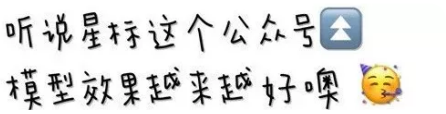
Transformer家族简史（PART II）

原创 kaiyuan NewBeeNLP 1月12日

收录于话题
#自然语言处理 43 #BERT巨人肩膀 44

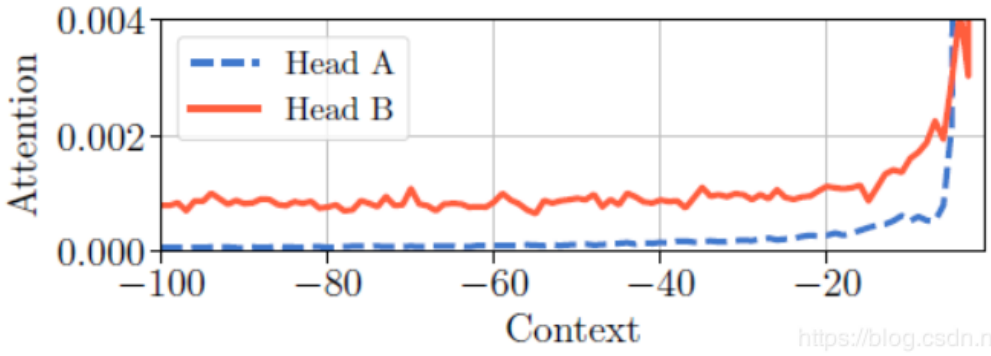


继 Transformer家族简史（PART I），PART II整理了三篇来自Facebook AI Research的论文，都比较好读：

- 「Span Transformer from FAIR，ACL2019」
- 「All-Attention from FAIR」
- 「PKM from FAIR，NeurIPS2019」

Adaptive Attention Span in Transformers[1]

这篇论文的重点是改进 Transformer 的计算效率，vanilla transformer 每个 attention head 处理的是等长的所有输入序列，但是在实验中发现 Transformer 不同 head 所关注的序列长度 span 是不一样的，一些 head（如 Head A）重点关注附近较短的信息，而另外一些 head（如 Head B）则关注在范围更大的全文。如果能在训练中利用这一特性，就可以显著减少计算时间和内存占用，因为两者都依赖于注意力范围的长度。

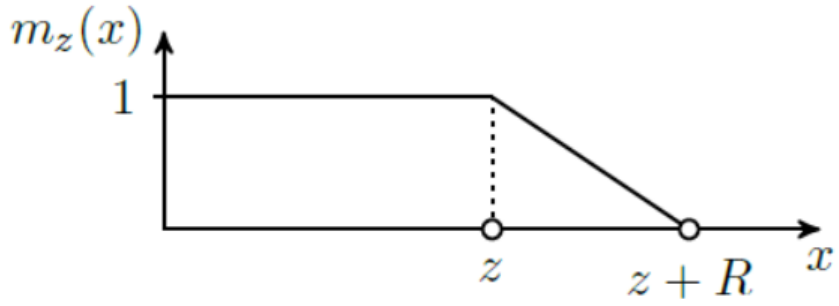


为此，作者提出一种「adaptive attention span」，可以让模型自适应地选择上下文长度进行处理。但是，attention span 的值是整数，因此是不可微的，不能像模型的其他参数那样通过反向传导直接学习它，于是通过 soft-masking function 来将其值转化为连续

值。masking 函数是非递增的，将跨度距离映射到[0,1]之间的值。这样，将 masking 函数应用到每一个 attention head 之后，就可以实现 attention span 的自适应控制。具体公式如下：

