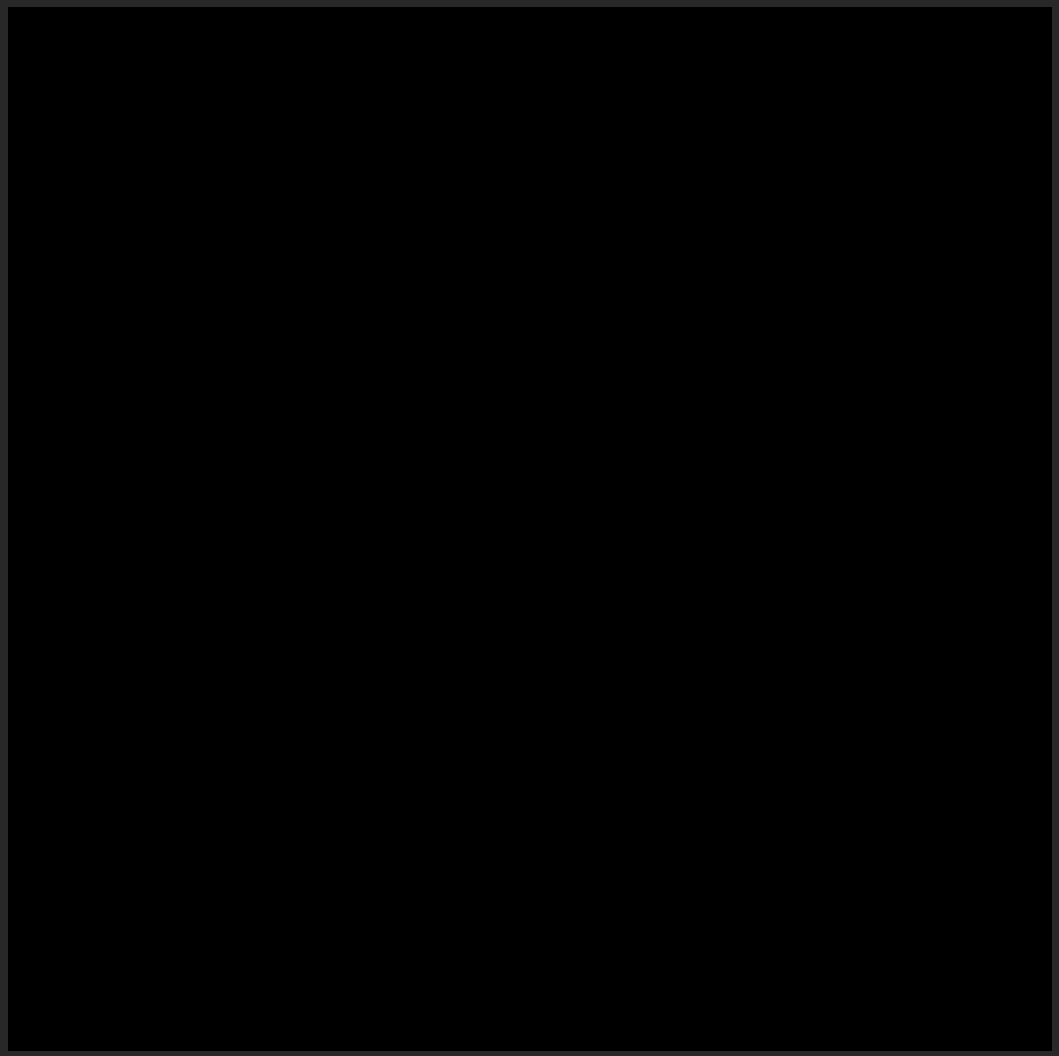
YONA：You Only Need Attention

~~YONA : You Only Need Attention~~

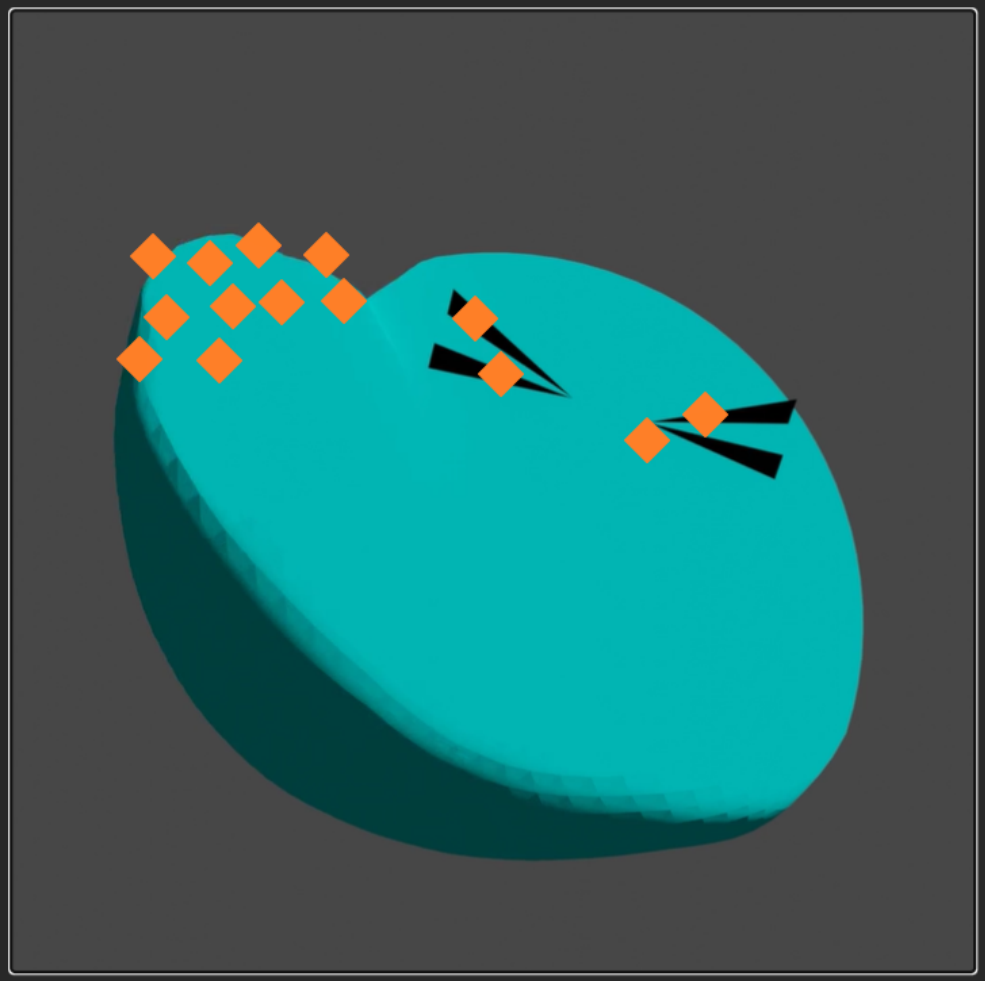
Attention is all you need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin



