中图分类号：TP391

论 文 编 号：10006ZY1706148

北京航空航天大學

专业硕士学位论文

基于深度学习的集群 trace 时序数据挖掘算法研究

作者姓名 柏宇

学科专业 计算机技术

指导老师 阮利 讲师

培养院系 计算机学院

Research on Cluster Trace Time-series Data Mining Algorithm Based on Deep Learning

A Dissertation Submitted for the Degree of Master

Candidate : Bai Yu

Supervisor: Lecturer Ruan Li

School of Computer Science and Engineering

Beihang University, Beijing, China

中图分类号：**TP391**

论 文 编 号：**10006ZY1706148**

硕　士　学　位　论　文

基于深度学习的集群 trace 时序数据挖掘算法研究

作者姓名　柏宇　　申请学位级别　工程硕士

指导老师姓名　阮利　　职　　称　　讲师

学科专业　计算机技术　　研究方向　　计算机系统结构

学习时间自　　年　月　　日　起至　　年　月　日止

论文提交日期　年　月　　日　论文答辩日期　年　月　日

学位授予单位　北京航空航天大学　学位授予日期　年　月　日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：_____ 日期：　　年　　月　　日

学位论文使用授权

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：_____ 日期：　　年　　月　　日

指导教师签名：_____ 日期：　　年　　月　　日

摘　要

集群以其高性价比、高可扩展性等优势，已经成为当前高性能计算和云计算系统的主流架构。近年来，如何对大规模、高动态性和异构的集群系统进行高效准确的性能分析，已经成为大规模集群系统降低成本、增强可靠性以及优化大规模应用面临的关键挑战之一。集群运行时跟踪日志（trace）数据因蕴含系统的负载波动、节点配置等重要的时空维度信息，是集群性能分析的最重要的数据集之一。集群 trace 的时序数据挖掘关系到集群监控、任务调度、负载均衡等重要系统资源管理场景，日渐成为当前国内外集群性能分析领域的重要研究课题之一。

一方面，虽然时序数据是对系统性能量化分析的有效着手点，现有的在计算机系统性能分析领域从时序数据挖掘角度开展的研究依然相对缺乏并且不够深入。包括存在：现有的时序负载预测往往关注点值预测而缺乏对趋势转折点的预测，以及在任务失败预测方面其精度仍需提高并且往往忽略实际应用场景等一系列问题。另外一方面，虽然基于深度学习的时序数据挖掘已经在金融和社交媒体数据分析领域取得了优异的表现，但是由于数据规模、数据特征和应用场景等方面存在的巨大差异，导致其它领域基于深度学习的时序数据挖掘算法并不能直接应用到集群 trace 时序数据上来。因此，基于深度学习模型构建适于 trace 数据领域特性的时序挖掘算法仍然是一个充满挑战性的研究课题。

针对上述问题，本文研究基于深度学习的集群 trace 时序数据挖掘算法，通过对运行时负载数据的特征分析，构建预测模型，实现对未来负载状态的准确预测，主要工作和创新点包括：

1）在负载转折点预测方面，针对现有方法中特征表达能力较弱、且基于单一时间窗口的特征构造方法无法描述时间维度的变化模式这两个问题，提出了一种特征增强的多任务 LSTM 负载转折点预测算法。该算法首先提取四种时序特征描述云环境下服务器负载转折点的波动特性。然后采用特征增强的多任务 LSTM 模型，将特征序列和原始负载序列分别视为两种不同的任务进行学习，以更好的提取波动模式；最后采用显式的融合结构，学习特征序列和原始负载序列之间的隐含关系，达到特征增强的效果。在谷歌集群 trace 上进行的实验表明该算法在 F1 指标上相比基线模型最少平均提升 2 个百分点。

2）在任务失败预测方面，针对监测指标序列内部和各检测指标序列之间存在的两种不同的特征难以同时提取的问题，提出了一种特基于通道注意力卷积模型的任务失败预测算法。该算法包含两部分，第一部分是通道内一维卷积模块，将每种监测指标序列视为通道，分别提取通道内特征；第二部分是通道间多头自注意力模块，负责提取不同通道间的交互特征。在谷歌集群 trace 上进行的实验表明该算法在 AUC 指标上超过了两种专门用于失败任务预测的深度模型以及目前最优的 ResNet 多变量时间序列分类模型。

3）在 I/O 负载预测方面，针对传统机器学习方法难以捕捉强烈波动的 I/O 负载时序特征这个问题，提出了一种特基于长短期记忆网络的存储系统 I/O 负载短期时序预测算法。该方法包括两部分，第一部分是负载数据预处理与后处理，预处理采用基于窗口的归一化方式减弱负载波动，后处理采用相反的操作恢复数据；第二部分是基于长短期记忆网络的时序预测模型，使用多对一的结构提取时序变化模式以进行负载预测。在实际的存储负载 trace 上进行的实验表明该算法相比 3 种典型预测方法在 MAPE 上有至少 1.1% 的提升。

关键字：时序数据，负载转折点预测，任务失败预测，负载预测，trace

Abstract

Clustering has the advantages of high cost performance and high scalability, and has become the mainstream architecture of current high-performance computing and cloud computing systems. In recent years, how to perform efficient and accurate performance analysis on large-scale, highly dynamic, and heterogeneous cluster systems has become one of the key challenges for large-scale cluster systems to reduce costs, enhance reliability, and optimize large-scale applications. The cluster runtime trace data is one of the most important data sets for cluster performance analysis because it contains important spatio-temporal information such as system load fluctuations and node configuration. The time-series data mining of cluster Trace is related to important system resource management scenarios such as cluster monitoring, task scheduling, and load balancing. It has become one of the important research topics in the field of cluster performance analysis at home and abroad.

However, on the one hand, although time-series data is an effective starting point for quantitative analysis of system performance, the existing research in the field of computer system performance analysis from the perspective of time-series data mining is still relatively lacking and not deep enough. There are a series of problems, such as: the existing time-series load prediction often focuses on point value prediction but lacks the prediction of the turning point of the trend, and the accuracy of the task failure prediction still needs to be improved, and the actual application scenario is often ignored. On the other hand, although time-series data mining based on deep learning has achieved excellent performance in the fields of financial and social media data analysis, due to the huge differences in data scale, data characteristics, and application scenarios, the deep learning based time-series data mining algorithm in other fields cannot be directly applied to cluster trace time series data. Therefore, constructing a deep learning based time-series data mining algorithm suitable for the characteristics of the trace data is still a challenging research topic.

In view of the above problems, this paper studies deep learning based cluster trace time-series data mining algorithms. By analyzing the characteristics of runtime load data, prediction models are built to achieve accurate prediction of future load states. The main work and

innovations include:

1) In the aspect of load turning point prediction, a feature-enhanced multi-task LSTM load turning point prediction algorithm is proposed, in order to solve the problems existing in existing methods, the feature representation ability is weak, and the feature construction method based on a single time window cannot be described The change pattern of the time dimension, these two issues. The algorithm first extracts four time-series features to describe the fluctuation characteristics of server load turning points in the cloud environment. Then use a feature-enhanced multi-task LSTM model to treat the feature sequence and the original load sequence as two different tasks for learning, in order to better extract the fluctuation pattern; finally, a fusion structure is used, which can learn the implicit relationship between the feature sequence and the original load sequence and help to achieve the effect of feature enhancement. Experiments on the Google cluster trace show that the algorithm improves the F1 metrics by at least 2 percentage points on average compared to the baseline model.

2) In terms of task failure prediction, A task failure prediction algorithm based on the a channel-wise multi-headed attention convolutional neural network is proposed which aims to solving the difficulty of simultaneously extracting intra-channel features existing in the monitoring metrics sequence and inter-channel features accross the monitoring metrics sequence. The algorithm consists of two parts. The first part is a one-dimensional convolution module, which treats each monitoring metrics sequence as a channel, and then extracts the intra-channel features separately. The second part is the multi-head self-attention module between channels, which is responsible for extracting inter-channel features. Experiments on the Google cluster trace show that the proposed model surpasses two depth models dedicated to failure task prediction and the current optimal ResNet multivariate time series classification model in the AUC metrics.

3) In terms of I/O load prediction, in order to solve the problem that traditional machine learning methods are difficult to capture the fluctuating I/O load temporal characteristics, a short-term time-series prediction algorithm for I/O load based on long-short term memory networks is proposed. The method includes two parts. The first part is preprocessing and post-processing of the load data. The pre-processing uses window-based normalization to reduce

load fluctuations, and the post-processing uses the opposite operation to recover the data. The second part is a time series prediction model based on long-short term memory networks. It uses a many-to-one structure to extract temporal patterns for load prediction. The experiments performed on the actual storage load trace show that this method has at least a 1.1% improvement in MAPE metrics compared to the three typical prediction methods.

Key words: time-series data, workload turning point prediction, task failure prediction, workload prediction, trace

目 录

图 清 单

表 清 单

第一章　绪论

1.1　课题来源

本论文来源于国家重点研发计划课题《面向 E 级系统的运行时架构研究和应用验证》课题，该课题研究面向 E 级系统的统一运行时架构，探索多级混合运行模型，研究通信隐藏、高可扩展分布式负载均衡等技术，研究大规模并行系统弹性可恢复技术，研究大规模通信可扩展和通信优化等技术。

本论文研究基于深度学习模型的集群 trace 时间序列挖掘方法，预测系统未来的负载状态，为自适应的智能集群系统管理与性能优化提供决策支持。

1.2　研究背景与意义

集群以其高性价比、高可扩展性等优势，已经成为当前高性能计算和云计算系统的主流架构。近年来，用户和应用数量的激增，导致了以高性能计算机和云计算平台为代表的集群规模都达到了万级以上，例如在 2019 年 6 月公布的超级计算机 top500 中，我国的神威太湖之光取得了第三的排名，该集群系统配备了多达 40960 个节点[1]。2018 年中国信息通信研究院发布的数据中心白皮书[2] 显示截止 2017 年底，全球大型数据中心达到了 1314 个，而 Google、Microsoft、eBay、Yahoo、Facebook、Amazon 等互联网公司均拥有超过十万台量级的服务器集群。

集群性能量化分析一直以来都是大规模集群进行性能优化的重要前提之一，是大规模集群领域一直以来的热点研究课题。并且近年来，如何对大规模、高动态性和异构的集群系统进行高效准确的性能分析，已经成为大规模集群系统降低成本、增强可靠性以及优化大规模应用面临的关键挑战之一，是当前国内外的热点研究方向。

集群运行时跟踪日志（tarce）是计算机集群在运行过程中采集到的系统监控记录，这些记录往往包含了系统负载、节点配置等重要的时间维度和空间维度的信息，隐含了系统在时空维度的运行特性。国内外一些实验室和互联网公司均相继发布了自己集群中的 trace 数据用以推动基于 trace 数据的系统优化研究，例如谷歌在 2011 年发布了其云服务集群中的 trace 数据[3]，用以研究工作负载的变化模式以优化任务调度算法。该集群 trace 是一个包含 12000 多台服务器、2500 万个任务的云服务集群的 trace 记录，仅

29 天的监测数据就达到了 41G。阿里巴巴在 2017 年发布了一个集群 trace[4]，记录了一个包含 13000 台服务器的商业集群中，时长 12 小时的作业运行监测数据，用以研究如何提升其集群资源利用率。美国洛斯阿拉莫斯国家实验室（LANL）[5] 发布了一个包含 256 节点的高性能计算集群 trace，该 trace 包含作业启动时间、终止时间，终止状态等数据，用以研究科学计算等高性能计算任务的调度。可见，对集群 trace 数据进行挖掘是开展系统性能优化的关键着手点之一。

由于 trace 数据本身拥有时序属性，它反映了系统在每个采样时刻的负载状态，是一种典型的系统性能大数据。对系统运行时产生的历史数据，通过时间序列挖掘方法对其未来状态进行预测，是集群监控、任务调度、负载均衡等重要系统资源管理的量化依据。时序数据挖掘包括时间序列点值预测[6]、时间序列趋势预测[7]、时间序列分类[8] 等具体任务，目前已经在金融[9]、社交媒体等领域得到广泛应用。然而，虽然时序数据挖掘算法是对 trace 数据进行分析的有效手段，但是在计算机系统性能分析领域从时序数据挖掘角度开展的研究仍然相对缺乏。现有的其它领域的时序挖掘算法，由于在数据规模，数据特征，应用场景等方面存在的差异，在系统 trace 时序数据上具有很强的不适应性，需要进行优化。

综上所述，一方面，随着集群规模、动态性和异构性的增强，集群系统积累了大量的 trace 时序数据，集群 trace 时序数据挖掘是集群系统性能分析的关键研究领域之一。另一方面，如何针对集群时序数据的特征，建立一套适于集群 trace 数据的预测算法，进而为系统监控、调度和均衡等子系统提供更为准确的数据决策分析依据，对云平台等大规模集群系统的性能提升、智能化管理具有重要的理论意义和工业应用价值。

1.3 研究内容与主要贡献

1.3.1 研究目标与研究内容

本文研究基于深度学习的集群 trace 时序数据挖掘算法，通过对运行时负载数据的特征分析，构建预测模型，实现对未来负载状态的准确预测。重点研究基于特征增强的多任务 LSTM 负载转折点预测算法、基于通道注意力卷积模型的任务失败预测算法、基于长短期记忆网络的存储系统 I/O 负载短期时序预测算法。本文的研究点结构如图1虚线框所示。

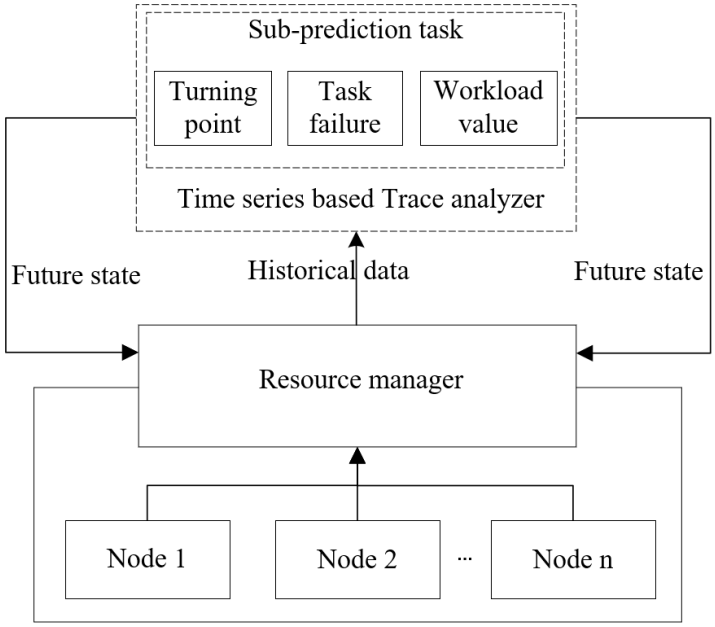


图 1 本文研究点结构图

1.3.2 主要贡献

本文主要创新之处和贡献分析如下：

1）提出了一种基于特征增强的多任务 LSTM 负载转折点预测算法。

云环境下服务器工作负载的转折点 (workload turning point) 即局部趋势的逆转点，是代表系统压力的峰值点或者是代表较低负载的谷值点，而非转折点则处于距离其最近的上一个转折点所转向的趋势中。预测这类点是向系统管理人员发出预警、采取合理的调度管理策略的前提。与点值负载预测方法相比，转折点预测可以提供未来负载变化的趋势信息，即上升趋势或下降趋势，因此也为设计更精细的资源管理方案提供了理论指导。

转折点意味着趋势的反转，本质上是波动规律的变化，而现有方法所使用的特征大都是基础的时间窗口统计特征，如均值，方差等，这些基础特征在表现负载波动性方面能力较弱；与此同时，现有的转折点预测模型往往只基于预测点附近的单个时间窗口内的统计特征构建分类器，而忽略了系统状态在时间维度的变化规律。本文针对这两个问题研究云环境下服务器负载转折点预测算法。基于自主提出的的时序波动性特征构建多任务 LSTM 结构，并采用融合层从原始负载序列和特征序列中学习隐含的特征，以达到特征增强的目的。

2）提出了一种基于通道注意力卷积模型的任务失败预测算法。

由于计算机群庞大的规模以及可能存在的异构特性，运行于其上的任务有一定概

率会执行失败，然而失败的任务消耗了大量的软硬件资源，造成资源浪费，所以在任务执行失败之前预测出这些任务以避免资源浪费就成为一个重要的问题。目前的任务失败预测方法的基本思路是采集任务运行时的各种监测指标，如 CPU 利用率，内存利用率等，然后采用机器学习或者深度学习模型对这些监测指标所构成的时间序列进行分类，以预测对应任务的终止状态。当前主流的预测算法主要使用采用循环神经网络结构，即将每一时刻的各种监测指标组成向量，采用循环神经网络捕捉各个时刻输入监测指标向量之间的时序依赖关系。

然而针对这种多监测指标数据，存在两个问题：其一，不同的监测指标序列在相同的时刻受到之前时刻的影响程度可能是不同的，即不同变量内部特征具有独立性；其二，不同的监测指标序列的时序变化也相互关联，即不同变量间的特征具有相关性。如果过将同一个时刻的监测指标同时输入循环神经网络模型，则很难捕捉到这种多变量内部和变量之间的不同作用关系。目前，虽然有些先进的基于深度学习的多变量分类模型考虑到了不同变量内部的作用关系的差异，但是对于变量间的作用关系却很少考虑到。所以针对这个问题，本文在云计算任务失败预测场景下，研究基于通道注意力机制的多变量时间序列分类模型。采用堆叠的一维卷积模块分别对每种监测指标序列进行时序特征提取，然后基于多头注意力机制学习不同序列之间的交互特征，以同时考虑上述两个问题，并研究在任务运行的不同时期进行预测的效果。

