上海财经大学

毕业论文

论文题目：时间序列 Transformer 变体

——概率建模和 Informer

姓名：韩修齐

学号：2020110868

学院：统计与管理学院

专业：经济统计学（金融统计与风险管理）

指导教师：贺莘

定稿日期：2024 年 5 月 18 日

声　　明

本人郑重声明所呈交的论文是我个人在指导老师的指导下进行的研究工作及取得的研究成果，不存在任何剽窃、抄袭他人学术成果的现象。我同意（√）/不同意（　）本论文作为学校的信息资料使用。

论文作者（签名）韩修齐

2024 年 5 月 28 日

时间序列 Transformer 变体——概率建模和 Informer

摘 要

本文针对时间序列预测问题，提出了对 Transformer 模型及其变体在时间序列预测中应用的深入研究。本文细致地展示了 Transformer 模型的核心机制，包括其组成部分和运行过程；随后探究其进行时间序列预测任务训练的原理，深入研究了其在该任务上的两种变体 Probabilistic Transformer 和 Informer 模型。通过在实际的发电机油温数据集上进行实验，本文评估了这些模型在短序列和长序列预测任务上的性能。结果表明，Probabilistic Transformer 能有效处理短序列预测，捕获预测的不确定性；而 Informer 模型在长序列预测任务上表现出速度和效果优势，解决了原始 Transformer 在处理长序列时的计算和存储挑战。本研究不仅扩展了 Transformer 模型在时间序列预测领域的应用，也为未来的优化提供了新的思路和方法。

关键词：时间序列预测，Transformer，Probabilistic Transformer，Informer

Abstract

Generally, this paper presents an in-depth study on the application of the Transformer model and its variants to time series forecasting problems. It begins by outlining the core mechanisms and advantages of the Transformer model, then explains the principles of its training for time series forecasting tasks, and introduces two variants: the Probabilistic Transformer and the Informer model. Experiments conducted on an actual generator oil temperature dataset assess these models' performance on short and long sequence forecasting tasks. The results indicate that the Probabilistic Transformer effectively handles short sequence forecasting, capturing the uncertainty of predictions; meanwhile, the Informer model demonstrates speed as well as accuracy advantages in long sequence forecasting tasks, addressing the computational and storage challenges of the original Transformer in processing long sequences. This study not only extends the application of the Transformer model in the field of time series forecasting but also provides new ideas and methods for future research.

Key Words: Time Series Forecasting, Transformer, Probabilistic Transformer, Informer

符号说明

D_{model}	隐藏层维度或特征数
K	键
Q	查询
V	值
X	原始数据矩阵
W	全连接层权重矩阵
d_k	键的维度
$Attention(Q,K,V)$	缩放点积注意力
b	偏置项
$ReLU$	激活函数
D_k	多头注意力切割后 Key 和 Query 的维度
D_v	多头注意力切割后 Value 的维度
h	多头注意力头数
$X_{pe}, X_{MHA}, X_{EL1}, X_{FF}, X_{EL2}, X_{EL2}^{(n)}$	编码器各组件输出
$X_{pe-s}, X_{MHA-s}, X_{DL1}, X_{MHA2-s}, X_{DL2}, X_{FF-s}, X_{DL3}, X_{DL3}^{(n)}$	解码器各组件输出
KL	KL 散度
q, p	KL 散度比较的两个分布的概率
N	时间序列样本数
n	编码器/解码器层数
C	原始时间序列特征数

目录

一、引言

（一）研究背景及意义

时间序列数据是各个领域中普遍存在的数据类型，从金融市场的股票价格、气象站的天气预报，到工业生产中的传感器数据等，时间序列数据无处不在。传统的时间序列分析方法，如 ARIMA 模型和指数平滑模型等，虽然在某些场景中表现良好，但它们通常假设数据是平稳的或遵循特定的统计分布，这限制了它们在处理复杂或非线性时间序列数据时的效果。近年来，随着深度学习技术的发展，基于神经网络的时间序列分析方法逐渐兴起，特别是 Transformer 模型由于其优越的序列处理能力而备受关注。Transformer 模型最初用于自然语言处理领域，其自注意力机制能够有效捕捉序列数据中的长距离依赖关系，这使得变压器模型在处理时间序列数据时显示出巨大的潜力。此外，Transformer 模型的自注意力机制为时间序列数据提供了一种新的表征方式，这在多变量时间序列分析中尤为重要，因为它可以同时处理并发现不同时间序列之间的相互关系。透彻剖析 Transformer 架构及其时序模型变体，将 Transformer 应用于时间序列分析，不仅可以提升模型对数据中复杂模式的学习能力，还可以为未来的研究进行铺垫，利于确定优化方向。

（二）国内外研究现状及文献综述

Transformer 模型，首次由 Vaswani 等（2017）在论文《Attention Is All You Need》中提出，标志着自然语言处理(NLP)领域的一个重大转折点。与以往依赖于循环神经网络(RNN)和卷积神经网络(CNN)的模型不同，Transformer 完全基于注意力机制，极大提高了模型处理长距离依赖关系的能力，并显著减少了训练时间。长期以来，RNN 和其变体，如长短期记忆网络(LSTM)和门控循环单元(GRU)，一直是序列建模和转换任务的主流方法。然而，这些模型因其固有的顺序计算特性而面临并行化难题，限制了训练速度和模型性能的进一步提升。Transformer 模型通过引入自注意力机制(self-attention)和位置编码(positional encoding)来捕捉序列内的依赖关系，无需依赖于序列的时间步骤计算。此外，Transformer 采用了编码器-解码器架构，其中包含多头注意力机制(multi-head attention)，不仅加强了模

型对不同位置信息的捕获能力，还提升了并行处理的效率。在 Transformer 中，注意力机制允许模型动态地聚焦于序列中的重要部分，而忽略不相关的信息。这一机制通过计算查询(query)、键(key)和值(value)之间的关系来实现，极大地增强了模型处理复杂语义关系的能力。Transformer 模型在训练过程中采用了诸如标签平滑(label smoothing)、残差连接(residual connections)和层归一化(layer normalization)等技术来提升模型的泛化能力和稳定性。经过广泛的实验验证，Transformer 在多个 NLP 任务上（如机器翻译和语法分析）取得了极佳的性能表现。此后，研究人员为 Transformer 设计了各种变体，充分发挥了编码器和解码器的性能。例如，Devlin 等（2018）设计了 BERT，利用了 Transformer 中的解码器部分，并为不同的下游任务作了架构上的微调；Radford 等（2018）提出了 GPT，利用了 Transformer 的解码器；Lewis 等（2019）设计的 BART 则综合利用了编码器和解码器。

Wen 等（2023）提出，Transformer 的多个优势中，较为突出的一点在于其捕获长距离依赖关系和交互的能力，这为时间序列建模带来了令人兴奋的进展，涵盖了各种时间序列应用。这些时间序列应用包含几个重要任务：预测、异常检测和分类。但是，直接将 Transformer 应用于时间序列预测面临一些挑战，主要是由于时间序列的特定特性，如周期性、趋势以及潜在的非线性和非平稳性。为了解决这些挑战，研究人员提出了多种 Transformer 变体，旨在更好地捕捉时间序列数据的复杂动态。Transformer 的时间序列预测任务变体主要包含两大类：模块级（Module-level）变体和架构级（Architecture-level）变体。模块级变体聚焦于对 Transformer 中的特定组件进行调整或改进，以适应时间序列预测的需求。这类变体通过修改原始 Transformer 模型的内部模块（如注意力机制、位置编码等）来增强其处理时间序列的能力。这类变体可以归结为 3 种类型：第一是注意力机制的改进，通过引入更高效或特定于任务的注意力机制来提升模型性能，改进的自注意力机制可以更好地捕获长期依赖关系，比较典型的是 Li（2019）提出的 LogTrans 和 Zhou（2022）提出的 FEDformer，此外 Liu 等（2021）设计了 Pyraformer 分层金字塔注意力模块，利用二叉树路径，以线性时间和内存复杂度捕获不同范围的时间依赖性，Chen 等（2022）的 Quatformer 提出了基于四元数的学习旋转注意力（LRA），引入了可学习的周期和相位信息来描述复杂的周期性模式；第二是对数据归一化方式的修改，例如 Liu（2022）提出的 Non-stationary Transformer，通过插入序列平稳化和非平稳化插件，解决了过度平稳化的问题，辅助提升注意力机制的额性能；此外还有输入表示的优化，这是由于考虑到了时间序列数据的特性，如趋势和季节性，这些变体通过改进输入数据的表示方法（如嵌入层）来增强模型对这些特性的捕获能力，例如 Wu（2021）提出的 Autoformer，以及 Nie（2023）的 PatchTST ，将时间序列分配给独立通道并共享嵌入。架构级变体涉及对 Transformer 整体架构的调整，

创建了专门为时间序列预测设计的新模型。这些变体通常引入了全新的架构概念，或者对原有 Transformer 架构进行了重大修改，以更好地适应时间序列数据的特点。为了捕获时间序列数据中存在的不同时间尺度的依赖关系，一些变体采用了多尺度或层次化的架构，允许模型同时捕获短期和长期的时间动态；考虑到长时间序列的计算复杂度，一些变体通过引入稀疏性注意力机制来降低模型的计算负担，这使得模型能够有效处理更长的序列，同时保持较高的预测精度；此外为了更好地利用时间序列数据中的历史和未来信息，一些变体采用了双向或流式处理模式，这种设计使得模型在预测时能够考虑到整个时间序列的上下文信息。本文关注模块级变体中另外两个极富代表性的例子。

