Google | 突破瓶颈，打造更强大的Transformer

原创 苏剑林 夕小瑶的卖萌屋 2020-04-15 22:24:08 手机阅读

来自专辑
卖萌屋@自然语言处理

一只小狐狸带你解锁**炼丹术&NLP**秘籍

> 作者：苏剑林（来自追一科技，人称“苏神”）

前言

《Attention is All You Need》一文发布后，基于Multi-Head Attention的Transformer模型开始流行起来，而去年发布的BERT模型更是将Transformer模型的热度推上了又一个高峰。当然，技术的探索是无止境的，改进的工作也相继涌现：有改进预训练任务的，比如XLNET的PLM、ALBERT的SOP等；有改进归一化的，比如Post-Norm向Pre-Norm的改变，以及T5中去掉了Layer Norm里边的beta参数等；也有改进模型结构的，比如Transformer-XL等；有改进训练方式的，比如ALBERT的参数共享等...

以上的这些改动，都是在Attention外部进行改动的，也就是说它们都默认了Attention的合理性，没有对Attention本身进行改动。而本文我们则介绍关于两个新结果：**它们针对Multi-Head Attention中可能存在建模瓶颈，提出了不同的方案来改进Multi-Head Attention。两篇论文都来自Google，并且做了相当充分的实验，因此结果应该是相当有说服力的了 。**

arxiv访问慢的小伙伴也可以在订阅号后台回复关键词【**0415**】下载论文PDF。

再小也不能小key_size

第一个结果来自文章《Low-Rank Bottleneck in Multi-head Attention Models》，它明确地指出了Multi-Head Attention里边的表达能力瓶颈，**并提出通过增大key_size的方法来缓解这个瓶颈。**

Multi-Head Attention

首先简单回顾一下Multi-Head Attention，Multi-Head Attention的基础是自然是Single-Head Attention，也叫Scaled-Dot Attention，定义如下：

（公式可以左右滑动哦）

$$Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{d_k}}\right)\boldsymbol{V} \quad （1）$$

其中$\boldsymbol{Q}\in\mathbb{R}^{n\times d_k},\boldsymbol{K}\in\mathbb{R}^{m\times d_k},\boldsymbol{V}\in\mathbb{R}^{m\times d_v}$。而Multi-Head Attention，就是将$\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}$分别用$h$个不同的投影矩阵投影$h$次，然后分别做$h$次Single-Head Attention，最后把结果拼接起来，即

$$\begin{aligned}
&\boldsymbol{Q}^{(1)}=\boldsymbol{Q}\boldsymbol{W}_Q^{(1)},\boldsymbol{K}^{(1)}=\boldsymbol{K}\boldsymbol{W}_K^{(1)},\boldsymbol{V}^{(1)}=\boldsymbol{V}\boldsymbol{W}_V^{(1)},\boldsymbol{O}^{(1)}=Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})\\
&\boldsymbol{Q}^{(2)}=\boldsymbol{Q}\boldsymbol{W}_Q^{(2)},\boldsymbol{K}^{(2)}=\boldsymbol{K}\boldsymbol{W}_K^{(2)},\boldsymbol{V}^{(2)}=\boldsymbol{V}\boldsymbol{W}_V^{(2)},\boldsymbol{O}^{(2)}=Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})\\
&\qquad\qquad\vdots\\
&\boldsymbol{Q}^{(h)}=\boldsymbol{Q}\boldsymbol{W}_Q^{(h)},\boldsymbol{K}^{(h)}=\boldsymbol{K}\boldsymbol{W}_K^{(h)},\boldsymbol{V}^{(h)}=\boldsymbol{V}\boldsymbol{W}_V^{(h)},\boldsymbol{O}^{(h)}=Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})\\
&\boldsymbol{O}=\left[\boldsymbol{O}^{(1)},\boldsymbol{O}^{(2)},\dots,\boldsymbol{O}^{(h)}\right]
\end{aligned}$$

Attention里有个瓶颈

在实际使用中，$\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}$一般具有相同的特征维度$d_k=d_v=d$（即hidden_size），比如BERT Base里边是768；h一般选择12、16、24等，比如BERT base里边是12；确定了d,h之后，通常的选择是让投影矩阵$\boldsymbol{W}\in\mathbb{R}^{d\times(d/h)}$，也就是说，每个Attention Head里边，是将原始的d维投影到d/h维，然后在进行Attention运算，输出也是d/h维，最后把h个d/h维的结果拼接起来，得到一个d维的输出。这里的d/h我们通常称为head_size。

在Attention中，关键的一步是

$$\boldsymbol{P}=softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\top}}{\sqrt{d_k}}\right) \quad （2）$$

这一步是描述了$\boldsymbol{Q}$与$\boldsymbol{K}$的两两向量之间的联系，我们可以将$\boldsymbol{P}$看成一个二元联合分布（实际上是n个一元分布，不过这个细节并不重要），如果序列长度都为n，也就是每个元有n个可能的取值，那么这个分布共有n^2个值。

但是，我们将$\boldsymbol{Q},\boldsymbol{K}$分别投影到低维后，各自的参数量只有$n\times(d/h)$，总的参数量是$2nd/h$，所以式$(2)$就相当于用$2nd/h$的参数量去逼近一个本身有$n^2$个值的量，而我们通常有$2nd/h\ll n^2$，尤其是$h$比较大时更是如此，因此**这种建模有点“强模型所难”，这就是原论文中的“低秩瓶颈（Low-Rank Bottleneck）”的含义.**

不妨试试增大key_size?

