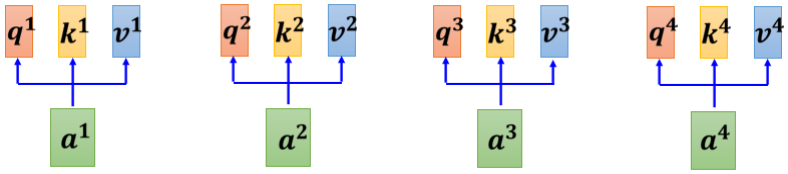
7月19日

一、李宏毅2021春机器学习课程第4.2节：自注意力机制（二）

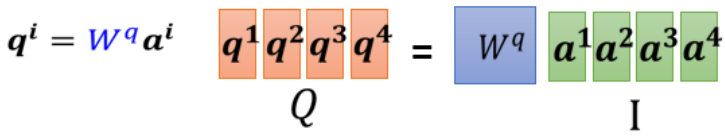
1 从矩阵的角度来理解Self-Attention的运作

接下来我们从矩阵乘法的角度来看一下Self-Attention是如何运作的。

我们现在已经知道每一个 a 都产生一个对应的 q, k, v:

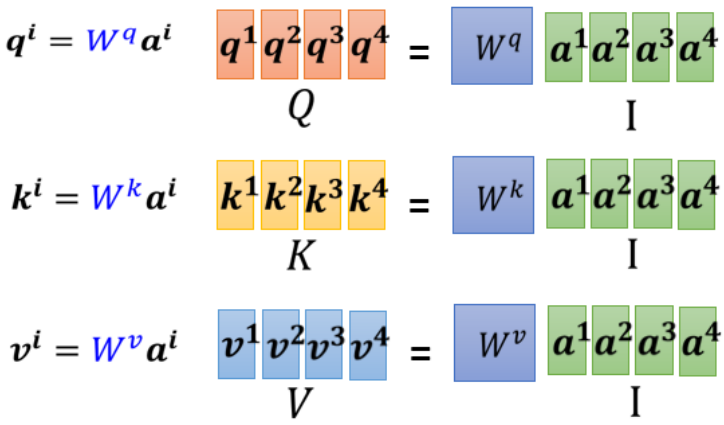


我们每一个 a 都要乘上一个矩阵 W^q 来得到对应的 q^i ，**这些不同的 a 其实合起来，当作一个矩阵来看待**，这个矩阵我们用 I 来表示，这个 I 矩阵的四个 column 就是 a^1 到 a^4。



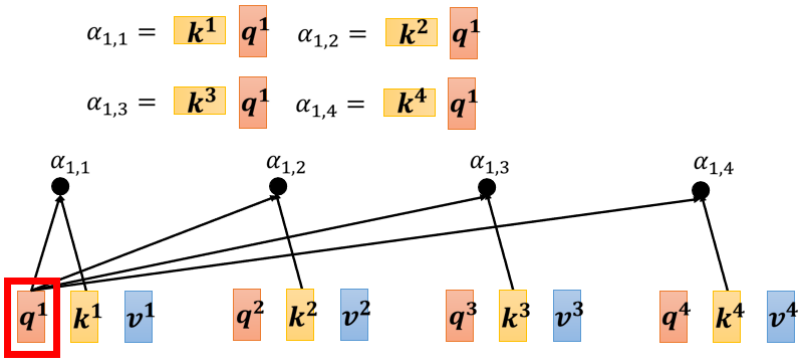
I 乘上 W^q 就得到另外一个矩阵，我们用 Q 来表示它，这个 Q 矩阵的四个 column 就是 q^1 到 q^4 。所以我们之前那个从 a^1 到 a^4，得到 q^1 到 q^4 的操作，看起来好像是分开计算的，但实际上就是**把 I 这个矩阵，乘上矩阵 W^q，得到矩阵Q**。所以说 q^1 到 q^4 其实是并行产生的，而W^q**是 network 的参数，它是会被 learn 出来的。**

接下来產生 k 跟 v 的操作跟 q 是一模一样的

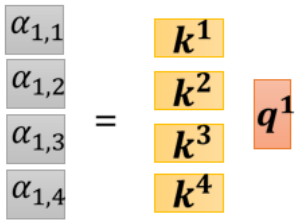


那事实上呢，我们把 I 分别乘上矩阵 W^q, W^k, W^v 就能得到相应的 Q, K, V 矩阵，也就得到了 a^1 到 a^4 分别对应的 q, k, v。

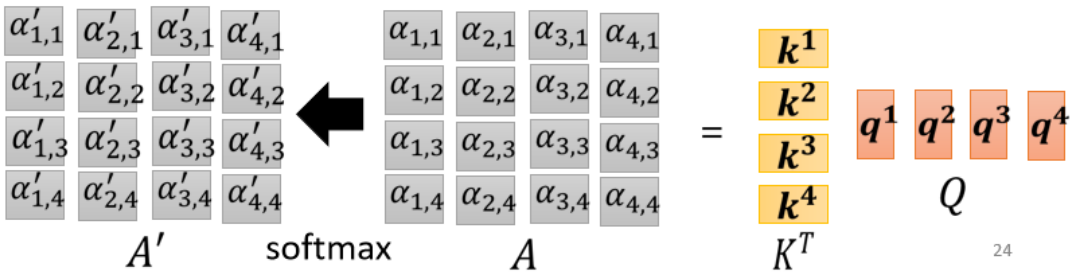
下一步是，每一个 q 都会去跟每一个 k，去计算 inner product，也就是对应位置逐元素相乘后相加，从而得到 attention 的分数，就比如 q^1 跟 k^1 做 inner product 会得到 $\alpha_{1,1}$，q^1 跟 k^2 做 inner product 会得到 $\alpha_{1,2}$，以此类推。



