《autoformer-decomposition-transformers-with-auto-correlation-for-long-term-series-forecasting-Paper》论文阅读笔记

chongz

1.摘要

(1) 基于 Transformers 的模型采用 self-attention，这使得模型找不到可靠的依赖关系；

(2) Transfoemers 采用 sparse attention version，限制了信息的利用；

(3) 提出一种带有 Auto-Correlation 机制的 Auto-former 架构。

2.建模方法：

(1) Series decomposition block

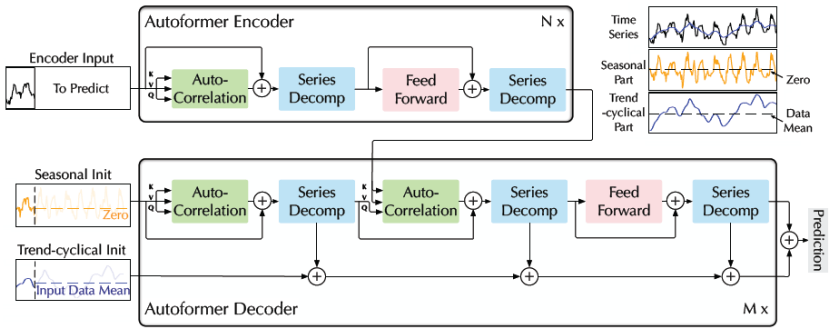
将序列分解成 trend-cyclical 和 seasonal 两部分，对于 future series 不可采用直接分解的方法，因为 future series 是未知的，论文提出了一个序列分解块作为 Autoformer 的内部操作，可以逐步从预测的中间隐藏变量中提取长期稳定的趋势。具体来说，论文采用移动平均法来平滑周期性波动并突出长期趋势：分解公式如下：

$$\mathcal{X}_t = AvgPol(Padding(\mathcal{X})) \tag{1}$$

$$\mathcal{X}_s = \mathcal{X} - \mathcal{X}_t \tag{2}$$

$\mathcal{X}_s$，$\mathcal{X}_t$ 分别表示 seasonal 和 trend-cyclical。

(2) Model Input



输入序列为 $\mathcal{X}_{en}$，长度为 I $\mathcal{X}_{des}, \mathcal{X}_{det}$，分别表示 seasonal 和 trend-cyclical，分解部分来自输入序列 $\mathcal{X}_{en}$ 的后面一半，长度为 I/2，分解得到的 seasonal 要拼接上一个长度为 $\mathcal{O}$ 的标量值序列，而 trend-cyclical 要拼接上 $\mathcal{X}_{Mean}$：

$$\mathcal{X}_{ens}, \mathcal{X}_{ent} = SeriesDecomp(\mathcal{X}_{en[\frac{I}{2}:I]}) \tag{3}$$

$$\mathcal{X}_{des} = Concat(\mathcal{X}_{ens}, \mathcal{X}_0) \tag{4}$$

$$\mathcal{X}_{det} = Concat(\mathcal{X}_{ent}, \mathcal{X}_{Mean}) \tag{5}$$

其中 $\mathcal{X}_{Mean}$ 中填充的是 $\mathcal{X}_{en}$ 的平均值。

（3）Encodre

每一层的 Encoder 的输入来自上一层的输出，第一层 Encoder 的输入就是输入序列，每一层 Encoder 的内部结构为：计算输入的 Auto-Correlation（计算时序数据在不同滞后时间点的相关性），Auto-Correlation 的结果和输入拼接，经过第一个分解器（SeriesDecomp），并且只取分解得到的 seasonal 部分；seasonal 部分经过一个 FeedForward 激活层，FeedForward 的输出和 seasonal 拼接，再将拼接的向量经过第二个分解器，同样只取 seasonal 部分作为整个 Encoder 的输出，公式如下：

$$\mathcal{S}_{\urcorner\backslash}^{\updownarrow,\infty}{}_{,=} SeriesDecomp(Auto - Correlation(\mathcal{X}_{en}^{l-1})) + \mathcal{X}_{en}^{l-1} \tag{6}$$

$$\mathcal{S}_{\urcorner\backslash}^{\updownarrow,\in}{}_{,=} SeriesDecomp(FeedForward(\mathcal{S}_{en}^{l,1})) + \mathcal{S}_{en}^{l,1} \tag{7}$$

其中 _ 表示被消除的 trend 部分，$\mathcal{X}_{en}^{l}$ 表示第 l 层 encoder 的输出，$\mathcal{S}_{en}^{l,i}$ 表示第 l 层 encoder 的第 i 个分解器的 seasonal 部分的输出。

