

Lecture 5 : Sequence to sequence

Create at 2022/06/01

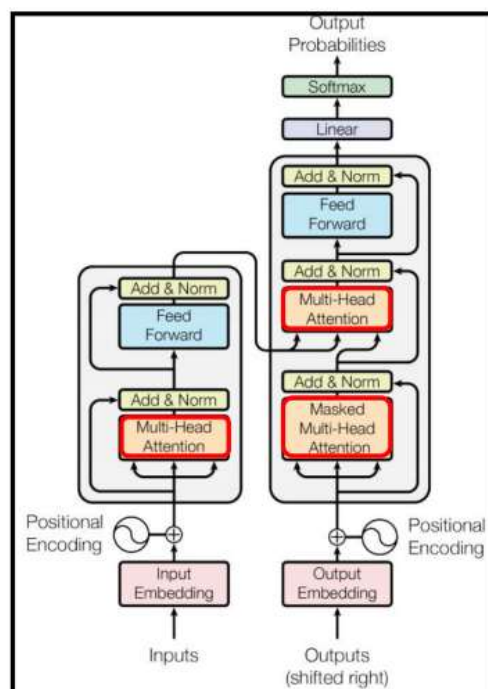
- Lecture 5 : Sequence to sequence
 - 各式各樣的 self-attention
 - Local attention / Truncated attention
 - Stride attention
 - Global attention
 - Clustering
 - Sinkhorn Sorting Network
 - Linformer
- 上課資源 :
 1. 各式各樣神奇的自注意力機制 (Self-attention) 變型 (https://www.youtube.com/watch?v=yHoAq1IT_oq).
- 預習教材 :
 - 【機器學習2021】自注意力機制 (Self-attention) (上) (<https://www.youtube.com/watch?v=hYdO9CscNes>).
 - 【機器學習2021】自注意力機制 (Self-attention) (下) (<https://www.youtube.com/watch?v=gmsMY5kc-zw>).

各式各樣的 self-attention

Notice

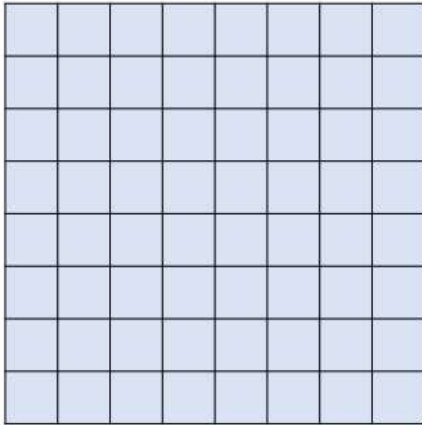
- Self-attention is only a module in a larger network.
- Self-attention dominates computation when N is large.
- Usually developed for image processing

256  256 $N = 256 * 256$



- self-attention 只是一個很大的 network 裡面其中一小部分
- 當 sequence 很長的時候，加快 self-attention 才有可能發揮真正的效果

Skip Some Calculations with Human Knowledge

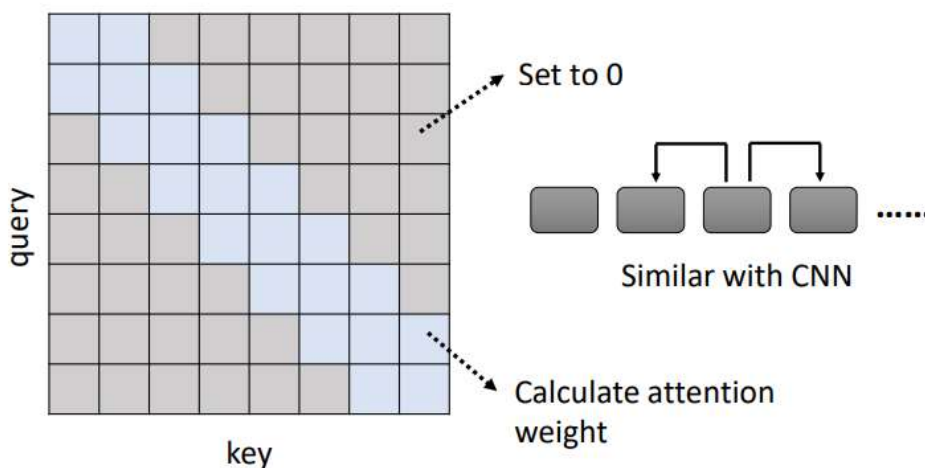


Can we fill in some values with human knowledge?

透過人類的知識直接填入正確資訊，減少運算

Local attention / Truncated attention

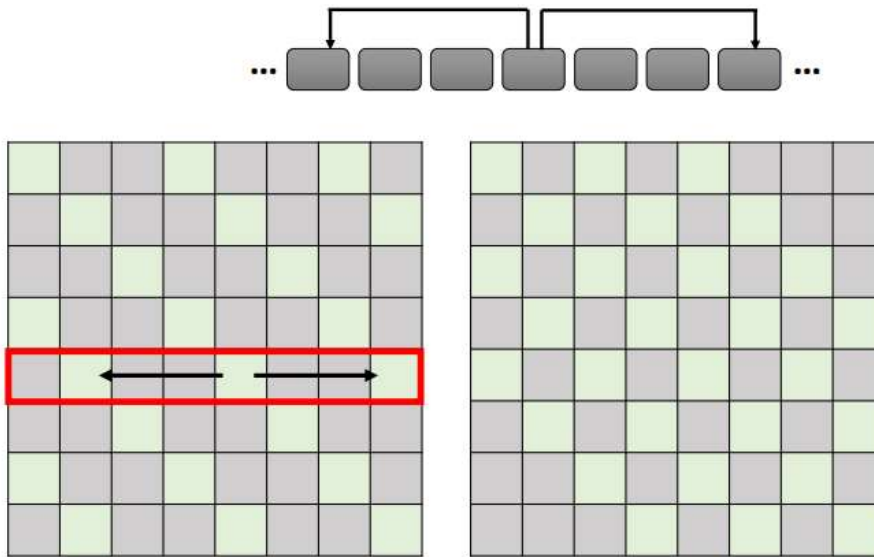
Local Attention / Truncated Attention



- 有些問題不需要看整個 sequence，只需要看左右鄰居就可以得到一個位置有什麼樣的資訊
- 可以加快運算速度，只計算藍色的部分
- Local attention 是可以加快 self-attention 的方法

