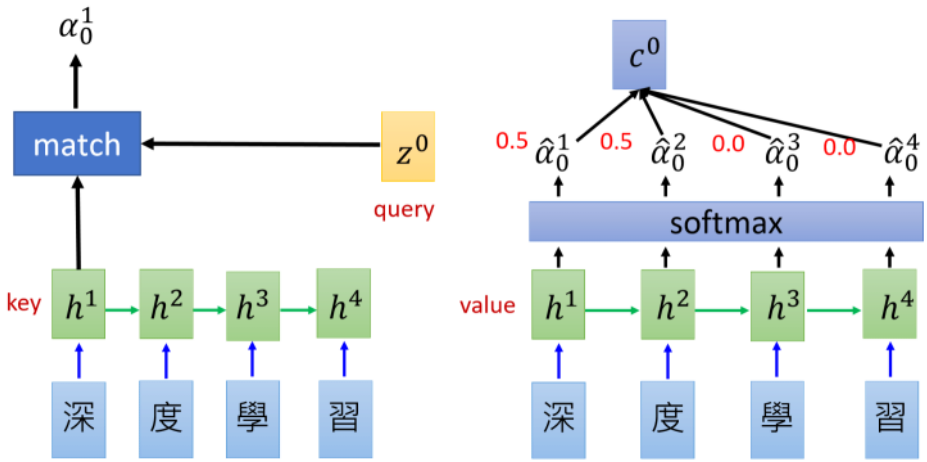
ADL—Transformer

2019年4月9日　　上午 09:16

一連串資料，長度不一樣
用不同方式合起來? 用model去學，把各字embedding合起來? (CNN)
RNN依照時間順序? (temporal information)
模型結構考慮input長相
-

- RNN
  - 可以學出不同長度的represnetation，最後尾端的長度vector是固定的，因此可以拿來input value
  - 用RNN把資訊agrregate
  - 缺點
    - 按照時間順序吃進來的，所以要計算其中一個時間點的vector一定要等前面完成，所以如果sentence長，要花比較久時間
    - 每個時間點都需要前面時間點資訊
    - RNN假設information可以保留到最後包含整句話。但其實沒有明顯捕捉此字跟前一字是否有關係
      - long-dependency (代名詞關聯)
        - 可能會捕捉，只是假設可以，但並沒有明顯說真的會
      - short-dependency
- CNN
  - word embedding放進去conv，要看左右多少，這個字可能會跟左右相關=>windows size=3
  - 在丟進去FC
  - 沒有時間順序相關，每一個時間點都是相同事情，都考慮旁邊的字，要考慮多旁邊可以依據filter設定
  - 優點
    - 可以平行畫
    - 沒有depend on前面算出來的資訊
    - CNN比較快
    - 也有稍微考慮dependency資訊，filter=3 考慮左右兩個
    - filter大一點可以考慮更多左右字，考慮是否有直接關聯
    - target input跟周圍input有某種程度關聯
    - 但是只能考慮local dependency
    - filter越大，FC要越深。
  - 缺點
    - long-distance dependency不可知
- Attention
  - 可以藉由加上attention考慮距離很遠的資訊

- decoder吐字的時候，encoder最一開始的字可能已經忘了，再加上不同吐字的時候要看的input恐怕不同
- 要明確找出相關的information，可以產生出正確的字
- 使可以允許decoding的時候input所有hidden state
- 可以access每個input的hidden state
- 想要取代RNN，每個時間點都可以直接學怎麼把資訊合起來就好
- Dot-Product Attention
  - 輸入query、key、value
  - 產生字的時候會去看encode出來的dot-product是高還是低
  - 輸出這個時間點的vector
  - 每一個key跟value都是vector=>matrix
  - 前半部是alpha:
  - $$A(q,K,V)=\sum_i \frac{\exp(q\cdot k_i)}{\sum_j \exp(q\cdot k_j)} v_i$$

    Inner product of query and corresponding key
    - Query q is a d_k-dim vector
    - Key k is a d_k-dim vector
    - Value v is a d_v-dim vector
  - 跟query關係越高機率越高，要產生的vector，要去跟所有hidden state計算出來
  - 再去normalize
  - 在去程上每個字的hidden state=>value
  - query跟key的關係
  - match就是dot-product
  - 算出四個alpha去經過ｓｏｆｔｍａｘ
  - 

