Transformer 的不足：

1. 平方时间复杂度 2.高内存占用 3.现有的编解码结构的局限性

Informer 解决：

1. 提出 ProbSparse Self attention，能够筛选出所有最重要的一部分 query，能够大大的减少计算的时间复杂度和空间复杂度。
2. 提出了一个 self attention distilling 操作，通过卷积和池化操作减少维度和网络参数量。
3. 提出 Generation Style Decoder 只需一步即可得到所有的预测结果，而不是 step-by-step，提升了长序列预测的速度。

摘要

许多现实世界的应用需要长序列时间序列的预测，如电力消耗计划。长序列时间序列预测(LSTF)要求模型具有较高的预测能力，即能够有效捕捉输出和输入之间精确的长期相关性耦合。最近的研究表明，潜在的 transformer 增加了预测容量。然而，有几个问题与防止直接应用程序崩溃的 transformer 有关，包括四个方面的复杂性、高内存使用量和对代码无关架构的一致模拟。为了解决这些问题，我们设计了一个基于高效 transformer 的模型，命名为 Informer，具有显著的特点:(i)一个自关注机制，它在时间复杂度和内存使用方面达到了$\mathcal{O}(L \log L)$，并且在序列的依赖对齐方面具有相当的性能。(ii)自我注意力提取通过将层叠层输入减半来突出主要注意力，并有效地处理长期序列。(iii)生成式解码器虽然概念简单，但它可以预测长时间序列的运行，而不是逐步进行，这极大地提高了长时间序列预测的推理速度。在我们的大规模数据集上进行了广泛的实验，验证了模型的有效性，并为问题提供了新的解决方案。

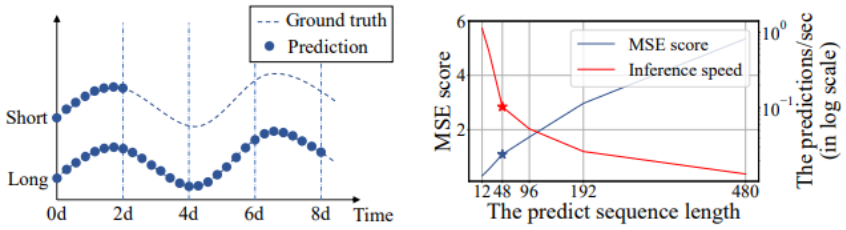
批注 [W1]: Transformer 存在的问题

批注 [W2]: Informer 的优势

1. 介绍

时间序列预测是许多领域的重要组成部分，如传感器网络监控(Papadimitriou 和 Yu2006)，能源和智能电网管理，生态-经济学与金融和疾病传播分析（Matsubaraaetal.2014），在这种情况下，我们可以对时间序列数据进行定量预测，即长序列时间序列预测(LSTF)。然而，现有的方法大多是为解决短期问题而设计的，如预测 48 点或者更少，(Hochreiter and Schmidhuber1997;Li et al.2018;Yu et al.2017;Liu et al.2019;Qin et al.2017;Wenetal.2017)。增加循环序列将模型的预测能力训练到该字符串所在的点。作为一个经验例子，图(1)显示了预测结果

批注 [W3]: 现有的都是解决短期问题的，对于长时间序列，超过 48 个点，预测性能急剧下降。



（a）序列预测　　(b) 在序列上运行 LSTM。

图 1：(a) LSTF 可以覆盖比短序列预测更长的时期，在政策规划和投资保护方面做出重要区分。

（b）现有方法的预测能力限制了 LSTF 的表现。例如，从长度=48 开始，均方误差上升得高得不可接受，推理速度迅速下降。

长时间序列的网络预测，其中，LSTM 网络预测从短期（12 点，0.5 天）到长期（480 点，20 天）的变电站每小时温度。当预测长度大于 48 点(图(1b)中的实心星)时，整体性能差距很大，此时均方误差上升到不令人满意的性能，推理速度急剧下降，LSTM 模型开始失败。

LSTF 面临的主要挑战是提高预测能力，以满足日益增长的长序列需求，这需要(a)非凡的远程比对能力和(b)长序列输入和输出的高效操作。最近，transformer 模型显示出比 RNN 模型更高的性能和更大的范围依赖性。这些 self attention 机制可以将工作信号传输路径的最大长度减少到最大长度$\mathcal{O}(1)$，并避免重复出现的结构，从而 transformer 显示出解决 LSTF 问题的巨大潜力。然而，self attention 机制违反了要求(b)，因为它的二次计算和内存消耗只有一个长度的输入/输出。一些大规模的 transformer 模型在自然语言处理任务上投入资源并产生令人印象深刻的结果(布朗等人，2020)，但是在几十个图形处理器上的训练和昂贵的部署成本使得这些模型在现实世界的 LSTF 问题上负担不起。Self attention 机制的有效性和 transformer 架构成为应用到长时间序列的问题上的瓶颈。这是需要解决的问题。因此，在这篇论文中，我们试图回答这个问题:我们能改进 transformer 模型去提高它的计算，内存和架构效率，并且保持较高的预测能力。

Vanilla Transformer (Vaswani et al. 2017)有三个明显的相似之处在解决长时间序列问题上。

（1）自我注意力机制的二次计算

自我注意机制的原子操作，即正则点积，导致时间复杂度和每层内存使用变为$\mathcal{O}(L^2)$。

（2）长输入堆叠层的内存瓶颈。

编码器/解码器层的堆叠使$\mathcal{O}(J \cdot L^2)$的总内存使用量增加，这限制了模型在接收长序列输入

批注 [W4]: 长时间序列预测需要解决的问题

批注 [W5]: Transformer 降低了以往模型的传输路径的最大长度

批注 [W6]: Transformer 并没有降低内存使用，并内存使用很高。这是 transformer 需要解决的问题。

批注 [W7]: Vanilla transformer 需要解决的问题。

时的可扩展性。

（3）预测长期产出的速度骤降。

Vanilla transformer 的动态解码使得逐步推理像基于 RNN 的模型一样慢(Fig.(1b))。以前有一些关于提高 self attention 效率的工作。Sparse transformer(Childetal.2019)、Logsparse transformer(Lietal.2019)和 longformer (Beltagy、Peters 和 han2020)都使用启发式方法去解决限制 1 和减少 self attention 机制的复杂度$\mathcal{O}(L\log L)$，在这种情况下，它们的效率受到限制((Qiuetal.2019))。Reformer(Kitaev，Kaiser，andLevskaya2019)也实现了本地敏感哈希 self attention 的$\mathcal{O}(L\log L)$，但它只适用于极长的序列。最近，Linformer (Wang et al.2020)提出了一个线性复杂性$\mathcal{O}(L)$，但对于现实世界中的长序列输入，投影矩阵是不固定的，可能有降解为$\mathcal{O}(L^2)$的风险。Transformer-XL（戴等人，2019)和 Compressive Transformer (Rae 等人，2019)使用辅助隐藏状态来捕获长期依赖关系，这可能放大限制 1，不利于打破效率瓶颈。所有这些工作主要集中在限制 1 上，消除 2 和 3 仍然是一个解决问题的方法。为了提高预测能力，我们必须克服这些限制，并在未来实现更高的效率。

为此，我们的工作深入研究了这三个问题。我们调查了 self attention 机制的缺陷，对组件之间的差异进行了改进，并进行了广泛的实验。本论文的内容总结如下：

- 我们提出了一个成功的方法来增强预测能力解决这个问题，这个方法验证了类似于 transformer 的模型的点特征值来适应单个长范围的时间序列输出和输入之间的相关性。
- 我们提出了一种 ProbSparse Self attention 来有效地替代经典的 self attention。它实现了$\mathcal{O}(L\log L)$时间复杂度和$\mathcal{O}(L\log L)$内存使用率的依赖对齐。
- 我们提出了 self attention distilling 的操作，以解决 J 层中特权支配的注意分数问题，并将总体空间复杂度大大降低到$\mathcal{O}((2-\epsilon)L\log L)$，这有助于接收长序列输入。
- 我们提出生成式解码器，只需要一个前向步骤就可以获得长序列输出，同时避免了推理阶段的累积误差传播。

批注 [W8]: 速度慢

批注 [W9]: 散列法（Hashing）或哈希法是一种将字符组成的字符串转换为固定长度（一般是更短长度）的数值或索引值的方法，称为散列法，也叫哈希法。由于通过更短的哈希值比用原始值进行数据库搜索更快，这种方法一般用来在数据库中建立索引并进行搜索，同时还用在各种解密算法中。

批注 [W10]: 目前针对 transformer 改进的模型，都是在问题 1 上（时间复杂度，也就是效率上），问题 2 和问题 3 并没有很好的解决

批注 [W11]: Informer 的三项改进
1..时空复杂度
2.减少网络参数并降维
3.提高效率

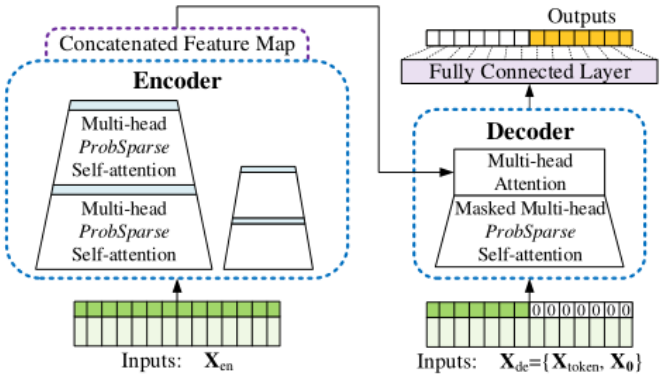


图 2:Informer 模型概述。

左:编码器接收大量长序列输入(绿色系列)。我们用提议的 ProbSparse Self attention 来代替标准的 Self attention。蓝色的梯形是 self attention distilling 操作，以选取主要的 self attention，，大大减小了网络规模。层堆叠副本增加了鲁棒性。右:解码器接收长序列输入，将目标元素填充为零，测量特征图的加权注意成分，并以生成方式立即预测输出元素(橙色系列)。

批注 [W12]: 填 0 是为了补长序列长度，使序列对齐么？

2. 初步措施

我们首先提供了 LSTF 问题定义。在带有固定大小窗口的滚动预测设置下，在时间 t 我们有了输入 $\mathcal{X}^t=\left\{\mathbf{x}_1^t,\ldots,\mathbf{x}_{L_x}^t \mid \mathbf{x}_i^t \in \mathbb{R}^{d_x}\right\}$，输出是预测相应的序列 $\mathcal{Y}^t=\left\{\mathbf{y}_1^t,\ldots,\mathbf{y}_{L_y}^t \mid \mathbf{y}_i^t \in \mathbb{R}^{d_y}\right\}$ LSTF 问题激励比以前的工作更长的输出长度Ly（Cho 等人，2014；Sutskever、Vinyals 和 Le 2014），特征维数不限于单变量情况（dy≥ 1)。