3）基于长短期记忆网络的存储系统 I/O 负载短期时序预测算法。

随着大数据时代的到来，存储系统的性能成为制约许多数据密集型应用发展的瓶颈，单纯的扩展存储集群的规模可能会造成大量的资源浪费。而 I/O 工作负载组成的时序数据是一种重要的有助于我们更好理解系统的数据，通过对历史 I/O 负载数据进行分析，提取时序变化模式以预测出未来某时刻的负载，有助于进行细粒度的负载均衡以及调度等任务。长短期记忆网络是一种专门用于序列建模的循环神经网络，由于其在捕捉长期依赖关系方面的卓越能力受到广泛关注。本文基于深度学习模型中的长短期记忆网构建一种存储系统 I/O 负载预测模型。

1.4 论文组织结构

本论文共分为五章，具体内容安排如下：

第一章为绪论。该章节主要介绍本论文的课题来源，研究背景与意义，概述了课题

的研究目标和主要研究内容。

第二章为国内外相关研究及发展趋势。首先介绍了研究较多的集群 trace 数据，然后对负载转折点预测问题、任务失败预测问题、负载预测问题及其相关技术进行介绍。

第三章为基于特征增强的多任务 LSTM 负载转折点预测算法的研究工作。首先提出了基于特征增强的多任务 LSTM 负载转折点预测算法，然后分别介绍该算法的两个组成部分：工作负载预处理以及基于特征增强的多任务 LSTM 预测模块，其中在预处理模块介绍了提出的 4 种波动性特征。最后介绍实验过程及结果。

第四章为基于通道注意力卷积模型的任务失败预测算法的研究工作。首先提出了基于通道注意力卷积模型的任务失败预测算法，然后介绍其三个组成部分：通道内一维卷积模块、通道间多头自注意力模块和全连接分类模块。最后介绍实验过程及结果

第五章为基于长短期记忆网络的存储系统 I/O 负载短期时序预测算法的研究工作。首先分析了 I/O 负载在时间维度的相关性。然后按照数据预处理、时间序列预测和数据后处理三个阶段介绍了本文提出的基于长短期记忆网络的存储系统 I/O 负载预测模型。

最后总结了本文的研究内容及成果，并对未来研究进行了展望。

第二章　国内外相关研究及技术

2.1　集群 trace 数据

集群 trace 数据是对系统中的某些过程或状态进行跟踪、记录而得到的数据，描述了系统在特定时刻发生的事件或者所处状态。计算机集群产生的 trace 往往包含丰富的系统性能信息，是对系统进行分析优化的关键。trace 数据按照采集的场景可以分为：存储系统 I/O trace，云服务作业 trace，高性能计算 trace 等。其中云服务作业场景下的 trace 研究较多，谷歌在 2012 年发布的 trace 数据是一个典型的代表。它主要由 Job events table、Task events table、Task constraints table、machine event table、Machine attributes table、Resource usage 6 张表组成。对于提交到集群中的作业，首先会会在 Job events table 中记录一个作业提交事件，包含作业的提交时间、提交用户、作业的调度类别。然后作业被拆分为一个或者多个任务，当任务被调度时会在 Task events table 中记录对应的调度动作，Resource usage table 中则记录了每个任务在间隔为 5 分钟的测量周期内的各种资源消耗数据，包括 CPU、内存使用率等。这 6 张表记录了多达 2500 万个任务、1 万多台服务器的时序信息，为采用深度学习技术进行负载数据的分析预测提供了强有力的数据支持。

2.2　负载转折点预测问题相关研究及技术

2.2.1　相关研究

目前的云环境下负载研究主要集中在点值预测，转折点预测的研究相对较少。2018 年 Xia 等人[10] 针对云容量规划（Cloud capacity planning）问题进行研究，作者提出用户对虚拟机请求总量的剧烈波动，往往会造成云服务供应商和和用户之间的不匹配现象。如果供应商事先准备了很多虚拟机，当用户的需求总量突然减小时，就会造成巨大的浪费；而如果供应商没有提供余量，当用户需求量突然变高时，由于虚拟机启动等会消耗一定时间，无法立即为用户提供服务。作者认为对虚拟机请求量进行点值预测无法应对这种突变点，所以提出了 PLR-WSVM 模型，专门对突变点也即转折点进行预测。此算法首先采用自顶向下的分段线性分割算法 PLR[11] 对虚拟机请求量的时间序列进行转折点标记，然后通过 WSVM[12] 模型来预测下一刻的负载点是否是突变点，WSVM 模型即

Weighted SVM，即针对不同的标签对每个样本设置不同的权重从而使 SVM 更加关注某些标签对应的样本，在权重设置方式上可以设置为每个标签对应的样本比例的倒数，这样较少样本的类别就能在损失函数中保证和较多样本的类别有相同贡献。作者首先设置了一个固定大小的观测窗口，包含了历史的请求量序列，然后使用两类特征来描述虚拟机请求数量的变化：第一类是虚拟机请求的元信息（meta information），包括当观测窗口中正在请求虚拟机服务的用户的分布、观测窗口中所请求的虚拟机的类型分布；第二类特征是观测窗口中的统计特征，包括当前的请求数量，请求量的均值、请求量的方差。这些特征被用来预测下一个点是否是转折点。然而 Xia 等人的研究存在几处不足：首先转折点本质上是波动规律的变化，而作者所使用的特征大都是时间窗口内基础的统计特征，如均值，方差等，这些基础特征在表现负载波动性方面能力较弱；其次，基于单个窗口中统计特征的建模方法可能无法捕捉请求序列在时间维度的变化规律，下一时刻的取值可能受到过去取值的影响；最后，作者采用的数据集规模太小，5 个月长度的请求数据按照天的单位进行聚合之后只得到大约 150 个点值。

2.2.2 相关技术

在文献中[12] 转折点是局部趋势的变化点，常常需要通过时间序列的线性分段算法[11]（PLR）进行标记，PLR 分割算法一般根据边界切分略分为三类：滑动窗口、自顶向下和自底向上。a）滑动窗口分割算法：它的基本思路是首先设置一个阈值，然后开始不断地将下一个点纳入当前窗口内，每次加入一个点值之后就对窗口内数据拟合一条直线并计算拟合误差。如果当前误差大于设定的阈值，就将窗口内的数据视为一个分割出来的片段，即将这些被分割出来的片段从源序列去掉。同时将窗口长度重新置为 0，开始下一次计算。如果小于设定的阈值就继续将下一个点加入当前窗口。b）自顶向下分割算法：自顶向下的算法考虑了时间序列的每一个可能的分割位置。首先也需要设定一个阈值，然后遍历整条时间序列中除两端点外的其他位置，对于每一个位置，分别以该位置和两端点作为新片段的端点，拟合一条直线并分别计算拟合误差。若某个误差超过设定的阈值，就需要对该新片段继续使用自顶向下算法进行分割，直到所有划分出来的片段的拟合误差都小于给定阈值。c）自底向上分割算法：自底向上分的算法同样需要设定一个阈值，它每次计算合并相邻的两个片段之后新片段的拟合误差，遍历所有相邻片段之后找出误差最小的两个相邻片段，然后将其合并。直到该最小误差大于设定的

阈值为止。算法开始时将相邻两点确定的线段视为一个片段，然后使用此思路逐步扩大片段长度。这三种算法中分割阈值的选取均会影响分割出来的片段的个数，分割阈值设置过大会增加每个片段所包含原始点的个数，从而使分出来的片段数量减小，同理小的阈值会产生较多的片段。2018 年 Xia 等人[10] 发现由于自顶向下的方法是从整个序列出发进行分割位置选择，所以能较好的描述转折点。

2.3　任务失败预测问题相关研究及技术

2.3.1　相关研究

由于计算机群庞大的规模以及可能存在的异构特性，运行于其上的任务可能会执行失败，然而失败的任务消耗了大量的软硬件资源，造成资源浪费[13–16]，所以在任务执行失败之前预测出这些任务以避免资源浪费就成为一个重要的问题。由于可能导致任务失败的原因有很多，比如硬件故障、out-of-memory 等，研究者们往往希望从集群监控角度出发，通过任务运行过程中的资源使用情况的变化，即“数据驱动”的方式，来预测一个任务是否会执行失败，这样不必依赖领域知识，去区分复杂的具体失败原因[13–15]。目前集群任务失败预测方法主要采用时间序列分类技术，通过任务运行过程中采集的监测信息，构建时序模型从而提取时序特征以进行预测。主要代表工作是 Chen 等人[13] 以及 Islam[14] 等人基于时间序列分类算法的研究。

2009 年 Fadishei[17] 等人研究了网格环境下 CPU 利用率、内存使用率等指标和任务是否会失败这两者之间的相关性。Pan 等人[18] 使用黑盒错误诊断的方法来诊断 MapReduce 系统的错误，他们发现失败节点和正常节点之间存在着不同的行为模式。

2014 年 Chen 等人[13] 在谷歌集群数据集[3] 上进行研究，使用每个任务执行时消耗的资源情况来描述任务的特征，具体来说包括：平均 CPU 使用率、平均内存使用率、页缓存使用率、平均硬盘 I/O 时间，平均硬盘使用率这 5 个资源指标所组成的 5 维时间序列描述一个任务。他们认为任务执行失败与否可以由任务前期运行时产生的资源使用波动反映出来，不同的变化模式预示着不同的终止状态，即正常结束或运行失败。为了捕捉这 5 个指标随时间的变化特征，他们使用深度学习模型中的循环神经网络来进行特征提取，以预测一个任务最后是否会失败。

2016 年刘春红[19] 等人主要研究云集群下作业的失败预测问题。他们将谷歌集群数

据中作业提交时的属性看做静态特征，包括作业划分的任务数量、作业调度类别、平均优先级、请求的平均 RAM 资源量、请求的平均 CPU 资源量、请求的平均磁盘资源量；同时将作业中各个任务的监测指标作为动态特征，包括任务重复提交次数、任务使用的 CPU 的最大值、均值、标准差等统计特征，并采用 SVM 作为分类器以预测失败作业。然而由于谷歌 trace 中作业的静态特征有较多的缺失，作者对一周的数据进行过滤仅得到 8498 个作业，试验规模较小。此外作者以作业作为研究对象粒度较粗，不利于对资源的节约与管理。

2017 年 Islam[14] 等人延续了 Chen 等人[13] 的工作，他们扩展了任务的特征，加入了一些静态信息包括：任务所属用户的用户名、任务所属作业的 ID，任务在作业中的索引编号、任务执行时间、重复提交次数、优先级、调度类别。这些静态信息和每个时间点监测到的资源消耗信息拼接起来作为该时刻任务的特征向量，并采用深度学习模型中更为先进的长短期记忆网络来提取时间序列蕴含的特征，最后通过全局平均池化层，从长短期记忆网络所有时间步输出的隐层表示中提取执行失败任务的全局时序特征，以进行预测。然而 Islam 等人扩展的的任务静态特征有些是不合理的：

1）对于用户名，虽然有些用户提交的任务确实更有可能执行失败，但是作为任务调度系统，如果使用该特征得到的预测结果对任务进行提前终止，显然是对某些用户的偏见，他们的任务可能有更高的误杀率。此外，集群系统中可能存在着大量的新用户，这些用户没有历史提交记录，使用旧用户的用户 ID 作为特征训练出来的模型显然不具有可信力。

2）对于任务所属作业的 ID、任务在作业中的索引编号这两个特征，前者会遇到和新用户一样的问题，对于后者，不同的作业可能被划分出数量不同的任务，而任务被划分时的顺序可能是随机的，这个索引号并不包含任何任务本身的信息。

3）对于任务执行时间，通常真实情况下只有当任务运行结束时才能确切的得到其运行总时间，这样的话就无法提前终止失败任务，预测将变得毫无意义。而如果通过预测得到估计出的运行时间又会引入误差，并造成额外的开销。Islam 的工作是在理想情况下进行的，直接根据 trace 数据中任务监测时序数据计算出任务执行时长，忽略了实际的使用情景。

4）对于重复提交次数，虽然重复提交次数更多的任务更有可能再次执行失败，但是初次提交时显然无法获得该特征，限制了模型的使用范围。同时该特征也属于带有偏

见的特征，不能仅仅因为上次执行失败被再次提交了，所以就预测该任务本次也可能失败。

整体上 Chen、Islam、的模型都主要关注于任务特征的选择、模型的优化，忽视了预测时机的选择，他们往往假设完全获得了任务执行信息，然后进行预测。虽然 Chen 也考虑了在任务总执行时间的四分之一时间，二分之一时间进行预测，但却是在获得了该任务总执行时长之后再确定的预测时机。这些原因都导致了理论研究和实际使用场景的不匹配。

2.3.2 相关技术

由于任务失败预测使用了多变量时间序列分类模型，所以此处对基于深度学习的多变量时间序列分类模型进行了总结。多变量时间序列是同时记录多个变量随时间变化的序列数据，是由多个时间序列组合在一起形成的一个整体，每一个变量都对应着其中的一条时间序列。在一段有限时间内按照时间先后顺序对所有变量进行采样就能得到一个数据流，该数据流即为一个多维时间序列。从形式上看它是一种矩阵类型数据。多变量时间序列的分类问题可以表述为：利用类别已知的多维时间序列训练样本建立分类模型，基于该模型对类别未知的样本进行类别预测。

2014 年 Ge 等人[20] 提出了 MCDCNN 模型专门用于多变量时间序列分类问题，考虑到每个变量对应的的时间序列内部可能含有不同的时序特征，它采用独立的卷积操作处理每一条时间序列。每一维时间序列都会通过两层的卷积模块，每个模块包含一个卷积层，以及一个最大池化层。最后，第二个模块的输出被拼接在一起送入一个全连接网络进行分类。虽然 MCDCNN 模型考虑到了不同变量内部的特征的独立性，但由于最后的拼接结构，无法捕捉不同变量间的交互作用。

2017 年 Wang[8] 等人提出了 Fully Convolutional Network(FCN)，该模型主要是由卷积网络组成，并且整个模型中没有使用局部的池化层。它由 3 个基本卷积模块堆叠组成，每个模块包含一个卷积层、一个 batch normalization 层、一个 ReLu 激活层。第三个基本模块的输出连接到一个全局平均池化层，然后是一个 softmax 分类层。该模型由于没有局部池化，所以时间序列的长度在三个模块中都保持等长，由最后的全局平均值化层负责提取不变性特征。

2017 年 Wang[8] 等人提出了 residual network (ResNet)，它是一种基于 ResNet 结构的

深度卷积网络，根据 Fawaz 等人在 2019 年的研究[21] 此模型是目前最优的多变量时间序列分类模型。基本结构借鉴了 ResNet 中的残差单元，在每个模块间加入 short cut，以保证深度网络中的梯度流通。每个模块包含 3 个等长卷积操作，卷积核大小依次为 8,5,3，每次卷积之后加入 Batch Nornalization 层以防止梯度消失，加速模型收敛，随后再通过一个 ReLu 激活层做非线性变换。整个模型的卷积操作和 FCNN 模型相同，都是使用一维卷积在时间维度上滑动，依次进行卷积操作。

2.4 负载预测问题相关研究及技术

2.4.1 相关研究

目前关于 trace 的研究大部分关注计算 trace，I/O trace 方面则较少，尤其是基于开源 I/O trace 的时序预测模型的研究则更加少见。Dong[22] 等人曾研究了并行文件系统中的负载均衡问题。为了解决因通信开销导致的负载信息过期问题，他们采用 ARIMA 时间序列预测算法，对数据服务器的吞吐量进行预测。近年来，虽然深度学习模型已经在一些云计算环境下工作负载预测问题上取得了进展，但是很少有研究将其应用于存储系统的 I/O 负载预测上。存储系统 I/O 负载是否在时间维度蕴含一定的长期依赖关系、是否适合使用序列模型来描述，还需要进一步研究。由于在本文第一个研究点：转折点预测问题上也会涉及到云环境下负载预测相关模型，所以有必要介绍一下云环境下负载预测的相关问题。

Gupta S[23] 等人在云平台 CPU 资源使用率的预测问题上成功使用了长短期记忆单元，他们将 CPU 使用率的预测问题建模为多变量时序预测问题，同时考虑：运行的任务个数、内存使用率、页缓存、内存使用率峰值、磁盘 I/O 时间、磁盘使用率等因素。并且他们使用一个双向的 LSTM 网络来捕捉过去对现在以及现在对过去这两方面的影响。实验结果表明，与单向模型相比，双向模型中未来-过去依赖关系的集成在拟合能力和预测质量上具有较大的优势。

Di[24] 等人首先利用朴素贝叶斯模型预测了云环境下主机的 CPU 和内存的平均使用率。他们使用观测窗口中的历史负载数据提取九个特征来对预测窗口中的均值负载进行预测。作者采用指数分段算法 (ESP) 对预测窗口按不同时间间隔进行划分，使短期预测对应的预测区间更小，长期预测对应的预测区间更大。然而其预测的对象是按照负载

值的大小，进行较粗粒度的离散化之后的负载等级，并不是真正意义上的负载值预测，并且由于云环境下主机负载具有很强的波动性，其仅能对一定范围内的负载均值进行预测。

Song 等人[25] 认为，云环境中负载的高方差是导致传统基于统计和基于机器的学习方法性能较差的主要原因。作者认为 LSTM 可以学习长期依赖关系，还可以决定何时记住和何时忘记历史数据。文中采用多对多的循环神经网络结构。在每个时间步，将历史窗口中的负载值输入模型，得到预测窗口中的多步预测值。然而这个结构只利用了多个历史窗口之间的依赖关系，没有在窗口内捕捉时间维度的依赖关系，因此当进行一次预测时，整个模型结构相当于在每个时间步中应用一个简单的前馈神经网络，相比于 DNN 并没有明显优势。

Zhang 等人[26] 研究主要关注不同服务器监控指标对负载预测的影响。他们认为在监测指标种类很多时，很难确定某一指标是否有利于工作负载的预测，所以他们提出一个基于深度学习的负载预测模型，主要包括两个部分：监测指标选择模块和循环神经网络预测模块。在指标选择模块使用 k-shape 时间序列聚类算法选择有用的指标，并将其送到双向的长短期记忆网络所组成的预测模块，进行负载预测。本文的第一个研究工作和作者的主要区别在于本文只使用 CPU 工作负载来预测 CPU 工作负载的转折点，所有的特征都是从原始 CPU 工作负载序列中提取出来的，而作者使用不同的指标来进行基于值的预测。