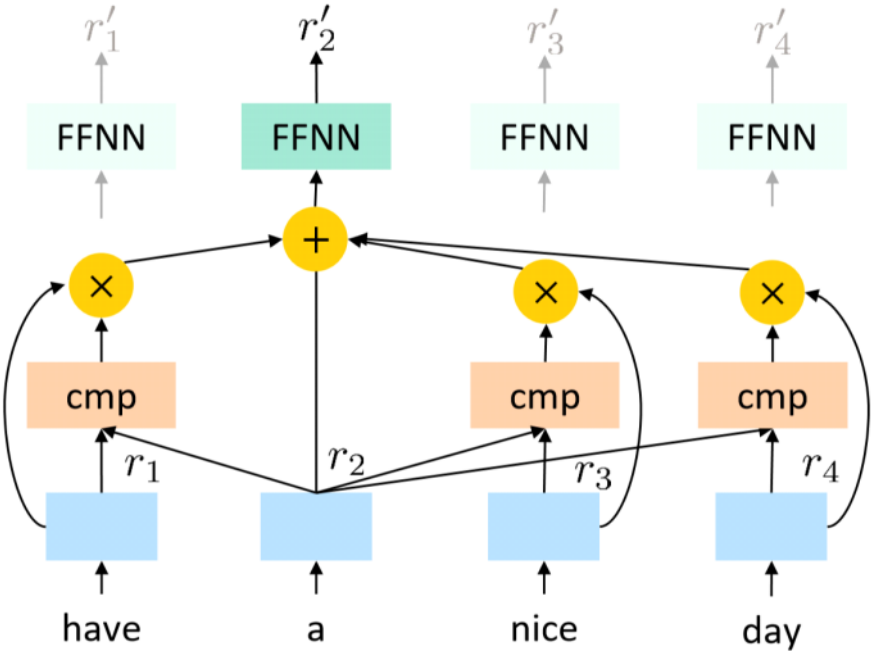
  - key跟query的相關程度=>alpha
  - 這邊的key跟value依樣=>weighted-sum的對象
  - 假設有多個query:

$$A(Q,K,V) = \text{softmax}(QK^T)V$$

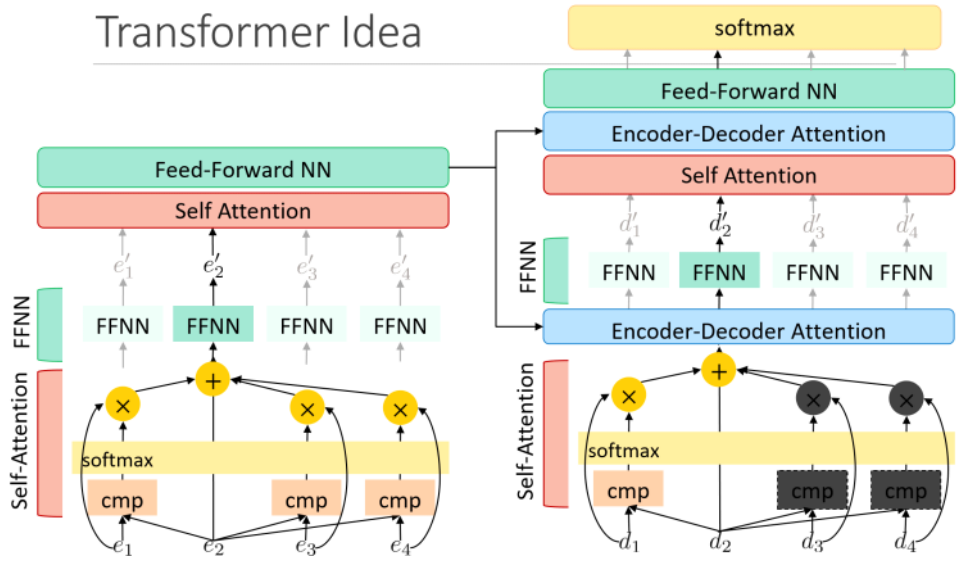
$$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$$