编解码器体系结构许多流行的模型被设计成将输入表示 $\mathcal{X}^t$ “编码”为隐藏状态表示 $\mathcal{H}^t$，并从 $\mathcal{H}^t=\left\{\mathbf{h}_1^t,\ldots,\mathbf{h}_{L_h}^t\right\}$ 中“解码”输出表示 $\mathcal{Y}^t$。该推理涉及一个称为“动态解码”的分步过程，其中解码器从先前的状态 $\mathbf{h}_k^t$ 计算新的隐藏状态 $\mathbf{h}_{k+1}^t$，并从第 k 步计算其他必要的输出，然后预测第（k+1）序列 y_{k+1}^t。

输入表示给出了统一的输入表示，以增强时间序列输入的全局位置上下文和局部时间上下文。为了避免繁琐的描述，我们将细节放在附录 B 中。

3. 方法

现有的时间序列预测方法可以大致分为两类。经典时间序列模型是时间序列预测的可靠工具(Box 等人，2015 年；Ray 1990；Seeger 等人，2017 年；西格、萨利纳斯和奴格特(2016)，深度学习技术主要是通过使用 RNN 和他们的变体开发一个编码器-解码器预测范式(Hochreiter 和 Schmidhuber 1997 李等 2018；Y u 等人，2017 年)。我们提出的 Informer 特有编码器-解码器架构，而目标是 LSTF 问题，请参考图(2)了解概况，并参考以下章节了解详情。

批注 [W13]: Informer 的目的是解决 LSTF 问题

高效的自我注意机制

（Vaswani et al.2017）中的经典 self attention 是基于元组输入（即 query、key 和 value）定义的，它将按比例的点积表示为 $\mathcal{A}(\mathbf{Q},\mathbf{K},\mathbf{V}) = Softmax\left(\mathbf{Q}\mathbf{K}^{\top}/\sqrt{d}\right)\mathbf{V}$，

其中 $\mathbf{Q}\in\mathbb{R}^{L_Q\times d},\mathbf{K}\in\mathbb{R}^{L_K\times d}$ $\mathbf{V}\in\mathbb{R}^{L_V\times d}$。为了进一步探讨自我注意机制，让$q_i$、$k_i$、$v_i$分别代表 Q、K、V 中的第 i 行。按照（Tsai et al.2019）中的公式，第 i 个 query 的注意力被定义为概率形式的核平滑器：

$$\mathcal{A}\left(\mathbf{q}_i,\mathbf{K},\mathbf{V}\right)=\sum_j\frac{k\left(\mathbf{q}_i,\mathbf{k}_j\right)}{\sum_l k\left(\mathbf{q}_i,\mathbf{k}_l\right)}\mathbf{v}_j=\mathbb{E}_{p\left(\mathbf{k}_j\mid\mathbf{q}_i\right)}\left[\mathbf{v}_j\right] \qquad (1)$$

$p\left(\mathbf{k}_j\mid\mathbf{q}_i\right)=k\left(\mathbf{q}_i,\mathbf{k}_j\right)/\sum_l k\left(\mathbf{q}_i,\mathbf{k}_l\right)$和 $k\left(\mathbf{q}_i,\mathbf{k}_l\right)$选择非对称指数核 $\exp\left(\mathbf{q}_i\mathbf{k}_j^{\top}/\sqrt{d}\right)$，自我注意通过计算概率 $p\left(\mathbf{k}_j\mid\mathbf{q}_i\right)$来组合这些值并获得输出。它需要二次乘积运算和 $\mathcal{O}\left(L_QL_K\right)$内存使用量，这是提高预测能力的主要因素。

先前的一些尝试已经揭示了自我注意概率的分布具有潜在的稀疏性，并且他们已经设计了对 $p\left(\mathbf{k}_j\mid\mathbf{q}_i\right)$的“选择性”计数策略而没有显著影响其性能。Sparse transformer（Child et al.2019）包含行输出和列输入，其中稀疏来自这两个部分的空间相关性。Logsparse transformer（Li et al.2019）注意到 self attention 的周期性模式，并迫使每个单元以指数级的步长关注前一个单元。Longformer（Beltagy，Peters，andCohan2020）将之前的两个工作扩展到了更复杂的稀疏配置。然而，这些方法局限于理论分析，只能从启发式的角度出发，用相同的策略来处理每个 multi-head self-attention，这限制了它们的进一步改进。

为了激励我们的方法，我们首先对学习到的注意力模式进行了定性评估。“稀疏”的 self attention 得分形成了一个长尾分布（详见附录 C），也就是说，少数点-乘积对贡献了主要的注意力，而其他点-乘积对则产生了微不足道的注意力。

批注 [W14]: 说明只有一部分 attention 是有用的，要找出有用的 attention，删除没有用的，减少计算量

查询稀疏度度量从式（1）中，第 i 个 query 对所有 key 的关注被定义为概率 $p(\mathbf{k}_j|\ \mathbf{q}_i)$，输出是和 V 的组合，主要点积对促使对应 query 的注意概率分布远离均匀分布。如果 $p(\mathbf{k}_j|\ \mathbf{q}_i)$ 接近于均匀分布 $q(\mathbf{k}_j|\ \mathbf{q}_i)=1/L_K$，那么 self attention 就变成了一个微不足道的值 V 的总和，对住宅输入是多余的。当然，分布 p 和 q 之间的 “相似性 ”可以用来区分 “重要”的 query。我们通过 Kullback-Leibler 散度来衡量 “相似性”

批注 [W15]: 可以通过 p 和 q 的相似性查找重要的 query。
越相似越重要。（p，q 都是关于 query 和 key 的公式）

$$KL(q\|\ p)=\ln\sum_{l=1}^{L_K}e^{\mathbf{q}_i\mathbf{k}_l^T/\sqrt{d}}-\frac{1}{L_K}\sum_{j=1}^{L_K}\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}-\ln L_K\ 。$$

去掉常数，我们将第 i 个查询的稀疏性度量定义为

$$M\left(\mathbf{q}_i,\mathbf{K}\right)=\ln\sum_{j=1}^{L_K}e^{\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}}-\frac{1}{L_K}\sum_{j=1}^{L_K}\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}} \qquad (2)$$

其中第一项是 qi 在所有 key 上的 Log-Sum-Exp(LSE)，第二项是它们的算术平均值。如果第 i 个查询获得较大的 $M\left(\mathbf{q}_i,\mathbf{K}\right)$。它的注意概率 p 更“多样化”，并且在长尾自我注意分布的头部区域中有很大的可能包含占支配地位的点积对。

批注 [W16]: $M\left(\mathbf{q}_i,\mathbf{K}\right)$ 是用来判断 p，q 的相似性的，KL 散度越大，差异越大，说明 attention 包含拖尾。

ProbSparse Self attention 机制 基于提出的方法。我们有 ProbSparse self-attention，允许每个 key 只关注 u 个主要的查询。

$$\mathcal{A}(\mathbf{Q},\mathbf{K},\mathbf{V})=\mathrm{Softmax}\left(\frac{\bar{\mathbf{Q}}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \qquad (3)$$

其中 Q 的平均是一个和 q 相同大小的稀疏矩阵，它只包含稀疏度量 $M\left(\mathbf{q}_i,\mathbf{K}\right)$ 下的 Top-u 个 query。由一个恒定的采样因子 c 控制，我们设置 $u=c\ln LQ$，这使得 Problesparse self-attention 只需要为每个 query-key 查找计算 $\mathcal{O}\left(\ln L_Q\right)$ 点积，并且层内存使用保持 $\mathcal{O}\left(L_K\ln L_Q\right)$。在多头情况下，这种关注为每个头生成不同的稀疏 query-key 对，从而避免了严重的信息损失。

批注 [W17]: Q 的平均是占主要地位的 query，前 U 个 query

然而，对测量 $M\left(\mathbf{q}_i,\mathbf{K}\right)$ 的所有查询的遍历需要计算每个点积对。二次 $\mathcal{O}\left(L_Q L_K\right)$，此外，LSE 操作还有潜在的数值稳定性问题。受此启发。我们提出了一种有效获取 query 稀疏度测量的经验近似方法。

引理 1：

For each query $\mathbf{q}_i \in \mathbb{R}^d$ and $\mathbf{k}_j \in \mathbb{R}^d$ in the keys set $\mathbf{K}$, we have the bound as $\ln L_K \leq M(\mathbf{q}_i,\mathbf{K}) \leq \max_j\{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\} - \frac{1}{L_K}\sum_{j=1}^{L_K}\{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\} + \ln L_K$. When $\mathbf{q}_i \in \mathbf{K}$, it also holds.

根据引理 1(证明在附录 D. 1 中给出)，我们建议最大均值测量为

$$\bar{M}(\mathbf{q}_i,\mathbf{K}) = \max_j\left\{\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\right\} - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}$$

Top-u 的范围在命题 1 的边界松弛中近似成立(参见附录 D.2)。在长尾分布下，我们只需要随机抽取 $U = L_K \ln L_Q$ 点积对来计算 $M\left(\mathbf{q}_i,\mathbf{K}\right)$，即用零来填充其他的点积对。然后，我们从中选择稀疏 Top-u 作为 Q。$M\left(\mathbf{q}_i,\mathbf{K}\right)$ 中的最大算子对零值不太敏感，而且在数值上很稳定。在实际应用中，query 和 key 的输入长度在 self attention 计算中通常是等价的，即 $L_Q = L_K = L$，这样，总的 ProbSparse 的时间复杂度和空间复杂度为 $\mathcal{O}(L \ln L)$。

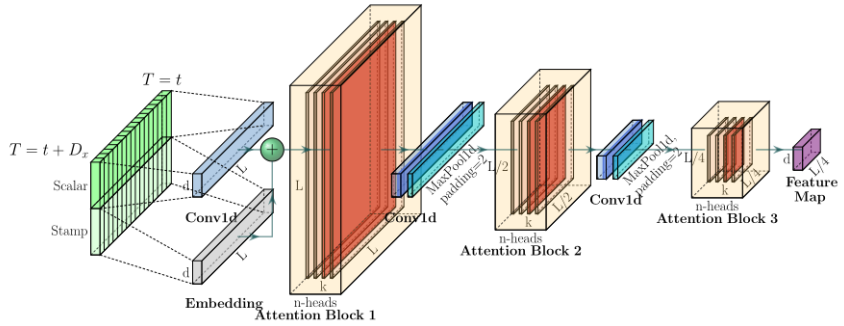


图 3：Informer 编码器中的单个堆栈。(1) Informer 编码器中的单个堆栈(1）水平堆栈表示图（2）中编码器副本中的单个副本。(2) 所提出的一个是接收整个输入序列的主栈。然后，第二个堆栈获取输入的半个切片，随后的堆栈重复。(3)红色层是点积矩阵，通过对每一层进行 self attention distilling，红色层得到级联减少。(4)连接所有堆栈的特征映射作为编码器的输出。

