①

Transformer 的逻辑解释

很多朋友问我, Transformer 是如何做逻辑推理? 在此尝试回答一下.

Curry-Howard isomorphism 将逻辑的 ⇒ 解释为函数的 →, 所以 任何作用在某些 states 之上的自同态 (endomorphism) 函数都可以看成是某种逻辑.

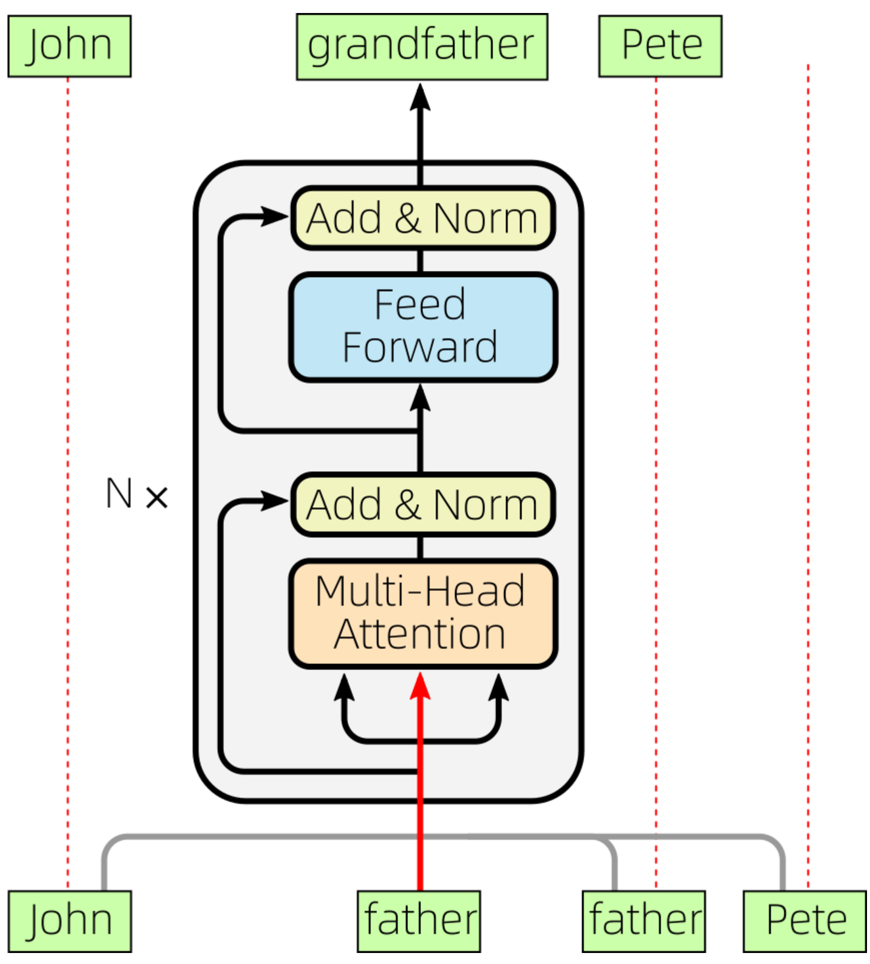
考虑这个例子:

$$\textit{John's father's father is Pete} \Rightarrow \textit{John's grandfather is Pete} \tag{1}$$

省略一些多余的 tokens, 加入变量[1]:

$$\forall X, Y.\ X\ \textit{father father}\ Y \Rightarrow X\ \textit{grandfather}\ Y \tag{2}$$

考虑这样一个简单的 N-层 Transformer, 它从左到右逐一处理 tokens, 红色表示 担任 Query 的 token (注意左边和右边的 tokens 都有参与 attention, 亦即 bi-directional 处理):



(3)

