Attention模型方法综述 -多篇经典论文解读

星期一, 六月 11, 2018　　3:48 下午

本文基于几篇经典的论文，对 Attention 模型的不同结构进行分析、拆解。

先简单谈一谈 attention 模型的引入。以基于 seq2seq 模型的机器翻译为例，如果 decoder 只用 encoder 最后一个时刻输出的 hidden state，可能会有两个问题（我个人的理解）。

1. encoder 最后一个 hidden state，与句子末端词汇的关联较大，难以保留句子起始部分的信息；

2. encoder 按顺序依次接受输入，可以认为 encoder 产出的 hidden state 包含有词序信息。所以一定程度上 decoder 的翻译也基本上沿着原始句子的顺序依次进行，但实际中翻译却未必如此，以下是一个翻译的例子：

英文原句： space and oceans are the new world which scientists are trying to explore.

翻译结果： 空间和海洋是科学家试图探索的新世界。

词汇对照如下：

可以看到，翻译的过程并不总是沿着原句从左至右依次进行翻译，例如上面例子的定语从句。

为了一定程度上解决以上的问题，14 年的一篇文章 **Sequence to Sequence Learning with Neural Networks** 提出了一个有意思的 trick，即在模型训练的过程中将原始句子进行

反转，取得了一定的效果。

为了更好地解决问题，attention 模型开始得到广泛重视和应用。

下面进入正题，进行对 attention 的介绍。

Show, Attend and Tell

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

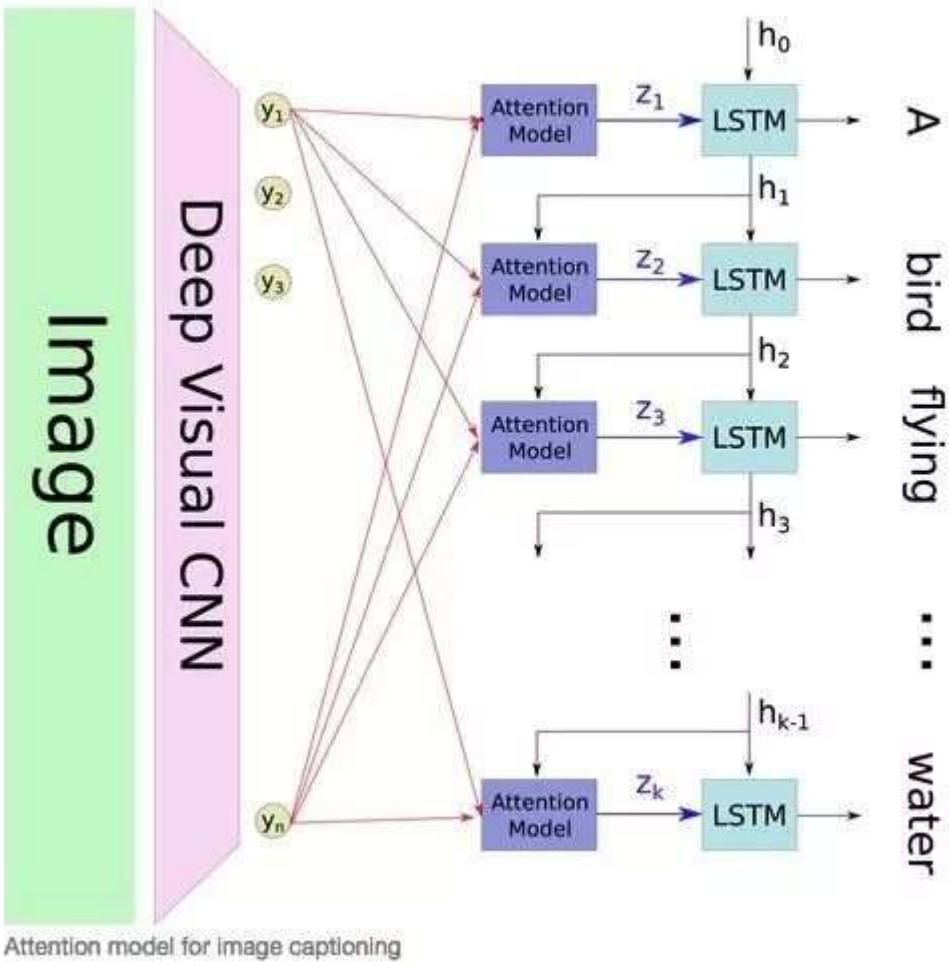
Kelvin Xu	KELVIN.XU@UMONTREAL.CA
Jimmy Lei Ba	JIMMY@PSI.UTORONTO.CA
Ryan Kiros	RKIROS@CS.TORONTO.EDU
Kyunghyun Cho	KYUNGHYUN.CHO@UMONTREAL.CA
Aaron Courville	AARON.COURVILLE@UMONTREAL.CA
Ruslan Salakhutdinov	RSALAKHU@CS.TORONTO.EDU
Richard S. Zemel	ZEMEL@CS.TORONTO.EDU
Yoshua Bengio	FIND-ME@THE.WEB

■ 论文 | Show, Attend and Tell: Neural Image Caption Generation with Visual Attention