批注 [W18]: 在解决复杂度的过程中，使用的 LSE 还有数值稳定性的问题需要解决。

批注 [W19]: 解决数值稳定性的方法

批注 [W20]: 用零填充不重要的点积对。

批注 [W21]: 编码器简图：通过添加 Conv1d 和 Max pooling 减少网络参数和降维，选取主要的 self attention

编码器：允许在内存使用限制下处理更长的顺序输入

编码器设计用于提取长序列输入的鲁棒长程相关性。在输入表示之后，第 t 个序列输入$\mathcal{X}^t$的形状变成一个矩阵$\mathbf{X}_{\text{en}}^t \in R^{L_x \times d_{\text{madel}}}$。为了清楚起见，我们在图（3）中给出了编码器的简图。

Self-attention Distilling

作为 ProbSparse self-attention 机制的自然结果，编码器的特征映射具有值 V 的冗余（多余的重复或罗嗦内容）组合。在下一层，我们利用提取操作对具有支配特征的优势特征进行特权化，并生成一个聚焦的 self attention 特征图。在图（3）中，我们可以看到注意力区块的权重矩阵（重叠的红色方块），它将输入的时间维度大幅削减。受扩张卷积的启发（Y u, Koltun, and Funkhouser 2017; Gupta 和 Rush 2017），我们的“distilling”程序从第 j 层向第 j 层进入第(j+1)层的过程为。

$$\mathbf{X}_{j+1}^t = MaxPool\left(ELU\left(Conv1d\left(\left[\mathbf{X}_j^t\right]_{\text{AB}}\right)\right)\right) \qquad (5)$$

其中[]代表注意力块。它包含了多头注意和基本操作，其中 Conv1d（·）使用 ELU（·）激活函数（Clevert、Unterthiner 和 Hochreiter2016）在时间维度上执行一维卷积滤波器（内核宽度=3）。我们在叠层后的半片中添加了一个最大池层（max-pooling layer）和步长为 2 的降低采样 Xt，从而将整个内存使用量减少到$\mathcal{O}((2-\epsilon)L\log L)$，其中$\epsilon$ 是一个小数字。

为了增强提取操作的鲁棒性，我们使用减半输入构建主堆栈的副本，并通过一次删除一层（如图（2）中的金字塔）来逐步减少 self-attention distilling 层的数量，从而使其输出维度对齐。因此，我们连接所有堆栈的输出，得到编码器的最终隐藏层表示。

解码器:通过一个正向过程产生长序列输出我们在图(2)中使用了一个标准的解码器结构(Vaswani 等人 2017)，它由两个相同的多头注意力层组成。然而，在长期预测中，采用生成性推理来缓解速度骤降的问题。我们给解码器提供如下向量

$$\mathbf{X}_{\text{de}}^t = \text{Concat}(\mathbf{X}_{\text{token}}^t, \mathbf{X}_{\mathbf{0}}^t) \in \mathbb{R}^{(L_{\text{makn}}+L_y)\times d_{\text{mood}}} \qquad (6)$$

其中$\mathbf{X}_{\text{token}}^t \in \mathbb{R}^{L_{\text{token}}\times d_{\text{model}}}$是开始标记，$\mathbf{X}_0^t \in \mathbb{R}^{L_y \times d_{\text{model}}}$是目标序列的占位符(将标量设置为 0)。将 Masked multi-head attention 应用于 ProbSparse self-attention 计算中，通过将 Masked 点积设置为-∞。它可以防止每个位置都关注未来的位置，从而避免了自回归。一个完全连通的层获得最终的输出，它的 dy 的大小取决于我们是在进行单变量预测还是在进行多变量预测。

批注 [W22]: 减少了网络参数和降维

批注 [W23]: Masked 的应用，遮住未来的信息

批注 [W24]: "自回归"是在**处理时间序列数据时使用的一个统计术语**，它指的是一个与之相关或依赖于，同一变量的先前值。

批注 [W25]: Dy 是什么？

生成推理 开始标记（Start token）在 NLP 的“动态解码”（Devlin et al.2018）中得到了有效应用，我们将其扩展为一种生成方式。我们没有选择特定的标志作为标记，而是在输入序列中采样一个$L_{\text{token?}}$长序列，例如输出序列之前的一个片段。以预测 168 个点为例（实验段 7 天温度预报），我们将已知的目标序列前 5 天作为“起始标记”，并向生成式推理解码器提供以下信息$\mathbf{X}_{\text{de}}=\{\mathbf{X}_{5d},\mathbf{X}_0\}$。$\mathbf{X}_0$ 包含目标序列的时间印记，也就是目标周的上下文。然后，我们提出的解码器预测输出的一个前进的过程，而不是耗时的“动态解码”在传统的编码器-解码器结构

批注 [W26]: 生成式解码器和传统解码器的区别

损失函数我们选择 MSE 损失函数对目标序列进行预测 w.r.t，并将损失从解码器的输出传回整个模型。

Methods		Informer		Informer†		LogTrans		Reformer		LSTMa		DeepAR		ARIMA		Prophet	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTh_1$	24	0.098	0.247	**0.092**	**0.246**	0.103	0.259	0.222	0.389	0.114	0.272	0.107	0.280	0.108	0.284	0.115	0.275
	48	**0.158**	**0.319**	0.161	0.322	0.167	0.328	0.284	0.445	0.193	0.358	0.162	0.327	0.175	0.424	0.168	0.330
	168	**0.183**	**0.346**	0.187	0.355	0.207	0.375	1.522	1.191	0.236	0.392	0.239	0.422	0.396	0.504	1.224	0.763
	336	0.222	0.387	**0.215**	**0.369**	0.230	0.398	1.860	1.124	0.590	0.698	0.445	0.552	0.468	0.593	1.549	1.820
	720	0.269	0.435	**0.257**	**0.421**	0.273	0.463	2.112	1.436	0.683	0.768	0.658	0.707	0.659	0.766	2.735	3.253
$ETTh_2$	24	**0.093**	**0.240**	0.099	0.241	0.102	0.255	0.263	0.437	0.155	0.307	0.098	0.263	3.554	0.445	0.199	0.381
	48	**0.155**	**0.314**	0.159	0.317	0.169	0.348	0.458	0.545	0.190	0.348	0.163	0.341	3.190	0.474	0.304	0.462
	168	**0.232**	**0.389**	0.235	0.390	0.246	0.422	1.029	0.879	0.385	0.514	0.255	0.414	2.800	0.595	2.145	1.068
	336	0.263	**0.417**	**0.258**	0.423	0.267	0.437	1.668	1.228	0.558	0.606	0.604	0.607	2.753	0.738	2.096	2.543
	720	**0.277**	**0.431**	0.285	0.442	0.303	0.493	2.030	1.721	0.640	0.681	0.429	0.580	2.878	1.044	3.355	4.664
$ETTm_1$	24	**0.030**	**0.137**	0.034	0.160	0.065	0.202	0.095	0.228	0.121	0.233	0.091	0.243	0.090	0.206	0.120	0.290
	48	0.069	0.203	**0.066**	**0.194**	0.078	0.220	0.249	0.390	0.305	0.411	0.219	0.362	0.179	0.306	0.133	0.305
	96	0.194	**0.372**	**0.187**	0.384	0.199	0.386	0.920	0.767	0.287	0.420	0.364	0.496	0.272	0.399	0.194	0.396
	288	**0.401**	0.554	0.409	**0.548**	0.411	0.572	1.108	1.245	0.524	0.584	0.948	0.795	0.462	0.558	0.452	0.574
	672	**0.512**	**0.644**	0.519	0.665	0.598	0.702	1.793	1.528	1.064	0.873	2.437	1.352	0.639	0.697	2.747	1.174
Weather	24	**0.117**	**0.251**	0.119	0.256	0.136	0.279	0.231	0.401	0.131	0.254	0.128	0.274	0.219	0.355	0.302	0.433
	48	**0.178**	0.318	0.185	**0.316**	0.206	0.356	0.328	0.423	0.190	0.334	0.203	0.353	0.273	0.409	0.445	0.536
	168	**0.266**	**0.398**	0.269	0.404	0.309	0.439	0.654	0.634	0.341	0.448	0.293	0.451	0.503	0.599	2.441	1.142
	336	**0.297**	**0.416**	0.310	0.422	0.359	0.484	1.792	1.093	0.456	0.554	0.585	0.644	0.728	0.730	1.987	2.468
	720	**0.359**	**0.466**	0.361	0.471	0.388	0.499	2.087	1.534	0.866	0.809	0.499	0.596	1.062	0.943	3.859	1.144
ECL	48	0.239	0.359	0.238	0.368	0.280	0.429	0.971	0.884	0.493	0.539	**0.204**	**0.357**	0.879	0.764	0.524	0.595
	168	0.447	0.503	0.442	0.514	0.454	0.529	1.671	1.587	0.723	0.655	**0.315**	**0.436**	1.032	0.833	2.725	1.273
	336	0.489	0.528	0.501	0.552	0.514	0.563	3.528	2.196	1.212	0.898	**0.414**	**0.519**	1.136	0.876	2.246	3.077
	720	**0.540**	**0.571**	0.543	0.578	0.558	0.609	4.891	4.047	1.511	0.966	0.563	0.595	1.251	0.933	4.243	1.415
	960	**0.582**	**0.608**	0.594	0.638	0.624	0.645	7.019	5.105	1.545	1.006	0.657	0.683	1.370	0.982	6.901	4.264
Count		32		12		0		0		0		6		0		0	

表 1:四个数据集(五个案例)的单变量长序列时间序列预测结果。

4. 实验

数据集

我们在四个数据集上进行了大量实验，包括 2 个收集的 LSTF 真实数据集和 2 个公共基准数据集。

ETT (电力变压器温度)ETT 是电力长期部署中的一个关键指标。我们从中国两个独立的县收集了两年的数据。为了探索 LSTF 问题的间隔尺寸，我们为 1 小时级别创建了{ ETTh1，ETTh2 }单独的数据集，为 15 分钟级别创建了 ETTm1 单独的数据集。每个数据点由目标值“油温”和 6 个功率负载特征组成。训练/验证/测试为 12/4/4 个月。

ECL（用电负荷）3:收集 321 个客户的用电量(Kwh)。由于数据缺失(李等 2019)，我们将数据集转换为 2 年的小时消耗量，并将“MT320”设定为目标值。训练/验证/测试为 15/3/4 个月。

Weather 该数据集包含 2010 年至 2013 年 4 年间近 1600 个美国地点的当地气候数据，数据点每 1 小时收集一次。每个数据点包括目标值“wet bulb”和 11 个气候特征。训练/验证/测试为 28/10/10 个月。

实验细节

我们简要地总结了基础知识，有关网络组件和设置的更多信息见附录 E。

基准线：我们选择了五种时间序列预测方法作为比较，

包括 ARIMA(Ariyo, Adewumi,andAyo2014)。Prophet(TaylorandLetham2018)。LSTMa（Bahdanau,Cho,andBengio2015）。LSTnet（Laietal.2018）和 DeepAR（Flunkert，Salinas，and Gasthaus2017）。）为了更好地探索 ProbSparse self attention 在我们提出的 Informer 中的表现，我们加入了典型的 self attention 变体（Informer†）。高效变体 Reformer（Kitaev,Kaiser,andLevskaya 2019）和实验中最最相关的工作 LogSparse self-attention Lietal.2019）。网络组件的细节在附录 E. 1 中给出。