如果我们从矩阵运算的角度来看，这四个步骤的操作同样可以拼起来，我们可以把**把 k^1 到 k^4 拼起来，当作是一个矩阵的四个 row**，然后把整个过程看作是**矩阵跟向量相乘。**

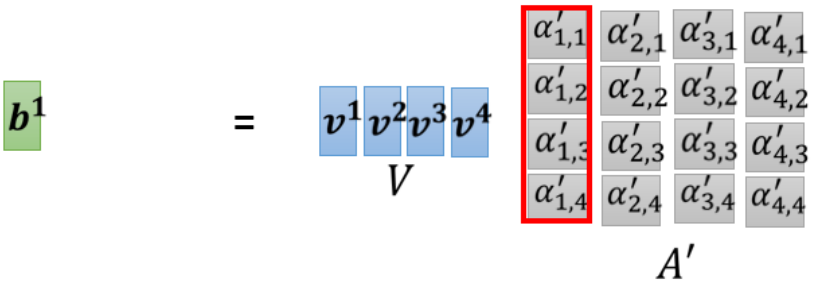


当然我们不只是 q^1 要对 k^1 到 k^4 计算 attention 的分数，q^2, q^3, q^4 也要对 k^1 到 k^4 计算 attention 的分数，所以其实我们也可以把**把 q^1 到 q^4 拼起来，当作是一个矩阵的四个 column**，所以这些**attention 的分数实际上可以看作是两个矩阵的相乘**，一个矩阵它的 row 就是 k^1 到 k^4，另外一个矩阵它的 column 就是 q^1 到 q^4 。

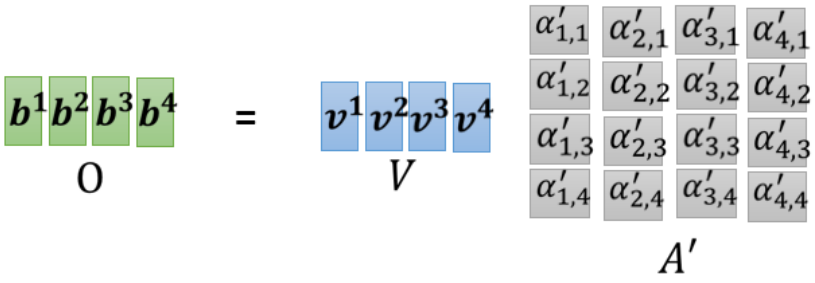


我们会在 attention 的分数输出之前**做一下 normalization**，比如说用 softmax，对 A 矩阵的每一个 column 做 softmax，让每一个 column 里面的值相加是 1，这样我们就得到了新的矩阵 A'。

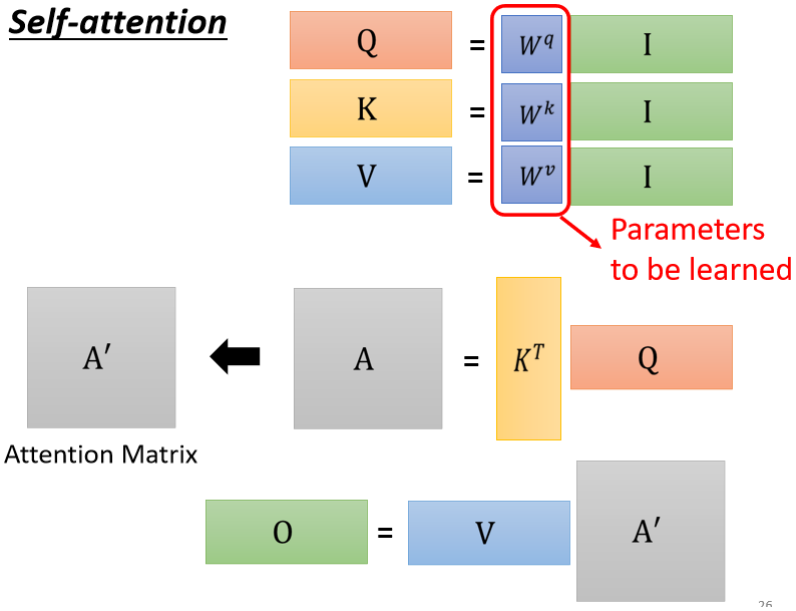
同样的，我们把 v^1 到 v^4 拼起来，当成是 V 这个矩阵的四个 column，然后接下来你把 V 乘上 A' 的第一个 column 以后，得到的结果就是 b^1：



接下来就是以此类推得到剩下的 b^2, b^3, b^4：



所以总的来看我们就是把 A' 这个矩阵，乘上 V 这个矩阵，得到 O 这个矩阵，O 这个矩阵里面的每一个 column就是 Self-attention 的输出，也就是 b^1 到 b^4。所以整个Self-attention的过程其实就是**一连串矩阵的乘法而已。**下图是一个总结：



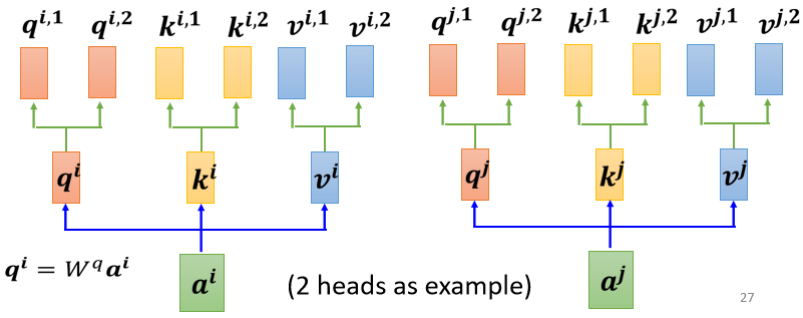
最后你会发现**其实 Self-attention layer 里面，唯一需要学出来的参数就只有 W^q W^k W^v 而已。**

