飞跃芝麻街：XLNet 详解

AINLP 昨天

以下文章来源于安迪的写作间 ，作者准备上天的

安迪的写作间
就是想写点东西。 深度学习，自然语言处理研究生。 喜欢有趣的东西。

“BERT 被碾压了！”

看到这个消息的瞬间，不由自主地在座位上直接喊了出来，大家纷纷转过头来询问，之后分享出论文链接，突然整个组就没了声音，我知道大家都在和我一样，认真读着 **XLNet** 的论文，都想参与这或许会见证历史的时刻，知道到底发生了什么。

现在，距 XLNet 发布已过去一周有余，因为其中**用到多个技巧，涉及很多细节**，所以反复阅读加上和朋友讨论后，才大概将各部分给搞明白，还自己思考了一些东西。

为了应题，可以勉强将 XLNet 想象成是一架组装飞机，为了飞跃芝麻街，用各种部件组装起来，加上足够多的燃料（更多数据），飞了过去。

当然这架飞机的造价也不菲，据 Reddit 热心网友计算，大概也就 **24 万美金**（512x2.5x24x8=245760），相比起来 BERT 的训练感觉只是洒洒水。

对于 XLNet，如果跳过其各种实现细节，我认为它显示出的最重要两点如下：

1. BERT 虽然用了深层双向信息，但没有对**被遮掩（Mask）的 token 之间的关系**进行直接学习，因此 XLNet 通过提出 **Permutation Language Model** （PLM）对其进行了学习。
2. **更多的数据**，还有**用 Transformer-XL 中的技巧带来的更大范围上下文**，对模型有正向加强。

关于 XLNet 各个部件的关系，可以分为如下，为了更好实现 PLM，需要 **Two-Stream Self-Attention** 和 **Partial Prediction**，为了更大的上下文信息，需要 Transformer-XL 中的两个技巧 **Segment Recurrence Mechanism** 还有 **Relative Positional Encoding**，最后为满足 XLNet 像 BERT 一样处理多段句子，加入了 **Relative Segment Encoding**。

接下来就让我来一一介绍这几个部件吧，最后再给出自己的一些看法。

动力系统核心：名为 PLM 的发动机

首先是整篇论文的核心思想，Permutation Language Model。

进入正题前，先来谈谈文中 **AR （AutoRegression， 自回归）** 和 **AE （AutoEncoder， 自编码器）**的提法。这也是很多人觉得很有意思的一个提法。

作者们认为，当前预训练最主要的两个目标可分成两类，一类便是**类似 GPT 的 AR 方式**，根据前面所有信息预测后一个 token，不断重复（自回归），本质上是在进行某种 Density Estimation （密度估计）；而另一类，则是**类 BERT 的 AE 方式**，做法是类似 DAE （Denoising AutoEncoder, 去噪自编码器）中把输入破坏掉一部分，然后还原，BERT 具体做法就是**随机将一些 token 替换成 “[MASK]” 特殊符**。

这种提法是很让人耳目一新，但在我看来可能并没有那么重要，两者界限并不是很明显，特别是后者 DAE 在我看来也能看作一种 Density Estimation，而且微软 UniLM 中的单向语言模型任务，其实已经将两者都包含进去了，它在替换掉最后一个 token 为 "[MASK]" 的同时进行自回归，但也没有显示两个目标结合会更好。

因此我更愿意将 BERT 中的 AE 方式更为特指化，叫做**内部损坏的自编码器（Inner Corrupted AutoEncoder，ICAE）**，因为这里我们想要的其实是**通过破坏掉中间部分然后复原，捕捉深层次的双向信息**。

所以，我想作者更想指出的是，AR 方式所带来的自回归性学习了预测 token 之间的依赖，这是 BERT 所没有的；而 BERT 的 ICAE 带来的对深层次双向信息的学习，却又是像 GPT 还有 UniLM 单向语言模型所没有的，不管是有没有替换 “[MASK]” .

