Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting

Haoyi Zhou, 1 Shanghang Zhang, 2 Jieqi Peng, 1 Shuai Zhang, 1 Jianxin Li, 1 Hui Xiong, 3 Wancai Zhang 4
1 Beihang University 2 UC Berkeley 3 Rutgers University 4 SEDD Company
zhouhy,pengjq,zhangs,lijx@act.buaa.edu.cn,shz@eecs.berkeley.eduxionghui,zhangwancaibuaa@gmail.com

摘要

许多实际应用需要长序列时间序列的预测，如电力消耗规划。长序列时间序列预测 (LSTF) 对模型的预测能力提出了很高的要求，即能够高效地精确捕捉输出与输入之间的长范围依赖耦合。最近的研究显示了 Transformer 在提高预测能力方面的潜力。

然而，Transformer 存在几个严重的问题，使其无法直接应用于 LSTF，包括二次时间复杂度、高内存使用以及编码器-解码器体系结构的固有限制。为了解决这些问题，本文设计了一个高效的基于变压器的 LSTF 模型，命名为 Informer，具有三个显著的特征:(i) ProbSparse 自注意机制，在时间复杂度和内存使用方面达到 O(L log L)，并且在序列依赖项对齐方面具有相当的性能。(ii) 自注意蒸馏通过将级联层输入减半来突出支配注意，并有效地处理极长输入序列。(iii) 生成式解码器虽然概念简单，但在一次向前操作中预测长时间序列，而不是一步一步地预测，这大大提高了长序列预测的推理速度。在 4 个大规模数据集上的大量实验表明，Informer 算法显著优于现有方法，为 LSTF 问题提供了一种新的解决方案。

关键词：时间序列；数据流；（深度）神经网络算法；能源

1 引言

时间序列预测是许多领域的关键因素，如传感器网络监测[1]、能源和智能电网管理、经济和金融[2]以及疾病传播分析[3]。在这些场景中，我们可以利用大量关于过去行为的时间序列数据来进行长期预测，即长序列时间序列预测 (LSTF)。但一直以来长序列预测并没有取得太好的效果。

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。最近的研究表明，Transformer 模型在捕获长期依赖关系方面表现出优于 RNN 模型的性能。Transformer 网络架构架构由 Ashish Vaswani 等人在 Attention Is All YouNeed 一文中提出，并用于机器翻译任务，和以往网络架构有所区别的是，该网络架构中，编码器和解码器没有采用 RNN 或 CNN 等网络架构，而是采用完全依赖于注意力机制的架构。自注意机制可以将网络信号传播路径的最大长度降低到理论上最短的 O(1)，并避免了循环结构，因此 Transformer 在 LSTF 问题上显示出很大的潜力。

然而，自注意力机制违反了要求 (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。因此自注意机制和 Transformer 架构的效率成为将其应用于 LSTF 问题的瓶颈。

Vanilla Transformer[4]在解决 LSTF 问题时有三个显著的局限性:

1. 自我注意的二次计算。自注意机制的原子运算，即标准点积，使得每层的时间复杂度和内存使用量为 O(L2)。

2. 内存瓶颈在堆叠层长输入。J 个编码器/解码器层的堆栈使得总内存使用为 O(J • L2)，这限制了模型在接收长序列输入时的可扩展性。

3. 预测长输出时速度下降。vanilla Transformer 的动态解码使得逐步推理与基于 RNN 的模型一样慢。

目前针对 Transformer 改进的模型，主要集中在限制 1 上，限制 2 和 3 在 LSTF 问题中仍然没有解决。为了解决上述问题，该论文作者为 LSTF 设计了一个高效的基于 Transformer 的模型，名为 Informer，其中明确地深入研究了自注意机制中的稀疏性，对网络组件进行了改进，并进行了大量的实验，最终解决了所有这些限制。Informer 模型实现了提高计算、内存和架构效率的同时保持更高的预测能力，具有以下三个独特的特征：

- 提出的 ProbSpare Self-attention 机制，能够替换规范 Self-Attention，使时间复杂度和内存使用量达到了 O(L log L)。
- 提出了自相关蒸馏（Self-attention Distilling）操作，将算法总时间复杂度降低到 $O\left((2-\varepsilon) L \log L\right)$。
- 提出了生成式 decoder（Generative Style Decoder），只需要一步即可输出整个解码序列，大大提高了长序列预测的推理速度，同时避免了推断期间的累计误差传播（cumulative error）。

2 相关工作

下面是长序列时间序列预测（LSTF）问题的相关文献综述。

2.1 时间序列预测

现有的时间序列预测方法大致可以分为两类: 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主力，具有可解释性和理论保证等吸引人的特性。现代扩展包括对缺失数据的支持和多种数据类型。基于深度学习的方法主要是利用 RNN 及其变体开发序列到序列的预测范式，取得突破性的性能[5]。