■ 链接 | https://www.paperweekly.site/papers/812

■ 源码 | https://github.com/kelvinxu/arctic-captions

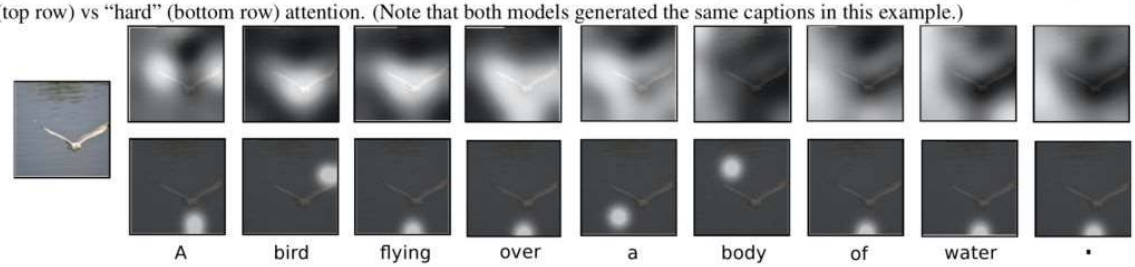
文章讨论的场景是图像描述生成（Image Caption Generation），对于这种场景，先放一张图，感受一下 attention 的框架。



Attention model for image captioning

文章提出了两种 attention 模式，即 hard attention 和 soft attention，来感受一下这两种 attention。

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)



可以看到，hard attention 会专注于很小的区域，而 soft attention 的注意力相对发散。模型的 encoder 利用 CNN (VGG net)，提取出图像的 L 个 D 维的向量

$$a_i, i = 1, 2, \ldots, L$$

，每个向量表示图像的一部分信息。

decoder 是一个 LSTM，每个 timestep t 的输入包含三个部分，即 context vector Zt 、前一个 timestep 的 hidden state

$$h_{t-1}$$

、前一个 timestep 的 output

$$y_{t-1}$$

。 Zt 由 {ai} 和权重 {αti} 通过加权得到。这里的权重 αti 通过attention模型 *f*att 来计算得到，而本文中的 *f*att 是一个多层感知机（multilayer perceptron）。

$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}.$$

从而可以计算

$$z_t = \sum_{i=1}^{L} \alpha_{ti} * a_i$$

。接下来文章重点讨论 hard（也叫 stochastic attention）和 soft（也叫 deterministic）两种 attention 模式。

1. Stochastic “Hard” Attention

记 St 为 decoder 第 t 个时刻的 attention 所关注的位置编号， Sti 表示第 t 时刻 attention 是否关注位置 i， Sti 服从多元伯努利分布（multinoulli distribution）， 对于任意的 t，Sti,i=1,2,...,L 中有且只有取 1，其余全部为 0，所以 [St1,St2,...,stL] 是 one-hot 形式。这种

attention 每次只 focus 一个位置的做法，就是“hard”称谓的来源。 Zt 也就被视为一个变量，计算如下：

$$p(s_{t,i} = 1 \mid s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$

$$\hat{\mathbf{z}}_t = \sum_i s_{t,i} \mathbf{a}_i.$$

问题是 αti 怎么算呢？把 αti 视为隐变量，研究模型的目标函数，进而研究目标函数对参数的梯度。直观理解，模型要根据 a=(a1,...,aL) 来生成序列 y=(y1,...,yC)，所以目标可以是最大化 log p(y|a)，但这里没有显式的包含 s，所以作者利用著名的 Jensen 不等式（Jensen's inequality）对目标函数做了转化，得到了目标函数的一个 lower bound，如下：

$$\begin{aligned} L_s &= \sum_s p(s \mid \mathbf{a}) \log p(\mathbf{y} \mid s, \mathbf{a}) \\ &\leq \log \sum_s p(s \mid \mathbf{a}) p(\mathbf{y} \mid s, \mathbf{a}) \\ &= \log p(\mathbf{y} \mid \mathbf{a}) \end{aligned}$$

这里的 s ={ s1,...,sC }，是时间轴上的重点 focus 的序列，理论上这种序列共有

L^C

个。 然后就用 log p(y|a) 代替原始的目标函数，对模型的参数 W 算 gradient。

$$\frac{\partial L_s}{\partial W} = \sum_s p(s \mid \mathbf{a}) \Bigg[\frac{\partial \log p(\mathbf{y} \mid s, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid s, \mathbf{a}) \frac{\partial \log p(s \mid \mathbf{a})}{\partial W} \Bigg].$$

然后利用蒙特卡洛方法对 s 进行抽样，我们做 N 次这样的抽样实验，记每次取到的序列是 $\tilde{s}^n$，易知 $\tilde{s}^n$ 的概率为 $\frac{1}{N}$，所以上面的求 gradient 的结果即为：

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{n=1}^{N} \Bigg[\frac{\partial \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid \tilde{s}^n, \mathbf{a}) \frac{\partial \log p(\tilde{s}^n \mid \mathbf{a})}{\partial W} \Bigg]$$