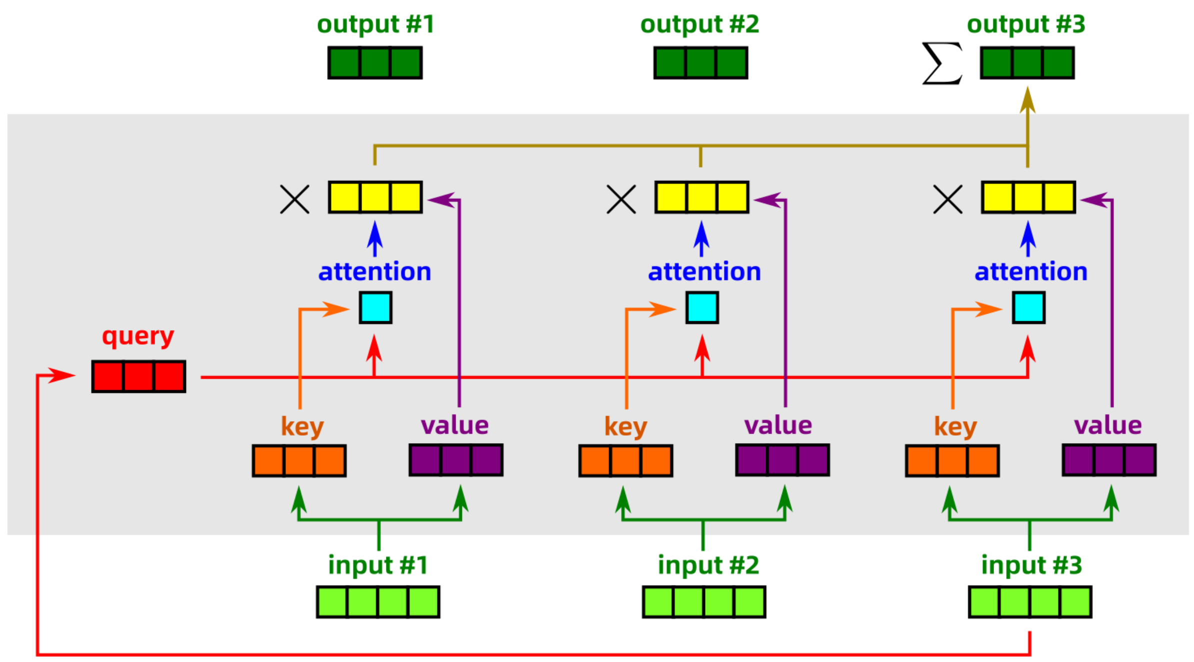
我们要回答两个问题:

- Query *father* 如何产生 *grandfather*?
- 第 4 输入位置的 *Pete* 如何出现在第 3 输出位置?

---

[1] 这里使用了 relation algebra 的表达方式, 这种方式更接近人类语言: aRb 表示 a 和 b 之间有关系 R. 关系之间可以 compose, 例如 $R \circ S$. 如果用 谓词逻辑 表达, 会比较累赘: father(X, Y) ∧ father(Y,Z) ⇒ grandfather(X,Z). 但我们不必太拘泥于逻辑形式的细节, 因为深度学习遵从「后结构主义」, 它只需要 抽取逻辑结构的某些特征, 将之变成 inductive bias 即可.

如果各位同学忘记了 **self-Attention** 机制是如何运作, 可以参考下图 重温一下 (图中第 3 个输入是 Query):



(4)

在我们的例子 (3) 里, Query($father_1$) 配上 Key($father_2$), 所谓「配对」是指 dot product: ⟨ Query, Key ⟩.

重温一下 Attention 的公式:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\langle Q, K\rangle}{\sqrt{d_k}}\right) V \tag{5}$$

Softmax 给出的是一组概率, 也就是注意力**权重**. 例如 给予 Value($father_1$) 30% 的权重, 给 Value($father_2$) 70%.

所以输出的向量是 Value($father_1$) 和 Value($father_2$) 的**线性组合**:

$$\boxed{\text{Output}} \quad grandfather = \alpha\,\text{Value}(father_1) + \beta\,\text{Value}(father_2) \tag{6}$$

留意上式中, *grandfather* 那些是 **词向量**, 即 vector embedding.

其实 Value($father_1$) 和 Value($father_2$) 是一样的, 如果不考虑 positional encoding.