什麼是注意力吶？

注意力就是給予部分資料更大的權重



什麼是注意力吶？

注意力就是給予部分資料更大的權重

設定權重的方式百百種

Self Attention 就是其中一種方法

超級難懂

超級有效

優勢：vs CNN

不同於 CNN 固定大小的感受野

Self Attention 的感受野是全域級的

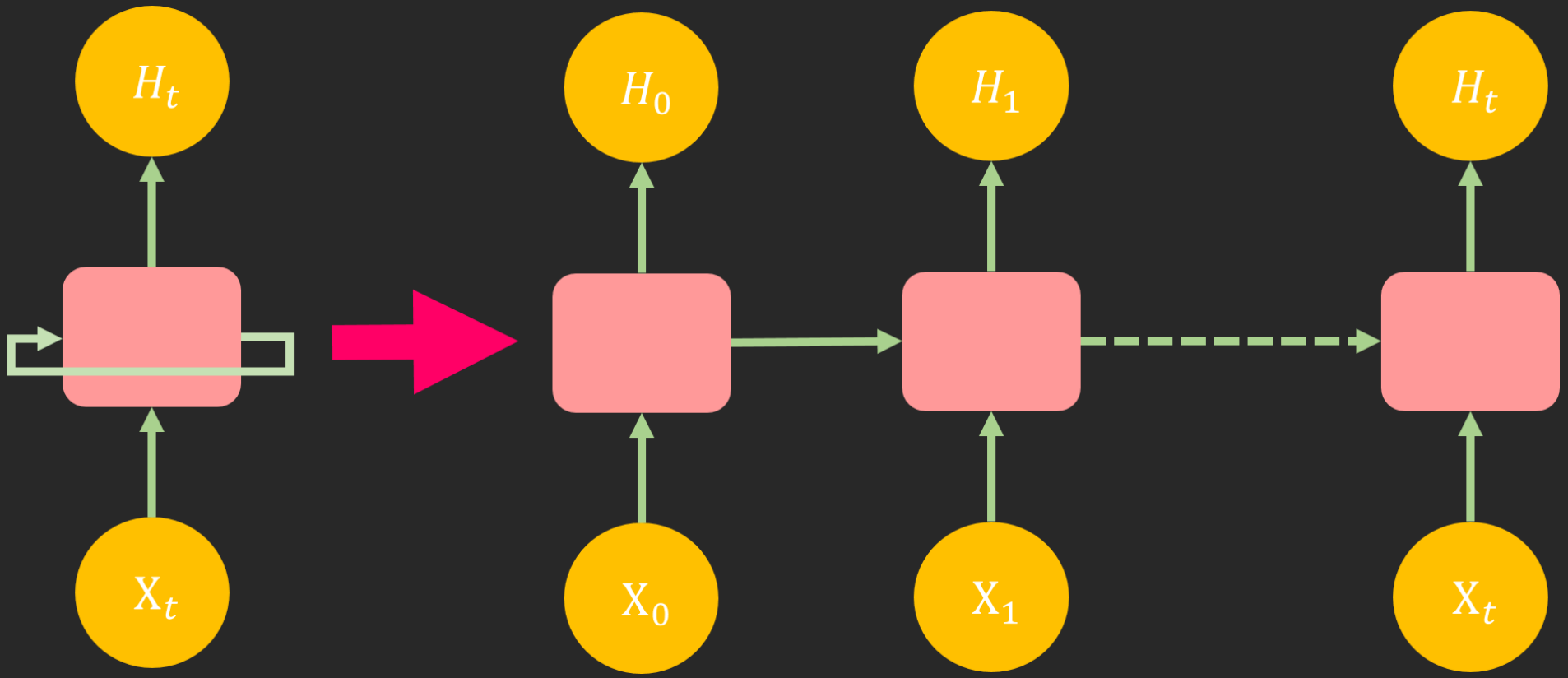
不管身在何處，總能牽起與你的那條紅線(嘔

優勢：vs RNN

RNN 的問題有兩個

1. 無法平行化處理，有 L 個輸入就相當於要通過 L 層。
2. 有可能會遺忘很久以前的輸入，強行留下這些記憶又會導致梯度爆炸。

優勢：vs RNN



優勢：vs RNN

而 Self Attention 具有一個
超巨大的感受野可以同時處理這些輸入，

既可平行化處理又沒有遺忘的問題

技術晚點再說，先講發展

最開始應用在機器翻譯的任務上，並推出了大名鼎鼎的

Transformer

效果絕佳、一戰成名
以更少的訓練成本擊敗了 RNN 系列模型

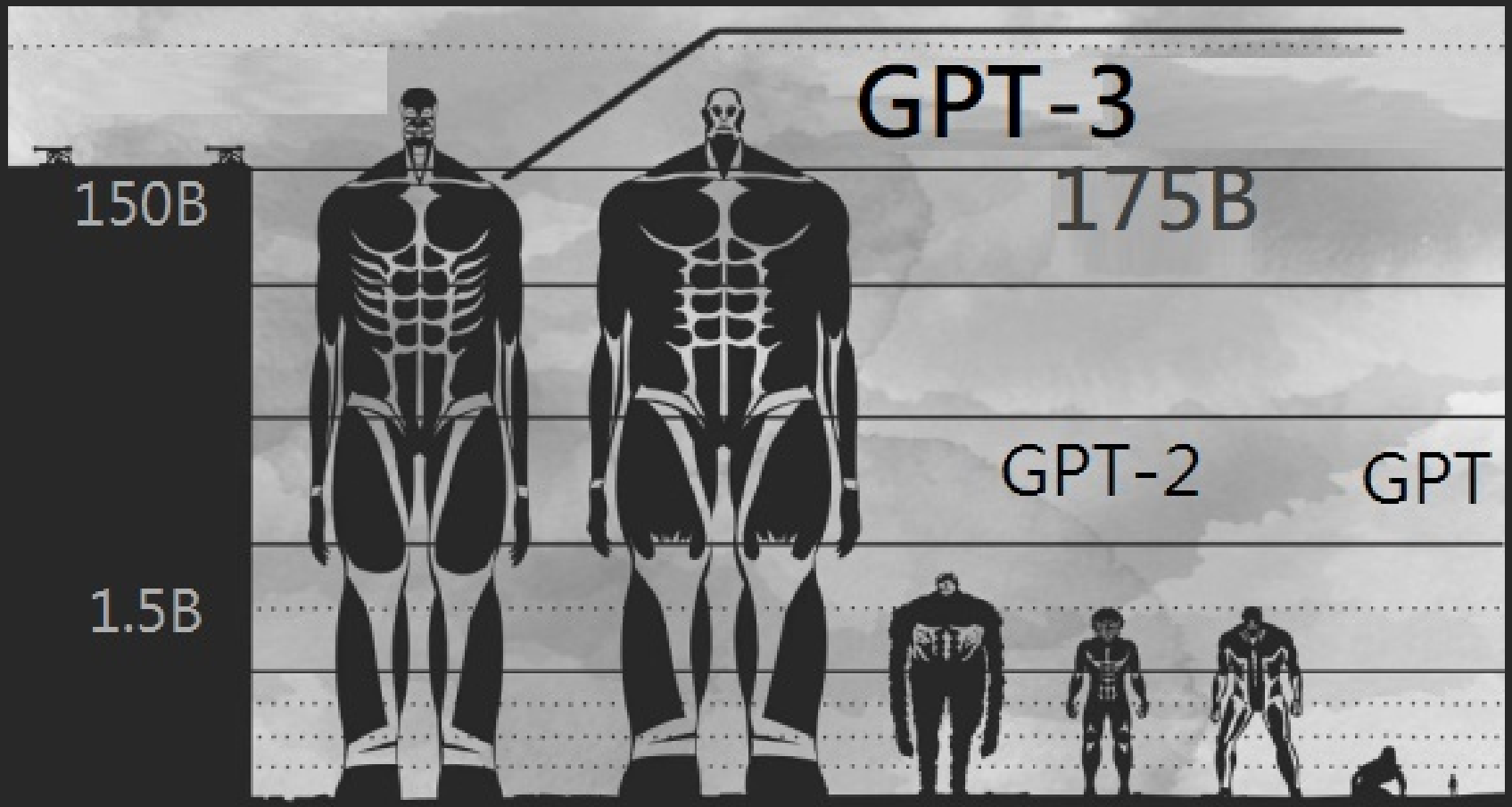
Transformer 切對半

Transformer 是由 Encoder(特徵抽取器) 與 Decoder(自回歸模型) 組成的

後來發展成兩派知名語言模型

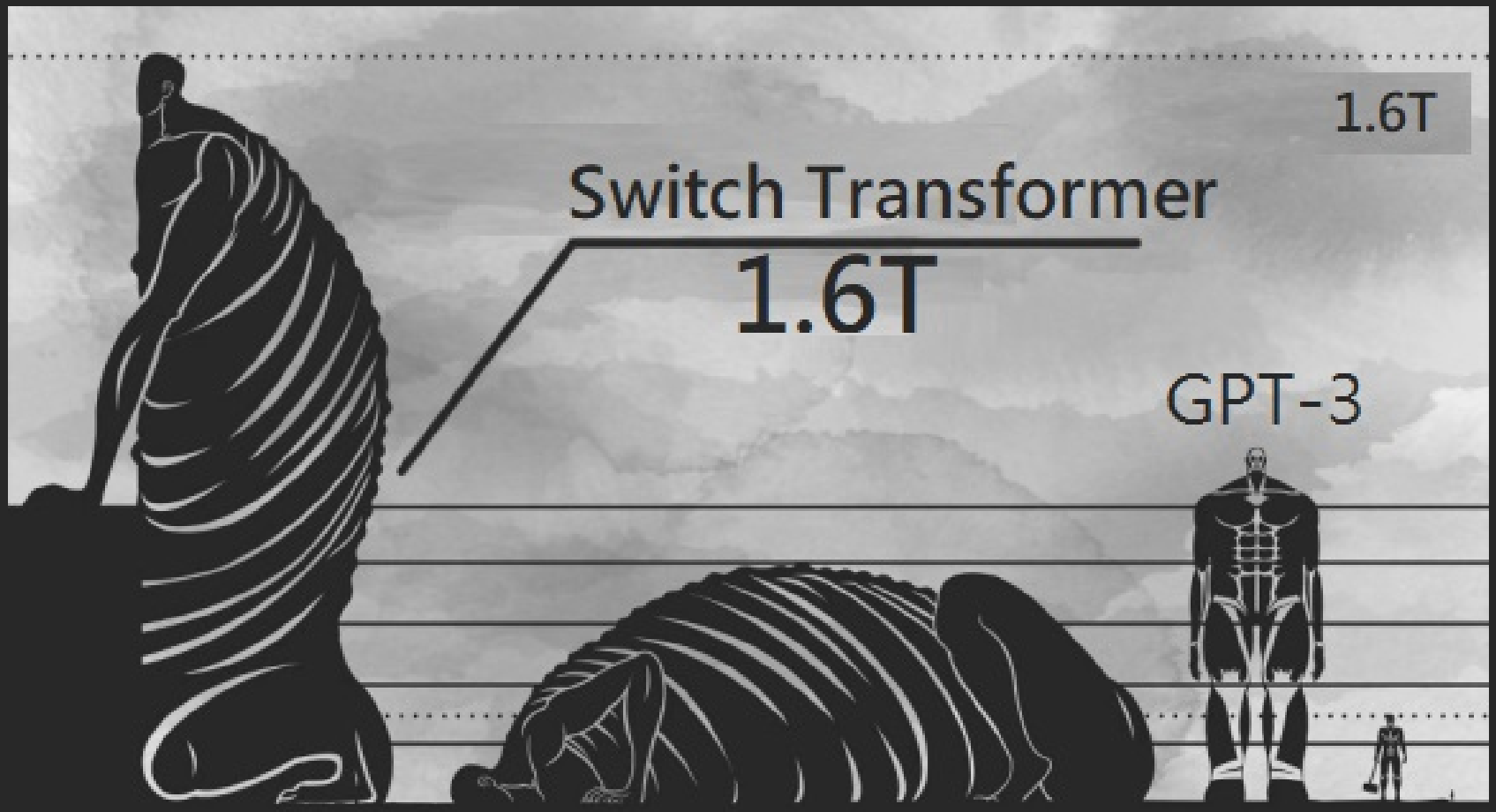
BERT (only Encoder)
與
GPT (only Decoder)