---

2 进阶版本：Multi-head Self-attention

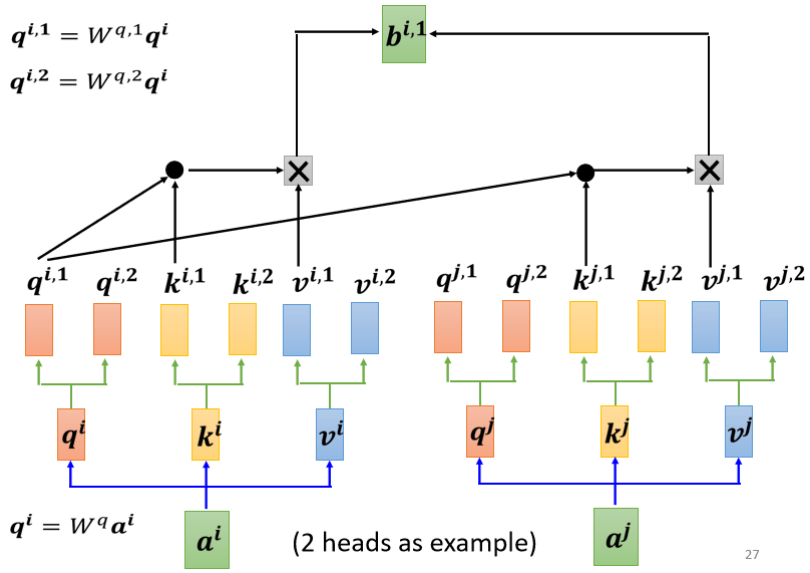
Self-attention 有一个进阶的版本，叫做 Multi-head Self-attention，这个进阶版本今天的使用也是非常广泛的。所谓的多个head，其实就是对于每一个输入的向量，可以有不止一个query。因为我们在做 Self-attention 的时候，就是用 q 去找相关的 k，但是**相关这件事情有很多种不同的形式**，有很多种不同的定义，所以也许我们不能只有一个 q，我们应该要有多个 q，**不同的 q 负责不同种类的相关性。**

至于我们**需要用多少个 head，这个又是另外一个 hyperparameter**，也是需要我们自己根据具体任务进行调整的。所以假设你要做 Multi-head Self-attention 的话，该如何操作呢？这里举一个有两个head的例子：

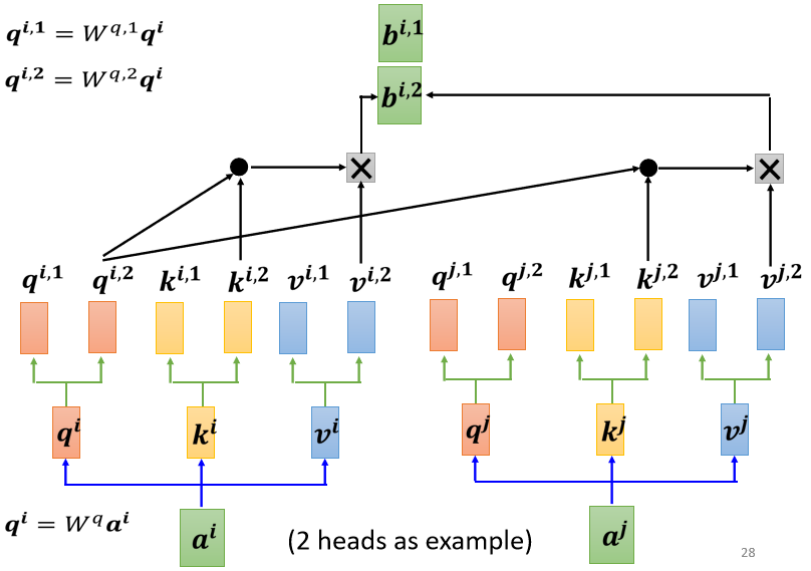


我们认为当前要处理的问题里面有两种不同的相关性，所以我们需要产生两种不同的 head，来分别寻找两种不同的相关性。我们来看其中的两个输入 $a^i a^j$，对于 a^i 来说，产生对应的 q_i, k_i, v_i 的步骤跟之前不是 Multi-head 时的情况一样，也就是 a^i 分别乘上矩阵 W^q, W^k, W^v。但接下来我们还需要将 q_i 乘上另外两个矩阵得到 $q^{i,1}$ 和 $q^{i,2}$，这里的两个上标中，i 代表的是位置，然后这个 1 跟 2 代表的是这个位置的第几个 q。

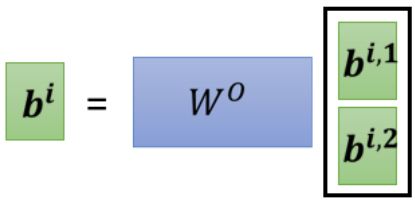
那既然 q 有两个，k 和 v 也要有两个，从 q 得到 $q^1 q^2$，从 k 得到 $k^1 k^2$，从 v 得到 $v^1 v^2$，其实接下来的过程与之前不是 Multi-head 时是完全一样的，只是因为现在我们的在算 Attention 的分数时，我们的 q, k, v 都有两份，所以计算的输出 b 也有两份，在计算 $b^{i,1}$ 时，就只看 $q^{i,1}, k^{i,1}, v^{i,1}$：