目前为止, 以上的操作是 Transformer 可以轻易做到的, so far so good ☺

"Copy" Mechanism

第二个问题是 将 *Pete* 从第 4 位置 "copy" 到第 3 位置.

Query 是第 3 位置的 $father_2$.

之前我做过一个分析, 发觉这种 copying 的条件很难满足 (见下面 box).

其实要做到 copying 实在太容易了, 因为 Transformer 有 equivariance 对称性, 我们只需在输出加上某种 "positional encoding" 即可!

例如在 relation algebra 里, 只需 用某种 encoding (或 "tag") 表示 subject 或 object 的位置.

其实在 Transformer 的 隐层 里面, 根本没有所谓 position 的概念, 而是很可能有各种学习出来的 "tags".

位置 Copy 机制的分析

用更简洁的符号表示 copying 的条件, 并略去 softmax:

$$\forall \vec{x}. \quad \vec{x} = \langle Qf_2, Kf_1 \rangle Vf_1 + \langle Qf_2, Kf_2 \rangle Vf_2 + \langle Qf_2, K\vec{x} \rangle V\vec{x} \tag{7}$$

代入 $\vec{x} = 0$ 可得 前两项 $= 0$, ie:

$$\forall \vec{x}. \quad \vec{x} = \langle Qf_2, K\vec{x} \rangle V\vec{x} \tag{8}$$

很明显 $\vec{x}$ 需要是矩阵 V 的 eigen-vector. 但如果 $\vec{x}$ 可取的值的数目 $>$ 矩阵的 rank 或 维数, 那么 V 必然是 $k \cdot I$ 的形式. 那么 $\langle Qf_2, K\vec{x} \rangle = 1/k =$ constant 表示 K 将 $\vec{x}$ 投射到一个超平面上.

如果 $\vec{x}$ 的取值只是 embedding 空间的一个子集, 约束 可能更宽松.

但如果上述条件满足, 那么 (7) 的前两项不会是 0 而是 $f_1 + f_2$, 导致矛盾. 换句话说, copying 的条件无法满足. 但注意, 我们忽略了 softmax 的作用.

似乎满足 copying 的方法是: softmax 令某些 dot products 的值是 practically zero, 所以实际 $V\vec{x}$ 那项的注意力接近 100%.

用这种方法, $V = kI$, 则 整层 Transformer 不能储存其他 rules, 它只能作为「**copy 层**」, 可以在多个位置做独立的 copy 动作.

在实际的 Language Models 里面, 这种 copying 机制存在吗? 可以验证一下.

Multiple Logic Rules

现在来考虑, 一层的 Transformer 能不能储存 多个逻辑 rules?

这是重要的, 因为要达到 人类 common sense 的知识, 估计需要的逻辑 rules 数量, 起码达到 百万或千万以上. BERT/GPT 的成功给了我们一个大约的估算.

考虑 这两条不同的 rules:

- *father sister* $\Rightarrow$ *aunt*
- *church sister* $\Rightarrow$ *nun*

这等于要约束:

- *aunt* $= \gamma V$ *father* $+ \delta V$ *sister*
- *nun* $= \epsilon V$ *church* $+ \zeta V$ *sister*

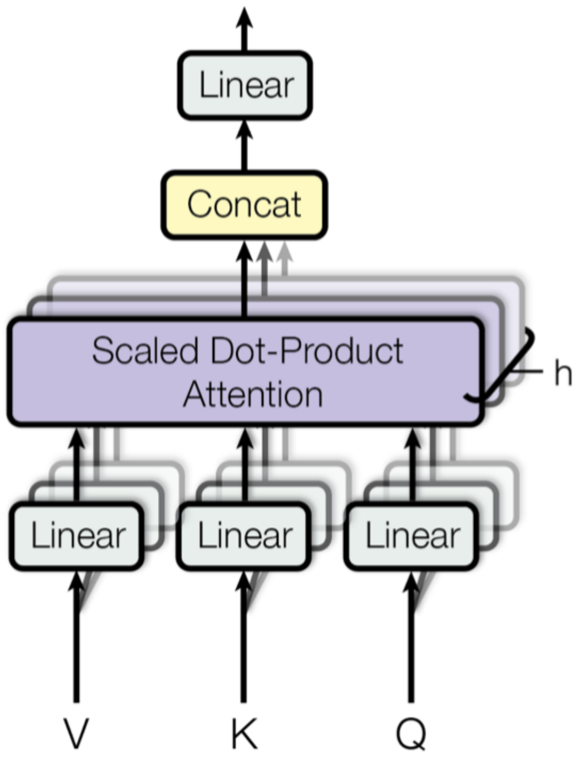
于是我们看到, 不同的 逻辑前提之下, 会产生不同的 Values 的**线性组合**[1], 这表达的能力是很高的.