$$m_z(x) = \min\left[\max\left[\frac{1}{R}(R+z-x), 0\right], 1\right]$$

其中 R 是一个用来控制平滑度的超参，函数的形状如下图：



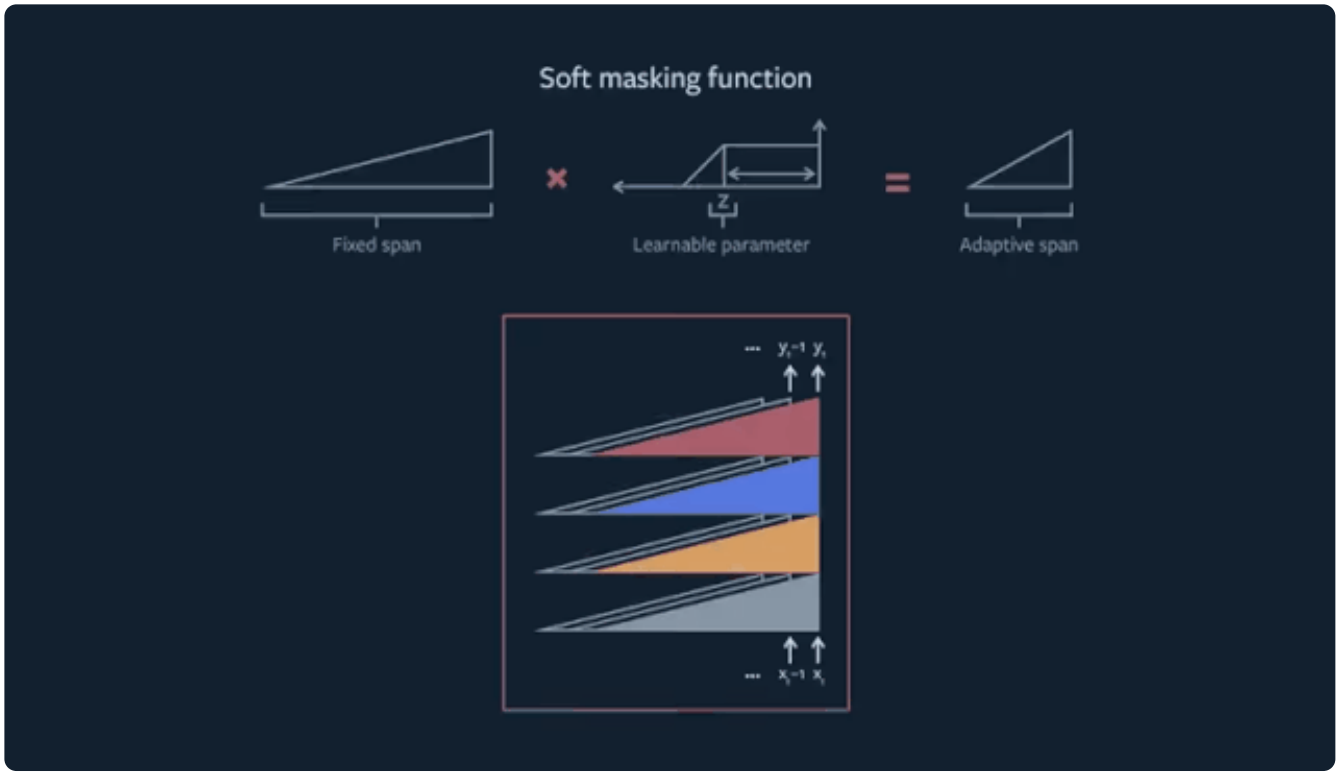
原始的 attention 计算公式改进为：

$$a_{tr} = \frac{m_z(t-r)\exp(s_{tr})}{\sum_{q=t-S}^{t-1} m_z(t-q)\exp(s_{tq})}$$