（4）Decoder

每一层 Decoder 的输入来自上一层 Decoder 的输出（$\mathcal{X}_{de}^{l} = Decoder(\mathcal{X}_{de}^{l-1}, \mathcal{X}_{en}^{N})$），在 decoder 里，不仅要计算 seasonal，还需要逐步累积 trend-cyclical，每一个 decoder 的输入包括了 seasonal、trend-cyclical 和 $\mathcal{X}_{en}^{N}$。特别地，第一层的 seasonal$\mathcal{X}_{de}^{0} = \mathcal{X}_{des}$，trend-cyclical$\mathcal{T}_{de}^{0} = \mathcal{X}_{det}$。然后对于 seasonal 部分，输入 seasonal，经过 Auto-Correlation 计算得到一个结果，该结果和输入的 seasonal 拼接，该拼接结果经过第一个分解器分别得到 $\mathcal{S}_{de}^{l,1}, \mathcal{T}_{de}^{l,1}$，$\mathcal{S}_{de}^{l,1}$ 和 $\mathcal{X}_{en}^{N}$ 经过 Auto-Correlation（计算两个时序数据在不同滞后时间点的相关性）得到一个结果，该结果经过第二个分解器分别得到 $\mathcal{S}_{de}^{l,2}, \mathcal{T}_{de}^{l,2}$，$\mathcal{S}_{de}^{l,1}$ 经过 FeedForward 层激活后与未经激活的 $\mathcal{S}_{de}^{l,1}$ 拼接，经过第三个解码器，得到 $\mathcal{S}_{de}^{l,3}, \mathcal{T}_{de}^{l,3}$；对于 trend-cyclical 部分，有三个权重矩阵 $W_{l,i}, i = 1, 2, 3$，分别用来提取每一个对应分解器分解得到的 trend-cyclical 的特征，最终的 trend-cyclical 为上一层的 trend-cyclical 结果加上当前层中三个经过权重矩阵抽取的 trend-cyclical 的结果。具体公式如下：

$$\mathcal{S}_{de}^{l,1}, \mathcal{T}_{de}^{l,1} = SeriesDecomp(Auto - Correlation(\mathcal{X}_{de}^{l-1})) + \mathcal{X}_{de}^{l-1} \tag{8}$$

$$\mathcal{S}_{de}^{l,2}, \mathcal{T}_{de}^{l,2} = SeriesDecomp(Auto - Correlation(\mathcal{S}_{de}^{l,1}\mathcal{X}_{en}^{N})) + \mathcal{S}_{de}^{l,1} \tag{9}$$

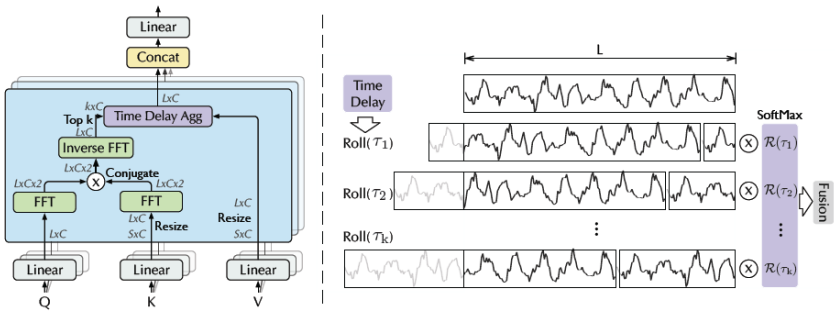
$$\mathcal{S}_{de}^{l,3}, \mathcal{T}_{de}^{l,3} = SeriesDecomp(FeedForward(\mathcal{S}_{de}^{l,2})) + \mathcal{S}_{de}^{l,2} \tag{10}$$

$$\mathcal{T}_{de}^{l} = \mathcal{T}_{de}^{l-1} + \mathcal{W}_{l,1} * \mathcal{T}_{de}^{l,1} + \mathcal{W}_{l,2} * \mathcal{T}_{de}^{l,2} + \mathcal{W}_{l,3} * \mathcal{T}_{de}^{l,3} \tag{11}$$

最终的预测结果为：

$$O = \mathcal{W}_{\mathcal{S}} * \mathcal{X}_{de}^{M} + \mathcal{T}_{de}^{M} \tag{12}$$

（5）Auto-Correlation



如上图，K，V 来自 $\mathcal{X}_{en}^{N}$ 并且会被 resize 成长度为 O，Q 来自之前的 decoder block。在本文中，通过 delay 来计算时序数据和 time delayτ 的时序数据的相关性，找出相关性最高的几个 time delayτ，这可以使相似的子序列对齐，使他们在同一时间阶段进行分析。举个例子，在基于 sparse 的注意力的 transformer 的架构中，通过逐点积挑选出一些重要的查询，每一个查询是时序数据中不同位置或不同时间的数据；而在本文中，是通过计算出几个重要的 time delayτ，计算每个延迟 τ 的子序列和原序列的相关性。所以他们的目的都是想利用稀疏性来加快计算，不过本文的方法比 attenion 的方法更好，因为利用子序列作为“查询”可以携带更多信息，并且减少了聚合次数。