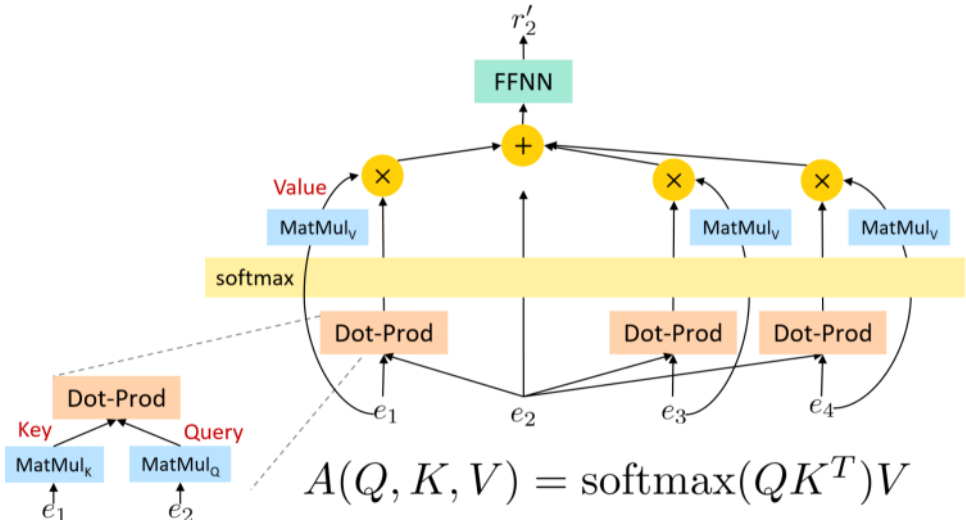
- 

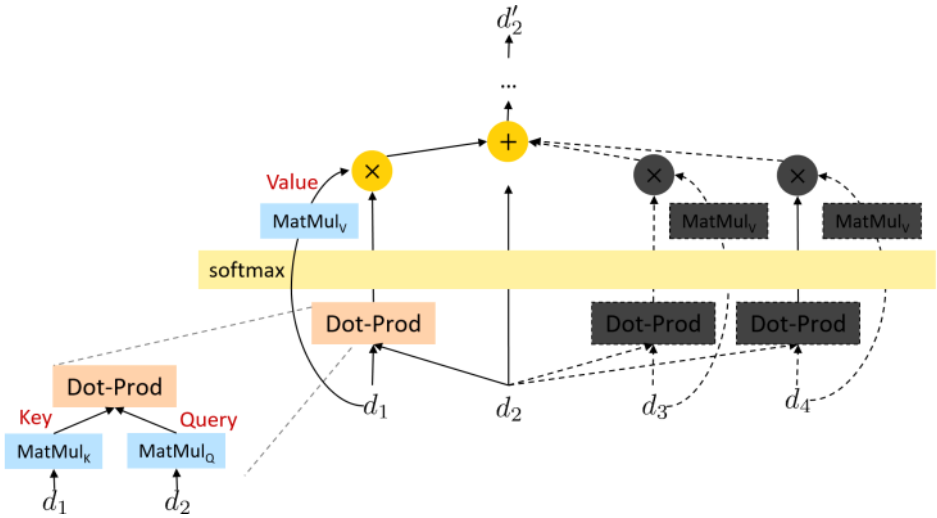
  - 每個value是一個vector，大V就是一個matrix

- Self-Attention
  - 自己對自己做attention (之前是對別人做attention: decoder對encoder做attention)
  - 第二個字當成target word，每個字跟自己的關聯
  - 把別人的vector拿過來計算match，得到alpha weight，把那個乘起來。只有我自己跟旁邊的人，去算出weighted sum
  - 

  - 透過FC得到representation或是downstream task
  - 每個字的word vector，去算跟旁邊人的關係，去算出weigted sum看其他跟我相關的字那些字重要
  - 自己對自己input的重要程度
  - 直接對input去做
  - 在任兩個位置之間，可以算出他們的關聯
  - 容易平行化，不用depend on之前算出來的
  - 每個時間點都可以平行
  - 可以放到seq2seq model
  - 。
  - encoder可以自己做很多次transformer self attention
  - decoder希望output每一個字的機率分布
  - 每個時間點會得到不同的attention dist.去做weigted-sum
  - 可以去做本來的attention
  - decode的時候可以先做attention，再把encoder decoder串再一起