2.4.2　相关技术

目前基于点值的负载预测算法大致可分为三种，基于统计的方法，基于传统机器学习的方法，基于深度学习的方法。基于统计的方法主要以 ARIMA 为代表，主要使用回归模型描述一个平稳的时间序列；基于机器学习的方法将时间序列视为一个各个维度相互独立的向量，从而转化为一般回归问题；基于深度学习的方法，大多采用循环神经网络等序列模型，对序列元素中的依赖关系进行建模。

经典时间序列模型（例如 AR、MA、ARMA 和 ARIMA 等[27–30]）及其相应的分析方法是一种基于统计学的时间序列预测模型，表达式由公式2.1给出：

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \tag{2.1}$$

其中 α_i 是模型的自回归部分，表示过去每个时刻对现在的贡献率，θ_i 是模型的滑动平均系数，$\mathcal{E}_i$ 为残差系数。相应的自回归部分阶数和滑动平均部分的阶数需由自相关系数 ACF，偏自相关系数 PACF 来确定其大致范围，最终通过 information criterion 即 ACI 准则来判断具体取值，ACI 准则是由日本统计学家 Akaike 于 1973 年提出的，他的全称是最小信息量准则，该准则的指导思想是认为一个模型的好坏可以从两个方面去考察：一方面是衡量拟合程度的似然函数值，另一方面是模型中未知参数个数，通常似然函数越大说明拟合效果越好；模型中未知参数越多说明模型变化越灵活。如果一味追求拟合精度，那么必然导致拟合参数变多，从而导致未知的风险越多，模型越容易收到噪声干扰。所以一个好的模型应该是一个拟合精度和未知参数的综合最优配置。在常见的时间序列模型中，ARIMA 模型能够很好的拟合不稳定的序列。在 ARIMA 模型的建立过程中，如果判断出一个时间序列不是一个平稳随的机过程，就需要先进行多次差分操作，直到其通过平稳性检验为止。而对于平稳序列 AIRMA 原本就有较好的拟合能力[22]。

相比 ARIMA 模型使用线性结构拟合历史数据的方式，基于机器学习的算法能提供更加优秀的非线性拟合能力。最近，随着统计机器学习的发展，时间序列预测已经被定义为一个回归问题。通常回归问题需要事先确定输入特征，对应到时间序列预测问题中，其输入特征即为观测窗口中的历史数据，而观测窗口的长度 n 通常需要交叉验过程确定。支持向量回归 (SVR)[31] 是解决这一问题的常见方式。支持向量回归拟合首先考虑用线性回归函数 $f(x) = \omega \cdot x + b$ 拟合 $(x_i, y_i), i = 1, 2, \ldots, n$，$x_i \in R^n$ 为输入量，$y_i \in R$ 为输出量，即需要确定 ω 和 b。经典 SVR 采用 ε-不灵敏度函数衡量拟和误差，即假设所有训练数据在精度 ε 下用线性函数拟合，拟合误差表示为：

$$\begin{cases} y_i - f(x_i) \leq \varepsilon + \xi_i \\ f(x_i) - y_i \leq \varepsilon + \xi_i^* \quad i = 1, 2, \ldots, n \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{2.2}$$

从而将线性拟合问题转化为最小化误差的凸二次最优化问题，可以使用格朗日乘子法重新表示为：

$$\begin{aligned} L = \frac{1}{2}\omega \cdot \omega + C \sum_{i=1}^{n} \left(\xi_i + \xi_i^*\right) - \sum_{i=1}^{n} \alpha_i \left[\xi + \varepsilon - y_i + f(x_i)\right] \\ - \sum_{i=1}^{n} \alpha_i^* \left[\zeta_i^* + \varepsilon - y_i + f(x_i)\right] - \sum_{i=1}^{n} \left(\xi_i \gamma_i + \xi^* \gamma_i^*\right) \end{aligned} \tag{2.3}$$

求解可得：

$$f(x) = \omega \cdot x + b = \sum_{i=1}^{n} \left(\alpha_i - \alpha_i^*\right) x_i \cdot x + b \tag{2.4}$$

此外，SVR 还可以通过非线性变换将输入空间映射到特征空间来捕捉输入和输出之间非线性的关系。并且利用核方法 (如 RBF)，SVR 可以很容易地处理维度灾难问题[32]，更好地利用高维特征空间。然而 SVR 将输入时序数据当做互相独立的特征，完全忽略时序数据本身的顺序关系，导致预测精度往往不高。

近年来，深度学习方法在金融市场预测[33]、天气预报[34] 等多个时间预测场景中都显示出了其优越性。循环神经网络 (RNN)[35] 是一种专门用于序列建模的深度神经网络，由于其在捕捉非线性关系方面的灵活性而受到广泛关注。然而，传统 RNN 面临着梯度消失的问题，因此很难捕获序列的长期依赖。最近，克服了这一限制的长短时记忆单元 (long short-term memory units, LSTM)[36] 在各种应用中取得了很大的成功。其组件结如图2所示：

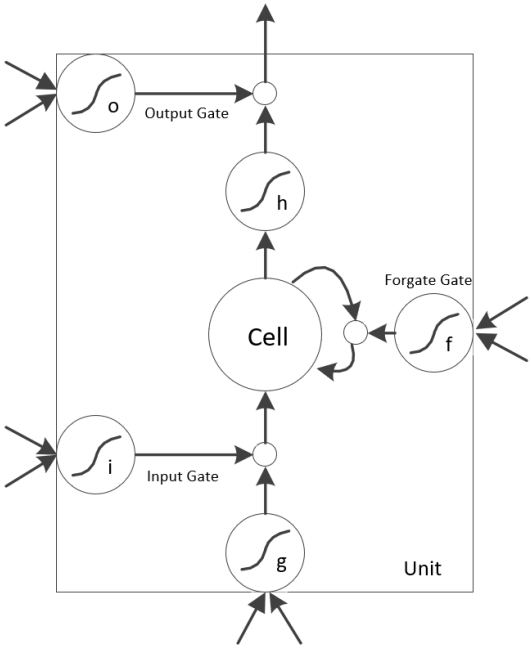


图 2 LSTM 结构

基础的 LSTM 单元包括一个记忆单元 $c_t \in R^H$ 以及三个控制门：输入门 Input Gate $i_t \in R^H$, 遗忘门 Forget Gate $f_t \in R^H$, 输出门 Output Gate $o_t \in R^H$, 其中 H 代表 LSTM 单元的隐层神经元个数, t 代表要处理的时间步长。当输入、记忆单元的初始状态，初始输出状态给定时，即给定 $\mathrm{x}_t, c_{t-1}, h_{t-1}$ 时，在后续每个时间步，LSTM 单元的状态及三个门

的状态由公式2.5给出：

$$
\begin{aligned}
z_t &= \tanh\left(W^z x_t + U^z h_{t-1}\right) \\
i_t &= \sigma\left(W^i x_t + U^i h_{t-1}\right) \\
f_t &= \sigma\left(W^f x_t + U^f h_{t-1}\right) \\
o_t &= \sigma\left(W^o x_t + U^o h_{t-1}\right) \\
c_t &= i_t \odot z_t + f_t \odot c_{t-1} \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}
\tag{2.5}
$$

其中输入门负责控制输入数据的流入比例，遗忘门负责控制历史信息的遗忘比例，输出门负责控制数据的输出比例。通过这三个门能够控制何时遗忘内容，何时保存完整信息。本文第一个研究点将利用 LSTM 的这些优秀特性提取转折点附近的时序变化模式。本文第二个研究点将针对 LSTM 在处理多变量时间序列时存在的问题提出更适合的模型结构。

2.5 本章小结

本章从负载转折点预测问题、任务失败预测问题、负载预测问题三个方面论述了各个问题的的国内外相关研究，并给介绍了涉及到的相关算法。

第三章　基于特征增强的多任务 LSTM 负载转折点预测算法

3.1　研究背景

由于云计算的高可伸缩性、细粒度资源分配和灵活性，能够满足不同计算和存储需求的特点，在过去的十年中，其商业使用需求以及科研活跃度激增。在保持较低水平的服务等级协议 (SLA) 冲突的同时，减少服务器集群中存在的资源浪费一直是云提供商[37] 非常关心的问题。因此，更好的资源管理策略，如虚拟机迁移[38]、服务器整合[39]、动态资源配置[40] 被研究者们密切关注。随着人工智能的发展，基于机器学习和基于深度学习的服务器负载预测模型与资源管理策略紧密的结合在了一起。通过负载预测算法获得的关于未来工作负载的信息越多，资源管理就越高效。

在时间序列预测问题中，可以分为点值预测和趋势预测。点值预测问题即预测未来的某个或某些点的取值的问题，而趋势预测则有着更宽泛的定义，大致分为三类：1）预测下一个点相对于当前点是上升还是下降。如 2019 年 Deng 等人在股票预测中结合知识图谱的研究[9]。2）预测下一个趋势片段的斜率和持续时间。如 2017 年 Lin[7] 等人研究，他们采用2.2.2小节介绍过的时间序列线性分段算法划分趋势片段，然后采用长短期记忆网络和卷积神经网络的混合结构构建预测模型，预测下一个趋势片段的斜率和持续时间。3）预测下一个点是否是转折点。如 2018 年 Xia 等人[10] 的研究，他们首先采用时间序列线性分段算法在原始时序中标记出转折点，然后构建机器学习模型预测下一个点是否是转折点。

目前对云环境下服务器负载预测的研究主要集中在点值预测：2012 年 Di 等人[24] 提出了基于观测窗口内统计特征的贝叶斯模型对谷歌云服务集群中服务器 CPU 使用进行预测；2018 年 Zhang 等人[26] 使用时间序列聚类算法结合双向长短期记忆网络对未来服务器的 CPU 使用率进行预测；2019 年 Duggan 等人[38] 为指导虚拟机迁移，提出使用循环神经网络对云环境下的服务器 CPU 和网络带宽占用率进行预测。尽管机器学习和深度学习方法在基于点值预测的场景下取得了极大的发展，但是由于云环境下庞大的任务提交数量、繁多的任务类型，导致短时间内的服务器负载精确预测依然是一个严峻的挑战[41]。

对于点值预测，即使可以预测出未来的负载值，也很难知道服务器的内部状态和

负载的动态变化趋势。“趋势”是由一段连续的负载点值按时间顺序排列而成，它反映了一段时间的服务器负载状态，趋势的变化反映了状态的转变，相比点值的预测，趋势变化的预测有两点优势：1）不易受单点抖动噪声的影响，云环境中服务器的负载可能在下一时刻发生强烈的抖动，如图3所示，如果在 t 时刻预测 $t+1$ 时刻的负载值，由于 $t+1$ 时刻出现抖动，偏离了 $t+2$ 时刻所在的真实趋势，根据此时预测出的点值进行资源分配可能会出现错误。2）蕴含更加丰富的长期状态信息。趋势的转折代表新旧趋势的交替，能够反映未来一段时间的负载状态。因此可以采用更恰当的资源管理方案分别进行处理。

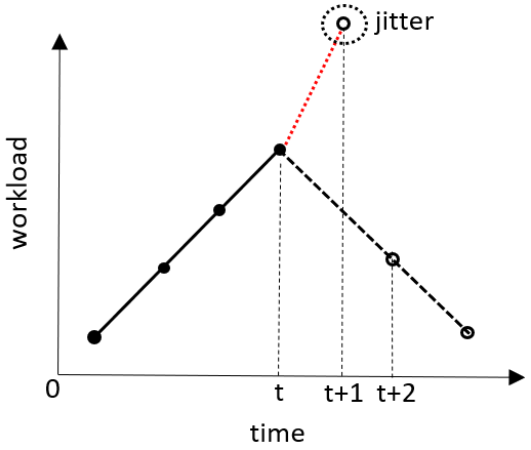


图 3 服务器负载的抖动噪声示意图

综上，本文研究云环境下服务器负载转折点预测问题。工作负载转折点 (workload turning point) 是代表系统压力的峰值点或者是代表系统空闲的谷值点，该点两侧呈现出相反的局部趋势；非转折点处于距离其最近的上一个转折点所转向的趋势中。图4展示了谷歌云服务集群中标号为 908054 的服务器约 20 小时的 CPU 负载变化，其中转折点被标记为三角形。图5是图4中的两类典型情况，左侧图中三角形标示出了一个典型的谷值点，该点两侧的虚线分别代表了局部趋势。可以看到左侧呈现出下降的趋势，右侧呈现出上升的趋势。右侧图中的三角形标示出了一个典型的峰值点，其左侧呈现出上升趋势，右侧呈现出下降趋势。图5展示的这两种点都是本文所指的转折点，由于局部趋势的逆转特性，可以很容易地通过其左侧的局部趋势的符号来区分这两种转折点，所以本文的研究目标为预测当前的负载是否为转折点。

和本文最相似的工作是 Xia 等人在 2018 年的研究[10]，他们使用在股票预测领表现最优的转折点预测模型 WSVM[12] 对虚拟机请求数量的突变进行预测。相比虚拟机请求数量，服务器负载的变化具有更强的非线性和波动性，为转折点的预测提出的更高的挑

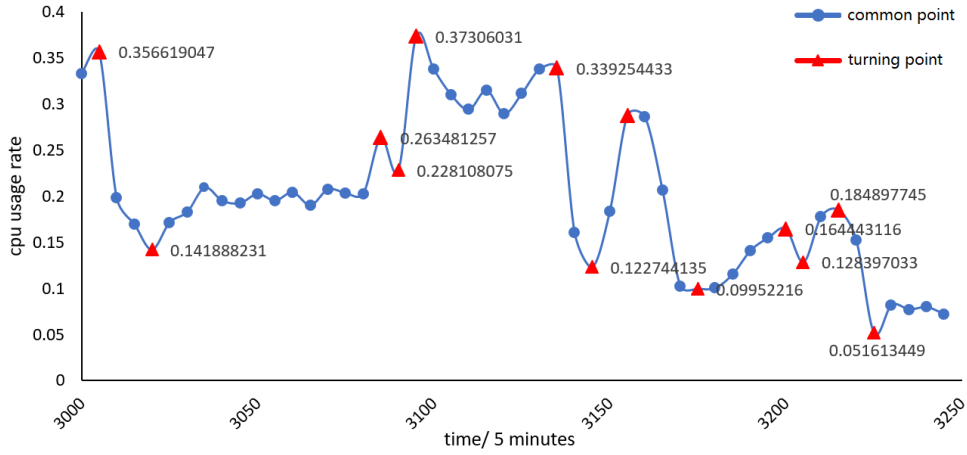


图 4　谷歌云服务集群 trace 服务器工作负载转折点图

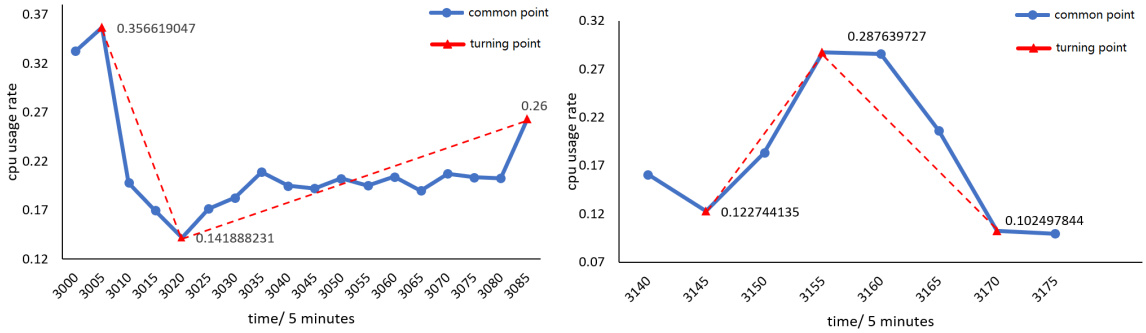


图 5　转折点标记效果图

战。此外，Xia 等人的研究存在两处不足：1）转折点意味着趋势的反转，本质上是波动规律的变化，而 Xia 等人所使用的特征大都是时间窗口内基础的统计特征，如均值，方差等，这些基础特征在表现负载波动性方面能力较弱；2）现有最优的转折点预测模型 WSVM 只基于预测点附近的单个时间窗口内的统计特征构建构型，忽略了系统状态在时间维度上可能存在的长期变化规律。本章针对这两个问题研究云环境下服务器负载转折点预测算法。

本方法的贡献如下：

1）提出了四种时序特征用于描述服务器负载转折点的波动特性，并基于滑动窗口提取预测点附近的序列形式特征，并通过对比实验验证了其相比单一窗口中提取的基础特征的优越性。

2）提出了一个特征增强的多任务 LSTM 模型，将特征序列和原始负载序列分别视为两种不同的任务进行学习，从特征序列和原始负载序列中提取波动模式；然后采用显式的融合结构，学习特征序列和原始负载序列之间的隐含关系，达到特征增强的效果。

3）在开源的的谷歌云服务集群 trace[3] 上进行的实验表明我们的模型在 F1 指标上优于五种基线模型，在 F1 指标上最少平均提升 2 个百分点，并通过对照实验验证了此

模型的特征增强机制以及多任务结构的有效性。

3.2 负载转折点预测问题建模

工作负载转折点的形式化定义如下：给定工作负载时间序列 $\mathbf{x} = (x_1, ...x_2, ...x_T)$，$T$ 是工作负载时间序列的总长度，工作负载转折点是工作负载局部趋势逆转的点。