那么，解决办法是什么呢？直接的想法是让$2nd/h$增大，所以要不就是减少head的数目h，要不就是增加hidden_size大小d。但是更多的Attention Head本身也能增强模型的表达能力，所以为了缓解低秩瓶颈而减少h的做法可能得不偿失；如果增加d的话，那自然是能够增强模型整体表达能力的，但整个模型的规模与计算量也会剧增，似乎也不是一个好选择。

那没有其他办法了吗？有！当我们用投影矩阵将$\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}$都投影到低维时，前面都是将它们投影到$d/h$维，但其实它们的维度不一定要相等，而是只需要保证$\boldsymbol{Q},\boldsymbol{K}$的维度相等就行了（因为要做内积），为了区别，我们通常称$\boldsymbol{Q},\boldsymbol{K}$的维度为key_size，$\boldsymbol{V}$的维度才叫head_size，改变key_size的大小而不改变head_size的话，也不影响模型的hidden_size。

所以，这篇论文提出来的解决方法就是增大模型的key_size，它能增加Attention的表达能力，并且不改变模型整体的hidden_size，计算量上也只是稍微增加了一点。

> **补充说明**：事实上原论文考虑的是同时增大key_size和head_size、然后Multi-Head Attention的输出拼接之后再用一个变换矩阵降维，但笔者认为由于拼接降维这一步只是一个线性变换，所以本质上的提升还是来源于增大key_size，所以本文只强调了增大key_size这一步。此外，如果同时增大key_size和head_size，那么会导致计算量和显存消耗都明显增加，而只增大key_size的话，增加的资源消耗就小很多了。

来看看实验结果～

增大key_size这个想法很简单，也容易实现，但是否真的有效呢？我们来看看原论文的实验结果，其实验都是以BERT为baseline的，实验结果图表很多，大家直接看原论文为好，这里只分享比较有代表性的一个：

这个结果显示，如果固定一个比较大的key_size（比如128），那么我们可以调整模型的hidden_size和head数，使得参数量可以跟原始的BERT设计一致，但是效果更优！所以，增加key_size确实是有意义的，哪怕将总体参数量重新调整到原来的一样大，也能一定程度上提升模型的效果。这无疑对我们设计新的Transformer模型（尤其是小规模的模型）有重要的指导作用。

最后，附上我们预训练的两个增大了key_size的RoBERTa小模型，欢迎大家使用（我们称之为**RoBERTa⁺**）：

> https://github.com/ZhuiyiTechnology/pretrained-models

再缺也不能缺Talking

对Multi-Head Attention改进的第二个结果来自论文《Talking-Heads Attention》，这篇论文虽然没有显式地指出它跟前一篇论文的联系，但笔者认为它们事实上在解决同一个问题，只不过思路不一样：**它指出当前的Multi-Head Attention每个head的运算是相互孤立的，而通过将它们联系（Talking）起来，则可以得到更强的Attention设计，即标题的“Talking-Heads Attention”。**

从单一分布到混合分布

在前一篇论文里边，我们提到了低秩瓶颈，也就是由于key_size太小所以$\boldsymbol{Q}^{(i)}\boldsymbol{K}^{(i)\top}$表达能力不足，因此softmax之后无法很好地建议完整的二元分布。为了缓解这个问题，除了增大key_size之外，还有没有其他方法呢？有，比如这篇论文使用的混合分布思路。

所谓混合分布，就是多个简单分布的叠加（比如加权平均），它能极大地增强原分布的表达能力。典型的例子是高斯混合模型：我们知道高斯分布只是一个常见的简单分布，但多个高斯分布叠加而成的高斯混合分布（也叫高斯混合模型，GMM）就是一个更强的分布，理论上来说，只要叠加的高斯分布足够多，高斯混合分布能逼近任意概率分布。这个例子告诉我们，想要增加Attention中分布的表达能力，又不想增加key_size，那么可以考虑**叠加多个低秩分布**。

