《Informer Beyond Efficient Transformer for Long Sequence Time-Series Forecasting》论文笔记

chongz

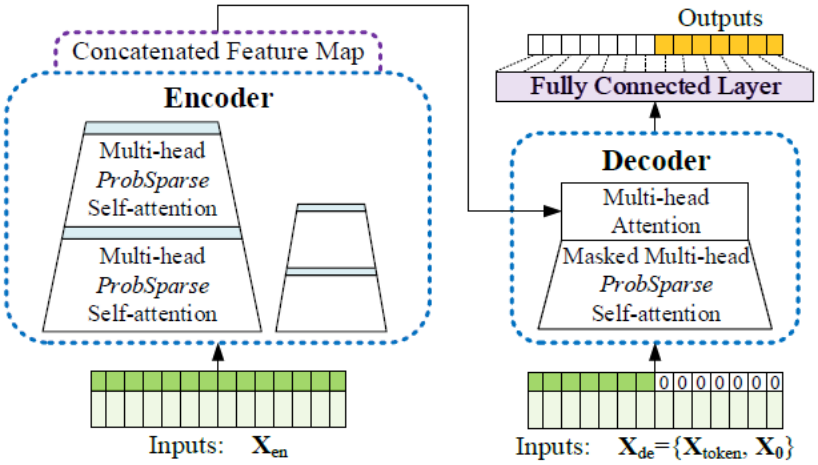
1.摘要

(1) 长序列时序预测（LSTF）要求模型具有捕捉输入和输出间的长程依赖关系

(2) 直接在 LSTF 任务上使用 Transformer 存在二次复杂性、内存大、encoder-decoder 架构的内在限制

(3) 提出基于 Transformer 的架构，命名为 Informer，有三个独特的特点：ProbSparse self-attention（时空复杂性都在 $\mathcal{O}(LlogL)$，在序列的依赖对其上也有好的表现）；self-attention distilling（通过减少级联层的一般，可以处理很长的输入序列）；generative style decoder（通过一次前向操作而不是逐步操作的方式，以极大地提高长序列预测的推断速度）

2.介绍



编码器和解码器分别处理输入和输出序列，并利用 ProbSparse 自注意力机制和自注意力蒸馏操作来提高模型的效率和性能。生成式解码器以一次前向操作的方式预测输出序列，从而提高了长序列预测的速度。这些设计和操作使得 Informer 模型在处理长序列时间序列预测任务时表现出色。

(1) 任务定义：一个在 t 时刻固定的滑动窗口的输入序列 $\mathcal{X}^t = \{x_{1t}, ..., x^t_{L_x} | x^t_i \in \mathbb{R}^{d_x}\}$，输出是 $\mathcal{Y}^t = \{y_{1t}, ..., y^t_{L_y} | y^t_i \in \mathbb{R}^{d_y}\}$，$d_y \geq 1$

(2) encoder-decoder 架构是一个一个输出预测序列：使用 $h^t_{L_x}$ 预测 y^t_1，使用 $h^t_{L_x+1}$ 预测 y^t_2...

3.建模方法：

多头自注意力机制：每一个 head 的查询向量的注意力被定义为如洗的概率形式：

$$\mathcal{A}(q_i, K, V) = \sum_j \frac{S(q_i, k_j)}{\sum_l S(q_i, k_j)} v_j = \mathbb{E}_{p(k_j|q_i)}[v_j] \tag{1}$$

$$p(kj|q_i) = \frac{S(q_i, k_j)}{\sum_l S(q_i, k_j)} \tag{2}$$

公式 2 表示了 q_i 对每一个 k_j 的注意力，公式 1 的计算时间复杂度是平方级的。

（1）Query Sparsity Measurement

如果注意力概率分布 $p(k_j|q_i)$ 和均匀分布 $q(k_j|q_i)$ 接近，那么 q_i 加权相当于对 v 的值进行求和。因此考虑计算 p 和 q 之间的不相似性程度来区分哪些 q_i 是重要的，本文利用 KL 散度来计算两个概率分布 q 和 p 之间的不相似性。

$$KL(q_i, K) = \sum_{j=1}^{L_K} ln(e^{\frac{q_i k_j^T}{d_k}}) - \frac{1}{L_K} \sum_{j=1}^{L_K} ln(\frac{q_i k_j^T}{d_k}) \tag{3}$$

前一项是 p 的分布，后一项是均匀分布 q。如果某一个 q_i 有一个很大的 KL，那么该 q_i 有很大可能包含主导注意力的点积对。也就是这个 q_i 是重要的。