超参数调节：我们对超参数进行网格搜索，详细范围见附录 E. 3. 。Informer 在编码器中包含一个 3 层堆栈和一个 1 层堆栈(1/4 输入)，以及一个 2 层解码器。我们提出的方法是用 Adam 优化器优化的，它的学习速率从 1e-4 开始，每个时期衰减两倍。epochs 总数为 8，适当提前停止。我们按照建议设置比较方法，batch_size 为 32。

设置：每个数据集的输入都是零均值归一化的。

批注 [W27]: 湿球温度计是指球部（温包）表面保持潮湿状态的温度计。干湿球湿度计的组成部分之一。通常用沾湿的纱布或棉花包在球表面。测量的温度称“湿球温度”。它与周围空气的湿度有关，湿度愈低，湿球上的水分蒸发就愈强，湿球温度愈低，干、湿球温度差就愈大；反之，湿度愈高，干、湿球温度差就愈小。利用这个原理，可制成干湿球湿度计。

批注 [W28]: 可能和使用模型时候的相关设置有关

批注 [W29]: Informer 使用 Adam 优化的，Adam 优化的相关参数。Adam 算法，即一种对随机目标函数执行一阶梯度优化的算法，该算法基于适应性低阶矩估计。Adam 算法很容易实现，并且有很高的计算效率和较低的内存需求

Methods		Informer		Informer†		LogTrans		Reformer		LSTMa		LSTnet	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTh_1$	24	**0.577**	**0.549**	0.620	0.577	0.686	0.604	0.991	0.754	0.650	0.624	1.293	0.901
	48	**0.685**	**0.625**	0.692	0.671	0.766	0.757	1.313	0.906	0.702	0.675	1.456	0.960
	168	**0.931**	**0.752**	0.947	0.797	1.002	0.846	1.824	1.138	1.212	0.867	1.997	1.214
	336	1.128	0.873	**1.094**	**0.813**	1.362	0.952	2.117	1.280	1.424	0.994	2.655	1.369
	720	**1.215**	**0.896**	1.241	0.917	1.397	1.291	2.415	1.520	1.960	1.322	2.143	1.380
$ETTh_2$	24	**0.720**	**0.665**	0.753	0.727	0.828	0.750	1.531	1.613	1.143	0.813	2.742	1.457
	48	**1.457**	**1.001**	1.461	1.077	1.806	1.034	1.871	1.735	1.671	1.221	3.567	1.687
	168	3.489	**1.515**	3.485	1.612	4.070	1.681	4.660	1.846	4.117	1.674	**3.242**	2.513
	336	2.723	1.340	2.626	**1.285**	3.875	1.763	4.028	1.688	3.434	1.549	**2.544**	2.591
	720	**3.467**	**1.473**	3.548	1.495	3.913	1.552	5.381	2.015	3.963	1.788	4.625	3.709
$ETTm_1$	24	0.323	**0.369**	**0.306**	0.371	0.419	0.412	0.724	0.607	0.621	0.629	1.968	1.170
	48	0.494	0.503	**0.465**	**0.470**	0.507	0.583	1.098	0.777	1.392	0.939	1.999	1.215
	96	**0.678**	0.614	0.681	**0.612**	0.768	0.792	1.433	0.945	1.339	0.913	2.762	1.542
	288	**1.056**	**0.786**	1.162	0.879	1.462	1.320	1.820	1.094	1.740	1.124	1.257	2.076
	672	**1.192**	**0.926**	1.231	1.103	1.669	1.461	2.187	1.232	2.736	1.555	1.917	2.941
Weather	24	**0.335**	**0.381**	0.349	0.397	0.435	0.477	0.655	0.583	0.546	0.570	0.615	0.545
	48	0.395	0.459	**0.386**	**0.433**	0.426	0.495	0.729	0.666	0.829	0.677	0.660	0.589
	168	**0.608**	**0.567**	0.613	0.582	0.727	0.671	1.318	0.855	1.038	0.835	0.748	0.647
	336	**0.702**	**0.620**	0.707	0.634	0.754	0.670	1.930	1.167	1.657	1.059	0.782	0.683
	720	**0.831**	**0.731**	0.834	0.741	0.885	0.773	2.726	1.575	1.536	1.109	0.851	0.757
ECL	48	0.344	**0.393**	**0.334**	0.399	0.355	0.418	1.404	0.999	0.486	0.572	0.369	0.445
	168	0.368	0.424	**0.353**	**0.420**	0.368	0.432	1.515	1.069	0.574	0.602	0.394	0.476
	336	0.381	**0.431**	0.381	0.439	**0.373**	0.439	1.601	1.104	0.886	0.795	0.419	0.477
	720	0.406	0.443	**0.391**	**0.438**	0.409	0.454	2.009	1.170	1.676	1.095	0.556	0.565
	960	**0.460**	**0.548**	0.492	0.550	0.477	0.589	2.141	1.387	1.591	1.128	0.605	0.599
Count		33		14		1		0		0		2	

表 2:四个数据集(五个案例)上的多元长序列时间序列预测结果。

在 LSTF 设置下，在{ETTh，ECL，Weather}，{6h，12h，24h，72h，168h}中，我们逐步地延长了预测窗口的大小，即{1d，2d，7d，14d，30d，40d}。**衡量标准:**我们使用两个评价指标，包括 $\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}-\hat{\mathbf{y}})^2$ and $\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}|\mathbf{y}-\hat{\mathbf{y}}|$,在每个预测窗口(多变量预测的平均值)，并以 stride=1 滚动整个集合。平台：所有的模型都在一个 Nvidia V100 32GB GPU 上训练/测试。源代码可在 https://github.com/zhouhaoyi/Informer2020。

批注 [W30]: 设备需求

结果和分析

表 1 和表 2 总结了所有方法在 4 个数据集上的单变量/多变量评价结果。随着对预测能力的更高要求，我们逐渐延长了预测时间，其中 LSTF 问题的设置被精确地控制为每种方法都可以在一个 GPU 上处理。最佳结果以黑体字突出显示。

单变量时间序列预测在这种情况下，每种方法都将预测作为时间序列中的单个变量。从表 1 中，我们可以观察到:(1) 从表 1 可以看出：（1）所提出的模型 Informer 显著提高了所有数据集的推理性能（最后一列中的获胜计数），在不断增长的预测范围内，它们的预测误差平稳而缓慢地上升，这说明 Informer 在提高 LSTF 问题的预测能力方面取得了成功。(2)Informer 击败其典型的降级 informer†主要是在获胜计数，即 32>12，它支持 query 稀疏性假设，提供了一个可比较的注意力特征图。我们提出的方法也执行最相关的工作。我们注意到，Reformer 保持动态解码，在 LSTF 表现不佳，而其他方法受益于生成式解码器作为非自

回归预测。(3) 与递归神经网络 LSTMa 相比，Informer 模型具有更好的结果。我们的方法的均方误差降低了 26.8%（168），52.4%（336）和 60.1%（720）。这表明在自注意机制中，较短的网络路径比基于 RNN 的模型具有更好的预测能力。(4) 该方法在均方误差上优于 DeepAR、ARIMA 和 Prophet，平均分别降低了 49.3%(168)、61.1%(336)和 65.1%(720)。在 ECL 数据集上，在 ECL 数据集上，DeepAR 在较短的范围上表现更好（≤ 336），并且我们的方法在更长的范围上超越了。我们把这归因于一个具体的例子，在这个例子中，预测能力的有效性反映在问题的可伸缩性上。

多元时间序列预测

在此设置中，一些单变量方法是不合适的，而 LSTnet 是最先进的基线。相反，通过调整最终的 FCN（Fully convolutional network）层，我们提出的 Informer 很容易从单变量预测转变为多变量预测。从表 2 中，我们观察到：（1）所提出的模型 Informer 大大优于其他方法，并且在单变量设置下的结果 1 和 2 仍然适用于多变量时间序列(2) 与基于 RNN 的 LSTMa 和基于 CNN 的 LSTnet 相比，Informer 模型的结果更好，MSE 平均下降了 26.6%（168），28.2%（336），34.3%（720）。与单变量预测结果相比，特征维数预测能力的各向异性导致了预测性能的下降。这超出了本文的研究范围，我们将在今后的工作中加以探讨

批注 [W31]: 说明 Informe 还是在 LSTF 问题上更有优势

批注 [W32]: 通过调整全卷积层，可以从单变量预测改成多变量预测

批注 [W33]: 导致预测性能下降的原因，但是不是本文要解决的问题。

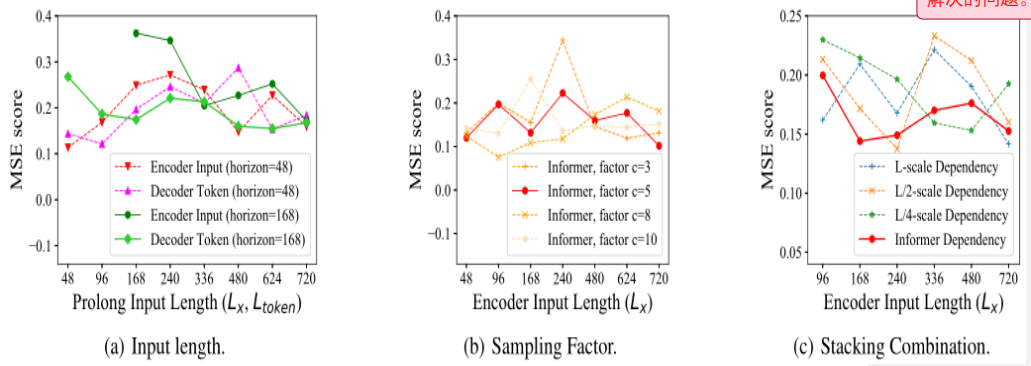


图 4：Informer 中三个组件的参数敏感性。

Prediction length		336			720		
Encoder's input		336	720	1440	720	1440	2880
Informer	MSE	0.249	0.225	0.216	0.271	0.261	0.257
	MAE	0.393	0.384	0.376	0.435	0.431	0.422
Informer†	MSE	0.241	0.214	-	0.259	-	-
	MAE	0.383	0.371	-	0.423	-	-
LogTrans	MSE	0.263	0.231	-	0.273	-	-
	MAE	0.418	0.398	-	0.463	-	-
Reformer	MSE	1.875	1.865	1.861	2.243	2.174	2.113
	MAE	1.144	1.129	1.125	1.536	1.497	1.434