而在计算 $b^{i,2}$ 时，就只看 $q^{i,2}, k^{i,2}, v^{i,2}$：



然后接下来我们可能会把 $b^{i,1}$ 跟 $b^{i,2}$ 拼起来，再乘上一个矩阵，得到我们最终的输出 b_i，这就是 Multi-head attention 的整个运作过程。

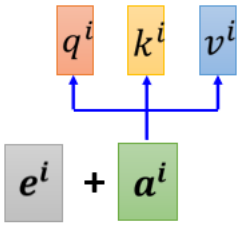


3 通过Positional Encoding考虑输入的位置信息

那到目前位置，其实你会发现对 Self-attention 而言，**位置 1 跟位置 2，3，4完全没有任何差别，这四个位置的操作其实是一模一样**，对它来说 q1 到跟 q4 的距离，并没有特别远，2 跟 3 的距离也没有特别近。

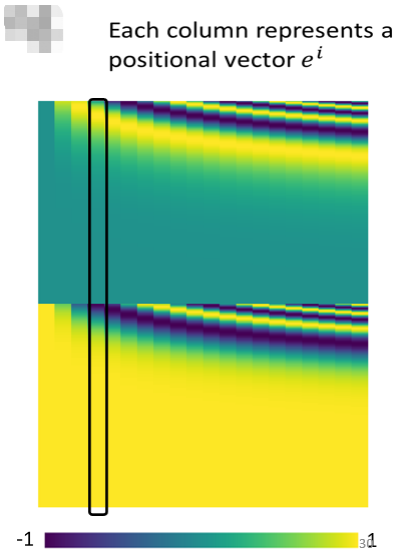
但是这样的设计可能会有一些问题，因为有时候位置的信息也许很重要，举例来说，我们在做 POS tagging，就是词性标记的时候，我们知道**动词比较不容易出现在句首**，所以如果我们知道某一个词汇它是放在句首的，那它是动词的可能性可能就比较低。

那我们要怎样把位置的资讯加到我们的输入中呢，这里就会用到一个叫做 positional encoding 的技术。



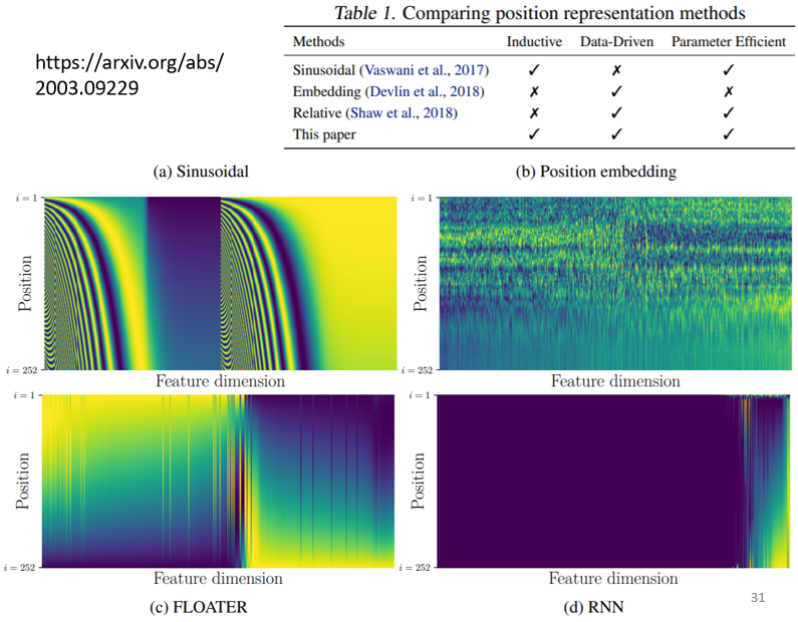
我们为每一个位置设定一个 vector，叫做 positional vector，**用 e^i 来表示，上标 i 代表是位置**，不同的位置都有一个它专属的 e，然后把这个 e 加到 a^i 上面，就结束了。这就是告诉你的 Self-attention 的 Network，如果它看到说 a^i 好像有被加上 e^i，它就知道说现在的输入 a^i 应该是在 i 这个位置出现的。

Attention Is All You Need 那篇 paper 里面，使用的 e^i 是下面这个样子：



这个图上每一个 column 就代表一个 e，第一个位置就是 e^1，第二个位置就是 e^2，第三个位置就是 e^3，以此类推。这样的 positional vector，是人为设定的，但既然是人设的那就会存在一些问题，就比如我在设计这个 positional vector 的时候只设计了128个位置，那对于长度为 129的 sequence 来说就会有问题。**所以我们需要探寻一个规则来产生这个 positional vector，这个规则就是所谓的 positional encoding。**

positional encoding 仍然是一个尚待研究的问题，你可以创造自己新的方法，甚至可以把 positional encoding 作为一个Network的参数，是根据资料学出来的。

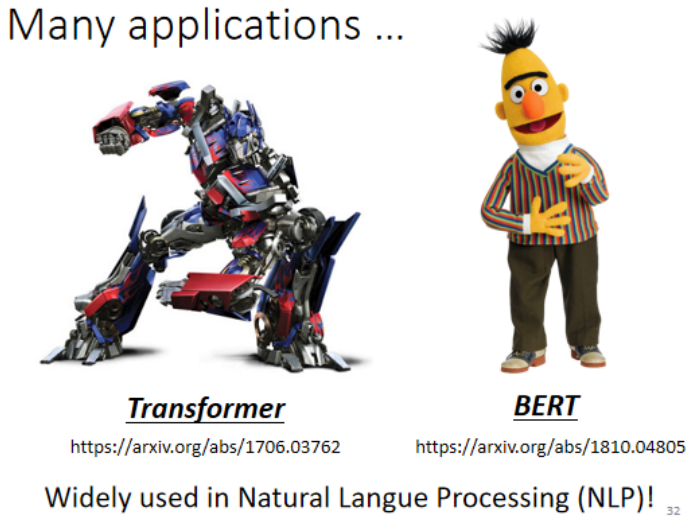


至于这个 positional encoding 的方法有很多，其中一篇可以参考的文献： [Learning to Encode Position for Transformer with Continuous Dynamical Model (arxiv.org)](), 总之这是仍然是一个尚待研究的问题，**你永远可以提出新的做法。**

