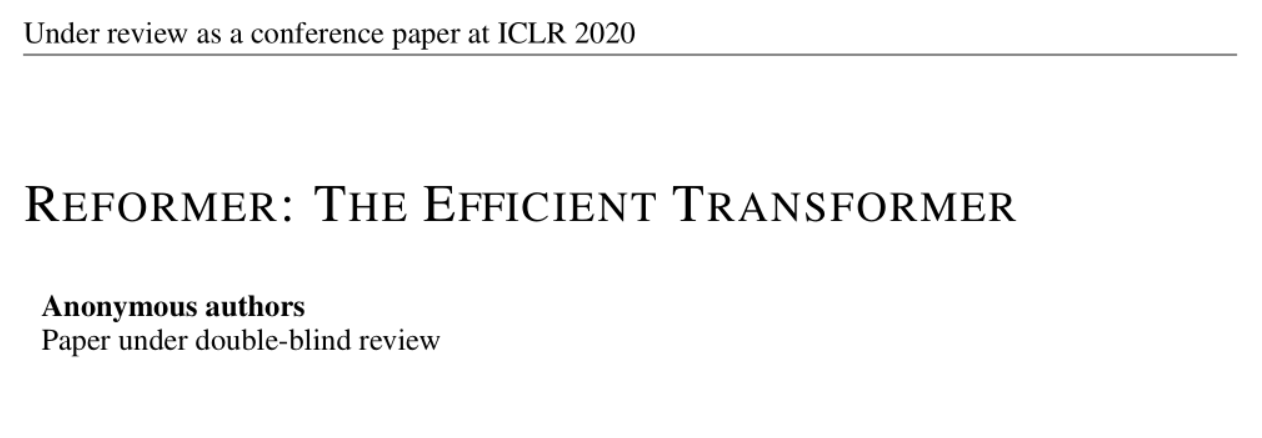
**Reformer: the efficient transformer**



Abstract: transformer effective but trains costly, especially long sequences

Two techniques to improve the efficiency of transformer

1. Locality-sensitive hashing to replace dot-product attention
2. Reversible residual layers to replace standard residual layer

Results: more memory-efficient and faster on long sequences

使得时间复杂度从O（l^2）降到了O（l）

1. introduction:

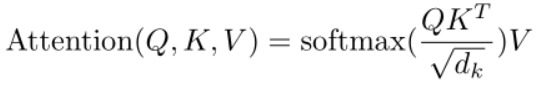
大参数的transformer模型在诸多nlp任务中都取得了较好的结果，但是参数大带来内存消耗以及训练消耗等问题

论文中以transformer-large模型为例，计算其最后一层所需的内存消耗（batch\_size = 8,embedding\_size = 1024, length = 64k）: 64k \* 1k \* 8 = 0.5B floats ~2GB的内存。如果计算N层的内存，因为梯度反向传播时存储的activations，所以内存消耗要比每层内存计算结果的N倍还要多；除此以外，还需要分配前向传播层的内存，以及计算attention时所需的时间复杂度和空间复杂度均为O（l^2）。

论文针对这一问题，在transformer结构中引入了locality-sensitive hashing和reversible-residual layer ，提出了reformer结构。并在实验中证实了这些改变对transformer的产生了微乎其微（negligible）的影响，而且使得模型加快训练过程、减少内存消耗。

**2．LSH attention**

原本transformer模型中attention的计算方式为dot-product attention，计算过程为将所有token的q，k，v都叠加成Q，K，V矩阵形式，shape为[dl, dk or dv]，然后Q，K矩阵相乘，并缩小sqrt(d\_k)，然后与V矩阵相乘，最后进行softmax。



这种attention的计算方式在处理长序列文本输入时需要大量的内存消耗，论文中以64K长的tokens为例，Q，K, V的shape为[batch, length=64k, dmodle]，则计算QK的时候，所需内存为64k\*64k\*8~16GB，并且在后向传播的时候还需要重新计算，因此这种attention计算方式在处理长序列输入时很不高效。

LSH attention：Q=K, Shared-QK transformer，在计算QK时我们只考虑softmax（QK），即对于每一个qi，我们只考虑与qi最接近的ki（乘积最大），例如在64k长度的K中，选取32或者64个与qi最接近的ki。

LSH局部敏感哈希就是实现找到最接近qi的ki的算法，局部敏感哈希算法可以将每个向量 x 转换为 hash (x)，一个hash 函数就是使用一个哈希表，将特定的值映射到不同的桶（bucket）中，和这个 x 靠近的哈希更有可能有着相同的哈希值。