Stride attention

Stride Attention



- 看三格之前的資訊與三格之後的資訊，可以看到比較大範圍的資訊
- 只計算綠色的部分

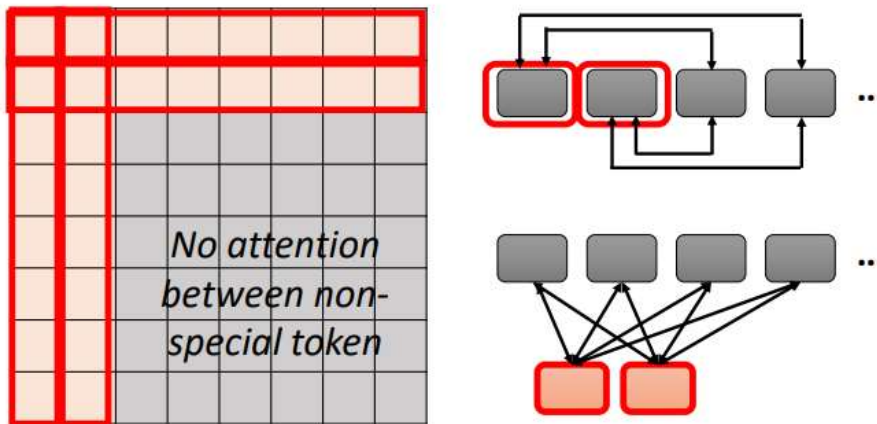
Global attention

Global Attention

special token = “token中的里長伯”

Add special token into original sequence

- Attend to every token → collect global information
- Attended by every token → it knows global information

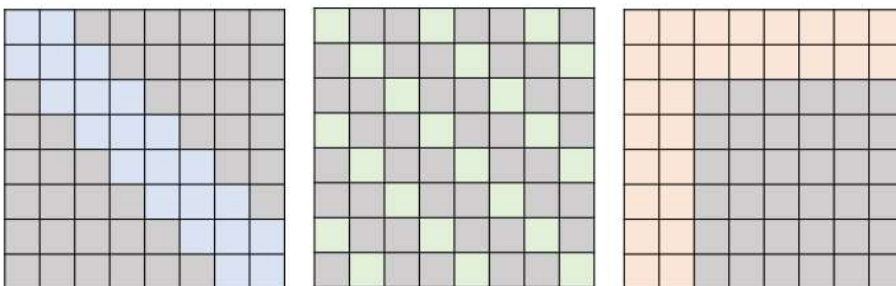


- 如果要看一整個 sequence 發生甚麼事，可以用 global attention
- 在原本的 sequence 裡面加上特殊的 token，代表這個位置需要做 global attention

Global attention 會做甚麼事呢？

- attend 到 sequence 裡的每一個 token，到 sequence 裡的每一個 token 收集資訊
- 法 1：在原本的 sequence 裡面直接 assign 一些 token 做為 special token
- 法 2：用外加 token，special token 會 attend 到所有其他的 token

Many Different Choices ...



小孩子才做選擇 . . .

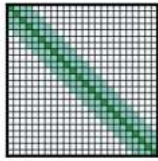
Different heads use different patterns.

真正好的選擇是...全部都用

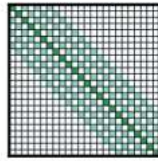
Many Different Choices ...

• Longformer

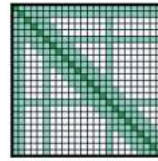
<https://arxiv.org/abs/2004.05150>



(b) Sliding window attention



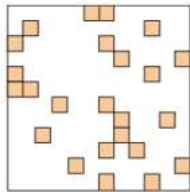
(c) Dilated sliding window



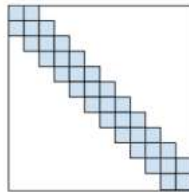
(d) Global+sliding window

• Big Bird

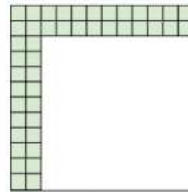
<https://arxiv.org/abs/2007.14062>



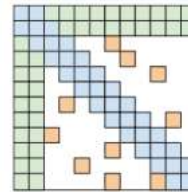
(a) Random attention



(b) Window attention



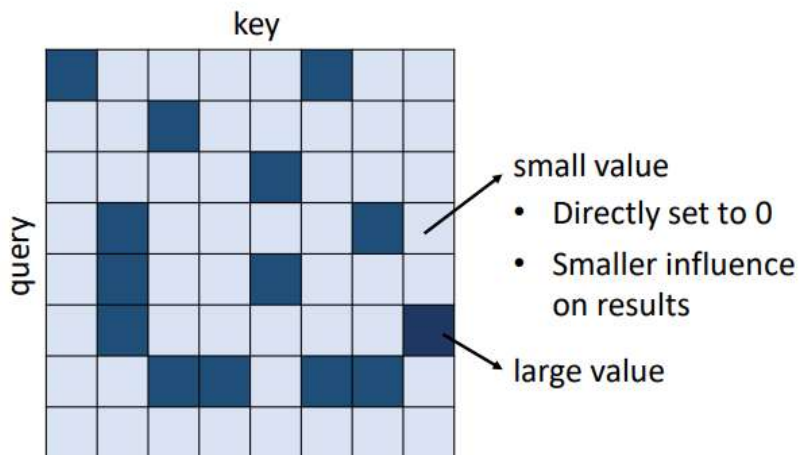
(c) Global Attention



(d) BiGBIRD

self-attention 的變形

Can we only focus on Critical Parts?



How to quickly estimate the portion with small attention weights?