经典方法（例如传统的 ARIMA 时间序列模型）针对数据集中的每个时间序列单独拟合，通常被称为“单个”或“局部”方法。Rogge 等（2022）提出，对于某些应用场景，当处理大量时间序列时，训练一个“全局”模型以覆盖所有可用时间序列变得非常有益，这使模型能够从许多不同来源学习潜在表示。经典方法是点值的（意味着，它们每个时间步只输出一个单一值），模型通过最小化与真实数据的 L2 或 L1 类型损失来训练。然而，由于预测通常用于现实世界的决策制定流程中，甚至涉及人工操作，提供预测的不确定性将更有益——这被称为“概率预测”，与“点预测”相对。这涉及到建模概率分布，并从中进行采样。由此 Rogge 等（2022）设计了全局概率模型——Probabilistic Time Series Transformer。在概率设置中，常见的做法是学习一些选定的参数分布的未来参数，如高斯或学生 T 分布；或者学习条件分位数函数；或者使用适应时间序列设置的共形预测框架。通过取经验均值或中位数即可将概率模型转换为点预测模型。神经网络可以从几个相关时间序列中学习表示，以及模拟数据的不确定性——这决定了使用 Transformer 进行概率建模的适用性。编码器-解码器架构在推理时很有帮助，其中通常希望对一些记录的数据进行预测，预测未来的一些步骤。这可以被认为类似于文本生成任务，给定一些上下文，采样下一个标记并将其反馈给解码器（也称为“自回归生成”）。同时利用贪婪采样/搜索，给定一些分布类型并从中采样以提供预测，直到我们期望的预测范围。其次，Transformer 有助于训练可能包含成千上万时间点的时间序列数据。尽管注意力机制的时间和内存约束致使无法一次性将时间序列的所有历史输入模型，可行的方法是考虑一些适当的上下文窗口，并从训练数据中采样这个窗口和后续预测长度大小的窗口，构建随机梯度下降（SGD）的批次。将解码器只能在学习下一个值时查看之前的时间步骤，相当于人们训练机器翻译的原始 Transformer 所采用的“教师强制”。

Zhou（2021）等针对长序列时间序列预测问题（LSTF），提出了一种名为 Informer 的高效 Transformer 模型。在许多真实场景中，对长序列时间序列进行预测是至关重要的。长

序列时间序列预测（LSTF）要求模型具有高预测能力，即能够有效地捕获输入和输出之间的长期依赖关系。尽管 Transformer 模型在增强预测能力方面展现了潜力，但它面临着几个严重问题，包括二次时间复杂度、高内存使用量和编码器-解码器架构的固有限制，这些问题阻碍了其在 LSTF 中的直接应用。为了解决这些问题，Zhou（2021）等设计了一种基于 Transformer 的高效模型 Informer，具有三个显著特征：（i）ProbSparse 自注意力机制，实现了时间复杂度和内存使用量的 O(L logL)，并且在序列依赖性对齐上具有可比的性能。（ii）自注意力蒸馏通过减半级联层输入，高效处理极长输入序列。（iii）生成式风格的解码器，在概念上简单，能够一次性预测长时间序列，而不是逐步方式，大大提高了长序列预测的推理速度。通过在四个大规模数据集上的广泛实验表明，Informer 显著优于现有方法，并为 LSTF 问题提供了新的解决方案。

国内现代应用上，段梦梦等（2023）提出了基于 Transformer 特征融合的时间序列分类网络，在训练上得到了更好的表现；骆钊等（2024）改进了 Transformer，添加多尺度时序块，对短期风电功率进行预测；王树斌等（2024）利用 Transformer 对矿井内因火灾进行了时间序列分析，为人民的生命财产安全提供保障。

二、Transformer 架构

Transformer 的架构如下：

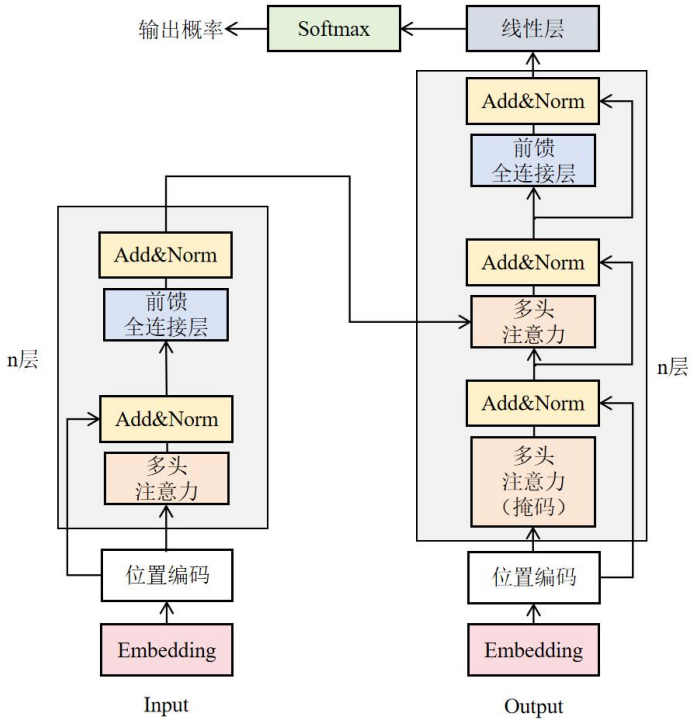

图 2-1 Transformer 架构示意图

Transformer 包含编码器（如图左侧）和解码器（如图右侧）两个部分。根据不同的任务，Transformer 将接收不同的原始数据。在数据传输给模型前，首先进行两步处理。第一步是嵌入，对数据的维度进行扩张，适于神经网络的训练。第二步是添加位置编码，帮助模型学习顺序信息。

编码器和解码器会分别接收处理后的数据的不同部分，编码器接收“输入”部分（此后称为 input），而解码器接收“输出”部分（此后称为 output）。例如在英译中翻译任务中，编码器接收原始的待翻译部分“I like you”，而解码器接收译文“我喜欢你”；在时间序列任务中，编码器接收过去的时间序列，解码器接收未来的时间序列。

编码器由多个编码器层堆叠而成。每个编码器层首先对数据进行多头注意力的计算，将结果与残差结合并进行归一化（如图 Add&Norm）。得到的输出使用前馈全连接层处理，再次结合残差并归一化。以上过程将在多个编码器层中重复，最终的结果作为编码器的输出。

同样，解码器由多个解码器层堆叠而成。与编码器层相比，解码器层首先进行掩码多头注意力的计算，与残差结合并进行归一化后，接收编码器的输出结果，实现交叉注意力，再进行与编码器层相似的前馈全连接层等处理。多个解码器层作用后，即形成解码器的输出（注意，该计算后的输出区别于解码器接收的 output）。

根据不同的下游任务，可以搭建不同的输出网络形成最终的预测结果，与真实值形成损失函数。由于注意力计算、全连接层等步骤均通过神经网络的神经元实现，因此可以利用反向传播机制对其中的权重矩阵进行优化，实现模型的训练。

接下来的小节中，将以 seq2seq 任务为例，从张量维度变化的角度详细介绍 Transformer 的各个组成部分的工作原理，以及这些组成如何串连成完整的 Transformer 架构。

（一）Transformer 组成部分

1. 嵌入

嵌入（Embedding）是一种将离散变量（如文本中的单词或字符）映射到连续向量空间中的技术，使得这些变量可以用于模型。在这个向量空间中，每个离散变量被表示为一个固定长度D_{model}的向量，这些向量在训练过程中学习得到，能够反映出单词间的语义和语法关系。从离散变量到向量的映射涉及到词典。以一个实际例子进行说明：比如词典有 1000 个词，每个词对应一个 0~999 之间的编号。隐藏层的大小D_{model}为 512，此时构建一个 512×1000 的权重矩阵，权重用均匀分布随机初始化。若训练样本是“you are very good”，则找到这些词在词典中对应的位置[100,508,2,421]，并提取出来 4 个长度为 512 的向量拼接为张量，此时张量大小为 4×512；如果有两个样本，则向量大小为 $2 \times 4 \times 512$。映射后的“长度”或“隐藏层的大小”在下文中称“特征”。

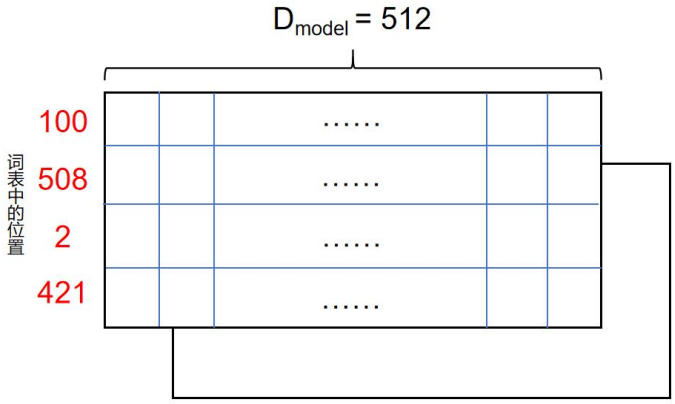

图 2-2 Embedding 的张量形状示意

2. 位置编码

接收序列信息时，序列中元素的顺序十分关键。Transformer 的自注意力（Self-Attention）机制（下一部分即将描述）并不区分序列中元素的顺序，位置编码因此成为引入序列顺序信息的关键方法。通过将位置编码向量添加到输入嵌入向量中，模型就能够根据序列中元素的位置进行区分。这使得模型能够捕获词序（word order）和句法结构等重要信息，从而在诸如文本翻译、文本生成和语音识别等任务中取得更好的性能。位置编码的具体计算公式为：

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{D_{model}}}}), \quad \text{式（2.1.1）}$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{D_{model}}}}). \quad \text{式（2.1.2）}$$

位置编码是沿着序列的时间步方向计算的，每一个特征的正弦映射函数（特别是其频率）是根据其编号的奇偶性及其在序列中的时间步号确定的。示意图如下：

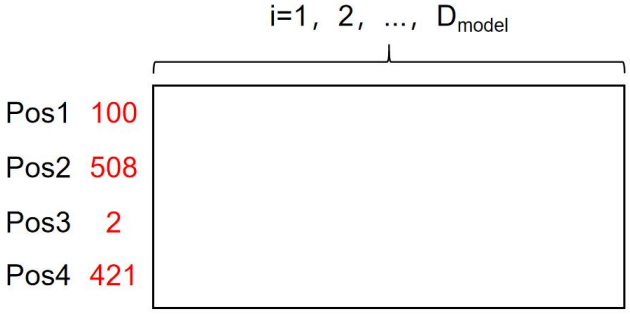

图 2-3 Positional Encoding

3. 自注意力和多头注意力

（1）自注意力

自注意力（Self-Attention）机制是 Transformer 模型的核心部分，它使模型能够在处理序列数据时考虑到序列中各个位置之间的依赖关系。自注意力机制的关键优势在于其能够动态地聚焦于输入序列中的不同部分，根据任务的需要分配更多的计算资源给更重要的信号。对于给定的输入序列 X（通常是嵌入向量的序列），分别通过三个不同的线性层 W（全

