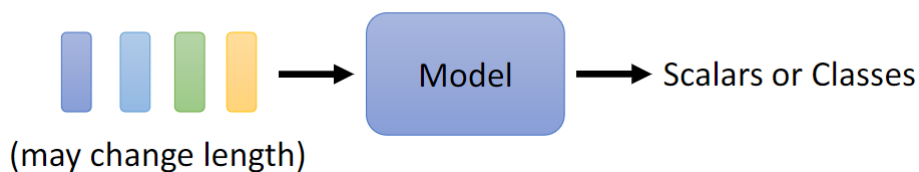


Lecture 4: 自注意力机制 (Self-Attention)

引入

- 致力于解决输入为一组向量的深度学习任务。


- Input is a **set of vectors**



例如👉) ——作业一自然语言处理

Vector Set as Input

this is a cat



One-hot Encoding

apple = [1 0 0 0 0]

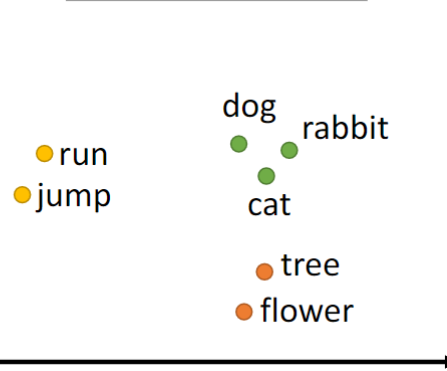
bag = [0 1 0 0 0]

cat = [0 0 1 0 0]

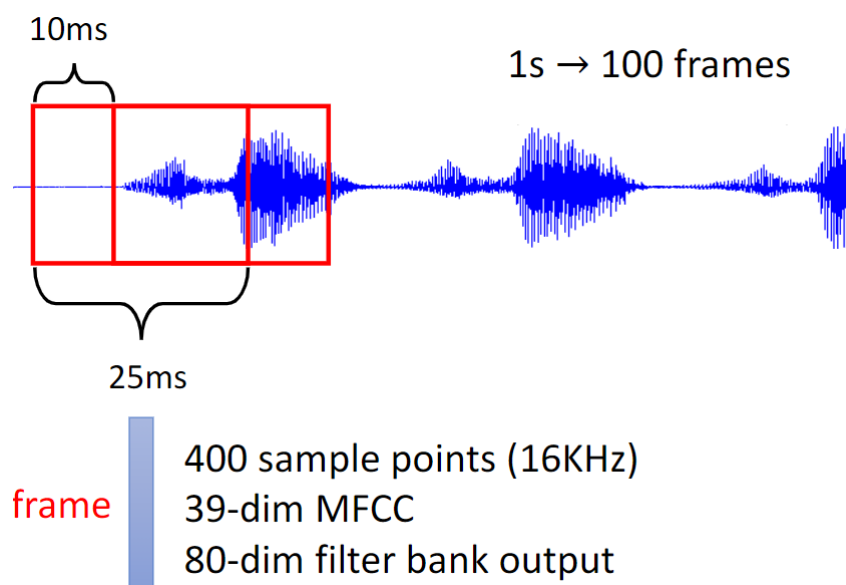
dog = [0 0 0 1 0]

elephant = [0 0 0 0 1]

Word Embedding

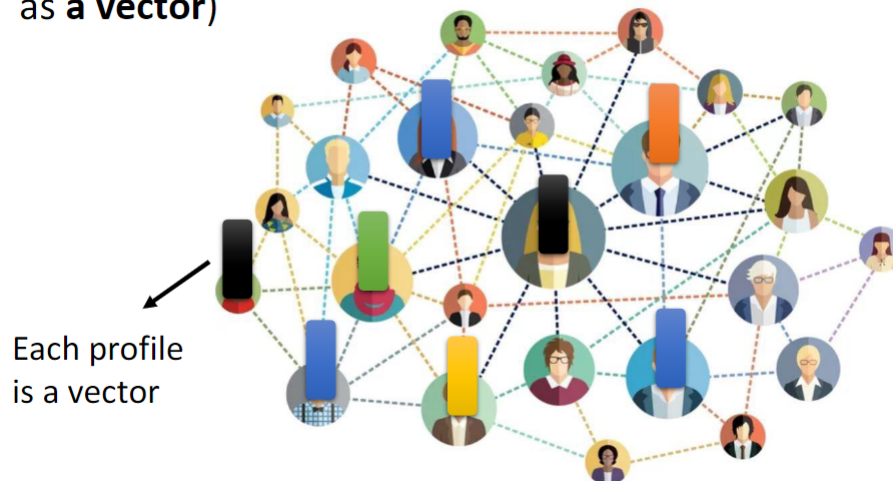


作业二——声音讯息👉



作业三——图👉, 每个节点都可以是一个向量, 包含了人物的社交信息

- Graph is also a set of vectors (consider each **node** as a **vector**)



分子也可以看作“Graph”：（这里化学元素每个原子用one-hot表示）

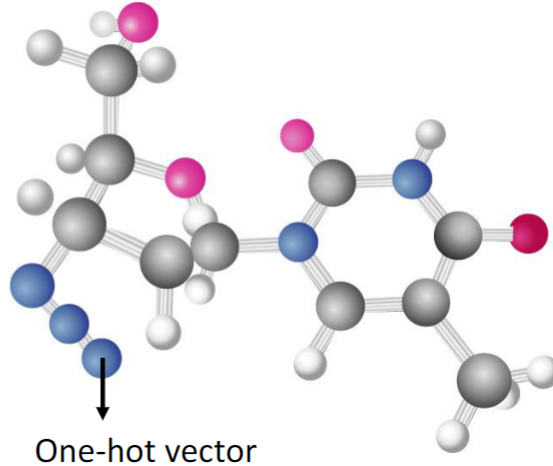
- Graph is also a set of vectors (consider each **node** as a **vector**)

$H = [1 \ 0 \ 0 \ 0 \ 0 \ \dots]$

$C = [0 \ 1 \ 0 \ 0 \ 0 \ \dots]$

$O = [0 \ 0 \ 1 \ 0 \ 0 \ \dots]$

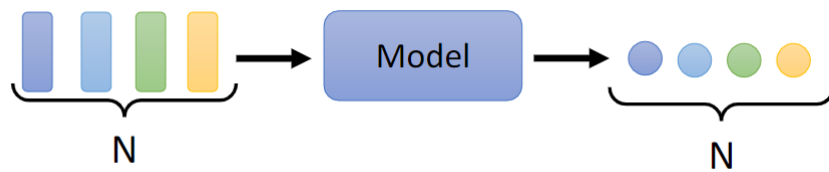
\vdots



- 输出的形式多样：

- 每个vector各自输出一个label：比方说文字处理中的词性标注、语音音素分类、社交网络标签

- Each vector has a label.