那么“多个”低秩分布哪里来呢？不是有Multi-Head嘛，每个head都带有一个低秩分布，就直接用它们叠加就行了，这就是Talking-Heads Attention了。具体来说，它的形式是：

$$\begin{aligned}
&\hat{\boldsymbol{J}}^{(1)}=\boldsymbol{Q}^{(1)}\boldsymbol{K}^{(1)\top},\quad \hat{\boldsymbol{J}}^{(2)}=\boldsymbol{Q}^{(2)}\boldsymbol{K}^{(2)\top},\quad\cdots,\quad \hat{\boldsymbol{J}}^{(h)}=\boldsymbol{Q}^{(h)}\boldsymbol{K}^{(h)\top}\\
&\begin{pmatrix}\boldsymbol{J}^{(1)}\\ \boldsymbol{J}^{(2)}\\ \vdots\\ \boldsymbol{J}^{(h)}\end{pmatrix}=\begin{pmatrix}\lambda_{11} & \lambda_{12} & \cdots & \lambda_{1h}\\ \lambda_{21} & \lambda_{22} & \cdots & \lambda_{2h}\\ \vdots & \vdots & \ddots & \vdots\\ \lambda_{h1} & \lambda_{h2} & \cdots & \lambda_{hh}\end{pmatrix}\begin{pmatrix}\hat{\boldsymbol{J}}^{(1)}\\ \hat{\boldsymbol{J}}^{(2)}\\ \vdots\\ \hat{\boldsymbol{J}}^{(h)}\end{pmatrix}\\
&\boldsymbol{P}^{(1)}=softmax\left(\boldsymbol{J}^{(1)}\right),\boldsymbol{P}^{(2)}=softmax\left(\boldsymbol{J}^{(2)}\right),\dots,\boldsymbol{P}^{(h)}=softmax\left(\boldsymbol{J}^{(h)}\right)\\
&\boldsymbol{O}^{(1)}=\boldsymbol{P}^{(1)}\boldsymbol{V}^{(1)},\quad \boldsymbol{O}^{(2)}=\boldsymbol{P}^{(2)}\boldsymbol{V}^{(2)},\quad ,\cdots,\quad \boldsymbol{O}^{(h)}=\boldsymbol{P}^{(h)}\boldsymbol{V}^{(h)}\\
&\boldsymbol{O}=\left[\boldsymbol{O}^{(1)},\boldsymbol{O}^{(2)},\dots,\boldsymbol{O}^{(h)}\right]
\end{aligned}$$

写起来很复杂，事实上很简单，**就是在“$\boldsymbol{Q}\boldsymbol{K}^{\top}$之后、softmax之前”用一个参数矩阵λ将各个$\boldsymbol{Q}\boldsymbol{K}^{\top}$的结果叠加一下而已。**这样就把原本是孤立的各个Attention Head联系了起来，即做了一个简单的Talking。

