“我爱你”和“你爱我”Encoding 的结果是相同。

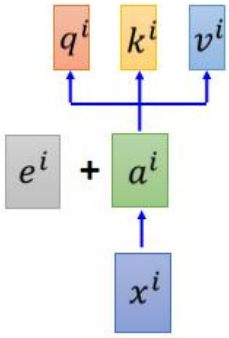
Position Embedding 一般用于：1.非 RNN 网络，由于不能编码序列顺序。需要显示的输入位置信息。如 CNN seq2seq 2.需要加强位置信息。如句子中存在核心词，需要对核心词的周边词着重加权等。

《Attention is All You Need》的 Transformer 模型用到 Position Embedding。

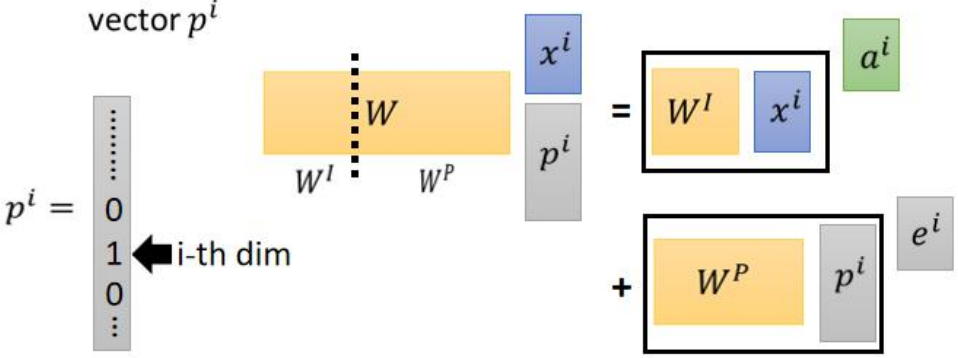
Transformer 摒弃了之前机器翻译任务中常用的 RNN 结构，使得并行性更好。RNN 的这种结构天生考虑了词语的先后顺序关系。当 Transformer 模型不使用 RNN 结构时，它就要想办法通过其它机制把位置信息传输到 Encoding 的部分。所以在该模型中中，每个时刻的输入是 Word Embedding+Position Embedding。

尚学堂·瓦战程序员

Positional Encoding



- No position information in self-attention.
- Original paper: each position has a unique positional vector e^i (not learned from data)
- In other words: each x^i appends a one-hot vector p^i



Position Embedding 有多种方法可以获得每个位置的编码：

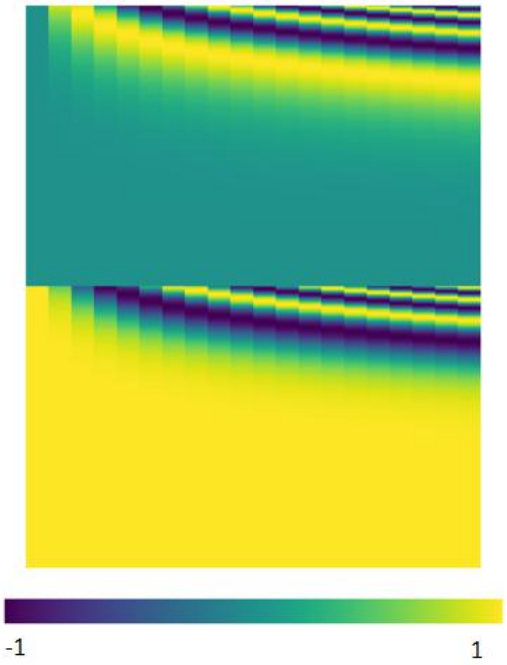
为每个位置随机初始化一个向量，在训练过程中更新这个向量；

《Attention is All You Need》使用正弦函数和余弦函数来构造每个位置的值。作者发现该方法最后取得的效果与 Learned Positional Embeddings 的效果差不多，但是这种方法可以在测试阶段接受长度超过训练集实例的情况。

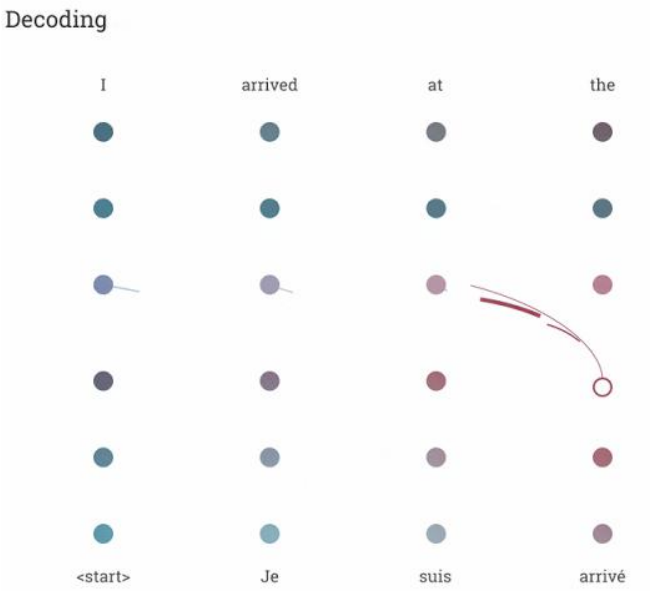
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

https://blog.csdn.net/qq_33278884



Decoder

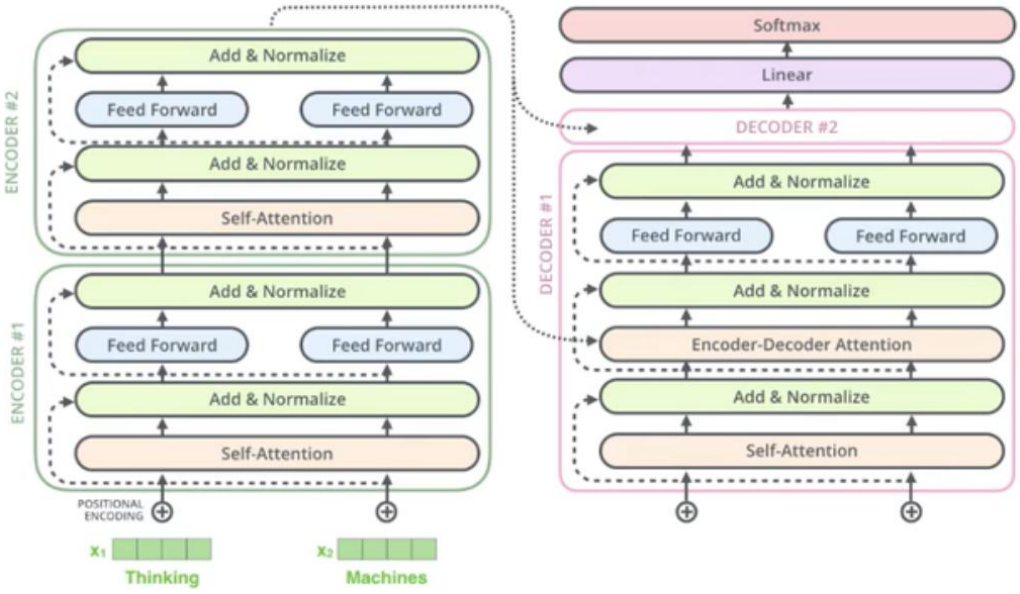


decoder 中的第一个 self-attention 是单向的！

decoder 中的从下往上的第一个 self-attention 是对之前的进行计算，比如现在是要预测 yt，那么肯定已经知道 y0，y1，y2 一直到 yt-1 了嘛，就是让 yt 对 y0，y1，y2 一直到 yt-1 这个已经预测好的来进行 self-attention 计算。

在 encoder-decoder attention 中是把刚讲过的 yt 对 y0，y1，y2 一直到 yt-1 做 self-attention 的当作 Q，之前的 encoder 的多个时刻的输出当作 K 和 V