Example Applications

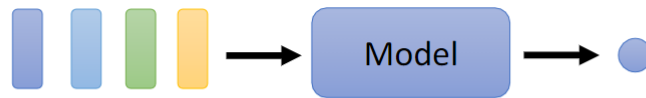
I saw a saw
 ↓ ↓ ↓ ↓
 N V DET N
POS tagging

a a b b
HW2

not
 buy buy

- 一整个sequence输出一个label: 比方说: 情感分析、语者辨认、给一个graph输出一个label

- The whole sequence has a label.

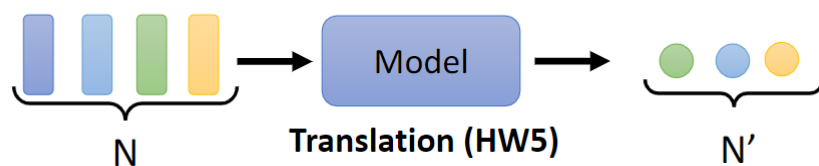


Example Applications



- 模型自己决定输出label的数量——seq2seq任务, 例如: 翻译、完整的语音辨识

- Model decides the number of labels itself. seq2seq

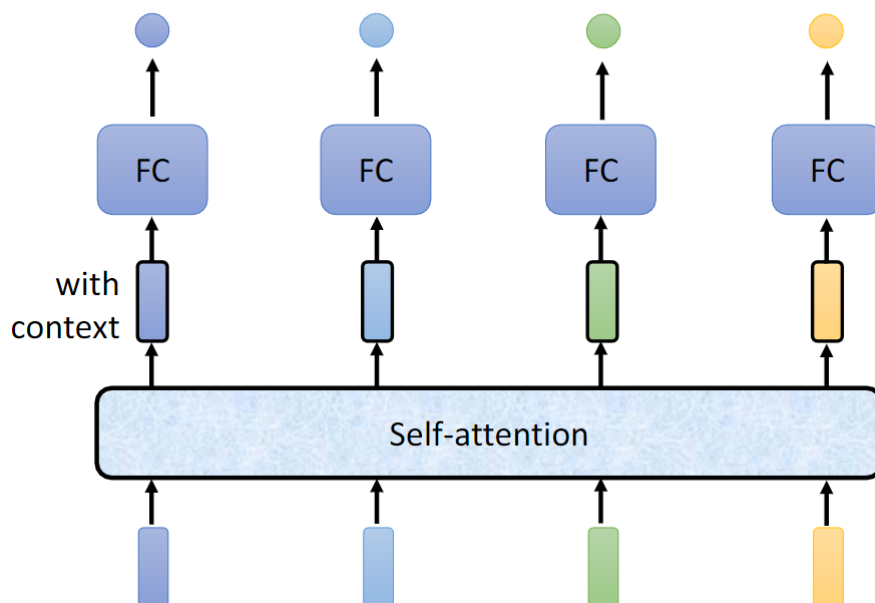


模型一: Sequence Labeling

即上文输入和输出数目一样多的情形。

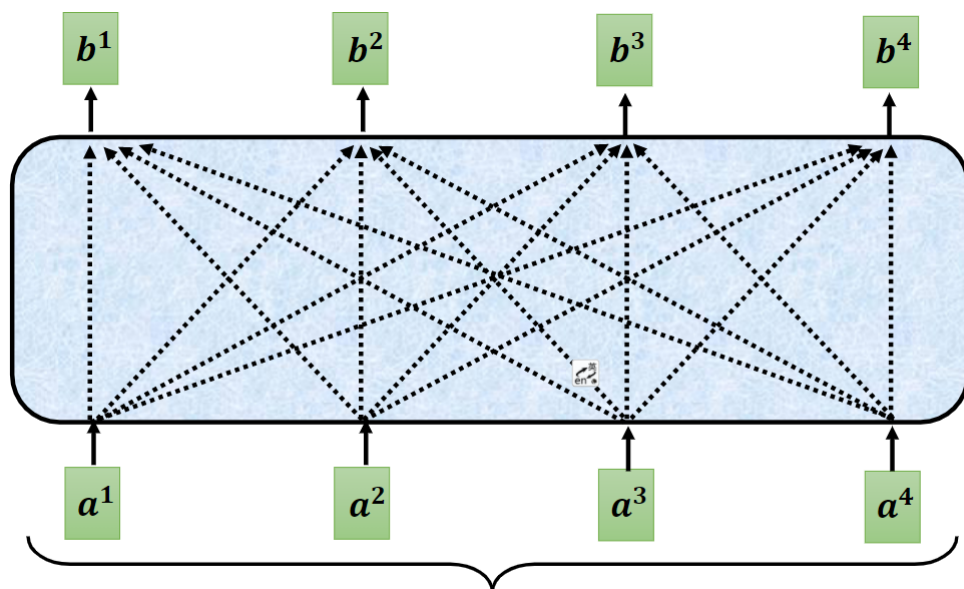
注意到, 对于单个输入vector要关注它的上下文信息。但是, 以某个vector为中心, 为了cover整个sequence, 开一个一定尺寸的窗口输入全连接层中——参数巨多而且有可能overfitting。Self-attention被用来化解这个困难。