对上述公式，做两点补充说明：

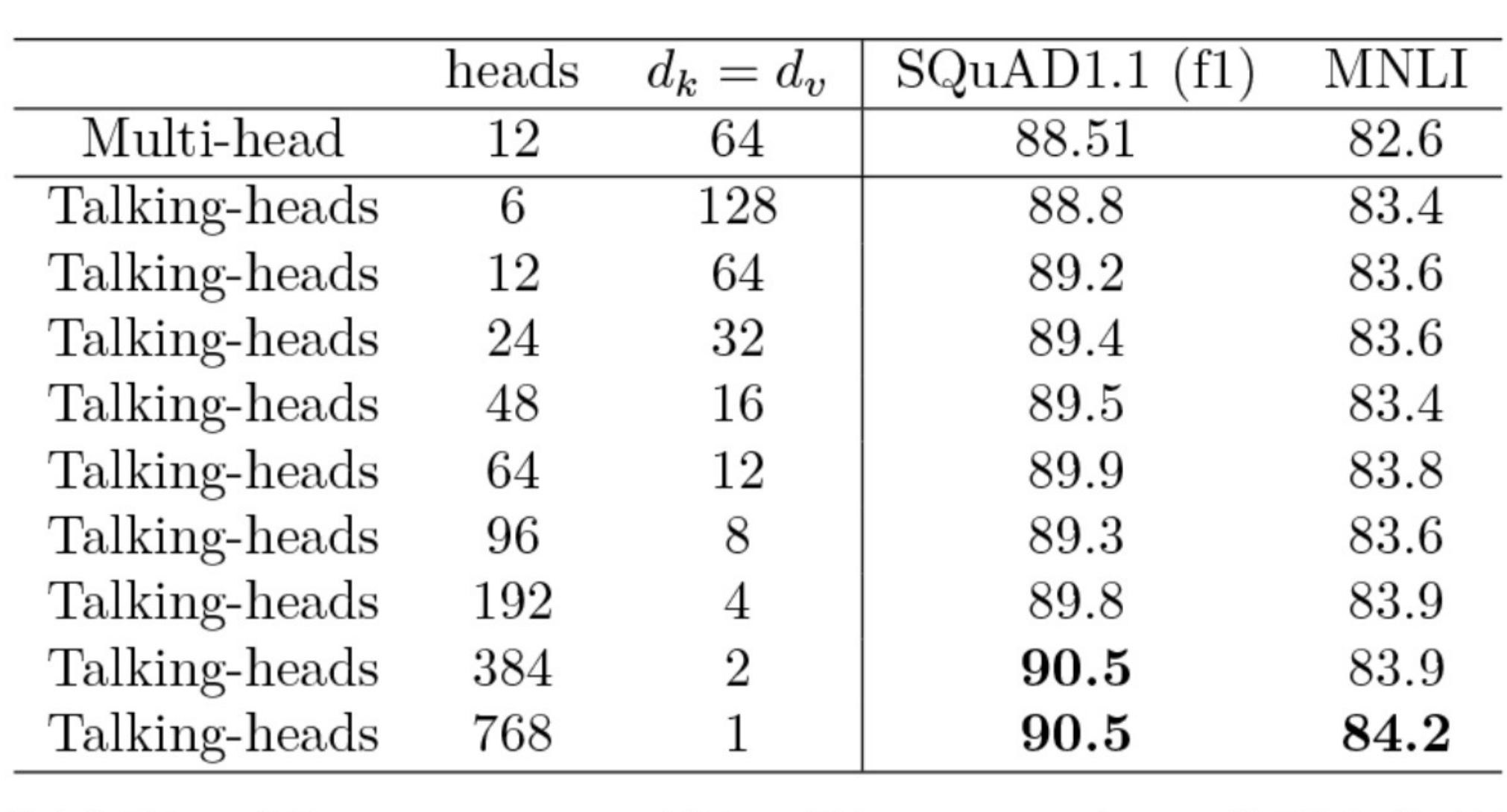
> 1、简单起见，上述公式中笔者省去了缩放因子$\sqrt{d_k}$，如果有需要，读者自行补充上去即可；
>
> 2、更一般的Talking-Heads Attention允许可以在$\boldsymbol{J}=\boldsymbol{\lambda}\hat{\boldsymbol{J}}$这一步进行升维，即叠加出多于$h$个混合分布，然后再用另一个参数矩阵降维，但这并不是特别重要的改进，所以不在主要篇幅介绍。

再来看看实验结果

是不是真的有效，当然还是得靠实验结果来说话。这篇论文的实验阵容可谓空前强大，它同时包含了BERT、ALBERT、T5为baseline的实验结果！众所周知，BERT、ALBERT、T5均是某个时间段的NLP最优模型，尤其是T5还是处在superglue的榜首，并且远超出第二名很多，而这个Talking-Heads Attention则几乎是把它们的辉煌战绩又刷到了一个新高度！

还是那句话，具体的实验结果大家自己看论文去，这里展示一个比较典型的结果：

Table 7: Talking-Heads attention on BERT.