去計算有可能有比較大的 attention value 的位置

Clustering

Clustering

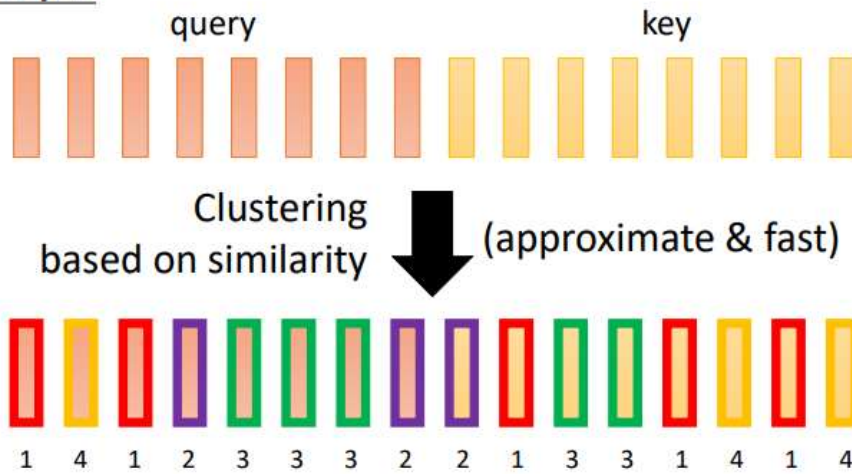
Reformer

<https://openreview.net/forum?id=rkgNKkHtvB>

Routing Transformer

<https://arxiv.org/abs/2003.05997>

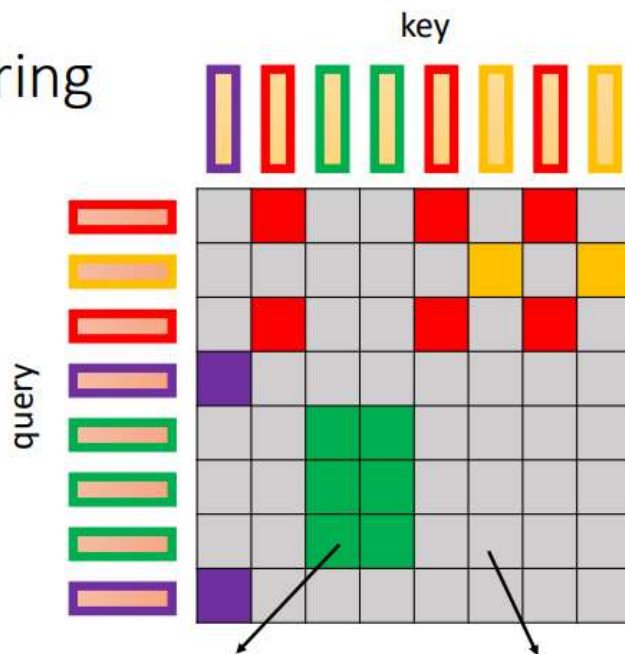
Step 1



- 先把 query 跟 key 拿出來，根據 query 跟 key 相近的程度做 clustering
- 相近的就分類在一起，比較遠的就屬於不同的 clustering (相同顏色框的是相同的 clustering)

Clustering

Step 2



Belong to the same cluster, then calculate attention weight

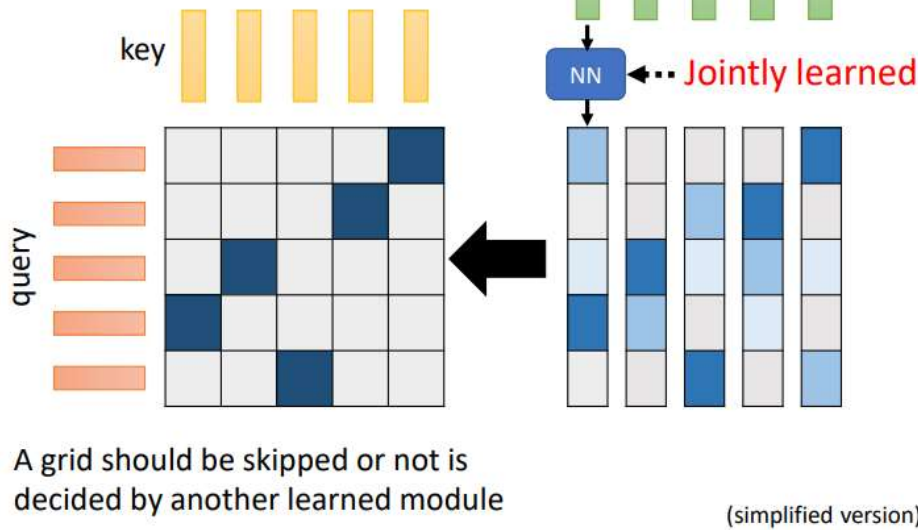
Not the same cluster, set to 0

- 當 query 跟 key 在同一個 clustering 裡面的話，去計算它的 attention weight
- 否則直接設為 0

Sinkhorn Sorting Network

<https://arxiv.org/abs/2002.11296>

Learnable Patterns Sinkhorn Sorting Network



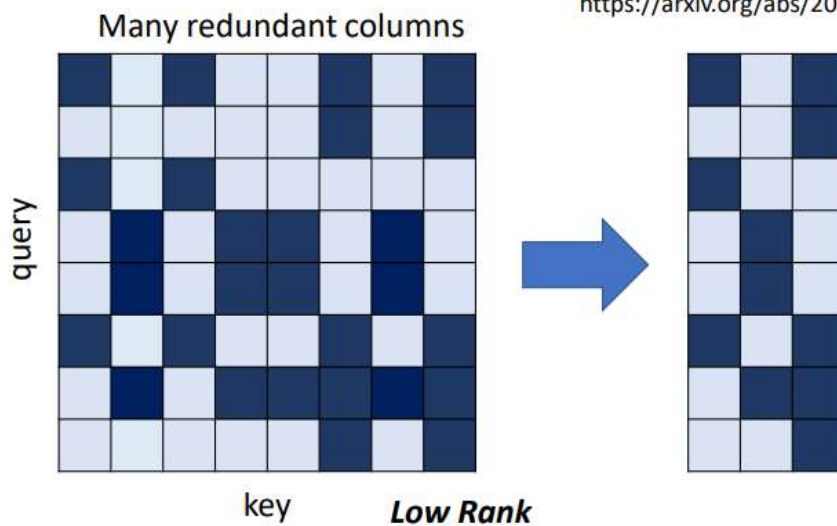
- 在 Sinkhorn Sorting Network 用另外一個 network 來決定哪些地方需要計算 attention

Linformer

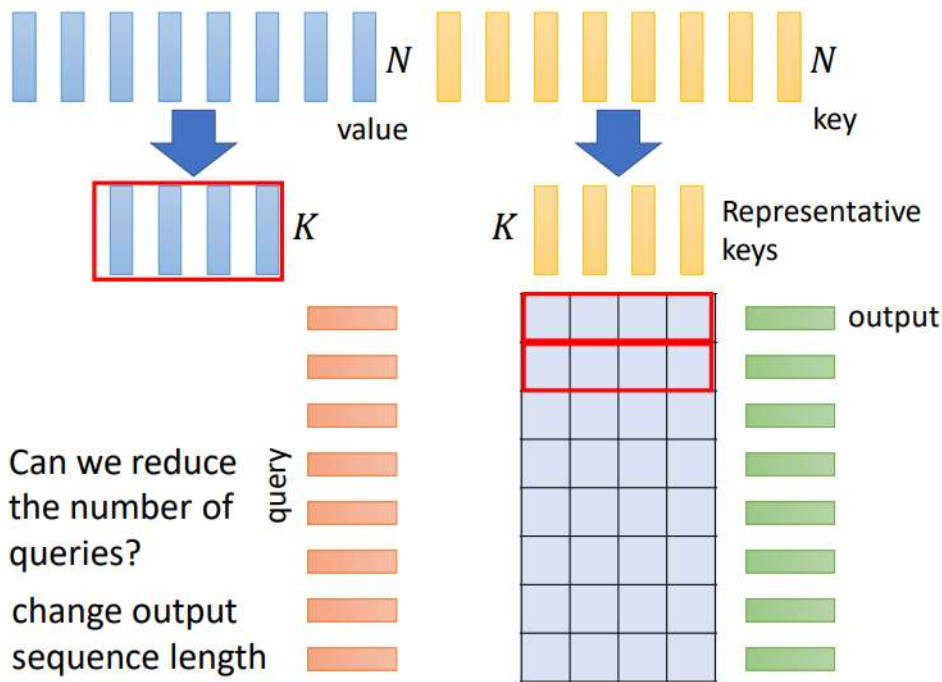
Do we need full attention matrix?