于是，自然就会想，**如何将两者的优点统一起来**？于是乎，就到了主角登场的时间。

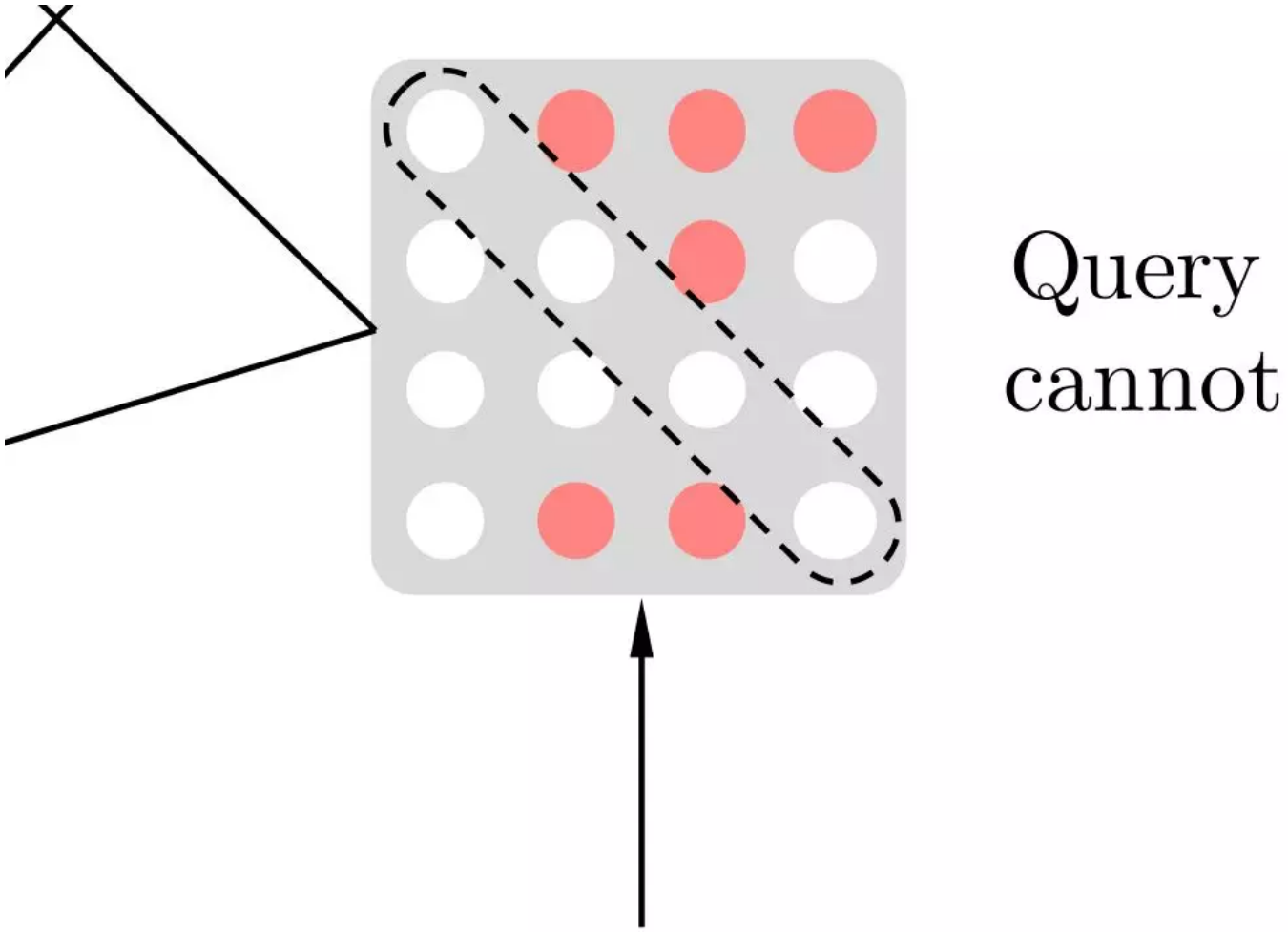
Permutation Language Model.

作者们发现，只要在 AR 以及 AE 方式之间再加入一个步骤，就能够完美地将两者统一起来，那就是**Permutation**.

具体实现方式是，**通过随机取一句话排列的一种，然后将末尾一定量的词给“遮掩”（和 BERT 里的直接替换 “[MASK]” 有些不同）掉，最后用 AR 的方式来按照这种排列方式依此预测被“遮掩”掉的词。**

相信聪明的同学已经发现通过随机取排列（Permutation）中的一种，就能非常巧妙地通过 **AR 的单向方式来习得双向信息**了。

论文中 Permutation 具体的实现方式是通过直接对 Transformer 的 **Attention Mask** 进行操作（对 Transformer 和 Attention Mask 不了解的可以查看[1],[2]）。



比如说序号依次为 1234 的句子，先随机取一种排列，3241。于是根据这个排列我们就做出类似上图的 Attention Mask，先看第1行，因为在新的排列方式中 1 在最后一个，根据从左到右 AR 方式，1 就能看到 234 全部，于是第一行的 234 位置是红色的（没有遮盖掉，会用到），以此类推，第2行，因为 2 在新排列是第二个，只能看到 3 于是 3 位置是红色，第 3 行，因为 3 在第一个，看不到其他位置，所以全部遮盖掉...