錢能解決的都不是問題



GPT 越長越大

問題是沒錢



NLP(自然語言處理) 已經成為軍備競賽

只要你有錢，處處都能 Self Attention

不只用在 NLP，最近在影像上也開始發展

影像分類可以用 Vision Transformer
影像分割可以用 TransUNet

物件偵測的 DETR 與 Context-Transformer

影像生成的 Taming Transformers、TransGAN、ImageGPT、DALL·E

沒錢只能 QAQ...？

便宜的替代方案

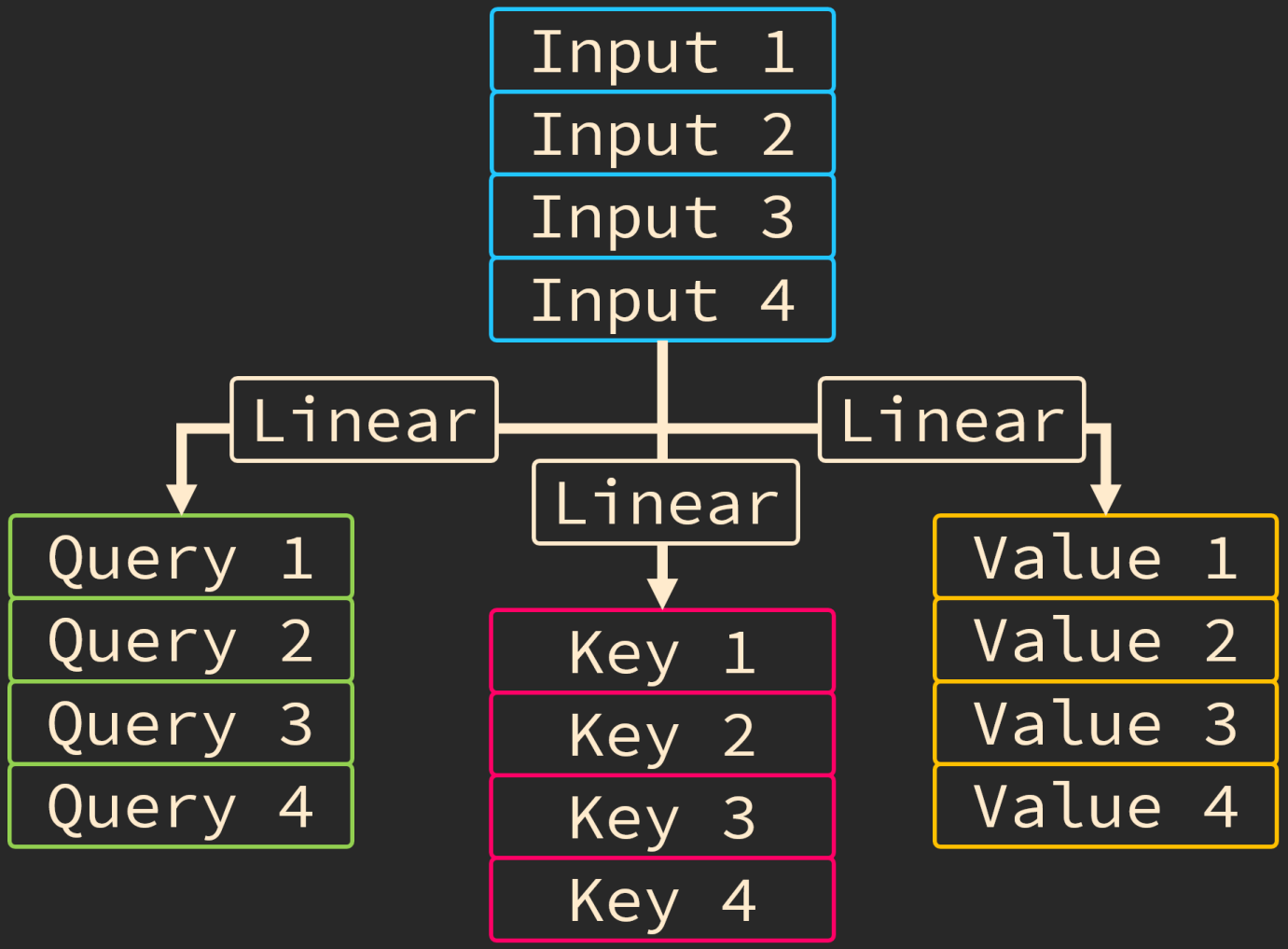
1. Reformer
2. Linformer
3. Performer
4. Lambda layer