Linformer

<https://arxiv.org/abs/2006.04768>



- 把重複的 column 拿掉，變成一個比較小的 attention matrix
- 可以加快運算速度



- 從 N 個 key 中，選有代表性的 K 個 key
- 從 N 個 value vector 中，選有代表性的 K 個 value vector
- 把 K 個 key 對第一個 query vector 算出來的 attention weight 對 K 個 value vector 做 weighted sum 得到 output
- 以此類推，可以得到整個 self-attention 的 output

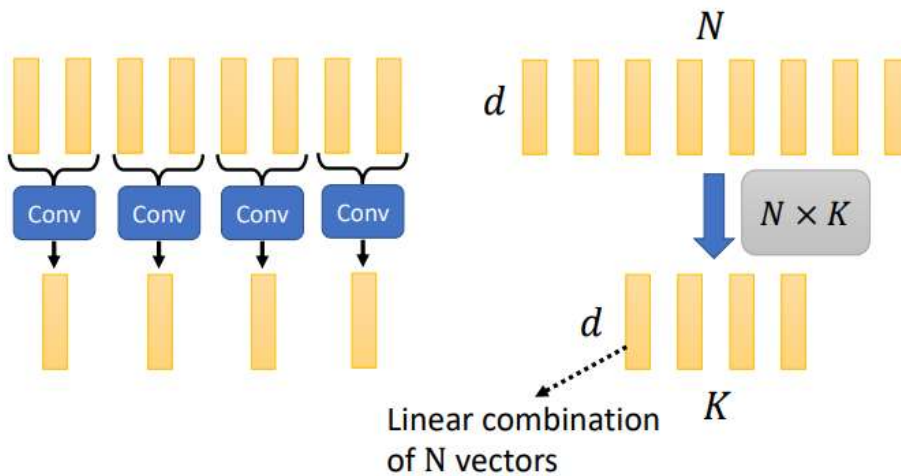
為甚麼不選有代表性的 query ?

- 如果只選有代表性的 query，會遇到 output sequence length 縮小，因為 output sequence length = query length
- 會不會受影響呢？
 - case by case
 - 如果 input 一段句子，只需要輸入一個 label，這種例子就不會受影響，可以選有代表性的 query 就好
 - 如果是 input sequence 的每個位置都需要 output 一個 label，就會有問題

Reduce Number of Keys

Compressed Attention
<https://arxiv.org/abs/1801.10198>

Linformer
<https://arxiv.org/abs/2006.04768>

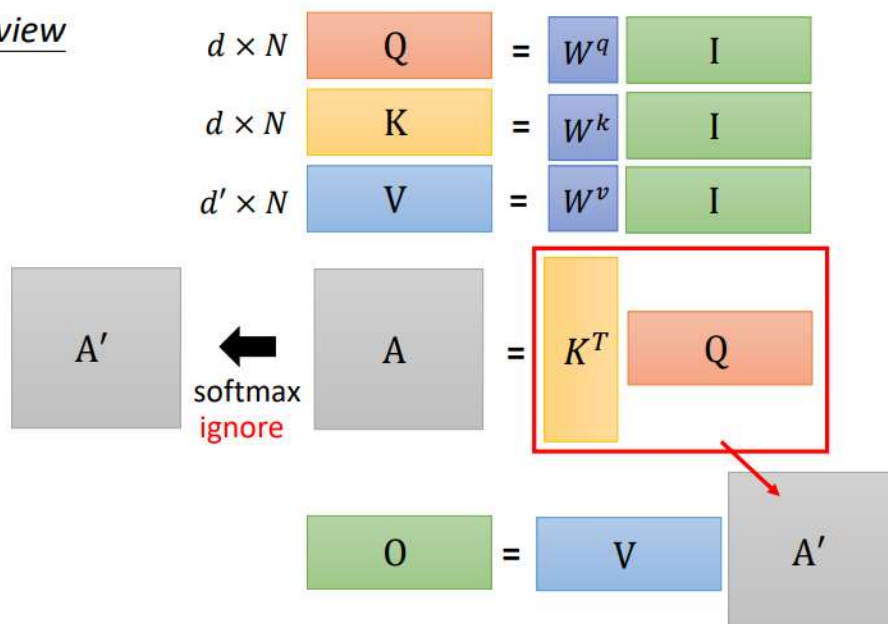


怎麼選出有代表性的 key ?

- 法1 : Compressed Attention
 - input 很長 key 的 sequence，用 CNN 掃過 sequence 將 sequence 長度變短，這些 sequence 就當成有代表性的 key
- 法2 : Linformer
 - input 的 key 集合起來，可以看做是 $d * N$ 的矩陣
 - 把 $d * N$ 的矩陣乘上 $N * K$ 的矩陣得到 $d * K$ 的矩陣 (Linear combination)
 - $d * K$ 的矩陣， K 個 column 就是有代表性的 key

Attention Mechanism is three-matrix Multiplication

Review



Attention Mechanism is three-matrix Multiplication

Attention Mechanism is three-matrix Multiplication

Review

$$\begin{aligned}
 d \times N \quad \text{Q} &= W^q \quad \text{I} \\
 d \times N \quad \text{K} &= W^k \quad \text{I} \\
 d' \times N \quad \text{V} &= W^v \quad \text{I}
 \end{aligned}$$

$$\begin{aligned}
 \text{O} &\approx \text{V} \underbrace{\text{K}^T \text{Q}}_{N \times d} \\
 d' \times N &\quad d' \times N \quad d \times N
 \end{aligned}$$

式子簡化

$$\begin{aligned}
 \text{O} &\approx \text{V} \underbrace{\text{K}^T \text{Q}}_{N \times d} \\
 d' \times N &\quad d' \times N \quad d \times N
 \end{aligned}$$

$$\begin{aligned}
 \text{O} &\approx \underbrace{\text{V} \text{K}^T}_{N \times d} \text{Q} \\
 d' \times N &\quad d' \times N \quad d \times N
 \end{aligned}$$

What is the difference?

其實可以再加速，上下兩個式子有甚麼不同？

- 得到的結果相同，但中間需要的運算量不同

Review Linear Algebra

Practical Issue

$k=1$ $m=1000$
 $n=1$ $p=1000$

- Let A and B be $k \times m$ matrices, C be an $m \times n$ matrix, and P and Q be $n \times p$ matrices
 - $A(CP) = (AC)P$

Left side (A * C) * P:

