预训练模型学习情况周报 4

姚凯

一、 本周学习：

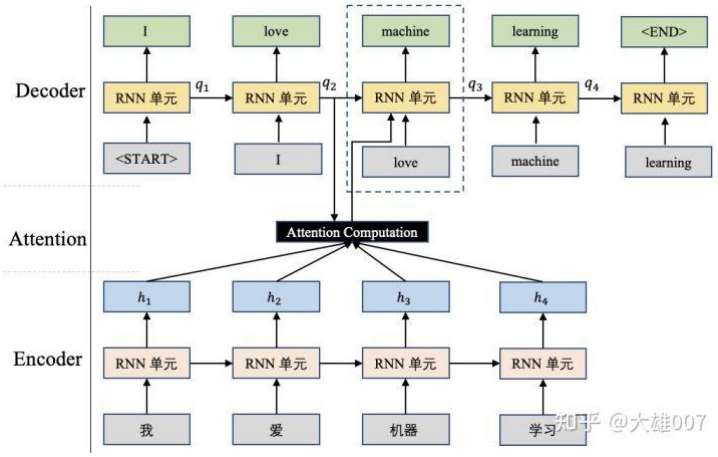
《自然语言处理的深度学习基础 PPT》P131-199

《自然语言处理中的预训练语言模型》P1-80

注意力机制

参考 https://zhuanlan.zhihu.com/p/393940472

模型某个时刻只有少量部分数据重要，以机器翻译为例：



$$a_i = softmax(s(h_i, q)) = \frac{exp(s(h_i, q))}{\sum_{j=1}^{n} exp(s(h_j, q))}$$

$$context = \sum_{i=1}^{n} a_i \cdot h_i$$

对于虚线框，先通过查询向量 q2 计算和每个 hi 的注意力分布，再加权求和得到模型当前应该关注的内容 context，再将 context 和上一时刻输出 love 送入虚线框内的 RNN 单元

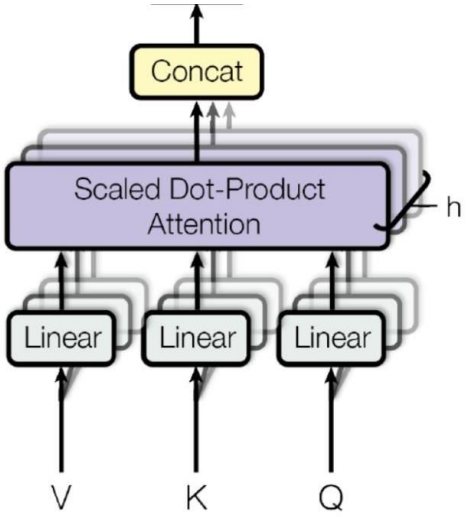
键值对注意力机制

Decoder 的输入不再是 H=[h1,h2…]，而是（K，V）,公式变为：

$$a_i = softmax(s(k_i, q)) = \frac{exp(s(k_i, q))}{\sum_{j=1}^{n} exp(s(k_j, q))}$$

$$context = \sum_{i=1}^{n} a_i \cdot v_i$$

多头注意力机制



对于多个查询向量和键值对 Q，K，V，a_{ij}代表第 i 个查询向量q_i与第 j 个输入信息k_j的注意力权重，$contest_i$表示由查询向量q_i计算得出的 Attention 向量，最后拼接所有的 context 输出向量

$$a_{ij} = softmax(s(k_j, q_i)) = \frac{exp(s(k_j, q_i))}{\sum_{t=1}^{n} exp(s(k_t, q_i))}$$

$$context_i = \sum_{j=1}^{n} a_{ij} \cdot v_j$$

$$context = context_1 \oplus context_2 \oplus context_3 \oplus \ldots \oplus context_m$$