尽管已有的算法取得了很大的进展，但仍不能以令人满意的精度预测长序列时间序列。典型的最先进的方法[6]，特别是深度学习方法[7]，仍然是一种循序渐进的序列到序列预测范式，具有以下局限性:

1) 虽然可以实现向前一步的准确预测，但它们往往会受到动态解码累积误差的影响，导致 LSTF 问题误差较大[5]。预测精度随预测序列长度的增加而降低。

2) 由于梯度消失和内存约束的问题[8]，大多数现有方法无法从时间序列的整个历史的过去行为中学习。

在作者的工作中，Informer 的设计旨在解决这两个限制。

2.2 长序列输入问题

从上面的讨论中，我们把第二个局限性称为长序列时间序列输入 (LSTI) 问题。我们将探讨相关工作，并与论文中的 LSTF 问题进行比较。

研究人员对输入序列进行截断/总结/采样，以在实践中处理一个非常长的序列，但在进行准确预测时可能会丢失有价值的数据。Truncated BPTT[9]不修改输入，只使用最后的时间步骤来估计权重更新中的梯度，Auxiliary Losses[10]通过添加辅助梯度来增强梯度流。其他尝试包括 Recurrent Highway

Networks[11]和 Bootstrapping Regularizer[12]。这些方法试图改善循环网络长路径中的梯度流，但在 LSTI 问题中，随着序列长度的增加，其性能是有限的。基于 CNN 的方法[13]使用卷积滤波器来捕获长期依赖关系，并且它们的接受域随着层的堆叠呈指数级增长，这损害了序列对齐。

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依赖项。但 LSTF 问题寻求的是增强模型对长序列输出的预测能力，这需要建立输出和输入之间的长期依赖关系。因此，上述方法直接用于 LSTF 是不可行的。

2.3 注意力模型

Bahdanau 等人首先提出了 addictive attention[14]来改善翻译任务中编码器-解码器架构的单词对齐。然后，它的变体[15]提出了广泛使用的位置注意、一般注意和点积注意。最近，流行的基于 self-attention 的 Transformer[4]作为序列建模的新思维被提出，并取得了巨大的成功，特别是在 NLP 领域。通过将其应用于翻译、语音、音乐和图像生成，已经验证了更好的序列对齐能力。在原论文的工作中，Informer 利用其序列对齐能力，使其适用于 LSTF 问题。

2.4 基于 Transformer 的时间序列模型

相关工作大多数都是从在时间序列数据中应用 Transformer 的试验开始的，并且在 LSTF 预测中失败，因为他们使用了 vanilla Transformer。其它的一些工作[16]注意到了 self-attention 的稀疏性，原作者在主要的上下文中讨论了它们。