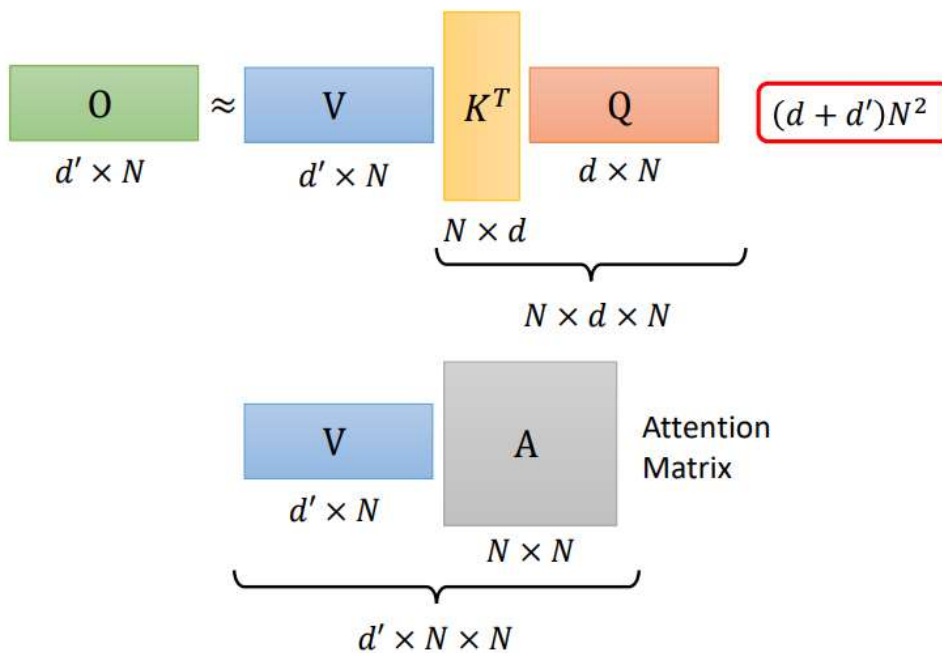
- A: $k \times m$
- C: $m \times n$
- P: $n \times p$
- AC: $k \times n$
- Operations: $m \times n \times p$ (for AC), $k \times m \times p$ (for final result)
- Values: 10^6 (for AC), 10^6 (for final result)

Right side (A * (C * P)):

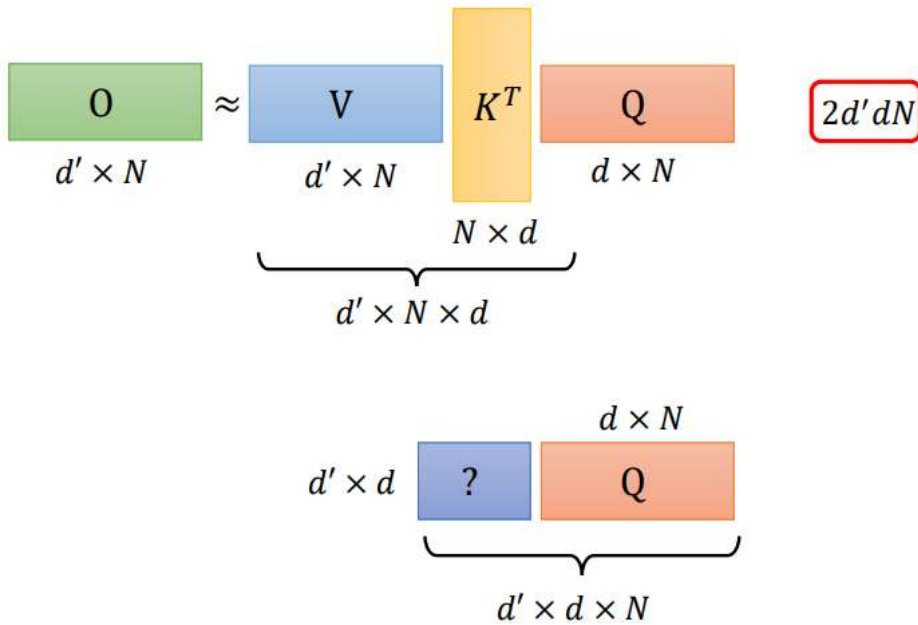
- A: $k \times m$
- C: $m \times n$
- P: $n \times p$
- AC: $k \times n$
- Operations: $k \times m \times n$ (for CP), $k \times n \times p$ (for final result)
- Values: 10^3 (for CP), 10^3 (for final result)

<https://youtu.be/yO8lDzf4jMs>

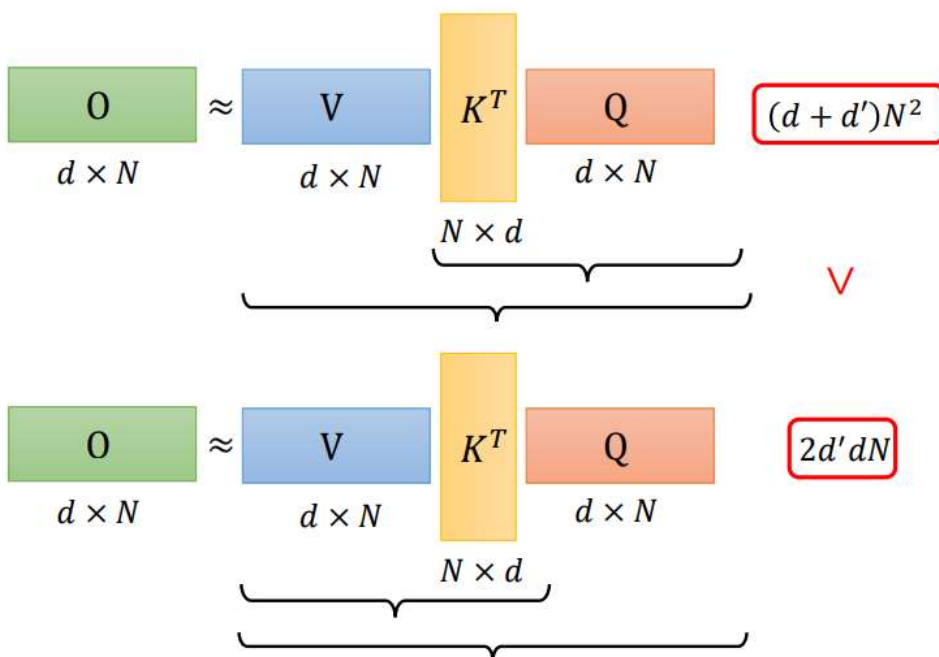
線性代數觀念複習



先做 $K^T * Q$ 得到 A ，再乘上 V
 得到的運算量總共是 $(d + d')N^2$



先做 $V * K^T$ ，再乘上 Q
 得到的運算量總共是 $2d'dN$



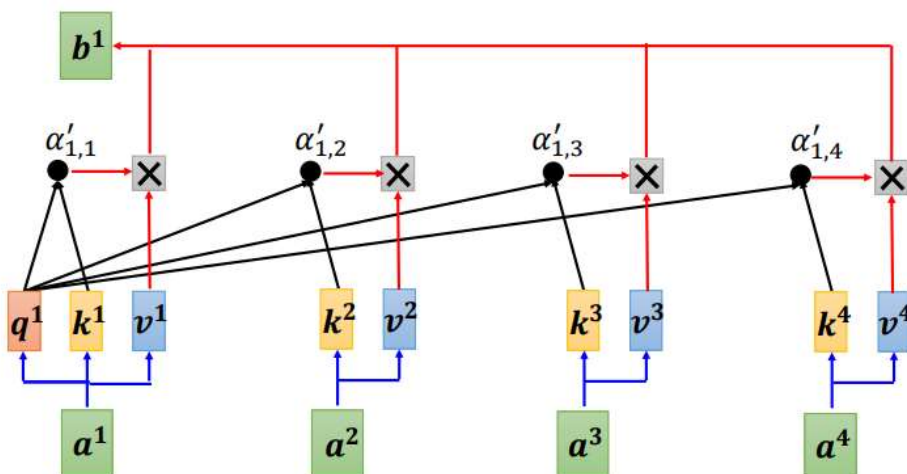
改變乘法的順序，運算的次數會有很大的改變

老師說不想聽數學很多的話，可以跳過ㄟ 那我要跳過ㄌ～

Let's put softmax back ...