尚学堂·瓦战程序员



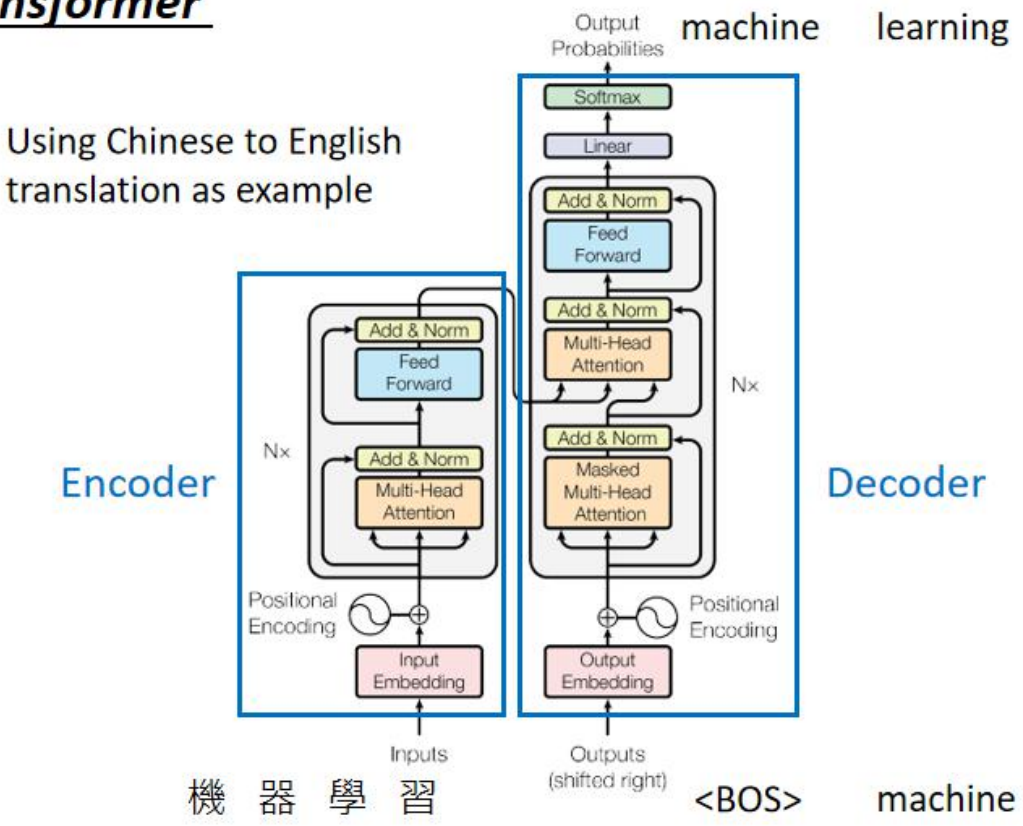
Masked

在预测 yt 时候，还不知道后面的时刻，所以需要屏蔽调，在 softmax 的时候把权值至为一个非常小的值就行了

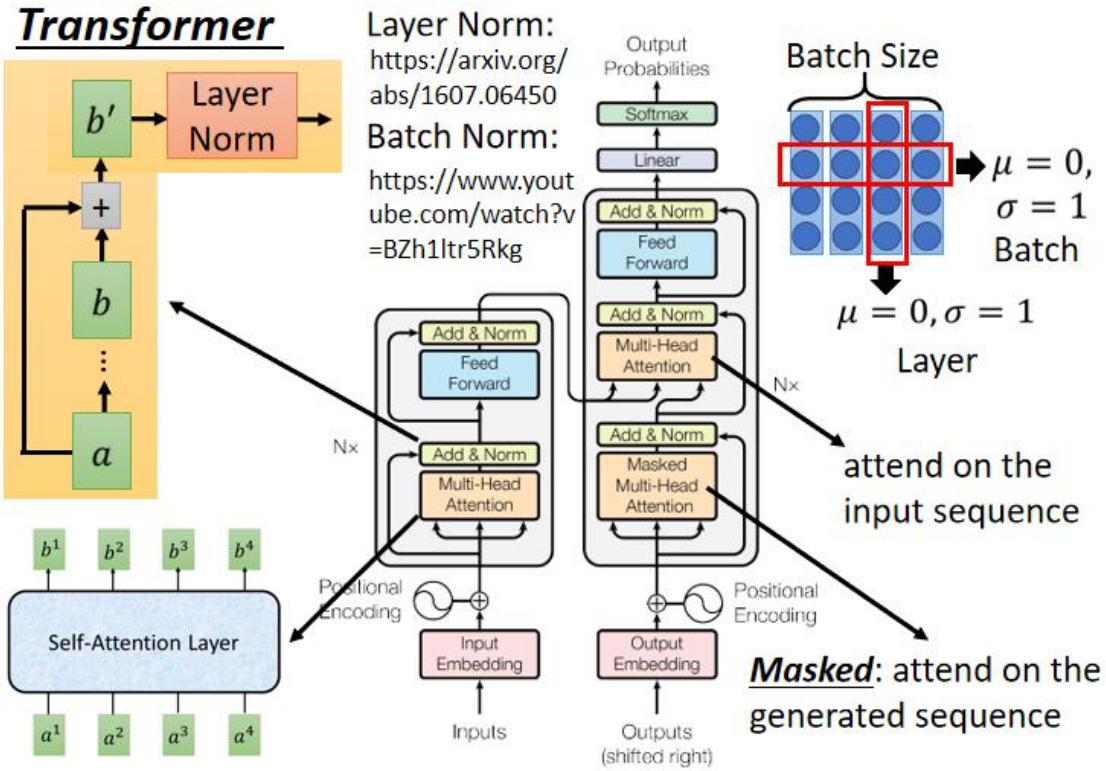
尚学堂·瓦战程序员

总结

Transformer



尚学堂·瓦战程序员

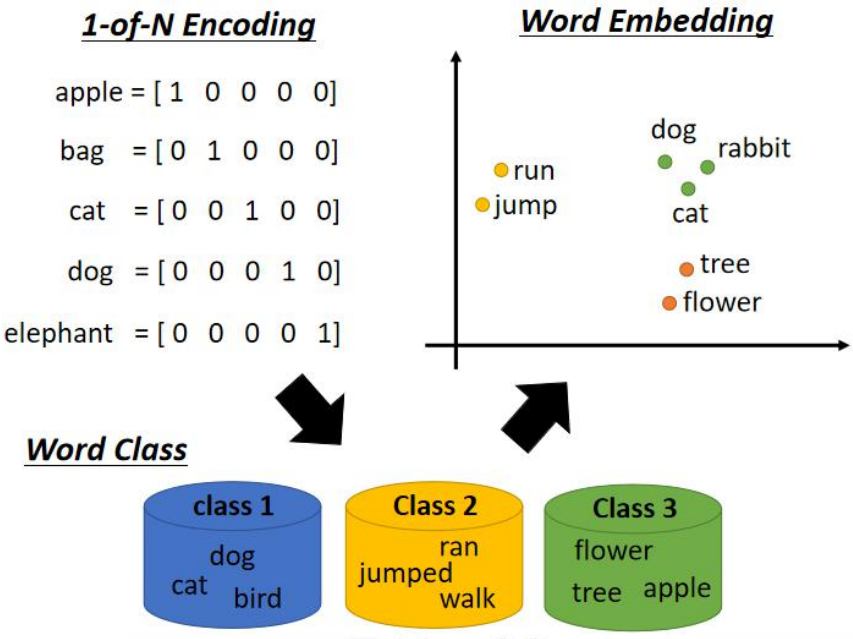


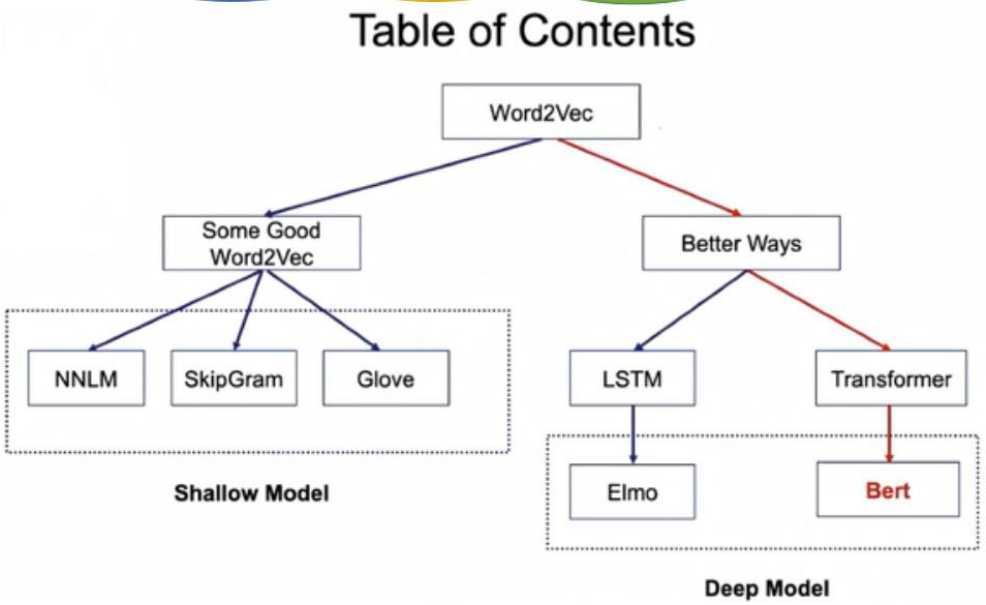
ELMO



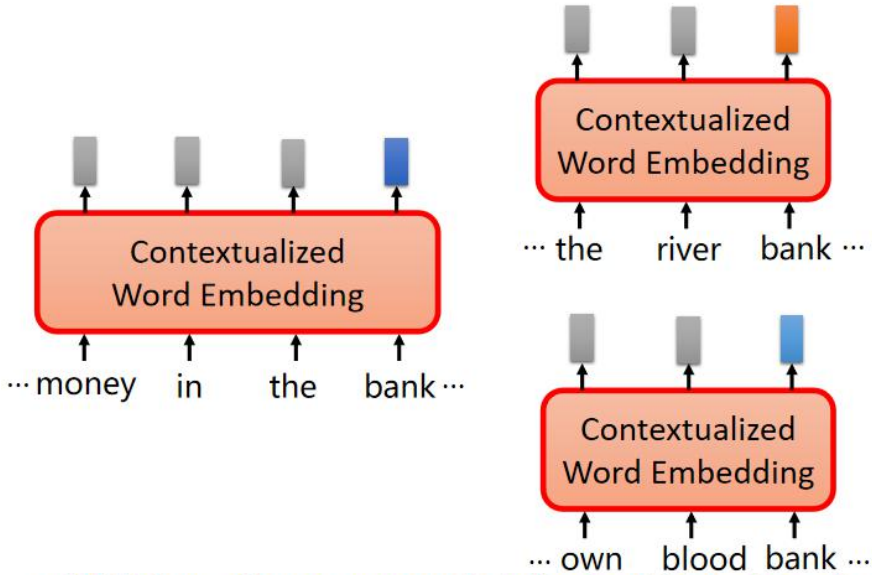