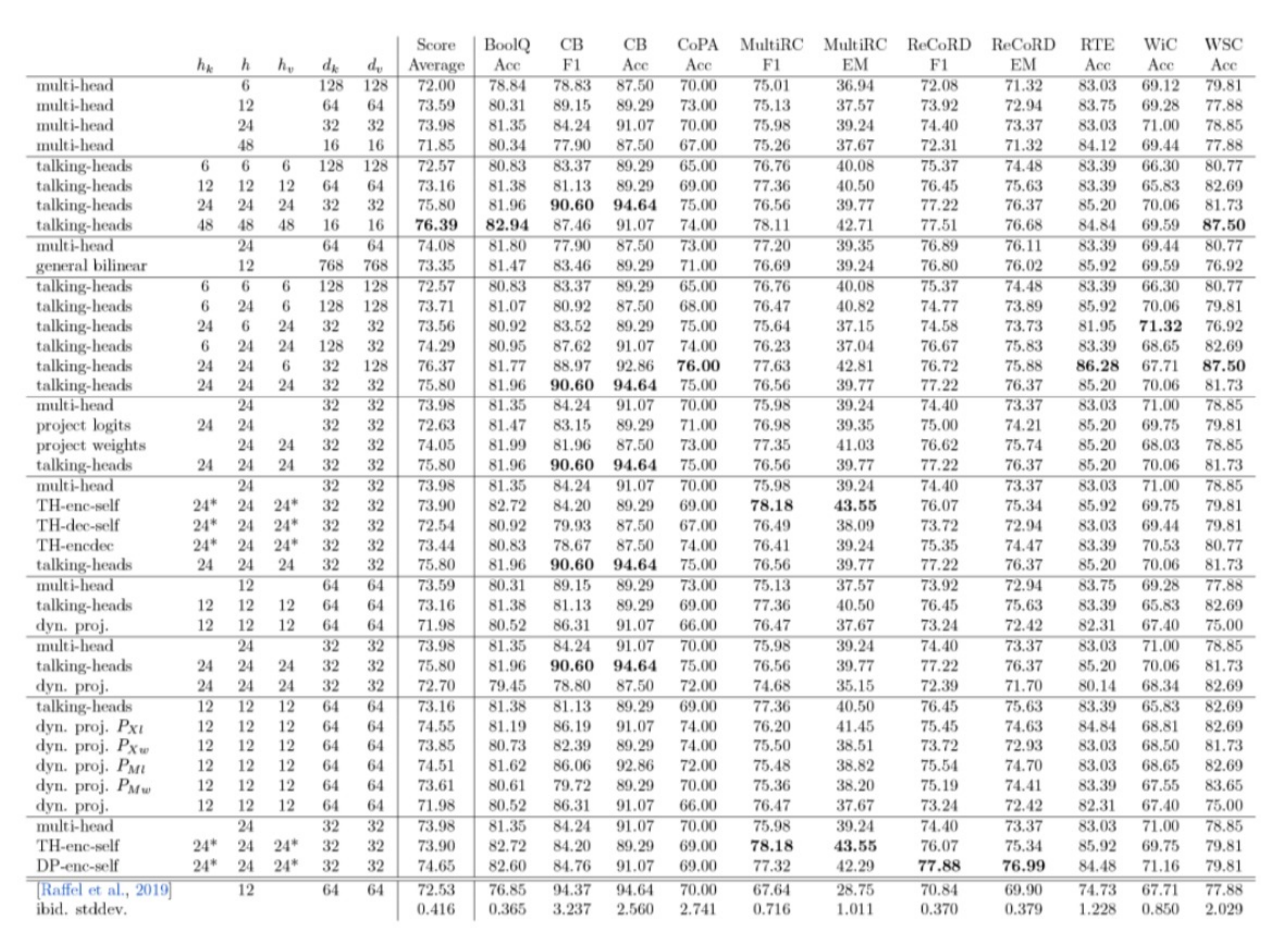


	heads	$d_k=d_v$	SQuAD1.1 (f1)	MNLI
Multi-head	12	64	88.51	82.6
Talking-heads	6	128	88.8	83.4
Talking-heads	12	64	89.2	83.6
Talking-heads	24	32	89.4	83.6
Talking-heads	48	16	89.5	83.4
Talking-heads	64	12	89.9	83.8
Talking-heads	96	8	89.3	83.6
Talking-heads	192	4	89.8	83.9
Talking-heads	384	2	**90.5**	83.9
Talking-heads	768	1	**90.5**	**84.2**

这个结果显示，使用Talking-Head Attention情况下，保持hidden_size不变，head数目越大（相应地key_size和head_size都越小），效果越优。这看起来跟前一篇增大key_size的结论矛盾，但事实上这正说明了混合分布对分布拟合能力明显提升作用，能够将key_size缩小时本身变弱的单一分布，叠加成拟合能力更强大的分布。当然，这不能说明就直接设key_size=1就好了，因为key_size=1时计算量会远远大于原始的BERT base，应用时需要根据实际情况平衡效果和计算量。

上述表格只是原论文实验结果的冰山一角，这里再放出一个实验表格，让大家感受感受它的实验阵容：

Table 12: T5 on SuperGLUE Language-Understanding Benchmark [Wang et al., 2019] (dev). Experiments described in Section 7.1 and appendix A.