在文献[12] 中，局部趋势往往是采用分段线性分割 (或称表示) 算法[11](PLR) 来描述的。时间序列线性分割算法的目标是将给定的时间序列 $\mathbf{x}$ 分割为一系列片段 $\mathbf{x_{seg}} = \{s_1\{x_1, x_2..x_{i_1}\}...s_j\{x_{i_{j-1}}, x_{i_{j-1}+1}...x_{i_j}\}...s_N\{x_{i_{N-1}}, x_{i_{N-1}+1}...x_{i_N}\}\}$，其中 $1 < i_1 < i_j < i_N = T$。$s_j\{x_{i_{j-1}}, x_{i_{j-1}+1}...x_{i_j}\}$ 是工作负载时间序列的第 j 段，$x_{i_{j-1}}$ 和 x_{i_j} 是第 j 段的边界，而 x 的下标 i_j 代表原始负载序列中的位置索引。其核心思想是在每一段中拟合一条直线，使总拟合误差最小。PLR 分割算法一般根据边界切分策略分为三类：滑动窗口、自顶向下和自底向上，详细情况已在2.2.2节讨论过。在这里，我们遵循 Xia 等人[10] 的思想使用自顶向下的方式分割工作负载时间序列，这种方法已被证明是较好的描述局部趋势的一种方法。Xia 等人只根据前一个片段和当前片段的端点值的大小进行转折点的标记。而由于 PLR 算法仅仅是基于拟合误差最小原则进行边界切分，并不能保证边界点两侧斜率相反，也即趋势相反。所以本文认为，分割出来的段的边界只是转折点的候选集合，需根据双边的端点值大小行过滤操作，保证该点两侧趋势相反。假设现在分割出了三个连续的线段：$s_{j-1}\left\{x_{i_{j-2}} \ldots x_{i_{j-1}}\right\}, s_j\left\{x_{i_{j-1}} \ldots x_{i_j}\right\}, s_{j+1}\left\{x_{i_j} \ldots x_{i_{j+1}}\right\}$。标签 $y_{i_{j-1}}$ 对应于第 j 段的左边界 $x_{i_{j-1}}$，y_{i_j} 对应于第 j 段的右边界 x_{i_j}，则它们的定义如下:

$$y_{i_{j-1}} = \begin{cases} 1, if\ \left(x_{i_{j-1}} - x_{i_{j-2}} > 0\right) \text{ and } \left(x_{i_{j-1}} - x_{i_j} > 0\right) \\ \text{or } \left(x_{i_{j-1}} - x_{i_{j-2}} < 0\right) \text{ and } \left(x_{i_{j-1}} - x_{i_j} < 0\right) \\ 0, \text{ others} \end{cases} \tag{3.1}$$

$$y_{i_j} = \begin{cases} 1, if\ \left(x_{i_j} - x_{i_{j-1}} > 0\right) \text{ and } \left(x_{i_j} - x_{i_{j+1}} > 0\right) \\ \text{or } \left(x_{i_j} - x_{i_{j-1}} < 0\right) \text{ and } \left(x_{i_j} - x_{i_{j+1}} < 0\right) \\ 0, \text{ others} \end{cases} \tag{3.2}$$

其中 1 表示转折点，0 表示普通点，其他的非分割边界的点也被标记为普通点。通过计算左右相邻两点和该点的大小，确保该点为正确的峰值或谷值点。给定历史观测窗

口中的一段工作负载时间序列 $\mathbf{x_t} = (x_{t-w+1}, x_{t-w+2}, \ldots x_t)$，其中工作负载为定义为某一种资源的使用率，如 cpu 使用率，工作负载转折点预测的目标是通过学习转折点指示函数 $y_t = f(\mathbf{x_t})$ 来判断当前的点 x_t 是否为负载时间序列的转折点，其中 $y_t \in (0,1)$，0 表示 x_t 是一个非转折点，即普通点，1 表示 x_t 是一个转折点。w 是历史观测窗口，表示使用多少历史工作负载数据进行转折点预测。如图6所示，空心圆点代表当前点 x_t，该值是已知的，要预测该点是否是转折点。而由于 t 时刻之后的未来值未知，所以用其左侧虚线代表未来负载值。

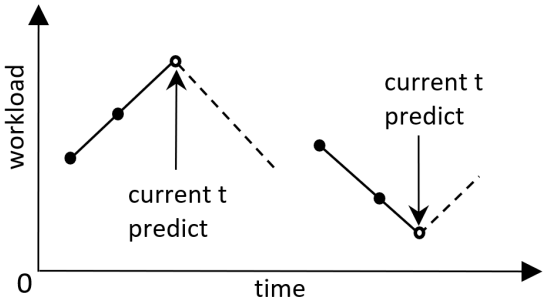


图 6　工作负载转折点预测示意图

3.3　算法框架

本文提出的服务器负载转折点预测算法框架如图7所示，它由两个部分组成：工作负载预处理模块和特征增强的多任务 LSTM 预测模块。模型的输入是云环境下服务器的 trace，可以是 CPU、内存和其他工作负载组成的时间序列。工作负载预处理模块负责从负载时间序列中提取波动性特征，以及为模型的训练生成标记数据，它们都被送入后续预测模块。预测模块采用本文提出的基于特征增强的多任务 LSTM 模型（FEMT-LSTM），预测转折点并输出相应的预测标签。本文将按照模型中的模块依次介绍：1. 工作负载预处理模块。2. 基于特征增强的多任务 LSTM 预测模块。

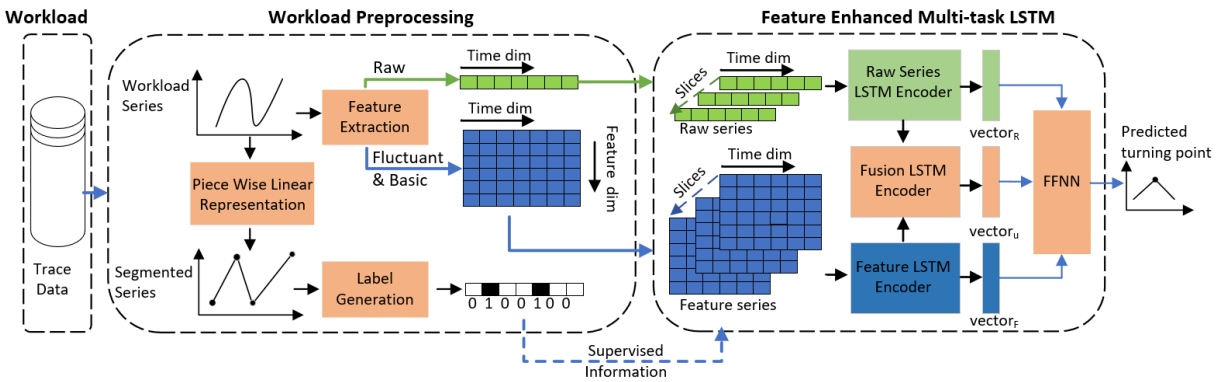


图 7　基于特征增强的 LSTM 负载转折点预测算法框架图

3.4 工作负载预处理

本文是基于监督学习模型对服务器负载转折点进行预测，训练监督学习模型需要监督数据，即对每个样本构建特征以及对应的标签。负载预处理模块需要从原始的负载时间序列中生成转折点标签，同时在观测窗口中提取时序波动特征。本小节首先介绍基于规过滤的转折点标签生成，然后介绍基于波动性的负载时序特征提取。

3.4.1 基于规则过滤的转折点标签生成

由于转折点并非天然的存在于原始时间序列中，而采用人工标注又无法严格的满足转折点定义，所以需要先按照3.2小节所述的定义采用分段线性分割法，自动生成转折点标签。如3.2小节所述，由 PLR 生成的段边界并不一定是转折点，所以需要进行无关点的滤除操作，这一步骤在之前的算法模型中往往被忽略，所以本文提出了一种基于规则过滤的转折点标记算法，如算法1所示。遵循的过滤规则如3.2节，公式3.1和3.2所示。

原始服务器负载时间序列 $\mathbf{x} = (x_1, ...x_2, ...x_T)$ 在经过 PLR 分段之后得到一系列片段 $\mathbf{x_{seg}} = \{s_1\{x_1, x_2..x_{i_1}\}...s_j\{x_{i_{j-1}}, x_{i_{j-1}+1}...x_{i_j}\}...s_N\{x_{i_{N-1}}, x_{i_{N-1}+1}...x_{i_N}\}\}$，每一个片段的左右两个端点就组成的二元组序列 $\mathbf{x_{boundary}} = (< x_1, x_{i_1} > ... < x_{i_{j-1}}, x_{i_j} > ... < x_{i_{N-1}}, x_{i_N} >)$ 这部分数据即为此算法的输入，输出为每个时刻负载 x_i 的标签。由于第一个片段的左端点是片段序列的起始位置，所以算法第一行设置该点的标签为 0。算法第 2 行，开始循环遍历片段二元组序列，第 3 行和第 4 行得到连续的两个片段的端点，其中前一个片段的右端点和后一个片段的左端点是重合的，所以都记为 x_1，第 5 行到第 10 行得到连续的三个端点之后根据中间点的值 x_1 和两侧点的值 x_0，x_2 判断 x_1 是否是三个点中的最小值或者最大值，如果是，则代表该点 x_1 即为转折点，相反则为非转折点。最后得到每个片段二元组的标记，而根据3.2节所述，其他的非片段端点也被标记为 0，从而得到了整条负载序列对应的标签序列。

3.4.2 基于波动性的服务器负载时序特征提取

云环境中的工作负载充满了波动性，高质量的特征将有助于模型捕获时间序列的局部趋势特征。如[10, 24] 中所讨论的，云环境下工作负载的时序特征通常是从具有固定大小的时间窗口中提取的，本文称此窗口为特征观测窗口，对于 t 时刻的负载 x_t，假设其对应的特征观测窗口大小为 n，则该窗口中的数据可以表示为 $(x_{t-n+1}, x_{t-n+2}...x_t)$，特征提取的过程就是从该数据中提取构造的特征。这些特征反映了特征窗口内负载的变化

算法 1: 负载转折点标签生成算法

Input : The segmented workload boundary series
$\mathbf{x_{boundary}} = (< x_1, x_{i_1} > \ldots < x_{i_{j-1}}, x_{i_j} > \ldots < x_{i_{N-1}}, x_{i_N} >)$;
Output: labels of workload series points $\mathbf{y} = (y_1, \ldots y_2, \ldots y_T)$;

```
labels = [0];
for j = 2; j ≤ len(x_seg); j++ do
    x_1, x_2 = x_seg[j];
    x_0, x_1 = x_seg[j − 1];
    if x_1 < x_0 and x_1 < x_2 then
        labels.append(1);
    else if x_1 > x_0 and x_1 < x_2 then
        labels.append(1);
    else
        labels.append(0);
    end
end
y = labelss
```

情况，不同的特征反映了不同的趋势变化，然后可以使用高效的学习模型来进行预测。因此，本文沿着这个思路同样采用基于窗口的方法来提取工作负载的局部趋势特征。

在 Xia[10] 等人的研究中使用两类特征来描述虚拟机请求数量的变化：第一类是虚拟机请求的元信息（meta information），包括当前时间窗口中正在请求虚拟机服务的用户的分布，当前时间窗口中所请求的虚拟机的类型分布；第二类特征是观测窗口中的统计特征，包括当前的请求数量，请求量的均值、请求量的方差。然而本文所关心的是云环境下服务器资源使用问题，很难对应这样的元信息，并且 CPU 使用率的负载序列反映了服务器内部的系统状态，具有更加强烈的不确定性，因此必须探索更有效的特征辅助负载转折点的预测。本文使用的特征列于表1. 除了三个基础统计特征（当前工作负载，工作负载的平均值，工作负载的方差），本文提出了四个波动性特征，以反映观测窗口临近点的负载趋势情况。

图8展示了当前时刻 $t = 4$，特征观测窗口的长度为 $n = 4$ 的具体例子。则该窗口中的负载值为 x_1, x_2, x_3, x_4，其中假设了 $x_2 - x_1 = x_4 - x_3 = \frac{1}{2}(x_2 - x_3) = d$，则对于 $t = 4$ 时刻，可以按照表1计算出一组特征，作为该时刻的特征向量。

（1）工作负载的趋势强弱：由于服务器负载每时每刻都处于变化状态，在划定的特征观测窗口内常常表现出有升有降的现象，当一段时间内负载的上升总量比下降总量高时，观测窗口内整体上可能呈现上升趋势；而当下降的总量比上升的总量高时，在观

表 1 基于长度为 n 的特征观测窗口的基础特征以及波动特征表

type	name	description	formula
basic	f_{cw}	current workload	x_t
	f_{avg}	average of the workload	$\frac{1}{n}\sum_{i=0}^{n-1} x_{t-i}$
	f_{var}	variance of the workload	$\frac{1}{n}\sum_{i=0}^{n-1}(x_{t-i}-f_{avg})^2$
fluctuant	f_{rsi}	relative strength of the workload	see Eq.3.3. 3.4. 3.5.
	f_{asc}	absolute sum of the workload changes	see Eq. 3.6.
	f_{msdc}	mean second derivative central of the workload	see Eq. 3.7.
	f_{lts}	linear trend's slope of the workload	-

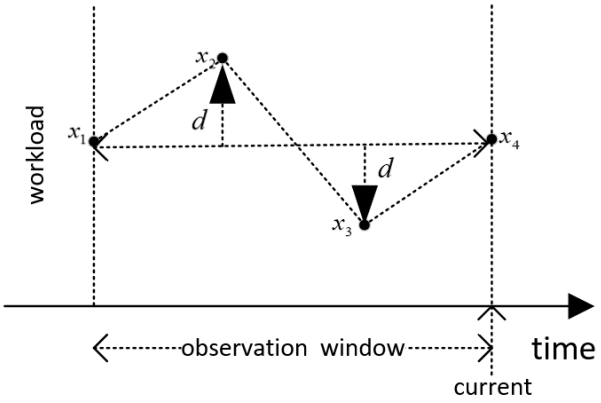


图 8 特征观测窗口图

测窗口内整体上可能呈现下降趋势。上升或下降趋势的变化量占总变化量的比值越大，说明该趋势的强度越大，不同趋势的强度反映了不同的波动特性，这一点和股票分析领域的相对强度指标[42] 类似。其计算方法如公式3.3所示，f_{rsi} 即为趋势强弱指标，等于总的上升趋势 x_{up} 占总变化量的比例，总的变化量为 x_{up} 以及 x_{down} 的和。其中上升的变化量如公式3.4所示，下降的变化量如公式3.5所示，公式中的 $I()$ 为示性函数。f_{rsi} 越接近 1 代表上升趋势的强度越大，越接近 0 代表下降强度越大。而当接近 $\frac{1}{2}$ 时，代表两种趋势的强度相同。例如，在图8中所示的特征观测窗口内，由于 $x_{up} = x_{down} = 2d$ 所以，该时刻 $f_{rsi} = 2d/(2d+2d) = 1/2$，代表在这个窗口中上升下降趋势的强度相同。

$$f_{rsi} = x_{up}/(x_{up} + x_{down}) \tag{3.3}$$

$$x_{up} = \sum_{i=1}^{n-1} |x_{t-i+1} - x_{t-i}| I(x_{t-i+1} - x_{t-i} > 0) \tag{3.4}$$

$$x_{down} = \sum_{i=1}^{n-1} |x_{t-i+1} - x_{t-i}| I(x_{t-i+1} - x_{t-i} < 0) \tag{3.5}$$

（2）工作负载变化量的绝对值总和：由于 f_{rsi} 只能衡量观测窗口内整体的趋势强弱，当上升和下降两种变化量相等时就无法体反映出更多的信息。所以可以计算观测窗口中的负载的变化量的绝对值之和 f_{asc}。如公式3.6，f_{asc} 越大，表明观测窗口中的负载变化越剧烈，相反则越平稳。例如，在图8中所示的特征观测窗口内，$f_{asc} = d + 2d + d = 4d$。

$$f_{asc} = \sum_{i=1}^{n-1} |x_{t-i+1} - x_{t-i}| \tag{3.6}$$

（3）工作负载二阶差分的均值：记为 f_{msdc}，如公式3.7，它衡量工作负载变化的“加速度”，即变化速度的快慢。该值越大代表观测窗口中的负载序列变化的越快，即波动性越强。

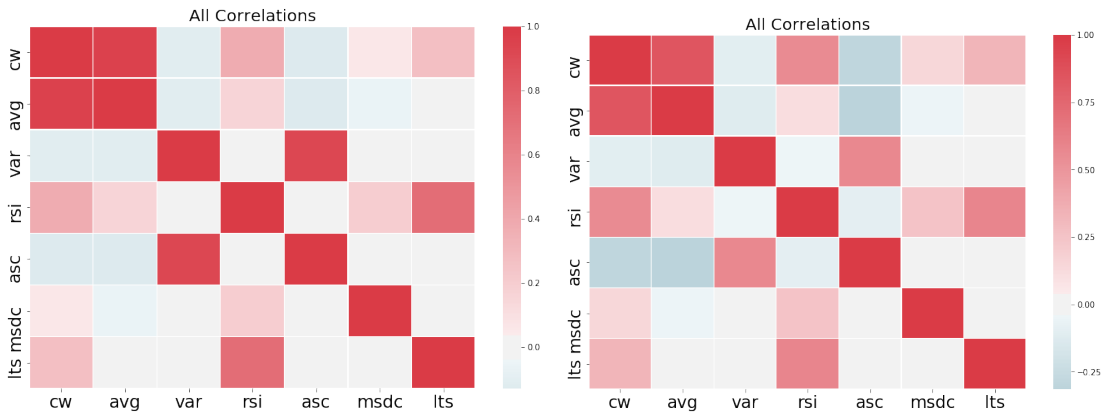
$$f_{msdc} = \frac{1}{n-2} \sum_{i=2}^{n-1} [(x_{t-i+2} - x_{t-i+1}) - (x_{t-i+1} - x_{t-i})]. \tag{3.7}$$

（4）工作负载的线性趋势斜率：本文使用线性函数 $y = ax + b$ 来拟合窗口中的工作负载, 并使用直线的斜率 a 作为特征之一。拟合时采用最小二乘法，目标是拟合误差最小。该值的绝对值越大，说明观测窗口中的负载序列越陡峭，反映出更加强烈的趋势倾向。