Embeddings from Language Model

尚学堂·瓦战程序员



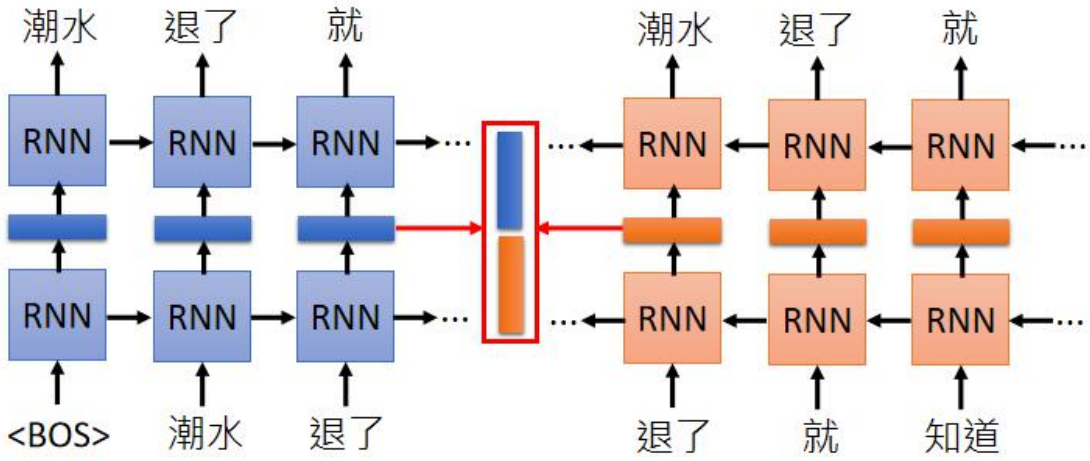


But a word can have multiple senses.



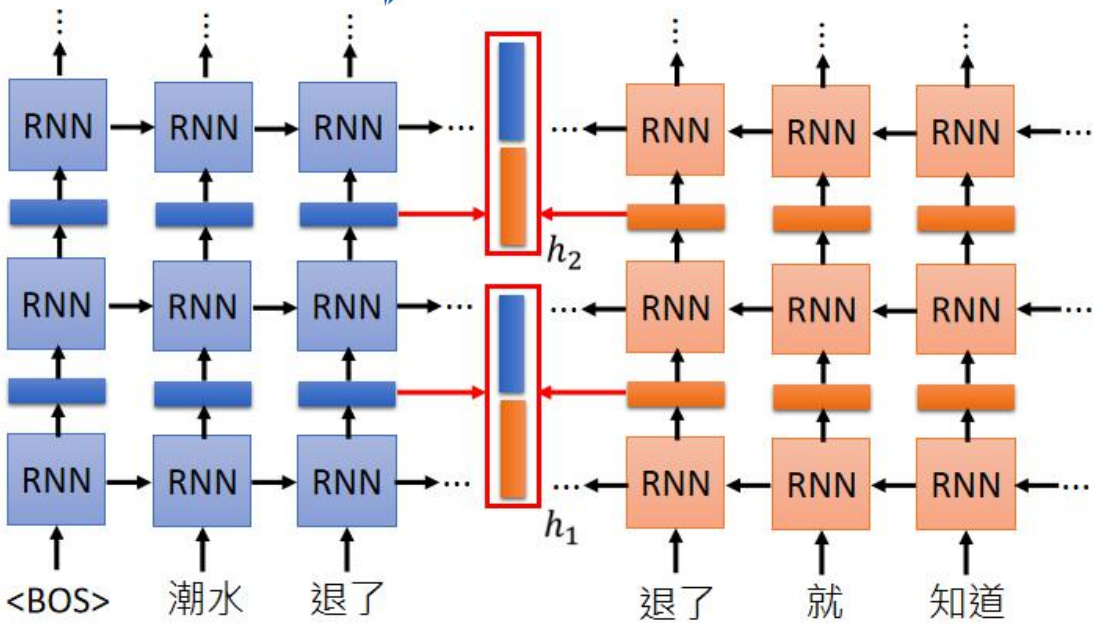
- RNN-based language models (trained from lots of sentences)

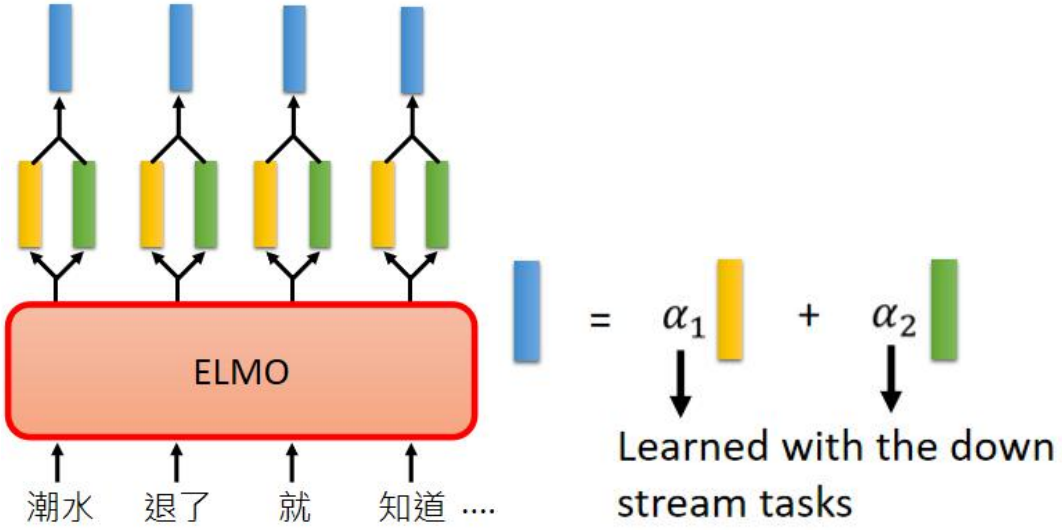
e.g. given "潮水 退了 就 知道 誰 沒穿 褲子"