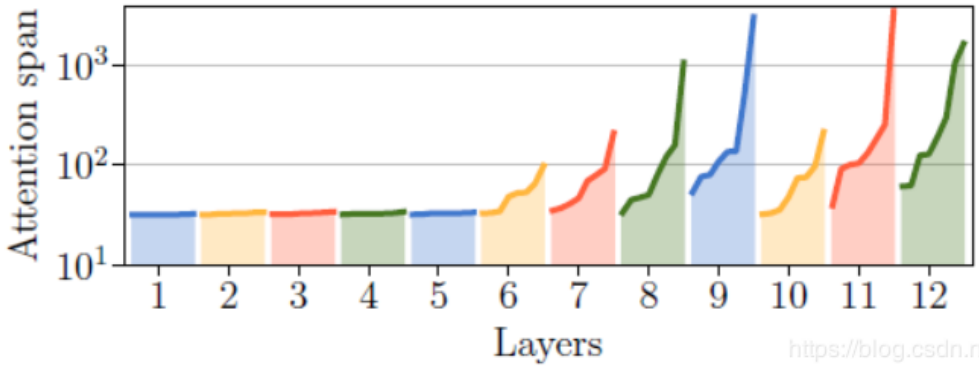
另外在损失函数中给 z 设置 L1 penalization：

$$L = -\log P(w_1, \ldots, w_T) + \frac{\lambda}{M}\sum_i z_i$$

整体过程如下动图，



- 此外，考虑了一种扩展「**dynamic attention span**」，根据输入动态调整 attention span；
- 在实现中，引用了Self-attention with relative position representations[2]和Transformer-XL[3]中的技巧；
- 实验结果显示在 12 层模型中，较低层的 attention span 较短，高层（8-12 层）的 attention span 较长



Reference

- Code Here[4]
- Making Transformer networks simpler and more efficient[5]
- Adaptive Attention Span in Transformers 分享视频[6]

Augmenting Self-attention with Persistent Memory[7]

当我们在讨论 Transformer 时，重点都在 self-attention 上，但是不要忘了网络中还有另外一层：前馈层 FFN，其包含了模型中最多的参数，大小通常是其他组件的四倍。FNN 的计算代价如此之高，那么有没有办法将模型简化呢？论文中提出将 FFN layer 替换为 attention layer，在不损失模型性能的前提下将模型结构大大简化。

2.1 FFN --> Attention

虽然表面上 FFN 和 attention 层看起来完全不同，但是通过将 FFN 中的 RELU 激活换成 Softmax 函数就可以将激活值转化为 attention weight。

- FFN

$$\mathrm{FF}(\mathrm{x}_t) = \mathbf{U}\sigma(\mathbf{V}_{\mathrm{x}_t}+\mathbf{b}) + \mathrm{c}$$

- FFN-->attention

$$\mathrm{y}_t = \mathrm{USoftmax}\left(\mathrm{Vx}_t\right) = \sum_{i=1}^{d_f} a_{ti}\mathrm{U}_{*,i}$$

我们把 V 看成 key，U 看成 value，是不是就跟 attention 的公式很像啦

2.2 All-attention

把 FFN 转化为 attention 之后就可以将原来的两层合并为一层，称为「**all-attention layer**」。

具体做法就是额外定义一组 key-value 向量对，称为「**persistent vectors**」，这些向量就和前馈子层的权值是一样的：固定的、可训练的且上下文无关的，可以捕获关于任务的 general knowledge。

- 定义 key 和 value，其中 M_k 和 M_v 是指 persistent vectors 对应的 key、value

$$[\mathbf{k}_1,\ldots,\mathbf{k}_{T+N}] = \mathrm{Concat}([\mathbf{W}_k\mathbf{x}_1,\ldots,\mathbf{W}_k\mathbf{x}_T],\mathbf{M}_k)$$
$$[\mathbf{v}_1,\ldots,\mathbf{v}_{T+N}] = \mathrm{Concat}([\mathbf{W}_v\mathbf{x}_1,\ldots,\mathbf{W}_v\mathbf{x}_T],\mathbf{M}_v)$$

- 计算 similarity score，其中 $p(t,c) = u_{t-c}$ 为相对位置编码

$$s_{tc} = \mathbf{x}_t^\top \mathbf{W}_q^\top (\mathbf{k}_c + \mathbf{p}(t,c))$$

- 计算输出，其中 a_{tc} 为 attention weight

$$\mathbf{y}_t = \sum_{c\in C_t^+} a_{tc}(\mathbf{v}_c + \mathbf{p}(t,c)) \text{ and } a_{tc} = \frac{\exp\left(s_{tc}/\sqrt{d_h}\right)}{\sum_{i\in C_t^+}\exp\left(s_{ti}/\sqrt{d_h}\right)}$$

扩展到多头注意力整体结构为，

$$\mathbf{y}_t = \text{ AddNorm }(\text{Mu1t iHeadAIIAttn }(\mathbf{x}_t))$$

2.3 Other tricks

- relative position embedding & catching mechanism[8]
- adaptive attention span[9]
- adaptive softmax[10]