接下来的一些细节涉及reinforcement learning，感兴趣的同学可以去看这篇 paper。

2. Deterministic “Soft” Attention

说完“硬”的 attention，再来说说“软”的 attention。 相对来说 soft attention 很好理解，在 hard attention 里面，每个时刻 t 模型的序列 [St1,...,StL] 只有一个取 1，其余全部为 0，也就是说每次只 focus 一个位置，而 soft attention 每次会照顾到全部的位置，只是不同位置的权重不同罢了。这时 Zt 即为 ai 的加权求和：

$$z_t = \sum_{i=1}^{L} \alpha_{ti} * a_i$$

这样 soft attention 是光滑的且可微的（即目标函数，也就是 LSTM 的目标函数对权重 αti 是可微的，原因很简单，因为目标函数对 Zt 可微，而 Zt 对 αti 可微，根据 chain rule 可得目标函数对 αti 可微）。

文章还对这种 soft attention 做了微调：

$$z_t = \beta_t * \sum_{i=1}^{L} \alpha_{ti} * a_i$$

其中

$$\beta_t = \sigma(f_\beta(h_{t-1}))$$

，用来调节 context vector 在 LSTM 中的比重（相对于 h_{t-1}、y_{t-1} 的比重）。

btw，模型的 loss function 加入了 αti 的正则项。

$$L_d = -\log(P(\mathbf{y}|\mathbf{x})) + \lambda \sum_{i}^{L}(1 - \sum_{t}^{C} \alpha_{ti})^2$$

Attention-based NMT

Effective Approaches to Attention-based Neural Machine Translation

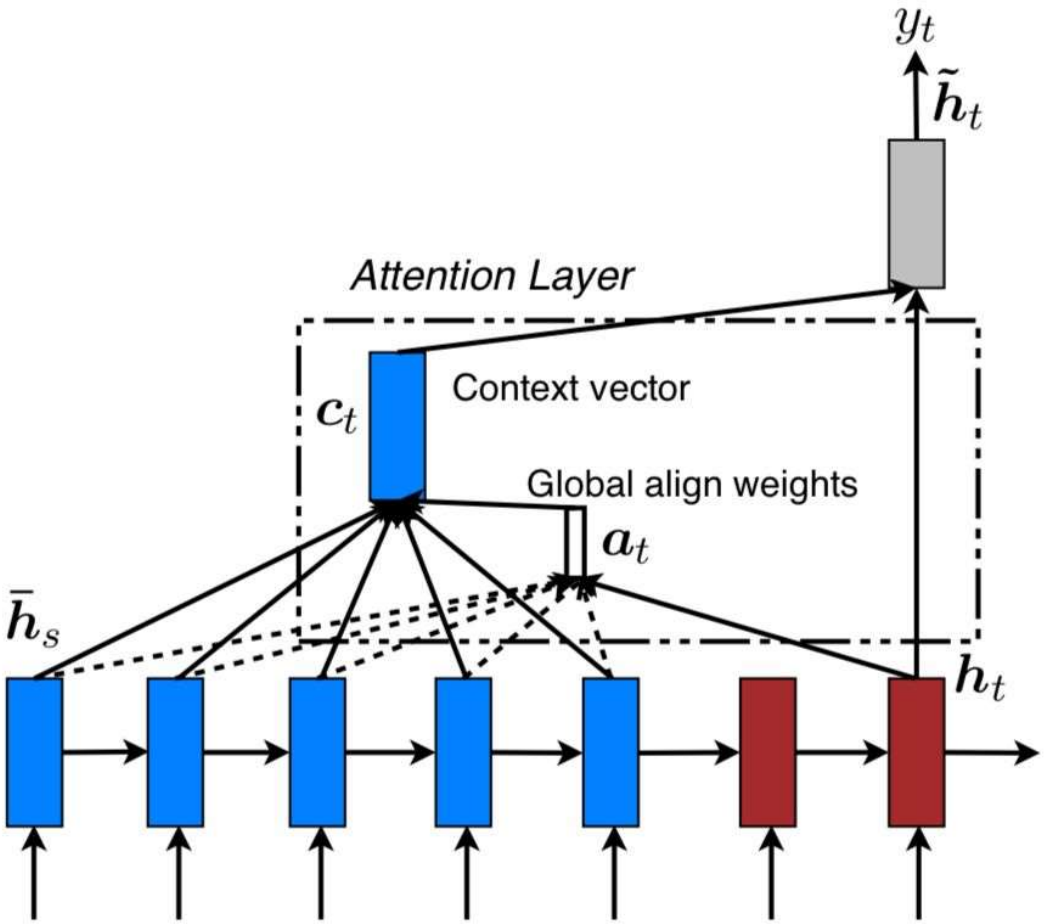
Minh-Thang Luong **Hieu Pham** **Christopher D. Manning**
Computer Science Department, Stanford University, Stanford, CA 94305
{lmthang,hyhieu,manning}@stanford.edu