Self-attention



FC = Fully Connected Layer

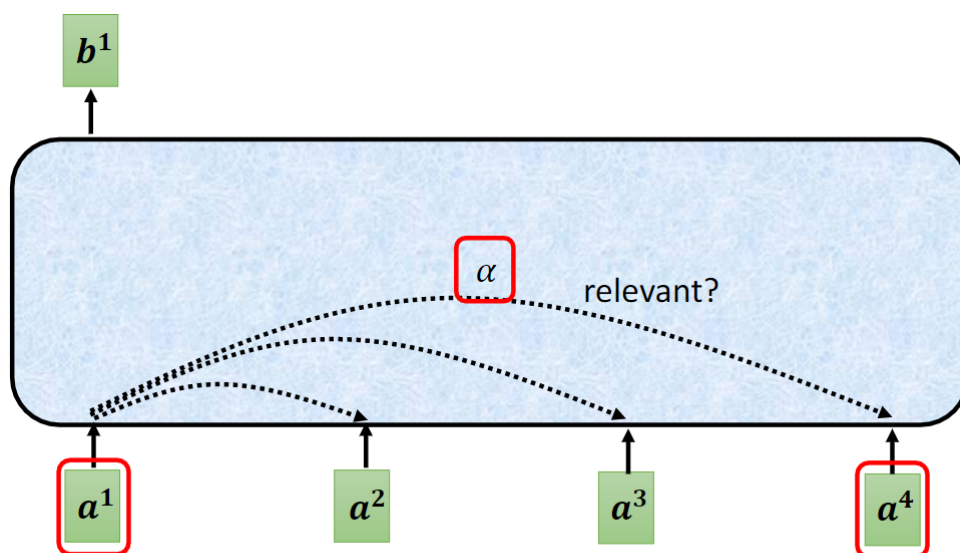
Self-attention考虑到整个sequence的信息, 有多少输入self-attention就有多少输出。模型中经常使用FC和Self-attention交替使用。 [Attention is all you need](#)



Can be either **input** or a **hidden layer**

Self-Attention的每个输出都考虑了所有的输入。以 b^1 为例：

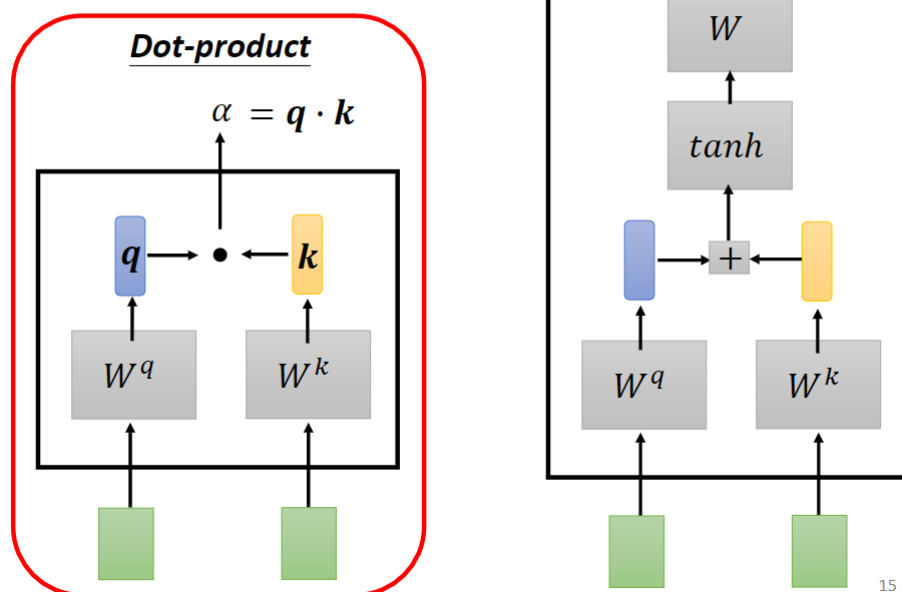
- 首先根据 a^1 找到跟 a^1 相似的向量（找到整个sequence里面哪些部分是重要的哪些部分和 a^1 同一个level、决定 a^1 回归结果数值或者来分类结果class所需要的信息）；每一个向量和 a^1 关联性的值用 α 表示



Find the relevant vectors in a sequence

这个Self-attention的module怎样自动决定两个向量的相关性？以下给出计算两个向量相关性的模组。

Self-attention

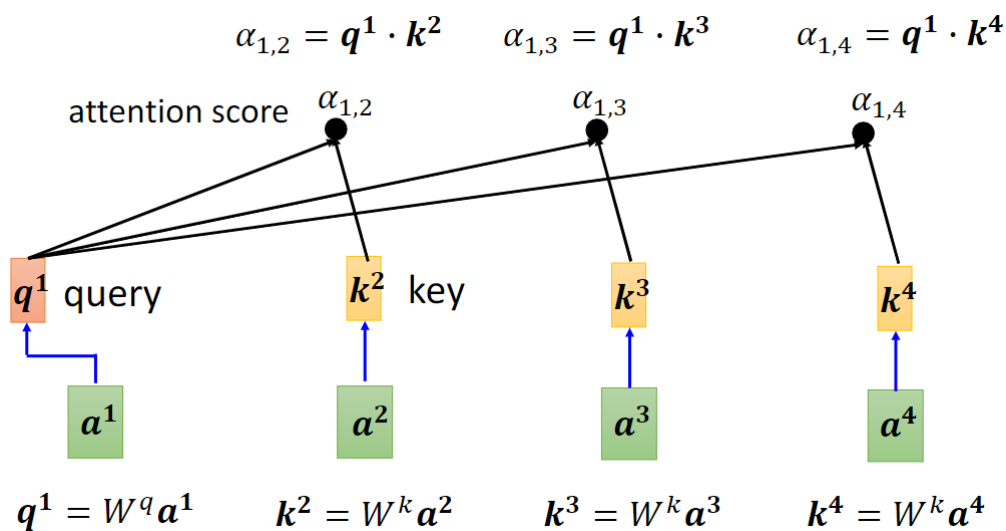


上述比较常见的做法——**Dot-product**：输入的这两个向量（需要计算关联度）分别乘上两个不同的矩阵 W^q 和 W^k ，得到两个向量 q 和 k ，然后这两个向量做 element-wise 相乘，得到 $\alpha = q \cdot k$ 。