针对服务器负载的波动性，完成了服务器负载特征的提取之后，我们还对特征的相关性进行了评估。增加多个特征的目的是为了提升分类精度，然而如果不恰当的选取特征，尤其是选取的特征之间相关性过强，将会产生冗余信息，反而会影响分类器精度。本文对谷歌云服务集群中节点编号 908054 的服务器产生的长达 29 天的 CPU 使用率负载序列进行特征提取，做出 7 个特征间的皮尔逊相关性系数图。图9分别为特征观测窗口长度设置为 3 和 18 个时间步时（每个时间步均为 5 分钟）的相关性图，可以看到代表较强相关性的深红色部分基本位于对角线附近，说明本文所提出的特征之间并未产生较强的冗余。

3.5　基于特征增强的多任务 LSTM 预测模型

本小节描述基于特征增强的多任务 LSTM 预测模型，首先介模型的输入数据构建，然后介绍预测模型结构。



(a) 特征观测窗口长度为 3　　(b) 特征观测窗口长度为 18

图 9 特征相关性图

3.5.1 基于滑动窗口的序列式特征

对于每个时刻的原始负载 x_t，都可以采用3.4.2小节所述的方法在长度为 n 的特征观测窗口内 $(x_{t-n+1}, x_{t-n+2}, \ldots x_t)$ 提取对应的 2 两种基础特征以及 4 种波动特征。Xia 等人[10] 采用 WSVM 模型，仅使用待预测点 x_t 所确定的这个时间窗口内的特征进行预测。然而系统状态的变化是存在惯性的，下一时刻的负载趋势变化可能受到之前若干个时刻的时序波动特征的影响。合理的方法应该是同时考虑之前一段时间 w 内的特征所组成的特征序列，以及原始负载序列。注意这里的 w 为考虑对序列建模时的时间窗口，反映了多远时间内的特征会对当前预测有帮助，本文将此窗口称为历史观测窗口。而 n 则为提取每个时刻所对应的特征时划定的特征观测窗口，两个值并不一定相同。为了构建训练样本，特征序列和原始负载序列均需按照此历史观测窗口 w 进行切片操作，切片时按照每 w 时间步分为一个样本，每次切片得到一个样本后窗口向前滑动一个单位。如图7中最右侧虚线方框左端部分所示。

3.5.2 多任务结构与特征增强特性

与原始负载序列 $\mathrm{x} = (x_{t-w+1}, x_{x-w+2} \ldots x_t) \in R^w$ 不同，提取到的特征序列含有更加丰富的高层次的波动性含义。设特征序列表示为：$F_t = (F_t^{avg} \ldots F_t^{lts}) \in R^{d,w}$，其中 w 为序列建模对应的时间窗口，d 为特征个数，每行为一种特征所组成的序列，如 $F_t^{msdc} = (f_{t-w+1}^{msdc}, f_{t-w+1}^{msdc} \ldots f_t^{msdc}) \in R^w$。由于 f_{msdc} 组成的特征序列 F_t^{msdc} 体现了“加速度”在历史观测窗口中的变化模式，所以一种合理的做法是将特征序列和原始的负载序列分别建模。所以这两种序列的建模过程可以看做两种独立的任务，考虑到长短期记忆网络[36]（LSTM）在捕捉长期依赖方面的优异表现，本文使用两个独立的 LSTM 网络分别

对原始负载序列和特征序列进行特征提取。如式3.8，其中输入 x_t 为该时刻原始负载值，$LSTM_{raw}$ 为负责该部分特征提取的 Raw Series LSTM Encoder，输出 h_t^r 为融合了之前原始序列信息的该时刻的隐层表示。输入 f_t 为该时刻对应的特征观测窗口中提取的特征向量，$LSTM_{feature}$ 为负责该部分特征提取的 Feature LSTM Encoder，输出 h_t^f 为融合了之前特征序列信息的该时刻的隐层表示。

$$\begin{aligned} h_t^r &= LSTM_{raw}(x_t) \\ h_t^f &= LSTM_{feature}(f_t) \end{aligned} \tag{3.8}$$

图7中所示的，Raw Series LSTM Encoder 以及 Feature LSTM Encoder 即为对应的部分。

此外，考虑到本文所提出的 4 种波动性特征主要衡量的是负载的趋势以及变化的程度，这些特征在不同的原始负载的取值下可能代表着不同的含义，比如在高负载时，较大的 f_{rsi} 值可能说明上升趋势已经接近上限，未来可能呈现下降趋势；而在低负载时，较大的 f_{rsi} 值可能说明还未达到最大值，未来可能还会上升。不同的原始负载值相当于不同的“参考点”，特征值在不同参考点下又会增加新的隐含特征。所以本文使用另外一个 LSTM 层，显式的将 Raw Series LSTM Encoder 以及 Feature LSTM Encoder 在每个时间步的隐层输出进行融合，试图捕捉特征序列和原始负载序列之间通过交互产生的隐含特征，从而达到特征增强的效果。结构如图10所示。

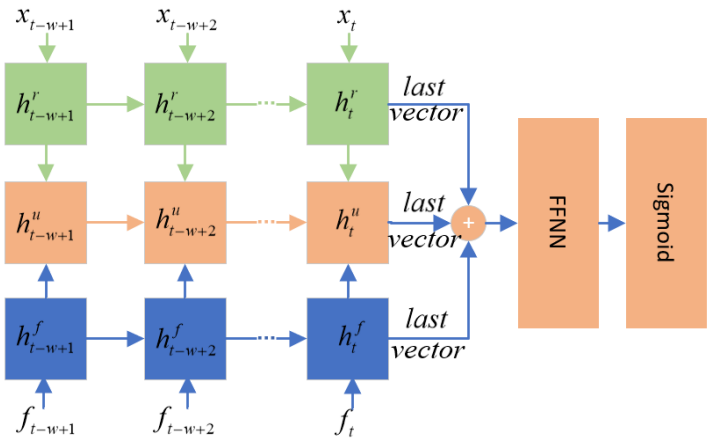


图 10　特征增强 LSTM 模型结构图

图10中，中间的一层即为 Fusion LSTM Encoder。$h^r_{t-w+1}, h^r_{t-w+2} \ldots h^r_t$ 是每个时间步由 Raw Series LSTM Encoder 产生的隐层表示. $h^f_{t-w+1}, h^f_{t-w+2} \ldots h^f_t$ 是每个时间步由 Feature LSTM Encoder 产生的隐层表示. 两者被拼接起来输入另外的 LSTM 做特征融合, 所以融合层的每一个时间步的输入即为 $x_t^f = concate(h_t^f, h_t^r)$. 可以在最后一个时间步得到三种

隐层表示：原始负载序列的隐层表示 h_t^r，特征序列的隐层表示 h_t^f，融合层的额隐层表示 h_t^u，最后，整个序列的隐层表示为 $v_{final} = h_t^u + h_t^r + h_t^f$，然后 v_{final} 通过一个激活函数为 $sigmoid$ 的单层前神经网络得到输出概率值，即 $\hat{y} = sigmoid(wv_{final} + b)$，对应为该点是转折点的概率。

3.6 实验验证

实验验证分为三个部分：1）波动特征有效性实验。为了验证本文提出的转折点波动特征的有效性，首先在 Xia 等人[10] 所采用的传统机器学习模型上进行对比实验。2）特征增强的多任务 LSTM 模型的有效性实验。为了验证本文提出模型的性能，需要分别进行加入波动性特征以及不加入波动性特征的传统机器学习模型和本模型的对比实验。3）模型简化实验。分别去除输入的额外特征序列以及多任务结构进行实验，以验证本模型各模块的有效性。

3.6.1 实验环境说明

实验环境见表2和表3所示。软件环境中 TA-Lib[43] 和 Tsfresh[44] 是两款时间序列分析工具，本文主要采用两者提供的特征计算函数进行特征提取。其中 TA-Lib 被广泛应用于金融时间序列的分析，提供了多种金融指数的计算函数，而 Tsfresh 更偏向通用的时间序列分析，提供了诸如滑动平均值、方差等基础特征的计算函数。Keras[45] 是一个由 python 编写的开源神经网络库，可以作为 Tensorflow[46] 的高阶应用程序接口，本文使用 Keras 进行模型的搭建。模型的评价指标方面使用 Scikit-learn[47] 中提供的函数进行计算。

表 2 负载转折点预测实验环境硬件配置表

硬件	详细描述
CPU	(英特尔)Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
内存	8.00 GB
显卡	NVIDIA GeForce GTX 1060 with Max-Q Design

表 3 负载转折点预测实验环境软件配置表

软件	详细描述
操作系统	Microsoft Windows 10 家庭中文版 (64 位)
TA-Lib	0.4.17
Tensorflow-gpu	1.10.0
Tsfresh	0.11.2
Keras	2.2.4
Scikit-learn	0.19.1

3.6.2　数据集

为了评估所提出模型的性能，我们在开源的谷歌云服务集群 trace 上进行了实验。此 trace 为谷歌在 2011 年发布的数据集，为超过 670,000 个作业的实际运行跟踪日志，包含 29 天内 12,500 多个节点上的约 2500 万个任务[16, 48]。每 5 分钟系统报告一次包括 CPU，RAM 等在内的各任务的资源使用情况。在本文的实验中，主要考虑 CPU 使用率。在给定时间点的服务器负载是该特定节点上所有正在运行的任务的总负载，因此本文累计每个节点上某时刻所有任务的资源消耗量作为该节点上该时刻的负载。每个节点总共有 8352 个样本数据点。随机选择 ID 为 207776314（M1），908054（M2），1273805（M3）的三个节点作为本文的三个实验数据集。由于每个数据集包含 8352 个点，如第2.2.2节所述，如果将 PLR 算法的分割阈值设置过大就会造成转折点过少，无法支持模型的训练，所以本文设置 PLR 算法的分割阈值为 0.0015 进行分割以及标记，PLR 的实现来自 Keogh[11] 等人。之后分别按照时间顺序将每个数据集划分为训练集，验证集和测试集。前 80％的数据用于训练，接着 10％的数据用于验证参数，最后 10％的数据用于测试。标记后的训练集效果分别如图11, 图12和图13所示。其中红色三角代表标记出来的转折点，蓝色圆点代表普通点。

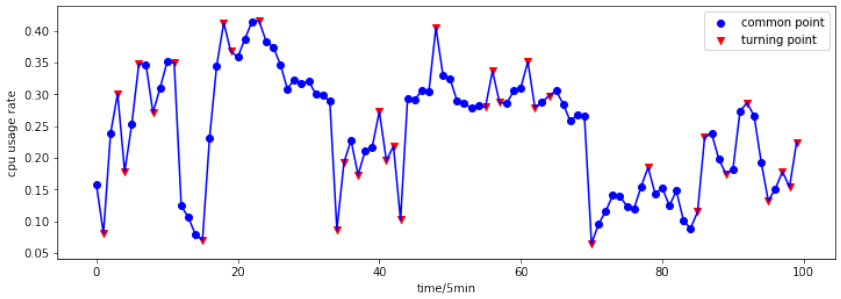


图 11　M1 标记效果图

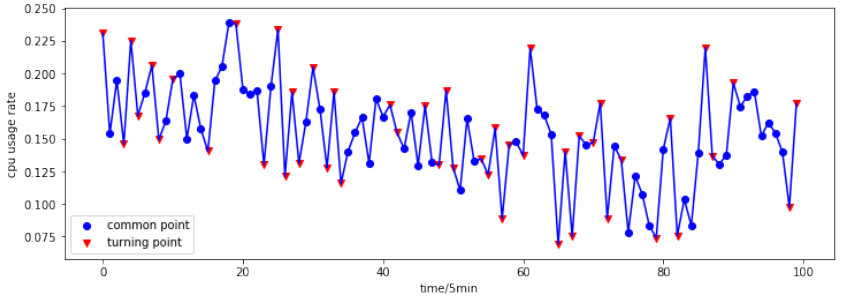


图 12　M2 标记效果图

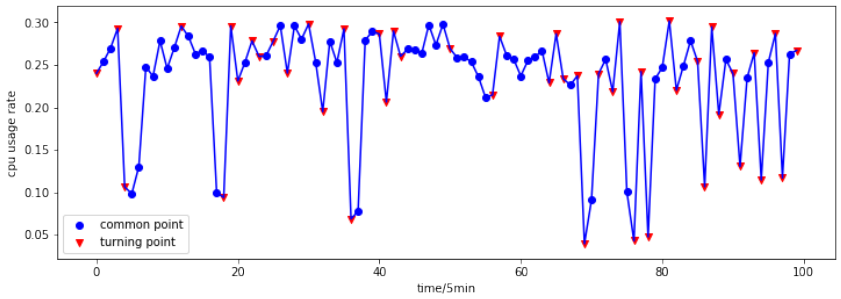


图 13 M3 标记效果图

3.6.3 训练方法

本文使用 minibatch 随机梯度下降和 Adam[49] 优化器来训练模型。learning rate 设置为 0.001。batch size 设置为 64。参数通过标准反向传播来学习，损失函数为二值化交叉熵：

$$loss = -\frac{1}{N}\sum_{n} y_n \log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n) \tag{3.9}$$

其中 N 为样本总数，y_n 为某样本对应的真实标签，$\hat{y}_n$ 为该样本对应的模型输出，代表将该样本预测为正例的概率。

3.6.4 评价指标

我们使用 F1 值作为模型评估的主要指标，F1 值是由精确率（precision）和召回率（recall）计算出来的。precision 反映了分类器的分类精度，表示所有预测为正例的样本中正确预测的样本比例。racall 反映了分类器的召回能力，表示所有正样本中被正确预测的样本比例。precision 值和 recall 均由真正例（TP）、假正例（FP）、假负例（FN）三种指标计算来。如公式3.10所示：

$$\begin{aligned} F1 &= \frac{2 \times precision \times recall}{precision + recall} \\ precision &= \frac{TP}{TP + FP} \\ recall &= \frac{TP}{TP + FN} \end{aligned} \tag{3.10}$$

3.6.5 对比算法

为了验证本文提出的波动性特征的有效性，需要对比加入此种特征前后算法性能的变化；为了验证本文提出的基于特征增强的多任务 LSTM（FEMT-LSTM）转折点预测模型的效果，需要和目前最优的转折点预测算法进行对比；同时还需进行 FEMT-LSTM

的模型简化实验，验证各部分的有效性。基于上述思路，选取基线模型如下：

LR-Basic：Logistic Regression（LR）是一种广泛应用于文本分类，欺诈检测等领域的线性分类方法，LR-Basic 使用表1中所列 3 种基础特征作为输入。对于此模型中的正则项参数 C，设置搜索范围 $1\times10^{-4},1\times10^{-3},1\times10^{-2},1\times10^{-1},0.5,1,10,15,20,100$；对于特征观测窗口长度 n 设置搜索范围 3,6,9,18，取 f1 值最优的参数组合。

LR-Fluctuant：为了验证本文提出的特征的有效性，本文称使用表1中所列全部 7 种特征作为输入的 Logistic Regression 为 LR-Fluctuant，正则项参数和特征观测窗口的搜索范围同 LR-Basic 模型。

WSVM-Basic：本文将使用表1中列出的基本特征作为输入的加权 SVM 称为 WSVM-Basic。加权 SVM（WSVM）[12] 是 SVM 的修改版本。SVM[50] 的主要思想是生成一个分类超平面，它将两类数据与最大边界分开。当每个训练实例具有不同的权重时，标准 SVM 被扩展到加权 SVM，使得普通点和转折点对损失提供不同的贡献。Xia 等人[10] 使用分段线性分段算法生成虚拟机需求的突变点，然后使用 WSVM 和基于窗口的基本特征来预测下一步虚拟机需求是否会突然改变。虽然这与本文研究的问题不同，但也将他们的方法作为基线之一。正负样本的权重设置为训练集中样本数的反比，采用 RBF 核函数，惩罚项参数 C 和核函数系数 $gamma$ 的搜索范围均和 LR 相同，特征窗口长度 n 的设置范围也相同。

WSVM-Fluctuant：为了验证本文提出的特征的有效性，本文使用表1中全部 7 种特征的 WSVM 模型来进行实验，将此模型称为 WSVM-Fluctuant。参数设置和 WSVM-Basic 模型相同。

P-LSTM：本文称只使用原始工作负载序列作为输入的 LSTM 模型为 P-LSTM 模型。为了验证 FEMT-LSTM 模型的特征增强特性，将此模型作为基线之一。

S-LSTM：本文称将原始工作负载序列和特征序列一起作为输入，但只采用单独的 LSTM 进行特征提取的模型称为 S-LSTM，即每一个时间步的输入均是该时刻对应的全部的 7 种特征。为了验证 FEMT-LSTM 模型多任务结构的作用，将此模型作基线之一。

三种深度学习模型：P-LSTM，S-LSTM 以及 FEMT-LSTM 中使用的 LSTM 隐层神经元个数均设置为 256，特征观测窗口大小 n 和 WSVM-Fluctuant 设置为同一值，历史观测窗口 w 设置为 10 或 5。所有模型均使用 sklearn 和 keras 实现。

3.6.6 实验结果与结论

三个数据集上的对比结果如表4和图14所示。本文的 FEMT-LSTM 在所有三个数据集中都获得了最高的 F1 得分，图14直观展示了全部 6 个模型在 3 个数据集上的 F1 值，可以看到代表 FEMT-LSTM 的绿色部分明显高于其它模型。相比 SVM-Fluctuant，FEMT-LSTM 在三个数据集上分别提升了：4.2，2.7，5.1 个百分点，平均提升 4.0 个百分点。相比另外 6 种模型，分别在三个数据集上最少提升 4.2，0.45，1.44 个百分点，平均提升 2 个百分点。

表 4 负载转折点预测算法对比结果表

Method	M1			M2			M3		
	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall
LR-Basic	0.5008	0.4029	0.6620	0.4505	0.3891	0.5348	0.2846	0.4176	0.2159
LR-Fluctuant	0.4973	0.4072	0.6385	0.4646	0.3962	0.5615	0.2966	**0.4483**	0.2216
WSVM-Basic	0.5000	0.4201	0.6206	0.4487	0.3736	0.5614	0.3949	0.4405	0.3579
WSVM-Fluctuant	0.5237	0.4703	0.6012	0.4789	**0.4715**	0.4866	0.4201	0.4143	0.4261
P-LSTM	0.4597	0.2993	**0.9906**	0.4964	0.4416	0.5668	0.4562	0.3651	0.6079
S-LSTM	0.5144	0.3898	0.7559	0.5011	0.4395	0.5828	0.4212	0.3290	0.5852
FEMT-LSTM	**0.5660**	**0.4774**	0.6948	**0.5056**	0.4346	**0.6043**	**0.4706**	0.3816	**0.6136**