---

4 Self-attention 的应用

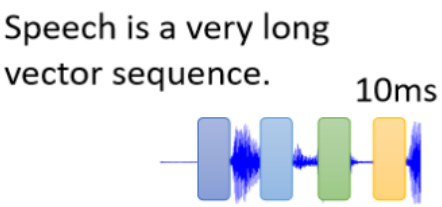
Self-attention 的应用是非常广泛的，我们之前已经提过很多次 transformer 这个东西：



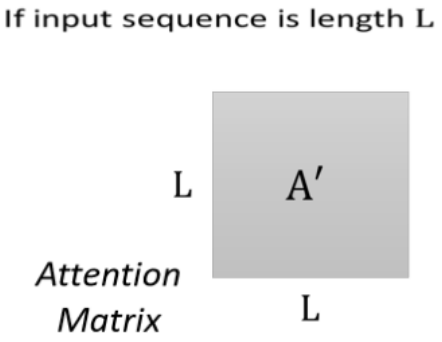
那大家也都知道，在 NLP 的领域有一个东西叫做BERT，这个BERT我们后面会详细介绍，BERT里面也用到了 Self-attention，所以 Self-attention 在 NLP 上面的应用是大家耳熟能详的。但 **Self-attention不是只能用在 NLP 相关的应用上，它还可以用在很多其他的问题上。**

4.1 Self-attention 在语音上的应用

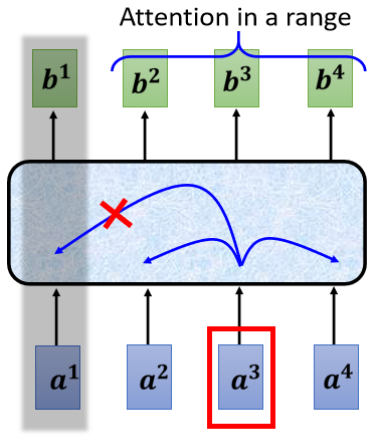
在做语音识别相关的应用时，我们也可以用 Self-attention 来做，不过**要把一段声音讯号表示成一排向量的话，这排向量可能会非常地长。**



在声音讯号的处理过程中，一般我们每一个向量只代表了 10 millisecond 的长度而已，所以如果今天是 1 秒钟的声音讯号，它就有 100 个向量了，随便讲一句话，都是上千个向量了。那表示声音讯号的向量过长会带来什么问题呢？我们其实知道在**计算 attention matrix 的时候，它的计算复杂度是矩阵长度L的平方。**那如果这个L的值很大，它的计算量就很可观，也需要非常大的memory，才能够把这个矩阵存下来。



所以在用Self-attention做语音识别的时候，有一招叫做 Truncated Self-attention。



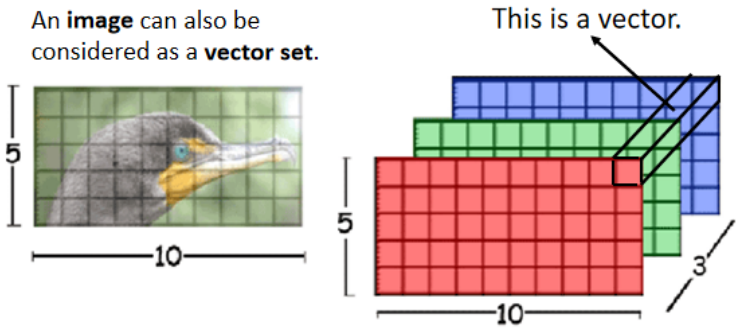
Truncated Self-attention

Truncated Self-attention 就是我们今天在做 Self-attention 的时候，**不要需要看完一整句话，而是只看一个小的范围就好，至于这个范围是多大则是人为设定的。**这样就能够在语音处理这种输入的向量很大的情况下，加快运算的速度。

那你可能要问之前提出 Self-attention 就是因为需要考虑整个Sequence的资讯，这里凭什么就可以只看一个小范围的资讯呢？这也是**与语音识别本身的特性有关的**，也许我们要辨识某个位置的音素是什么，某个位置有怎样的内容，大部分情况下我们并不需要看完整句话，而是只要看目标位置跟它前后一定范围之内的资讯就可以判断。总而言之，**对于网络结构的选取和设计，一直都是具体问题具体分析，绝不是一成不变的。**

4.2 Self-attention 在图像上的应用

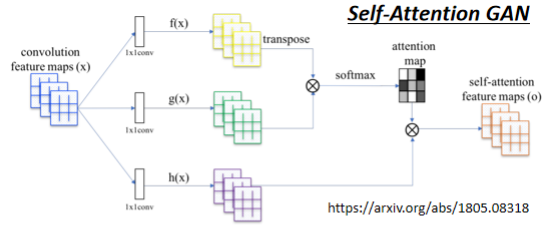
一张图片，在讲到CNN的时候，我们把它看作是一个很长的向量，**那其实一张图片，我们也可以换一个观点，把它看作是一个 vector set。**

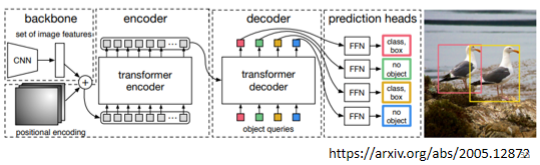


Source of image: https://www.researchgate.net/figure/Color-image-representation-and-RGB-matrix_fig15_282798184