自注意力机制

自注意力机制的查询向量使用输入信息进行生成，而不是选择与任务相关的查询向量。相当于模型读到输入信息后，根据输入信息本身决定当前最重要的信息。对于信息 H，分别乘以对应的参数矩阵得到查询空间 Q，键空间 K 和值空间 V

$$Q = HW_q$$

$$K = HW_k$$

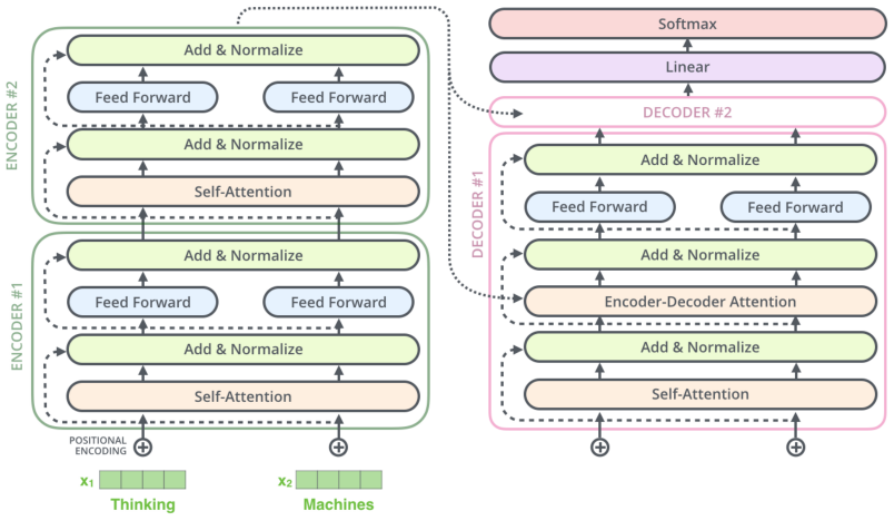
$$V = HW_v$$

$$context_i = \sum_{j=1}^{n} softmax(s(q_i, k_j)) \cdot v_j$$

Transformer

参考 https://zhuanlan.zhihu.com/p/338817680

详细了解可看 https://www.bilibili.com/video/BV1J94y1f7u5?p=32



以机器翻译为例：

编码器：对词嵌入矩阵 X，乘以对应的线性变换矩阵得到 Q，K，V，计算输出 Z=attention(Q,K,V)

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

d_k是Q, K矩阵的列数，即向量维度

Add 是一种残差连接，用于解决多层网络训练问题，让网络只关注当前差异部分

Norm 将每一层神经元的输入都转成均值方差一样的，可以加快收敛

Feed Forward 是两层全连接层

解码器：在第一个自注意力机制是单向的，对输入 X 计算 Q, K, V，再用 mask 矩阵遮挡操作，得到输出矩阵 Z=attention(Q,K,V)。

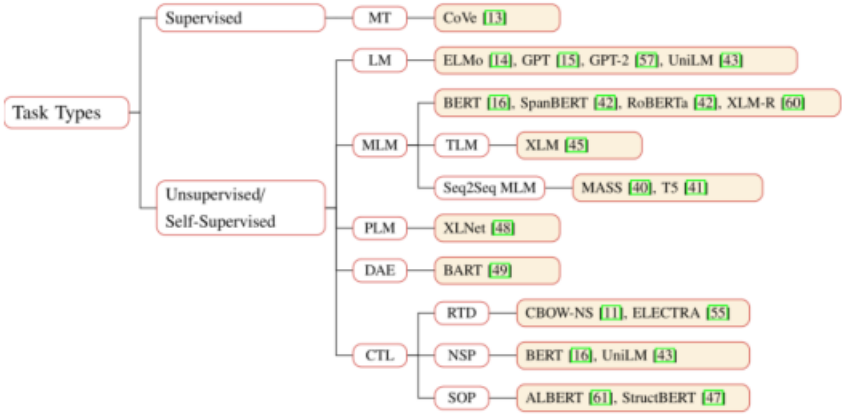
$$Attention(Q,K,V) = Softmax(\frac{QK^T \odot M}{\sqrt{d_k}})V$$

第二个自注意力机制从编码器的输出 C 得到 K，V，与 Z 计算 Q，得到的 Q，K，V 后用之前方法计算输出矩阵。

预训练

将大量低成本收集的训练数据放在一起，经过某种预训方法去学习其中的共性，然后将其中的共性“移植”到特定任务的模型中，再使用相关特定领域的少量标注数据进行“微调”。预训练可以获得一个好的初始化，也是一种有效的正则化手段