■ 论文 | Effective Approaches to Attention-based Neural Machine Translation

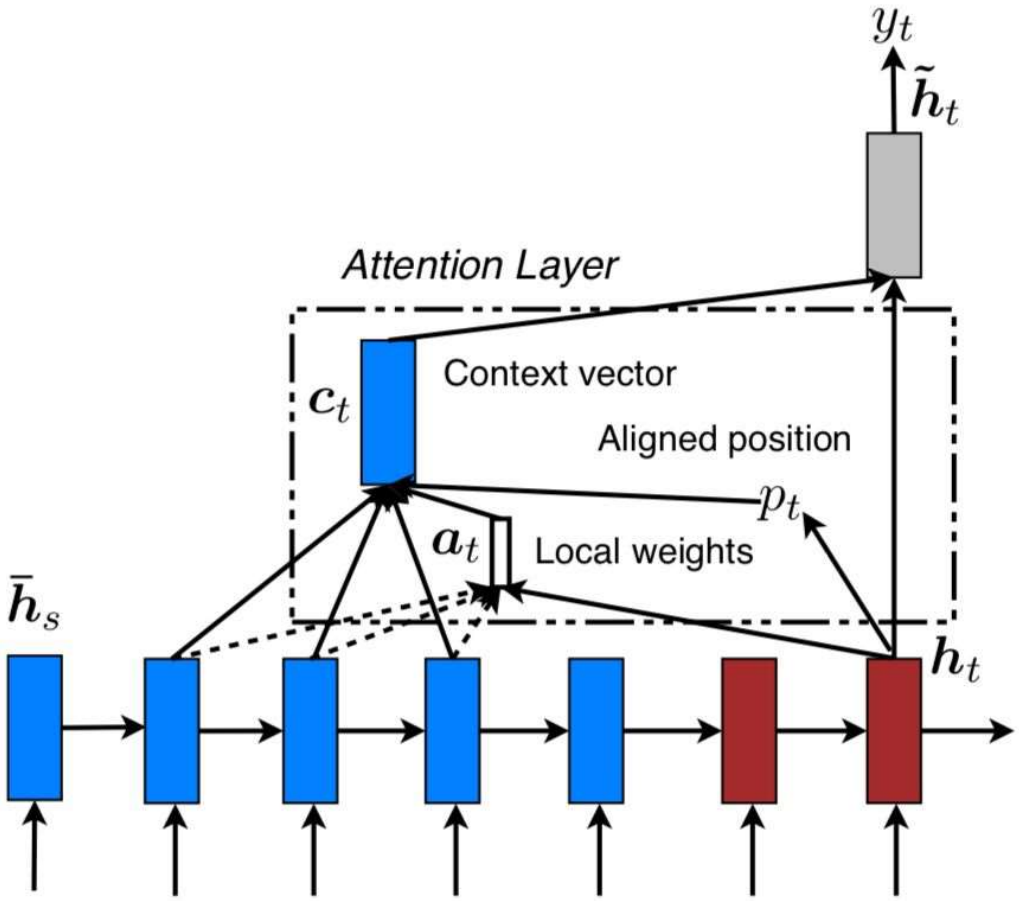
■ 链接 | https://www.paperweekly.site/papers/806

■ 源码 | https://github.com/lmthang/nmt.matlab

文章提出了两种 attention 的改进版本，即 global attention 和 local attention。先感受一下 global attention 和 local attention 长什么样子。



▲ Global Attention



▲ Local Attention

文章指出，local attention 可以视为 hard attention 和 soft attention 的混合体（优势上的混合），因为它的计算复杂度要低于 global attention、soft attention，而且与 hard attention 不同的是，local attention 几乎处处可微，易与训练。 文章以机器翻译为场景，x1,...,xn 为 source sentence， y1,...,ym 为 target sentence， c1,...,cm 为 encoder 产生的 context vector，objective function 为：

$$J_t = \sum_{(x,y)\in\mathbb{D}} -\log p(y|x)$$

Ct 来源于 encoder 中多个 source position 所产生的 hidden states，global attention 和 local attention 的主要区别在于 attention 所 forcus 的 source positions 数目的不同：如果 attention forcus 全部的 position，则是 global attention，反之，若只 focus 一部分 position，则为 local attention。

由此可见，这里的 global attention、local attention 和 soft attention 并无本质上的区别，两篇 paper 模型的差别只是在 LSTM 结构上有微小的差别。

在 decoder 的时刻 t，在利用 global attention 或 local attention 得到 context vector Ct 之后，结合 ht，对二者做 concatenate 操作，得到 attention hidden state。

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t])$$

最后利用 softmax 产出该时刻的输出：

$$p(y_t|y_{<t}, x) = \text{softmax}(\boldsymbol{W_s}\tilde{\boldsymbol{h}}_t)$$

下面重点介绍 global attention、local attention。

1. global attention

global attention 在计算 context vector ct 的时候会考虑 encoder 所产生的全部hidden state。记 decoder 时刻 t 的 target hidden为 ht，encoder 的全部 hidden state 为

$$\bar{h}_s, s = 1, 2, \ldots, n$$

，对于其中任意

$\bar{h}_s$

，其权重 αts 为：

$$\begin{aligned}\boldsymbol{a}_t(s) &= \text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \\ &= \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}\end{aligned}$$

而其中的

$score(h_t, \bar{h}_s)$

，文章给出了四种种计算方法（文章称为 alignment function）：

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

$$\boldsymbol{a}_t = \text{softmax}(\boldsymbol{W_a} \boldsymbol{h}_t) \qquad location$$