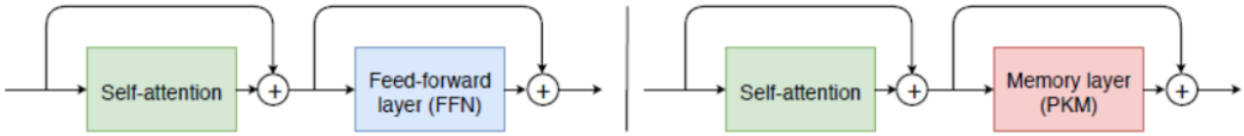
实验结果验证了FFN层和persistent vector的重要性，缺少的话效果非常差。每一层persistent vector的数量在N=1024时已经达到比较好的效果。

2.4 reference

- Code Here（没找到 - -）
- Making Transformer networks simpler and more efficient[11]
- Open Review[12]

Large Memory Layers with Product Keys[13]

同样来自FAIR的工作，解决的痛点：更好的模型性能-->更大的模型capacity-->更大的计算成本 。提出了一种structured memory，在明显增加模型capacity的同时计算成本的增加可以忽略不计，而且是简单可插拔式设计，下图是文中将vanilla transformer中的（部分）FFN层替换为memory layer的示例。

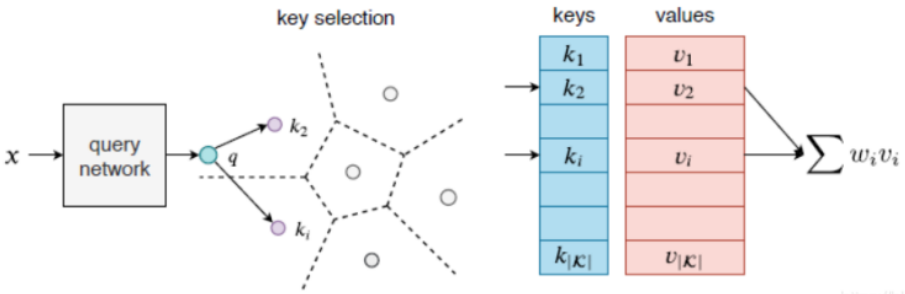


3.1 Overall Structure

来看整体的设计，包含了三个部分：

- 「**Query Network:** 」 通过函数 $q: x \mapsto q(x) \in \mathbb{R}^{d_q}$ 将 d 维输入降维映射到latent space生成维度为 $d_q = 512$ 的 query ；
- 「**Key Selection:** 」 计算query和每个key的相似度得分，挑选出top-k个，本文的关键工作主要在这一步，会在下文具体介绍；
- 「**Value Lookup:** 」 根据上一步得出的top-k个得分和value计算weighted sum；

整体对应的流程图和公式如下：



$$\mathcal{I} = \mathcal{T}_k\left(q(x)^T k_i\right)$$
$$w = \text{Softmax}\left(\left(q(x)^T k * i\right) * i \in \mathcal{I}\right)$$
$$m(x) = \sum *i \in \mathcal{I} w * iv\_i$$

其中 $\mathcal{I}$ 表示 k 个最相关的keys对应的下标。第二步和第三步式子只需要计算 k 个key，计算效率较高；但是第一步需要计算整个key集合的inner product，计算量非常大。

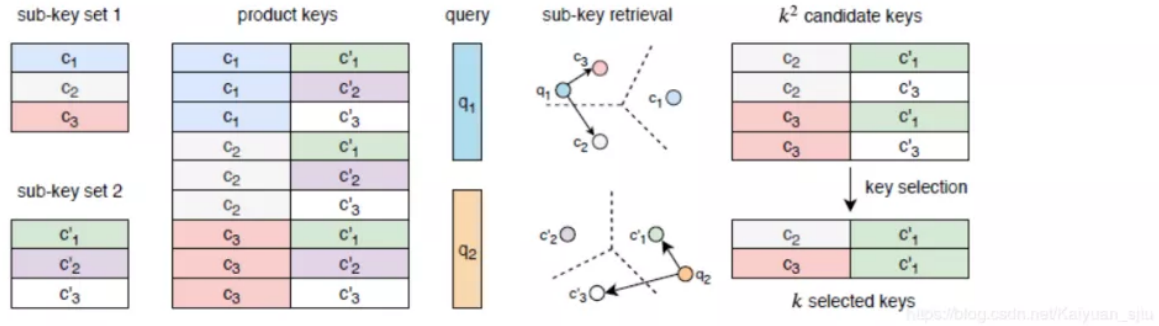
3.2 Product Keys

对上述公式第一步进行优化，思想来源于**product quantization**[14]，

> Product quantization，乘积量化，这里的乘积是指笛卡尔积（Cartesian product），意思是指把原来的向量空间分解为若干个低维向量空间的笛卡尔积，并对分