上面这个示例中的图片是一个解析度 5 × 10，channel为3的图片。**你可以把每一个位置的 pixel，看作是一个三维的向量，所以整张图片其实就是 5 × 10个向量的set。**这样一来，我们应该也可以用 Self-attention 来处理图片了。事实上也已经有了很多把 Self-attention 用在影像处理上的工作，这里放两篇Paper的链接供参考：

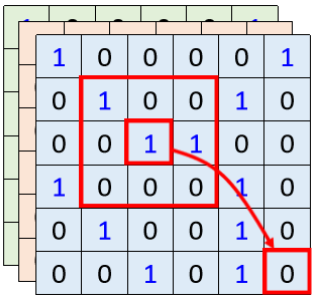
- Self-Attention Generative Adversarial Networks (arxiv.org)
- End-to-End Object Detection with Transformers (arxiv.org)



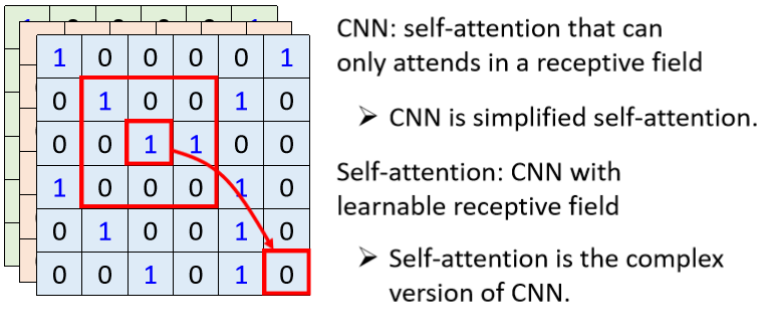


接下来我们来比较一下，Self-attention 跟 CNN 之间有什么样的差异或者是关联性。

如果我们用 Self-attention 来处理一张图片，这就是说假设其中某一个pixel使我们要考虑的，那它自己需要产生 query，其他所有的 pixel 都要产生key，并与其计算一个attention的分数。这也就是说，**我们对每一个像素点的处理都考虑到了整张图片的资讯。**



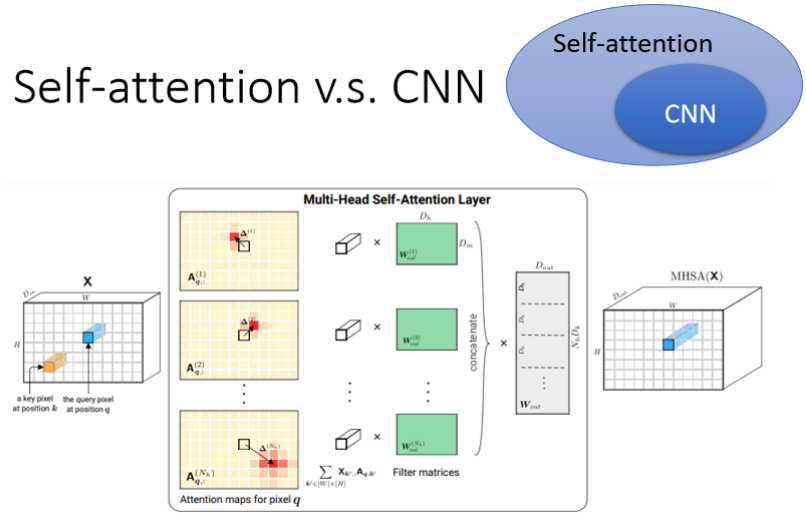
但是我们在做 CNN 的时候，会设定一个 receptive field，**每一个 filter，每一个 neural，都只考虑它自己的 receptive field 范围内的资讯。**



所以事实上，**CNN 可以看作是一种简化版的 Self-attention**，反过来说，**Self-attention 是一个复杂化的 CNN。**

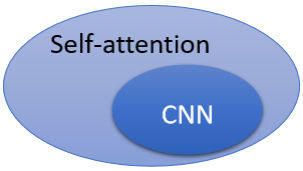
在 CNN 里面，我们要人为划定 receptive field 的大小和范围，而对 Self-attention 而言，我们用 attention 去找出相关的 pixel，就好像是 **receptive field 是自动被学出来的**，network 自己决定说，receptive field 的形状长什么样子，哪些 pixel 是我们真正需要考虑的，**所以在 Self-attention 下 receptive field 的范围不再是人工划定，而是让机器自己学出来。**

On the Relationship between Self-Attention and Convolutional Layers (arxiv.org)，这篇论文的作者在他的文章中就阐明了这个观点，并且用数学的方式严谨的告诉你说，其实**CNN 就是 Self-attention 的特例，Self-attention 只要设定合适的参数，它可以做到跟 CNN 一模一样的事情。**

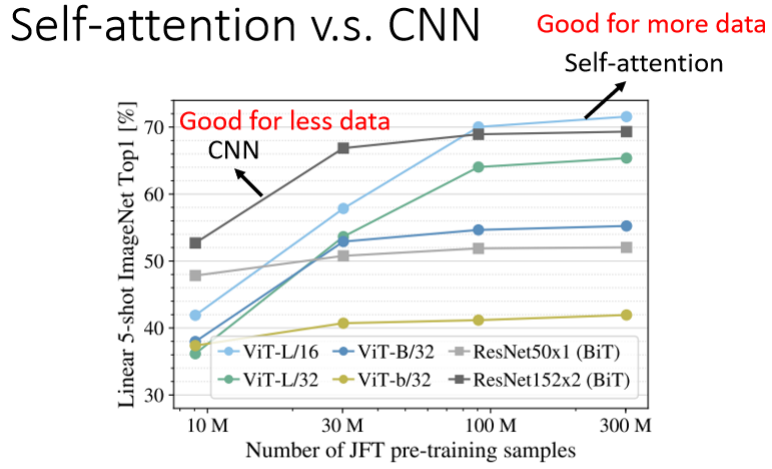


On the Relationship between Self-Attention and Convolutional Layers https://arxiv.org/abs/1911.03584