	h_k	h	h_v	d_k	d_v	Score Average	BoolQ Acc	CB F1	CB Acc	CoPA Acc	MultiRC F1	MultiRC EM	ReCoRD F1	ReCoRD EM	RTE Acc	WiC Acc	WSC Acc
multi-head		6		128	128	72.00	78.84	78.83	87.50	70.00	75.01	36.94	72.08	71.32	83.03	69.12	79.81
multi-head		12		64	64	73.59	80.31	89.15	89.29	73.00	75.13	37.57	73.92	72.94	83.75	69.28	77.88
multi-head		24		32	32	73.98	81.35	84.24	91.07	70.00	75.98	39.24	74.40	73.37	83.03	71.00	78.85
multi-head		48		16	16	71.85	80.34	77.90	87.50	67.00	75.26	37.67	72.31	71.32	84.12	69.44	77.88
talking-heads	6	6	6	128	128	72.57	80.83	83.37	89.29	65.00	76.76	40.08	75.37	74.48	83.39	66.30	80.77
talking-heads	12	12	12	64	64	73.16	81.38	81.13	89.29	69.00	77.36	40.50	76.45	75.63	83.39	65.83	82.69
talking-heads	24	24	24	32	32	75.80	81.96	**90.60**	**94.64**	75.00	76.56	39.77	77.22	76.37	85.20	70.06	81.73
talking-heads	48	48	48	16	16	**76.39**	**82.94**	87.46	91.07	74.00	78.11	42.71	77.51	76.68	84.84	69.59	**87.50**
multi-head		24		64	64	74.08	81.80	77.90	87.50	73.00	77.20	39.35	76.89	76.11	83.39	69.44	80.77
general bilinear		12		768	768	73.35	81.47	83.46	89.29	71.00	76.69	39.24	76.80	76.02	85.92	69.59	76.92
talking-heads	6	6	6	128	128	72.57	80.83	83.37	89.29	65.00	76.76	40.08	75.37	74.48	83.39	66.30	80.77
talking-heads	6	24	6	128	128	73.71	81.07	80.92	87.50	68.00	76.47	40.82	74.77	73.89	85.92	70.06	79.81
talking-heads	24	6	24	32	32	73.56	80.92	83.52	89.29	75.00	75.64	37.15	74.58	73.73	81.95	**71.32**	76.92
talking-heads	6	24	24	128	32	74.29	80.95	87.62	91.07	74.00	76.23	37.04	76.67	75.83	83.39	68.65	82.69
talking-heads	24	24	6	32	128	76.37	81.77	88.97	92.86	**76.00**	77.63	42.81	76.72	75.88	**86.28**	67.71	**87.50**
talking-heads	24	24	24	32	32	75.80	81.96	**90.60**	**94.64**	75.00	76.56	39.77	77.22	76.37	85.20	70.06	81.73
multi-head		24		32	32	73.98	81.35	84.24	91.07	70.00	75.98	39.24	74.40	73.37	83.03	71.00	78.85
project logits	24	24		32	32	72.63	81.47	83.15	89.29	71.00	76.98	39.35	75.00	74.21	85.20	69.75	79.81
project weights		24	24	32	32	74.05	81.99	81.96	87.50	73.00	77.35	41.03	76.62	75.74	85.20	68.03	78.85
talking-heads	24	24	24	32	32	75.80	81.96	**90.60**	**94.64**	75.00	76.56	39.77	77.22	76.37	85.20	70.06	81.73
multi-head		24		32	32	73.98	81.35	84.24	91.07	70.00	75.98	39.24	74.40	73.37	83.03	71.00	78.85
TH-enc-self	24*	24	24*	32	32	73.90	82.72	84.20	89.29	69.00	**78.18**	**43.55**	76.07	75.34	85.92	69.75	79.81
TH-dec-self	24*	24	24*	32	32	72.54	80.92	79.93	87.50	67.00	76.49	38.09	73.72	72.94	83.03	69.44	79.81
TH-encdec	24*	24	24*	32	32	73.44	80.83	78.67	87.50	74.00	76.41	39.24	75.35	74.47	83.39	70.53	80.77
talking-heads	24	24	24	32	32	75.80	81.96	**90.60**	**94.64**	75.00	76.56	39.77	77.22	76.37	85.20	70.06	81.73
multi-head		12		64	64	73.59	80.31	89.15	89.29	73.00	75.13	37.57	73.92	72.94	83.75	69.28	77.88
talking-heads	12	12	12	64	64	73.16	81.38	81.13	89.29	69.00	77.36	40.50	76.45	75.63	83.39	65.83	82.69
dyn. proj.	12	12	12	64	64	71.98	80.52	86.31	91.07	66.00	76.47	37.67	73.24	72.42	82.31	67.40	75.00
multi-head		24		32	32	73.98	81.35	84.24	91.07	70.00	75.98	39.24	74.40	73.37	83.03	71.00	78.85
talking-heads	24	24	24	32	32	75.80	81.96	**90.60**	**94.64**	75.00	76.56	39.77	77.22	76.37	85.20	70.06	81.73
dyn. proj.	24	24	24	32	32	72.70	79.45	78.80	87.50	72.00	74.68	35.15	72.39	71.70	80.14	68.34	82.69
talking-heads	12	12	12	64	64	73.16	81.38	81.13	89.29	69.00	77.36	40.50	76.45	75.63	83.39	65.83	82.69
dyn. proj. P_{Xl}	12	12	12	64	64	74.55	81.19	86.19	91.07	74.00	76.20	41.45	75.45	74.63	84.84	68.81	82.69
dyn. proj. P_{Xw}	12	12	12	64	64	73.85	80.73	82.39	89.29	74.00	75.50	38.51	73.72	72.93	83.03	68.50	81.73
dyn. proj. P_{Ml}	12	12	12	64	64	74.51	81.62	86.06	92.86	72.00	75.48	38.82	75.54	74.70	83.03	68.65	82.69
dyn. proj. P_{Mw}	12	12	12	64	64	73.61	80.61	79.72	89.29	70.00	75.36	38.20	75.19	74.41	83.39	67.55	83.65
dyn. proj.	12	12	12	64	64	71.98	80.52	86.31	91.07	66.00	76.47	37.67	73.24	72.42	82.31	67.40	75.00
multi-head		24		32	32	73.98	81.35	84.24	91.07	70.00	75.98	39.24	74.40	73.37	83.03	71.00	78.85
TH-enc-self	24*	24	24*	32	32	73.90	82.72	84.20	89.29	69.00	**78.18**	**43.55**	76.07	75.34	85.92	69.75	79.81
DP-enc-self	24*	24	24*	32	32	74.65	82.60	84.76	91.07	69.00	77.32	42.29	**77.88**	**76.99**	84.48	71.16	79.81
[Raffel et al., 2019]		12		64	64	72.53	76.85	94.37	94.64	70.00	67.64	28.75	70.84	69.90	74.73	67.71	77.88
ibid. stddev.						0.416	0.365	3.237	2.560	2.741	0.716	1.011	0.370	0.379	1.228	0.850	2.029