连接层）转换为三组新的向量，分别代表 Query（Q）、Key（K）和 Value（V）。

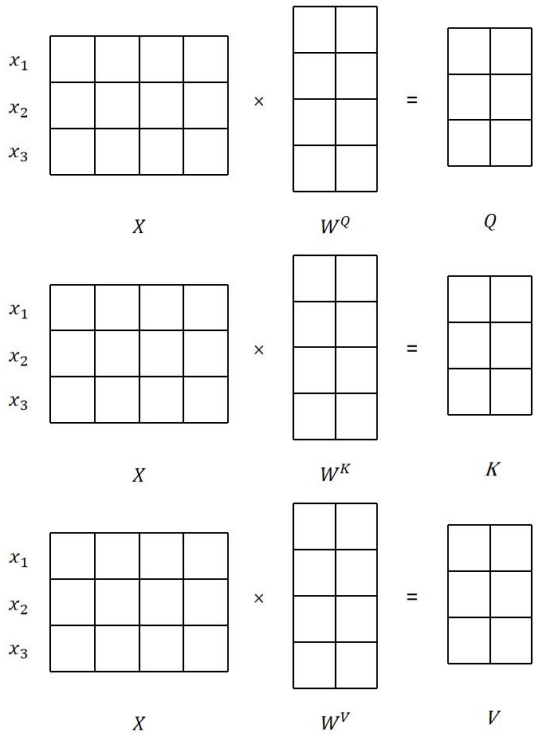

图 2-4 输入与查询、键、值

对于序列中的每个元素（即每个 Query），计算它与序列中所有 Key 的点积，得到的结果称为注意力分数（Attention Scores）。

使用 softmax 函数对注意力分数进行标准化，使得每一行的分数之和为 1。为了同时控制方差以保证模型训练时的稳定性，并且预防梯度消失，softmax 前的结果须除以键（key）的维度d_k。这样得到的注意力权重（Attention Weights）表示了每个元素对当前元素的相对重要性。将注意力权重与对应的 Value 向量相乘，然后对所有元素求和，得到输出向量，即缩放点积注意力。每个输出向量是输入序列中所有位置的加权表示，权重取决于它们与当前位置的相关性

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \quad \text{式（2.1.3）}$$

（2）注意力掩码

注意力分数反映了序列中每个元素对当前元素的重要性。在序列生成任务中，每个时

间步的输出应仅依赖于该时间步之前的输入，以避免“看到未来”的信息，因此将注意力掩码（Attention Mask）直接应用于计算得到的注意力分数上，便于可以有效地屏蔽（或忽略）序列中未来的信息（或某些任务中的填充（padding）项），以控制信息流动。

如下图所示，QK^T注意力得分进行掩码（与 Mask 相加）之后，仅保留了上三角部分。假设x_1, x_2, x_3分别代表 3 个时间步的序列，进行掩码后，第一行注意力得分只保留了第一个时间步的信息，第二行注意力得分则保留了第二个时间步的信息，第三行注意力得分保留第三个时间步的信息。由此，Transformer 在训练中可以并行地不断预测新的时间点。

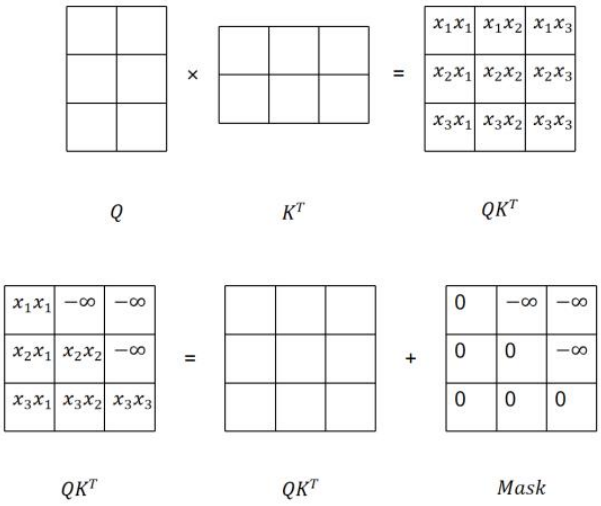

图 2-5 注意力掩码

（3）多头注意力

为了让模型能够同时关注来自不同表示子空间的信息，自注意力被扩展为多头注意力（Multi-Head Attention）。在多头注意力中，Q、K、V 被分割成多组，每组对应一个头，分别进行自注意力计算（即训练网络权重的过程）；然后将所有输出拼接起来，再次经过一个线性层得到最终输出。多头注意力可以利用或不利用注意力掩码。

4. 前馈全连接层

为了适应不同下游任务或输出目标，注意力模型的输出需要被映射到任务所需的维度。神经网络中的全连接层能够很好的实现这一点。两层的网络结构可表示如下：

$$f(x) = (ReLU(xW_1 + b_1))W_2 + b_2. \quad \text{式（2.1.4）}$$

即对含有线性层的和偏移项的结果进行激活函数，并再次经过线性层。为了降低复杂度，对$Relu(xW_1 + b_1)$进行随机丢弃（Dropout）。

5. Add & Norm

该组件包括残差连接（Residual Connection）和层归一化（Layer Normalization）两部分。残差连接将某一层的输入直接加到该层的输出上。对于模型中的某一层，

$$Output = Layer(Input) + Input. \quad \text{式（2.1.5）}$$

这样有助于缓解深度网络中的梯度消失或梯度爆炸问题，使得深层模型的训练变得更加稳定。它允许梯度直接通过网络传播，从而支持构建更深的模型结构。

层归一化往往应用于残差连接后，对每个样本在所有特征上进行归一化。不同于批归一化（Batch Normalization）是在批次的维度上进行归一化，层归一化在每一层内部进行，独立于批次大小。

$$LN(x_i) = \gamma(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}) + \beta. \quad \text{式（2.1.6）}$$

对于每个样本x_i，计算该层所有神经元输出的均值和方差（方差加上一个很小的数ϵ以防除零），使用上述均值和方差对该层输出进行归一化处理，最后对归一化的输出进行缩放γ和偏移调整β。

（二）Transformer 各部分串连

本节将结合张量的维度变化，阐述 Transformer 如何处理输入数据，即其架构的搭建。

设原始数据 X 大小为$[Batch\ Size, Context\ Length]$。通过进行词嵌入将原始数据扩增为$D_{model}$（一般取 512）个特征，此时数据$X_{embedding}$扩张为$[Batch\ Size, Context\ Length, D_{model}]$。进行位置编码，位置编码与嵌入向量对应位置的元素相加，不改变嵌入向量的维度，输出为X_{pe}。

1. 编码器（Encoder）

将X_{pe}传入编码器。X_{pe}分别与维度为$[D_{model}, D_{model}]$的W_K，W_Q，W_V做点积，进行线性变换，成为K，Q，V。此后，若多头注意力的头数为h，则传入每个头的维度为：

$$D_k = D_v = D_{model}||h. \quad \text{式（2.2.1）}$$

K 被切割为 K_1，K_2，……，K_h。设每一部分为 K_i，$i = 1,2,\ldots\ldots,h$。K_i 的大小是 $[Batch\ Size, Context\ Length, D_k]$，$Q_i$和$V_i$最后一个维度上的大小分别为$D_k$和$D_v$。经过式（2.1.3）的注意力得分计算，$Attention(Q_i, K_i, V_i)$ 的维度仍然是 $[Batch\ Size, Context\ Length, D_v]$（取$D_k = D_v$时）。再在最后一个维度上拼接，得到多头注意力得分$X_{MHA}$，大小仍为$[Batch\ Size, Context\ Length, D_{model}]$。

进行残差连接和归一化（此过程不改变矩阵的形状），

$$X_{EL1} = Norm(X_{MHA} + X_{pe}). \quad \text{式（2.2.2）}$$

进行前馈全连接，

$$X_{FF} = (Relu(X_{EL1}W_1 + b_1))W_2 + b_2. \quad \text{式（2.2.3）}$$

其中，W_1的大小为$[D_{model}, D_{ff}]$，W_2的大小为$[D_{ff}, D_{model}]$。因此该全连接层将X_{EL1}最后一维的维度扩张为D_{ff}后复原至D_{model}。再进行一次残差连接和归一化，

$$X_{EL2} = Norm(X_{EL1} + X_{FF}). \quad \text{式（2.2.4）}$$

此时，X_{EL2}即为编码器的输出。

以上过程是一个编码器层中实现的。一个模型可以有多个编码器层，每一层编码器的输出可以作为下一层编码器的输入，直到最后一层编码器层处理并输出结果，如果有 n 层编码器，则最终输出的结果为$X_{EL2}^{(n)}$，该结果将传递给 Decoder。这个过程称为编码器层的堆叠。

2. 解码器（Decoder）

训练阶段，在序列任务中，为了实现教师强制和防止信息泄露，原始数据 X 作为目标序列，先进行向右位移操作（即，在序列开始处添加一个特殊的<start>标记）。此后同样被转换成词嵌入向量，并加入位置编码，记为X_{pe-s}。经过线性变换和多头注意力切分，可以得到$Q_{i-s}, K_{i-s}, V_{i-s}$。解码器中的第一个多头注意力为掩码注意力，其注意力得分与掩码M相加，即

$$Attention(Q_{i-s}, K_{i-s}, V_{i-s}) = softmax(\frac{Q_{i-s}{K_{i-s}}^T + M}{\sqrt{d_k}})V. \quad \text{式（2.2.5）}$$

由此得到多头注意力得分X_{MHA-s}。再进行残差连接和归一化，

$$X_{DL1} = Norm(X_{MHA-s} + X_{pe-s}). \quad \text{式（2.2.6）}$$

接下来进入交叉注意力部分，解码器层会接收编码器的输出（后文中解码器层堆叠的过程

中，都是同样接收编码器的最终输出）。编码器的$X_{EL2}^{(n)}$将直接作为多头注意力的K，V，而Q使用掩码多头注意力得分X_{DL1}。因此解码器的第二个多头注意力中自注意力得分为

$$X_{MHA2-s} = Attention(X_{DL1}, X_{EL2}^{(n)}, X_{EL2}^{(n)}). \quad \text{式（2.2.7）}$$