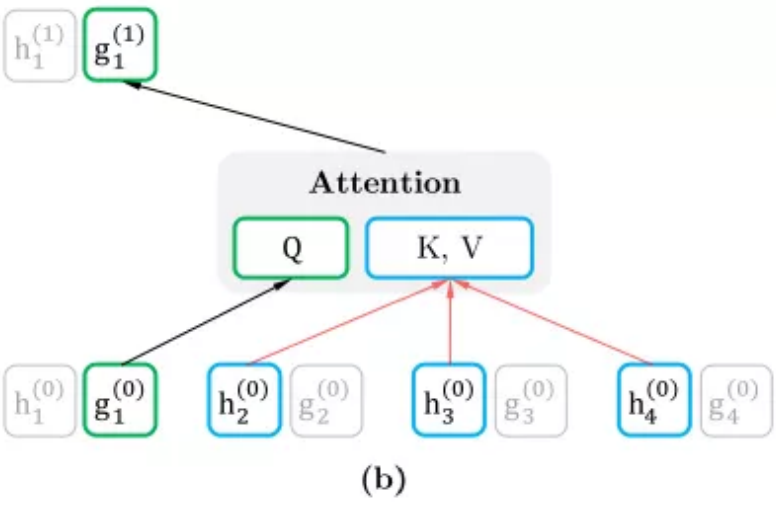
这就是这篇论文的核心思想，看到这里其实已经能去和小伙伴们吹了，接下来会介绍，**XLNet 对 PLM 理念的实现**，文末会列出一种我认为可能的另一种实现方式。

辅助动力系统1：Two-Stream Self-Attention

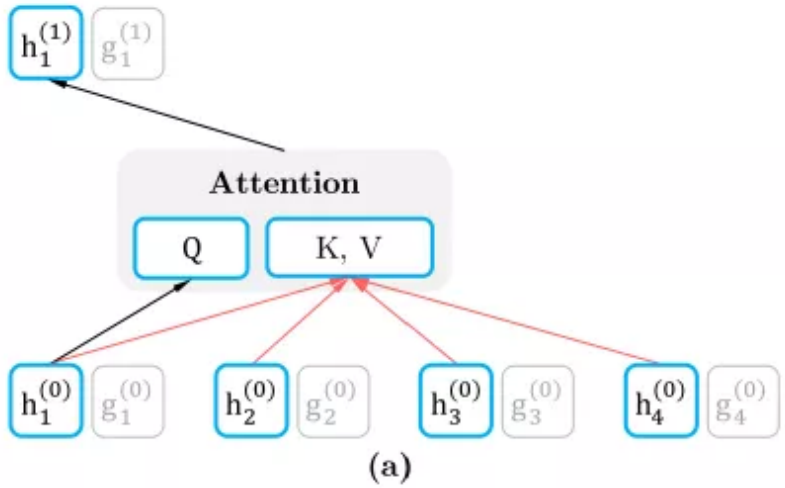
为了实现 Permutation 加上 AR 预测过程，首先我们会发现，打乱顺序后**位置信息非常重要**，同时对每个位置来说，需要预测的是内容信息（对应位置的词），于是**输入就不能包含内容信息**，不然模型学不到东西，只需要直接从输入 copy 到输出就好了。

于是这里就造成了位置信息与内容信息的割裂，因此在 BERT 这样的位置信息+内容信息输入 Self-Attention (自注意力) 的流（Stream）之外，作者们还增加了另一个**只有位置信息作为 Self-Attention 中 query 输入的流**。文中将前者称为 **Content Stream**，而后者称为 **Query Stream**。

这样子就能利用 Query Stream 在对需要预测位置进行预测的同时，又不会泄露当前位置的内容信息。具体操作就是用两组隐状态（hidden states），g 和 h，其中 g 只有位置信息，作为 Self-Attention 里的 Q，h 包含内容信息，则作为 K 和 V.



