Transformer

RNN劣势

> attention 应用在RNN 并行度高

- 长序列内存开销
- 时序计算的信息遗忘

CNN

- 多输出通道

Tranformer 注意力机制

- multi-head 汲取CNN优势
- 只依赖注意力的encoder和decoder的注意力模型
  - 自注意力和ppoint-wise，全连接层一个个堆在一起

> auto-regression
> 自回归
> 最开始z，过去时刻的输入也会作为当前时刻的输出。

- shifted right
  - 一个个向右移 -编码器
  - Nx
    - N个神经块（block）
    - 注意力+MLP+残差连接
- 解码器
  - mask
- 具体实现

> Encoder

- N=6 完全一样的6个block
  - （1）multi-head
  - （2）position-wise fully connected feed-forwar network（MLP多层感知机）
    - ☐ 这里采用不同position单独一个MLP
    - 单隐藏层MLP
      - $FFN（x） = \max{(0,xW_1+b_1)}W_2+b_2$
      - 先扩大倍数，输出的时候由于前面用到了残差块，需要投影回去。
  - 子层残差连接（Add）
    - 参数要求输入和输出同样大小，否则需要投影操作（固定长度dk->调参简单）
      - d_k为word embedding 的结果
  - layer normalizarion

        - 大部分和batch norm一样，但是是对（feature，batch每一 batch 使得方差为1和均值为0
        - 三维的时候，对每一个样本做norm（相比batch norm不受样本seq长度影响）
    - 不使用batch norm
        - faults：每一个特征（feature）在小\mini-batch中使得方差为1和均值为0
        - 学习一个λ_1、β，向量通过学习可以缩放一个任意方差、均值的东西。
        - 三维的时候，对固定feature的做norm
        - 第一个自注意力输出，提取出了需要的语义信息。（语义空间）
        - 区别：
            - RNN是把上一个输出传入下一个mlp单元作为输入
            - transformer里面是通过attention层在全局中拉取整个序列的信息，再进行mlp的转换。
        - 相同点：RNN & Transformer
            - 关注点都在于怎么有效利用序列的信息。

> Decoder

- N=6 同样层，前两个子层与编码器一样，第三个子层为一个multi-head attenion
- 带掩码的注意力机制 mask 首层加mask，实际上计算的时候仍然是对全局进行了attetnion运输，但是输出的时候只对 $k_1 -> k_{t-1}$ 进行运算
    - 根据lstm的逻辑 t时刻不应该接受到t时刻之后的信息。
    - multi-head V、K、Q 分别进入线性层（投影到低维度），scale dot-product投影h次，再做h次的注意力函数，再做concat。
    - $MultiHead(Q,K,V)=Concat(head_1,...,head_h)W^o$
    - where $head_i = attention(QW_i^Q,KW_i^k,VW_i^V)$
    - 根据想要的东西，将编码器的输出的目标物品摘出来。（相似度计算）
    - **根据当前状**态**下的**编码**器**输**入**计**算**结**果得出当前**应该**在**编码**器**输**出**结**果中**应该**感兴趣和注意到的**东**西**。
    - 第二种注意力层，输入key和value的是编码器的输出（n$\times$d），输入query的是解码器（m$\times$d）的中间输出
        - 这里注意到，理解为何n和m不同向量长度，例如中译英：hello和你好，英文一个单词，中文两个汉字。
        -
- Attention
    - ☐ 注意力函数将query 和一些 key-value函数对输出为一个向量
    - compatibility function 相似度计算
    - key相当于名字， value是分数， query表示我想看谁的分数
        - 所以说通过query去查key对应的value
        - query与哪一个key比较像就会给出key对应相对高的权重
        - 不同相似函数给出不同注意力版本
        - 特点：

        - 一个query回合所有的key做运算，输出是query和所有的value做加权。

- [ ] transformer 中如何使用注意力
    - 自注意力机制（Key value）实际上为同一个东西
        - 在初始输入的时候，由于query也是Key，权重来自于value和key，输出是query根据权重求出来的结果（不考虑多头和投影），这么这里相当于权重就是位置单位向量。第一个注意力层若有多头、投影时，学习h个不一样的距离空间。

- Scaled DOT-Product Attention
- query 和 key 等长
- 步骤：
    - Q*K
    - scale（Q*K）
    - mask（scale（Q*K））
    - Softmax（mask（scale（Q*K）））
    - matmul（Softmax（mask（scale（Q*K））），V）
    - $Attention(Q,K,V)=softmax(\frac{QK^T}{\sqrt{d_k}})V$
    - softmax 是对每行做softmax
- [ ] 加型注意力机制
- [ ] 点积注意力机制
    - 这里采用了点乘
- 关于 Softmax
    - 当d_k不是很大的时候，除不除d_k关系不大
    - 当d_k很大，相对差距比较大，容易导致置信度最大的值相当接近于一，softmax结果近于收敛导致算梯度比较小训练困难，因此在softmax之前先除以$\sqrt{d_k}$
- [ ] embedding
    - 给定任意一个词，学习一个长度为d的向量来表示他
        - d_{model}
        - 添加位置位置：
            - 编码器，解码器输入，解码器softmax之前的线性层
            - 权重乘以$\sqrt{d_{model}}$以免最后学习的时候，因为维度过高，每个维度的数值过小，在残差中需要于positional encoding相加，为了两个向量在scale上相差不大。
- [ ] positional Encoding
    - 应用原因：attention不含有时序信息，顺序改变的时候输出没有改变
    - 输入中加如时序信息
        - 计算机，词在嵌入层中，长为512的向量，通过一些正余弦的操作求得。
- Why self attention

- conplexity per layer
  - 计算复杂度越小越好
- maximum path length
  - 一个数据点走到下一个数据点，复杂度越小越好。
- sequential operations
  - 下一步计算需要等前面多少步计算完成（并行度高则不需要等）