Warning of math

$$b^1 = \sum_{i=1}^N \alpha'_{1,i} v^i = \sum_{i=1}^N \frac{\exp(q^1 \cdot k^i)}{\sum_{j=1}^N \exp(q^1 \cdot k^j)} v^i$$



$$b^1 = \sum_{i=1}^N \alpha'_{1,i} v^i = \sum_{i=1}^N \frac{\exp(q^1 \cdot k^i)}{\sum_{j=1}^N \exp(q^1 \cdot k^j)} v^i$$

$$\begin{aligned} \frac{\exp(q \cdot k)}{\sum_{j=1}^N \exp(q \cdot k^j)} &\approx \frac{\phi(q) \cdot \phi(k)}{\sum_{j=1}^N \phi(q) \cdot \phi(k^j)} v^i \\ \text{Diagram: } q \rightarrow \phi \rightarrow \phi(q) &= \frac{\sum_{i=1}^N [\phi(q^1) \cdot \phi(k^i)] v^i}{\sum_{j=1}^N \phi(q^1) \cdot \phi(k^j)} \end{aligned}$$

$$\phi(q^1) \cdot \sum_{j=1}^N \phi(k^j)$$

Diagram: A yellow box containing the sum $\sum_{j=1}^N \phi(k^j)$ is connected by a dotted arrow to a red box containing a yellow bar and a vertical orange bar, labeled $\phi(q^1)$.

$$b^1 = \sum_{i=1}^N \alpha'_{1,i} v^i = \frac{\sum_{i=1}^N [\phi(q^1) \cdot \phi(k^i)] v^i}{\phi(q^1) \cdot \sum_{j=1}^N \phi(k^j)}$$

$$\sum_{i=1}^N [\phi(q^1) \cdot \phi(k^i)] v^i \quad \phi(q^1) = \begin{bmatrix} q_1^1 \\ q_2^1 \\ \vdots \end{bmatrix} \quad \phi(k^1) = \begin{bmatrix} k_1^1 \\ k_2^1 \\ \vdots \end{bmatrix}$$

$$= [\phi(q^1) \cdot \phi(k^1)] v^1 + [\phi(q^1) \cdot \phi(k^2)] v^2 + \dots$$

$$= (q_1^1 k_1^1 + q_2^1 k_2^1 + \dots) v^1 + (q_1^1 k_1^2 + q_2^1 k_2^2 + \dots) v^2 + \dots$$

$$= \underline{q_1^1 k_1^1 v^1} + \underline{q_2^1 k_2^1 v^1} + \dots + \underline{q_1^1 k_1^2 v^2} + \underline{q_2^1 k_2^2 v^2} + \dots + \dots$$

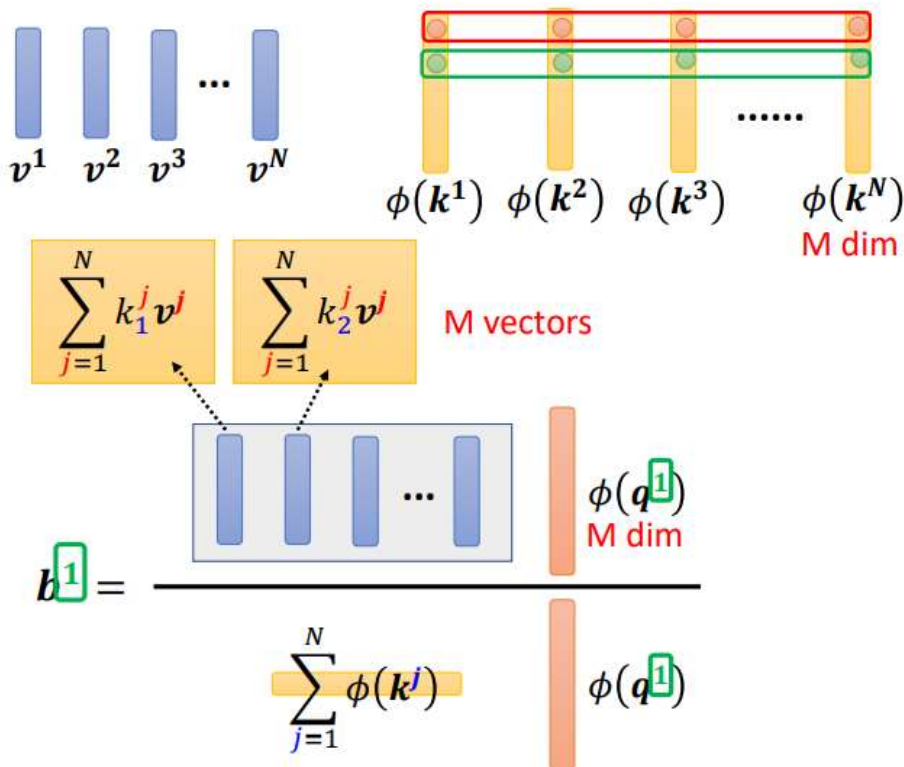
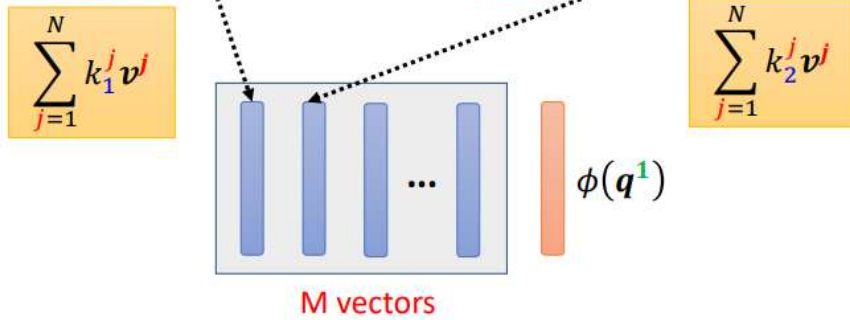
$$= q_1^1 (k_1^1 v^1 + k_2^1 v^2 + \dots) + q_2^1 (k_2^1 v^1 + k_2^2 v^2 + \dots)$$