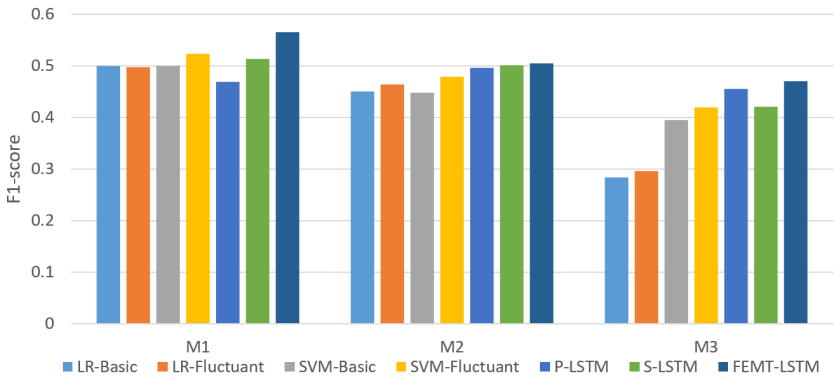


图 14 负载转折点预测算法 F1 值对比结果图

（1）**波动特征的有效性分析：**图15展示了表4上半部分的四组实验对比，即对比 LR 和 WSVM 在加入波动性特征前后的 F1 值变化。可以看出加入波动特征的传统机器学习模型优于只使用基础特征的模型。对比在三个数据集上没有波动特征的 WSVM-Basic，加入 4 个波动性特征后，分别提升了 2.3，3.0，2.5 个百分点，平均提升 2.6 个百分点。而 LR 在加入 4 个波动性特征后，虽然在 M1 数据集上效果略差于基础特征的 LR，F1 降低 0.4 个百分点，但是在其他两个数据集上分别提升 1.4 和 1.3 个百分点，所以平均有 0.8 个百分点的提升。这意味着我们提出的的特征对转折点预测是有效的。同时可以发现 LR 效果较差，这是由于 LR 模型是简单的线性模型，当特征数量较少并且样本区

分难度较大时效果往往弱于使用核函数的 SVM 模型。而 SVM 采用了径向基函数作为核函数，能够将原始的输入数据投影到高维的特征空间中，从而在高维特征空间中找到合适的超平面可以较好的将原始样本点区分开来。

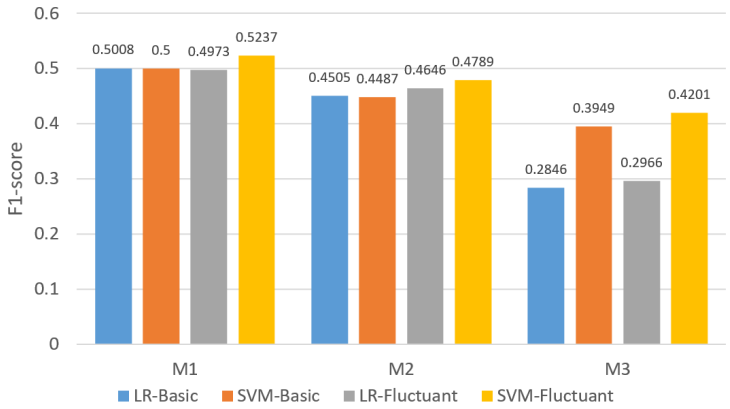


图 15　波动性特征效果对比图

（2）**特征序列的有效性分析：**为了说明采用滑动窗口方式提取的特征序列的有效性，图16展示了采用滑动窗口提取序列特征的模型 P-LSTM 和 S-LSTM 相比采用单一窗口提取特征的 WSVM 模型的对比效果。相比传统的以单个时间窗口内的特征作为输入的机器学习模型，以特征序列作为输入考虑了特征窗口间的关联性后，模型效果也能有一定的提升。本文提出的两个 LSTM 基线模型 P-LSTM 和 S-LSTM 在 M2 和 M3 两个数据集上均优于 WSVM-Fluctuant 模型，在 M2 上分别提升 1.8 和 2.2 个百分点，在 M3 上分别提升 3.6 和 0.11 个百分点。根据图16中数据计算 WSVM-Fluctuant、P-LSTM、S-LSTM 三个模型在三个数据集上的平均 F1 值分别为：0.4742、0.4707、0.4789。相比使用全部 7 种特征组成特征序列的 S-LSTM，P-LSTM 只使用了原始的负载序列，没有考虑到其他 6 种特征序列间的关联性，所以 P-LSTM 效果弱于 S-LSTM。此外，S-LSTM 效果也优于 WSVM-Fluctuant，这说明使用多个特征窗口，将特征组成特征序列的形式相比只使用单一窗口提取的特征可能达到更好的效果。

（3）**FEMT-LSTM 模型的简化实验分析：**为了说明模型结构的有效性，图17比较了简化实验过程中的对比结果。FEMT-LSTM 相比 P-LSTM（没有使用特征序列作为输入，仅使用原始负载序列），在三个数据集上分提升了 10.6，0.9，1.4 个百分点，平均提升 4.3 个百分点。这表明融合了手工设计的波动特征之后，通过学习特征序列间的相互作用关系能够增强模型的性能。同时可以看到，FEMT-LSTM 模型优于 S-LSTM（没有采用分离式的结构），在三个数据集上分别提升 5.1，0.5，4.9 个百分点，平均提升 3.5 个百分点。由于原始负载序列和特征序列有着不一行的含义：特征序列是在原始序列基础

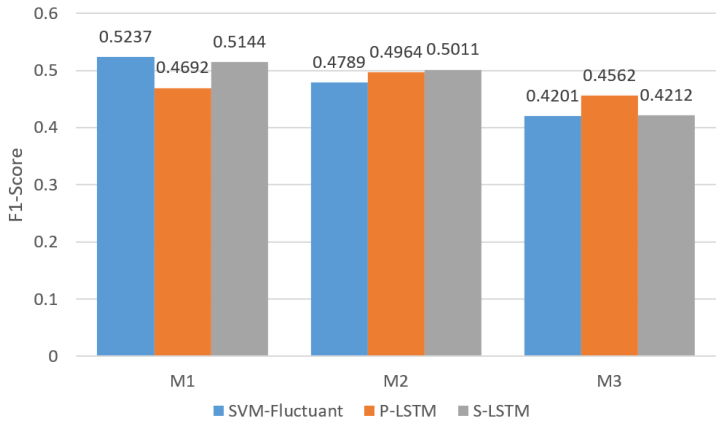


图 16 特征序列效果对比图

上计算得来，描述了人为规定的指标的变化，而更多有效的信息，还需要通过参考原始序列得到，所以这种分开建模然后再进行融合的策略是有效的。

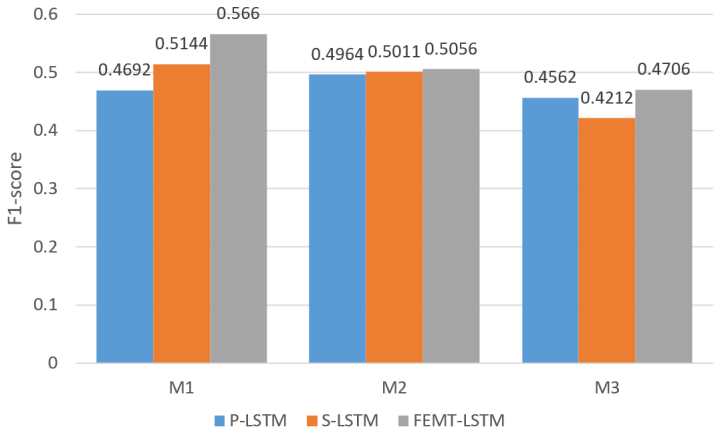


图 17 模型简化实验对比图

3.7 本章小结

本章针对现有时序转折点预测模型的两个缺点，首先从特征出发，提出了 4 种波动性特征，并给出了合理的解释。然后基于一种多任务结构，采用 3 个长短期记忆网络分别融合各类特征，使网络能够显式的捕捉由两种信息信息交互产生的隐式特征，最终起到特征增强的效果。在谷歌 trace 数据上进行的实验表明本文的模型在 F1 指标上至少提升 2 个百分点。

第四章　基于通道注意力卷积模型的任务失败预测算法

4.1　研究背景

云环境中的任务可能由于多种因素的影响而执行失败：由于云平台系统的大规模、异构性以及分布式的结构，节点故障是不可避免的现象[13]，节点故障发生之后，往往导致运行于其上的任务执行失败。任务也可能由于自身的 Bug 而发生一些异常导致运行失败。此外，由于云环境下大量不同任务共享软硬件资源，这种资源竞争也可能导致任务最终的执行失败。由多元因素导致的任务执行失败的现象是复杂的。然而这些失败的任务会大量浪费系统的软硬件资源，造成云平台上的资源浪费、效率低下，这种情况对执行时间较长的任务来说尤其明显[15]。所以在任务真正的执行失败之前区别出这些任务，能够有效地降低不必要的系统开销。

最近的研究发现，任务的资源消耗情况和失败的现象之间有着某种关联性。失败任务在这些资源使用指标上可能会存在一定的模式[14, 16, 51]。建模并提取这些失败任务中蕴含的特定模式，以便提早终止失败的任务，节省系统资源，是十分重要的。目前最佳的任务失败预测方法是基于时间序列分类技术进行的，即将任务的各种资源消耗指标所组成的时间序列视为多变量时间序列，采用循环神经网络结构，捕捉时间维度各资源使用时间序列的变化模式，进而区分出失败任务和成功任务[14]。本文也沿用这个思路研究云环境下基于多变量时间序列分类模型的任务失败预测算法。

多种资源消耗监测数据组成的多变量时间序列往往存在以下两个特点：其一，不同变量在相同的时刻受到之前时刻的影响程度可能是不同的，即不同变量内部特征具有独立性。比如 CPU 使用率可能会发生突变，其受到长期的历史 CPU 利用率的影响可能较小，而内存利用率则变化较缓慢，其受到较长时间的历史数据影响的可能性较大；其二，不同变量的时序变化特征也可能不具有同时性，即不同变量间的特征通过交互作用，具有相关性。比如当某段时间内 I/O 访问较频繁，那么之后一段时间 CPU 使用率才可能有较高的值。如图18所示，Channel 1 到 Channel 3 代表 3 种监测到的资源消耗时间序列，各通道的变化频率是不同的，且先后峰值的出现可能代表着某种通道间的交互式特征。图19是谷歌云服务集群中一个真实的任务在执行时的各种资源消耗时间序列，方框中呈现了两种资源消耗时间序列的变化，CPU 利用率首先下降，之后内存利用率

才出现明显的上升，这种峰值交替出现的情况可能代表着一些任务执行中特定的行为，比如加载数据等；与此同时，可以看出 CPU 和内存的变化频率是明显不同的，而在特定的任务行为下，这种波动频率的区别可能也代表着某些特殊情况的出现。

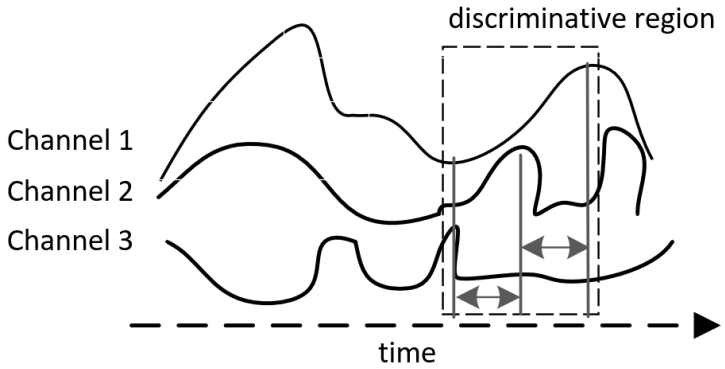


图 18 多变量资源消耗时间序列特点示意图

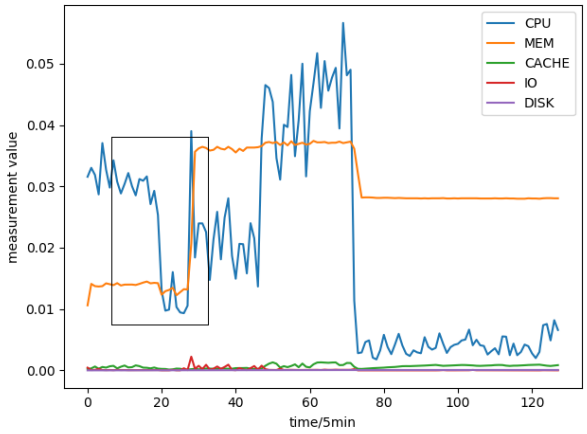


图 19 谷歌云服务集群中任务示例图

然而目前任务失败预测的研究主要是基于时间序列分类算法，却往往忽略了上述的资源消耗时间序列的变化特点，或者只考虑其中之一。例如，Chen[13] 以及 Islam[14] 提出的基于循环神经网络的任务失败预测模型，将同一时刻的多个监测值同时输入循环神经网络模型，即每一时刻各监测指标组成该时刻对应的资源消耗向量 $x_t = (cpu_t, ram_t, io_t ...)$，循环神经网络通过考虑该向量在时间维度的变化逐步提取序列特征。但是这种做法认为 x_t 是一个整体，忽略了各个监测指标内部 cpu_t, ram_t 等不同的时序变化模式。目前，在多变量时间序列分类任务中表现最优的模型是深度残差网络（ResNet）[8]，它采用一维卷积，沿着时间维度同时提取各个变量在卷积核所确定的时间跨度内的特征，然而由于卷积核是长宽固定的矩形，导致对所有变量都使用同一个时间跨度，不利于提取多个变量内部可能具有的不同时间跨度的依赖关系。虽然有些基于深度学习的多变量时间序列分类模型考虑到了不同变量内部的作用关系的差异，例如 2014 年提出的 MCDCNN[20]

，它采用分离式的卷积结构分别对每条时间序列进行特征提取，但是却没有考虑到变量间的作用关系。

针对上述问题，本节的贡献如下：

1）设计并实现了基于通道注意力卷积模型（CMSA-CNN）的多变量时间序列分类算法以预测失败任务。该算法采用分离式的卷积结构，提取不同通道内部各自的时序变化特征，而后采用多头自注意力机制提取通道之间的时序变化特征。在谷歌云服务集群 trace 上进行了多组实验，在 AUC 指标上相比 Chen[13] 的模型在早期和晚期预测时平均提升 9.5 个百分点，相比 Islam[14] 的模型平均提升 1.1 个百分点，相比目前最优的多变量时间序列分类模型深度残差网络[8] 平均提升 0.2 个百分点。

2）通过对照实验，分别去除分离式的卷积模块和通道间自注意力模块，验证了本文提出的基于通道注意力卷积模型（CMSA-CNN）的多变量时间序列分类算法各模块的作用。

4.2　问题定义

任务失败预测问题的形式化定义如下：给定监测到的某任务执行过程中消耗的各种资源数据 $X = (x_1, x_2 \dots x_i \dots x_T) = \left(x^1, x^2 \dots x^j \dots x^D\right) \in R^{T,D}$ ，其中 $x_i \in R^D$ ，$x^j \in R^T$ 。T 为时间序列长度，D 为监测指标个数，也即 D 个通道，学习函数 f ，将其正确映射到预定义标签集合 $L = \{c_1, c_2\}$ 中的某一个元素上。X 由多条监测数据时间序列组成。其中每条时间序列都是一种任务运行时的监测指标，这样在每一个时间点，可以得到一个 D 维的向量，完全描述了次任务在此刻的执行状态。预测的目标是根据给定的维度为 D 的 T 组特征对此任务的执行结果进行预测。执行结果定义为两种：“执行成功”标记为 0，“执行失败”标记为 1。本文采用谷歌集群 trace[3] 说明文档中关于任务执行结果的定义方式，如图20所示。图中的圆形代表任务所处的状态，连线代表发生的事件，由图知有 5 种事件能够使任务由 RUNNING 状态变为 DEAD 状态，即：EVICT、FAIL、KILL、LOST、FINISHED。其中“执行失败”包含四种状态：EVICT、FAIL、KILL、LOST，“执行成功”包含一种状态：FINISHED，实验部分将按照此种定义方式从谷歌集群 trace 中抽取数据。为了能够在任务结束之前根据预测结果采取对应的措施，以节约系统资源，就需要决定预测的时机。预测时机越早，即 T 越小，预测出失败任务可能带来的资源节省量就越多，但同时由于得到的资源消耗时间序列长度变短，时序特征的提取难度变

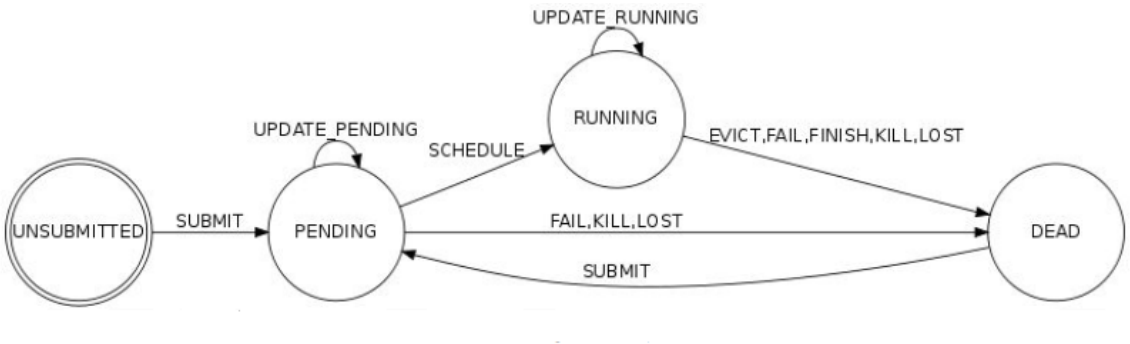


图 20 谷歌集群 trace 文档中的任务状态转换图

大，预测效果可能会变差；相反的预测时机越晚，即 T 越大，预测出失败任务带来的资源节省量就越小，而由于得到的资源使用时间序列的长度变长，较容易提取出时序特征，可能得到更好的预测结果。