回到正題

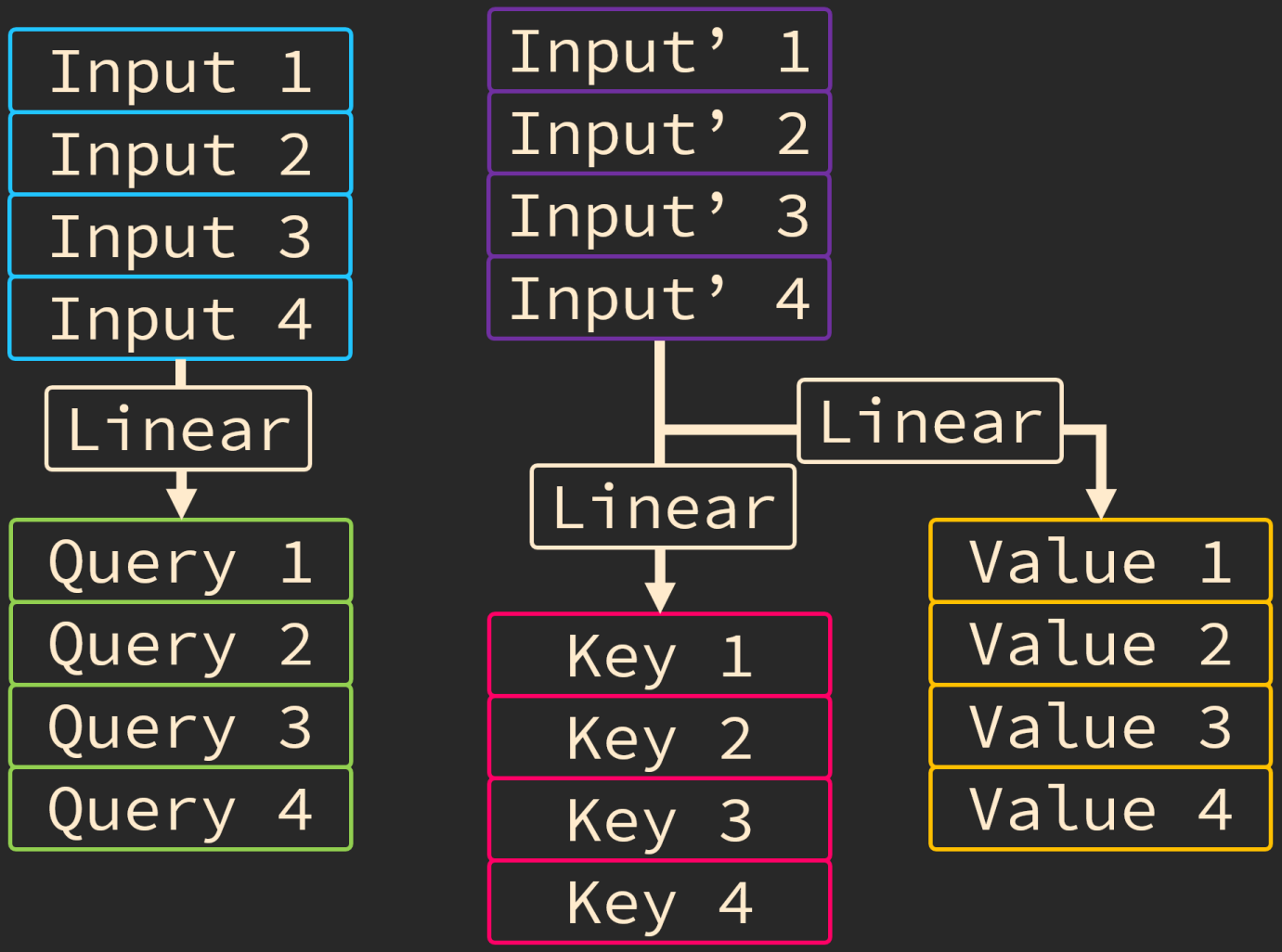
「Multi Head」Self Attention

1. Self Attention
2. Multi Head
3. Positional Encoding

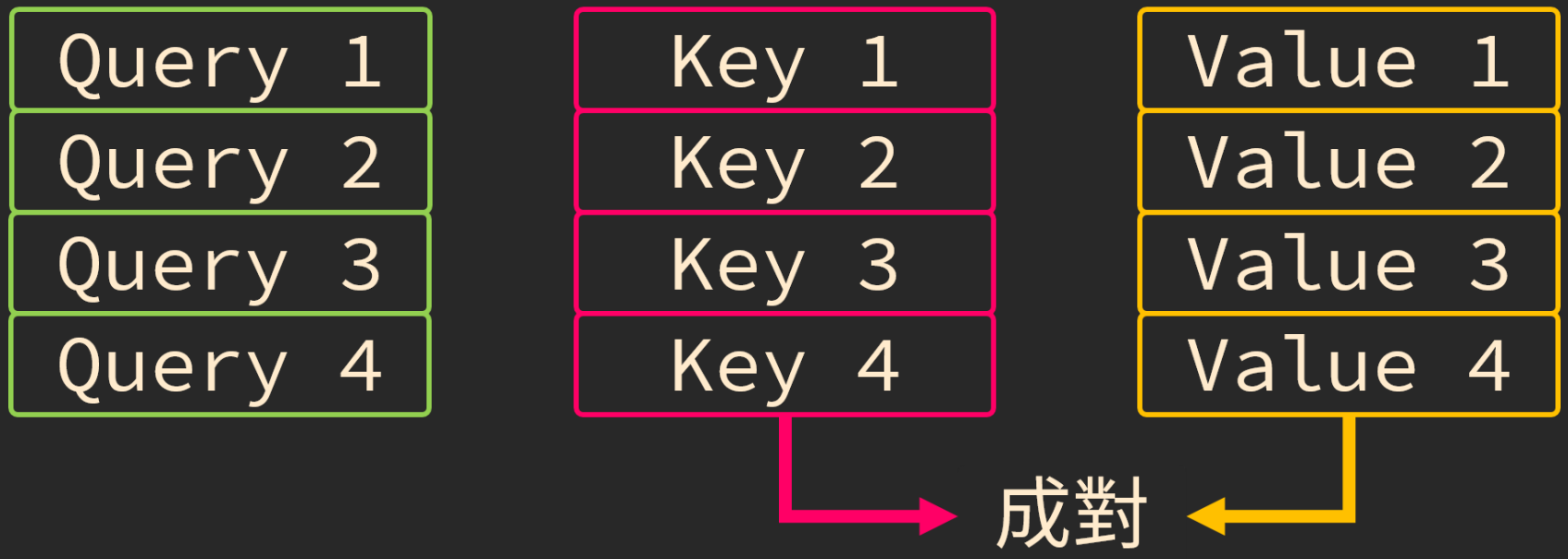
情況1：
輸入一組 Feature Vector



情況2：
輸入兩組 Feature Vector

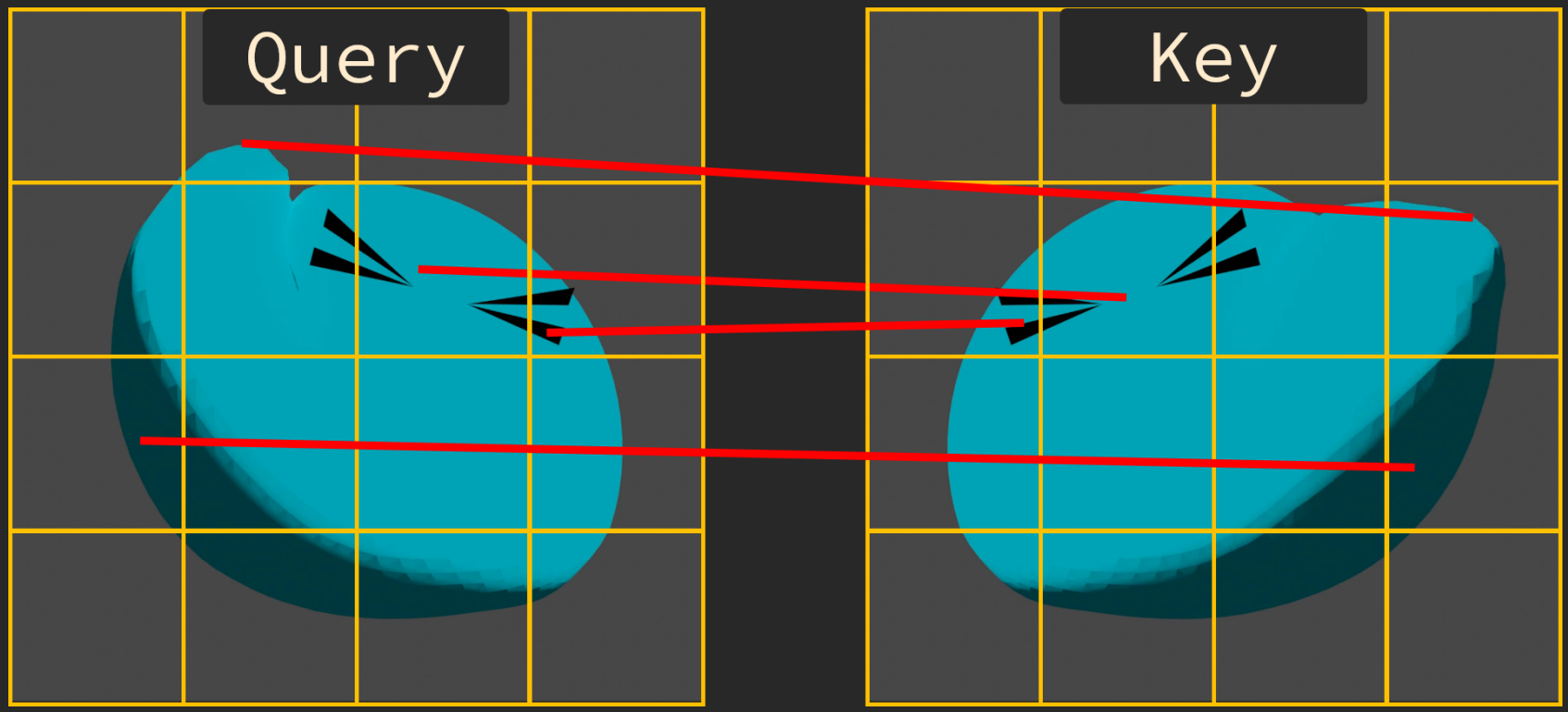


Query、Key 與 Value

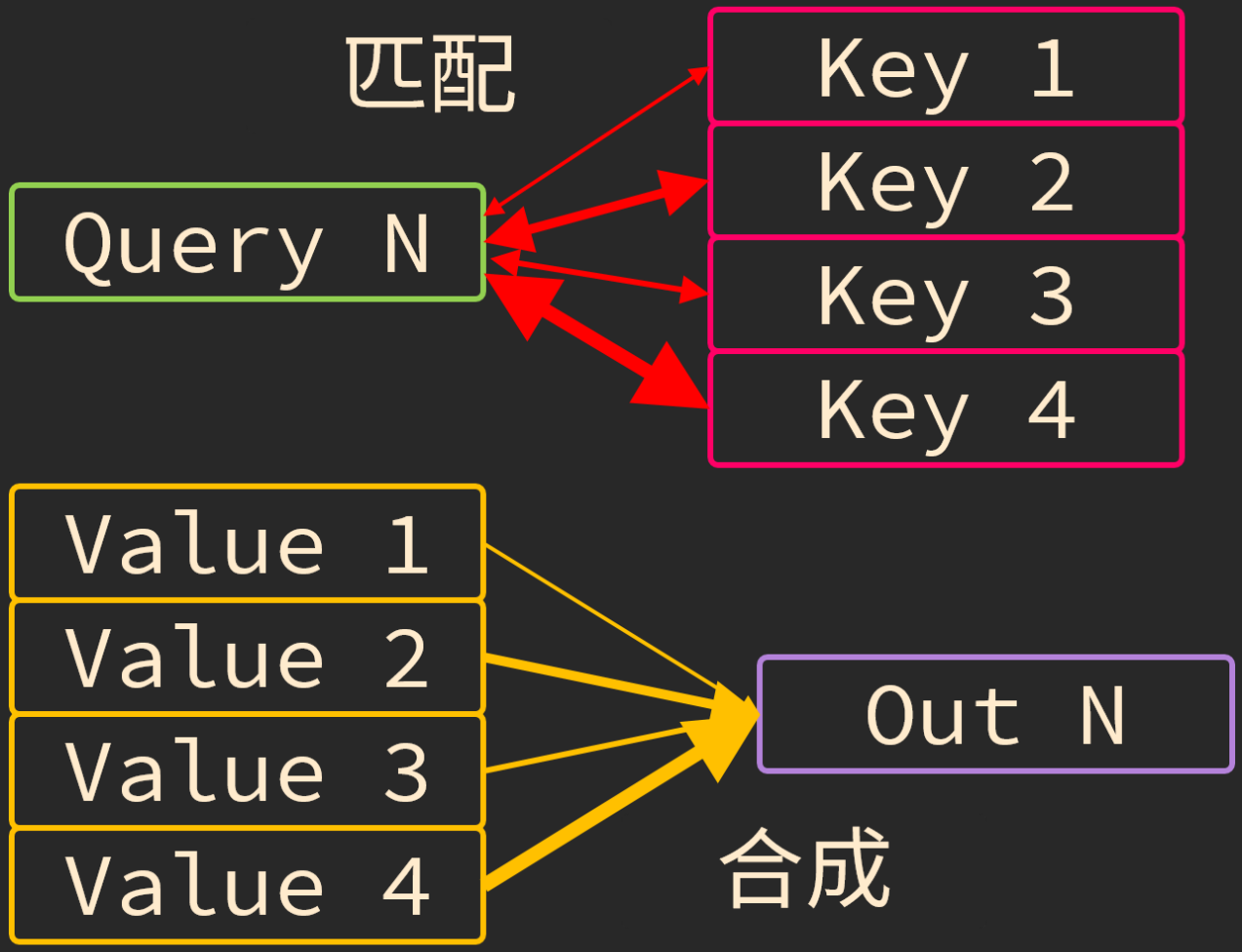


Query 與 Key 大多數都源於同樣的資料，所以稱為 Self

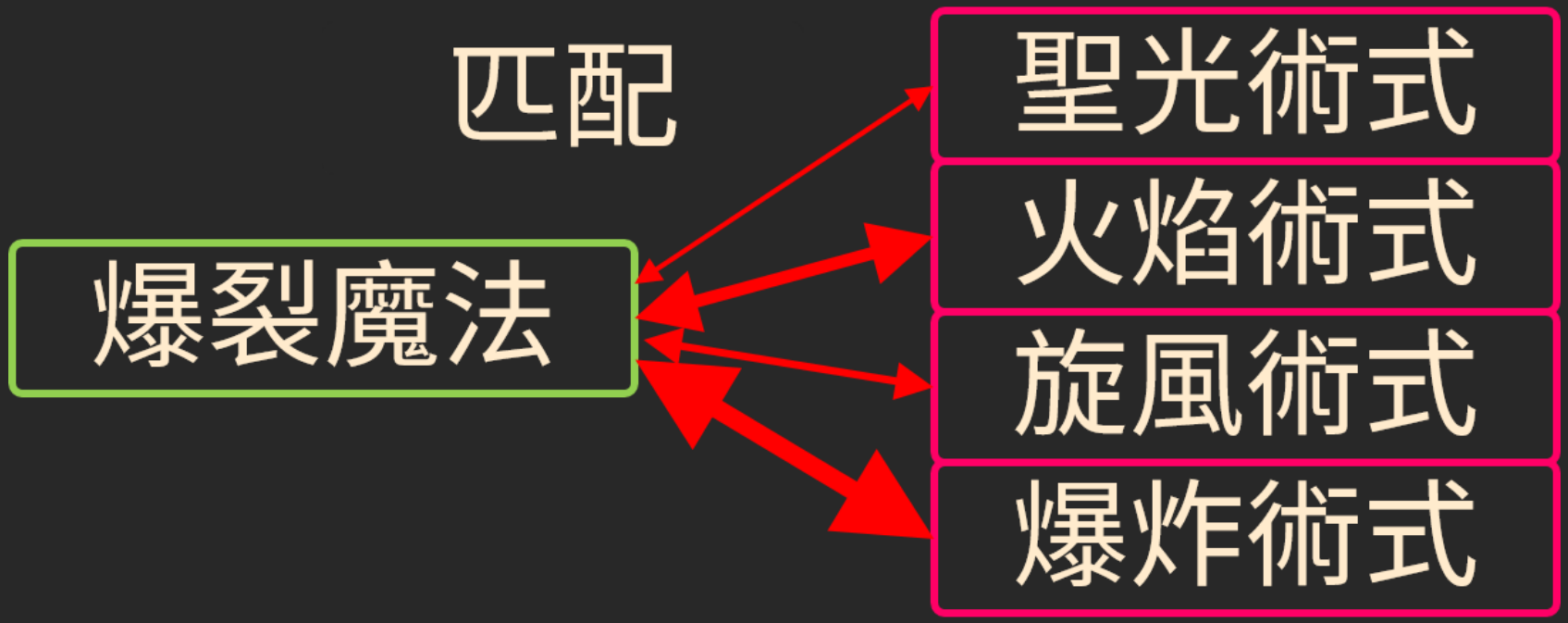