3 本文方法

3.1 本文方法概述

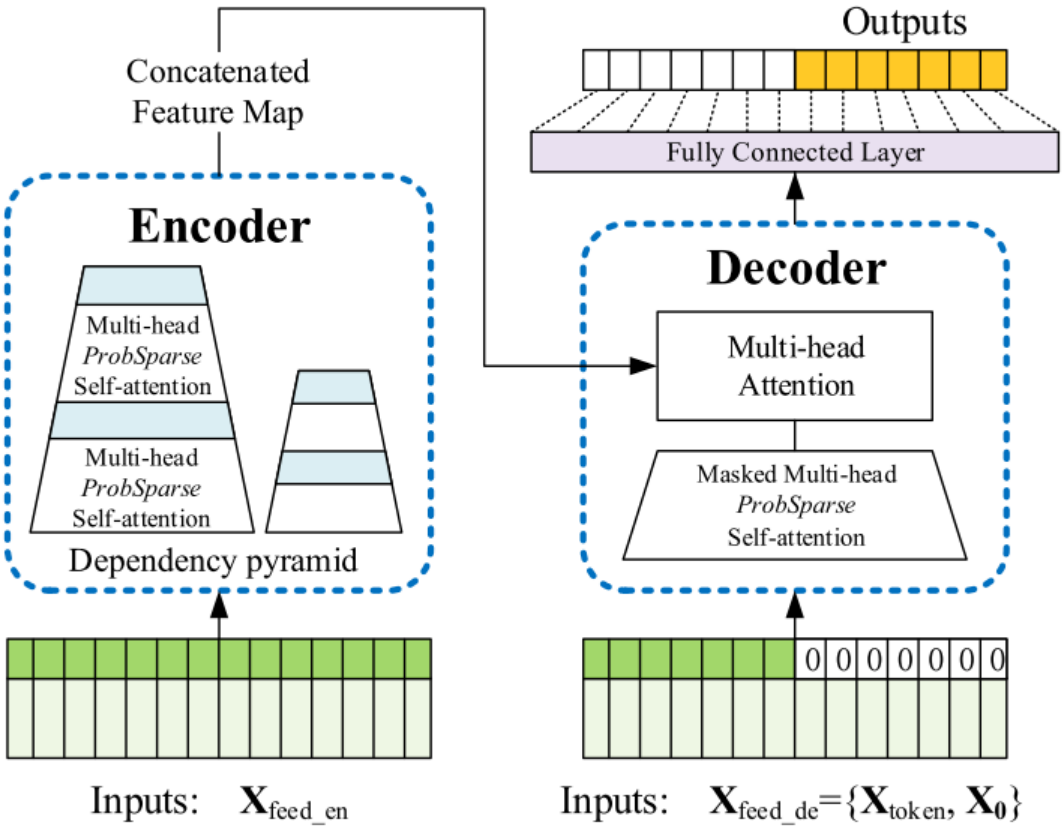

图 1: Informer 模型结构

上图为 Informer 的整体模型结构，左侧为编码器，其中的绿色序列为编码器接收的大量长序列输入。用论文中提出的 ProbSparse Self-attention 来代替标准的 Self-attention。白色梯形为稀疏注意力的计算，蓝色梯形为 Self-attention Distilling 操作，用以提取主要的 Self-attention，使得网络规模急剧缩小，层堆叠副本增加了鲁棒性。

右侧为解码器，其中的绿色序列为解码器接收的大量长序列输入，将目标元素填充为零，测量特征图的加权注意力成分，并立即以生成方式预测输出序列，即橙色序列。

3.2 高效的 Self-attention 机制

在 Informer 中的 ProbSparse self-attention，作者提到，虽然已经有很多优化 self-attention 的工作，但是他们：

- 缺少理论分析
- 对于 multi-head self-attention，每个 head 都采取相同的优化策略

传统的 self-attention 输入为 (query,key,value)，表示为：$A\left(Q,K,V\right)=Softmax\left(\frac{QK^T}{\sqrt{d}}\right)$，其中 $Q\in R^{L_Q*d}$, $K\in R^{L_K*d}$, $V\in R^{L_V*d}$，为了进一步探讨 Self-Attention 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 queery 的注意力被定义为 kernel 平滑的概率形式：

$$A\left(q_i,K,V\right)=\sum_j\frac{k\left(q_{i,}k_j\right)}{\sum_l k\left(1_{i,k_i}\right)}v_j=E_{p(k_j|q_i)}[v_j] \tag{1}$$

Self-Attention 需要 $O\left(L_QL_K\right)$ 的内存以及二次的点积计算代价，这是预测能力的主要因素。

根据刚刚提到的 query 重要性的度量方法，作者提出了 ProbSpare Self-Attention，允许每个 key 只关注 u 个

$$\mathrm{A}\left(Q,K,V\right)=softmax\left(\frac{\bar{Q}K^T}{\sqrt{d}}\right)V \tag{2}$$

其中 $\bar{Q}$ 是一个稀疏矩阵，宽度和 q 相同，但只包含 M 中最大的 $c\cdot lnL_Q$ 个 query。这样就只需要做 $O\left(lnL_Q\right)$ 次点乘，只需要 $O\left(L_KlnL_Q\right)$ 的内存。但是在计算 M 时还是需要 $O\left(L_KL_Q\right)$ 次计算，而且 LSE 的计算可能会出现数值不稳定的问题（上溢及下溢）。因此作者提出了一种近似计算的方法：

$$\bar{M}\left(q_i,K\right)=max_j\left\{\frac{q_ik_j^\top}{\sqrt{d}}\right\}-\frac{1}{L_K}\sum_{j=1}^{L_K}\frac{q_ik_j^\top}{\sqrt{d}} \tag{3}$$

这里取 max 的做法和 LSE 的近似计算类似。在长尾分布的前提下，只需要采样 $U=L_KlnL_Q$ 个点乘对进行计算。一般情况下 $L_Q=L_K=L$ 所以 ProbSpare Self-Attention 具有 O(LlnL) 的时空复杂度。

先前的一些尝试已经揭示了自注意概率的分布具有潜在的稀疏性，并且通过该方法解决了 Transformer 的第一个问题，即：self-attention 机制的二次计算复杂度问题。

3.3 编码器: 允许在内存使用限制下处理更长的顺序输入

编码器的设计目的是提取长序列输入的鲁棒长程依赖关系。在输入表示之后，第 t 个序列输入 χ^t 被塑造成一个矩阵 $X_{en}^t\in\mathbb{R}^{L_x\times d_{model}}$。为了清晰起见，在图 (3) 中给出了编码器的草图。