四种方法都比较直观、简单。在得到这些权重后， ct 的计算是很自然的，即为

$\bar{h}_s$

的 weighted summation。

2. local attention

global attention 可能的缺点在于每次都要扫描全部的 source hidden state，计算开销较大，对于长句翻译不利，为了提升效率，提出 local attention，每次只 focus 一小部分的 source position。

这里，context vector ct 的计算只 focus 窗口 [pt-D,pt+D] 内的 2D+1 个source hidden states（若发生越界，则忽略界外的 source hidden states）。

其中 pt 是一个 source position index，可以理解为 attention 的“焦点”，作为模型的参数， D 根据经验来选择（文章选用 10）。 关于 pt 的计算，文章给出了两种计算方案：

- **Monotonic alignment (local-m)**

$$p_t = t$$

- **Predictive alignment (local-p)**

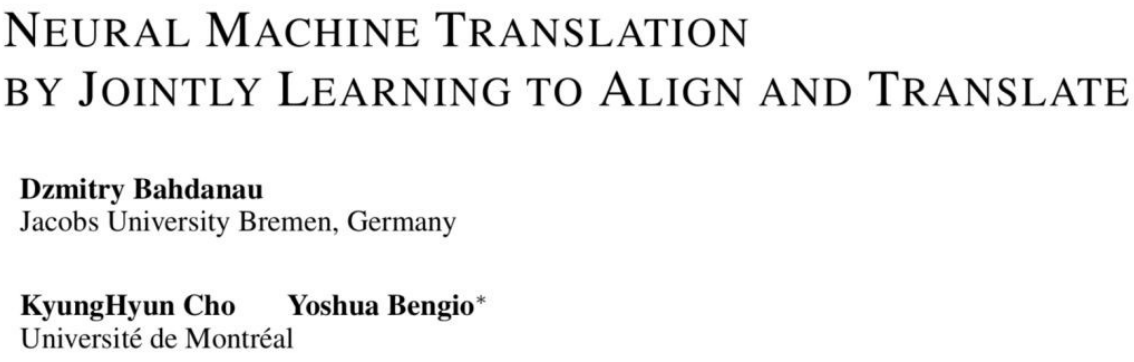
$$p_t = S \cdot \text{sigmoid}(\boldsymbol{v}_p^\top \tanh(\boldsymbol{W_p h}_t)),$$

其中 Wp 和 vp 是模型的参数， S 是 source sentence 的长度，易知 pt∈[0,S] 。 权重 αt(s) 的计算如下：

$$\boldsymbol{a}_t(s) = \text{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right)$$

可以看出，距离中心 pt 越远的位置，其位置上的 source hidden state 对应的权重就会被压缩地越厉害。

Jointly Learning



NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

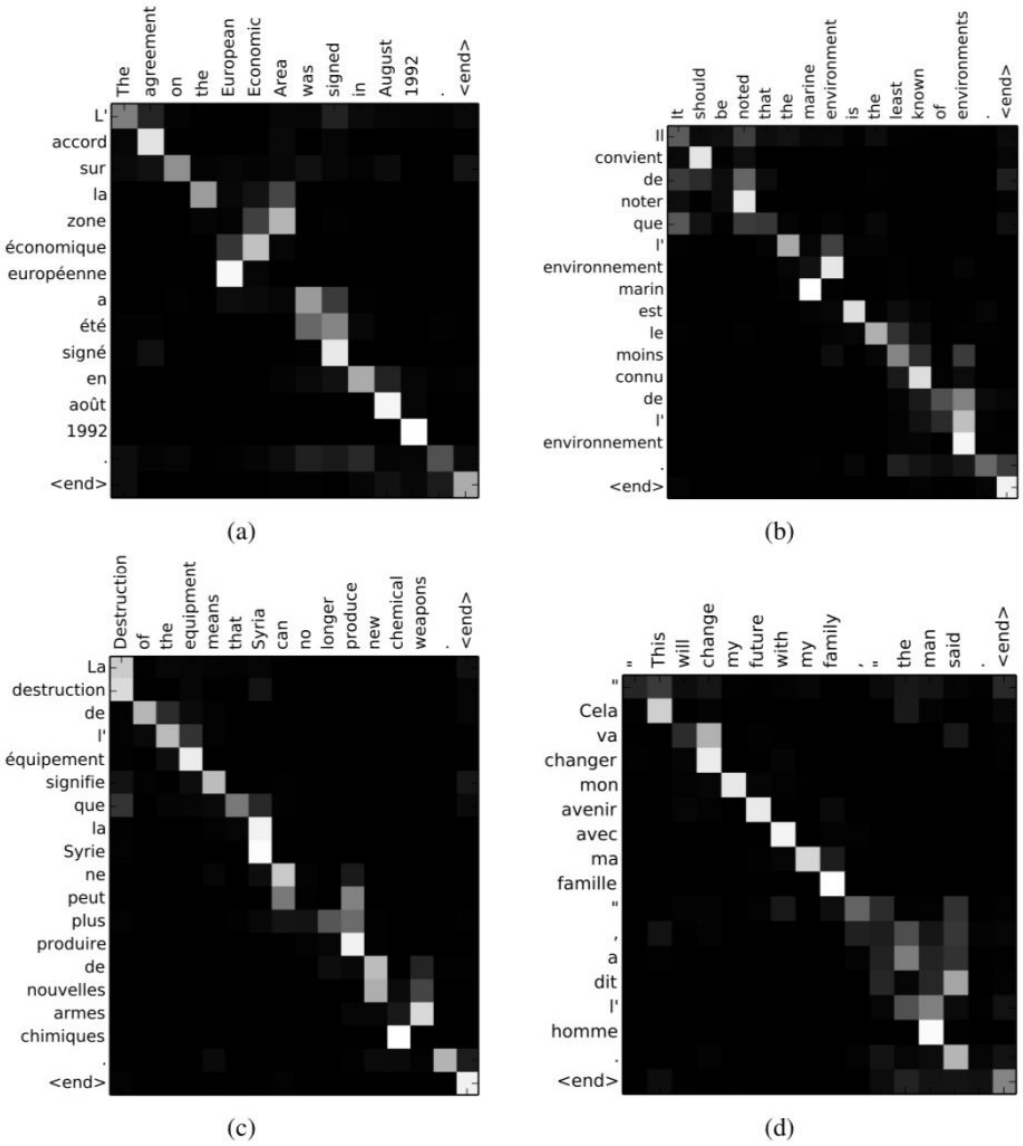
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