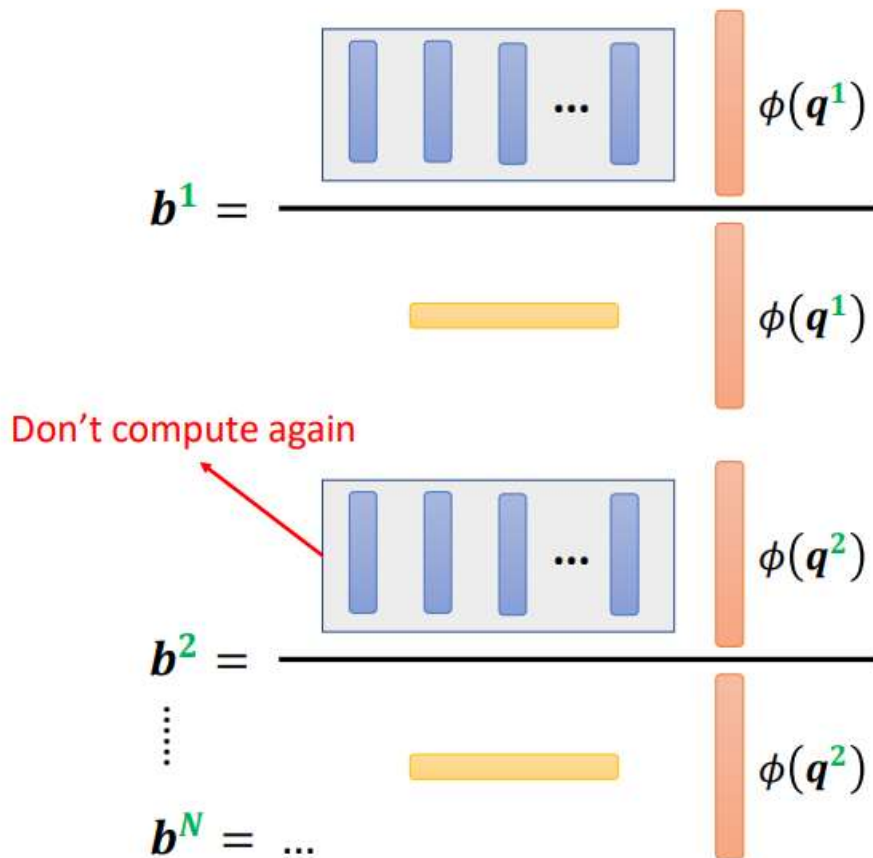
$$b^1 = \sum_{i=1}^N \alpha'_{1,i} v^i = \frac{\sum_{i=1}^N [\phi(q^1) \cdot \phi(k^i)] v^i}{\phi(q^1) \cdot \sum_{j=1}^N \phi(k^j)}$$

$$\sum_{i=1}^N [\phi(q^1) \cdot \phi(k^i)] v^i \quad \phi(q^1) = \begin{bmatrix} q_1^1 \\ q_2^1 \\ \vdots \end{bmatrix} \quad \phi(k^1) = \begin{bmatrix} k_1^1 \\ k_2^1 \\ \vdots \end{bmatrix}$$

M dim

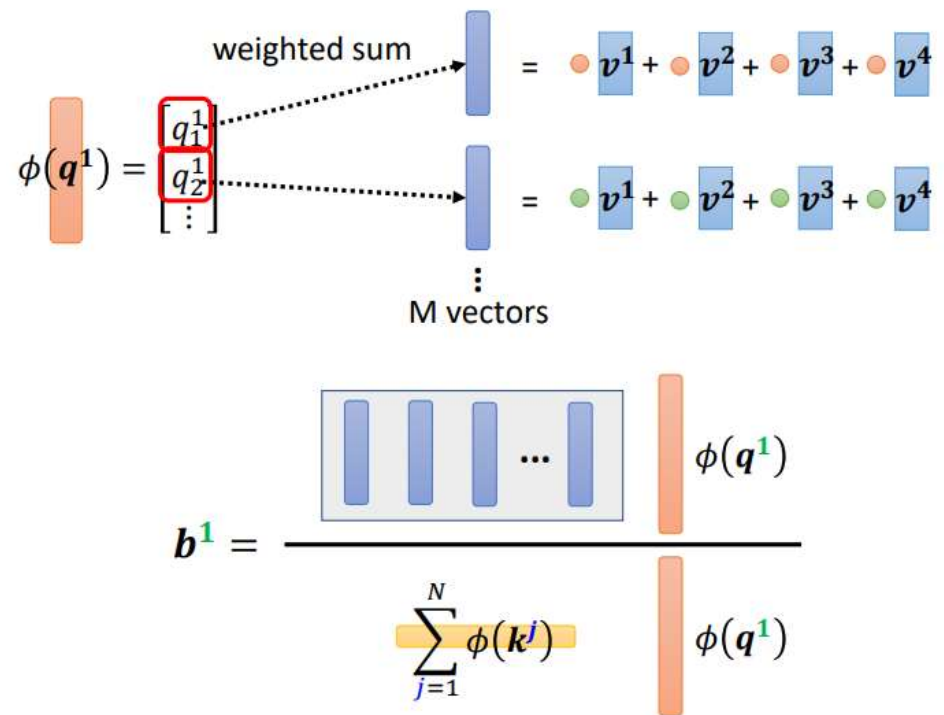
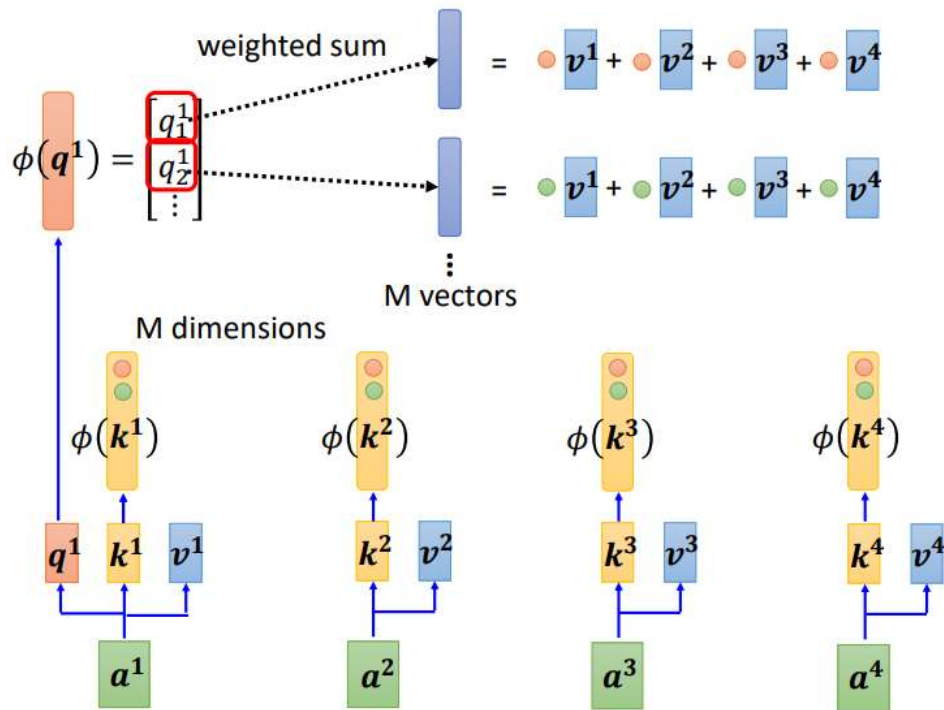
$$= q_1^1 (k_1^1 v^1 + k_1^2 v^2 + \dots) + q_2^1 (k_2^1 v^1 + k_2^2 v^2 + \dots)$$