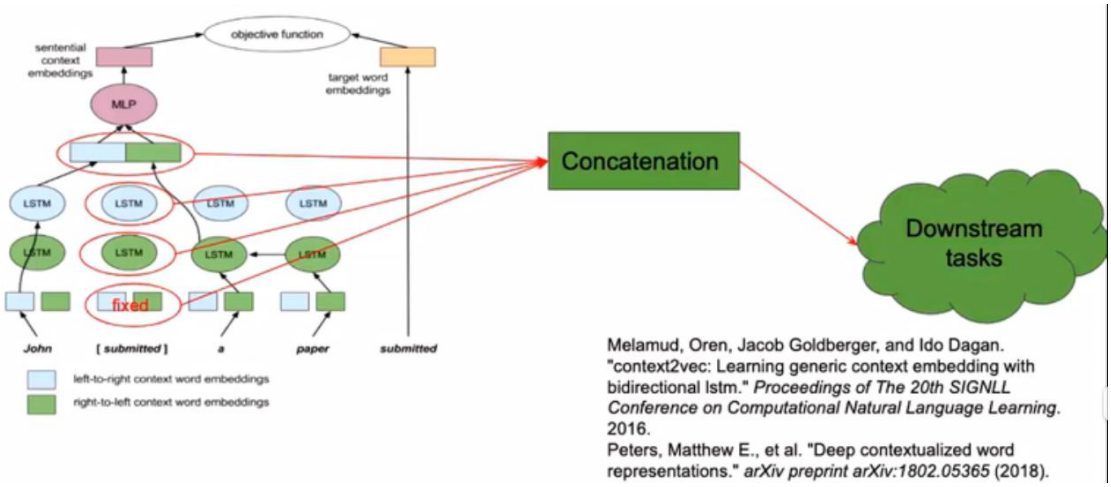
Each layer in deep LSTM can generate a latent representation.

Which one should we use???





把开始到中间层的词表达拿过来放在一起



Melamud, Oren, Jacob Goldberger, and Ido Dagan. "context2vec: Learning generic context embedding with bidirectional lstm." *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. 2016.
Peters, Matthew E., et al. "Deep contextualized word representations." *arXiv preprint arXiv:1802.05365* (2018).