■ 论文 | Neural Machine Translation by Jointly Learning to Align and Translate

■ 链接 | https://www.paperweekly.site/papers/434

■ 源码 | https://github.com/spro/torch-seq2seq-attention

这篇文章没有使用新的 attention 结构，其 attention 就是 soft attention 的形式。文章给出了一些 attention 的可视化效果图。



上面 4 幅图中，x 轴代表原始英文句子，y 轴代表翻译为法文的结果。每个像素代表的是纵轴的相应位置的 target hidden state 与横轴相应位置的 source hidden state 计算得到的权重 αij，权重越大，对应的像素点越亮。可以看到，亮斑基本处在对角线上，符合预期，毕竟翻译的过程基本是沿着原始句子从左至右依次进行翻译。

Attention Is All You Need

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

■ 论文 | Attention Is All You Need

■ 链接 | https://www.paperweekly.site/papers/224

■ 源码 | https://github.com/Kyubyong/transformer

WEIGHTED TRANSFORMER NETWORK FOR MACHINE TRANSLATION

Karim Ahmed, Nitish Shirish Keskar & Richard Socher
Salesforce Research
Palo Alto, CA 94103, USA
{karim.ahmed,nkeskar,rsocher}@salesforce.com

■ 论文 | Weighted Transformer Network for Machine Translation

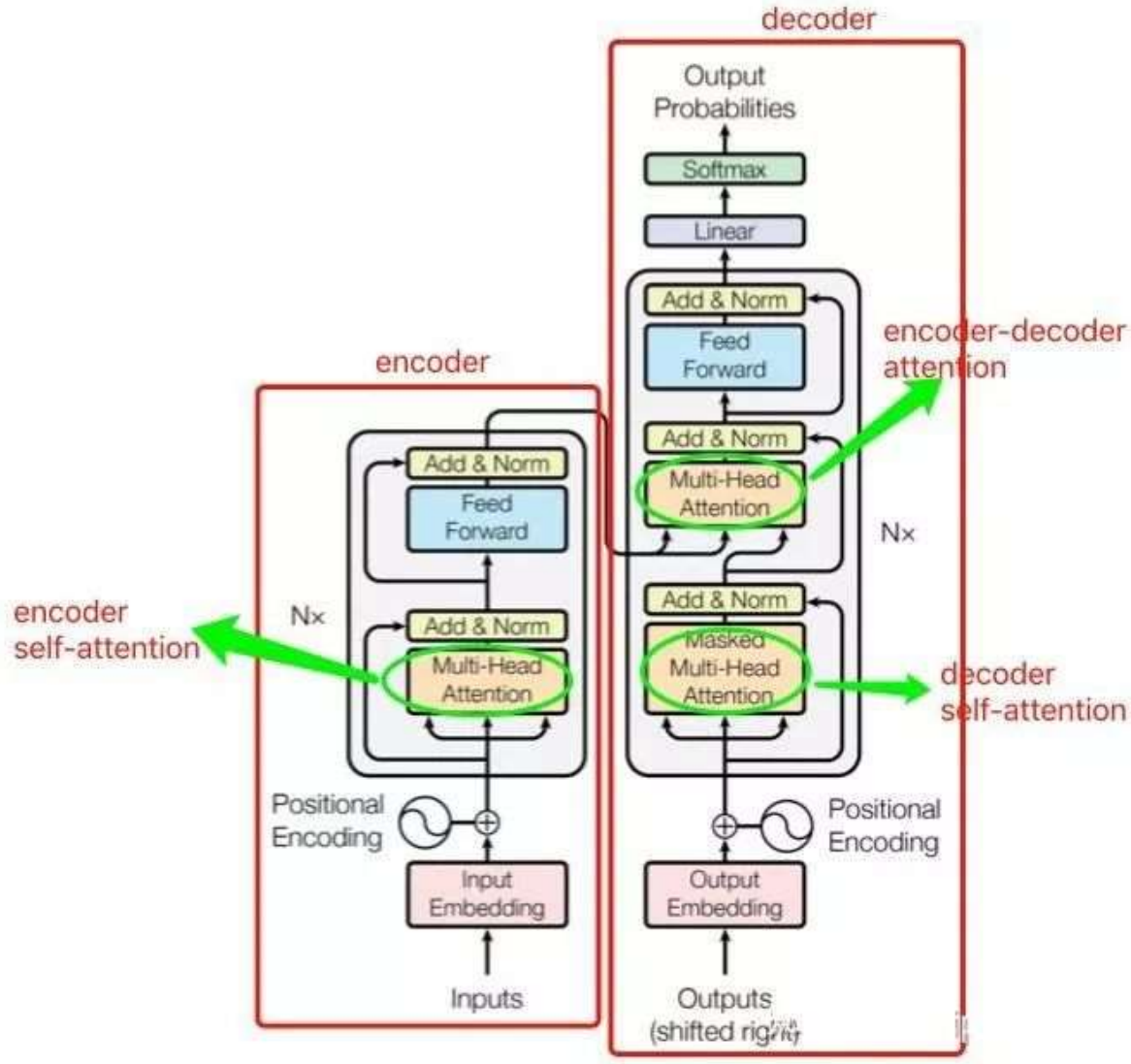
■ 链接 | https://www.paperweekly.site/papers/2013