假如说，模型只有一层的话，其实这样只有 Query Stream 就已经够了。但如果将层数加上去的话，为了取得更高层的 h，于是就需要 Content Stream 了。h 同时作为 Q K V。



于是组合起来就是这样。

这篇论文一个缺点就是内容真的很多，很多地方讲解不是很详细，得看几遍，然后读引用文献，才容易搞明白。比如说这幅图中的两点：

第一点，最下面一层蓝色的 Content Stream 的输入是 e(x)，这个很好懂就是 x 对应的词向量(Embedding)，不同词对应不同向量，但看旁边绿色的 Query Stream，就会觉得很奇怪，为什么都是一样的 w？这个和后面的Relative Positional Encoding 有关，之后细说。

第二点，当然这是实现细节了，（b）图中为了便于说明，只将当前位置之外的 h 作为 K 和 V，但实际上实现中应该是所有时序上的 h 都作为 K 和 V，最后再交给 （c）图中的 Query stream 的 Attention Mask 来完成位置的遮盖。

辅助动力系统2：Partial Prediction

接着是动力系统最后一部分，XLNet 对 PLM 实现的一个细节，**Partial Prediction** （部分预测），非常好理解。

因为当我们按上面提到的实现，在 Permutation 后对每个位置进行预测的话，会导致优化过难，**训练难以收敛**，于是作者们就做了和 BERT 中类似的操作。训练时，**只对每句话部分位置进行预测**。

这些预测位置如何选取呢，选当前排列的最后几个位置。举个例子，假如有 1234567，先随机挑一个排列，5427163，那么假设对最后两个位置预测，于是就需要依此对6和3进行预测。通过挑结尾的位置，在 AR 中，就能在预测时用到尽可能多的可知信息。

这里再谈一个有意思的点，挑选最后几个，那么**到底该挑选几个**呢，总得给个标准吧。于是作者这里设了一个超参数 K，**K 等于总长度除以需要预测的个数**。拿上面的例子，总长为 7 而需要预测为 2，于是 K = 7/2.

而论文中实验得出的最佳 K 值介于 6 和 7 （更好）之间，其实如果我们取 K 的倒数，然后转为百分比，就会发现**最佳的比值介于 14.3% 到 16.7% 之间**，还记得 BERT 论文的同学肯定就会开始觉得眼熟了。因为 BERT 里**将 Token 遮掩成 “[MASK]” 的百分比就是 15%**，正好介于它们之间，我想这并不只是偶然，肯定有更深层的联系。