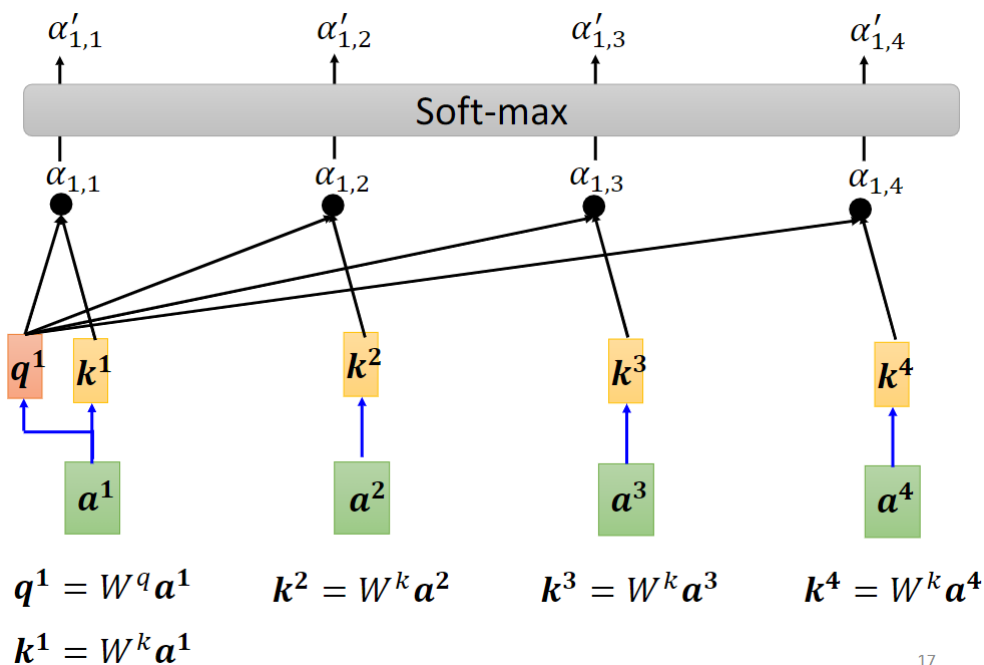
上述另一种计算方式——**Additive**：同样地把这两个向量（需要计算关联度）分别乘上两个不同的矩阵 W^q 和 W^k (inner-product)，得到两个向量 q 和 k ；再然后把这个两个向量串起来，扔进激活函数 \tanh ，然后通过一个 Transform 再得到 α 。（ W 是随机初始化的，然后训练出来的）

在本文中用到的方法默认为左边的 **Dot-product**

- 在 **Self-attention** 中，分别用 a^1 和 a^2 、 a^3 、 a^4 求得对应的 α ——**attention score**。求法如下：



- 自己与自己计算关联性：



17

再把算出来的 α 通过 $Softmax$ 处理:

Softmax:

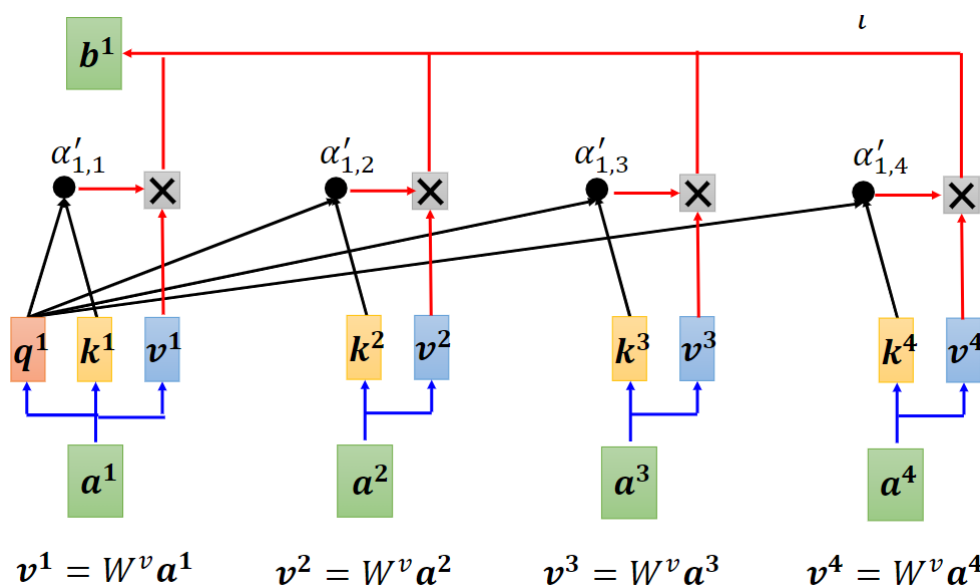
$$\alpha'_{1,i} = \frac{\exp(\alpha_{1,i})}{\sum_j \exp(\alpha_{1,j})} \quad (1)$$

不一定要用Softmax，只要是激活函数，有人用Relu效果也很好。

- Extract information based on attention scores. 根据这些 α' 去抽取整个sequence中比较重要的咨询。

把 a^1 乘上 W^v 得到新的向量 $[v^1, v^2, v^3, v^4]$ 。然后，再把每个 v 乘上 α' ，然后再把它们加起来。

$$b^1 = \sum_i \alpha'_{1,i} v^i \quad (2)$$



上边谁的值最大，谁的那个Attention的分数最大，谁的那个 v 就会dominant你抽出来的结果。举例说：上述中如果 a^2 支计算出来的值最大，那么 b_1 就最接近 v^2