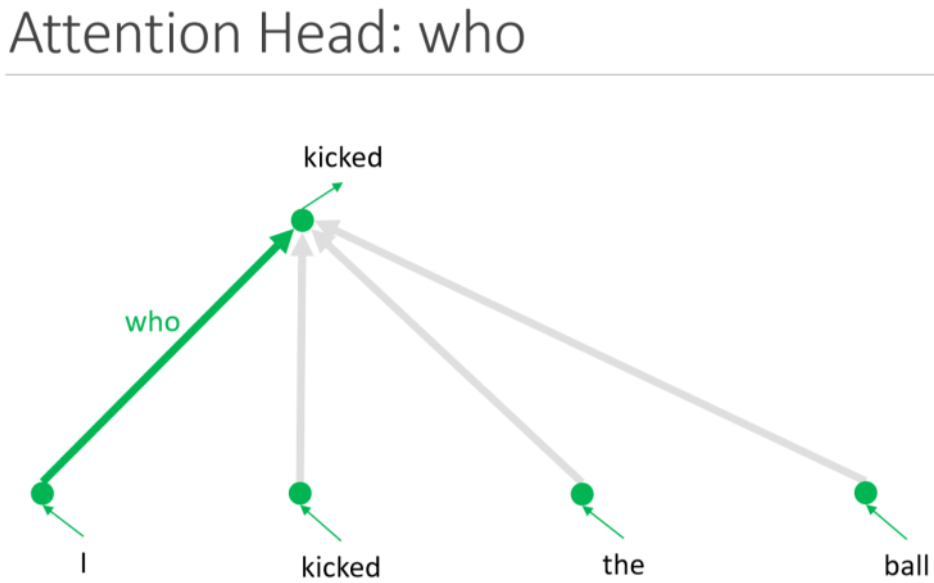
- 取代本來RNN，改成用self-attention計算
- 

- 細節:
- c2去跟旁邊的字算dot-prodct => key(其他字)、query(中間字target word)
- 經過soft max變成機率分布
- 然後alpha在跟自己相乘(value)，
- 

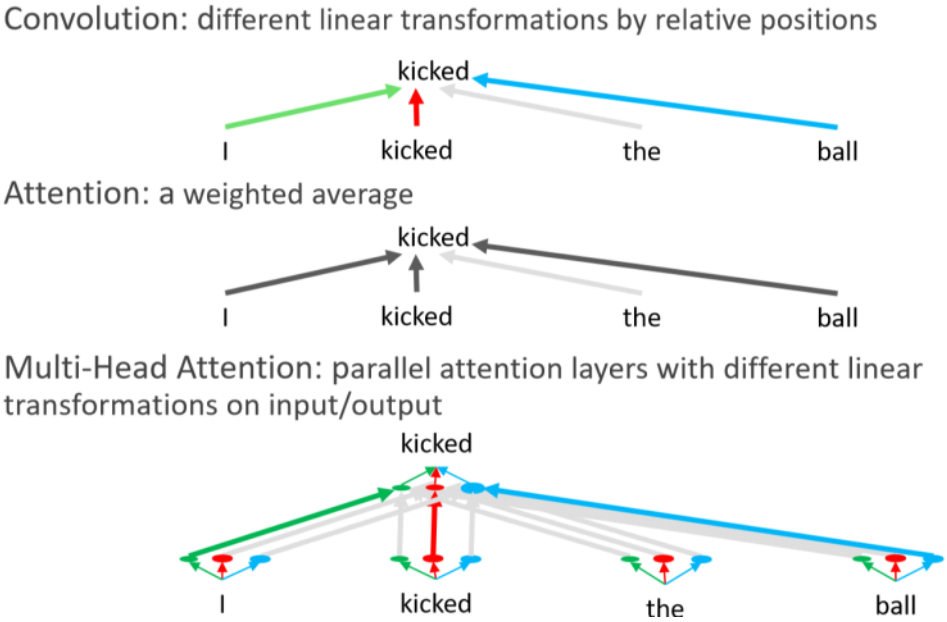
- 。
- decoder
- 因為沒辦法access後面的字，看不到後面的字，只能用前面的字去做self-attention
- 