1. Informer 使用规范的自我关注机制。
2. "-"表示内存不足导致失败。

表 3:ProbSparse 自我关注机制的消融研究。

批注 [W34]: 这里看出在较短序列上 Informer+效果更好，

Methods	Training		Testing
	Time	Memory	Steps
Informer	$\mathcal{O}(L \log L)$	$\mathcal{O}(L \log L)$	1
Transformer	$\mathcal{O}(L^2)$	$\mathcal{O}(L^2)$	L
LogTrans	$\mathcal{O}(L \log L)$	$\mathcal{O}(L^2)$	$1^\star$
Reformer	$\mathcal{O}(L \log L)$	$\mathcal{O}(L \log L)$	L
LSTM	$\mathcal{O}(L)$	$\mathcal{O}(L)$	L

1. LSTnet 很难以封闭的形式呈现。
2. 那个*表示应用我们提出的解码器

表 4：每层与 L 有关的计算统计学。

批注 [W35]: Informer 的复杂度更小

考虑到 LSTF 的间隔尺寸

我们进行了一个额外的比较，以探索不同间隔尺寸的性能。ETTm1（分钟级）的序列{96、288、672}与 ETTh1 的序列{24、48、168}对齐（小时级别）。即使序列处于不同的间隔尺寸级别，Informer 的性能也优于其他基线。

批注 [W36]: 调整序列尺寸大小，Informer 的性能也优于其他的

参数的敏感性

我们对提出的 Informer 模型在单变量设置下对 ETTh1 进行敏感性分析。

输入长度：在图（4a）中，当预测短序列（如 48）时，最初增加编码器/解码器的输入长度降低性能，但是进一步增加导致 MSE 下降，因为它带来重复的短期模式。然而，在预测长序列（如 168）时，输入越长，MSE 越低。因为较长的编码器输入可能包含更多的依赖项，并且较长的解码器令牌具有丰富的本地信息。

采样因子：采样因子控制着公式（3）中 ProbSparse self-attention 的信息带宽。我们从小因子(=3)开始到大因子，总体性能略有提高，最后稳定在图(4b)中。它验证了我们的 query 稀疏性假设，即在自我注意机制中存在多余的点积对。我们在实践中设置样本系数 = 5(红线)。

层堆叠的组合：层的复制对于自我注意提取是互补的，我们在图（4c）中研究了每个堆栈{L，L/2，L/4}的行为。堆栈越长，对输入越敏感，部分原因是接收到的长期信息越多。我们的方法的选择（红线），即加入 L 和 L/4，是最稳健的策略。

Ablation study：Informer 的工作情况如何

考虑到 Ablation，我们还对 ETT1 进行了额外的实验。

ProbSparse 自我关注机制的性能

在总体结果表 1 和表 2 中，我们限制了问题设置，以使经典自我注意的记忆使用可行。在本研究中，我们将我们的方法与 LogTrans 和 Reformer 进行了比较，并深入探讨了它们的极限性能。为了排除内存效率问题，我们首先将设置减少为{batch size=8，heads=8，dim=64}，并在单变量情况下保持其他设置。在表 3 中，被试的自我注意表现出比被试更好的表现。

LogTrans 在极端情况下会出现 OOM，因为它的公共实现是掩盖的全注意。它仍然有$\mathcal{O}(L^2)$的内存使用。我们提出的 ProbSparse 自关注从公式(4)中的 query 稀疏性假设带来的简单性中避免了这一点，参考附录 E.2 中的伪代码，并达到较小的内存使用。

Prediction length		336					480				
Encoder's input		336	480	720	960	1200	336	480	720	960	1200
Informer†	MSE	0.249	0.208	0.225	0.199	0.186	0.197	0.243	0.213	0.192	0.174
	MAE	0.393	0.385	0.384	0.371	0.365	0.388	0.392	0.383	0.377	0.362
Informer‡	MSE	0.229	0.215	0.204	-	-	0.224	0.208	0.197	-	-
	MAE	0.391	0.387	0.377	-	-	0.381	0.376	0.370	-	-

1. Informer++消除了 Informer+的自我关注。
2. “-”表示内存不足导致失败。

表 5:自我注意提取的消融研究。

批注 [W37]: 预测长序列时，输入越长越好。

批注 [W38]: 采样因子的设置

批注 [W39]: 实际上 ablation study 就是为了研究模型中所提出的一些结构是否有效而设计的实验。
比如你提出了某某结构，但是要想确定这个结构是否有利于最终的效果，那就要将去掉该结构的网络与加上该结构的网络所得到的结果进行对比，这就是 ablation study

批注 [W40]: OOM 全称"Out Of Memory"，即内存溢出。 内存溢出已经是软件开发历史上存在了近 40 年的“老大难”问题。 在操作系统上运行各种软件时，软件所需申请的内存远远超出了物理内存所承受的大小，就叫内存溢出。

批注 [W41]: Informer 提出的 ProbSparse 解决了 OOM 问题

Prediction length		336					480				
Prediction offset		+0	+12	+24	+48	+72	+0	+48	+96	+144	+168
Informer‡	MSE	0.207	0.209	0.211	0.211	0.216	0.198	0.203	0.203	0.208	0.208
	MAE	0.385	0.387	0.391	0.393	0.397	0.390	0.392	0.393	0.401	0.403
Informer§	MSE	0.201	-	-	-	-	0.392	-	-	-	-
	MAE	0.393	-	-	-	-	0.484	-	-	-	-

1. Informer$$用 Informer++中的动态解码代替了我们的解码器。
2. “-”表示不可接受的度量结果失败

表 6:生成式解码器的消融研究。

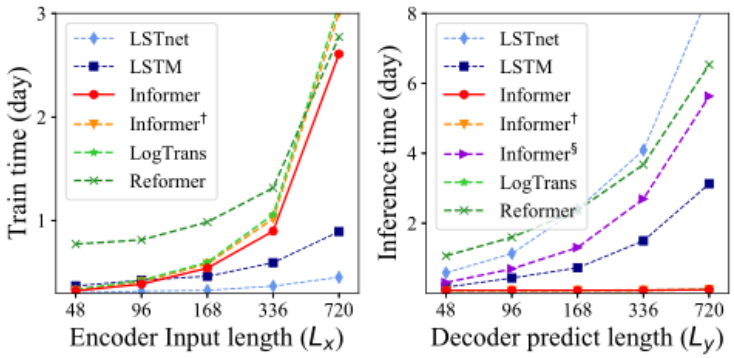


图 5:训练/测试阶段的总运行时间

The performance of self-attention distilling（自我注意蒸馏的表现）

在这项研究中，我们使用“informer+”作为基准来消除预先自我注意的额外影响。另一个实验设置与单变量时间序列设置一致。从表 5 中可以看出，在进行了高级排序后，Informer 已经完成了所有的实验并取得了更好的性能。比较方法 Informer‡删除了蒸馏操作，并在较长的输入（>720）中达到 OOM。关于长序列输入问题的好处，我们得出结论，自我关注提取是值得采用的，特别是当需要长预测时。

生成式解码器的性能

在这项研究中，我们证明了我们的解码器在获得“生成”结果方面的潜在价值。与现有方法不同的是，我们提出的解码器的预测完全依赖于时间戳，时间戳可以用偏移量进行预测。从表 6 可以看出，Informer‡的一般预测性能随着偏移量的增加而抵抗，而对于动态解码，对应的预测性能则失败。证明了该译码器能够捕获任意输出之间的长程依赖关系，避免错误积累。

批注 [W42]: 从图中可以看出，Informer 的训练时间不是最好的，但是解码预测时 Inference time 是最快的。

批注 [W43]: 通俗的讲，时间戳是一**份能够表示一份数据在一个特定时间点已经存在的完整的可验证的数据。** 它的提出主要是为用户提供一份电子证据，以证明用户的某些数据的产生时间。

批注 [W44]: Informer 完全依赖时间戳，可以使用偏移量预测时间戳，在模型中偏移量需要设置嘛?

计算效率

在多变量设置和所有方法的当前最佳实现的情况下，我们在图（5）中进行了严格的运行时比较。在训练阶段，在基于 transformer 的方法中，前者（红线）的训练效率最高。在测试阶段，我们的方法比其他生成式解码方法要快得多。表 4 总结了理论时间复杂度和内存使用的比较。Informer 的性能与运行时实验一致。请注意，LogTrans 侧重于改进自我注意机制，我们在 LogTrans 中应用我们提出的解码器进行公平比较（表 4 中的*）。

5. 结论

本文研究了长序列时间序列的预测问题，提出了一种预测长序列的新方法。具体来说，我们设计了一种 ProbSparse self attention 机制和 distilling 操作来处理普通 transformer 中二次时间复杂度和二次内存使用的挑战。此外，精心设计的生成解码器减轻了传统编码器-解码器架构的限制。在真实数据上的实验证明了该算法对于提高 LSTF 问题的预测能力的有效性。

批注 [W45]: 提出了什么方法解决了什么问题

附录

附录 A 相关工作

下面我们提供一篇关于长序列时间序列预测(LSTF)问题的文献综述。

时间序列预测

现有的时间序列预测方法可以大致分为两类:经典模型和基于深度学习的方法。经典的时间序列模型是时间序列预测的可靠工具，具有吸引人的特性，如可解释性和理论证明(Box 等人，2015 年；Ray 1990)。现代扩展包括对缺失数据的支持(西格等人，2017 年)和多种数据类型(西格、萨利纳斯和不及格，2016 年)。基于深度学习的方法主要通过使用 RNN 和它们的变体来开发序列到序列预测范式，取得突破性性能(Hochreiter 和 Schmidhuber 1997 李等 2018；Y u 等人，2017 年)。尽管取得了长足的进步，但现有的算法仍然不能以令人满意的精度预测长序列时间序列。典型的最新方法(Seeger 等人，2017 年；Seeger，Salinas，and Ballowert 2016)，尤其是深度学习方法(Y u 等人 2017；秦等 2017；奴才，萨利纳斯，加斯特豪斯 2017；Mukherjee 等人，2018 年；文等(2017)，仍然是一种循序渐进的预测范式，存在以下局限性:(i)虽然可以实现一步到位的精确预测，但往往受到动态解码累积误差的影响，导致问题的误差较大(刘等，2019；秦等 2017)。预测精度随着预测序列长度的增加而衰减。(ii)由于消失梯度和记忆约束的问题(Sutskever，Vinyals，Le 2014)，现有的大多数方法不能从时间序列的整个历史的过去行为中学习。在我们的工作中，Informer 的设计旨在解决这两个局限。