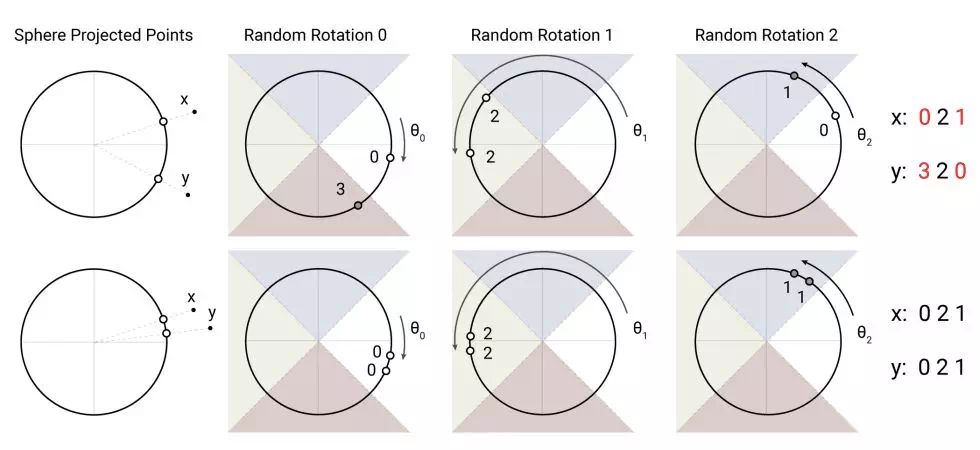
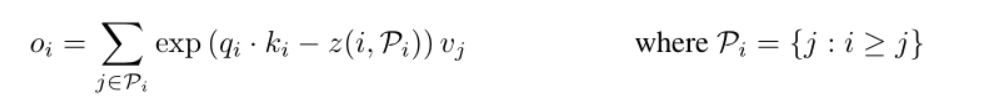


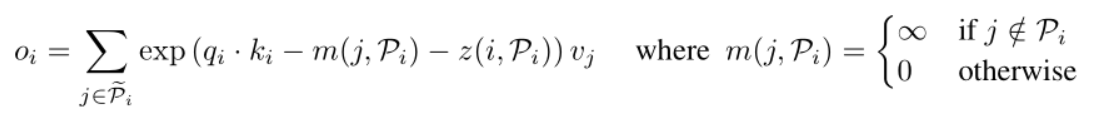
图 1：研究中使用的局部敏感哈希算法。这种算法使用随机旋转的方法，对投影的点建立分块，建立的规则依据对给定轴的投影进行比较。在本图中，两个点 x、y 由于三次随机旋转投影中的两次都不靠近，所以不太可能有相同的哈希值。而另一个例子中他们投影后都在同一个。

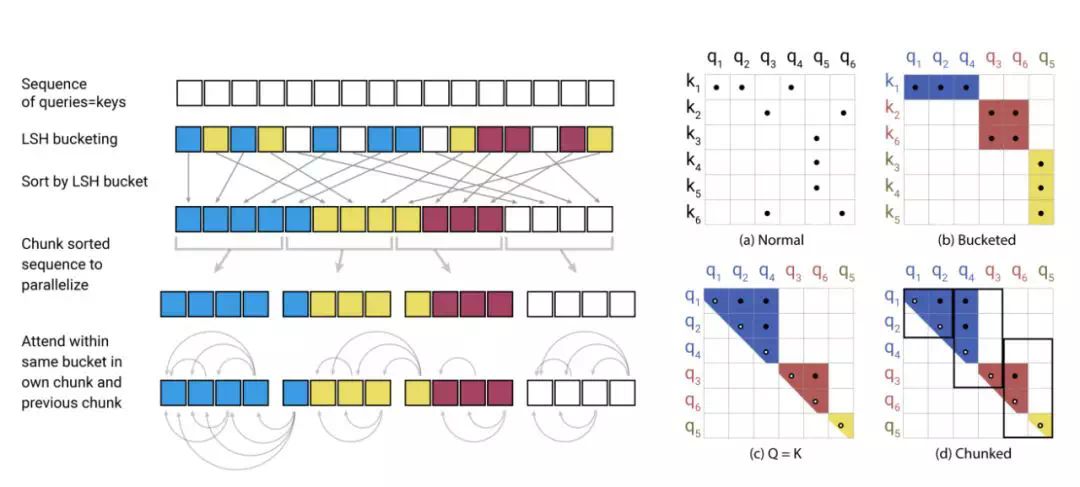
对于每一个qi 的LSH attention计算方式如下：



其中，z表示分区函数，相当于*softmax*中归一化项。与原始attention计算不同的是，LSH省略了的缩放。

考虑decoder的attention时，只考虑位置i之前的K\_j, 那么每一个位置i都有不同的P\_i集合，这样处理起来不够高效。对于i之后位置的k\_j，attention的结果为0，就可以采用q\_i\*k\_i-正无穷，再经过exp后便是0实现，这样p\_i集合就只用{0，1，2，。。。l}就可以了。LSH attention计算方式如下：