配對每個 Query 與 Key 獲得兩者間的相似度



再透過相似度
合成每個 Key 對應的 Value



換個說法就是



Explosion！！！

炫光傷害
燃燒傷害
旋風傷害
爆炸傷害

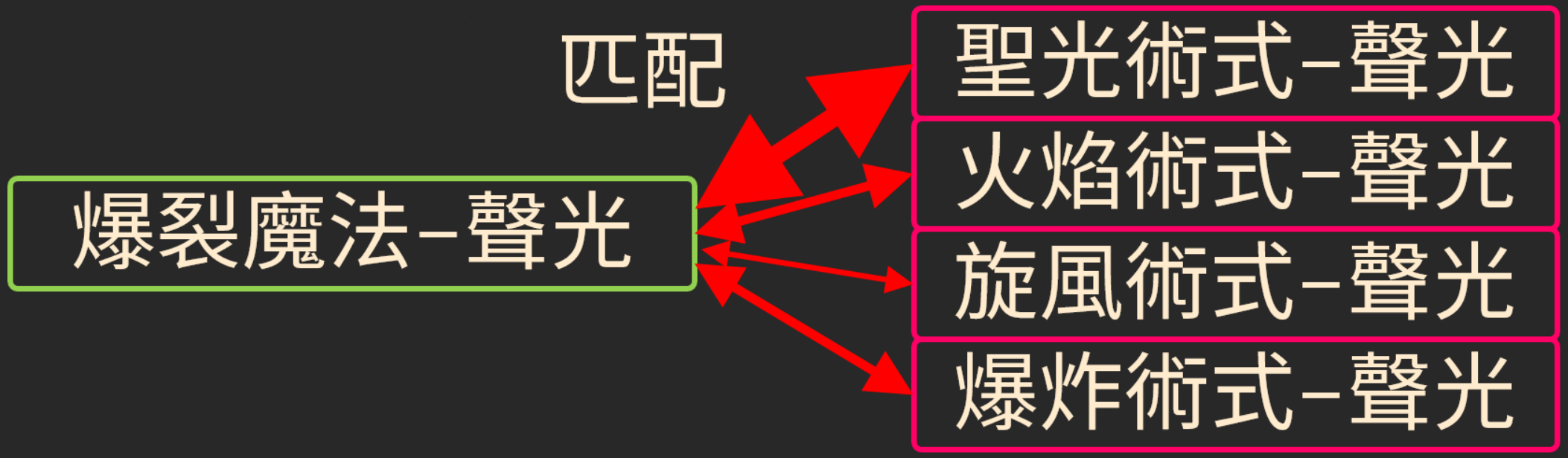
少了點東西

但這只是很厲害的爆炸

不是中二破表的爆裂魔法

所以要再製作出爆裂魔法的聲光效果

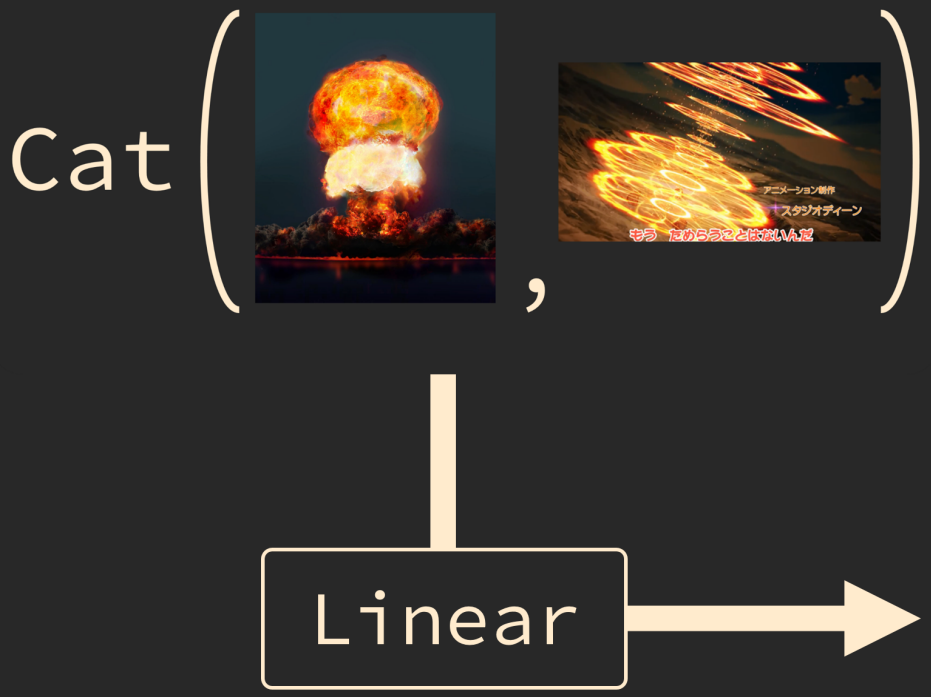
Explosion-聲光篇

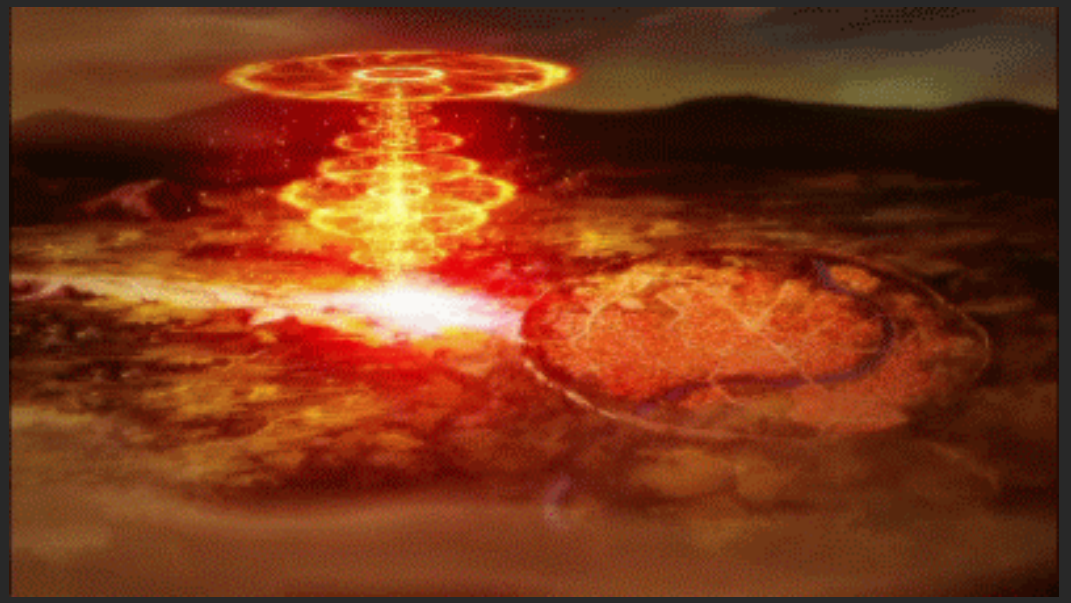


Explosion-聲光篇

這樣就獲得了絢麗十足的特效了

混合傷害與特效





真正的，Explosion！！！

Multi Head == Multi Feature