还有一点需要格外指出，被预测之前的其实取不取 Permutation 都没关系，因为本身位置信息也都在里面，permutation 反而有些更难理解。

上面就是论文的主要部分，下面是一些更细节实现，比如如何从 **Transformer-XL** 借来各种部件。

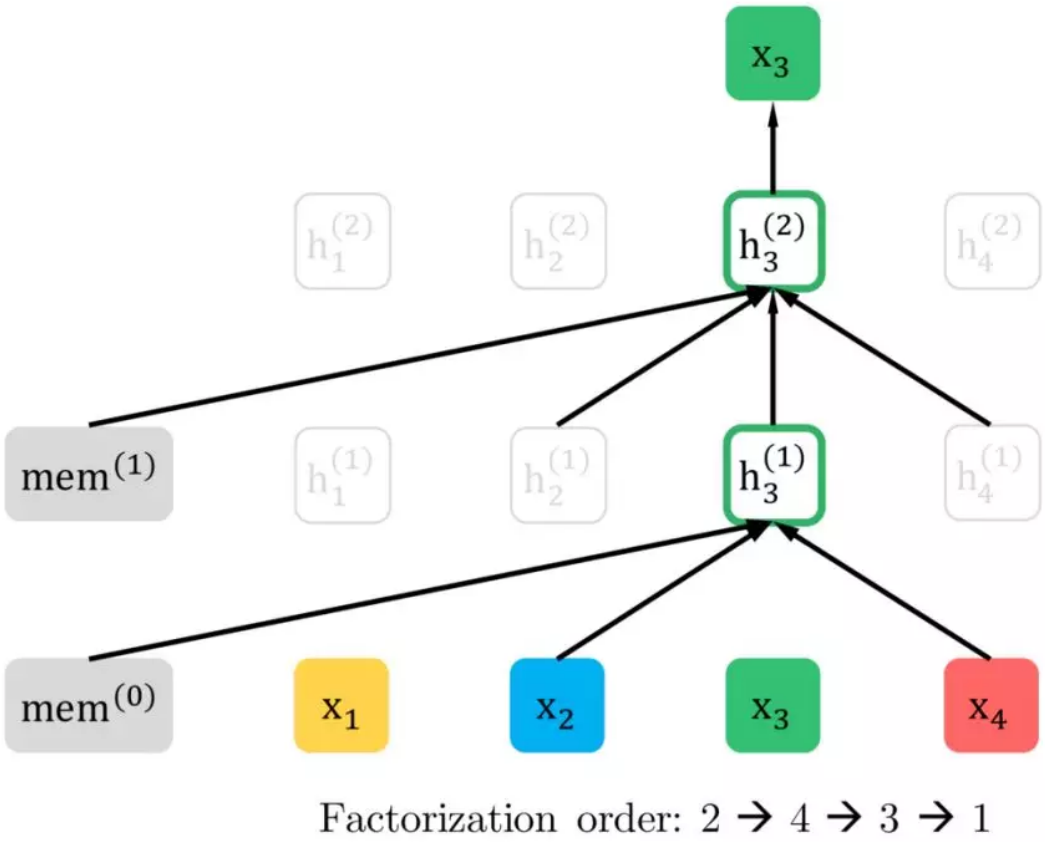
机身：Segment Recurrence Mechanism

Transformer-XL 的重要组件之一，**Segment Recurrence Mechanism**（段循环机制）。

其实思想很简单，因为一般训练 Transformer 时，会按照一定长度，将文本处理成一段（segment）一段的。比如说 BERT 预处理时，就会先处理成一个个 512 长度的样本，即使可能处理前的文本更长。这样子的话，有些更长的上下文信息，模型就是学习不到的。

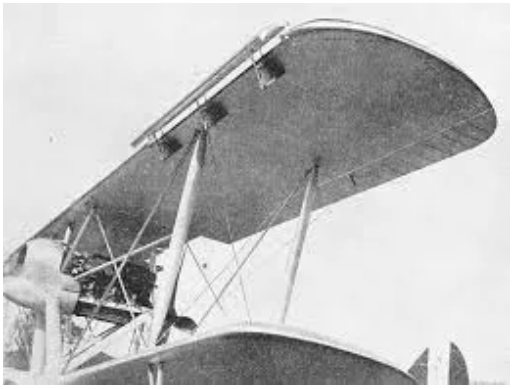
于是 Segment Recurrence Mechanism 想做的就是，能不能在前一段计算完后，将它计算出的隐状态（hidden states）都保存下来，放入一个 Memory 中去，之后在当前分段计算时，**将之前存下来的隐状态和当前段的隐状态拼起来作为 Attention 机制的 K 和 V**，从而**获得更长的上下文信息**。

于是乎文中 Fig1 图中



最左边这个一开始看很是神秘的 mem 的身份也就很明显了，就是 Segment Recurrence Mechanism 中用到的 memory，存放着之前 segment 的隐状态。

机翼1：Relative Positional Encoding



Transformer-XL 的另一重要组件，**Relative Positional Encoding**（相对位置编码），其实很大程度上是为了解决上一个机制中位置信息表示的问题。

这个问题是，假设在 segment1 中已经用了从 1 开始编码的绝对位置向量，那么在 segment2 中，我们**该用什么样的位置编码**呢。

从 1 开始的绝对位置编码吗？这样的话，在复用 segment1 时，整个过程中就会有两个 1 位置，这样是会出问题的，因为模型会搞不清想让它学习的位置信息。

当然也有个做法就是从 segment1 长度 +1 开始给 segment2 加上位置编码，但这样会让位置编码表过长，而且不一定能充分学习，还有就是这样不太符合人类写作的常识，我们其实都是一段段写，不会有人会认真数我现在写到了第 1000 个字，然后第 1000 个字会和第 10 个字有什么关系，更多会关心在某一段中一个字词和其他字词的相对关系。

因此就可以用上这里提到的，**相对位置编码**，不再关心句中词的绝对信息，而是相对的，比如说两个词之间隔了多少个词这样的相对信息。Transformer-XL 中提出的相对位置编码，虽然是为了解决上面的问题，但也非常有趣，将位置信息编码分析得很透彻。

可以简单介绍一下，如何从绝对位置信息编码到相对位置信息编码的。首先，简单定义一下，**E 是词向量**，也可以把它当作主要内容承载者，**U 是绝对位置向量**，可看作绝对位置信息承载者，W 主要是用于 attention 机制 QK 的转换，于是绝对位置信息编码的注意力由下式得出：

乍一看，WTF，这什么鬼，其实只是简单的矩阵运算，用上些定律，简单点可当成类似下面的乘法运算：

具体点的话就是这样：

这四个项也都各有各的意义，（a）表示纯基于内容之间的寻址，（b）和（c）则分别是 i 位置的内容和位置信息分别相对于 j 位置的位置和内容信息进行的寻址，（d）则是纯基于位置之间的寻址。于是要改进的话，就需要对后三个和位置信息相关的项进行改进。

Transformer-XL 给出的改进方案是这样：

主要有三条改进：

- 先把有位置信息 U_j 的地方都替换成相对位置信息 R_ij;
- 之后将(c)和(d)里的 U_i W_q 分别替换成，u 和 v 可学习向量；
- 最后将 K 转换中的矩阵 W_k，分成两个 W_kE 和 W_kR，分别给内容向量和相对位置向量用。