所以 **Self-attention 是弹性更大的 CNN**，而 **CNN 是受限制的 Self-attention**。那之前就已经说过**弹性比较大 的 model，训练的时候需要更多的 data，并且 overfitting 的风险更大。**



下面这个例子证实了刚刚的说法，下图的实验结果来自这篇文章：[An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arxiv.org)]



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale https://arxiv.org/pdf/2010.11929.pdf

横轴是训练集的数据量，从10M到300M。Self-attention 是浅蓝色的这一条线，而 CNN 是深灰色的这条线。可以看到，**随着资料量越来越多，Self-attention 的结果就越来越好，最终在资料量达到300M 的时候，Self-attention 的表现可以超过 CNN，但在资料量少的时候，CNN 的表现比 Self-attention 好很多。**

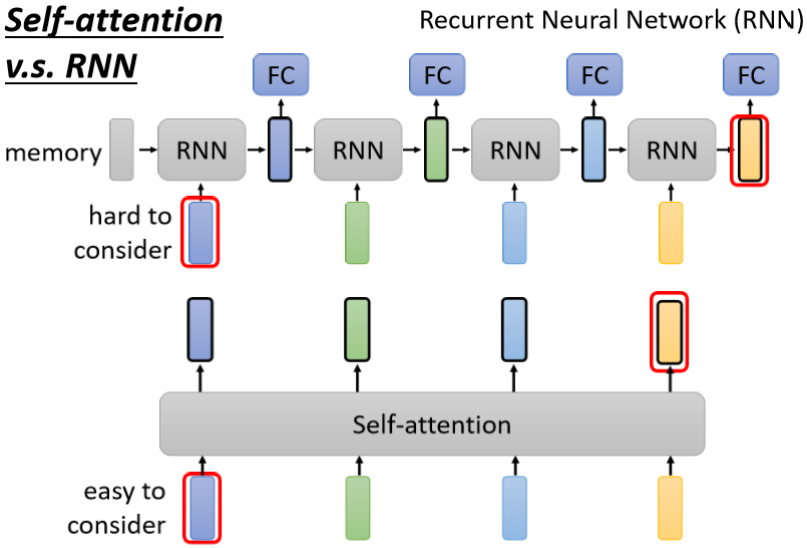
这个结果可以从 CNN 跟 Self-attention 的弹性大小来解释：

- Self-attention 的弹性比较大，所以需要比较多的训练资料，训练资料少的时候容易 overfitting 导致结果不好，而训练资料足够的情况下结果会比较好。
- CNN 弹性比较小，在训练资料少的时候结果比较好，但训练资料多的时候，它没有办法从更大量的训练资料得到好处。

4.3 Self-attention 可以替换 RNN

我们来比较一下 Self-attention 跟 RNN，RNN就是 recurrent neural network（循环神经网络），**目前看来， RNN的角色很大一部分都可以用 Self-attention 来取代了。**

至于 RNN 是什么可以参考这篇文章：[一文搞懂RNN（循环神经网络）基础篇 - 知乎 (zhihu.com)](#)，这里只是简单介绍一下，RNN 跟 Self-attention 一样，都是要处理 input 是一个 sequence 的状况。



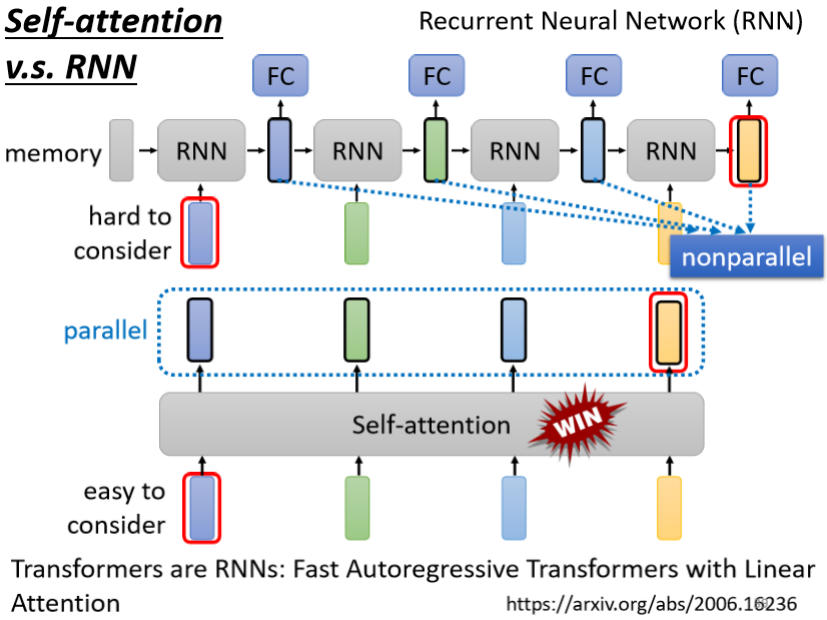
那 Self-attention 跟 RNN 有什么不同呢?