剛剛的例子就是 2 head self attention

因為，萬事都不只有一個面貌

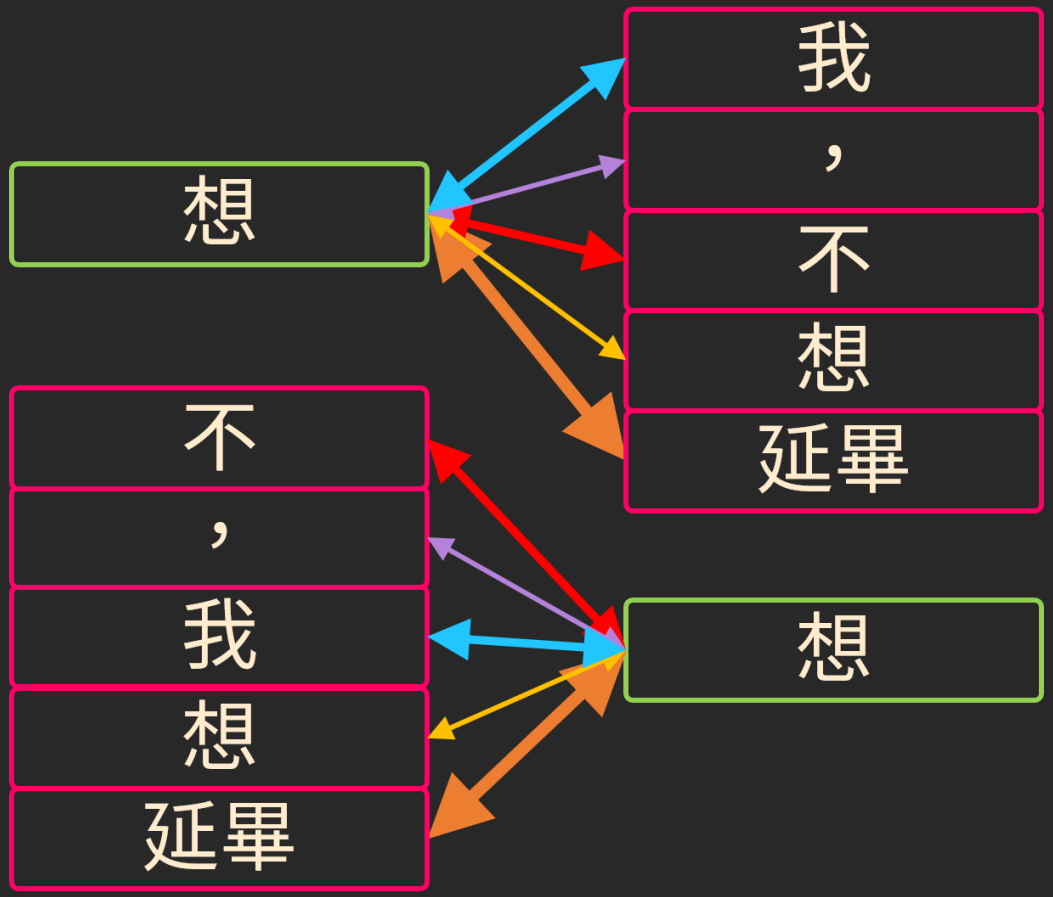
(這不是在說人生道理)

如果只去注意單一面向就會產生偏見

(這依然不是在說人生道理)

為了解決這問題，Multi Head 便從多種角度去觀察同一件事

我，不想延畢 == 不，我想延畢...？



在沒有外加資訊的情況下

前面所使用的計算方式無法判斷不同位置有什麼不一樣

Position Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{MaxLength^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{MaxLength^{\frac{2i}{d_{model}}}}\right)$$