总结

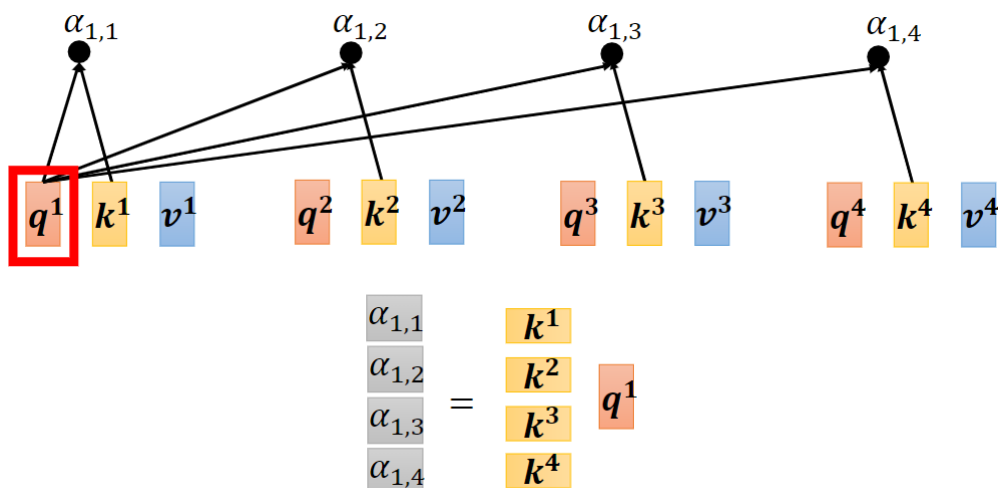
Self-attention就是一排input的vector得到相同数量的output的vector。计算中涉及到三个Transform 矩阵 W^q, W^k, W^v ，是network的参数，是学习（learn）得来的，可以看作是带有权重的，以下认为是 self-attention的矩阵运算。

$$\begin{aligned} q^i &= W^q a^i \\ k^i &= W^k a^i \\ v^i &= W^v a^i \end{aligned} \quad (3)$$

$$\begin{aligned} q^i &= W^q a^i \quad \begin{matrix} q^1 & q^2 & q^3 & q^4 \end{matrix} = \begin{matrix} W^q \end{matrix} \begin{matrix} a^1 & a^2 & a^3 & a^4 \end{matrix} \\ & \quad \quad \quad Q \quad \quad \quad I \\ k^i &= W^k a^i \quad \begin{matrix} k^1 & k^2 & k^3 & k^4 \end{matrix} = \begin{matrix} W^k \end{matrix} \begin{matrix} a^1 & a^2 & a^3 & a^4 \end{matrix} \\ & \quad \quad \quad K \quad \quad \quad I \\ v^i &= W^v a^i \quad \begin{matrix} v^1 & v^2 & v^3 & v^4 \end{matrix} = \begin{matrix} W^v \end{matrix} \begin{matrix} a^1 & a^2 & a^3 & a^4 \end{matrix} \\ & \quad \quad \quad V \quad \quad \quad I \end{aligned}$$

每一个self-attention层只有一个 W^q, W^k, W^v 矩阵。

然后为了得到得分，计算内积



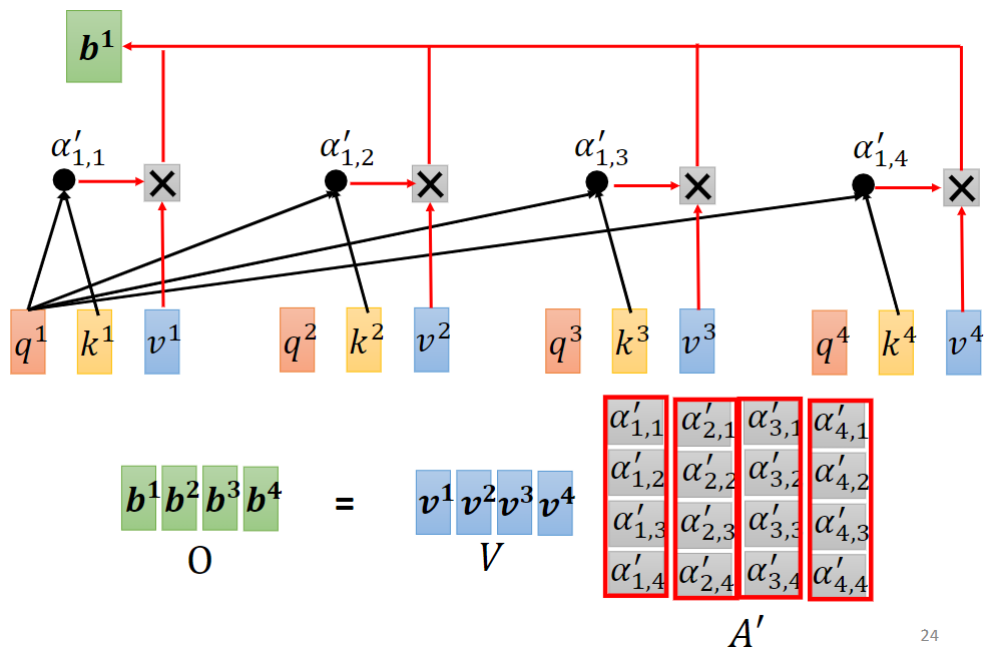
同理

$$\begin{matrix} \alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1} \\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,2} & \alpha'_{4,2} \\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3} \\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4} \end{matrix} \xleftarrow{\text{softmax}} \begin{matrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{matrix} = \begin{matrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{matrix} \begin{matrix} q^1 & q^2 & q^3 & q^4 \end{matrix}$$