在 Chen[13] 的研究中，作者假设已经获知了每个任务的执行总时间，然后在任务执行时间的四分之一、二分之一和结束处进行预测。然而真实情况下任务的执行时间一般难以获得，也就不易根据某个任务的执行时间调整预测时机。所以本文采用固定时机的预测方法，即设置时间长度 T，当收集满长度为 T 的资源消耗时间序列之后立即进行预测。所以本文根据资源使用时间序列长度 T 的不同设置，在早期和晚期两个阶段进行预测，研究不同监测长度对预测效果的影响。

4.3 算法框架

本文提出的的算法框架如图21所示，它由三个部分组成: 通道内一维卷积模块、通道间多头自注意力模块、全连接分类模块。输入为监测到的某任务一段时间内指定的资源消耗指标数据组成的时间序列 $X \in R^{T,D}$ 首先将输入的多变量时间序列按照通道进行切分，得到 D 条长度为 T 的单变量时间序列，对应着每个指标各自的的监测时间序列；由于不同监测指标序列在时间维度上可能表现出不同的变化模式，而若将各个指标一起考虑，势必会丢失掉一些不同序列内部的信息。所以对于每条单变量时间序列，本文首先采用一维通道内卷积模块提取通道内特征。而接下来的多头自注意力模块负责计算两两通道间的相关性，从而捕捉不同通道间的交互作用特征。最后通过一个全连接网络进行一次维度变换，并使用 sigmoid 激活函数计算得到每个标签的概率，输出概率最大的标签，作为此任务对应的预测终止状态。由于本算法的分离处理多个监测指标通道的思路类似于 MCDCNN，下边也给出多变量时间序列分类模型 MCDCNN[20] 的算法框

架，如图22所示，可以看到 MCDCNN 仅仅考虑不同通道具有各自独立的特征，而使用一维卷积分别进行提取，但是却忽略了不同通道间可能存在的相关性特征。此外为了提取更显著的通道内特征，MCDCNN 每次在一维卷积之后使用局部的最大池化操作，这样虽然缩小了得到的特征图的规模，但是仅根据局部信息就决定忽略掉某些位置的特征是不妥当的，非局部最显著的特征有可能在全局范围的视角下起作用，尤其是可能对通道间信息的提取是有利的。所以为了在全局视角下保留更多的通道内信息，以辅助通道间信息的提取，本文在每次卷积之后不进行局部最大池化，而是在卷积模块之后采用全局平均池化。

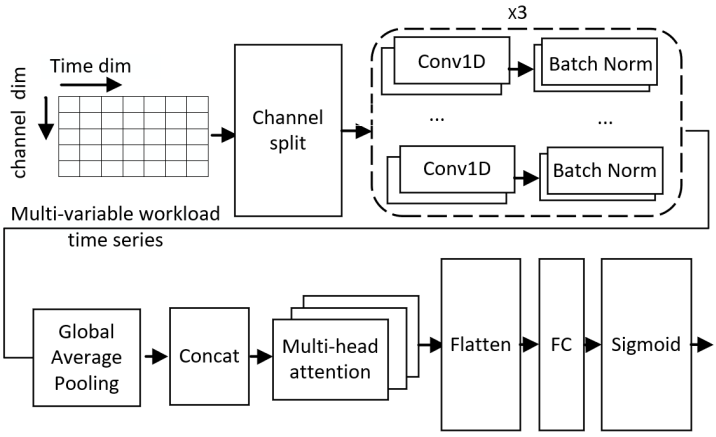


图 21 任务资源消耗时序特征提取框架图

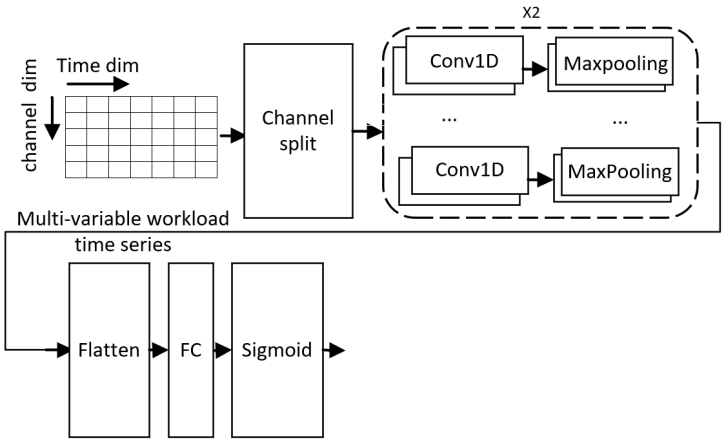


图 22 MCDCNN 算法框架图

4.4 通道内一维卷积模块

由于不同变量内部的特征可能具有独立性，分别提取每个维度的时序特征是一个合理的选择。针对 $T \times D$ 维的多变量时间序列，本文采用切分的方法，单独剥离出各个维度的时间序列，分别对每个变量进行时序特征提取。这样一个 $T \times D$ 的矩阵变换为 D

个 $T \times 1$ 的向量，向量中的第 i 行的值代表时刻 i 这个变量的取值。对每个 $T \times 1$ 的向量，本文使用一维卷积模块进行特征提取。由于 LSTM 等循环神经网络在结构上的优势，以往常常被用来从序列中提取特征，然而随着序列长度的增加，这种循环式的结构难免会丢失部分远距离的信息，另一方面，由于失败任务可能仅仅在某些时间片段内表现出突出的特征，LSTM 并不擅长提取这种局部的特征。所以本文使用一维卷积来进行通道内特征的提取。一维卷积结构是指卷积核的移动方向为单一方向的卷积操作。其卷积核的大小往往和输入数据的某一维相等，卷积核在另一维度上进行滑动，依次计算卷积结果。相比循环神经网络一维时间卷积也能捕捉一定范围的时间维度依赖关系。为了提取较长范围的时序特征，本文采用更深的网络：堆叠 3 层的卷积结构，以此扩大后续卷积层的覆盖范围，并在每个卷积层之后加入 Batch Normalization 层[52]，组成一个卷积单元。该模块共由三个卷积单元组成，如图23所示。假设给定输入多变量时间序列

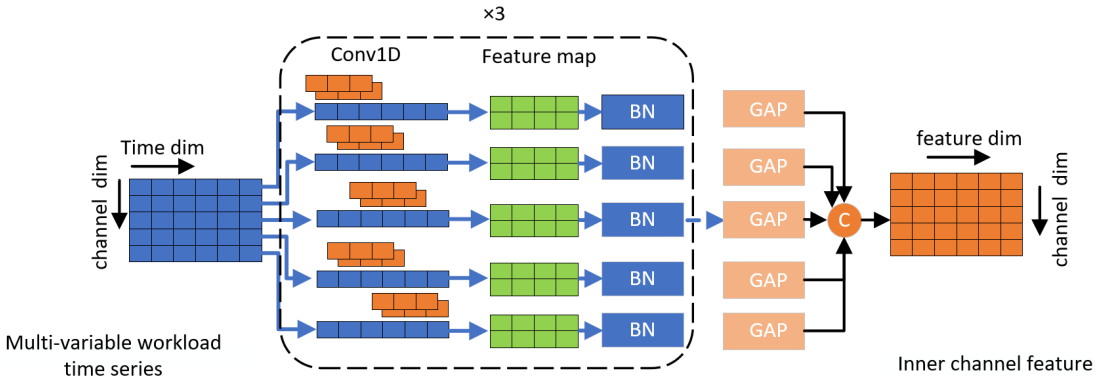


图 23 通道内一维卷积模块结构图

$X = (x_1, x_2 \dots x_i \dots x_T) = \left(x^1, x^2 \dots x^j \dots x^D\right)$，其中 $x_i \in R^D$，$x^j \in R^T$，例如图中 $T = 6$，$D = 5$。每一个变量即为一个通道，切分操作即为将给定输入按照通道切分为 D 条时间序列：

$$\text{split}(X) = \left(x^1, x^2 \dots x^j \dots x^D\right) \tag{4.1}$$

4.4.1 卷积单元

对于任意一条时间序列 $x^j \in R^T$，均采用 N 个 kernel size 为 K 的一维卷积进行卷积操作。第一个卷积单元的卷积核大小记为 $\text{Kernel}_1 \in R^{K,1}$，这是因为 $x^j \in R^T$ 是一维的向量，只需使用宽为 K 的 1 维卷积核即可覆盖全部通道维度：

$$F_i^n = Conv1D(x_{i:i+K}^j \otimes W_1^n + b_1^n) \tag{4.2}$$

其中 $\otimes$ 为卷积运算，$F_i^n \in R$ 为得到的特征图中的一个激活输出，$x_{i:i+K}^j \in R^K$ 为时间序列中一段长度为 K 的连续片段，$W_1^n \in R^K$ 为第一个卷积单元中的第 n 个卷积核，$b_1^n \in R^1$ 为偏置项。即每次考虑时间序列中的连续 K 个取值，计算卷积结果。一维卷积从长度为 K 的时间片段中抽取特征，生成一个标量，代表此位置对应的时序特征，滑动过程结束之后得到 $F^n = (F_1^n, F_2^n ... F_{T-K+1}^n) \in R^{T-K+1}$ ，为这个卷积核进行一次卷积操作之后得到的一个特征图。同时使用 N 个大小相同的卷积核进行卷积，则得到 $F = (F^1, F^2 ... F^N) \in R^{T-K+1,N}$，为经过第一个卷积模块后得到的所有特征图。接下来加入 Batch Normalization 层，以防止梯度消失，加速模型收敛。通过堆叠相同的模块操作，例如第二卷积单元，第三卷积单元，可以逐层抽取通道内时序特征，逐渐扩大捕捉到的远距离依关系。本文采用 3 层卷积模块堆叠的形式后续过程和此过程相似不在赘述。所有卷积单元的卷积核个数 N 设置为 128，第 1、2、3 卷积单元的卷积核大小 K 分别取 8，5，5。

4.4.2　通道内时序特征

输入数据经过最后一个卷积单元之后得到特征图为：$F_{last} = (F_{last1}, F_{last2} ... F_{lastD})$ ，其中 $F_{lasti} \in R^{T', K}$ 为第 i 个通道的特征图，T' 为特征图在时间维度的宽度，K 为每个卷积模块中使用的卷积核个数。为了简化模型，本文在不同的卷积模块中使用相同数量的卷积核，但卷积核大小有不同的调整。由于 $F_{last} \in R^{D, T', K}$ 为三维张量，过多的参数可能会导致过拟合从而减低模型的泛化能力。所以对每一个通道，本文在最后一个卷积模块之后加入全局池化层（global average pooling），沿着时间维度得到特征图的全局平均值：

$$F_{outi} = GlobalAveragePooling(F_{lasti}) \in R^K \tag{4.3}$$

所以所有通道内时序特征向量就组成了通道内特征矩阵 $O = (F_{out1}, F_{out2} ... F_{outD}) \in R^{D,K}$，即为提取到的最终的通道内特征。每一行对应着一个特征通道。

4.5　通道间多头自注意力模块

进行失败任务预测的前提是获得一个较好的任务监测指标组成的多维时间序列的特征表示，除了原始多变量时间序列片段中的时序依赖性，不同的时间序列之间的相关性，也即相互影响的特性，也是表征系统状态的必要条件。由于之前通道内一维时间卷

积模块，将不同通道分类开来，并未考虑不同变量 (通道) 的相互影响，所以需要设计一种结构来捕捉通道间的相互作用。

注意力机制（Attention mechanism）[53] 常常用来辅助循环神经网络对时间模式进行建模，它可以对输入序列分配不同的权重，来使输出分别以不同的程度关注输入的每一部分。通过注意力得分的计算，输入的每部分都会得到一个分数，代表其重要性高低。用这个分数在进行归一化操作之后对原始序列进行加权，就能够得到一个更加关注于重要部分的上下文表示。自注意机制（self-attention），是一种特殊的注意力机制，它旨在捕获单个序列的依赖性。自注意机制已成功用于各种 NLP 任务中，包括阅读理解[54]和自动摘要[55]。

为了描述不同通道间的相互作用，本文受到 NLP 领域机器翻译模型 Transformer [53] 的启发，采用 Transformer 中的多头自注意力机制建模通道特征的相关性。值得注意的是，本文采用自注意力机制是用来计算每一个通道特征和所有通道特征的相关性，进而得到一个融合了通道间相关性的特征表示；而传统的序列建模任务常常以时间步为单位描述不同时间步间的相关性。通常，自注意力机制可以定义为将查询 (query) 和一组键值对 (key-value pair) 映射到一个输出，其中查询，键，值和输出都是向量。对于每个位置，我们计算该位置的查询向量与每个其他位置的键向量之间的内积，以获得对该序列中其他位置的注意力得分。使用这些注意力得分，可以计算 value 向量的加权组合。如图24所示，与 transformer attention 不同，本文提出的通道间自注意力机制相当

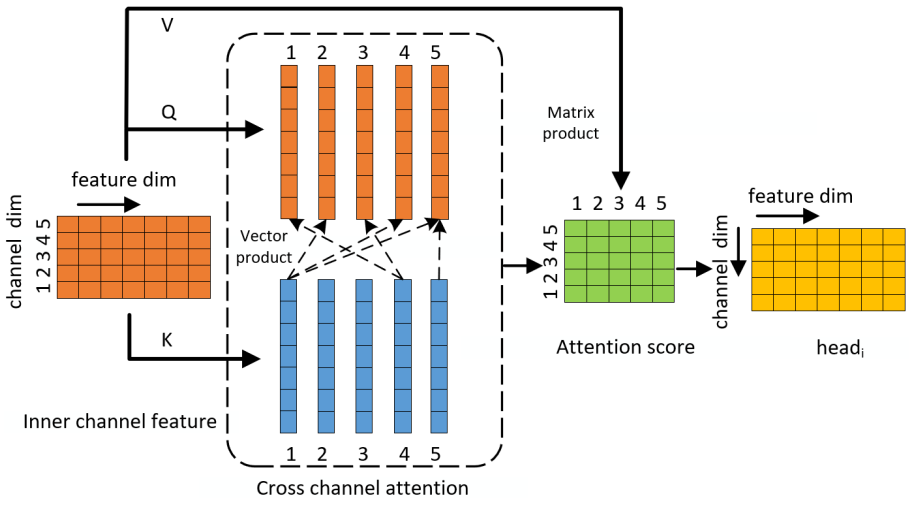


图 24 通道间自注意力模块结构图

于在不同通道间构造了一个二分图，图中的边连接两两通道，边上的权为注意力得分的大小，代表着通道间的相关性。通过通道内一维卷积模块可以得到通道内特征矩阵

$O = (F_{out1}, F_{out2}...F_{outD}) \in R^{D,G}$，$D$ 为通道个数，G 为上文所述的卷积核个数 K，此处是为了避免和下文“键”矩阵的记号混淆。如图中左侧矩形所示。由于不同通道并没有严格的时序上的顺序关系，所以每次计算注意力得分时并不需要考虑位置信息，相反，原始的 Transformer 结构为了建模位置信息还加入了 Position embedding，这一点也是本文和 transformer attention[53] 的不同之处，如图25所示。具体的，通道间自注意力机制可以描述为：

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{G}})V \tag{4.4}$$

其中 Q,K,V 分别为 query，key 和 value 矩阵，由于本文使用的是自注意力，所以有 $Q = K = V = O \in R^{D,G}$，公式4.4中分母里的 G 为 Q 矩阵和 K 矩阵的列的维度。为了提高 attention 机制抽取特征的能力，本文采用和 Transformer attention[53] 相同的多头（multi-head）机制，将 Q,K,V 分别投影到不同的空间中，然后再使用式4.5计算自注意力，即：

$$\begin{aligned} &H = MultiHead(Q,K,V) \in R^{D,hG} \\ &MultiHead(Q,K,V) = Concat(head_1...head_h)W_m \\ &head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4.5}$$

其中 h 为注意力 head 的数量，W 均为参数，$W_i^Q \in R^{G,G}$，$W_i^K \in R^{G,G}$，$W_i^V \in R^{G,G}$，$W_m \in R^{hG,hG}$。得到的 $MultiHead(Q,K,V) \in R^{D,hG}$ 即为融合了通道内特征以及通道间相关性的特征矩阵。将此矩阵通过维度变换即：$p = Flatten(MultiHead(\mathrm{Q},\mathrm{K},\mathrm{V})) \in R^{hDG}$，作为一个激活函数为 sigmoid 的单层全连接网络的输入，由于任务失败预测是二分类所以该全连接分类网络的神经元数量设置为 1，输出即为最终模型的预测输出，代表该任务执行失败的概率。

4.6　实验验证

本文针对谷歌云服务集群 trace[3] 数据集中的任务进行研究，实验流程分为四步：(a)trace 数据集中任务抽取。(b) 预测时机的选取。(c) 和经典任务失败预测算法的对比试验。(d) 和先进的多变量时间序列分类算法的对比试验。由于 google trace 数据集只是原始的系统级 trace 记录，以往各研究论文中并未公开构造好的实验数据集，所以本试验部分的第一个操作就是数据集的构造。数据集构造分为两步：首先从原始 trace 记录中