这样就获得了文中的相对位置编码方法。

那么相对位置编码是不是只有这一种，并不是，这只是一种实现方式，比如现在我们就能想出另一种实现方式：

如果再借鉴一下这里的第三条改进，就可以变成，

很自由的。

机翼2：Relative Segment Encodings

为了通过输入形式 **[A, SEP, B, SEP, CLS]** 来处理句子对任务，于是需要加入标识 A 句和 B 句的段信息。BERT 里面很简单，直接准备两个向量，一个加到 A 句上，一个加到 B 句上。

但当这个遇上 Segment Recurrence Mechanism 时，和位置向量一样，也出问题了。万一出现了明明不是一句，但是相同了怎么办，于是我们就需要最后一块补丁，同样准备两个向量，**s+** 和 **s-** 分别表示在一句话内和不在一句话内。

具体实现是在计算 attention 的时候加入一项：

$$s_{ij} = \begin{cases} s_+ \text{ if } i, j \text{ in same segment} \\ s_- \text{ if } i, j \textbf{ not } \text{in same segment} \end{cases}$$
$$a_{ij} = (q_i + b)^T s_{ij}$$

当 i 和 j 位置在同一段里就用 s+，反之用 s-，在 attention 计算权重的时候加入额外项。

燃料：更多的数据

XLNet 这架飞机硬件部分都造好了，于是就只差最后一件东西，也就是，Data（数据）.

XLNet 到底用了多少数据呢?

- BooksCorpus + English Wikipedia （13GB）
- Giga5 （16GB）
- ClueWeb 2012B （19GB）
- Common Crawl （78GB）

加起来有 13+16+19+78=**126GB 纯文本数据**，一个恐怖的数据量。

而 BERT 训练时只用到了第一项，也就是 13GB 的数据。因此就数据而言，XLNet 就用了将近 BERT 十倍的数据。

所以也难怪会有人说 **XLNet 不过是更多数据+更广的上下文信息**。

最后关于训练，值得一说的是，和 BERT 一样也是同时构建正例（正确的连续句子）和负例（随机下一句的例子），之后分别对每段进行 Permutation 处理，然后预测，对于正例，后一段会用前一段的信息，而对于负例就不用。

关于训练 loss，XLNet 只用了 PLM 的 loss，却没有像 BERT 一样用 Next Sentence Prediction （下句预测）loss，但是它在句子级别任务表现却不差，对于这个现象感觉非常神奇，按理说应该是会有帮助的。

飞跃芝麻街！！！

哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒哒......

XLNet 号起飞了！它能否飞跃芝麻街呢！让我们拭目以待！

首先是第一关，RACE 数据集，这是一个难度非常高的阅读理解数据集，以其长度著称，诸多模型都没能在它上面占到丝毫便宜，只见 XLNet 一个桶滚动作，轻松地越了过去，成绩还比前辈们好上很多。这充分说明了 XLNet 对**长片段信息的捕捉能力**。

于是，我们来到了 SQuAD 阅读理解数据集，这是由 SQuAD1.1 和更具难度的 SQuAD2.0 组成的双重关卡。XLNet 在 RACE 上就已经取得很好的成绩了，相信它在更简单的 SQuAD 上也不成问题，果然， XLNet 一个平螺旋，刷刷，连续绕过了 SQuAD1.1 和 SQuAD2.0，在后者上甚至直接甩了 BERT 近7个点！！！

接下来，开始进入一系列文本分类障碍，它们有 IMDB, Yelp-2, Yelp-5, DBpedia, AG, Amazon-2, and Amazon-5。很多名字一看起来就像炮灰， XLNet 一系列动作下来，迅速越过，最后还轻松地做了一个殷麦曼（别问我这是什么，我也不懂）动作。可以看出 XLNet **在句子表征方面性能也很棒**。

之后进入一个特殊关卡，ClueWeb09-B 文档排序数据集，主要用于测试模型生成的词向量效果。只见 XLNet 上拉，下落，一个完美的钟形机动，从上方绕了过去。这证明了光从模型获得基于语境的词向量，XLNet 也是要优于 BERT 的。