残差连接则使用上一层的注意力输出X_{DL1}。归一化后得到$X_{DL2} = Norm(X_{MHA2-s} + X_{DL1})$。类似编码器，进行全连接，得到

$$X_{DL3} = Norm(X_{DL2} + X_{FF-s}). \quad \text{式（2.2.8）}$$

经过 n 层解码器层堆叠之后，得到$X_{DL3}^{(n)}$，作为编码器也即整个 Transformer 的输出。

3. 输出

$X_{DL3}^{(n)}$的维度依旧为$[Batch\ Size, Context\ Length, D_{model}]$。线性层中权重矩阵的大小为$[D_{model}, D_{vocab}]$，$D_{vocab}$即为字典的大小，经过线性层和不影响矩阵维度的 softmax 变换，$X_{DL3}^{(n)}$被映射为X_{prob}，大小为

$$[Batch\ Size, Context\ Length, D_{vocab}].$$

此时每个时间步（即$Context\ Length$中的每个位置）都对应一个概率分布，这个概率分布反映了给定当前位置之前的上下文时，序列中每个可能词汇作为“下一个词”的概率。在训练阶段，这允许模型并行地计算整个序列的损失，而不是单独地、逐步地预测每个词。对于序列中的每个位置，根据这个概率，模型可以确定在该位置最可能出现的“下一个词”。在训练阶段，这个预测结果与实际的下一个词（来自位移后的目标序列）进行比较，以计算损失并进行参数更新。注意，此处所计算的概率，区别于下一章中 Probabilistic 变体的概率建模。

三、Time Series Transformer

本节将阐述如何利用 Transformer 架构进行时间序列预测任务。为了适应 Transformer 的训练架构，训练时间序列前需进行一些特定的预处理。

（一） 时序数据预处理

假设训练集原数据为 N 个时间序列，每个时间序列长度为 M，且维度为 1。原始数据抽样可视化如下图 1 左。接下来数据将通过 AddTimeFeatures、AddAgeFeature 和 InstanceSplitter 三个组件组成的数据管道（Pipe）。

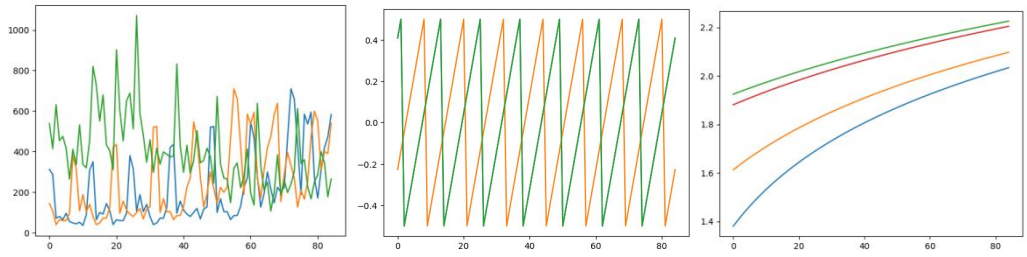

图 3-1 时序数据预处理可视化

首先，AddTimeFeatures 为时间序列添加$time\ features$个特征。其中，AddTimeFeatures 添加时间周期特征，例如添加月份数，如上图中；AddAgeFeature 添加“年龄”特征，并进行对数化，如上图右；此后，InstanceSplitter 抽取长度为 $ContextLength\ +\ PredictionLength$ 的窗口。注意，该长度和应该小于原始时间序列的长度 M；抽取是随机的，即从长度 M 的原序列中截取一段长度为$ContextLength\ +\ PredictionLength$的序列。对于多条时间序列，可以为每一条序列添加静态分类特征和静态实值特征。例如 N 个时间序列有两个大小为[N,1]的静态特征，每个元素对应一条序列。最后从原数据为 N 个时间序列中抽取样本，数量应该等于批次大小$Batch\ Size$。若原序列共有C个特征，当前批次输入的数据原序列尺寸为

$$[Batch\ Size, ContextLength\ +\ PredictionLength, C],$$

添加新特征，尺寸变为

$$[Batch\ Size, ContextLength\ +\ PredictionLength, time\ features],$$

以及两个静态变量

$$[Batch\ Size, 1].$$

（二） 使用时间序列数据训练 Transformer

1. Vanilla Transformer Time Series

（1）对 Batch 的进一步处理

如上文所属，原始 Transformer 处理时间序列前，需要对时序数据进行一些预处理，例如序列任务中的位置编码和时间周期序列的频率特征。在形成训练批次（Batch）前还需进一步进行嵌入。

Vanilla Transformer 使用一维卷积刻画原始序列。首先创建一个一维卷积层，假设该层的卷积核大小为 3，共有D_{model}个滤波器。该卷积层沿着序列维度，每次移动一个时间步，将输入数据的特征通道从C（初始序列特征数）转换为 D_{model}；同时采用'circular'策略，使得扩张维度后的序列长度能够和原始序列对齐。此外，考虑到激活函数 leaky_relu 的特性，使用 Kaiming 初始化方法初始化卷积层的权重，旨在保持各层激活值的分布一致，从而有助于模型的稳定和快速训练。卷积转换有助于将原始特征映射到一个更适合后续模型处理的维度空间。

进行通过卷积层后，张量与位置编码、广播（BroadCasting）后的时间周期特征相加，向模型输入大小为$[Batch\ Size, ContextLength + PredictionLength, D_{mdoel}]$的数据张量。

（2）训练过程

以单变量时间序列为例，在嵌入前，原始数据的大小为$[Batch\ Size, ContextLength + PredictionLength, 1]$。其中，将 Prediction Length 的部分分解为 $\text{Label}_{\text{len}}$ 和 Pred_{len} 两部分，$PredictionLength = \text{Label}_{\text{len}} + \text{Pred}_{\text{len}}$

训练阶段，将 Pred_{len} 部分全部替换为 0 后（如下图左）进行卷积扩张维度，再正常添加其余的时间特征（如下图右）。每一批次的训练时，ContextLength 部分正常输入编码器，将替换 0 后的 Prediction Length 部分输入解码器（这一点将在下节中进一步在维度变化上阐述）。经过 Transformer 计算，此前被 0 覆盖的 Pred_{len} 部分将被预测出来，与被覆盖前的序列产生损失，通过反向传播进行参数的调优。

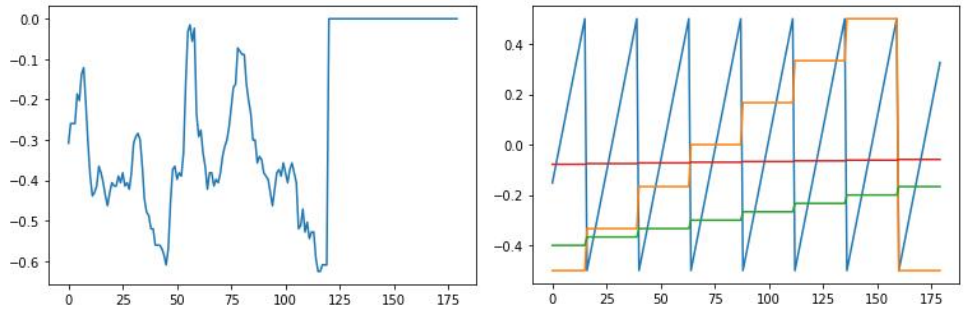

图 3-2 Transformer 训练阶段数据输入的示意图

2. Probabilistic Time Series Transformer

（1）对 Batch 的进一步处理

Probabilistic Time Series Transformer 需要更多的序列信息，因此其批次处理的主要思想是继续为批次内的序列添加特征。

首先对静态分类特征嵌入，转换为连续的向量表示。每个分类特征对应的嵌入层将原始的整数索引映射到高维空间中的一个向量，以增强模型处理分类数据的能力。如果有多个静态分类特征，每个特征经过嵌入层转换后被拼接起来。假设有 F 个静态分类特征，每个特征的嵌入维度为 E，那么拼接后的向量尺寸将是 $[Batch\ Size, F \times E]$。

将静态分类特征和 D 个静态实值特征的向量拼接起来，形成一个包含所有静态信息的向量，大小为$[Batch\ Size, F \times E + D]$。为了与时间依赖的特征（如动态时间特征）在模型中一起使用，将静态特征向量扩展到整个序列的每个时间步上——在时间维度上复制静态特征向量，使得静态特征的尺寸变为 $[Batch\ Size, ContextLength + PredictionLength, F \times E + D]$。

Probabilistic Time Series Transformer 使用滞后子序列刻画原始序列的特征。首先我们选择对于该序列的训练考虑滞后几阶的信息，即采用某一个时间点前几个时间步的值，记为 L（形式上，例如希望关注滞后 lag sequence[1，2，10，11]步的信息，则 L=4）。如果序列含有C个变量，那么每个变量都取前 lag sequence 步的值，就一共需要考虑$C \times L$个滞后值。考虑全部样本以及每一个时间步的滞后值，滞后张量的尺寸为$[Batch\ Size, ContextLength + PredictionLength, C \times L]$。

将上述所有特征组合起来，得到输入 Transformer 的向量尺寸为

$$[Batch\ Size, ContextLength + PredictionLength, C \times L + F \times E + D + time\ features]$$

对于时间序列预测任务，训练时向 Transformer 传入序列数据的方式比较独特。将输入

向量分为 Context 和 Prediction 两部分。向编码器传输为过往时序信息，即$[Batch\ Size, ContextLength, C \times L + F \times E + D + time\ features]$部分；向解码器传输为未来时序信息，即$[Batch\ Size, PredictionLength, C \times L + F \times E + D + time\ features]$部分。尽管两部分的序列长度不同，交叉注意力部分的计算并不会出现维度上的不匹配。考虑后两个维度，由解码器提供的Q与编码器提供的K^T进行内积，维度是$[PredictionLength, ContextLength]$，再与编码器提供的$V$作内积，维度回归到$[PredictionLength, C \times L + F \times E + D + time\ features]$。这也是解码器的最终输出维度。

得到解码器的输出后，模型使用 ParameterProjection 层将这些输出映射到特定概率分布的参数上。例如对于学生 t 分布，args_dim 定义了其所需的三个参数（自由度 df、位置 loc、尺度 scale）的维度。ParameterProjection 层包含一系列线性层，每个线性层负责生成分布的一个参数（这些参数最初是未受限制的，可能不满足特定分布参数的要求）。ParameterProjection 层的输出需要通过 domain_map 函数进行转换，以确保这些参数符合特定概率分布的要求。例如，如果预测的分布是正态分布，domain_map 会确保方差参数是正的。