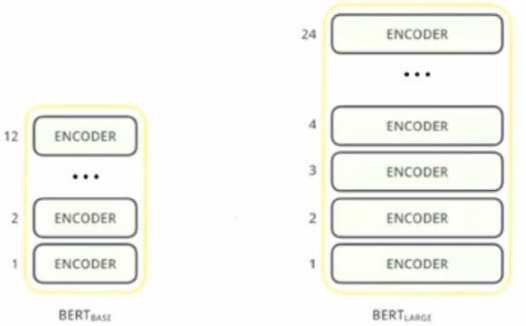
尚学堂·瓦战程序员

Bert

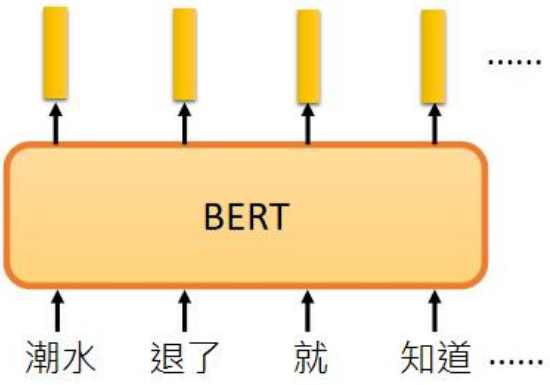


Bidirectional Encoder Representations from Transformers

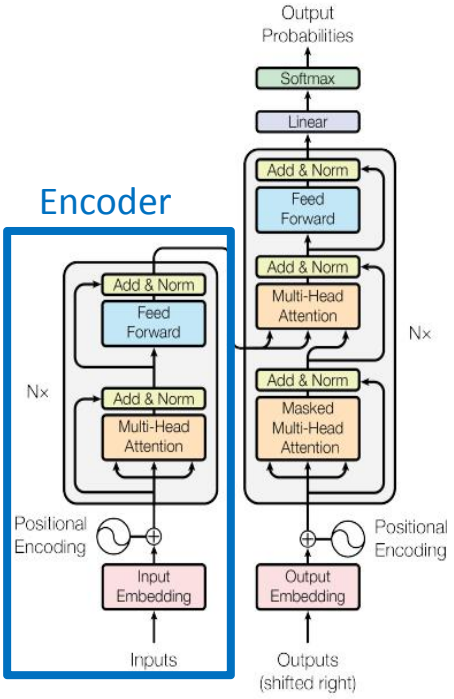
Bert = Stacks of Transformer Encoder



Learned from a large amount of text without annotation



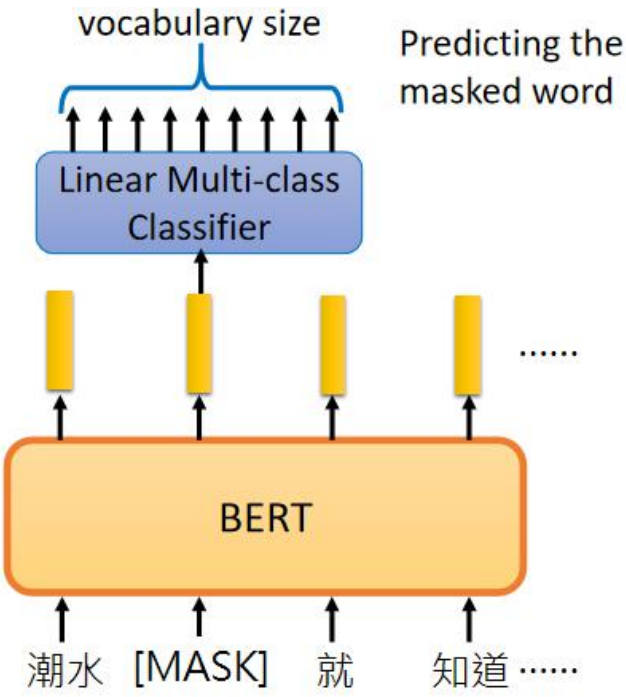
Encoder



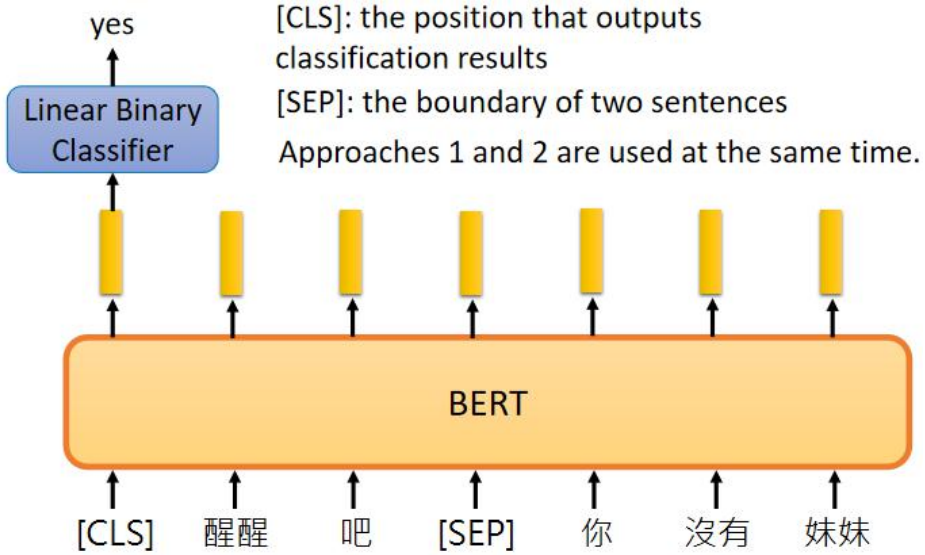
尚学堂·瓦战程序员

BERT 的训练方法

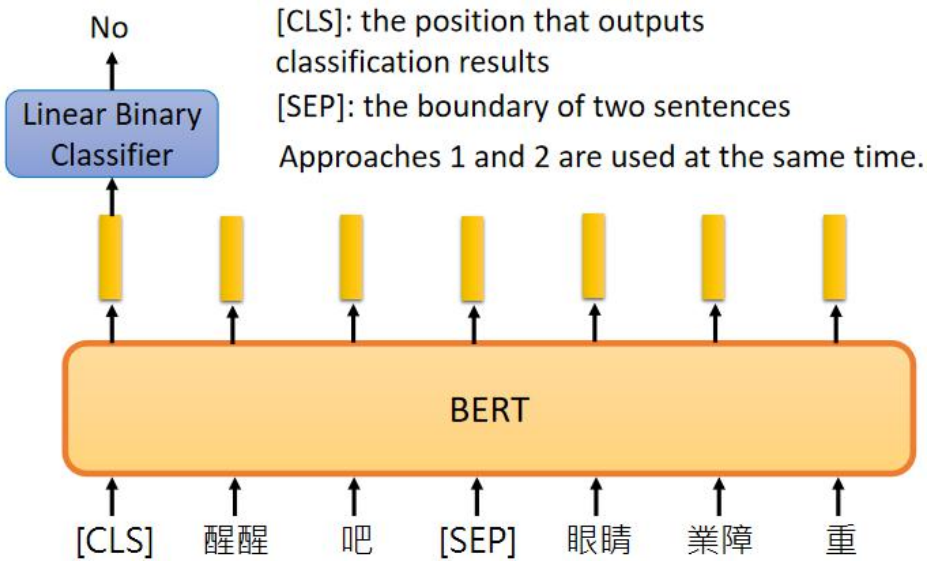
方法一：预测 Masked 词汇 (language model)