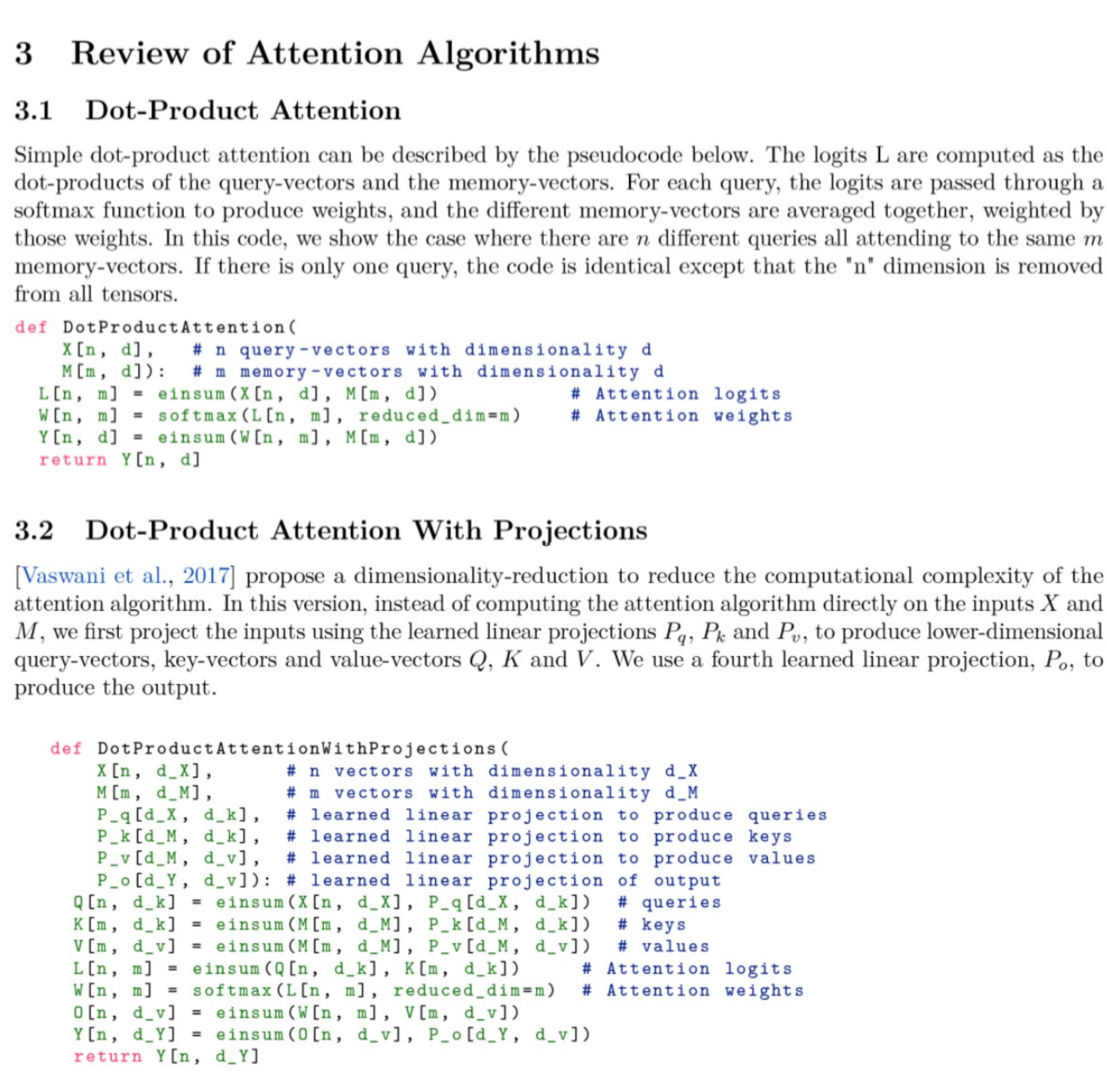
几乎每个任务、每个超参组合都做了实验，并给出实验结果。如此强大的实验阵容，基本上也就只有Google能搞出来了，而且整篇论文明显是浓浓的“T5 Style”（还没看过T5论文的读者，可以去Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer》感受一下），果不其然，作者之一Noam Shazeer也正是T5的作者之一。