最后，终于到了最关键环节，GLUE 数据集，为了在自然语言理解上全面测试模型性能，这是一个由九个自然语言理解任务组成的数据集，号称“死亡九连环”。XLNet 没有丝毫犹豫，一头冲了进去，旋转，跳跃，翻滚，我闭着眼，带着几处轻微刮蹭成功冲了出来！正当大家心潮跃起准备欢呼，只见它机头猛然抬起，突然失速，大家以为它要就此翻车，只见它却又逐渐恢复水平，原来是眼镜蛇机动！悬着的心，高高跃起，芝麻街传来片片欢呼声。

看看成绩，发现除了一两个任务，XLNet 都是 SOTA（最好成绩），而相对于 BERT 则是全面碾压了。

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
Single-task single models on dev									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	**89.8/-**	**93.9**	**91.8**	**83.8**	**95.6**	**89.2**	**63.6**	**91.8**	-
Single-task single models on test									
BERT [10]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
Multi-task ensembles on test (from leaderboard as of June 19, 2019)									
Snorkel* [29]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8
MT-DNN* [18]	87.9/87.4	96.0	89.9	**86.3**	96.5	92.7	68.4	91.1	89.0
XLNet*	**90.2/89.7**†	**98.6**†	90.3†	**86.3**	**96.8**†	**93.0**	67.8	**91.6**	**90.4**

模型比较与其他

与 BERT 的比较

论文最后有消融实验部分，将 XLNet 和 BERT 进行更公平的对比，可以发现 XLNet 中 Transformer-XL 更大的上下文信息贡献了大概一半提高，而 PLM 贡献剩下一半。更多详细分析可以看张俊林博士的文章。

#	Model	RACE	SQuAD2.0 F1	SQuAD2.0 EM	MNLI m/mm	SST-2
1	BERT-Base	64.3	76.30	73.66	84.34/84.65	92.78
2	DAE + Transformer-XL	65.03	79.56	76.80	84.88/84.45	92.60
3	XLNet-Base ($K = 7$)	66.05	**81.33**	**78.46**	**85.84/85.43**	92.66
4	XLNet-Base ($K = 6$)	66.66	80.98	78.18	85.63/85.12	**93.35**
5	- memory	65.55	80.15	77.27	85.32/85.05	92.78
6	- span-based pred	65.95	80.61	77.91	85.49/85.02	93.12
7	- bidirectional data	66.34	80.65	77.87	85.31/84.99	92.66
8	+ next-sent pred	**66.76**	79.83	76.94	85.32/85.09	92.89

其实我更想看到的是 XLNet 这种 Two-Stream 实现方式，和下面提到的 BERT 替换 "[MASK]" 实现方式的对比。因为作者一直有说， BERT 的 pretrain 模式中 “[MASK]” 符号的加入，因为 finetune 时没用到，所以会有不一致产生，从而影响性能，但并没有用实验来证明这一点。

PLM 的另一种实现思路

于是进入 PLM 的另一种实现思路。

当你理解了 PLM，将文中 Permutation，Two-stream 给剥离开，你就会发现，实际上整个 PLM，最主要就是为了**建立预测 token 与 token 之间关系**，使得 XLNet 对于相同目标能够学到更多信息，这是单纯 BERT 直接遮掩然后同时预测所没有的，就如文中提到的 “New York” 的例子。

既然如此，那是不是我们只需要对 BERT 进行一些小小改动就能实现相同的目的，也就是，让它被遮掩的 token 不再是单独一起输出结果，而是依此用类似 AR 的方式输出。

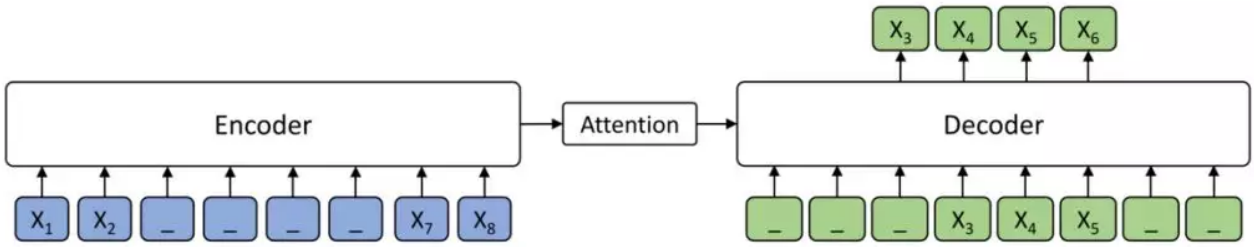
比如说1到7位，随机两个位置，2和6，然后再随机依此预测两个，这个也同样可以用 PLM 中对 attention mask 的操作。而各个位置内容信息泄露的问题，也可以直接用 “[MASK]” 符号避免掉。