批注 [W46]: 长序列时间预测未能很好地解决

批注 [W47]: 深度学习预测方法存在的问题，也是 Informer 要解决的问题。
1.误差累积
2.长序列时间序列输入问题

长序列输入问题

从上述讨论中，我们把第二个限制称为长序列时间序列输入（LSTI）问题。我们将探讨相关的工作，并与我们的 LSTF 问题进行比较。研究人员对输入序列进行截断/总结/取样，以处理非常长的序列。以处理实际中的长序列，但在进行准确预测时可能会丢失有价值的数据。Truncated BPTT（Aicher, Foti, and Fox 2019）没有修改输入，而是只使用最后的时间步骤来估计权重更新中的梯度，Auxiliary Losses（Trinh 等人，2018）通过添加辅助梯度来增强梯度流。其他尝试包括 Recurrent Highway Networks（Zillyetal.2017）和 Bootstrapping Regularizer（Cao and Xu 2019）。这些方法试图改善梯度流在递归网络的长路径中，但随着 LSTI 问题中序列长度的增长，其性能是有限的。基于 CNN 的方法（Stolleretal.2019；Bai, Kolter,和 Koltun2018）使用卷积滤波器来捕捉长时间的依赖性，并且不受影响的领域随着层的堆叠而呈指数级增长，这对这些序列对齐有伤害。在 LSTI 问题中，主要任务是要提高模型接收长序列输入的能力，并从这些输入中提取长程依赖关系。但 LSTF 问题是为了提高模型的的预

测能力。这就需要建立输出和输入之间的长期依赖关系。因此，上述方法对于 LSTF 来说是不可行的。

注意力模型

Bahdanau 等人首先提出了 addictive attention（Bahdanau,Cho,andBengio2015），以改善翻译任务中编码器-解码器架构的单词对齐。然后，其变体（Luong,Pham, and Manning2015）提出了广泛使用的位置、一般和点积注意力。最近，流行的基于 self attention 的 Transformer（Vaswanietal.2017）作为序列建模的新思维被提出，并取得了巨大的成功，尤其是在 NLP 领域。通过将其应用于翻译、语音、音乐和图像生成，更好的序列对齐能力得到了验证。在我们的工作中，Informer 利用其序列对齐的能力，使其适合于 LSTF 问题。

基于 transformer 的时间序列模型

相关工作大多数(宋等 2018；Ma 等人，2019 年；李等人(2019)都是从在时间序列数据中应用 Transformer 的尝试开始的，**并且在预测中失败**，因为他们使用了普通的 Transformer。从其他一些工作中(Child 等 2019；李等人(2019)注意到了 self attention 机制的稀疏性，我们在主要的上下文中讨论了它们。

附录 B 统一输入表示法

RNN 模型(舒斯特和帕利瓦尔 1997；Hochreiter 和 Schmidhuber 1997Chung 等人，2014 年；Sutskever，Vinyals，Le 2014 秦等 2017；Chang 等人(2018 年)通过循环结构本身捕捉时间序列模式，几乎不依赖于时间戳。Vanilla transformer (V aswani 等 2017；Devlin 等人(2018)使用逐点自我注意机制，时间戳作为局部位置上下文。然而，在 LSTF 问题中，捕捉长期独立性的能力需要全局信息，如分层时间戳(周、月和年)和不可知时间戳(假日、事件)。这些很难在经典的 self attention 中得到利用，因此编码器和解码器之间的 query-key 不匹配会导致预测性能的潜在下降。我们提出了一个统一的输入表示来缓解这个问题，图(6)给出了一个直观的概述。

假设我们有第 t 个序列输入 $\mathcal{X}^t$ 和 p 类型的全局时间戳，输入表示后的特征维数为 $d_{\text{model?}}$。

我们首先使用固定位置嵌入来保持本地上下文：

$$\mathrm{PE}_{(\text{pos?}2j)} = \sin\left(pos/\left(2L_x\right)^{2j/d_{\text{model?}}}\right)$$
$$\mathrm{PE}_{(pos,2j+1)} = \cos\left(pos/\left(2L_x\right)^{2j/d_{\text{model?}}}\right) \quad (7)$$

批注 [W48]: LSTF 需要时间戳来捕捉全局信息，以防止不匹配问题，self attention 机制很难做到。因此提出了一种解决方式。

批注 [W49]: Informer 的时间戳使用位置编码 PE 表示

$j \in \{1,\ldots,d_{\text{model?}}/2\}$ 每个全局时间戳都被一个可学习的戳嵌入 $\text{SE}_{(\text{pos})}$ 所采用，其词汇量有限（最多 60 个，即以分钟为最小间隔尺寸）。也就是说，self attention 的相似度计算可以访问全局上下文，并且计算消耗在长输入上是可以承受的。为了调整维度，我们用一维卷积滤波器（kernel width=3, stride=1）将标量上下文 $\mathbf{x}_i^t$ 投影到 $d_{\text{model?}}-\dim$ 向量 $\mathbf{u}_i^t$ 中。因此，我们有一个输入矢量

$$\mathcal{X}_{\text{feed}[i]}^t = \alpha \mathbf{u}_i^t + \text{PE}_{(L_x\times(t-1)+i,)} + \sum_p \left[\text{SE}_{(L_x\times(t-1)+i)}\right]_p \quad (8)$$

其中 i∈{1, . . . , Lx}，而α是平衡标量投影和局部/全局嵌入之间大小的因素。如果序列输入已经被归一化，我们建议α=1。

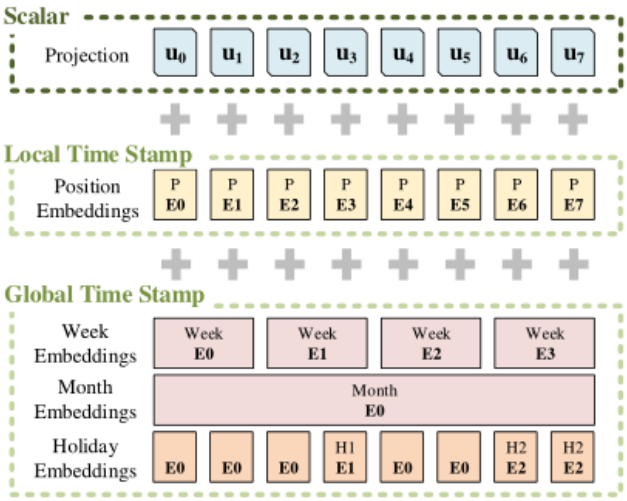


图 6: Informer 的输入表示。输入的嵌入由三个独立的部分组成，标量投影、本地时间戳(位置)和全局时间戳嵌入(分钟、小时、周、月、假日等)。

批注 [W50]: 图 6 中 Scalar 那部分

批注 [W51]: 投影的公式

批注 [W52]: scalar 标量
projection 投影
Local Time Stamp 本地时间戳
Position Embeddings 位置编码
Global Time Stamp 全局时间戳
周/月/节日编码

附录 C self attention 特征图中的长尾分布

我们在 ETTh1 数据集上运行了 vanilla Transformer 来研究自我注意特征图的分布。我们选择{Head1，Head7}@Layer1 的注意力得分。

图（7）中的蓝线形成长尾分布，即少数点积对引起主要注意，其他点积对可以忽略。

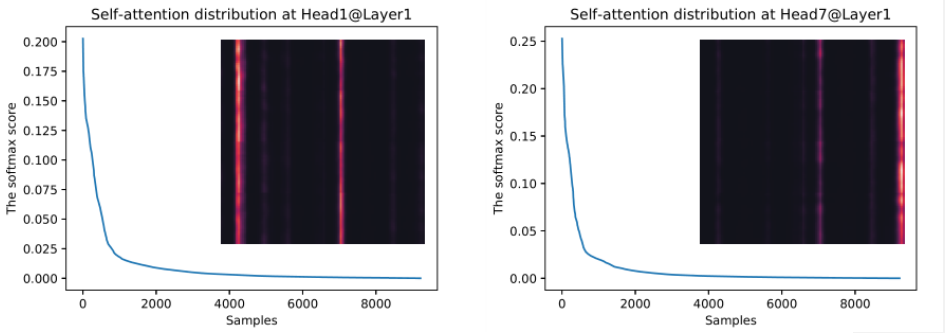


图 7： 在 ETTh1 数据集上训练的 4 层典型转化器的自我注意的 Softmax 得分

附录 D 引理 1 证明细节

证明：对于单个的$\mathbf{q}_i$，我们可以将离散的 key 松弛成连续的 d 维变量，即向量$\mathbf{k}_j$。query 的稀疏度测量被定义为

批注 [W53]: Query 稀疏度测量公式，是凸函数

$$M\left(\mathbf{q}_i,\mathbf{K}\right)=\ln\sum_{j=1}^{L_K}e^{\mathbf{q}i\mathbf{k}_j^\top/\sqrt{d}}-\frac{1}{L_K}\sum_{j=1}^{L_K}\left(\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\right)$$

首先，我们研究一下不等式的左边部分。对于每个查询$\mathbf{q}_i$，$M\left(\mathbf{q}_i,\mathbf{K}\right)$的第一项成为固定查询$\mathbf{q}_i$和所有 key 的内积的对 log sum exp，我们可以定义

$$f_i(\mathbf{K})=\ln\sum_{j=1}^{L_K}e^{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}}$$

根据 Log sum exp 网络（Calafiore、Gaubert 和 Possieri 2018）中的等式（2）和进一步分析，函数$f_i(\mathbf{K})$是凸的。此外，$f_i(\mathbf{K})$加上κ_j的线性组合使得它们$M\left(\mathbf{q}_i,\mathbf{K}\right)$成为固定 query 的凸函数。然后，我们可以对单个向量进行度量的验证

$$\frac{\partial M\left(\mathbf{q}_i,\mathbf{K}\right)}{\partial\mathbf{k}_j}=\frac{c^{q_ik_j^\top/\sqrt{d}}}{\sum_{j=1}^{L_K}e^{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}}}\cdot\frac{\mathrm{q}_i}{\sqrt{d}}-\frac{1}{L_K}\cdot\frac{\mathrm{q}_i}{\sqrt{d}}$$

为了达到最小值，我们让$\vec{\nabla}M\left(\mathbf{q}_i\right)=\vec{0}$并且以下条件被获取为

$$\mathbf{q}_i\mathbf{k}_1^\top+\ln L_K=\cdots=\mathbf{q}_i\mathbf{k}_j^\top+\ln L_K=\cdots=\ln\sum_{j=1}^{L_K}e^{\mathbf{q}_i\mathbf{k}_j^\top}$$

当然，它需要$\mathbf{k}_1=\mathbf{k}_2=\cdots=\mathbf{k}_{L_K}$，我们的最小测量值为$\ln L_K$，

批注 [W54]: 求导之后求得的$M\left(\mathbf{q}_i,\mathbf{K}\right)$最小值

$$M\left(\mathbf{q}_i,\mathbf{K}\right)\geq\ln L_K \tag{9}$$

其次，我们研究不等式的右面部分。如果我们选择最大的内积$\max_j\left\{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\right\}$，很容易

$$\begin{aligned}M\left(\mathbf{q}_i,\mathbf{K}\right)&=\ln\sum_{j=1}^{L_K}e^{\frac{q_ik_j^\top}{\sqrt{d}}}-\frac{1}{L_K}\sum_{j=1}^{L_K}\left(\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\right)\\&\leq\ln\left(L_K\cdot\max_j\left\{\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\right\}\right)-\frac{1}{L_K}\sum_{j=1}^{L_K}\left(\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\right).\\&=\ln L_K+\max_j\left\{\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\right\}-\frac{1}{L_K}\sum_{j=1}^{L_K}\left(\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\right)\end{aligned} \tag{10}$$