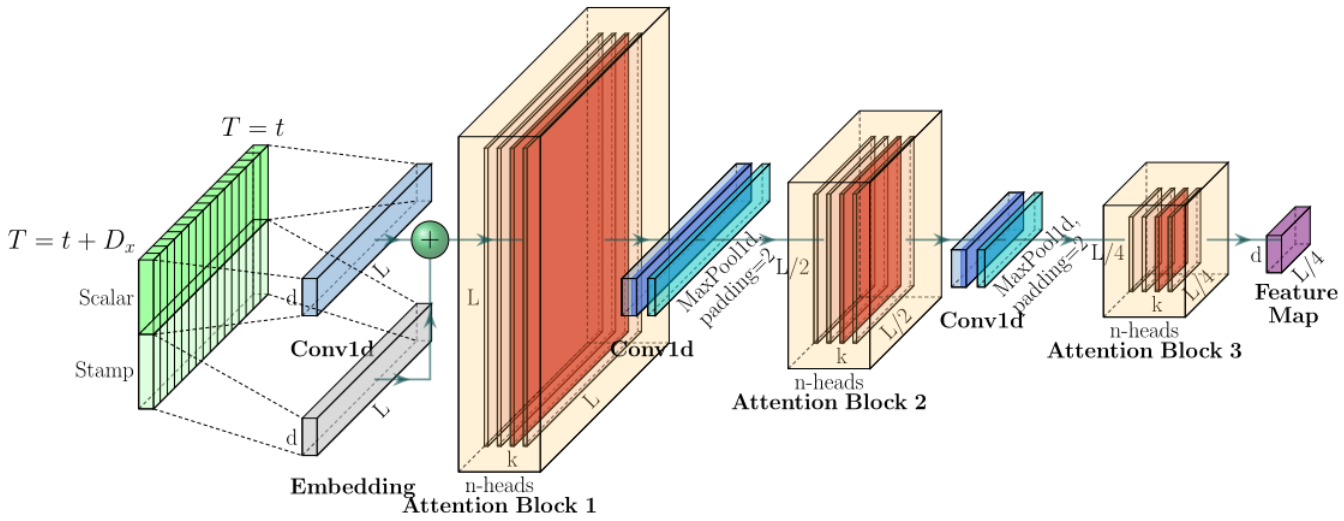

图 2: Informer 的编码器结构

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（多余的重复或啰嗦内容）组合。Informer 使用 distilling 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 2 中，我们可以看到注意力块的 n 个头权重矩阵 (重叠的红色方框)，它对输入的时间维度进行了大幅修剪。受扩张卷积的启发[17]，Informer 的“Distilling”过程从第 j 层向前推进到 (j + 1)-第 (j + 1) 层:

$$X_{j+1}^{t} = MaxPool\left(ELU\left(Convld\left(\left[X_{j}^{t}\right]_{AB}\right)\right)\right) \tag{4}$$

其中 $[\cdot]_{AB}$ 表示注意力块。它包含多头 ProbSparse self-attention 和基本操作，其中 Conv1d(•) 在时间维度上使用 ELU(•) 激活函数执行 1-D 卷积滤波器 (内核宽度 =3)。在堆叠一层后，Informer 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 $O\left((2-\varepsilon)\, LlogL\right)$，其中 ε 是一个很小的数字。

为了增强蒸馏操作的鲁棒性，Informer 构建了输入减半的主堆栈副本，并通过一次删除一层来逐步减少 self-attention distilling 层的数量，就像图 (2) 中的金字塔，从而使其输出维度对齐。因此，我们将所有堆栈的输出连接起来，并得到编码器的最终隐藏表示。

3.4 解码器：通过一个正向过程生成长序列输出

使用图一中的标准解码器结构[4]，它由两个相同的多头注意力层的堆栈组成。在此基础上，采用生成推理的方法，缓解了长时间预测的速度骤降问题，给解码器提供如下向量：

$$x_{de}^{t} = Concat\left(X_{token}^{t}, X_{0}^{t}\right) \in R^{(L_{token}+L_{y})\times d_{model}} \tag{5}$$

其中 $X_{token}^{t} \in R^{L_{token}\times d_{model}}$ 是开始标记，$X_{0}^{t} \in R^{L_{y}\times d_{model}}$ 是目标序列的占位符（设标量为 0）. 将 Masked multi-head attention 应用于 ProbSparse Self-attention 计算中，并把 masl 的点积设置为-∞。可以防止每个位置都关注接下来的位置，从而达到了避免自回归的目的。一个完全连接的层获得最终的输出，它的超大小取决于正在执行的任务是单变量预测还是多变量预测。

3.5 生成推理

起始标记（Start token）在 NLP 的“动态解码”中得到了有效应用，Informer 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_{token} 长序列，例如输出序列

之前的一个片段。以预测 168 点为例 (实验部分为 7 天温度预测)，将已知的目标序列前 5 天作为“start token”，用 $X_{de} = \{X_{5d}, X_0\}$ 给生成式推理解码器喂食。X_0 包含目标序列的时间戳，即目标周的上下文。然后，Informer 提出的解码器通过一个前向过程来预测输出，而不是传统编码器-解码器架构中耗时的“动态解码”。