■ 源码 | https://github.com/JayParks/transformer

作者首先指出，结合了 RNN（及其变体）和注意力机制的模型在序列建模领域取得了不错的成绩，但由于 RNN 的循环特性导致其不利于并行计算，所以模型的训练时间往往较长，在 GPU 上一个大一点的 seq2seq 模型通常要跑上几天，所以作者对 RNN 深恶痛绝，遂决定舍弃 RNN，只用注意力模型来进行序列的建模。

作者提出一种新型的网络结构，并起了个名字 Transformer，里面所包含的注意力机制称之为 self-attention。作者骄傲地宣称他这套 Transformer 是能够计算 input 和 output 的 representation 而不借助 RNN 的唯一的 model，所以作者说有 attention 就够了。

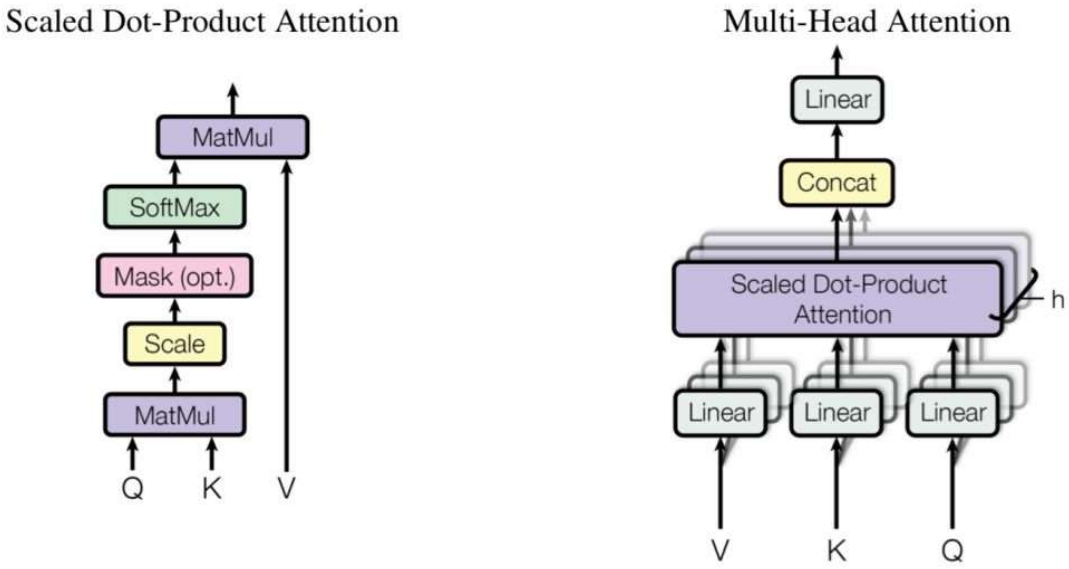
模型同样包含 encoder 和 decoder 两个 stage，encoder 和 decoder 都是抛弃 RNN，而是用堆叠起来的 self-attention，和 fully-connected layer 来完成，模型的架构如下：



从图中可以看出，模型共包含三个 attention 成分，分别是 encoder 的 self-attention，decoder 的 self-attention，以及连接 encoder 和 decoder 的 attention。

这三个 attention block 都是 multi-head attention 的形式，输入都是 query Q 、key K 、value V 三个元素，只是 Q 、 K 、 V 的取值不同罢了。接下来重点讨论最核心的模块 multi-head attention（多头注意力）。

multi-head attention 由多个 scaled dot-product attention 这样的基础单元经过 stack 而成。



那重点就变成 scaled dot-product attention 是什么鬼了。按字面意思理解，scaled dot-product attention 即缩放了的点乘注意力，我们来对它进行研究。