Multi-head Attention

- conv中，在把資訊合起來，會去學不同字跟這個字的interaction不一樣，每個字代表的意義不一樣
- conv抓的資訊不一樣，不同interaction會不一樣
- self-attention他們的matrix會一樣，沒有說要學什麼關係
- 都是去把key拿進來跟query去算dot-product。self-attention只是去看相關程度，沒有去看有什麼interaction
- conv只能看local的，self-attention只能抓到關係大小不知道是甚麼
- 合起來:
- 先學第一個attentioni-head=who，就是去學interation的關係，因為只有第一個字才是who所以第一個字的attention比較高
- 第二個attention head，再去學一整組的self-attention，只有kicked的attention值比較高
- 下一個head=to whom，所以ball的self-attention值比較高
- 原本的self-attention只可以抓相關程度，不知道哪一種aspect的attention。所以multi-head就可以focus在各字那一種的attention
- 參數會增加，但是可以學到不同aspect又可以知道重要程度
- 所以整合了兩種好處

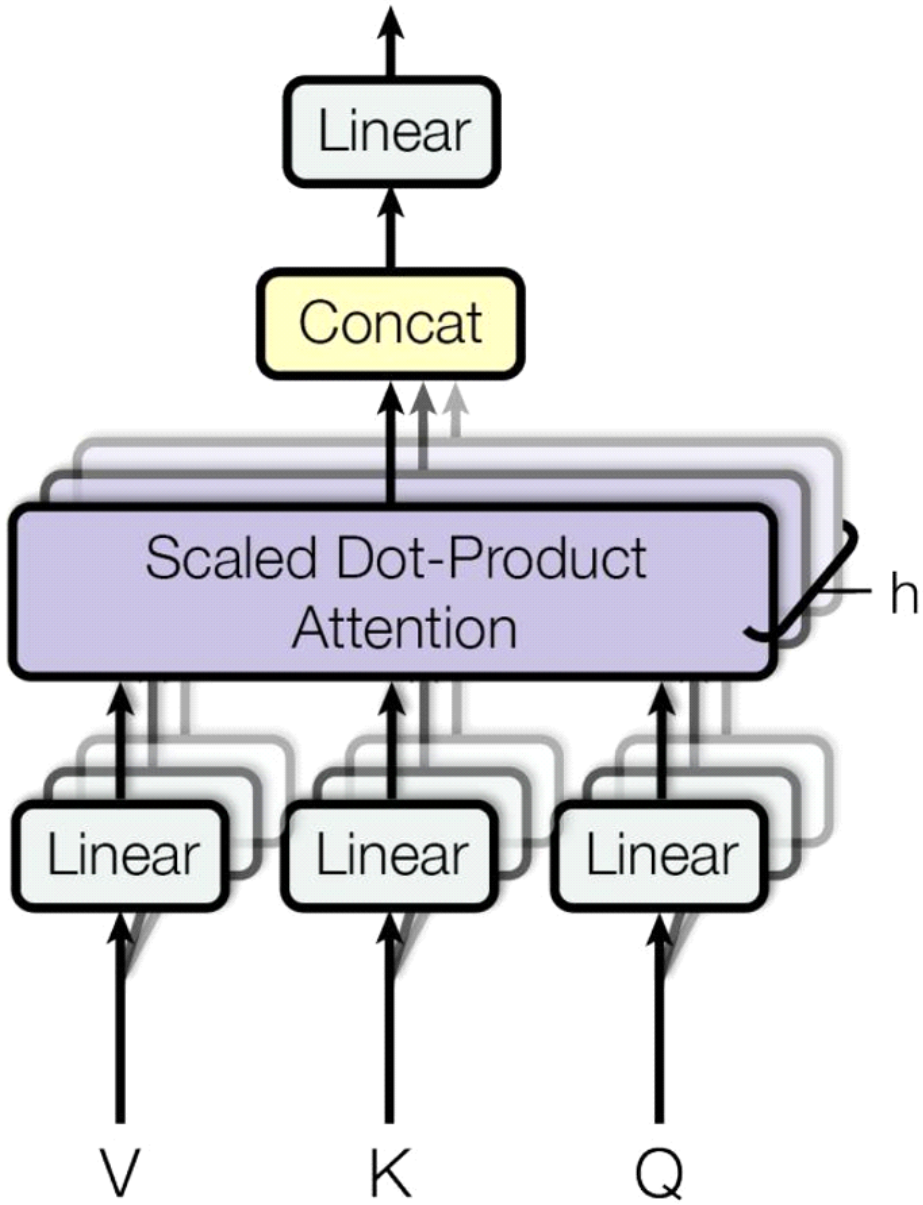


- 比較
  - cnn可以根據相對位置來做，所以只要相差在一定範圍，就會某一種matrix attention
  - 一般attention只是去看相關程度，關係大還是小，不知道關聯是怎樣的attention
  - Multi-head每一個head都會關注不同的aspect，所關注的information會不一樣，去看之間的關係，捕捉不同aspect資訊又有程度上的差異，所要的資料量比較多