方法二：预测下一个句子



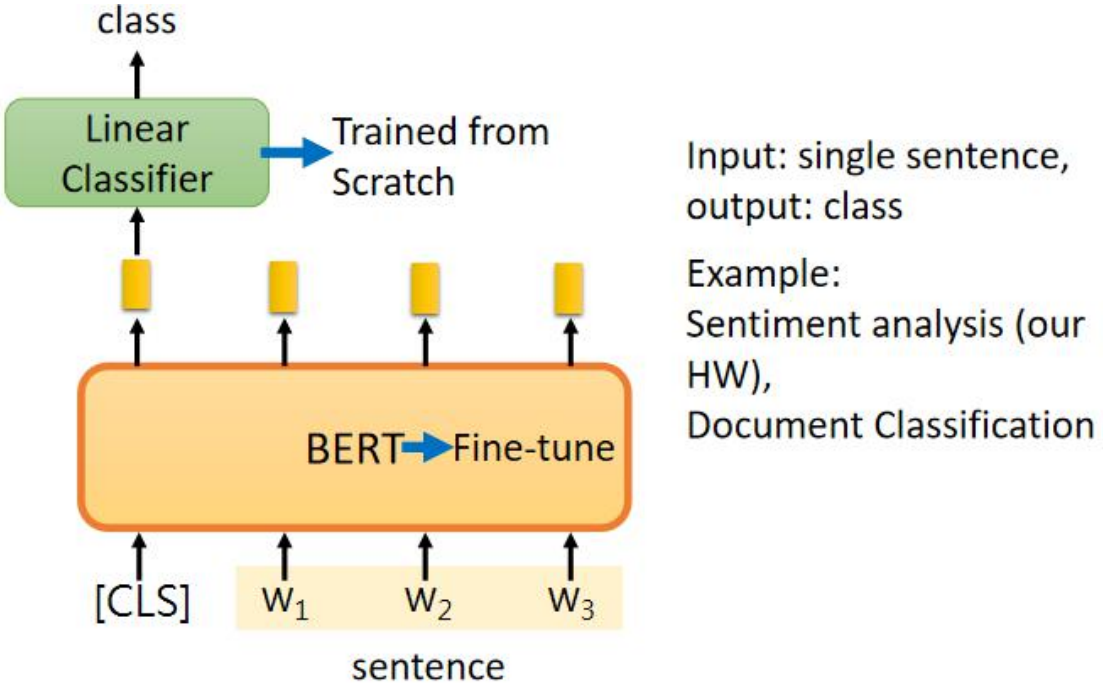
尚学堂·瓦战程序员



GPT 是单方向的，相当于是前面的去预测后面的词，BERT 是双方向的，相当于两边的来预测中间的

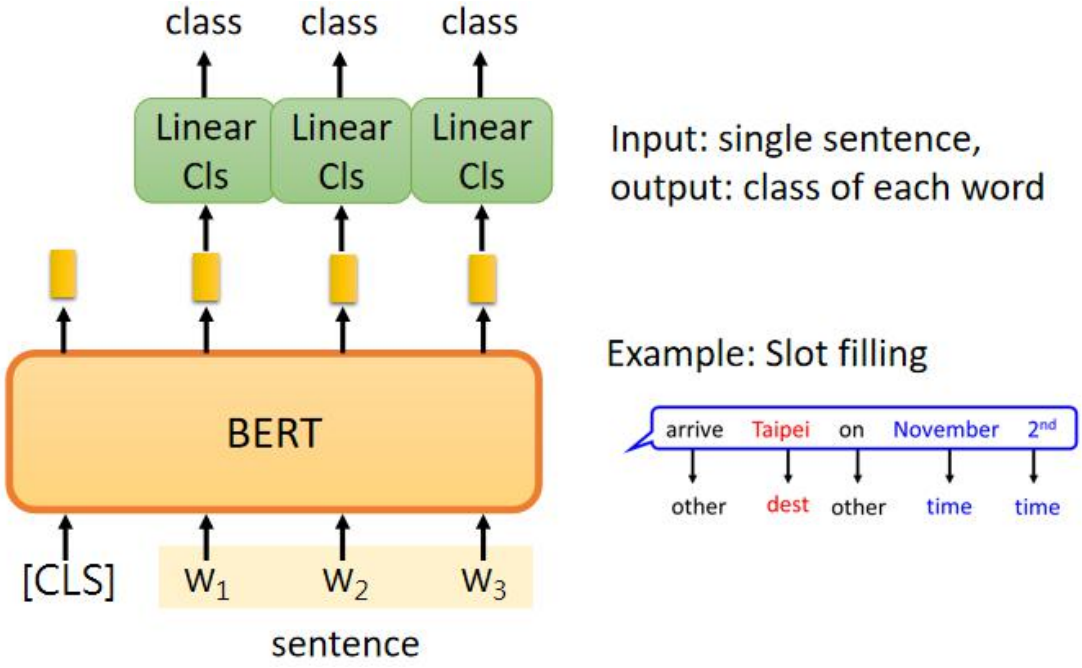
模型的使用

分类

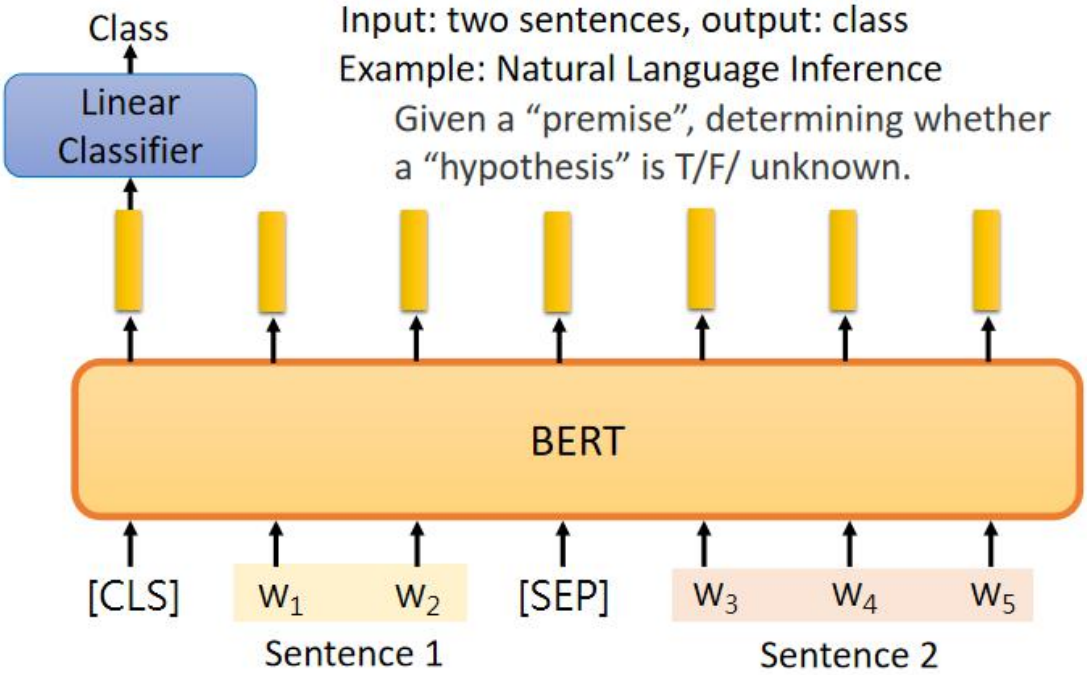


尚学堂·瓦战程序员

序列标注



成对句子分类



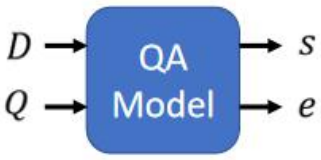
尚学堂·瓦战程序员

QA

- Extraction-based Question Answering (QA) (E.g. SQuAD)

Document: $D = \{d_1, d_2, \cdots, d_N\}$

Query: $Q = \{q_1, q_2, \cdots, q_N\}$



output: two integers (s, e)

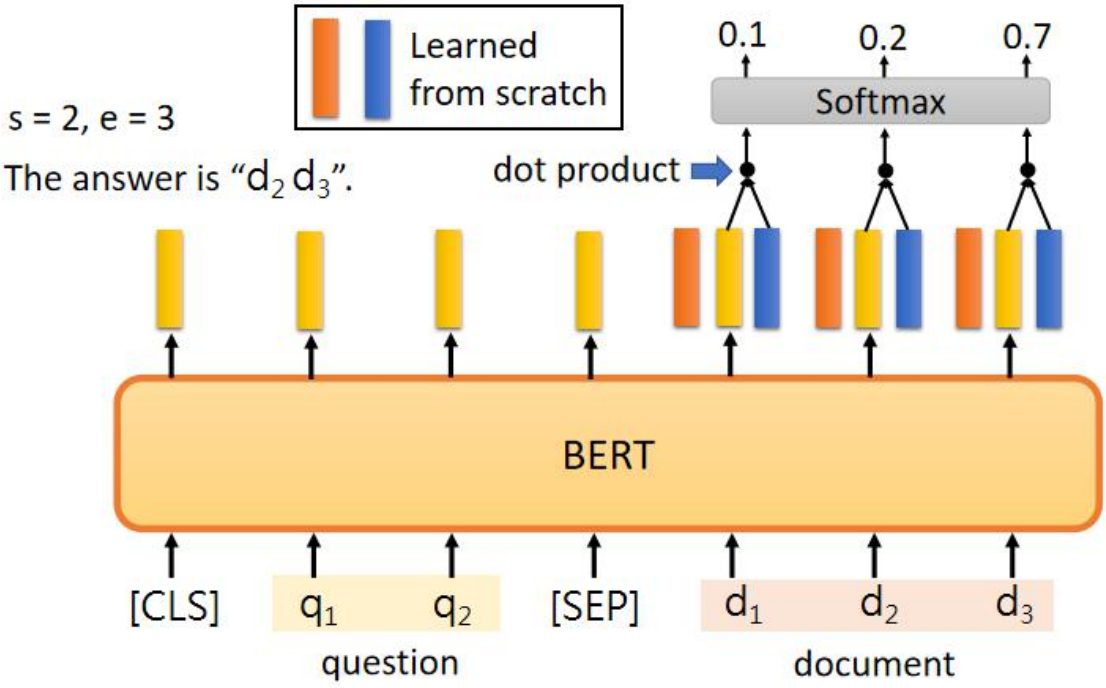
Answer: $A = \{q_s, \cdots, q_e\}$

In meteorology, precipitation is any product of the condensation of 17 spheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain 77 atte 79 cations are called "showers".

What causes precipitation to fall?
gravity $s = 17, e = 17$

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud $s = 77, e = 79$

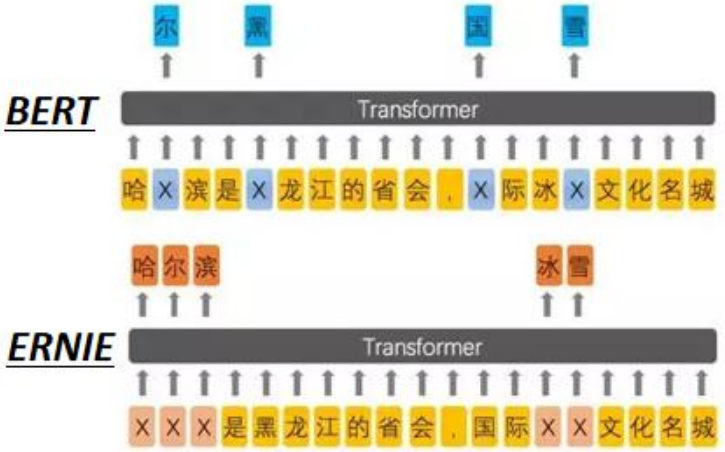


ERNIE

Enhanced Representation through Knowledge Integration

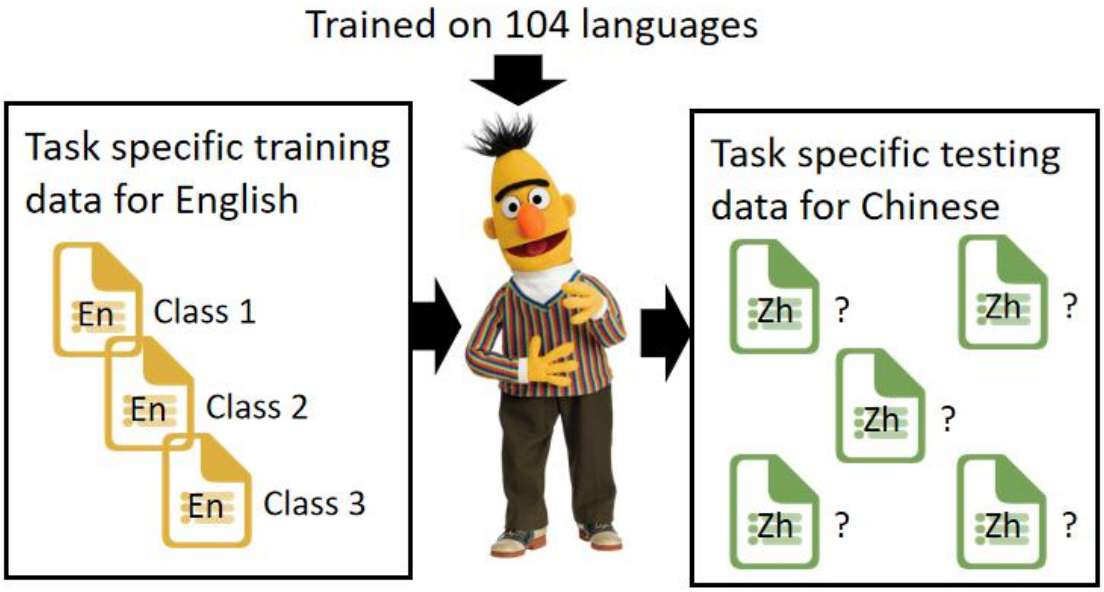
尚学堂·瓦战程序员

- Designed for Chinese





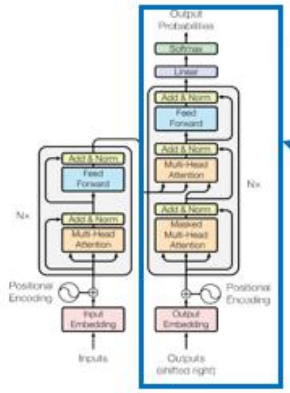
Multilingual BERT



GPT (OpenAI)