笔者只想说，这种庞大的实验轰炸，仿佛在向我们宣告着：

> 不用质疑，该调的参数我们都调了，就我们的Talking-Heads Attention最好～

插曲：神奇的论文画风

话说回来，笔者在Arxiv上首次刷到《Talking-Heads Attention》这篇论文时，第一感觉是一篇垃圾论文。为啥？因为它的画风是这样的：



3 Review of Attention Algorithms

3.1 Dot-Product Attention

Simple dot-product attention can be described by the pseudocode below. The logits L are computed as the dot-products of the query-vectors and the memory-vectors. For each query, the logits are passed through a softmax function to produce weights, and the different memory-vectors are averaged together, weighted by those weights. In this code, we show the case where there are n different queries all attending to the same m memory-vectors. If there is only one query, the code is identical except that the "n" dimension is removed from all tensors.

```
def DotProductAttention(
    X[n, d],    # n query-vectors with dimensionality d
    M[m, d]):   # m memory-vectors with dimensionality d
  L[n, m] = einsum(X[n, d], M[m, d])          # Attention logits
  W[n, m] = softmax(L[n, m], reduced_dim=m)   # Attention weights
  Y[n, d] = einsum(W[n, m], M[m, d])
  return Y[n, d]
```

3.2 Dot-Product Attention With Projections

[Vaswani et al., 2017] propose a dimensionality-reduction to reduce the computational complexity of the attention algorithm. In this version, instead of computing the attention algorithm directly on the inputs X and M, we first project the inputs using the learned linear projections P_q, P_k and P_v, to produce lower-dimensional query-vectors, key-vectors and value-vectors Q, K and V. We use a fourth learned linear projection, P_o, to produce the output.

```
def DotProductAttentionWithProjections(
    X[n, d_X],        # n vectors with dimensionality d_X
    M[m, d_M],        # m vectors with dimensionality d_M
    P_q[d_X, d_k],    # learned linear projection to produce queries
    P_k[d_M, d_k],    # learned linear projection to produce keys
    P_v[d_M, d_v],    # learned linear projection to produce values
    P_o[d_Y, d_v]):   # learned linear projection of output
  Q[n, d_k] = einsum(X[n, d_X], P_q[d_X, d_k])  # queries
  K[m, d_k] = einsum(M[m, d_M], P_k[d_M, d_k])  # keys
  V[m, d_v] = einsum(M[m, d_M], P_v[d_M, d_v])  # values
  L[n, m] = einsum(Q[n, d_k], K[m, d_k])      # Attention logits
  W[n, m] = softmax(L[n, m], reduced_dim=m)  # Attention weights
  O[n, d_v] = einsum(W[n, m], V[m, d_v])
  Y[n, d_Y] = einsum(O[n, d_v], P_o[d_Y, d_v])
  return Y[n, d_Y]
```

谁能想象到，一篇如此强大的论文，里边居然没有一条数学公式，取而代之的全是伪代码！！其实伪代码都算不上，感觉更像是直接把实验中的Python代码复制到了论文中，还是复制到论文主体上！笔者印象里，只有那些不入流的水论文才会这样做，所以笔者看到的第一想法就是水文一篇。也就Google的大佬们才能这么任性，要不是耐着心多扫了几眼，要不是不小心扫到了T5等字眼，要不是回去看作者居然清一色是Google的，这篇强大的论文就被笔者当作垃圾论文放到回收站了 不过，任性还是有任性的代价的，这篇实验阵容这么强大又这么有效的论文，发布至今也有一个多月了，但似乎也没什么反响，估计也跟这个任性的风格有关系～

来自文末的小结

本文介绍了两个关于Multi-Head Attention的后续改进工作，虽然改进细节不一致，但可以说它们都是针对“低秩瓶颈”这个问题而提出的，有种殊途同归之感。两个工作都来自Google，实验内容都很丰富，所以结果都比较有说服力，正在做模型结构改进工作的读者可以参考参考。

arxiv访问慢的小伙伴也可以在订阅号后台回复关键词【**0415**】下载论文PDF。

可能喜欢

- ACL2020 | FastBERT：放飞BERT的推理速度
- ACL2020 | 对话数据集Mutual：论对话逻辑，BERT还差的很远
- LayerNorm是Transformer的最优解吗?
- 如何优雅地编码文本中的位置信息？三种positioanl encoding方法简述
- 知乎搜索框背后的Query理解和语义召回技术
- 详解ERNIE-Baidu进化史及应用场景
- 深入解析GBDT二分类算法（附代码实现）