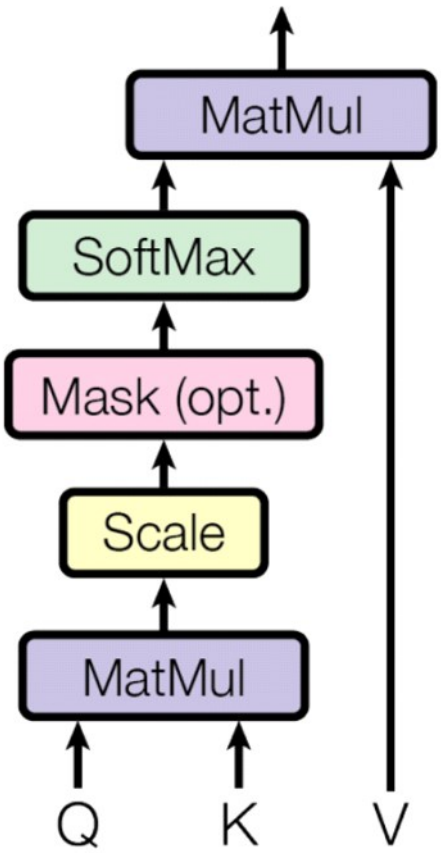
- 

Transformer

- attention is all you need
- 完全沒有recurrent，字近來都是靠attention把所有字合在一起
- harvord pytorch=>implement transformer video: http://nlp.seas.harvard.edu/2018/04/03/attention.html
- 為了捕捉不一樣的資訊跟程度上的差異=>multi-head attention
- value、key、query會先mapping到比較低維度的空間去，才會去用dot-product attention。有不同的head，所以會有不一樣的interaction要去學
- 把不同head aspect concat，再去做linear

- 

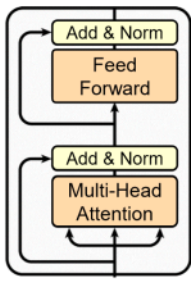
- h個head出來結果concate一起
  - head是一般的dot-prodct attention而來
  - 會自動把不同head放到interaction上
- 如果是重要aspect會對最後有影響，可以focus在該interaction上對output做出貢獻
- Scaled Dot-Product attention
  - scale數值
  - 當要對非常多數值要去算attention weight，會導致variance比較大，比較多人是1
  - 被拉很高的，有的就會相對很小，很難對ouput做出貢獻 (當k dim很大的時候)
  - 怕gradient不夠

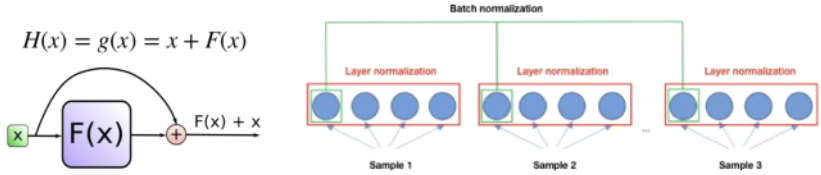


- 橘線不好

- Add& Norm
  - 會保有進入multi-head以前的資訊，防止沒被計算過的information可以保有下來
  - norm=layer normalization
  - residual x+layer(x)，打經過layer的相加
  - 把每個layer的input使平均為0 variance為1
  - ch block has
    nulti-head attention
    2-layer feed-forward NN (w/ ReLU)