3.6 Informer 的 Embedding 层

1) 输入 Embedding：

   输入的 Embedding 采用一维卷积神经网络计算得到，随整体模型训练而优化。

2) 位置 Embedding：

   输入数据的位置 Embedding 采用 PE 算法计算。

3) 时间 Embedding：

   输入数据的时间 Embedding 采用全连接神经网络计算得到，随整体模型训练而优化。

   故 Informer 的 Embedding 层的表达式为：

$$\chi_{feed[i]}^{t} = \alpha u_i^t + PE_{(L_x \times (t-1)+i,)} + \sum_p [SE_{(L_x \times (t-1)+i)}]_p$$

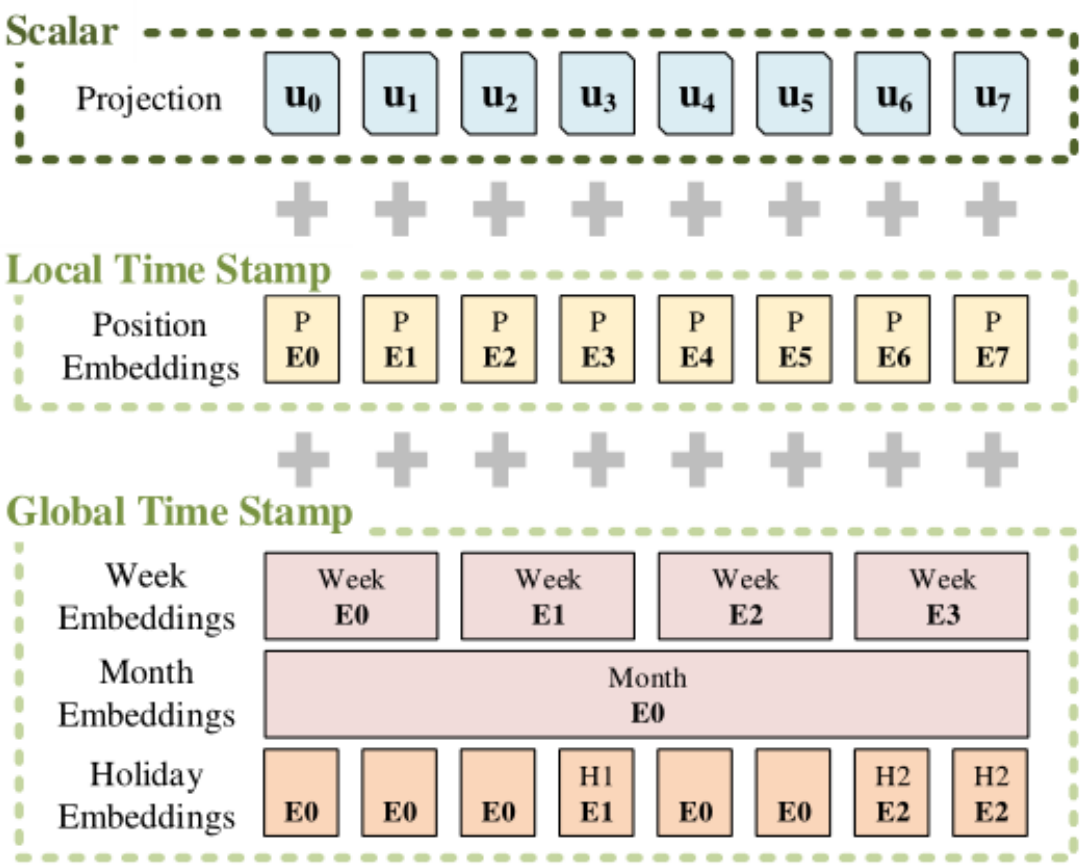

图 3: Informer Embedding 组成原理图

3.7 损失函数定义

论文在预测 w.r.t 目标序列时选择 MSE 损失函数作为最终的 loss，并将损失从解码器的输出传播回整个模型。

4 复现细节

4.1 与已有开源代码对比

本研究成功复现了 Informer 模型，Informer 是一种全新的长时间序列预测模型，基于 Transformer 架构，包含编码器、解码器以及注意力机制。此外，本研究还成功地应用 Informer 模型来预测股票价格走势。由于 Informer 主要处理的是以小时或分钟为单位采集的时间序列数据，而股票数据则是根据交易日进行计算，因此我在此基础上重新构建了 Informer 中的自定义数据处理类，并将其应用于 CSI300 数据集中的收盘价预测。最后，为了方便实时掌握训练效果，另外编写了 plot 程序，采用 matlab 中的二维画图函数 plot，实时绘制并输出 Learning Rates 以及 losses 的图像。

代码参考开源地址：https://github.com/zhouhaoyi/Informer2020

4.2 实验环境搭建

实验基于 pytorch 实现，运行在单个 NVIDIA GPU 上，核心包的环境如下：

- python 3.6
- matplotlib == 3.1.1
- numpy == 1.19.4
- pandas == 0.25.1
- scikit_learn == 0.21.3
- torch == 1.8.0