Position Encoding

position 1 : 0

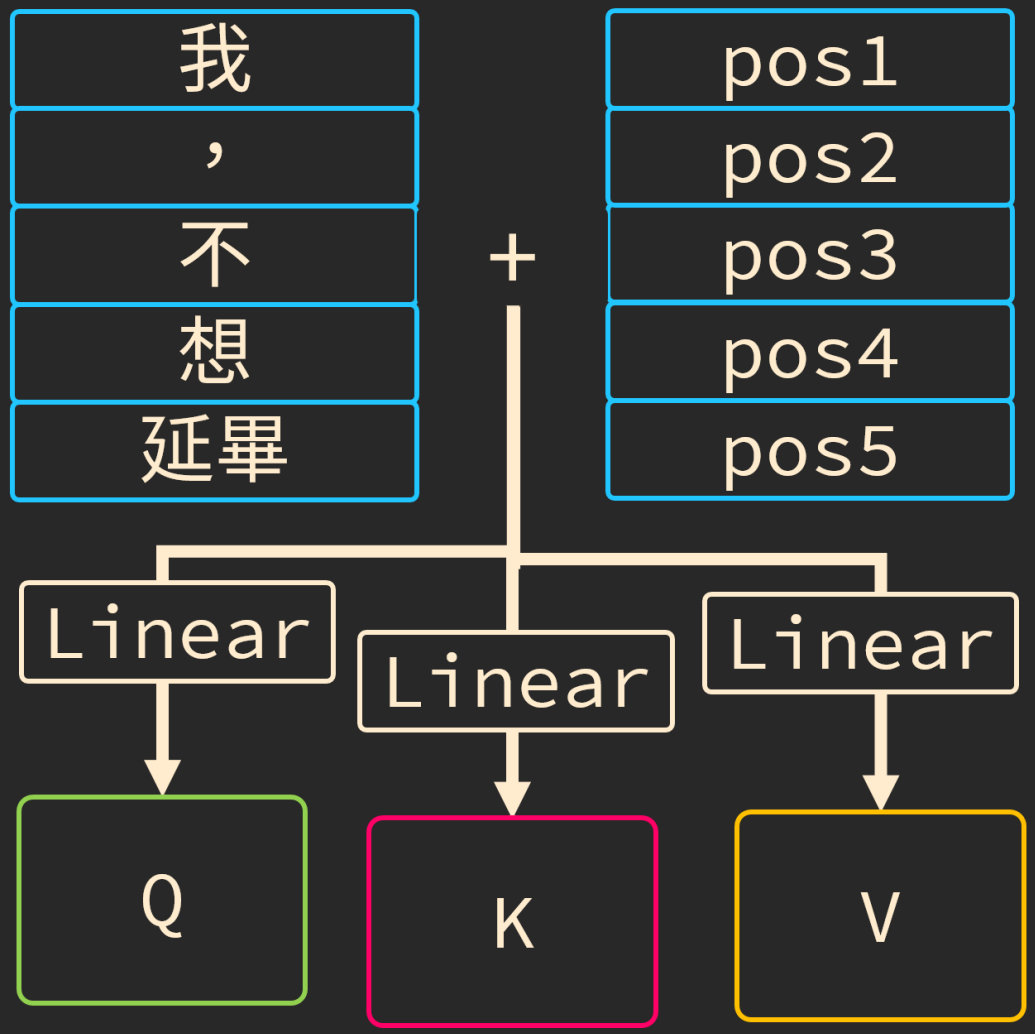
position 2 : 0

d_{model} : 256

max length : 10000

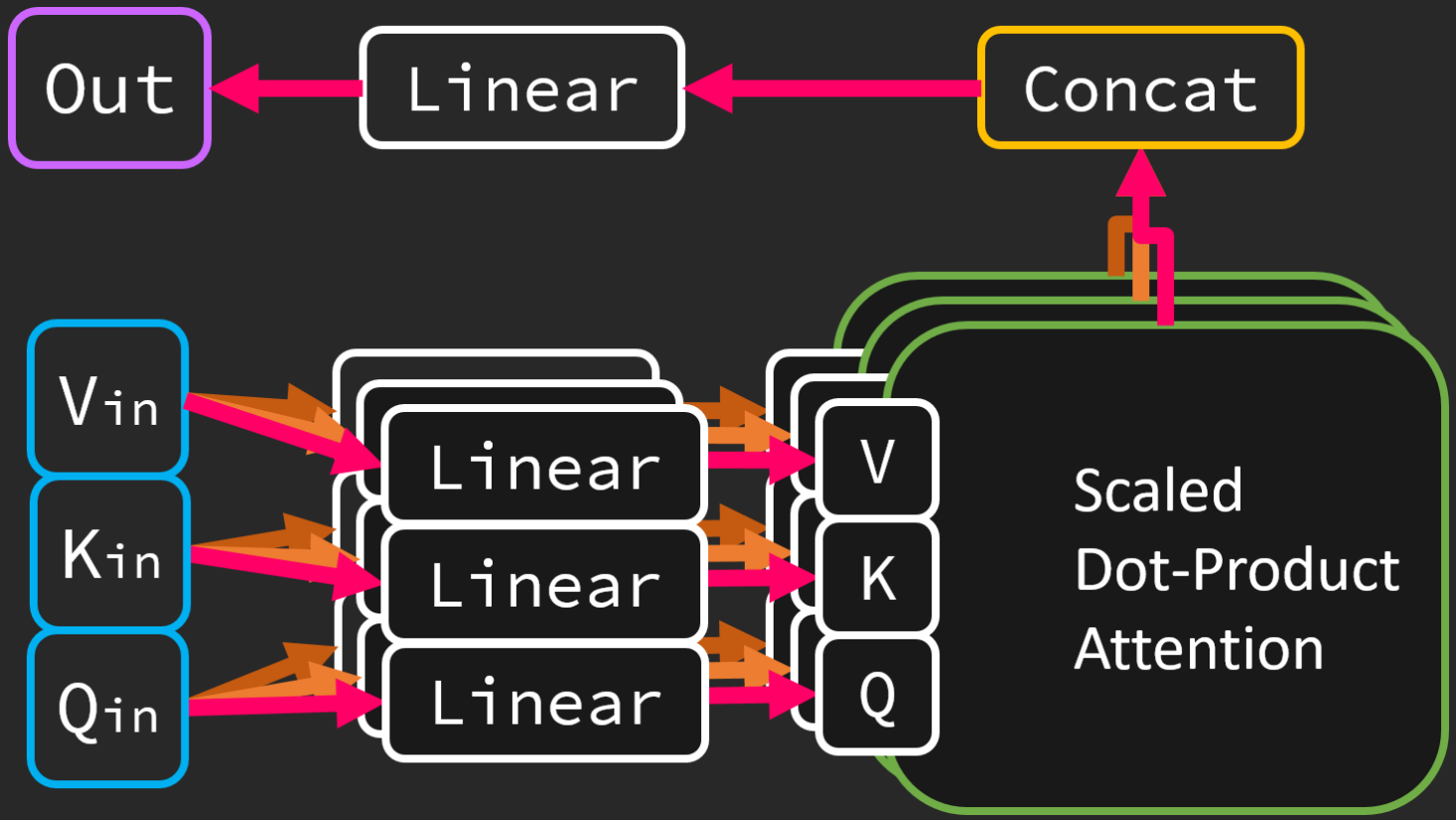
calc

直接與輸入相加做使用



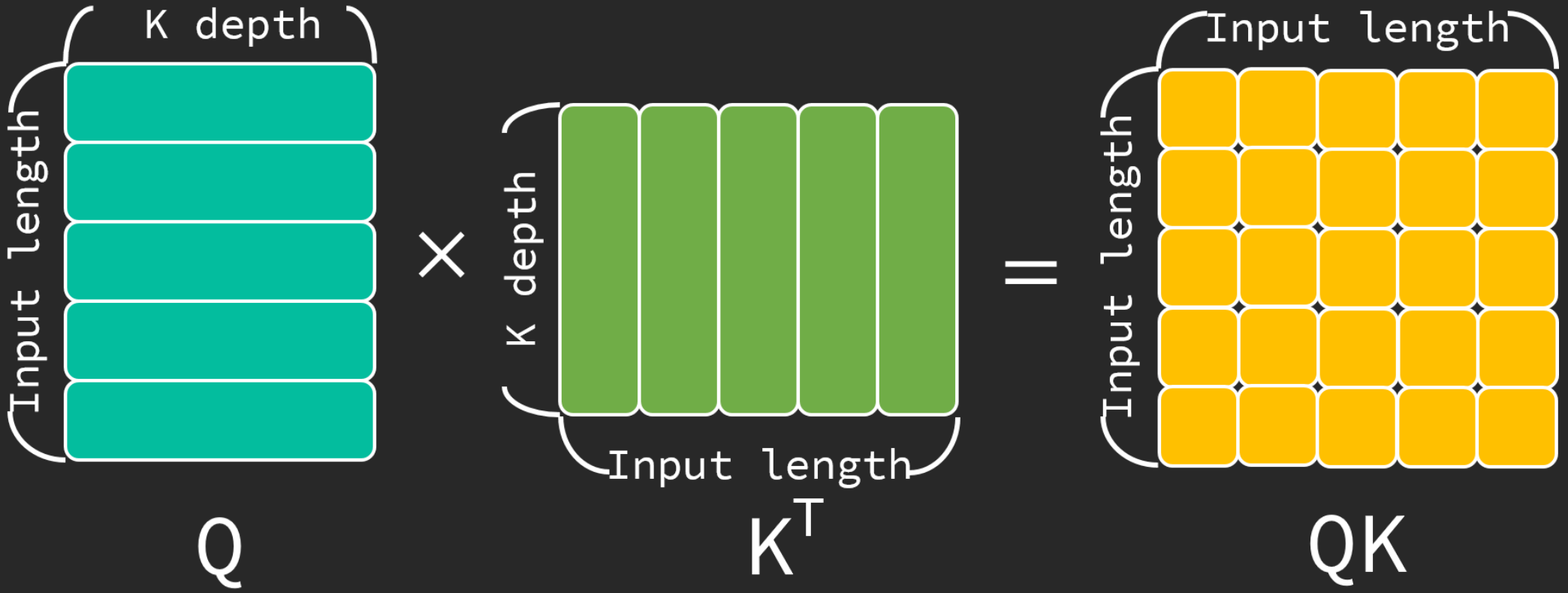
統整

Multi Head Self Attention



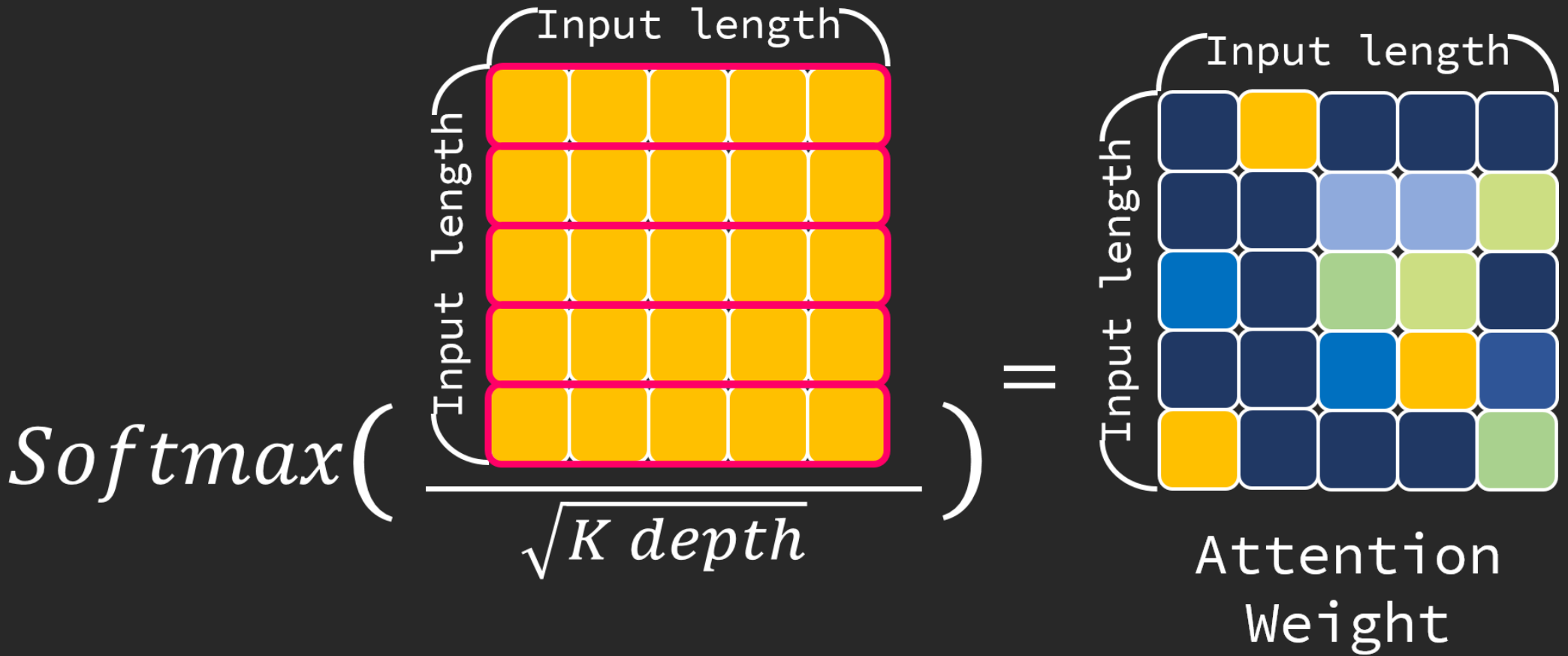
這邊的 Scale Dot Product Attention 就是前面說的 Self Attention