解得到的低维向量空间分别做量化（quantization）。这样每个向量就能由多个低维空间的量化组合表示。

目标是从总数为 K 的key集合中挑选出 k 个最相关的key，每个key维度为 d_q



1. 将原key集合拆分为两个subkeys，$\mathcal{C}$ 和 $\mathcal{C}'$，其中每个key的维度为 $d_q/2$，于是「**product keys**」可以表示为：

$$\mathcal{K}=\left\{\left(c,c'\right)|c\in\mathcal{C},c'\in\mathcal{C}'\right\}$$

2. query也拆分成对应的两个subquery，q_1 和 q_2；
3. 将这两个subquery与其对应的subkeys集合中的key进行相似度计算，例如 q_1 和 $\mathcal{C}$ 中的每个key计算得出「**topk**」：

$$\mathcal{I}_{\mathcal{C}}=\mathcal{T}_k\left(\left(q*1(x)^Tc*i\right)_{i\in\{1\ldots|\mathcal{C}|\}}\right),\quad \mathcal{I}_{\mathcal{C}'}=\mathcal{T}_k\left(\left(q_2(x)^Tc*j'\right)*j\in\left\{1\ldots\left|\mathcal{C}'\right|\right\}\right)$$

4. $\mathcal{I}_{\mathcal{C}}$ 和 $\mathcal{I}_{\mathcal{C}'}$ 中元素一一concat得到 k^2 个向量，最终从中选取 k 个，整体的复杂度为

$$\mathcal{O}\left(\left(\sqrt{|\mathcal{K}|}+k^2\right)\times d\_\mathrm{q}\right)$$

3.3

- 所有存储器参数都是可训练的，但是在训练时对于每个输入仅更新了少数（k 个）memory slots；
- 实验显示，模型增加内存比增加网络层数对效果提升更有效（12层+单内存的模型>24层无内存的模型）；
- 在query network配合使用「**Batch Normalization**」效果更佳；
- memory layer放置的最佳位置为模型网络的中间层；

3.4 reference

- Code Here[15]
- LeCun力荐：Facebook推出十亿参数超大容量存储器[16]

Over，以及预告马上会有的PART III。

一起交流

想和你一起学习进步！『**NewBeeNLP**』目前已经建立了多个不同方向交流群（**机器学习 / 深度学习 / 自然语言处理 / 搜索推荐 / 面试交流 /** 等），名额有限，赶紧添加下方微信加入一起讨论交流吧！（注意一定要**备注信息**才能通过）

本文参考资料

[1] **Adaptive Attention Span in Transformers:** https://www.aclweb.org/anthology/P19-1032/

[2] **Self-attention with relative position representations:** http://xxx.itp.ac.cn/pdf/1803.02155.pdf

[3] **Transformer-XL:** http://xxx.itp.ac.cn/pdf/1803.02155.pdf

[4] **Code Here:** https://github.com/facebookresearch/adaptive-span

[5] **Making Transformer networks simpler and more efficient:** https://ai.facebook.com/blog/making-transformer-networks-simpler-and-more-efficient/

[6] **Adaptive Attention Span in Transformers分享视频:** https://vimeo.com/384007585

[7] **Augmenting Self-attention with Persistent Memory:** https://arxiv.org/abs/1907.01470

[8] **relative position embedding & catching mechanism:** https://arxiv.org/abs/1901.02860

[9] **adaptive attention span:** https://arxiv.org/abs/1905.07799

[10] **adaptive softmax:** https://arxiv.org/pdf/1609.04309

[11] **Making Transformer networks simpler and more efficient:** ht