4.3 界面分析与使用说明

代码文件说明：

- checkpoints/：检查点文件
- data/：存放数据集

  data_loader.py：加载数据、数据预处理
- exp/：训练、测试循环代码

  exp_basic.py：训练基类

  exp_informer.py：训练主类
- model/：模型库，包含了模型的详细结构实现
- results/：存放训练结果
- utils/：通用工具，包含了模型的评估指标、时间轴的时间特征处理、指数缩减学习率、提前停止训练策略、数据标准化策略等功能，以及作者提出的新型 mask。
- main_informer.py：项目的启动入口，包含程序入口、变量初始化设置等（开始训练和测试）。
- plot_history.py：采用 matlab 中的二维画图函数 plot，绘制历史数据图。
- process_custom_data.py：输出最近两次运行结果中的 MSE 和 MAE 值。

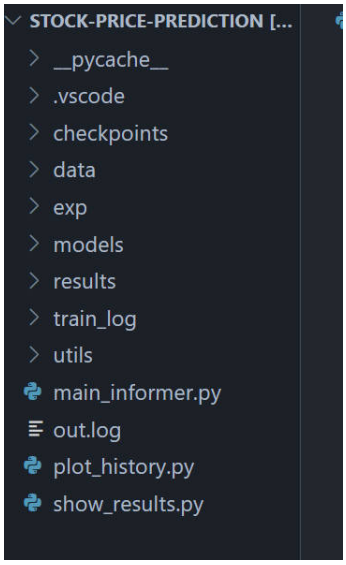

图 4: 代码结构图

分别在数据集 ETTh1、ETTh2 和 ETTm1 上使用 ProbSparse 自注意训练和测试模型的命令:

- ETTh1

  python -u main_informer.py –model informer –data ETTh1 –attn prob –freq h

- ETTh2

  python -u main_informer.py –model informer –data ETTh2 –attn prob –freq h

- ETTm1

  python -u main_informer.py –model informer –data ETTm1 –attn prob –freq t

4.4 创新点

当不使用作者提供的 ETT 数据集时，应该使用 data_loader.py 中的 Dataset_Custom 类来进行数据读入，作者同时提供了基于 pandas dataframe 的数据结构，可以读入 excel、csv 和 txt 等格式的数据文件。因此本项目对 Dataset_Custom 类进行了重写，以读取 CSI300.csv 中的数据。

所使用的自定义数据集 CSI300：

- 名称：CSI300
- 下载：https://cn.investing.com/
- 时间范围：2007/01/04 - 2022/01/28
- 数据量：3670 条记录
- 特征数量：7 个变量
- 目标变量：”T(close)”