预训练任务

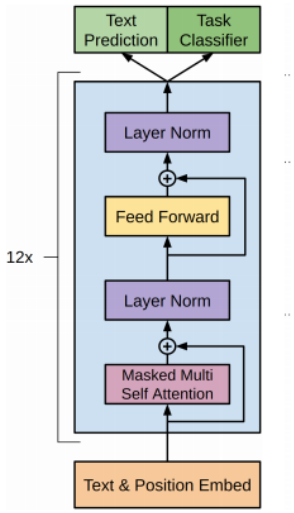


OpenAI GPT

参考 https://zhuanlan.zhihu.com/p/59286975

先通过无标签的文本去学习语言模型的初始参数，再根据具体的 NLP 任务，通过有标签的数据对模型进行 fine-tuning 参数微调。模型由多层单向的

transformer 组成。



训练的两个阶段：

1. 无监督的预训练：

对于无标签的文本 U={u1,u2…un}，最大化语言模型的极大似然函数，k 为文本上下文窗口。

$$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta)$$

单向 transformer 为 masked self-attention，只能对自己在内的前面所有词进行 attention。输入是词嵌入矩阵，输出 P(U)是词的概率分布，计算公式：

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\ P(u) &= \texttt{softmax}(h_n W_e^T) \end{aligned}$$

2. 有监督的 fine-tuning

设有标签的数据集每一条数据为{x1,x2..xn},标签为 y，通过上面的 transformer 块得到输出向量h_l^m，送入线性输出层，来预测标签 y

$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y).$$

Loss 函数：

$$L_2(C) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

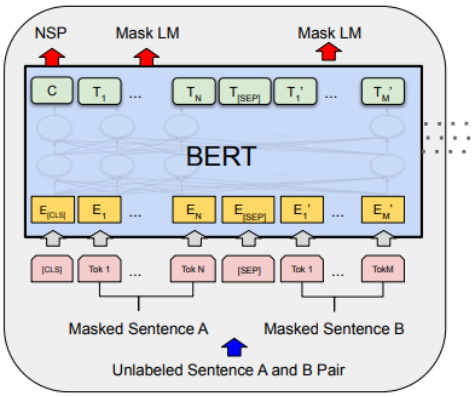
两阶段的目标函数相加训练整个模型:

$$L_3(C) = L_2(C) + \lambda * L_1(C)$$

BERT

参考 https://zhuanlan.zhihu.com/p/98855346

BERT 采用的是不经过 mask 的多层双向 Transformer



Pre-training

BERT 的预训练构建两个预训练任务，Masked Language Model 和 Next Sentence Prediction，两个任务的 loss 加起来就是整体的预训练 loss

MLM

因为既想知道上文和下文信息，又要保证模型不看到要预测的词的信息，故 BERT 在输入的句子中，用 mask 遮挡需要预测的词，用上下文分析预测被挖掉的词，类似完形填空。但[mask]不会产生在微调阶段，预训练和微调阶段产生不匹配。故使用以下规则:

1. 输入数据中随机选择15%的词用于预测，这15%的词中
2. 80%的词向量输入时被替换为<MASK>
3. 10%的词的词向量在输入时被替换为其他词的词向量
4. 另外10%保持不动

NSP

NSP 用来使模型有能力理解句子间的关系。每个训练样例由句子 A 和句子 B 组成，50%的概率 B 是 A 的下一句(标为 IsNext)，另外 50%B 和 A 无关(标为 NotNext)

训练样例

Input1=[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

Label1=IsNext

Input2=[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]

Label2=NotNext

Pytorch 学习

看完了 b 站《pytorch 深度学习快速入门教程》p1-p8

https://www.bilibili.com/video/BV1hE411t7RN?p=8&spm_id_from=pageDriver

配置了 pytorch 的环境，比较 pycharm 熟悉了 jupyter notebook 的使用，学习了 pytorch 加载模型的模块

二、 下周学习：

1）动手实践 Transformer 预训练语言模型框架，同时通过李宏毅的教程加深了解 https://www.bilibili.com/video/BV1J94y1f7u5?p=32

2）继续学习 Pytorch 教程