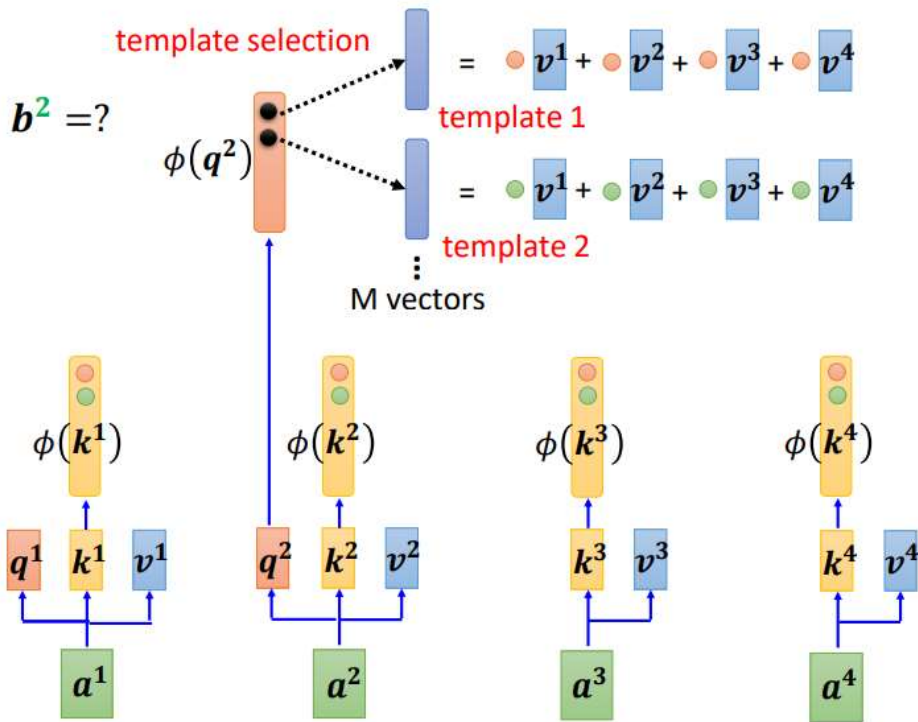




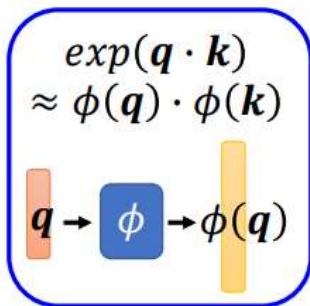
Let's put softmax back ...

End of warning



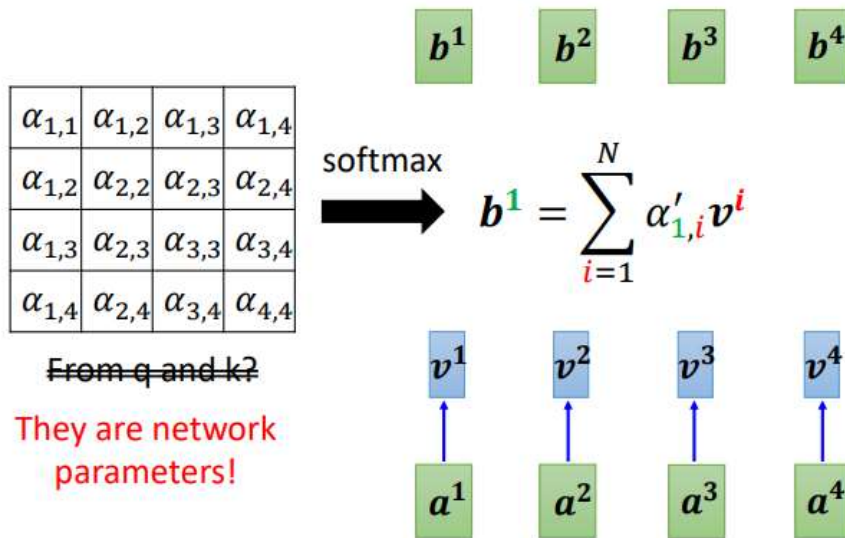


Realization

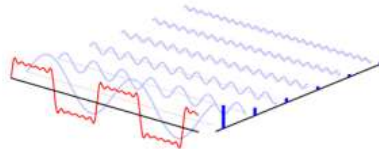


- Efficient attention
<https://arxiv.org/pdf/1812.01243.pdf>
- Linear Transformer
<https://linear-transformers.com/>
- Random Feature Attention
<https://arxiv.org/pdf/2103.02143.pdf>
- Performer
<https://arxiv.org/pdf/2009.14794.pdf>

Do we need q and k to compute attention? Synthesizer!



Attention-free?



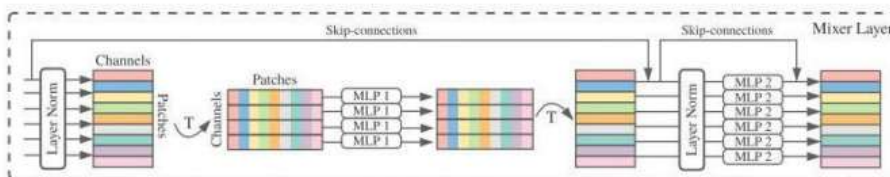
- Fnet: Mixing tokens with fourier transforms

<https://arxiv.org/abs/2105.03824>

- Pay Attention to MLPs <https://arxiv.org/abs/2105.08050>

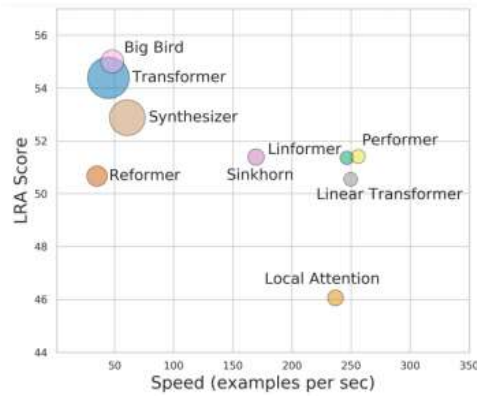
- MLP-Mixer: An all-MLP Architecture for Vision

<https://arxiv.org/abs/2105.01601>



Summary

- Human knowledge
 - Local Attention, Big Bird
- Clustering
 - Reformer
- Learnable Pattern
 - Sinkhorn
- Representative key
 - Linformer
- k,q first \rightarrow v,k first
 - Linear Transformer, Performer
- New framework
 - Synthesizer



課程網頁 (<https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php>).

tags: 2022 李宏毅_機器學習