在这之前，我们先回顾一下上文提到的传统的 attention 方法（例如 global attention，score 采用 dot 形式）。

记 decoder 时刻 t 的 target hidden state 为 ht，encoder 得到的全部 source hidden state为

$$\bar{h}_s, s = 1, \ldots, n$$

，则 decoder 的 context vector ct 的计算过程如下：

$$e_t(s) = h_t^T * \bar{h}_s$$

$$\alpha_t(s) = \frac{exp(e_t(s))}{\sum_{i=1}^{n} exp(e_t(i))} = softmax(e_t(s))$$

$$c_t = \sum_{s=1}^{n} \alpha_t(s) * \bar{h}_s = \sum_{s=1}^{n} softmax(h_t^T * \bar{h}_s) * \bar{h}_s \quad ----------(*)$$

作者先抛出三个名词 query Q、key K、value V，然后计算这三个元素的 attention。

$$Attention(Q, K, V) = softmax(QK^T)V$$

我的写法与论文有细微差别，但为了接下来说明的简便，我姑且简化成这样。这个 Attention 的计算跟上面的 (*) 式有几分相似。

那么 Q、K、V 到底是什么？论文里讲的比较晦涩，说说我的理解。encoder 里的 attention 叫 self-attention，顾名思义，就是自己和自己做 attention。

抛开这篇论文的做法，让我们激活自己的创造力，在传统的 seq2seq 中的 encoder 阶段，我们得到 n 个时刻的 hidden states 之后，可以用每一时刻的 hidden state hi，去分别和任意的 hidden state hj,j=1,2,...,n 计算 attention，这就有点 self-attention 的意思。

回到当前的模型，由于抛弃了 RNN，encoder 过程就没了 hidden states，那拿什么做 self-attention 来自嗨呢?

可以想到，假如作为 input 的 sequence 共有 n 个 word，那么我可以先对每一个 word 做 embedding 吧？就得到 n 个 embedding，然后我就可以用 embedding 代替 hidden state 来做 self-attention 了。所以 Q 这个矩阵里面装的就是全部的 word embedding，K、V 也是一样。

所以为什么管 Q 叫query？就是你每次拿一个 word embedding，去“查询”其和任意的

word embedding 的 match 程度（也就是 attention 的大小），你一共要做 n 轮这样的操作。

我们记 word embedding 的 dimension 为 dmodel，所以 Q 的 shape 就是 n*dmodel，K、V 也是一样，第 i 个 word 的 embedding 为 vi，所以该 word 的 attention 应为：

$$Attention(Q_i, K, V) = softmax(v_i * [v_1^T, v_2^T, \dots, v_n^T]) * \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} = softmax(Q_i K^T)V$$

那同时做全部 word 的 attention，则是：

$$Attention(Q, K, V) = softmax(\begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} * [v_1^T, v_2^T, \dots, v_n^T]) * \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} = softmax(QK^T)V$$

scaled dot-product attention 基本就是这样了。基于 RNN 的传统 encoder 在每个时刻会有输入和输出，而现在 encoder 由于抛弃了 RNN 序列模型，所以可以一下子把序列的全部内容输进去，来一次 self-attention 的自嗨。

理解了 scaled dot-product attention 之后，multi-head attention 就好理解了，因为就是 scaled dot-product attention 的 stacking。

先把 Q、K、V 做 linear transformation，然后对新生成的 Q'、K'、V' 算 attention，重复这样的操作 h 次，然后把 h 次的结果做 concat，最后再做一次 linear transformation，就是 multi-head attention 这个小 block 的输出了。

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1,...,\mathrm{head}_\mathrm{h})W^O$$
$$\text{where } \mathrm{head}_\mathrm{i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

以上介绍了 encoder 的 self-attention。decoder 中的 encoder-decoder attention 道理类似，可以理解为用 decoder 中的每个 vi 对 encoder 中的 vj 做一种交叉 attention。

decoder 中的 self-attention 也一样的道理，只是要注意一点，decoder 中你在用 vi 对 vj 做 attention 时，有一些 pair 是不合法的。原因在于，虽然 encoder 阶段你可以把序列的全部 word 一次全输入进去，但是 decoder 阶段却并不总是可以，想象一下你在做 inference，decoder 的产出还是按从左至右的顺序，所以你的 vi 是没机会和 vj (j>i) 做 attention 的。

那怎么将这一点体现在 attention 的计算中呢？文中说只需要令 score(vi,vj)=-∞ 即可。为何？因为这样的话：

$$\alpha_{ij} = softmax(score(v_i, v_j)) = softmax(-\infty) = 0$$

所以在计算 vi 的 self-attention 的时候，就能够把 vj 屏蔽掉。所以这个问题也就解决了。

模型的其他模块，诸如 position-wise feed-forward networks、position encoding、layer normalization、residual connection 等，相对容易理解，感兴趣的同学可以去看 paper，此处不再赘述。

总结

本文对 attention 的五种结构，即 hard attention、soft attention、global attention、local attention、self-attention 进行了具体分析。五种 attention 在计算复杂度、部署难度、模型效果上会有一定差异，实际中还需根据业务实际合理选择模型。

• END •

From: PhilippZheng PaperWeekly