用的是提前训练好的 transformer 的 decoder

Generative Pre-Training (GPT)



Transformer Decoder

BERT (340M)

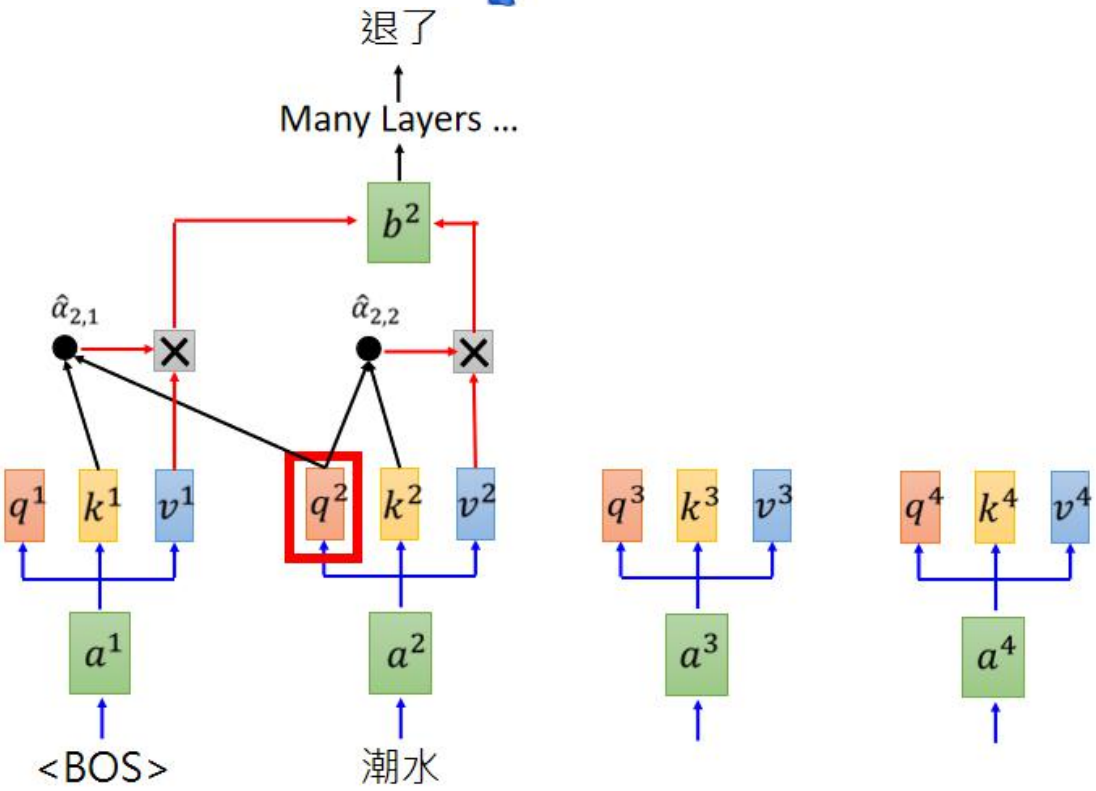
ELMO (94M)



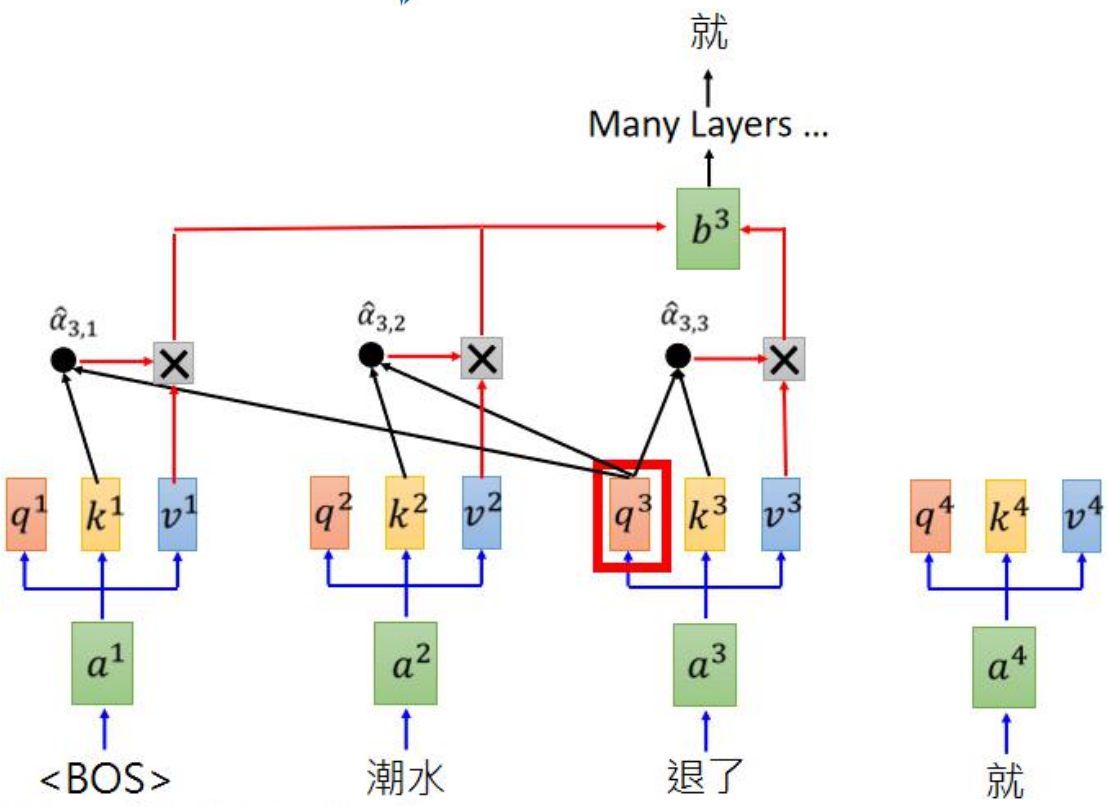




GPT-2 (1542M)



尚学堂·瓦战程序员

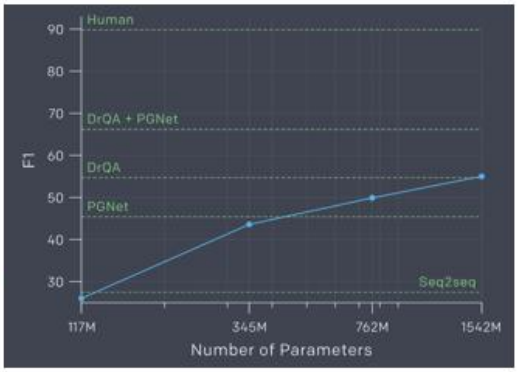


Zero-shot Learning?