此外还编写了 plot_history.py 文件，它可用于同步输出学习率以及损失值的图像，可在 checkpoint 路径下进行查看，效果如下图所示：

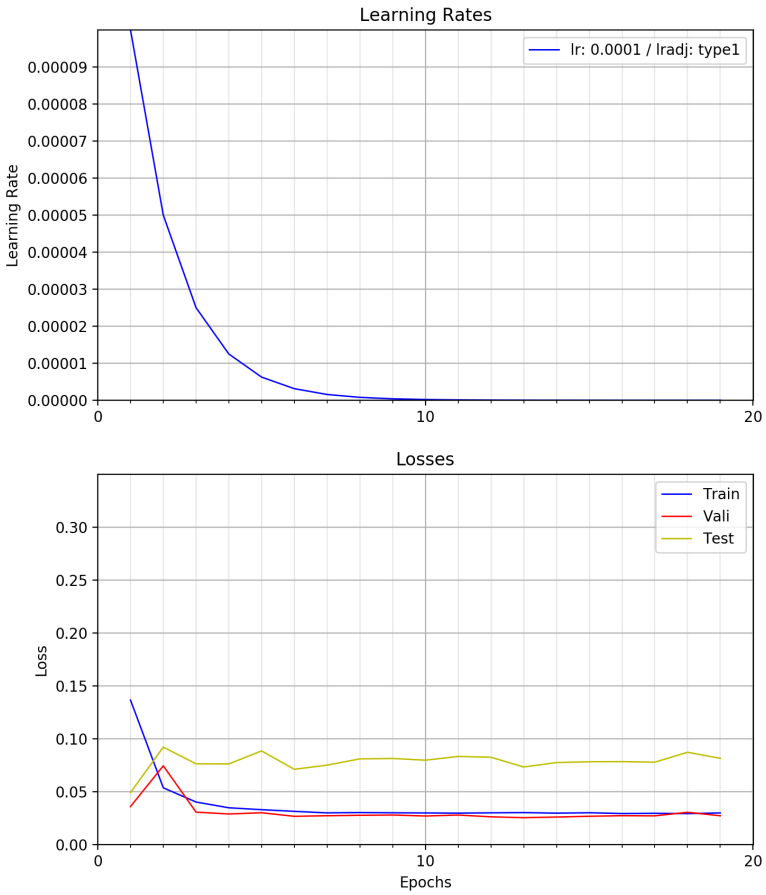

图 5: Learning Rates and losses

如图 5 所示，对 Learning Rates 和 losses 的值进行了输出，从图表中的走势可以看出，Informer 在股票价格的预测上有着一定的效果。

show_result.py 用于输出最近保存的两次实验结果中的 MSE 和 MAE 值，运行效果如下图所示。

```
(informer01) wangshu@gpu02:~/Stock-Price-Prediction$ /home/wangshu/anaconda3/envs/informer01/bin/python /home/wangshu/Stock-Price-Prediction/show_results.py

Metrics of informer_CSI300_ftMS_sl30_ll15_pl7_dm512_nh8_el2_dl1_df2048_atprob_fc5_ebtimeF_dtTrue_mxTrue_test_0:
MSE:    0.09105136 | MAE:  0.24275866

Metrics of informer_CSI300_ftMS_sl30_ll15_pl7_dm512_nh8_el2_dl1_df2048_atprob_fc5_ebtimeF_dtTrue_mxTrue_test_1:
MSE:    0.07956496 | MAE:  0.22703436

(informer01) wangshu@gpu02:~/Stock-Price-Prediction$
```

图 6: 输出最近两次实验的结果

5 实验结果分析

5.1 复现论文实验精度

单变量预测结果 - Univariate forecasting results （S）

表 1: 单变量预测结果

数据集	序列长度	标签长度	预测长度	MSE	MAE
ETTh1	48	48	24	0.0790	0.2185
ETTh1	96	48	48	0.1200	0.2774
ETTh1	168	168	168	0.3253	0.4972
ETTh1	336	168	168	0.1507	0.3165

多变量预测结果 - Multivariate forecasting results（M）

表 2: 多变量预测结果

数据集	序列长度	标签长度	预测长度	MSE	MAE
ETTh1	48	48	24	0.5164	0.5169
ETTh1	96	48	48	0.8009	0.6920
ETTh1	168	168	168	1.0684	0.8358
ETTh1	336	168	168	1.3587	0.9492

由表 1可知，在单变量的设定下，随着预测长度不断从 24 增加到 168，在 ETTh1 数据集上的 MAE（平均绝对值误差）值，分别达到 0.2185、0.2774、0.4972、0.3165。由表 2可知，在多变量的设定下，随着预测长度不断从 24 增加到 168，在 ETTh1 数据集上的 MAE 值，分别达到 0.5169、0.6920、0.8358、0.9492。

5.2 自定义数据集的实验结果

复现完原论文提出的 Informer 模型所相关的实验后，进行自定义数据集的实验。

```
(informer01) wangshu@gpu02:~/Stock-Price-Prediction$ /home/wangshu/anaconda3/envs/i
nformer01/bin/python /home/wangshu/Stock-Price-Prediction/main_informer.py
Args in experiment:
Namespace(activation='gelu', attn='prob', batch_size=32, c_out=1, checkpoints='./ch
eckpoints/', cols=None, d_ff=2048, d_layers=1, d_model=512, data='CSI300', data_pat
h='CSI300.csv', dec_in=6, des='test', detail_freq='d', devices='0,1,2,3', distil=Tr
ue, do_predict=False, dropout=0.05, e_layers=2, embed='timeF', enc_in=6, factor=5,
features='MS', freq='d', gpu=0, inverse=False, itr=1, label_len=15, learning_rate=0
.0001, loss='mse', lradj='type1', mix=True, model='informer', n_heads=8, num_worker
s=0, output_attention=False, padding=0, patience=10, pred_len=7, root_path='./data'
, s_layers=[3, 2, 1], seq_len=30, target='Close', train_epochs=50, use_amp=False, u
se_gpu=True, use_multi_gpu=False)
Use GPU: cuda:0
>>>>>>>start training : informer_CSI300_ftMS_sl30_ll15_pl7_dm512_nh8_el2_dl1_df2048
_atprob_fc5_ebtimeF_dtTrue_mxTrue_test_0>>>>>>>>>>>>>>>>>>>>>>>>>>
./data/CSI300.csv
train 2533
./data/CSI300.csv
val 361
./data/CSI300.csv
test 728
50
Epoch: 1 cost time: 13.833788394927979
Epoch: 1, Steps: 79 | Train Loss: 0.1267508 Vali Loss: 0.0412803 Test Loss: 0.08449
```

图 7: 实验参数

在图 7中，我们将参数打印出来。可以发现，此时的序列长度为 30，标签长度为 15，预测长度为 7。这意味着以 30 天的数据作为输入，窗口大小为 15，每次预测后 7 天的收盘价。