其实，XLNet 用到的 Two-Stream 感觉只是为了可以避免用 “[MASK]” 符号，但说到预训练与下游 finetune 任务的不一致，Two-Stream 里也是有的，比如说预训练的时候用的是 g 来预测，而下游任务时却用的是 h。

因此如果不做实验好好比较一下，是很难说 Two-Stream 就要比 BERT 的 “[MASK]” 符号替换更好。

与 MASS 的比较

其实说到 PLM，还有一篇比较类似的预训练论文值得关注，那就是微软的 MASS，它在今年的 WMT2019 取得非常好的成绩。



它的思路是在预训练阶段，对于一句话，我们先遮盖掉，比如说图中的一句话 12345678，我们遮盖掉 3456，然后将被遮盖的 1278 输入 encoder，获得上下文双向信息，之后以此为基础，用 AR 的方式依此预测 3456.

这个方法与 XLNet 的 PLM 做法的不同大体两点：

1. MASS 用的是 BERT 的"[MASK]"遮盖方法；
2. MASS 是整段整段的遮掩，而 XLNet 是类似与 BERT 的随机 1/K 的遮掩。

鉴于 MASS 在生成上比较好的效果，我觉得可以试试将 MASS 的方法拿来和 XLNet 一起训练，或者将 MASS 的方法借鉴来用于 XLNet 的生成式。

还有个很好玩的地方是，MASS 的遮盖比取 50% 的时候是最好的，这个又与 BERT 和 XLNet 里的接近 15% 有些不同了，而造成这个不同的原因，我认为可能是因为连续遮盖需要更长句子来习得生成所需要的依赖信息。

生成式可能性探讨

最后关于 XLNet 生成式方法的探讨，可能大家觉得 XLNet 因为是用 AR 做的所以生成式应该很简单，水到渠成。

但仔细想想就会发现其实会很 tricky，如果这么好做相信早做了，实验结果放论文里了。

将该方法直接用于生成式，首先需要 Permutation，然后之前 finetune 中没用到的 g 就派上用场了，我们按照 permutation 的结果来获得 g，预测结果，然后根据预测结果生成 h, 再生成 g，以此类推。

但这样会带来两个问题

1. 我们如何预先知道生成句子的长度；
2. 预训练时只用了 1/K 长度，这和之后生成时的长度不一致，有很大的 gap。

因此实际上看上去用 AR 做的 XLNet 好像很好做生成式，但其实并没有那么简单。

尾声

XLNet 带领我们飞跃了芝麻街，可以预见之后各大榜上的模型估计也不大会继续取芝麻街里的名字了，我们开始离开被芝麻街统治的时代踏上新的旅途。

XLNet 结构还有很多能改进的地方，当然最重要的是它让大家突破了 BERT 的思想，可能会在 XLNet 的基础上再做出一波突破，非常期待之后的研究。

Reference

1. Illustrated Transformer: https://jalammar.github.io/illustrated-transformer/
2. The Annotated Transformer: http://nlp.seas.harvard.edu/2018/04/03/attention.html
3. Dissecting Transformer-XL：https://mc.ai/dissecting-transformer-xl/
4. XLNet:运行机制及和Bert的异同比较：https://zhuanlan.zhihu.com/p/70257427
5. XLNet 论文：https://arxiv.org/abs/1906.08237
6. Transformer-XL 论文：https://arxiv.org/abs/1901.02860
7. UniLM 论文：https://arxiv.org/abs/1905.03197v1
8. MASS 论文：http://arxiv.org/abs/1905.02450
9. BERT 论文：http://arxiv.org/abs/1810.04805
10. 特技飞行：https://www.glafly.com/Flyzixun/inforshow.aspx?fxzx_id=00031000320003400038

本文转载自公众号：安迪的写作间，作者：Andy

推荐阅读

AI界最危险武器 GPT-2 使用指南：从Finetune到部署

SemBERT: BERT 的语义知识增强

BERT fintune 的艺术

子词技巧：The Tricks of Subword

T5 模型：NLP Text-to-Text 预训练模型超大规模探索

BERT 瘦身之路：Distillation，Quantization，Pruning

Transformer (变形金刚，大雾) 三部曲：RNN 的继承者

关于AINLP

AINLP 是一个有趣有AI的自然语言处理社区，专注于 AI、NLP、机器学习、深度学习、推荐算法等相关技术的分享，主题包括文本摘要、智能问答、聊天机器人、机器翻译、自动生成、知识图谱、预训练

模型、推荐系统、计算广告、招聘信息、求职经验分享等，欢迎关注！加技术交流群请添加AINLP君微信(id：AINLP2)，备注工作/研究方向+加群目的。