- ***Reading Comprehension***

$d_1, d_2, \cdots, d_N$,
"Q:", $q_1, q_2, \cdots, q_N$,
"A:"





- ***Summarization*** $d_1, d_2, \cdots, d_N$,"TL;DR:"

- ***Translation***

English sentence 1	=	French sentence 1
English sentence 2	=	French sentence 2
English sentence 3	=	

尚学堂·瓦战程序员

《天龍八部》 前文脈絡	段譽和王語嫣面面相對，呼吸可聞，雖身處污泥，心中卻充滿了喜樂之情，誰也沒想到要爬出井去。兩人同時慢慢的伸手出來，四手相握，心意相通。
LSTM 生成結果	南海鱷神哈哈一笑，說道：「這裏斜行走了出去，肚中空蕩蕩地，顛為心驚。只聽他說「不出」，沒想到這三步來的變招，居然心下惴惴，伸掌在膝頭褲子、套心甚是指尖，那人可隨便運勁，將那本事大喝一聲。虛竹不去救司眾的臉上，全身酒水淋漓，密密熱炭的在那裏水來。游坦之乘著一名弟子，當時只覺馬鞭擊了過，說道：「我這一曲可就不錯。」李延宗道：「書上的事情卻到了大哥？我這長老賊的眼光之親，便是穿新衣衫，手背陳長老後頸微微一紅，突然氣往上衝，腦袋上內咬下來的兵刃掉在地下，額角撞了個圓圈之中，就算不出南來三俠的衝到，將兩柄飛刀到了，猛聽得蕭峯對敵傾覆，有的伸手叫好，喝道：「你怎知我受苦受寒毒的折磨你，我早生了。」一面哭笑，一個女子聲音笑道：「我嫁了王姑娘，我愛你做花。」奔向出去，見四具屍體匆匆忙逼？踏了下去，大聲急衝揚手，將他抱在身上。阿紫柔聲安慰，向蕭峯道：「孩兒，懂得甚麼？四大惡人齊去應敵。」蕭峯道：「我送了給外人，是否衝劍出仔細說罷。眾位姑娘，嗯，在下姑蘇燕子塢慕容先生仙逝。」蕭峯低沉著聲音陰神看了一眼，心下雪亮，見蕭峯背負布袋，兀自未曾出外了。穿過一個少年深的肌膚，一口都喝到一聲：「啊喲！」發足上湧，怒氣漸漸鬆軟 ...
《天龍八部》 前文脈絡	段譽和王語嫣面面相對，呼吸可聞，雖身處污泥，心中卻充滿了喜樂之情，誰也沒想到要爬出井去。兩人同時慢慢的伸手出來，四手相握，心意相通。
生成結果	過了良久，鳩摩智嘆了口氣，說道：「咱們一面看去，不過得你爺兒倆的事，那就誰也不會大意了。」段譽道：「我不會再來見你。」鳩摩智道：「你既不見我，我也不會見我。我只求你一件事，以便遵命。」 段譽道：「你這就奇了。我是甚麼？」鳩摩智道：「我是這樣說，你總是要見到我的。」段譽胸口一酸，心想：「要我幹麼？你一求，我便幹，就幹麼？」說道：「要我答允你，我是永遠不離開你的啦。」 王語嫣坐起身來，月光之下，只見鳩摩智的衣袖在月光下，向她微微掃視，她的容貌秀麗，美麗的臉龐上帶著幾分嬌艷，不禁心中一動：「段公子，你的心早就一定要過來了。」 段譽胸口一熱，頭暈紅了，卻不想要答應他的話，卻說不出口。王語嫣道：「段公子，咱們是再度相會，你這一生一世，決不見外了。世上有許多事情，你一輩子寂寞，豈不是好？」 慕容復哼了一聲，道：「你這人當真不要？你這人雖然清俏，心中卻也不快活。我跟你說，你是不是真的待你好色之人，是不是？」 王語嫣抬起頭來，淺淺一笑，道：「我要是見到你這等容貌，從來沒跟我談起，更沒想到這件事。你剛才在過湖裏，為了你，要跟我談談，你卻不許？」 段譽胸口一熱，淚水便欲奪眶而出，哽咽道：「姑娘，我實是不願意 ...

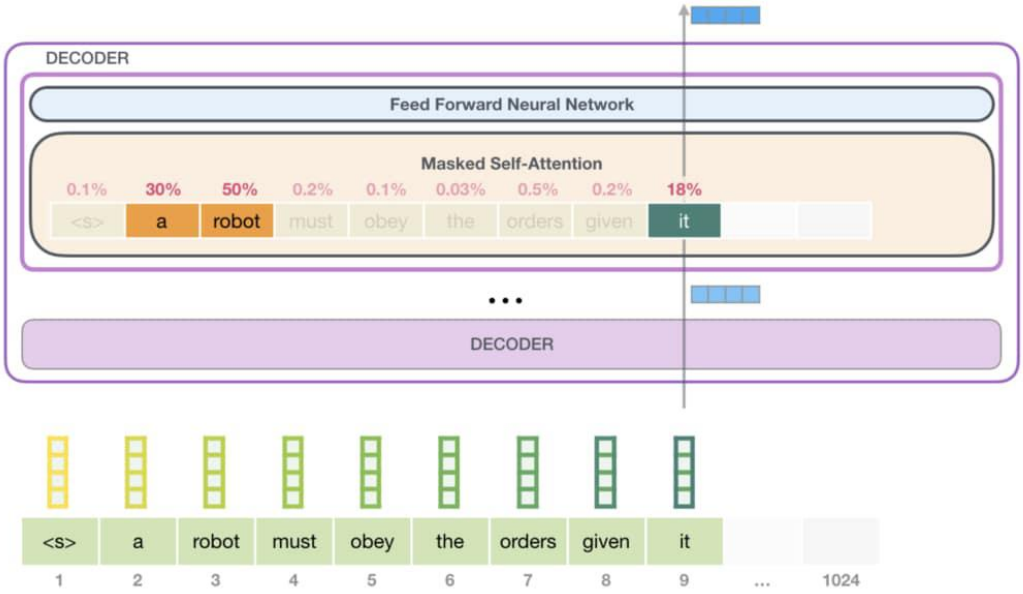
> 因為顧慮到這項技術可能會遭到惡意運用，我們目前並不打算釋出已訓練好的模型。但為了促進研究我們將釋出規模小很多的模型供研究者參考。
>
> — OpenAI, 2019/02

不過至少目前看來 OpenAI 只是採取相對謹慎的態度在釋出模型。該團隊在今年 2 月將最小的 124M GPT-2 Small(1.2 億參數)與論文一起釋出 ,並在 5 月釋出 355M 的

GPT-2 Medium。而就在不久前的 8 月釋出了有 7.74 億參數的 GPT-2 Large，其模型大小是 1558M GPT-2 完全體的一半。
https://openai.com/blog/gpt-2-6-month-follow-up/



GPT 選擇 Transformer 裡的 Decoder，訓練目標為一般的語言模型，預測下個字
BERT 選擇 Transformer 裡的 Encoder，訓練目標則為漏字填空以及下句預測



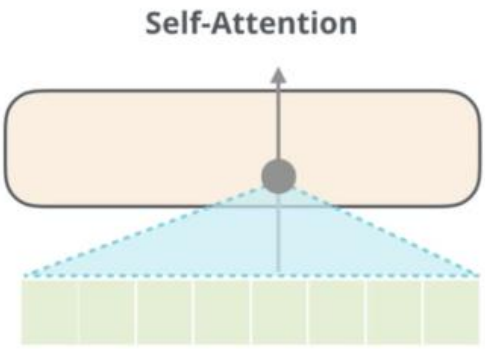
以上例而言，訓練好的 GPT 可以在看到 it 時知道該去關注前的面 a 及 robot，並進而調整 it 在當下代表的意思（即修正該詞彙的 vector representation）。而被融入前文語意的新 representation 就是所謂的 Contextual Word Representation。

關於自注意力機制（Self-Attention），有個值得記住的重要概念：在 GPT 之後問世的 BERT 是同時關注整個序列來修正一個特定詞彙的 representation，讓該詞彙的 repr. 同時隱含上下文資訊；而 GPT 是一個由左到右的常見語言模型，會額外透過遮罩技巧（Masking）來確保模型只會關注到

尚学堂·瓦战程序员

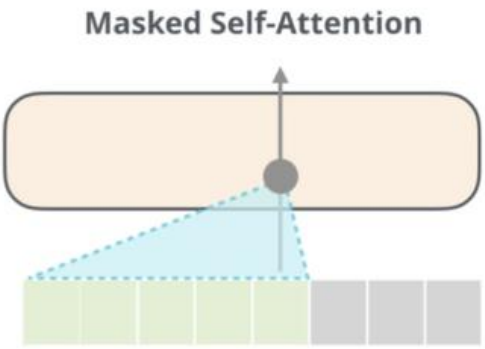
某詞彙以左的上文資訊。

一次關注整個序列



Transformer Encoder
知名模型：BERT

只關注當前詞彙**左側**的序列



Transformer Decoder
知名模型：GPT(-2)