$A' \quad \quad \quad A \quad \quad \quad K^T \quad Q$

softmax 不是唯一的选项，也可以用其他激活函数。

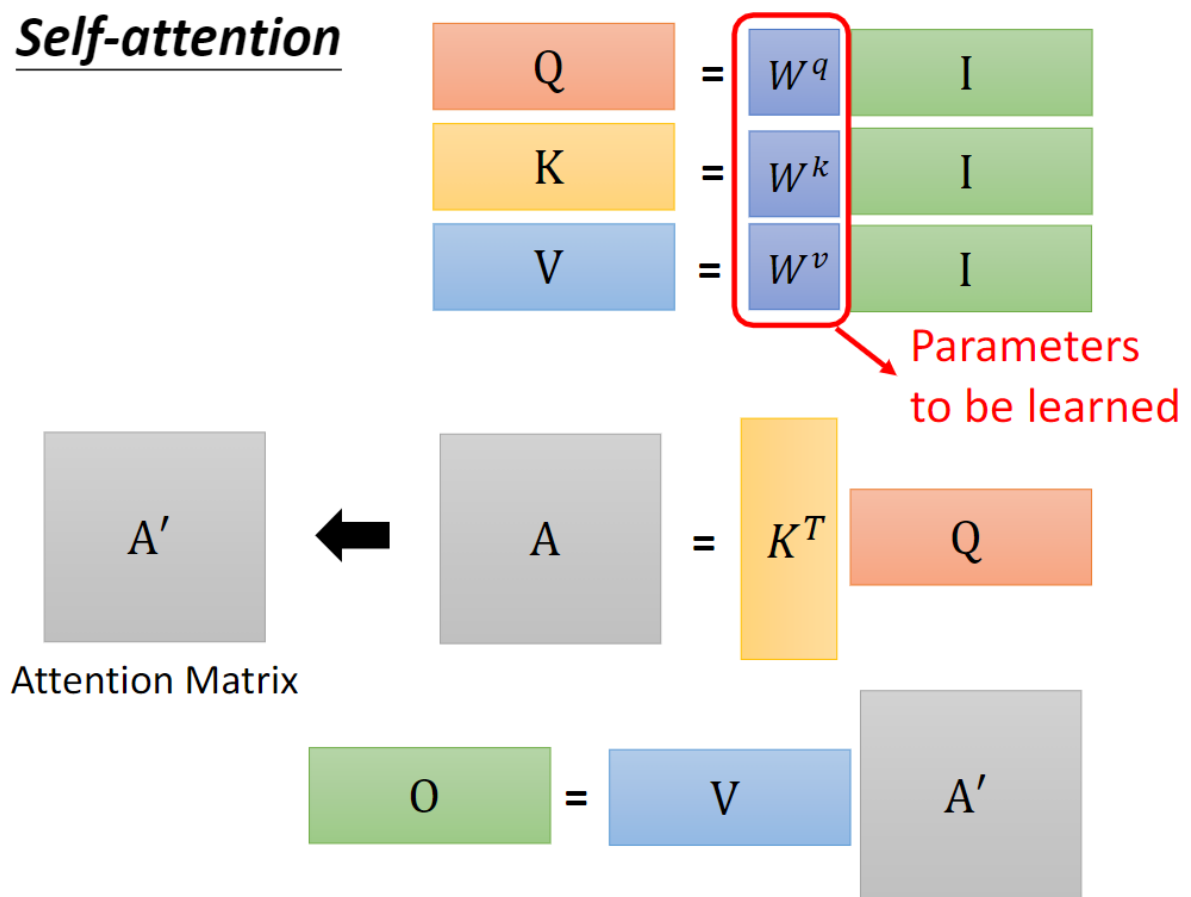
接下来



24

这一串操作全是矩阵运算，不用加循环体，方便编程。把上述过程可以精简为

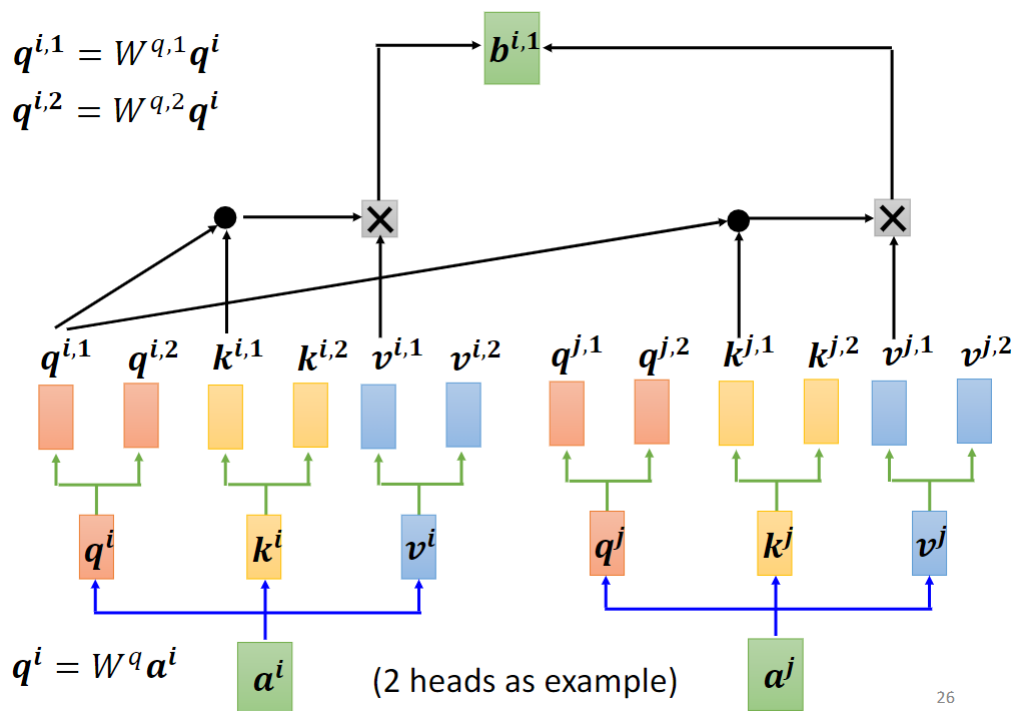
Self-attention



A' 称之为**Attention Matrix**。在整个**Self-attention**中输入是 I ，输出是 O ，其中又只有 W^q, W^k, W^v 是未知的，需要透过训练集（training data）学习得到。

self-attention进阶版——Multi-head Self-attention

为什么我们需要多一点的head呢？——关系中蕴含着不同种类的关联性，以下 2-head 为例：



26

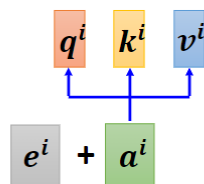
我们需要两种不同的相关性，所以需要产生两种不同的head， q, k, v 都有两个，另外一个位置做相似处理。head 1和head 2相对独立，分开做，如上图， q^1 只和 k^1, v^1 运算。