具体来讲，DistributionOutput 类利用 get_parameter_projection 方法返回一个映射层，将模型输出映射到分布参数。这一步骤通过一个神经网络层 ParameterProjection 类实现，它根据 in_features 和 args_dim 生成所需的分布参数。

假设使用学生 t 分布预测。如果解码器输出的是维度为$[Batch\ Size, PredictionLength, dims]$的隐状态，上述的神经网络层会将这个输出映射为 3 个参数矩阵（对应 t 分布所需的三个参数），维度均为$[Batch\ Size, PredictionLength, 1]$。即，对于每个样本的每个时间步会得到一组参数。

一旦得到了适当转换的分布参数，TimeSeriesTransformerForPrediction 模型就会使用这些参数构建一个概率分布。这个分布由 DistributionOutput 类的一个实例来定义，该实例负责根据提供的参数构建分布：调用 distribution 方法，根据给定的分布参数（distr_args），构造一个分布实例。如果提供了 loc 和 scale，则返回一个经过位移和缩放变换的分布实例（AffineTransformed）。

进行可视化实验，生成一个大小为[256,24,22]的随机张量，从样本 0 上随机取 9 个时间步的参数，并拟合其密度函数曲线。

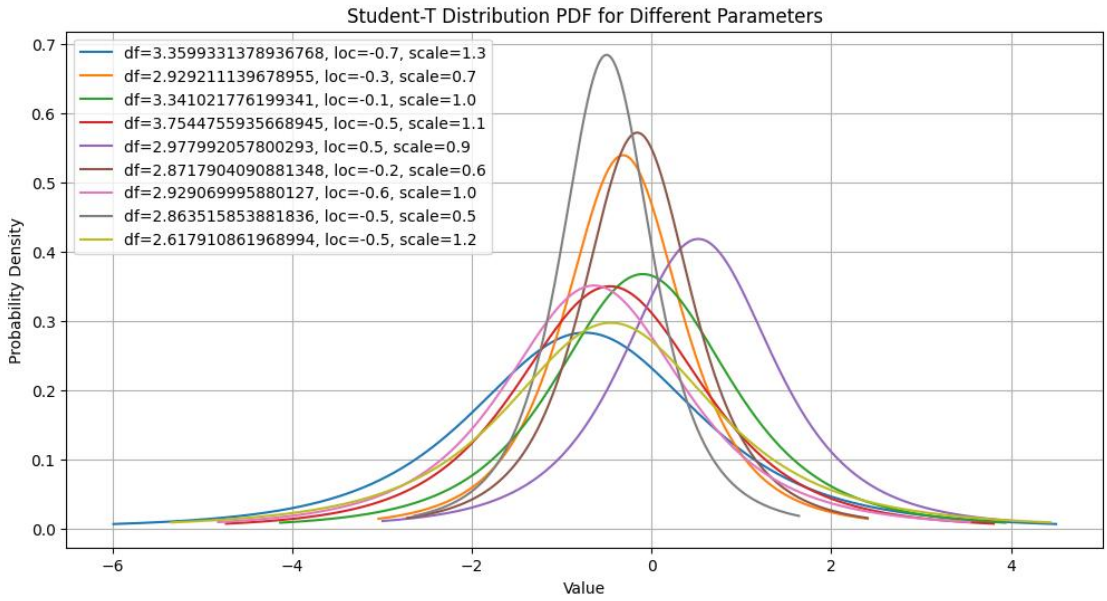

图 3-3 Probabilistic Transformer 的概率分布建模

根据参数得到分布后，对于训练过程，如果提供了未来的真实值（future_values），模型可以计算预测分布与实际观测值之间的损失，例如对数似然损失（衡量给定模型预测分布下，观察到的数据样本出现的概率的负对数值；更低的 NLL 值意味着模型的预测分布更接近真实分布）。将多个样本的对数似然损失取平均，用于模型训练，通过反向传播更新权重。

（2）推断和预测时 Probabilistic Transformer 的运行

对于预测（推理）过程，模型可以直接从构建的概率分布中采样，以生成对未来值的预测。这通常是通过 generate 方法实现的，该方法负责自回归地生成预测序列，即每次生成一个未来时间步的预测，并将其作为下一步预测的输入。

以一个简单的示例（样本数量为 1 的单变量时间序列）可视化训练和预测阶段 Transformer 接收的信息的区别。

图 3-4 自回归预测的输入和输出

原始序列中，已知的信息为 con1~pred4，pred5 是未知的。训练阶段，将不同滞后阶数的序列作为特征传给模型（过去值输入给编码器，未来值输入给解码器）。此时模型会输出 pred4 的分布，再与真实的 pred4 计算对数似然损失，进行反向传播。而在预测阶段，输入的数据与训练阶段相比整体更新一个时间步，输出 pred5 的预测分布后，通过随机采样，得到 pred5 的预测值。

3. Informer

Zhou 等（2021）提到，Transformer 模型在长时间序列预测任务上的存在三个主要障碍，限制了其在此类任务上的应用和性能。

第一是自注意力机制的二次方计算复杂度（Quadratic computation of self-attention）。自注意力机制的核心操作，即规范化的点积，导致了每一层的时间复杂度和内存使用量都是$O(L^2)$，其中 L 是输入/输出的长度。这意味着随着序列长度的增加，所需的计算资源和时间成二次方增长，这对于长序列来说尤其成问题。

第二是对长输入堆叠层时的内存瓶颈（The memory bottleneck in stacking layers for long inputs）。Transformer 模型通过堆叠多个编码器/解码器层来增强模型的能力。但是，当输入序列很长时，每堆叠一个层，模型的总内存使用量大约为$O(nL^2)$，其中 n 是层数。这会极大限制模型处理长序列输入的能力，因为可用内存是有限的。

第三是预测长输出时速度下降（The speed plunge in predicting long outputs）。原始 Transformer 在解码（预测）阶段采用的是逐步解码的策略，即每次预测一个输出后，再基

于之前的输出进行下一个预测。这种方式使得预测长序列时的推断速度大幅下降，与基于循环的模型（如 RNN）类似，影响了效率。

由此，Zhou 等（2021）提出了 Informer 模型，旨在改进传统 Transformer，优化对于以上几个问题的表现。Informer 的数据预处理和批次形成方式与 Vanilla Transformer 相同。

（1）Prob Sparse 自注意力机制

对于第一个问题，作者提出 Prob Sparse 自注意力机制。这种机制通过识别和仅计算那些“重要”的注意力得分来减少计算量，而不是在自注意力中计算所有可能的键和查询对。这基于一个观察（observation）——即在自注意力的得分分布中存在稀疏性，意味着只有少数的键-查询对贡献了主要的注意力得分，而其他大多数得分对最终的注意力输出贡献较小。

如下图，对训练后的注意力得分（即式 2.1.3 中的QK^T）绘制热力图。

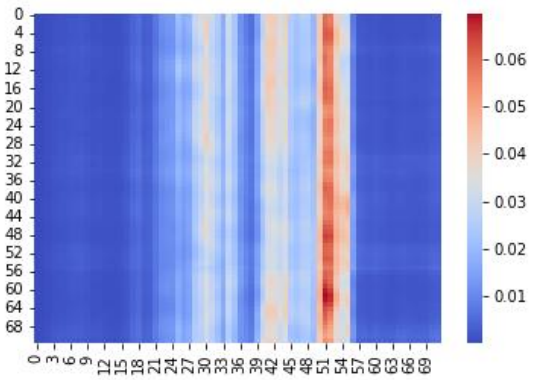

图 3-5 注意力得分的权重分布

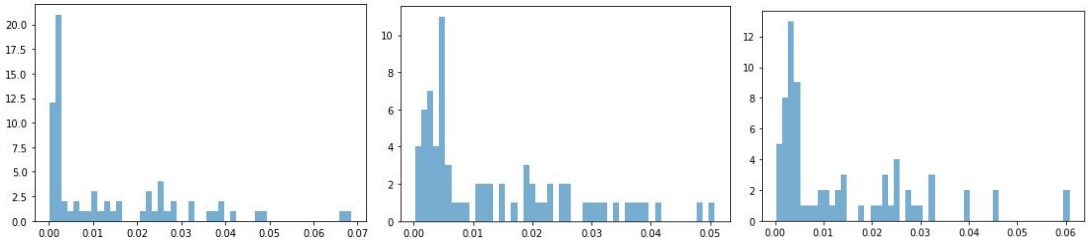

图 3-6 抽取单行注意力得分的分布

抽取部分注意力得分的行，可以看到注意力得分的分布不同，例如上图左，分布较为集中，其余的分布较均匀。

ProbSparse 自注意力通过对键-查询对的重要性进行量化，选择出最有可能具有高注意力得分的键-查询对进行计算。具体来说，它通过引入一个基于 Kullback-Leibler 散度的稀疏测量方法，估计每个查询在所有键上的注意力分布的稀疏程度，然后只选择 KL 散度最大（即

注意力分布与均匀分布差比最大）的一小部分键-查询对进行计算。

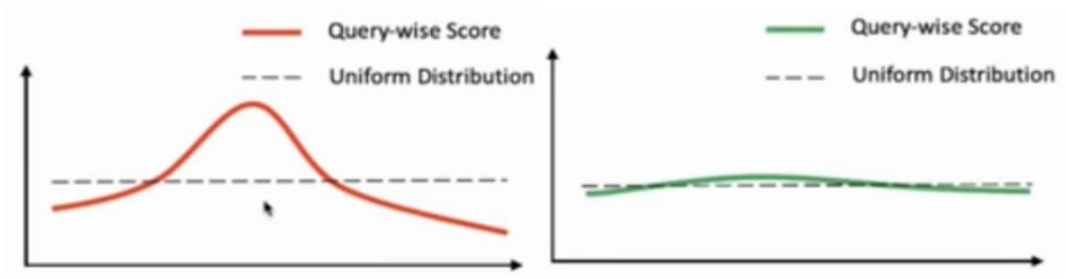

图 3-7 KL 散度示意

上述 KL 散度具体计算方式如下：

$$Attention(Q,K,V)=softmax(\frac{QK^T}{\sqrt{d_k}})V \rightarrow Attention(q_i,K,V)=\sum_j \frac{k(q_i,k_j)}{\sum_l k(q_i,k_l)}v_j, \quad \text{式（3.1）}$$

$$KL(q||p)=qln\frac{q}{p}=\sum_{j=1}^{L_k}\frac{1}{L_k}ln\frac{\frac{1}{L_k}}{\frac{k(q_i,k_j)}{\sum_l k(q_i,k_l)}}=ln\sum_{j=1}^{L_k}e^{\frac{q_i k_j^T}{\sqrt{d}}}-\frac{1}{L_k}\sum_{j=1}^{L_k}\frac{q_i k_j^T}{\sqrt{d}}-lnL_k. \quad \text{式（3.2）}$$