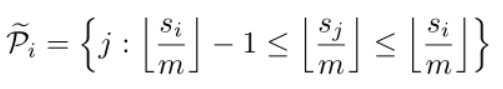
图二总结了LSH attention计算的全部过程。

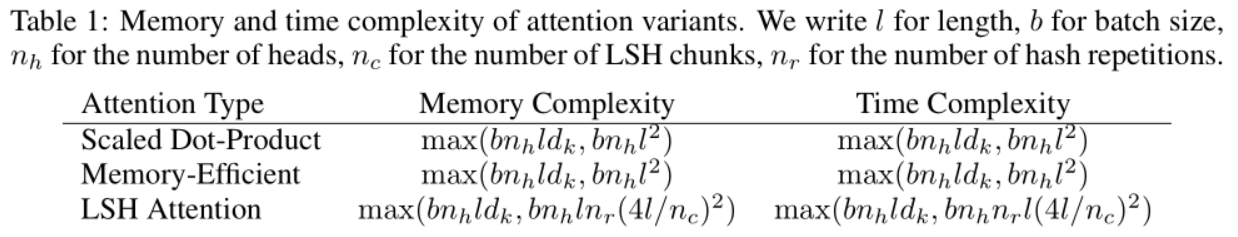
右边图中 （a）：我们可以看到在q和k不同的情况下，即普通的attention机制中，黑点代表的是需要softmax中占主导的位置，注意这边的attention使用的是encoder的attention。可以看到，对于需要计算的位置是稀疏的，可以利用这个降低计算attention的时空复杂度。

（b）：不改变q和k，但是这次使用了LSH，即只计算相同bucket的位置的keys。按照bucket进行排序，然后对于同一个bucket又按照原本的位置进行排序得到图b。可以看到，对于同一个bucket，可能出现一个bucket中有多个query但是很少keys的情况，例如图中蓝色的bucket。

（c）：为了减小bucket中q和k不均衡的问题，论文使用了share-QK attention，即通过设置k\_i= q\_i/|q\_i|，使得h(k\_i) = h(q\_i)。然后在按照bucket 排序，每个bucket中，仍按照原本的position 位置大小排序。得到图c。这时候就能保证对角线都是attend to的，而且q和k在bucket中的个数一样（因为Q=K）。注意到对角线的点为空心，这是因为在share-QK的实现下，如果计算本身的attention，会导致其值特别大，其他的值特别小，经过softmax之后，其他都是0，就自己本身是1。所以为了避免这种情况，q不会去计算与自身位置的k值的attention，除非只有自己本身可以attend to（例如图二c中q3和q5的）。

（d）：即使使用了shared-QK，但是还是会出现一个问题就是，有的bucket中个数多，有的bucket中个数少。比如，对于两个bucket，其中一个bucket占据了所有的keys，另一个bucket为空，那么LSH attention就没有起到作用。于是在c的基础上，增加了chunk的操作。具体的操作就是在对输入进行排序之后（先bucket排序，同个bucket内按照token 的 position排序）得到新的序列顺序s\_i即 i-> s\_i 。论文中将设每个bucket的个数为m=l/(2\*n\_bucket), (其中l为输入query的个数)，然后对于bucket中的每个query，都可以与该bucket以及前一个bucket 中相同哈希值的key计算attention。 其选集可表示为P\_i：





**3.reversible transformer**

除了attention的复杂度之外，transformer还有feed-forward层的内存消耗，可以用b\*l\*d\_ff\*n\_l来表示，d\_ff表示ff层隐藏状态维度，n\_l表示层数。以transformer-large为例，d\_ff = 4k，n\_l= 16，假定序列长度为64k，那么内存消耗约为64k\*4K\*16\*4(float类型)~16GB。