Multi Head Self Attention



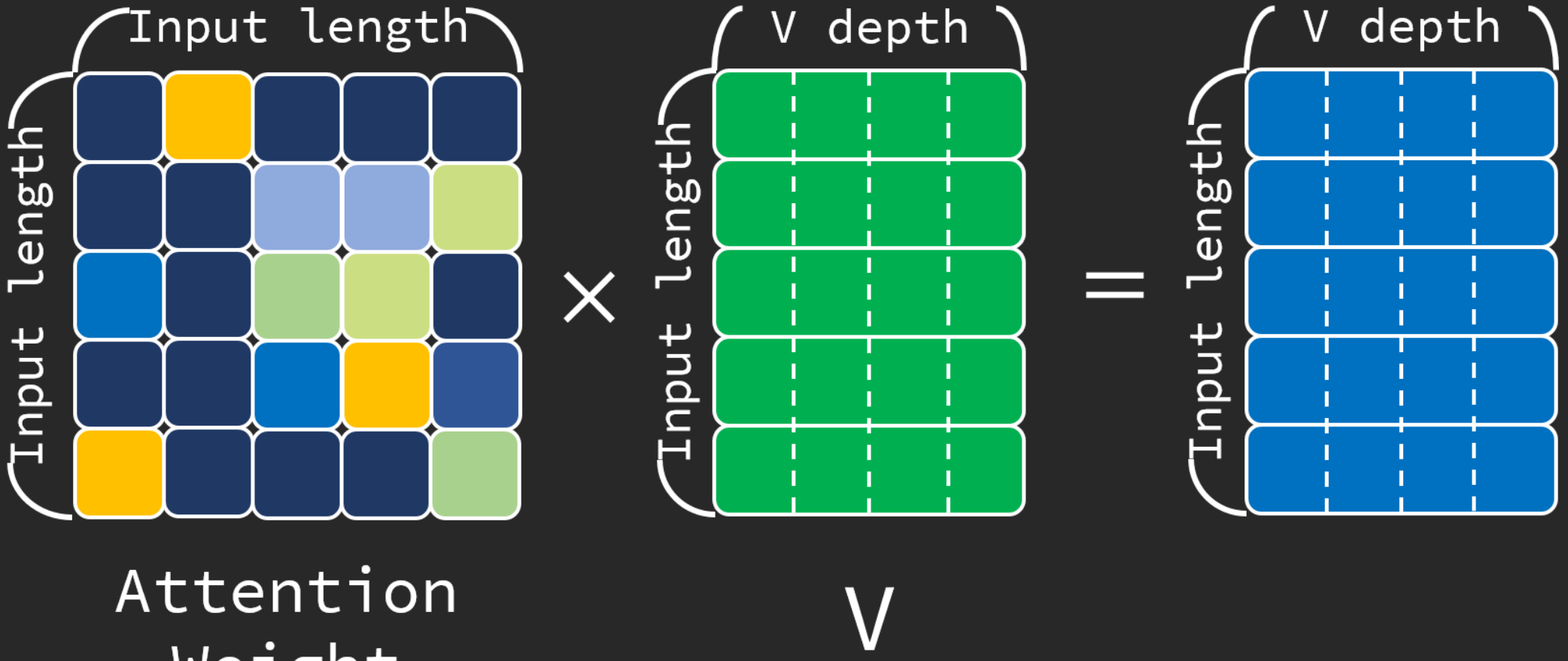
每個 Q K 都利用內積計算相似度

Multi Head Self Attention



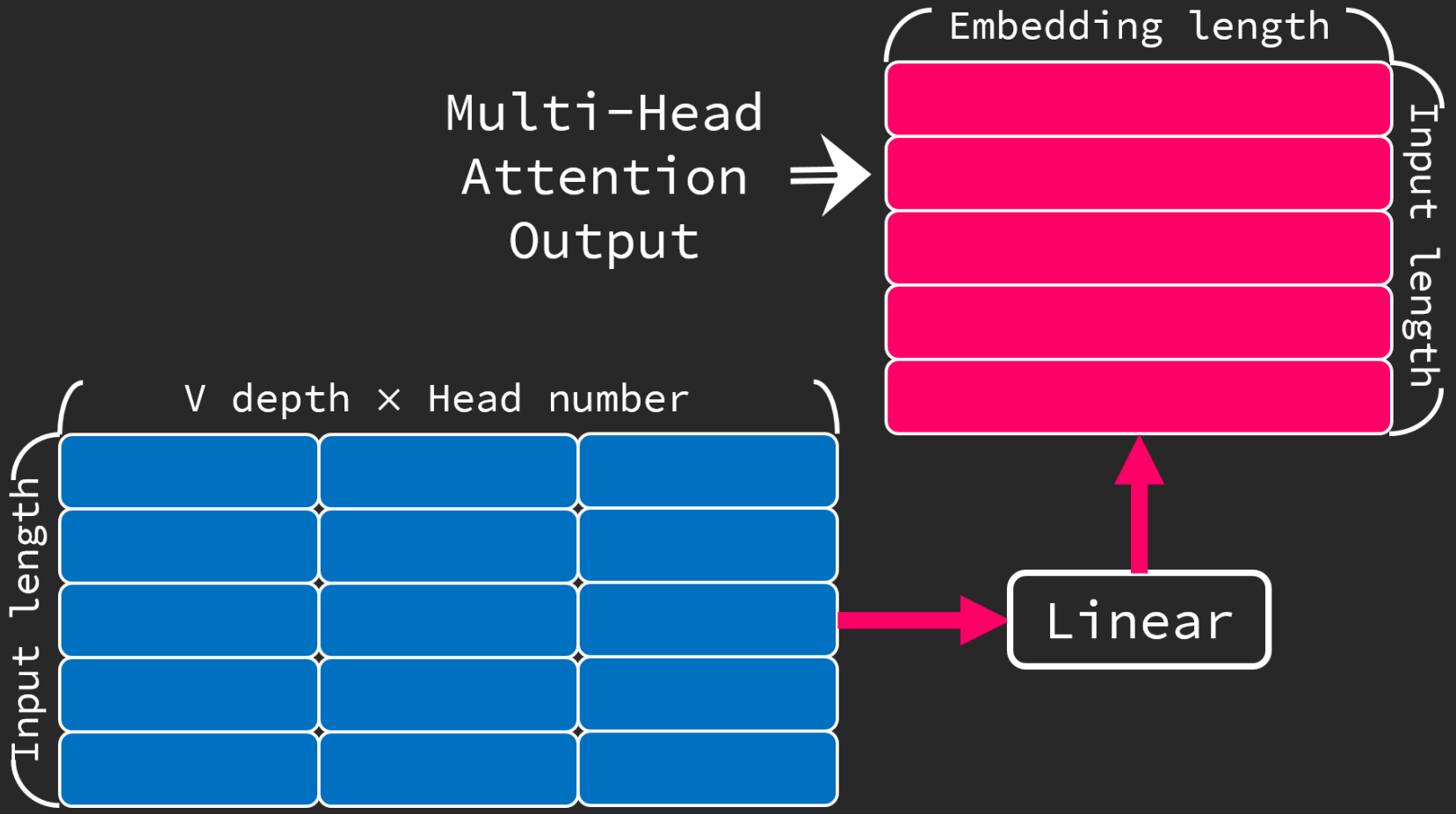
再透過 Softmax 正規化成注意力權重

Multi Head Self Attention



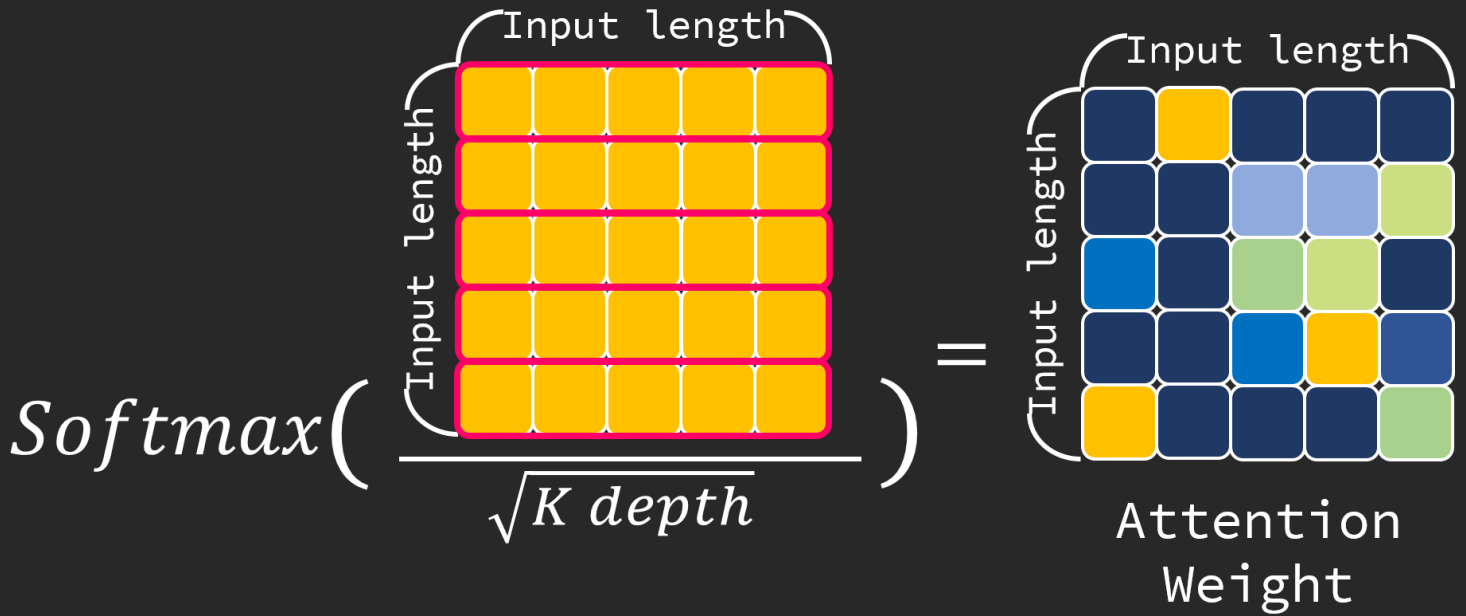
利用注意力權重與 V 合成 Single Head Output

Multi Head Self Attention



將每個 Head Output 融合為輸出

缺點：記憶體複雜度是 N^2



在計算相似度與注意力權重時要耗費巨量的記憶體，因此最初的 Self Attention 處理高維度問題時非常昂貴(物理上)

延伸閱讀

1. Tensorflow 官方手把手教你刻 Transformer
2. OpenAI Researcher 寫的 Transformer 系列講解
3. 很難看的 Transformer 講解
4. Transformer 各種高效變形