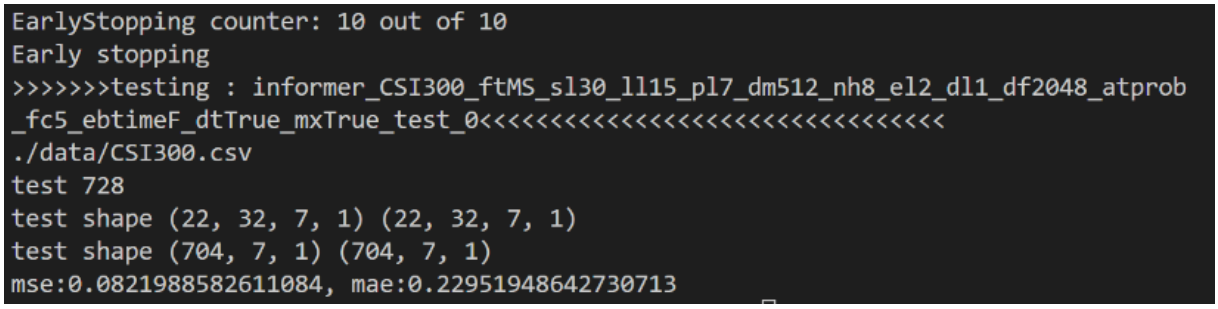

图 8: 实验结果

由图 8可知，在上述参数设置下，最终成果基于 Informer 实现股票收盘价预测，实验结果中 MSE 值约为 0.0822，MAE 值约为 0.2295。

6 总结与展望

论文研究了长序列时间序列的预测问题，作者提出了一种基于 Informer 的长序列预测方法。具体来说，该方法通过设计 ProbSparse 自注意力机制和 distilling 操作，克服了 vanilla Transformer 中二次时间复杂度和二次内存使用的挑战。此外，精心设计的生成式解码器减轻了传统编码器-解码器架构的局限性。

本项目以 Informer 为基础，实现了股票预测功能，不足之处未对预测值进行反归一化处理，同时评估预测效果的参数还可以考虑从从 MSE、MAE 转换为 ACC。

参考文献

[1] PAPADIMITRIOU S, YU P. Optimal Multi-Scale Patterns in Time Series Streams[M]. Association for Computing Machinery, 2006: 647-658.

[2] ZHU Y, SHASHA D. Chapter 32 - StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time.[G]//VLDB '02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann, 2002: 358-369.

[3] MATSUBARA Y, SAKURAI Y, van PANHUIS W G, et al. FUNNEL: Automatic Mining of Spatially Coevolving Epidemics[C]//: vol. 10. Association for Computing Machinery, 2014.

[4] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is All you Need[C]//Advances in Neural Information Processing Systems: vol. 30. Curran Associates, Inc., 2017.

[5] LIU Y, GONG C, YANG L, et al. DSTP-RNN: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction[J]. Expert Systems with Applications, 2020, 143: 113082.

[6] SEEGER M, RANGAPURAM S, WANG Y, et al. Approximate Bayesian Inference in Linear State Space Models for Intermittent Demand Forecasting at Scale[Z]. 2017. eprint: arXiv:1709.07638.

[7] LI Y, YU R, SHAHABI C, et al. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting[Z]. 2017. eprint: arXiv:1707.01926.

[8] SUTSKEVER I, VINYALS O, LE Q V. Sequence to Sequence Learning with Neural Networks[C]//Advances in Neural Information Processing Systems: vol. 27. Curran Associates, Inc., 2014.

[9] AICHER C, FOTI N J, FOX E B. Adaptively Truncating Backpropagation Through Time to Control Gradient Bias[C]//: vol. 115. PMLR, 2020.

[10] TRINH T, DAI A, LUONG T, et al. Learning Longer-term Dependencies in RNNs with Auxiliary Losses [C]//: vol. 80. PMLR, 2018.

[11] ZILLY J G, SRIVASTAVA R K, KOUTNÍK J, et al. Recurrent Highway Networks[C]//: vol. 70. PMLR, 2017.

[12] CAO Y, XU P. Better Long-Range Dependency By Bootstrapping A Mutual Information Regularizer [C]//: vol. 108. PMLR, 2020.

[13] STOLLER D, TIAN M, EWERT S, et al. Seq-U-Net: A One-Dimensional Causal U-Net for Efficient Sequence Modelling[Z]. 2019. eprint: arXiv:1911.06393.

[14] BAHDANAU D, CHO K, BENGIO Y. Neural Machine Translation by Jointly Learning to Align and Translate[Z]. 2014. eprint: arXiv:1409.0473.

[15] LUONG M T, PHAM H, MANNING C D. Effective Approaches to Attention-based Neural Machine Translation[Z]. 2015. eprint: arXiv:1508.04025.

[16] LI S, JIN X, XUAN Y, et al. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting[C]//: vol. 32. Curran Associates, Inc., 2019.

[17] YU F, KOLTUN V, FUNKHOUSER T. Dilated Residual Networks[C]//. 2017.