一个显而易见的不同是，对于 Self-attention，输出的每一个 vector都考虑了整个 input 的 sequence，而 RNN 输出的每一个 vector只考虑了左边已经输入的 vector。但是 **RNN 也可以是双向的**，所以如果用 bidirectional 的 RNN，那每一个输出的vector也可以看作是考虑了整个 input 的 sequence。

但是比较两者的 output 产生的过程，就算是用 bidirectional 的 RNN，仍然存在差别：

- 对 RNN 来说，假设最右边这个输出的黄色的 vector 与最左边的这个输入相关性很大，那**它必须要把最左边的输入存在 memory 里面，然后接下来都不能够忘掉，一路带到最右边**，这样才能够在最后一个时间点被考虑。
- 但对 Self-attention 来说没有这个问题，它只要这边输出一个 query，另一边输出一个 key，**只要它们 match 的相关性很大，就可以很轻易地从非常远的 vector 上抽取资讯**，所以这是 RNN 跟 Self-attention 一个不一样的地方。

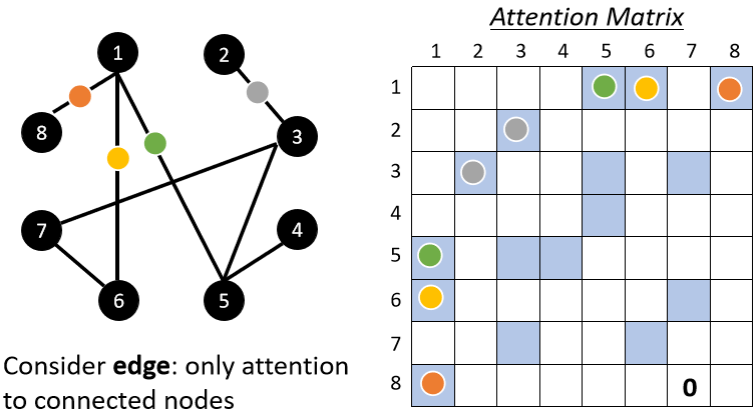
另外一个更主要的不同是，**RNN 在处理的过程中是没有办法并行化的**，因为每一个 RNN 模块的输入都依赖上一个时刻 RNN 模块的输出。但之前我们就提到过 Self-attention 是可以并行处理所有输入的，**输出的四个 vector 是并行产生的，并不需要等谁先运算完才能把其他运算出来。**所以在运算速度上，Self-attention 会比 RNN 更有效率，这是 Self-attention 相较于 RNN 一个非常大的优势。



所以你会发现如今**很多的应用都把 RNN 的架构，逐渐改成 Self-attention 的架构了。**

4.4 Self-attention 在图神经网络上的应用

Graph 也可以看作是一堆 vector，那如果是一堆 vector，就可以用 Self-attention 来处理，但是当我们把 Self-attention用在Graph 上面的时候，又有什么特别的地方呢?



This is one type of **Graph Neural Network (GNN)**.

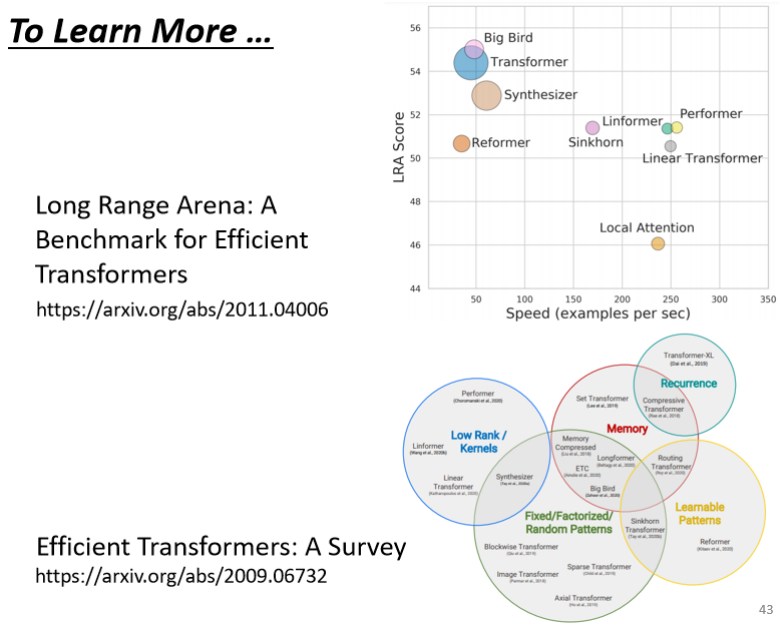
在 Graph 上面，每一个 node 可以表示成一个向量，但**不只有 node 的资讯，还有 edge 的资讯**，我们知道哪些 node 之间是有相连的，也就是哪些 node 是有关联的。

既然我们已经知道哪些向量间是有关联的，之前我们在做 Self-attention 的时候，所谓的关联性是需要 network 自己找出来的，但是现在已经有了 edge 的资讯，**这个图上面的 edge 已经暗示了我们 node 跟 node 之间的关联性。**

所以在我们把 Self-attention 用在 Graph 上面的时候，有一个选择是在做 Attention Matrix 计算的时候，可以**只计算有 edge 相连的 node 就好。**如果两个 node 之间没有相连，那其实很有可能就暗示我们这两个 node 之间没有关系，**既然没有关系，我们就不需要再去计算它们的 attention score，直接设为 0 即可。**

5 Self-Attention 的未来展望

其实 Self-attention 有非常多的变种，你可以看一篇 paper 叫做 Long Range Arena: A Benchmark for Efficient Transformers (arxiv.org)，里面比较了各种不同的 Self-attention 的变种：



因为 Self-attention 最大的问题就是**它的运算量非常地大**，所以怎样减少 Self-attention 的运算量，是一个未来的重点，**到底什么样的 Self-attention 才能够真的又快又好，这仍然是一个尚待研究的问题。**