缺陷——self-attention少了关于位置（上下文）的资讯，因此一下介绍相关的完善方法。

Positional Encoding——把位置的咨询塞进self-attention

- Each position has a unique positional vector e^i （为每一个位置设定一个vector，不用的位置就有专属的一个vector）

- 把 e^i 加到 a^i 上：

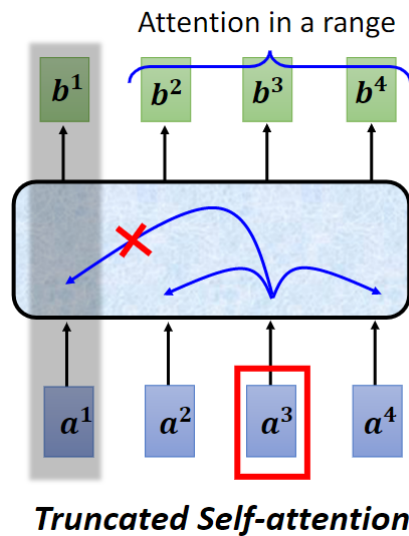


- 这样子的Positional Encoding是**hand-crafted**的，人设的问题包括：可能sequence的长度超过人设的范围。在[Attention is all you need](#)中这个代表位置的vector是透过一个规则产生的：一个神奇的sin、cos的function
- Positional Encoding任然是一个尚待研究的问题，可以创造一个新的产生办法，可以**learn from data**

[这篇论文](#)讨论了Positonal Encoding的生成方法。

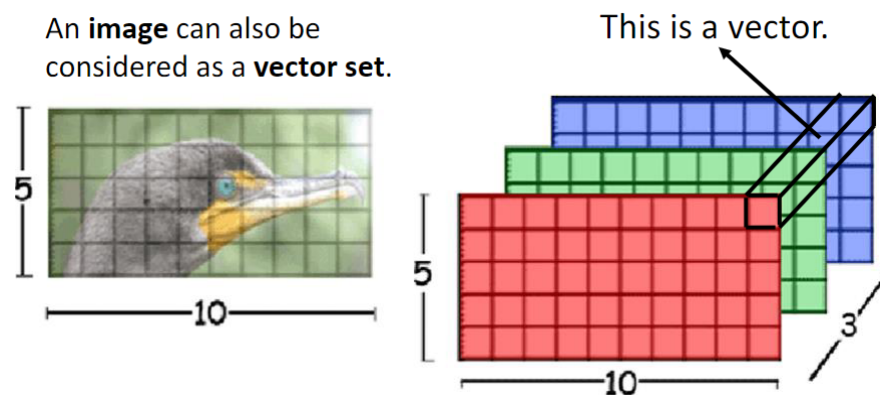
Many applications of Self-attetntion

- [Transformer](#)
- [BERT](#)
- 不仅在NLP领域，self-attention也可以用来语音识别（Speech）：[Transformer-Transducer](#)。文章中，self-attention被做了小小的改动。语音是一条非常长的sequence，由于时间序列下，为了描述一个时间段的语音信号，向量维数非常大，如果sequence的长度为 L ，为了计算Attention Matrix，需要做 $L \times L$ 次的inner product，算力和memory的压力很大。**Truncated Self-attention**被设计用来在只看一个小的范围（范围由人设定）而非整句话，以加快运算速度。



- **self-attention for Image:**

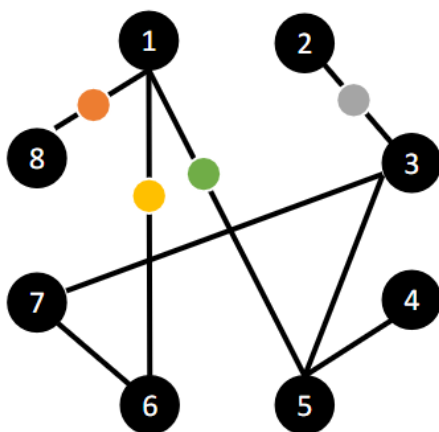
一张图片可以看作是一个vector的set



例如上图：每个位置的pixel都可以看作是一个三维的vector，所以这张图片是一个 5×10 的 vectors set。self-attention处理图片的工作的例子：[Self-Attention GAN](#)、[DEtection Transformer\(DETR\)](#)

- **self-attention for Graph:**

在Graph里，每个**node**看作一个向量（保存有相关的信息）；另外，graph里还有**edge**的信息。哪些node相连——哪些node有关联性：因此，邻接矩阵表示了在做self-attention的计算时，只需要计算相连的node之间的关联性就好了。



Consider **edge**: only attention to connected nodes

Attention Matrix

	1	2	3	4	5	6	7	8
1					●	●		●
2			●					
3		●						
4								
5	●							
6	●							
7								
8	●						0	

没有相连的nodes之间就不用计算attention score了，可设置为0，因为这些可能是domain knowledge暗示下的这种nodes间没有关系。

由此，提出了一个很fancy的网络：**Graph Neural Network (GNN)图神经网络**。老师表示水很深，把握不住，感兴趣可以另外自行学习。

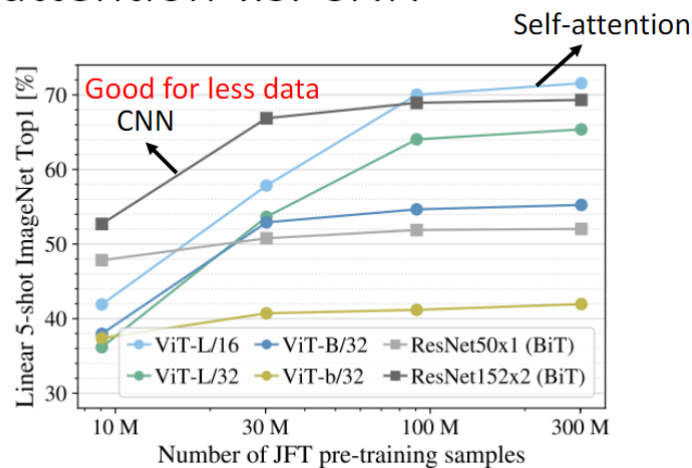
Self-attention和CNN的比较

CNN可以看作是一种简化版的Self-attention，它只关注于receptive field；而self-attention则关注整张图像。self-attention看作是复杂化的CNN，用attention找到有关联性的pixel，仿佛是network自动learn且决定自己的“receptive field”（不再是人工划定）