但是 KL 的计算复杂度也是平方级的，所以提出了新的优化公式，时空复杂性都是 $\mathcal{O}(LlnL)$：

$$\bar{KL}(q_i, K) = \max_j \frac{q_i k_j^T}{\sqrt{d}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{q_i k_j^T}{\sqrt{d}} \tag{4}$$

（2）ProbSparse Self-Attention

基于上面的 QSM，ProbSparse Self-Attention 的计算公式如下：

$$\mathcal{A}(Q, K, V) = softmax(\frac{\bar{Q} K^T}{\sqrt{d_k}}) V \tag{5}$$

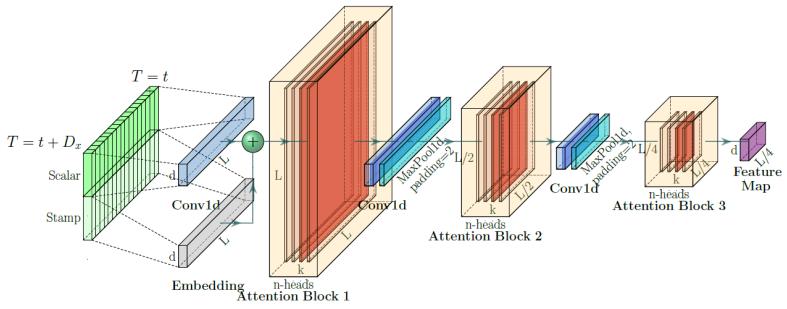
其中 $\bar{Q}$ 是通过 QSM 得到的 Top-u 个查询 q_i，$u = c\dot{l}nL_Q$，c 是一个常数因子。所以有以下结果：

(1) 每一个查询只需要计算 $\mathcal{O}(lnL_k)$ 次点积；

(2) 每一个查询的内存使用限制在 $\mathcal{O}(L_k lnL_Q)$；

(3) 对于每一个 head，生成了不同的查询，避免了严重的信息损失。

(3) Encoder: Allowing for Processing Longer Sequential Inputs under the Memory Usage Limitation



由于输入序列的不同位置可能具有相似的特征，导致注意力权重在不同位置具有相近的值，因此相近的特征因被多次组合而导致该特征被过度强调，忽略了其他重要的特征。所以引入蒸馏技术提取最重要的一些特征。当前 encoder 输入到下一个 encoder 的向量需要经过下面的处理：

$$X_{j+1}^{t} = MaxPool(ELU(Conv1d([X_j^t]_{AB}))) \tag{6}$$

其中 $[\dot{]}_{AB}$ 表示 Attention Block，$Conv1d$ 表示 1 维宽度 3 的卷积核，$ELU(\dot{})$ 表示激活函数

(1) 水平 stack 代表 Fig.(2) 中编码器副本中的一个单独的副本。这意味着 Informer 模型使用了多个编码器副本来处理输入序列。

(2) 提到的主要 stack 接收整个输入序列。然后，第二个 stack 获取输入的一半切片，随后的 stack 以此类推。这表明 Informer 模型的编码器部分采用了分层的方式，每个 stack 处理输入序列的一部分；

(3) 红色层表示点积矩阵，它们通过对每个层应用自注意力蒸馏（self-attention distilling）进行级联减少。这意味着通过自注意力机制，每个 stack 的注意力权重会逐层进行压缩和调整；

(4) 将所有 stack 的特征图连接在一起作为编码器的输出。这意味着 Informer 模型的编码器将每个堆栈的特征图合并起来作为最终的编码表示。

(4) Decoder: Generating Long Sequential Outputs Through One Forward Procedure

在解码器部分，传统的“动态解码”速度慢、灵活性高，本文使用一次前向计算生成整个输出序列

(1) 初始解码状态：encoder 的最终输出的隐藏状态 h。

(2) 起始标记：在动态解码中，生成一个特定的起始标记 X_start；在本文中，使用输出序列的前 n 个 token 作为起始标记 X_{nday}，并使用一个 X_0 序列包含了一次性输出的时间戳；

(3) 在动态解码中，是通过迭代生成预测序列，每一次迭代只生成预测序列中的一个；而本文的一次前向计算可以一次性生成 X_0 中包含的时间戳长度的序列；

(4) 将所有 stack 的特征图连接在一起作为编码器的输出。这意味着 Informer 模型的编码器将每个堆栈的特征图合并起来作为最终的编码表示。