结合式（14）和式（15），我们得到引理 1 的结果。当 key 集和 query 集相同时，上述讨论也成立。

命题 1：假设$\mathbf{k}_j\sim\mathcal{N}(\mu,\Sigma)$并且我们让$\mathbf{q}K_i$表示集合$\left\{\left(\mathbf{q}_i\mathbf{k}_j^\top\right)/\sqrt{d}\mid j=1,\ldots,L_K\right\}$

然后$\forall M_m=\max_i M\left(\mathbf{q}_i,\mathbf{K}\right)$存在$\kappa>0$这样：在间隔

$\forall\mathbf{q}_1,\mathbf{q}_2\in\{\mathbf{q}\mid M(\mathbf{q},\mathbf{K})\in[M_m,M_m-\kappa)\}$，如果$\bar{M}(\mathbf{q}_1,\mathbf{K})>\bar{M}(\mathbf{q}_2,\mathbf{K})$

并且$Var\left(\mathbf{q}\mathbf{k}_1\right)>Var\left(\mathbf{q}\mathbf{k}_2\right)$

我们有很大的可能使$M\left(\mathbf{q}_1,\mathbf{K}\right)>M\left(\mathbf{q}_2,\mathbf{K}\right)$

命题 1 的证明

证明：为了简化进一步的讨论，我们可以注意到$a_{i,j}=q_ik_j^T/\sqrt{d}$，因此定义了数组

$A_i=\left[a_{i,1},\cdots,a_{i,L_k}\right]$，此外，我们表示$\frac{1}{L_K}\sum_{j=1}^{L_K}\left(\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\right)=mean\left(A_i\right)$

那么我们可以表示$\vec{M}\left(\mathbf{q}_i,\mathbf{K}\right)=\max\left(A_i\right)-mean\left(A_i\right),i=1,2$

至于$M\left(\mathbf{q}_i,\mathbf{K}\right)$，我们表示每个组件$a_{i,j}=mean\left(A_i\right)+\Delta a_{i,j},j=1,\cdots,L_k$

那么我们有以下内容

$$
\begin{aligned}
M\left(\mathbf{q}_i,\mathbf{K}\right) &= \ln\sum_{j=1}^{L_K} e^{\mathbf{q}_i\mathbf{k}_j^{\top}/\sqrt{d}} - \frac{1}{L_K}\sum_{j=1}^{L_K}\left(\mathbf{q}_i\mathbf{k}_j^{\top}/\sqrt{d}\right) \\
&= \ln\left(\Sigma_{j=1}^{L_k} e^{mean(A_i)}e^{\Delta a_{i,j}}\right) - mean\left(A_i\right) \\
&= \ln\left(e^{mean(A_i)}\Sigma_{j=1}^{L_k} e^{\Delta a_{i,j}}\right) - mean\left(A_i\right) \\
&= \ln\left(\Sigma_{j=1}^{L_k} e^{\Delta a_{i,j}}\right)
\end{aligned}
$$

而且很容易找到$\Sigma_{j=1}^{L_k}\Delta a_{i,j}=0$

我们定义了函数$ES\left(A_i\right)=\Sigma_{j=1}^{L_k}\exp\left(\Delta a_{i,j}\right)$，等价地定义了$A_i=\left[\Delta a_{i,1},\cdots,\Delta a_{i,L_k}\right]$，紧接着我们的命题可以写成等价形式：

For $\forall A_1, A_2$, if

1. $\max\left(A_1\right)-mean\left(A_1\right)\geq\max\left(A_2\right)-mean\left(A_2\right)$
2. $Var\left(A_1\right)>Var\left(A_2\right)$

然后我们把原来的结论改写成更一般的形式，即$ES\left(A_1\right)>ES\left(A_2\right)$有更高的概率，概率与$Var\left(A_1\right)-Var\left(A_2\right)$正相关。

此外，我们考虑一个很好的例子$\forall M_{\mathrm{m}}=\max_i M\left(\mathbf{q}_i,\mathbf{K}\right)$，there exist$\kappa>0$，使得在该间隔中$\forall\mathbf{q}_i,\mathbf{q}_j\in\{\mathbf{q}\mid M(\mathbf{q},\mathbf{K})\in[M_m,M_m-\kappa)\}\ if\ \max(A_1)-\mathrm{mean}(A_1)\geq\max(A_2)-\mathrm{mean}(A_2)\ and\ \mathrm{Var}(A_1)>\mathrm{Var}(A_2)$，我们有大概率$M\left(\mathbf{q}_1,\mathbf{K}\right)>M\left(\mathbf{q}_2,\mathbf{K}\right)$，相当于等式$ES\left(A_1\right)>ES\left(A_2\right)$。

在原命题中，$\mathbf{k_j}\sim\mathcal{N}(\mu,\Sigma)$遵循多元高斯分布，这意味着$k_1,\cdots,k_n$是一维高斯分布，因此定义为期望为 0 的一维高斯分布。由此定义的维纳-辛钦大数定律，$a_{i,j}=q_i k_j^T/\sqrt{d}$是一维高斯分布，期望为 0 如果$n\to\infty$的情况下。回到我们的定义$\Delta a_{1,m}\sim N\left(0,\sigma_1^2\right),\Delta a_{2,m}\sim N\left(0,\sigma_2^2\right),\forall m\in 1,\cdots,L_k$，我们的命题等价于对数正态分布和问题。

对数正态分布和问题相当于精确地逼近$ES\left(A_1\right)$的分布，其历史在文章（Dufresne2008），

（Vargasguzman 2005）中有很好的介绍。逼近对数正态和的对数正态性是众所周知的经验法则，对于对数正态和不能给出一般的 PDF 函数。然而，(Romeo，Da Costa，和 Bardou 2003) 和(Hcine 和 Bouallegue 2015) 指出在大多数情况下，对数正态和仍然是对数正态分布，并应用中心极限定理证明了这一点，我们可以很好地近似 $ES\left(A_1\right)$ 是对数正态分布，并且我们有 $E\left(ES\left(A_1\right)\right)=ne^{\frac{\sigma_1^2}{2}}$ ，

$$Var\left(ES\left(A_1\right)\right)=ne^{\sigma_1^2}\left(e^{\sigma_1^2}-1\right).?\text{Equally},?E\left(ES\left(A_2\right)\right)=ne^{\frac{\sigma_2}{2}},Var\left(ES\left(A_2\right)\right)=ne^{\sigma_2^2}\left(e^{\sigma_2^2}-1\right)$$

我们 $B_1=ES\left(A_1\right),B_2=ES\left(A_2\right)$，概率 $\Pr\left(B_1-B_2>0\right)$ 是总体条件下目标的最终结果，有 $\sigma_1^2>\sigma_2^2$ WLOG。对数正态分布的差异仍然是一个很难解决的问题。

利用(Lo 2012)中给出的定理，给出了对数正态分布和差的概率分布的一般近似。即 S1 和 S2 是服从随机微分方程的两个对数正态随机变量 $\frac{dS_i}{S_i}=\sigma_i dZ_i, i=1,2$，其中 $dZ_{1,2}$ 分别表示与 $S_{1,2}$ 相关的标准韦纳过程，and $\sigma_i^2=\text{Var}(\ln S_i), S^{\pm}\equiv S_1\pm S_2, S_0^{\pm}\equiv S_{10}\pm S_{20}$。至于联合概率分布函数 $P\left(S_1,S_2,t;S_{10},S_{20},t_0\right)$，时刻 t>t0 的 s1 和 S2 的值由它们在初始时刻 t0 的初始值 S10 和 S20 提供。上面的 Weiner 过程等价于对数正态分布（Weiner 和 Solbrig 1984），下面的结论是用包含对数正态分布近似的和和和差的一般形式来表示的± 求和+和差− 分别。在边界条件下

$$\bar{P}_{\pm}(S^{\pm},t;S_{10},S_{20},t_0\rightarrow t)=\delta(S_{10}\pm S_{20}-S^{\pm})$$

它们的封闭形式概率分布函数由下式给出

$$\begin{aligned} & f^{\text{LN}}\left(\tilde{S}^{\pm},t;\tilde{S}_0^{\pm},t_0\right) \\ = & \frac{1}{\tilde{S}^{\pm}\sqrt{2\pi\tilde{\sigma}_{\pm}^2(t-t_0)}} \\ \cdot & \exp\left\{-\frac{\left[\ln\left(\tilde{S}^{+}/\tilde{S}_0^{+}\right)+(1/2)\tilde{\sigma}_{\pm}^2(t-t_0)\right]^2}{2\tilde{\sigma}_{\pm}^2(t-t_0)}\right\} \end{aligned}$$

它是一个近似正态分布，并且 $\tilde{S}^{+},\tilde{S}^{-}$ 是对数正态随机变量，$\tilde{S}_0^{\pm}$ 是上面我们定义的初始条件。(注意，为了使该近似值有效，$\tilde{\sigma}_{\pm}^2(t-t_0)$ 应该很小。在我们的模拟实验中，我们设置 $t-t_0=1$

WLOG)从 $\tilde{S}_0^- = (S_{10} - S_{20}) + \left(\frac{\sigma_-^2}{\sigma_1^2 - \sigma_2^2}\right)(S_{10} + S_{20})$，和 $\begin{matrix}\sigma_- = (\sigma_1^2 - \sigma_2^2)/(2\sigma_-) \\ \sigma_- = \sqrt{\sigma_1^2 + \sigma_2^2}\end{matrix}$

注意到 $E(B_1) > E(B_2)$，$Var(B_1) > Var(B_2)$，近似正态分布的均值和方差与 $\sigma_1^2 - \sigma_2^2$ 呈正相关。另外，闭合式 PDF（概率密度函数） $f^{LN}(\tilde{S}^{\pm}, t; \tilde{S}_0^{\pm}, t_0)$也显示与 $\sigma_1^2 - \sigma_2^2$ 正相关。由于 $\tilde{\sigma}_{\pm}^2(t - t_0)$应足够小的限制，这种正相关性在实验室实验中并不显著。

通过使用（Lo 2012）中的 Lie Trotter 算子分割法。我们可以给出 B1-B2 分布的说明性数字例子，在实际 LLLT 实验中，这些参数被很好地选择来适应我们的近似值 top-u 实验。图中显示，当 $\sigma_1^2 > \sigma_2^2$ 时，不等式成立的概率很高，即 B1>B2， $ES(A_1) > ES(A_2)$

