Attention从入门到精通

Attention在干嘛？Transformer在干嘛？

Query, Key, Value又在干嘛？

$$\text{Attention}(Q,\ K,\ V)\ =\ \text{softmax}\left(\frac{Q\,K^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$ 又又又在干嘛？

完全理解三个内容

- Attention要做的事情
- Attention出现之前大家的做法
- Attention清晰完整的计算过程 & 它有哪些好处

NLP的几项任务

- 句子分词
- 每个词转化为词嵌入向量

  每个句子对应一个矩阵 $X = (x_1, x_2, x_3, \dots, x_t)$
- 把得到的词嵌入向量转化为蕴含上下文信息的特征表示

  “学习上下文”

  这一过程是由encoder（编码器）来完成的

  $X \rightarrow y_t$

  y_t 就是学习到的上下文信息

Encoder部门的三个员工

- 为了完成**学习上下文**这件事情
  三个员工的三种不同方法
- RNN
  递归式学习上下文

- CNN
  窗口式遍历学习上下文

- Attention
  仅凭注意力学习上下文

RNN

- $y_t = f(y_{t-1}, x_t)$

  t时刻学到的上下文y_t，是结合t-1时刻的与当前词x_t来得到的

- 缺点

  无法并行，因此不能利用GPU等来并行计算提高效率

  难以捕捉全局的结构信息

CNN

- FaceBook《Convolutional Sequence to Sequence Learning》
- 窗口式遍历学习上下文

  设想一个宽度为k的滑块在句子上连续滑动

  每次都学习窗口内的k个单词，学出一个y_t

  这个窗口就是卷积核kernel
- 例如一个尺寸为3的卷积核

  那么$y_t = f(x_{t-1}, x_t, x_{t+1})$
- CNN可以并行计算，也能捕捉到一些全局的结构信息

Attention

- $y_t = f(x_t, A, B)$

  直接将x_t与每个词进行比较，得到y_t

  如果这里取A=B=X，就变成了self-attention（自注意力）

- $\text{Attention}(Q,\ K,\ V)\ =\ \text{softmax}\left(\ \frac{Q\,K^{\top}}{\sqrt{d_k}}\ \right)\ V$

看小猫，它好可爱

- 把这个句子里的每个词转化为词嵌入向量

  看(x_1) 小猫(x_2) ， 它(x_3) 好(x_4) 可爱(x_5)

- 每个词嵌入向量经过矩阵运算得到Q,K,V

$$
\begin{aligned}
x_1 &\xrightarrow{W_Q} Q_1,\ x_1 \xrightarrow{W_K} K_1,\ x_1 \xrightarrow{W_V} V_1 \\
x_2 &\xrightarrow{W_Q} Q_2,\ x_2 \xrightarrow{W_K} K_2,\ x_2 \xrightarrow{W_V} V_2 \\
x_3 &\xrightarrow{W_Q} Q_3,\ x_3 \xrightarrow{W_K} K_3,\ x_3 \xrightarrow{W_V} V_3 \\
x_4 &\xrightarrow{W_Q} Q_4,\ x_4 \xrightarrow{W_K} K_4,\ x_4 \xrightarrow{W_V} V_4 \\
x_5 &\xrightarrow{W_Q} Q_5,\ x_5 \xrightarrow{W_K} K_5,\ x_5 \xrightarrow{W_V} V_5
\end{aligned}
$$

看小猫，它好可爱

- Q Query

  “我要找什么信息”
- K Key

  “我有什么信息”
- V Value

  “我能提供的具体内容”

看小猫，它好可爱

$$x_1 \xrightarrow{W_Q} Q_1,\ x_1 \xrightarrow{W_K} K_1,\ x_1 \xrightarrow{W_V} V_1$$
$$x_2 \xrightarrow{W_Q} Q_2,\ x_2 \xrightarrow{W_K} K_2,\ x_2 \xrightarrow{W_V} V_2$$
$$x_3 \xrightarrow{W_Q} Q_3,\ x_3 \xrightarrow{W_K} K_3,\ x_3 \xrightarrow{W_V} V_3$$
$$x_4 \xrightarrow{W_Q} Q_4,\ x_4 \xrightarrow{W_K} K_4,\ x_4 \xrightarrow{W_V} V_4$$
$$x_5 \xrightarrow{W_Q} Q_5,\ x_5 \xrightarrow{W_K} K_5,\ x_5 \xrightarrow{W_V} V_5$$

- 以x_1为例

$$\text{Attention}(Q,\ K,\ V)\ =\ \text{softmax}\left(\ \frac{Q\,K^{\mathsf{T}}}{\sqrt{d_k}}\ \right)V$$

- 把Q_1与所有词的K分别做内积

$$<Q_1, K_1>=7, <Q_1, K_2>=5, <Q_1, K_3>=4, <Q_1, K_4>=9, <Q_1, K_5>=11$$

- 把得到的这些关注度的值除以$\sqrt{d_k}$，再softmax

  得到0.04,0.01,0.02,0.1,0.85

- Attention= $0.04V_1$ +0.01 V_2 +0.02 V_3 +0.1 V_4 +0.85 V_5

- Q Query

  “我要找什么信息”

- K Key

  “我有什么信息”

- V Value

  “我能提供的具体内容”

看小猫，它好可爱

- Softmax

$softmax(x_i) = \frac{e^{x_i}}{\sum x_i}$能够将一个K维向量“压缩”到另一个K维实向量中，每一个元素的范围都在(0,1)之间，并且所有元素的和为1

- 为什么要除以$\sqrt{d_k}$?

我们反正要softmax，反正要压缩，为什么还要除以$\sqrt{d_k}$?

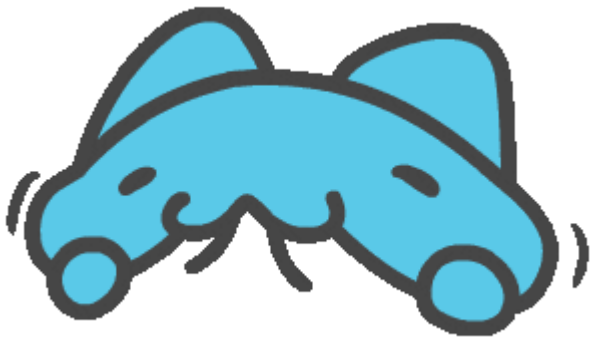
- 原因

某个x_i太大，它的e^{x_i}会碾压其他项，使得自己的softmax权重几乎为1； Softmax 在大输入时的梯度会趋近于 0

除以$\sqrt{d_k}$本质是归一化其方差

Attention就是这样学习上下文的

- 第一步：每个词先当Query
  “我要找什么信息？”
- 第二步：求该词的Query与所有词的key的内积（关注度）
  “谁最和我相关？”（“我要找的是谁？”）
- 第三步：关注度转化为权重
  相关程度具体占多少
- 第四步：用权重对每个词的Value加权求和
  学到上下文信息



Multi-Head Attention

- 用多个独立的‘小注意力模块’（**头**）分别学不同的上下文关系，再把这些关系拼起来融合成全面的上下文表示
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

  这是第i个attention头
- 接下来我们把h个attention头拼接在一起

  $MultiHead(Q, K, V) = concact(head_1, head_2, \dots, head_h) \cdot W_O$

  其中W_O是最终的一步线性映射

我们刚才做的实际上是 $Y = MultiHead(X, X, X)$的第一步

- Google所用的其实也是Self Multi-Head Attention

$$Y = MultiHead(X, X, X)$$

- 论文也表明了self-attention在机器翻译（甚至是一般的Seq2Seq任务）的序列编码上是相当重要的