而且 还不要忽略 MLP 层的 非线性作用.

[1]有人指出这是 convex combination, 因为 $\sum$ softmax = 1.

Multi-Head Attention

最后谈一下 Multi-Head Attention 从逻辑的角度看, 其意义为何?
首先重温这张图:



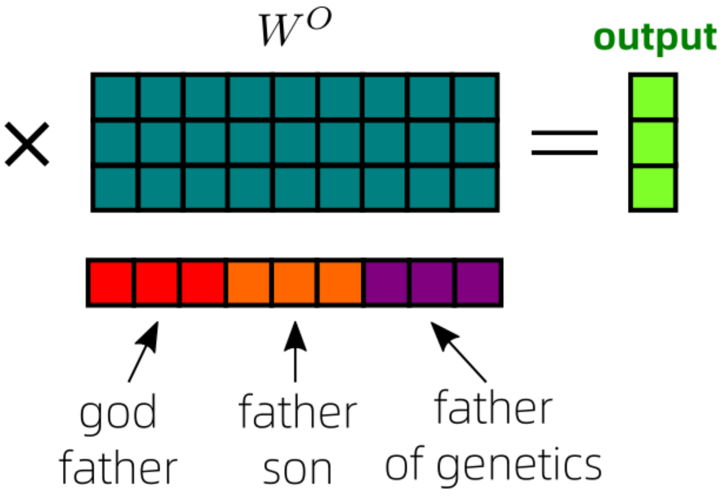
(9)

它将输入 X 乘以 W_i^Q, W_i^K, W_i^V 分别得出 Q_i, K_i, V_i, 即 h 个独立的「头」, 然后将这些独立的 Attention 结果 concatenate 到一起, 但最后输出时 还是要乘上一个 output 矩阵 W^O 变成一个 标准大小 的输出.

从逻辑角度看, 一个 rule 是被「前提 ⇒ 结论」决定的, 它不关心产生这个 rule 的过程的复杂性.

Multi-head 的情况下, rule 的结论 仍是只有一个, 但这个结论是可以根据输入变化的, 所以它可以包含很多不同的 rules.

例如 以 *father* 为 Query, 可以有「教父」「父子」「遗传学之父」等 不同的 heads. 这 3 个 heads 的结果 并接起来, 再产生一个 输出:



(10)

这个 输出 似乎可以看成是某种 "complex concept embedding", 关于 *father* 的更细致的概念表示.

从逻辑 rule 的角度考虑, 例如有 head_1:

$$\textit{sunny} \wedge \textit{no rain} \wedge \textit{not windy} \Rightarrow \textit{play tennis} \tag{11}$$

在 multi-head 情况下, 可能有另一个 head_2:

$$\textit{sunny} \wedge \textit{no rain} \wedge \textit{windy} \Rightarrow \textit{surfing} \tag{12}$$

在经典逻辑里, 这些 heads (或其成分) 可以用 ∧ (and) 或 ∨ (or) 来粘合. 在 multi-head 里, 各个成分 透过 "concat and then matrix-multiply" 的方法来做 conjunction. 后者这种方法 似乎有 **叠加** 的效果, 类似于 "or", 但似乎很难做到 "and" 的效果.

不要忘记 multi-head 可以产生不只一个结论, 而由于 MLP 层的作用, 这些结论可以分布在任意的 语义空间位置.

结论: Multi-head 在逻辑上相当于容许更多的 rules 储存在单一层的 Transformer 里.