本论文使用reversible layer可逆层来解决 n\_l 问题，然后展示了如何利用分块来解决 d\_ff 问题。

在介绍reversible layer之前，先比较一下residual layer常规的残差层的区别：

Residual layer其函数形式可以表示为：y=x+f(x)。

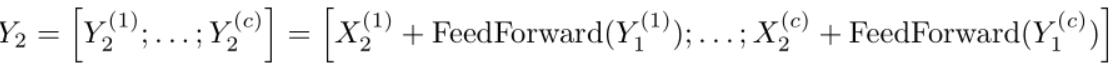
reversible layer则参考了RevNet的思想，reformer将注意力和前馈层结合在RevNet模块中，其函数形式如下所示：



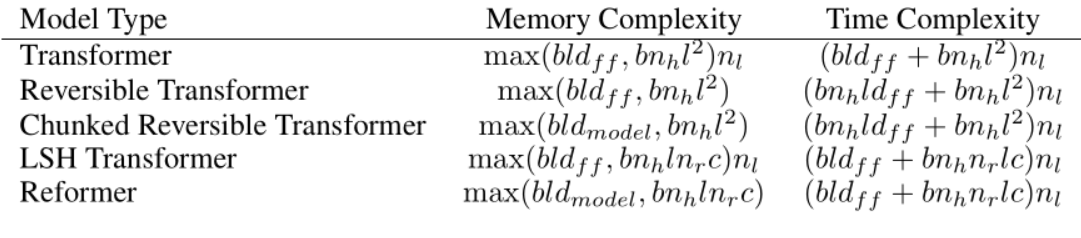
其结果使得reformer不需要在每层存储激活，并且无需nl项。

**4.分块**

尽管reversible transformer消除了n\_l项，但是对于前馈层较多情况下，仍然会使用大量内存，针对这种情况，由于前馈层的计算在序列中时独立的，本论文采用分块的策略：



本论文还对相关工作进行了比较，其结果如下图所示：



Table比较了相关模型的时间和空间复杂度，可以看到：空间复杂度是前馈层和注意力层两者中的较大值，当处理长序列输入时，l通常会很大，reformer通过LSH attention和reversible layer和分块，是得模型的时空复杂度变成L的线性。

**3.实验结果：**

实验结果就不多叙述了，就讲一下下面两张实验结果图：

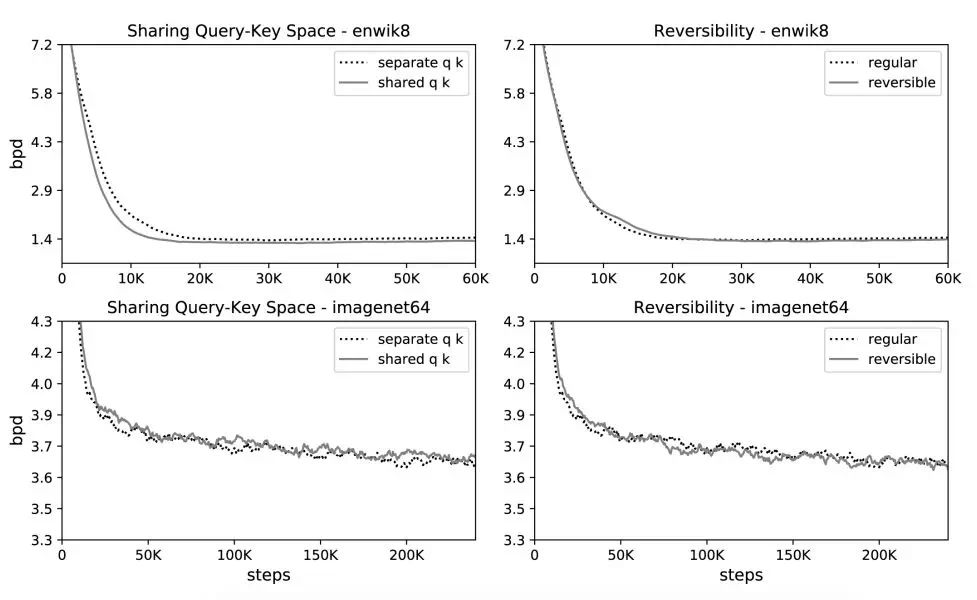


图3在enwik8和imagenet64数据集上对shared QK和reversibility对transformer的性能影响进行了ablation 实验，实验结果表明以上两种改进对transformer性能的影响很小。

图4研究者描述出不同注意力类型的速度和序列长度的变化曲线图，同时保持 token 总数量不变。结果显示，常规注意机制随着序列长度的增加而速度减缓，而 LSH 注意机制速度保持平稳。