其中，L_k是均匀分布的概率。通过上述稀疏性原理，ProbSparse 自注意力机制将自注意力的时间复杂度和内存使用量从$O(L^2)$优化到了$O(LlnL)$。这使得模型能够更高效地处理长序列数据，显著减少了计算资源的需求。

（2）Self-Attention Distilling

针对第二个问题，希望精炼和压缩每一层的输入，降低模型对内存的需求。自注意力蒸馏（Self-Attention Distilling）通过在堆叠的每一层中减少输入序列的时间维度大小来实现。具体来说，它利用最大池化操作和一维卷积来对每一层的输出进行下采样，从而减少了下一层的输入长度。这一过程在每一层重复执行，有效减少了每一层及整个模型的内存使用量。为了增强自注意力蒸馏操作的鲁棒性，模型在编码器中构建了多个堆叠副本，每个副本接收的输入序列长度依次减半，形成一个金字塔式的结构。这些不同长度输入的堆叠副本的输出最后被级联起来，共同作为编码器的最终输出，这样做可以从多尺度捕获序列的特征，增强模型的表示能力。

通过自注意力蒸馏操作，模型在接收长序列输入时的总体空间复杂度从 $O(n\cdot L^2)$降低到了 $O((2-e)\cdot L\ log\ L)$，其中 n 是堆叠的编码器/解码器层数。这一显著的降低使得模型能够在有限的内存资源下处理更长的序列。

上述策略有效解决了原始 Transformer 在处理长序列数据时面临的内存瓶颈问题，提高了模型处理长序列的能力，同时保持了对计算资源的高效利用。

（3）Generative Style Decoder

针对第三个问题——预测长输出时速度下降，文章提出了生成式风格的解码器（Generative Style Decoder）作为解决策略。这种解码器的设计允许模型以一种高效的方式一次性生成长序列的预测，而不是传统的逐步预测方式。Informer 采用了一种新的解码器结构，使得在进行长序列预测时，可以避免传统的逐步（step-by-step）解码过程。相对于传统的逐步解码，这种方法可以在一次前向传播中直接生成整个长序列的预测输出，显著提升了长序列预测的推理速度。传统的逐步解码方法在每一步都依赖于前一步的预测结果，因此容易在预测过程中累积误差，特别是在长序列预测时这一问题尤为严重。生成式风格的解码器通过一次性生成所有的预测输出，有效避免了这种累积误差的传播，从而提高了预测的准确性。此外，在生成式风格的解码过程中，模型使用包含起始标记和目标序列的时间戳信息的输入，来指导整个长序列的生成。这样做不仅提高了预测的效率，而且还允许模型根据时间戳直接预测任意时间范围内的序列，增加了模型的灵活性和适用范围。通过引入生成式风格的解码器，文章成功解决了原始 Transformer 在长序列预测时推理速度下降的问题，大幅提高了模型在长序列时间预测任务上的推理效率和准确性。这一策略为 Transformer 模型在长序列预测领域的应用提供了新的可能性。

四、实验

实验部分数据集采用发电机油温数据。电力分配问题是电网根据顺序变化的需求管理电力分配到不同用户区域。但要预测特定用户区域的未来需求是困难的，因为它随工作日、假日、季节、天气、温度等的不同因素变化而变化。现有预测方法不能适用于长期真实世界数据的高精度长期预测，并且任何错误的预测都可能产生严重的后果。因此当前没有一种有效的方法来预测未来的用电量，管理人员就不得不根据经验值做出决策，而经验值的阈值通常远高于实际需求。保守的策略导致不必要的电力和设备折旧浪费。值得注意的是，变压器的油温可以有效反映电力变压器的工况。我们提出最有效的策略之一，是预测变压器的油温同时设法避免不必要的浪费。

为了解决这个问题，Zhou 等（2021）的团队与北京国网富达科技发展公司建立了一个平台并收集了 2 年的数据。我们用它来预测电力变压器的油温并研究电力变压器极限负载能力。该数据集提供了两年的数据，每个数据点每 15 分钟记录一次（用 m 标记），它们分别来自中国同一个省的两个不同地区，每个数据集包含 2 年 * 365 天 * 24 小时 * 4 = 70,080 数据点。 该数据集可以变换为一个小时级别粒度的数据集变体使用。 每个数据点均包含 8 维特征，包括数据点的记录日期、预测值“油温”以及 6 个不同类型的外部负载值。由于电脑配置限制，本文仅采用其目标值“油温”作为训练和预测的时间序列对象，即单变量时间序列研究。

（一）描述性统计

对原始数据进行可视化，时序图以及 ACF 图如下。

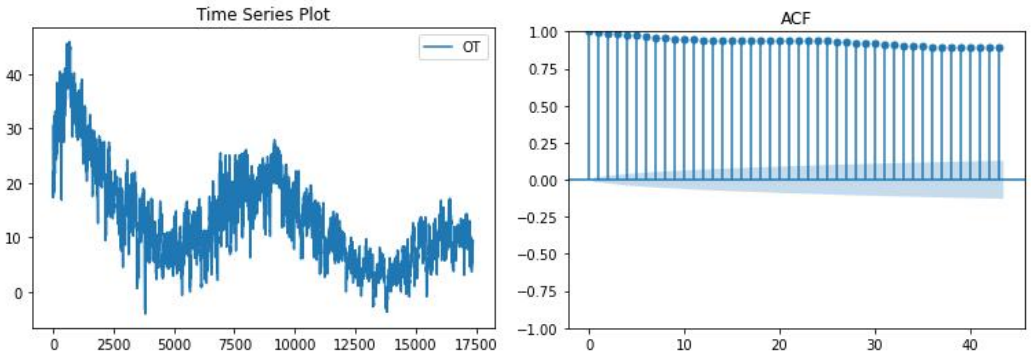

图 4-1 发电机油温数据时序图和 ACF 图

从时序图中可以看到，该数据集时间步较多，短期内呈现周期性；且由 ACF 图，该序列是非平稳的。不稳定性和充分的训练样本适合使用深度学习进行训练。

（二）实验描述

Probabilistic Transformer 对概率进行建模并采样，采用逐步预测的方式，适于处理短序列时间序列；Informer 则专为处理长序列时间序列设计。因此，实验部分在比对 Vanilla Transformer 与 Probabilistic Transformer 性能时，利用短序列训练并预测；在比对 Vanilla Transformer 与 Informer 性能时，利用长序列训练并预测。进行训练前，对数据集进行划分，同时对模型超参数进行设置。

表 4-1 数据集划分

	Transformer 和 Informer	Probabilistic Transformer
训练集	总时间步*0.7	总时间步*0.8
验证集	总时间步*0.1	
测试集	总时间步*0.2	总时间步*0.2

由于 Probabilistic Transformer 的结构自身已经设置了验证集进行参数调整，因此仅设置训练集和测试集。测试集的使用上，从测试集的第 1 个时间步开始，每个窗口向后移动一个时间步，向模型输入$ContextLength + PredictionLength$长度的数据，一共获取总时间步*0.2 个窗口。此外，对于 Transformer 模型的训练结构，预先设置一些超参数。

表 4-2 模型超参数设置

超参数名称	值
多头注意力头数 num_head	8
时间序列特征数（卷积输出）d_model	512
全连接层隐藏层维度 d_ff	2048
激活函数	GeLU
批次大小 batch	128/256
训练迭代轮次	6

（三）短序列训练和预测

数据选取上采用如下策略：

表 4-3 短序列训练的样本大小

序列部分	$ContextLength$	$PredictionLength$	
		label_len	pred_len
长度	96	48	24

根据预先设置好的模型参数，进行模型训练

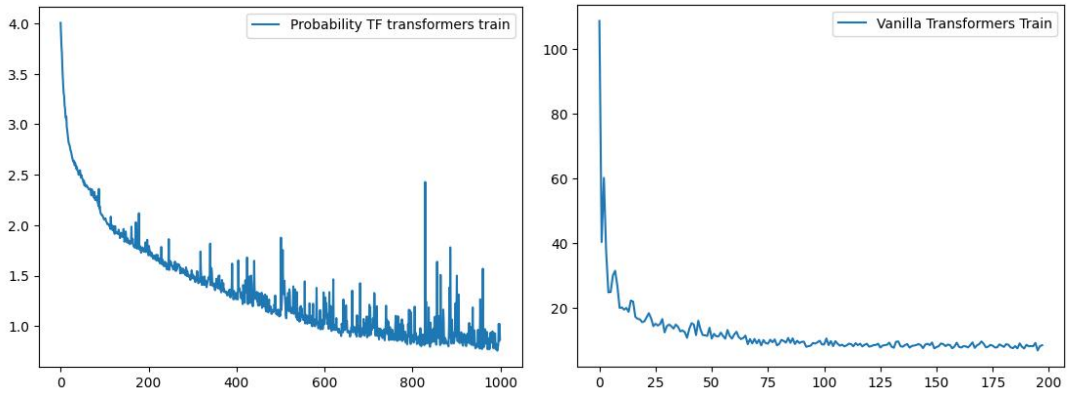

图 4-2 训练损失变化

可以看到，Prob Transformer 的训练过程收敛程度不如 Vanilla Transformer。这可能是由于其输出使用随机取样导致的，此外二者损失的计算方式也不相同。

表 4-4 短序列预测结果

模型	测试集上的评价指标		
	MAE	MSE	RMSE
Vanilla Transformer	2.1409	7.0088	2.6474
Probabilistic TF Transformer	2.5931	8.7568	2.9592