[On the Relationship between Self-Attention and Convolutional Layers](#)用数学的方式严谨的证明CNN是self-attention的一个特例。self-attention设定特定的参数就可以做到和CNN一样的事情。

由于self-attention相较于CNN更加flexible，为了避免过拟合，需要更多的数据才能达到更好的效果。而CNN在训练资料较少时表现相对较好，因为随着数据增多，CNN并没有得到更多好处。

Self-attention v.s. CNN



[An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#)这篇文章用self-attention处理影像，它把一张图像拆成 16×16 个patch，把每个patch当作一个word处理。（当然这个数据集量一般研究者很难搜集到，这篇文章来自Google）

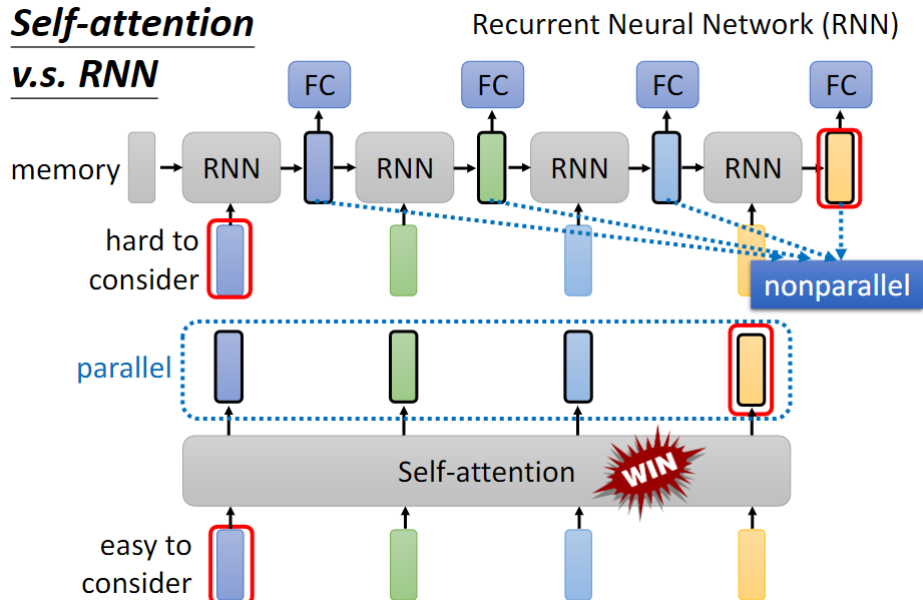
Conformer：一个CNN和Self-attention的混合体。

Self-attention和RNN的比较

RNN: Recurrent Neuroal Network (循环神经网络)和self-attention一样都是处理input是一个sequence的状况，在第一个RNN里扔进input第一个vector，然后output一个东西 \Rightarrow hidden layer \Rightarrow FC \Rightarrow prediction，对于第二个RNN需要input第一个吐出来的东西以及input第二个vector再output东西，以此类推，如下图👇

Self-attention

v.s. RNN



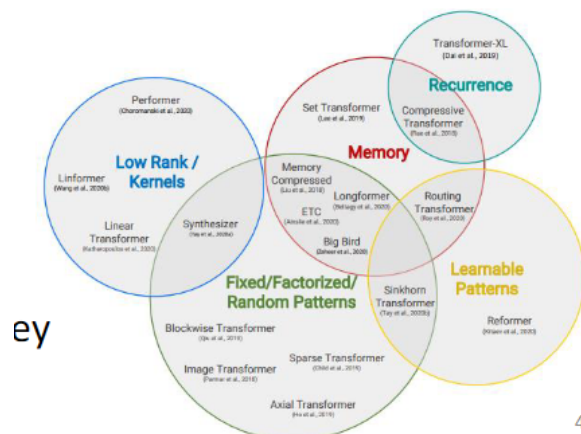
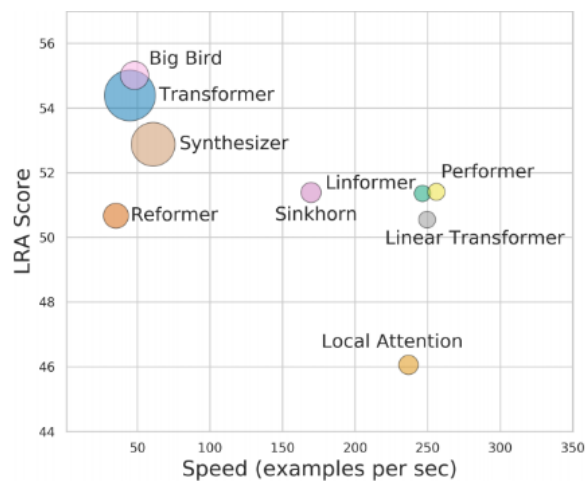
当然，RNN可以是双向的。两者不同的地方：对于RNN而言，距离较远的两个vector，如果前者不被memory一直记忆到输入处理后的网络，两个向量很难产生关联性；而再attention里，输入向量是平行的，输出向量是平行的，只要match到，就可以产生任意两个向量的关联性。——天涯若比邻，aha

所以目前来看attention优于RNN，许多RNN架构都改为attention了。进一步了解两者关系：

[Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention](#)，attention加一点东西就会变成RNN。

延展

Self-attention有非常多的变形：[Long Range Arena: A Benchmark for Efficient Transformers](#)、[Efficient Transformers: A Survey](#)



由于self-attention最大的问题就是运算量大，所以未来相关的问题很多关于如何变形以减少运算量，提高运算速度。如何使attention越来越好，也是未来尚待研究的问题。