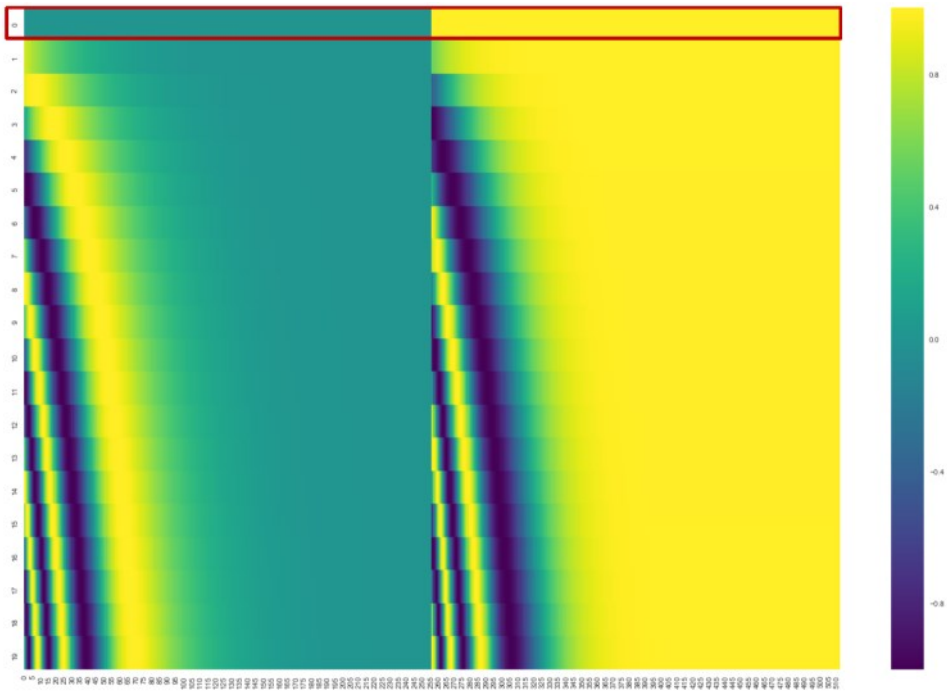
    th parts contain
    Residual connection & layer normalization (LayerNorm)
    LayerNorm(x + sublayer(x))
    Change input to have 0 mean and 1 variance per layer & per training point



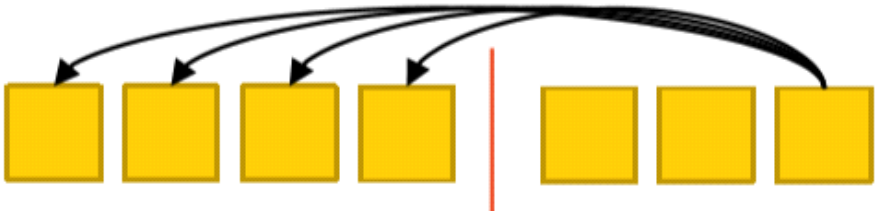


$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \qquad h_i = f(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i)$$

- encoder在做input的時候沒有temporal資訊: 我愛你跟你愛我一樣。attention沒考慮位置資訊
- 所以增加positional encoding
  - one-hot: 不知道現在有多少位置=>要假設最大長度二十，超過就

切成segment，第一個字跟第二十一個字會一樣。不好model關係
  - 改用sin跟cos的關係，利用兩者偏移差異當成對於位置的資訊
  - 用512維度表示sin跟cos位置
  - 每一個row就是一個位置
  - 0~20都會代表不同的值，固定dim代表不同位置，所以不用設定最長資訊多長
  - 
  - input長度變化不大的時候，one-hot也不會差，希望位置資訊可以放進來而已。因為有不同vector知道在哪個位置上面

Multi-Head Attention Details

- encoder: 自己對自己attention，算出來的weight也是算在自己身上
- decoder: masked=> decoding的時候會看不到後面的字，只能往前看，也不會用emcodeing資訊，跟encoder一樣，只是要擋掉後面的
- 
- encoder-decoder: decoder來的是query，去找在encoder裡面每個encoding當成key去算attention，key value都是encoder來的
- 

TIPS

Byte-pair encodings

Checkpoint averaging

ADAM optimizer with learning rate changes

Dropout during training at every layer just before adding residual

Label smoothing

Auto-regressive decoding with beam search and length penalties

- 優點
  - 跑比較快，可以平行化
  - 可以捕捉不同aspect，用attention概念抓出程度
  - 讓不同aspect合在一起
  - positional encoding保留temporal information
  - multi-head attention如果是有用的就可以apply block了