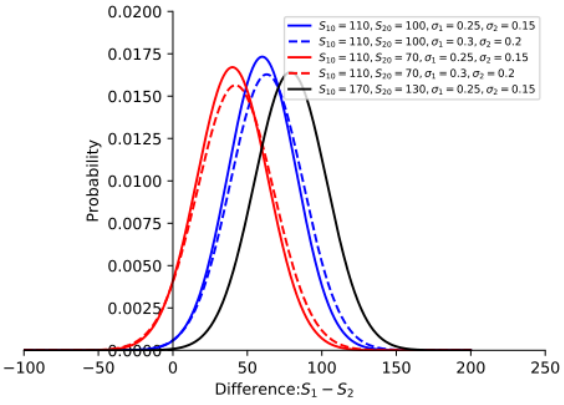


图 8：移位对数正态分布近似的概率密度

在给出一般条件下的命题之后，我们可以考虑一个更具体的条件，如果

$\mathbf{q}_1, \mathbf{q}_2 \in \{\mathbf{q} |\ M(\mathbf{q}, \mathbf{K}) \in [M_m, M_m - \kappa)\}$，这个命题仍然很有可能成立。

首先，我们有 $M(q_1, \mathbf{k}) = \ln(B_1) > (M_m - \kappa)$保持$\forall q_1, q_2$在这个区间。既然我们已经证明 $E(B_1)) = ne^{\frac{\sigma_i^2}{2}}$，我们可以得出结论$\forall q_i$在给定的时间间隔内，$\exists \alpha, \sigma_i^2 > \alpha, i = 1, 2$。既然我们有 $\tilde{S}_0^- = (S_{10} - S_{20}) + \left(\frac{\sigma_-^2}{\sigma_1^2 - \sigma_2^2}\right)(S_{10} + S_{20})$，这也显示出正相关 $\sigma_1^2 + \sigma_2^2 > 2\alpha$，并且与 $\sigma_1^2 - \sigma_2^2$ 正相关。所以由于近似正态分布 PDF 的性质，if $\sigma_1^2 > \sigma_2^2$ WLOG，$\Pr(M(q_1, \mathbf{k}) > M(q_2, \mathbf{k})) \approx \Phi\left(\frac{\tilde{S}_0^-}{\sigma_-}\right)$也与 $\sigma_1^2 + \sigma_2^2 > 2\alpha$ 呈正相关。

我们在图(8)中给出了上述近似的示例性数值例子。在我们实际的 LTTnet 实验中，我们选择 A1、A2 的 Top-k，而不是整个集合。实际上，我们可以做一个天真的假设 A1、A2 的 $top-[\frac{1}{4}L_k]$ 表示为 $A_1^{'},A_2^{'}$，变量 σ_1,σ_2 不会发生显著变化，但是预期 $E\left(A_1^{'}\right),E\left(A_2^{'}\right)$ 会明显上升，这导致初始条件 S_{10},S_{20} 显著上升，因为初始条件将从顶部 $top-[\frac{1}{4}L_k]$ 变量开始，而不是整个集合。

在我们实际的 LTTnet 实验中，我们设置了 U，即选择大约 $top-[\frac{1}{4}L_k]$ 的 A1 和 A2，它可以保证在 $\left[M_m,M_m-\kappa\right)$ 区间的概率超过 99%，如图（8）的黑色曲线所示。通常情况下，条件 2 可以放宽，我们可以相信，如果 q1，q2 符合我们命题中的条件 1，我们有 $M\left(q_1,\mathbf{k}\right)>M\left(q_2,\mathbf{k}\right)$。

附录 E 再现性

实验的细节

表 7 总结了提出的 Informer 模型的细节。对于 ProbSparse 的自我注意机制，我们将 d=32,n=16，并增加了残余连接，位置前馈网络层（内层尺寸为 2048）和 dropout 层（p=0.1）也是如此。请注意，我们为每个数据集保留 10%的验证数据，所以所有的实验都是在 5 个随机训练/验证移位的过程中进行的。因此，所有的实验都是在 5 次随机的 训练/验证 随时间的移动选择上进行的，结果是 5 次运行的平均值。所有的数据集都进行了标准化处理，如使得变量的平均值为 0，标准差为 1。

表 7:详细的 informer 网络组件

Encoder:			N
Inputs	1x3 Conv1d	Embedding ($d=512$)	
ProbSparse Self-attention Block	Multi-head ProbSparse Attention ($h=16, d=32$)		4
	Add, LayerNorm, Dropout ($p=0.1$)		
	Pos-wise FFN ($d_{\text{inner}}=2048$), GELU		
	Add, LayerNorm, Dropout ($p=0.1$)		
Distilling	1x3 conv1d, ELU		
	Max pooling (stride $=2$)		
Decoder:			N
Inputs	1x3 Conv1d	Embedding ($d=512$)	
Masked PSB	add Mask on Attention Block		2
Self-attention Block	Multi-head Attention ($h=8, d=64$)		
	Add, LayerNorm, Dropout ($p=0.1$)		
	Pos-wise FFN ($d_{\text{inner}}=2048$), GELU		
	Add, LayerNorm, Dropout ($p=0.1$)		
Final:			
Outputs	FCN ($d=d_{\text{out}}$)		

ProbSparse self-attention 的实现

我们已经用 Pytorch 1.0 在 Python 3.6 中实现了 ProbSparse self-attention。伪代码用 Algo(1)给出.源代码可在 https://github.com/周浩义/Informer2020 获得。所有的程序都可以高效的进行操作和维护。masked 版可以通过在步骤 6 上应用 positional mask 和在步骤 7 的 mean(·) 中使用 cmusum(·) 来实现。在实践中，我们可以使用 sum(·) 作为 mean(·) 的简单实现。

批注 [W55]: 下表中的步骤 6 和步骤 7

Algorithm 1 ProbSparse self-attention

Require: Tensor $\mathbf{Q} \in \mathbb{R}^{m \times d}, \mathbf{K} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{n \times d}$

1: **print** set hyperparameter c, $u = c \ln m$ and $U = m \ln n$
2: randomly select U dot-product pairs from $\mathbf{K}$ as $\bar{\mathbf{K}}$
3: set the sample score $\bar{\mathbf{S}} = \mathbf{Q}\bar{\mathbf{K}}^{\top}$
4: compute the measurement $M = \max(\bar{\mathbf{S}}) - \text{mean}(\bar{\mathbf{S}})$ by row
5: set Top-u queries under M as $\bar{\mathbf{Q}}$
6: set $\mathbf{S}_1 = \text{softmax}(\bar{\mathbf{Q}}\mathbf{K}^{\top}/\sqrt{d}) \cdot \mathbf{V}$
7: set $\mathbf{S}_0 = \text{mean}(\mathbf{V})$
8: set $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_0\}$ by their original rows accordingly

Ensure: self-attention feature map $\mathbf{S}$.

超参数调谐范围

对于所有方法，对于 ETTh1、ETTh2、天气和电力数据集，经常性分量的输入长度从{24、48、96、168、336、720}中选择；对于 ETTm 数据集，从{24、48、96、192、288、672}中选择。对于 LSTMa 和 DeepAR，**隐藏状态**的大小从{32，64，128，256}中选择。对于 LSTnet，递归层和卷积层的隐藏维度从递归跳过层的{64，128，256}和{32，64，128}中选择，对于 ETTh1、ETTh2、天气和 ECL 数据集，递归跳过层的跳过长度设置为 24，对于 ETTm 数据集，设置为 96。对于 Informer，编码器层从{6，4，3，2}中选择，解码器层设置为 2。多头注意力的头数从{8，16}中选择，多头注意力输出的维度设置为 512。编码器的输入序列长度和解码器的起始标记从{24，48，96，168，336，480，720}中选择，用于 ETTh1、ETTh2 天气和 ECL 数据集，而{24，48，96，192，288，480，672}用于 ETTm 数据集。在实验中，解码器的起始标记是从编码器的输入序列中截取的一段，因此解码器的起始标记的长度必须小于编码器的输入长度。

批注 [W56]: Informer 的编码器，解码器和多头的设置

基于 RNN 的方法在预测窗口上执行具有左移的动态解码。我们提出的方法 Informer-series 和 LogTrans(我们的解码器)执行非动态解码。

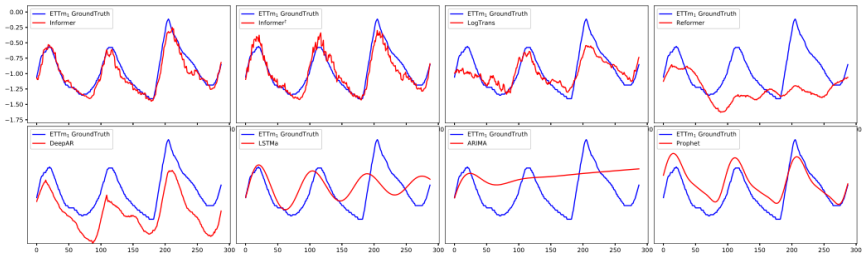


图 9：在 ETTm 数据集上预测(len=336)Informer、Informer+、LogTrans、Reformer、DeepAR、LSTMa、ARIMA 和 Prophet。红色/蓝色曲线代表预测/基本事实的片段。

附录 F 额外的实验结果

图（9）显示了 8 个模型的预测一部分。最真实的工作 LogTrans 和 Reformer 显示可接受的结果。LSTMa 模型不适用于长序列预测任务。ARIMA 和 DeepAR 可以捕捉长序列的长趋势。Prophet 探测到了变化点，并用比 ARIMA 和 DeepAR 更好的平滑曲线来拟合它。我们提出的模型通知和前者显示了比上述方法更好的结果。Prophet 检测到了变化点，并以平滑的曲线进行拟合。比 ARIMA 和 DeepAR 更好。我们提出的模型 Informer 和 Informer†显示出比上述方法更好的结果。

附录 G 计算基础设施

所有实验都是在 Nvidia Tesla V100 SXM2 GPUs (32GB memory)上进行的。其他配置包括 2*Intel Xeon Gold 6148 CPU, 384GB DDR4 RAM and 2*240GB M.2SSD，这是所有组件的有效配置。

参考文献(和时间序列预测相关的文献)

[1]Chang, Y .-Y .; Sun, F.-Y .; Wu, Y .-H.; and Lin, S.-D. 2018.A Memory-Network Based Solution for Multivariate Time-Series Forecasting.arXiv:1809.02105.

[2]Liu, Y .; Gong, C.; Y ang, L.; and Chen, Y . 2019. DSTP-RNN:a dual-stage two-phase attention-based recurrent neural net-works for long-term and multivariate time series prediction. CoRRabs/1904.07464.

[3]Qin, Y .; Song, D.; Chen, H.; Cheng, W.; Jiang, G.; and Cot-trell, G. W. 2017. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. InIJCAI 2017,2627–2633.

[4]Papadimitriou, S.; and Y u, P . 2006. Optimal multi-scale pat-terns in time series streams. InACM SIGMOD 2006, 647–658. ACM.

[5]Ray, W. 1990. Time series: theory and methods.Journal of the Royal Statistical Society: Series A (Statistics in Society) 153(3): 400–400.

[6]Y u, R.; Zheng, S.; Anandkumar, A.; and Y ue, Y . 2017. Long-term forecasting using tensor-train rnns.arXiv:1711.00073