从测试集的结果来看，各项评价指标上 ProbTransformer 的表现均不如 VanillaTransformer。因此，进一步地抽取测试集上第 1、501、1001、1501、2001 个测试样本的预测结果，进行可视化。

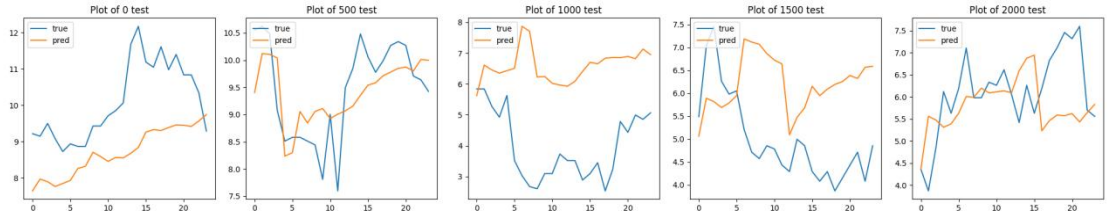

图 4-3 Vanilla Transformer 测试集抽样

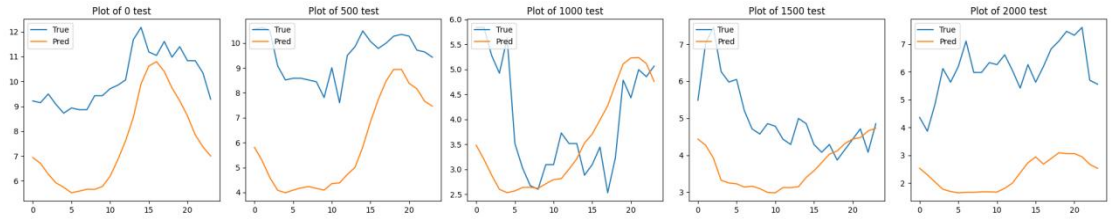

图 4-4 Probabilistic TF Transformer 测试集抽样

尽管 Probabilistic Transformer 在预测准确性指标上不如 Vanilla Transformer，通过具体的可视化却可以看出，Probabilistic Transformer 对于趋势的判断比 Vanilla Transformer 更加准确。Probabilistic Transformer 往往能够刻画出序列低谷与峰值间的转换，而 Vanilla Transformer 却不时地表现出错位性，甚至出现与原始序列走势相反的情况。

（四）长序列训练和预测

对于长序列实验，主要进行 Vanilla Transformer 和 Informer 性能上的比对。为了更好地比较二者在不同长度长序列预测上的训练效果，分别进行两组实验，在数据选取上采用如下策略。

表 4-5 长序列训练的样本大小

序列部分	$ContextLength$	$PredictionLength$	
		label_len	pred_len
实验 1 长度	240	120	60
实验 2 长度	360	180	90

训练过程中的损失变化情况如下。两个实验中，两个模型均达到了收敛。

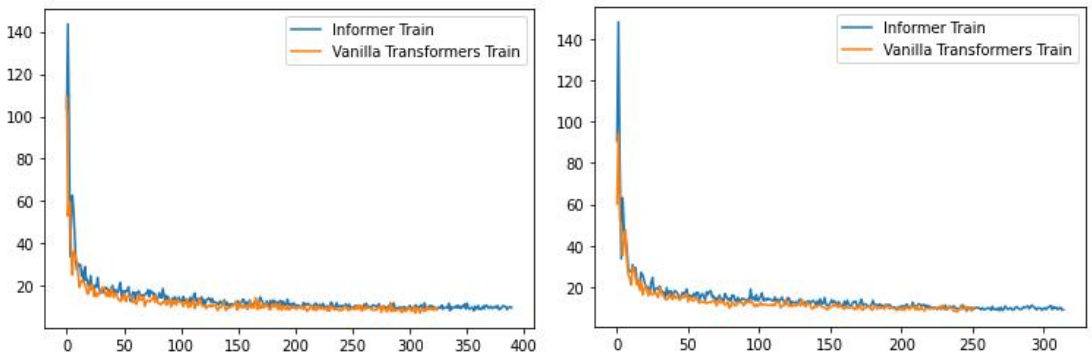

图 4-5 训练损失变化

实验 1 结果

表 4-6 长序列预测结果 1

序列长度	模型	测试集上的评价指标		
		MAE	MSE	RMSE
实验 1	Vanilla Transformer	3.0780	14.2921	3.7805
	Informer	3.1378	14.2104	3.7696

可以看到，对于长序列实验 1，Vanilla Transformer 和 Informer 在预测结果上没有表现出显著差异，Informer 的表现略优于前者。

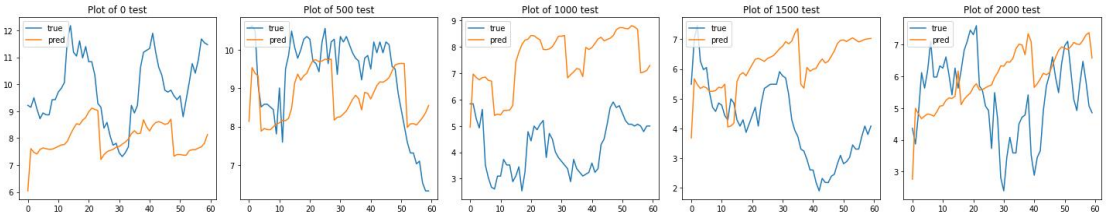

图 4-6 Vanilla Transformer 实验 1 测试集抽样

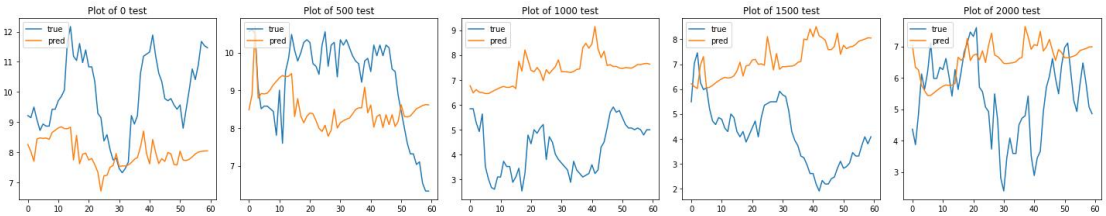

图 4-7 Informer 实验 1 测试集抽样

对预测结果进行可视化，可以看到趋势上 Vanilla Transformer 有着更加准确的判断，Informer 的表现不够理想。

表 4-7 长序列训练时间

序列长度	模型	Epoch 时间（s）					
		0	1	2	3	4	5
实验 1	Vanilla Transformer	1009.49	1121.28	940.43	941.02		
	Informer	798.15	850.15	839.59	823.25	792.03	704.18

训练速度上，Informer 比 Vanilla Transformer 具有显著优势。

实验 2 结果

表 4-8 长序列预测结果 2

序列长度	模型	测试集上的评价指标		
		MAE	MSE	RMSE
实验 2	Vanilla Transformer	3.6074	18.5725	4.3095
	Informer	3.2338	15.4008	3.9243

对于长序列实验 2，Vanilla Transformer 在更长的输入和预测任务下表现出的下降较为严重，而 Informer 的表现尽管有所下降，与 Vanilla Transformer 相比优势却更为突出。

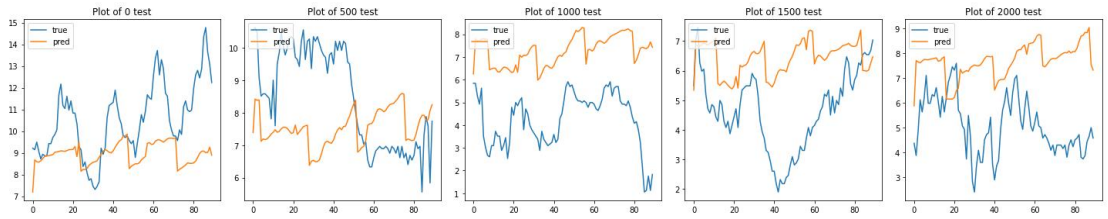

图 4-8 Vanilla Transformer 实验 2 测试集抽样

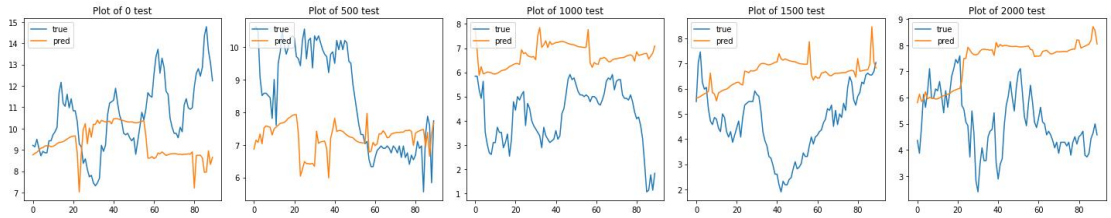

图 4-9 Informer 实验 2 测试集抽样

同样进行结果可视化，尽管 Informer 在指标上具有更好的结果，但无法判断出序列的周期性和涨跌趋势，这弱于 Vanilla Transformer 的表现。

比较模型的训练时间：

表 4-9 长序列训练时间

序列长度	模型	Epoch 时间（s）					
		0	1	2	3	4	5
实验 2	Vanilla Transformer	1402.87	1415.70	1086.84	1073.47		
	Informer	837.55	859.03	859.24	809.62	800.58	

速度上，依然是 Informer 占据绝对优势。

Informer 微调与优化方向

Transformer 和 Informer 模型有大量超参数可以调整，第一类超参数是基于深度学习训练本身，例如优化器的选取、学习率的调整、迭代次数、批次大小等等；第二类超参数是与 Transformer 和 Informer 本身相关的，例如注意力的头数，Prob sparse 的保存率等。由于算力和时间有限，本研究不采用网格搜索的方式逐一比对各个超参数调整对模型产生的影响。但在实验过程中，不同的训练参数产生了一些值得比较的结果。

Batch Size 的调整

训练批次的大小可能会产生几方面的影响，例如收敛性、泛化能力、训练速度等。对于长序列上 Informer 的训练，尝试。因此对 Informer 部分训练超参数进行调整，将 batch size 由 128 提升到 256，同时增加迭代的轮次，得到新的结果。与初始的 Informer 结果对比：

表 4-10 Informer 实验 2 调整比对

序列长度	模型	测试集上的评价指标		
		MAE	MSE	RMSE
实验 2	Informer（初始）	3.2339	15.4008	3.9244
	Informer（调整）	3.1036	14.5553	3.8151

评价指标上看，更大的 batch size 和更多轮的训练一定程度上提升了 Informer 的性能。进一步可视化地比较测试集上 Informer 的预测结果：

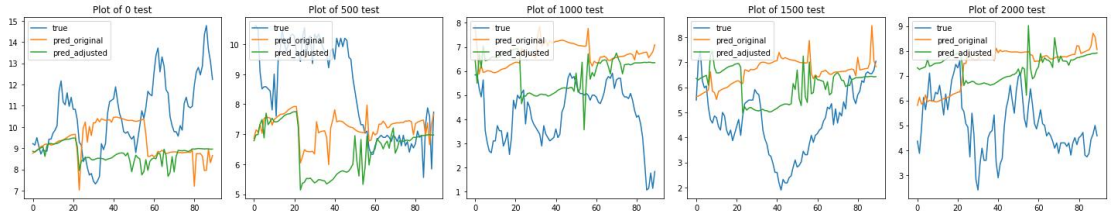

图 4-10 Informer 实验 2 调整比对

Informer 的表现有所提升，更加贴近初始序列。计算时间上则产生了更大的开销。

表 4-11 Informer 实验 2 调整后训练时间

序列长度	模型	Epoch 时间（s）					
		0	1	2	3	4	5
实验 2	Informer	1056.92	1021.77	1008.43	1010.74	1009.40	

Attention 策略的使用

为了进一步探究 Informer 调整自注意力机制的作用，进一步进行实验。首先选取短序列样本：

表 4-12 Informer 测试样本长度

序列部分	Context Length	Prediction Length	
		Label Length	Pred Length
样本长度	48	24	12

测试集上的预测结果如下：

表 4-13 Informer 测试预测结果

模型	模型	测试集上的评价指标		
		MAE	MSE	RMSE
	Vanilla	1.7185	4.4996	2.1212
	Prob Sparse	1.7045	4.4938	2.1198
	Prob Sparse+Distill	1.5255	3.9968	1.9992

可以看到，进行注意力概率稀疏和蒸馏后，模型的输出效果有比较显著的提升。输出注意力得分，并绘制热力图：

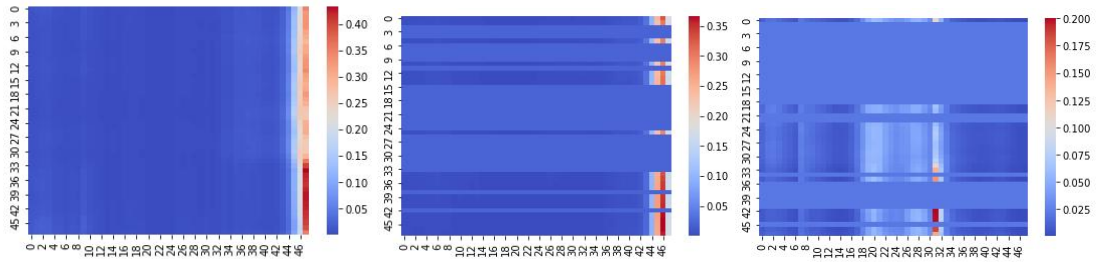

图 4-11 短序列注意力得分热力图

如上图左，全量注意力得分本身是比较稀疏的，按行来看，“重要”注意力与“不重要”的注意力差异比较显著。上图中和右是进行概率抽样和蒸馏操作后的注意力得分，有效地捕获了较为“重要”的注意力，保存了重要的信息。

在一次实验过程中，Vanilla Transformer 获得了如下结果

表 4-14 Informer 较优结果

MAE	MSE	RMSE
2.5907	10.9242	3.3052

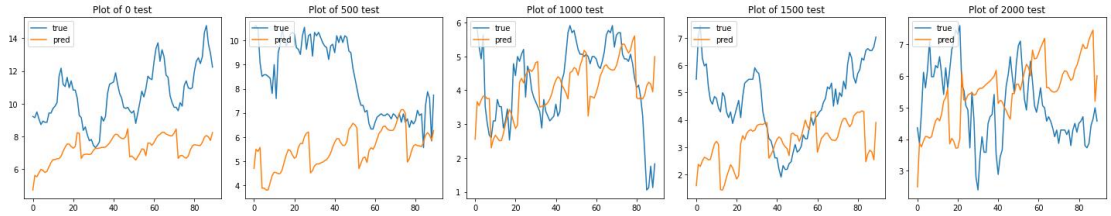

图 4-12 Vanilla Transformer 长序列预测结果抽样

不仅指标上显著优于 Informer，也较好的捕获了原序列的周期性和趋势。对其注意力得分进行可视化，如下图左侧。

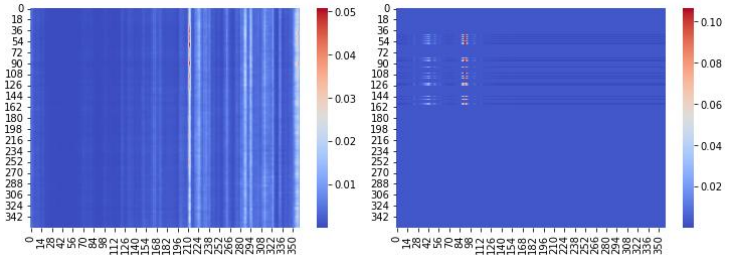

图 4-13 长序列预测任务注意力得分比较

可以看到，各行注意力得分的区别并不显著，如果进行概率选取和蒸馏，可能导致重要信息损失，如右图 Informer 的注意力得分。因此，Informer 的性能并不总是优于 Vanilla

Transformer 的，应该视注意力得分的情况决定是否采用概率稀疏和蒸馏策略。

五、结论

本文通过深入探讨和对比了 Transformer 模型及其在时间序列预测领域的几种变体，尤其是 Probabilistic Transformer 和 Informer 模型，以解决长序列时间序列预测问题（LSTF）。本文首先回顾了 Transformer 模型的基本构架和原理，强调了其在处理长距离依赖关系方面的优势，以及在时间序列预测任务中面临的主要挑战。通过对 Probabilistic Transformer 的研究，展示了如何通过概率建模和采样来捕获和表达预测的不确定性，进而改善模型在短序列预测任务上的性能。然而，对于长序列数据，Informer 模型通过其创新的 ProbSparse 自注意力机制、自注意力蒸馏技术和生成式解码器设计，有效解决了原始 Transformer 模型在处理长时间序列时遇到的计算复杂度高和内存限制问题，提升了长序列预测的效率和准确性。

本文通过在发电机油温数据集上的实验，对比了 Vanilla Transformer、Probabilistic Transformer 和 Informer 模型在短序列和长序列预测任务上的性能。实验结果显示，Probabilistic Transformer 在短序列预测任务上能够有效捕获序列的趋势变化，Vanilla Transformer 却能够更精准的进行点估计；长序列预测任务上，Informer 模型凭借其高效的长序列处理能力，展现出和更快的训练速度和更准确的预测结果。此外，实验表明训练批次等超参数的调整可以进一步优化预测的结果；在时间和内存允许的情况下，Vanilla Transformer 处理长序列也可能会拥有更好的表现，因为 Informer 的蒸馏和采样机制可能致使信息提取不够充分；这为未来实践中模型的进一步微调提供了方向。

本文的研究不仅为时间序列预测提供了有力的工具和方法，同时也为深入探索 Transformer 模型在更广泛的时间序列分析和预测应用中的潜力铺平了道路。未来的工作将聚焦于进一步优化 Probabilistic Transformer 和 Informer 模型的结构和参数，探索其在不同领域和更复杂时间序列数据集上的应用，以及开发新的模型变体以更好地捕获长序列数据的动态特性和复杂依赖关系。

参考文献

段梦梦,金城.基于 Transformer 特征融合的时间序列分类网络[J].计算机科学,2023,50(12):97-103.

骆钊,吴谕侯,朱家祥,等.基于多尺度时间序列块自编码 Transformer 神经网络模型的风电超短期功率预测[J].电网技术,2023,47(09):3527-3537.DOI:10.13335/j.1000-3673.pst.2022.2286.

王树斌,王旭,闫世平,等.基于 Transformer 的矿井内因火灾时间序列预测方法[J].工矿自动化,2024,50(03):65-70+91.DOI:10.13272/j.issn.1671-251x.2023100084.

Chen W, Wang W, Peng B, et al. Learning to rotate: Quaternion transformer for complicated periodical time series forecasting[C]//Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining. 2022: 146-156.

Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

Huggingface.transformers/src/transformers/time_series_utils.py[EB/OL].GitHub. https://github.com/huggingface/transformers/blob/main/src/transformers/time_series_utils.py#L164

Lewis M, Liu Y, Goyal N, et al. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension[J]. arXiv preprint arXiv:1910.13461, 2019.

Li S, Jin X, Xuan Y, et al. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting[J]. Advances in neural information processing systems, 2019, 32.

Liu S, Yu H, Liao C, et al. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting[C]//International conference on learning representations. 2021.

Liu Y, Wu H, Wang J, et al. Non-stationary transformers: Exploring the stationarity in time series forecasting[J]. Advances in Neural Information Processing Systems, 2022, 35: 9881-9893.

Nie Y, Nguyen N H, Sinthong P, et al. A time series is worth 64 words: Long-term forecasting with transformers[J]. arXiv preprint arXiv:2211.14730, 2022.

Radford A, Narasimhan K, Salimans T, et al. Improving language understanding by generative pre-training[J]. 2018.

Rogge N, Rasul K.Probabilistic Time Series Forecasting with Transformers[EB/OL].Hugging Face. https://huggingface.co/blog/time-series-transformers,2022

Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

Wu H, Xu J, Wang J, et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting[J]. Advances in neural information processing systems, 2021, 34: 22419-22430.

Zhou H, Zhang S, Peng J, et al. Informer: Beyond efficient transformer for long sequence time-series forecasting[C]//Proceedings of the AAAI conference on artificial intelligence. 2021, 35(12): 11106-11115.

Zhou T, Ma Z, Wen Q, et al. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting[C]//International conference on machine learning. PMLR, 2022: 27268-27286.