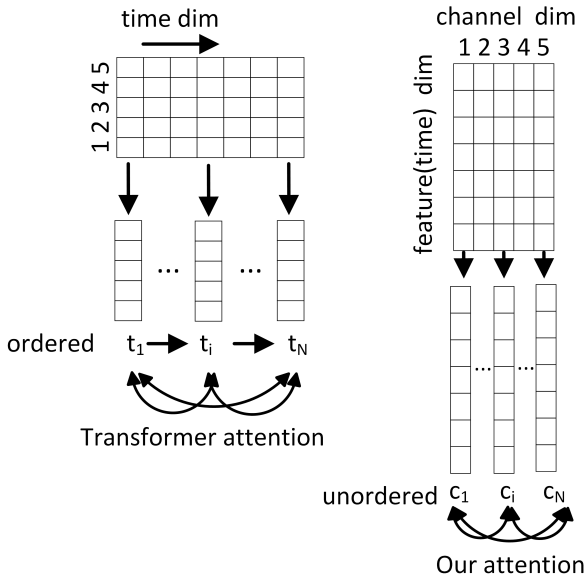


图 25 本文的注意力机制和 Transformer 的对比图

提取进行研究的目标，即“任务”，其次针对每个提取出的任务，需要在 trace 记录中解析出其每个测量周期的资源使用情况，得到由监测指标组成的多变量时间序列。同时，如前文章节4.2所述，为了研究预测时机的不同，即监测时长 T 的不同设置对预测效果的不同影响，本文对每组实验分别设置一个较小的 T 值和一个较大的 T 值，分别构造两个数据集，整体进行两套完整实验。实验的第三步是和现有的任务失败预测算法进行对比试验，而由于本文在任务失败预测算法中引入了多变量时间序列模型的一些组件，所以实验的第四步是和现有的多变量时间序列分类模型的对比试验。

谷歌集群 trace 主要由 Job events table、Task events table、Task constraints table、machine event table、Machine attributes table、Resource usage table 这 6 张表组成，其中和本研究点相关的有 (a)Task events table (b) Resource usage table 这两张表。每个作业（job）是由一个或者多个任务（task）组成，task 是谷歌集群中进行追踪记录的最小单位，由于本文是针对任务级别进行执行状态的预测研究，所以只关注这两个表的信息。

(a) Task events table：如表5所示，其中最主要的是“event type”字段，该字段记录了这条记录对应的任务事件，包括：SUBMIT(0)、SCHEDULE(1)、EVICT(2)、FAIL(3)、FINISH(4)、KILL(5)、LOST(6)、UPDATE PENDING(7) UPDATE RUNNING(8)，其中任务的终止状态分为两种：“正常结束”记为 finishe；以及“非正常结束”记为 failed。“非正常结束”包含三种：EVICT、FAIL、KILL。此表的其他字段记录了每个任务发生某事件时的属性信息。我们主要从此表中得到有关任务终止状态的信息。

(b) Resource usage table：如表6所示，记录了某个任务任务在某时刻的对某资源的

表 5 任务事件信息表

序号	字段	含义
1	timestamp	事件发生时的时间戳
2	job ID	此任务所属的作业 ID
3	task index-within the job	任务在作业中的索引
4	machine ID	此任务运行的物理机 ID
5	event type	事件类型
6	user name	任务所属的用户的用户名
7	scheduling class	任务的调度类别
8	priority	任务的调度优先级

使用情况。系统在每秒中都会进行数据采集，而标准测量周期是 300s，也即每条记录报告的是由“start time of the measurement period”和“end time of the measurement period”之间的所确定的 300s 的范围内采集到的数据的均值。和 Chen[13] 以及 Islam[14] 的研究相同，本文也仅采用其中的 5 个指标：平均 CPU 使用率、平均内存使用率、总页缓存使用率、平均磁盘 IO 时间、平均硬盘使用率。我们主要从该表中得到某个任务在某时刻的某个指标的使用情况。

表 6 任务资源使用信息表

序号	字段	含义
1	start timeof the measurement period	测量开始时间
2	end timeof the measurement period	测量结束时间
3	job ID	作业 ID
4	task index	任务索引
5	machine ID	物理机 ID
6	mean CPU usage rate	平均磁盘使用率
7	mean memory usage	平均内存使用率
8	assigned memory usage	分配的内存使用率
9	unmapped page cache memory usage	未映射页缓存使用率
10	total page cache memory usage	总页缓存使用率
11	maximum memory usage	最大内存使用率
12	mean disk I/O time	平均 I/O 时间
13	mean local disk space used	平均磁盘使用率
14	maximum CPU usage	最大 cpu 使用率
15	maximum disk IO time	最大磁盘 I/O 时间
16	cycles per instruction (CPI)	每条指令的时钟周期数

4.6.1 任务抽取

根据谷歌云服务集群 trace 文档[3]，一个作业内的任务可能重复提交，但提交之后还会被分配原始的任务 ID 号，所以为了简便，我们只考虑某个作业中的第一任务。而由于原始 trace 数据过于庞大，共计 41G，包含 29 天，共计 2500 万的任务执行信息，所以我们参考文献[19]，使用前一周共计 7 天的任务数据进行研究。具体的，针对 Task events table 中的某条记录，解析其字段之后判断时间戳是否在一周之内；对于符合条件的记录，继续判断事件类型，如果是 EVICT(2)、FAIL(3)、KILL(5)、LOST(6) 四种状态中的一种，则将该任务标记为”Failed”，代表执行失败，如果是 FINISH(4) 状态则标记为“Finished”代表执行成功。并将通过作业 ID 以及任务索引的字符串的拼接结果作为 key，任务的真实状态作为 value 保存在一个字典中记为“任务元信息字典”。当得到抽取到的任务 ID 以及对应的任务终止事件之后，就需要在 Resource usage table 中提取对应的时序数据，作为预测模型的输入特征。具体的，本文针对 Resource usage table 中的符合时间范围（第一周内）的每条记录生成“任务时序字典”，然后“任务元信息字典”挑选出符合终止状态要求的任务，共计 2.443339×10^6 个任务，其时序数据长度的分布如图26所示。可以看到主要时长是集中在 0-200 个测量周期内，每个测量周期为 5 分钟，也即 $200 \times 5 = 1000$ 分钟内。统计一周之内任务终止状态分布，如图27所示，总体上执行成功的任务接近未成功执行的任务的 2 倍。

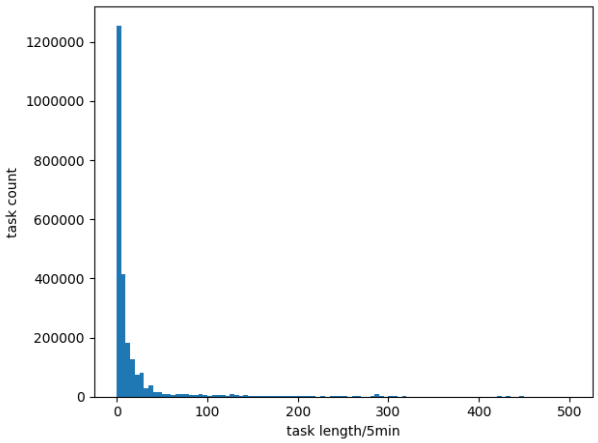


图 26 一周内任务执行时长分布图

4.6.2 预测时机的选取

预测时机的选取直接决定了输入的监测指标序列的长度，从而可能导致选取不同时机预测时的难度存在差别。为了研究预测时机的不同对本模型性能的影响，本文分别

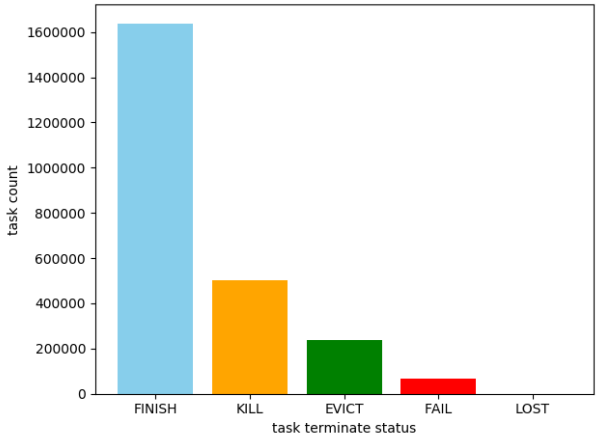


图 27　一周内任务执行终止状态分布图

在早期和晚期两个时间点进行两套完整的实验。

早期预测：越早进行预测，终止失败任务得到的收益越大。为了研究预测时机和预测性能之间的关系，本文设定了一个预测时间 $T_0 = 50$ 个测量周期，在收集满 50 个测量时间的数据之后进行一次预测。统计符合条件的任务，即执行时长大于设定的阈值 T_0 的任务，其终止状态分布如图28。可以看到执行成功的任务所占的比例相比不进行长度过滤时变低了，这说明任务执行时长越长，越可能失败。本文使用经过阈值 T_0 过滤后的任务数据作为早期预测实验的数据集。进行失败标签合并后的信息如表7所示。

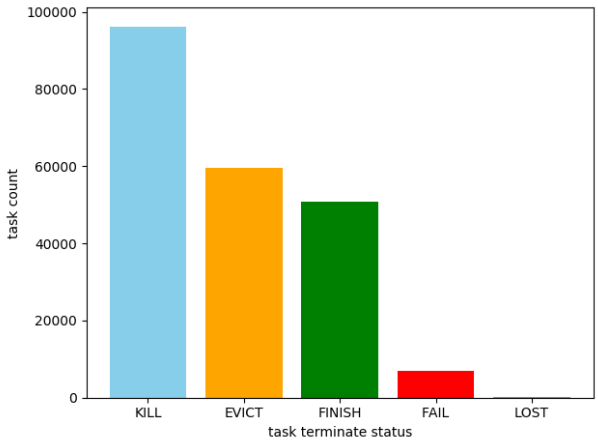


图 28　执行时长超过 50 个测量周期的任务执行结果分布图

表 7　早期预测数据集信息表

任务类别	个数
失败任务	162635
成功任务	50854
总计	213489

晚期预测：由于对于执行时间越长的任务，进行任务失败预测的收益越大，所以本文需要设定一个较长的预测时间，记为 T_1 个测量周期，在收集满 T_1 个测量周期的数据之后进行另一次预测。统计执行时长超过 50 个测量周期的任务的长度分布，做出长度分位数图，如图29所示。横轴是百分比，纵轴是任务数量，折线上的某点 (x, y) 代表执行时长在所有长度超过 50 个测量周期的任务的执行时长中处于比例为 x 的那个任务的执行时长为 y。可以看到中位数位置的执行时长大约为 150。所以本文选择设置晚期预测时间 $T_1 = 128$，以覆盖大部分经过 T_0 过滤后的任务。统计符合条件的任务，即长度大于阈值 T_1 的任务。其终止状态分布如图30，可以看到成功任务所占的比例进一步变少。本文使用经过阈值 T_1 过滤后的任务数据作为晚期预测实验的数据集。进行失败标签合并后，成功任务和失败任务的个数如表8所示。

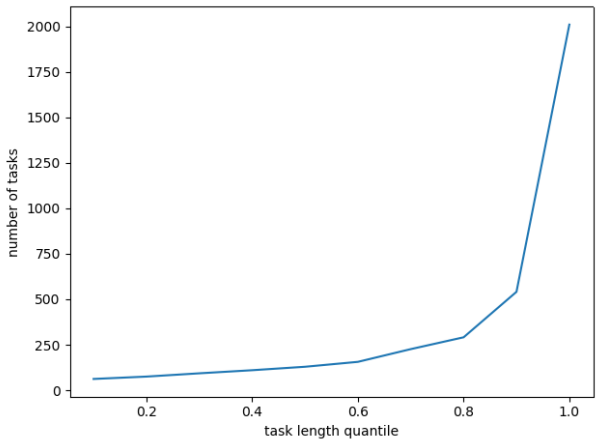


图 29 执行时长超过 50 个测量周期的任务执行时长分位数图

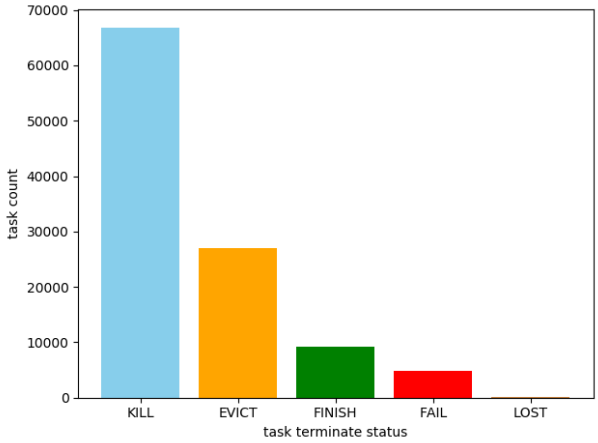


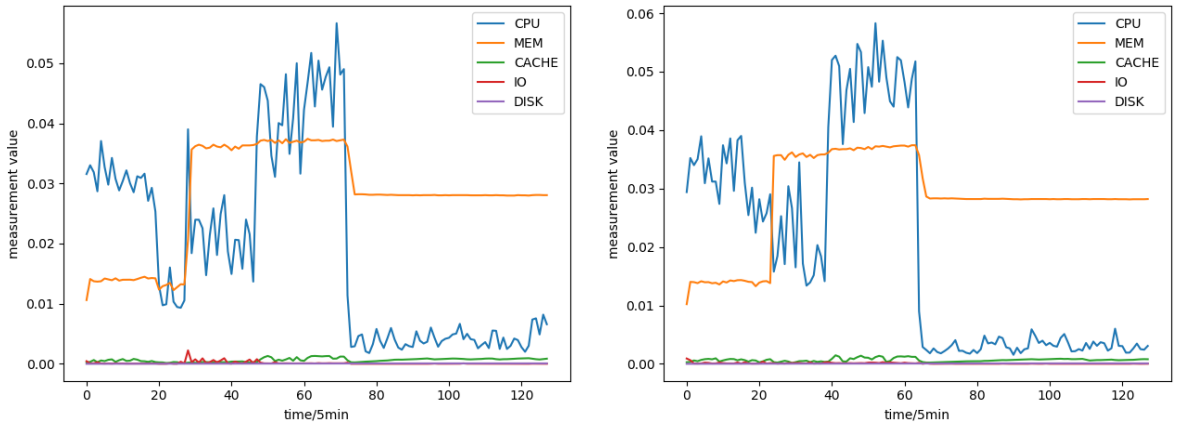
图 30 执行时长超过 128 个测量周期的任务执行结果分布图

图31(a) 和图31(b) 分别展示了执行失败和执行成功的任务的 5 个监测指标的时间序列，本文的目的是采用时间序列分类的方法对这些监测指标组成的时间序列进行分类，

表 8　晚期预测数据集信息表

任务类别	个数
失败任务	98802
成功任务	9204
总计	108006

从而预测出任务的最终执行状态。对于早期预测数据集和晚期预测数据集，本文均采用 8:1:1 的比例划分为训练集、验证集和测试集进行实验。



(a) 失败任务示例图　　(b) 成功任务示例图

图 31　任务监测指标示例图

4.6.3　实验环境说明

实验环境见表9和表10所示。Keras[45] 是一个由 python 编写的开源神经网络库，可以作为 Tensorflow[46] 的高阶应用程序接口，本文使用 Keras 进行模型的搭建。模型的评价指标方面使用 Scikit-learn[47] 中提供的函数进行计算。

表 9　任务失败预测实验环境硬件配置表

硬件	详细描述
CPU	(英特尔)Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
内存	8.00 GB
显卡	NVIDIA GeForce GTX 1060 with Max-Q Design

表 10　任务失败预测实验环境软件配置表

软件	详细描述
操作系统	ubuntu 16.04 stl (64 位)
Tensorflow-gpu	1.10.0
Keras	2.2.4
Scikit-learn	0.19.1

4.6.4 训练方法

本文使用 minibatch 随机梯度下降和 Adam[49] 优化器来训练模型。learning rate 设置为 0.0001，batchsize 设置为 128。参数通过标准反向传播来学习，使用二值交叉熵作为损失函数:

$$loss = -\frac{1}{N}\sum_{n} y_n \log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n) \tag{4.6}$$

其中 N 为样本总数，y_n 为某样本对应的真实标签，$\hat{y}_n$ 为该样本对应的模型输出。为了防止过拟合，采用早停策略：如果验证集上的损失超过 10 个轮没有下降的话就停止训练，并设置最多训练 20 轮。同时为了加快收敛采用学习率衰减策略：当超过 2 轮验证集损失没有下降的话对 Adam 优化器的全局学习率进行 0.5 倍的衰减。

4.6.5 评价指标

本文主要使用 AUC[56] 来进行模型的评估，AUC 是基于动态阈值调整的一种评价指标，其定义为 ROC 曲线下面积，表示预测的正例排在负例前面的概率。而 ROC 是根据动态调整阈值，计算 FPR（假正例率）和 TPR（真正利率）绘制而成，二者计算如公式4.7所示。TPR 衡量正确预测的失败任务占所有失败任务的比例，FPR 衡量执行成功的任务被错误预测为失败任务的比例。Chen[13] 的研究给出了 TPR 和 FPR 的试验结果，本文进一步给出 AUC 的结果。

$$TPR = \frac{TP}{TP+FN} \tag{4.7}$$

$$FPR = \frac{FP}{FP+TN} \tag{4.8}$$

参考 Islam[14] 的研究，本文也报告了各模型在 F1 值上的表现，F1 值是由 Precision 值和 Recall 值计算而来，precision 和 recall 分别代表查准率和查全率，这三个指标在上一章有详细描述此处不再赘述。TPR，FPR，F1，Precision，Recall 以及传统的准确率 (Acc) 这六个指标都是按照预测输出的概率值是否大于 0.5 来判定，是基于固定阈值的评价指标，即:

$$\text{label-predicted} = \begin{cases} 1, \hat{y} > 0.5 \\ 0, \hat{y} < 0.5 \end{cases} \tag{4.9}$$

更优秀的模型能使 AUC、F1、Precision、Recall、TPR、Acc 尽可能的高，而 FPR 尽可能的低。

4.6.6　对比算法

本实验需要对比目前主流的任务失败预测算法，以及先进的多变量时间序列分类算法，选用的对比算法如下所示：

RNN：此模型来自 Chen[13] 于 2014 年的研究，方法为将每一时刻的所有检测指标视为一个特征向量 $x_t = (cpu_t, ram_t, io_t, disk_t, cache_t)$，作为该时刻的 RNN 的输入。目的是利用 RNN 自身的循环结构捕捉不同时间步特征项间的长期依赖关系。而后使用最后一个时间步的 RNN 隐层状态，作为融合了所有时间步信息的特征表示 $h_t = RNN(x_t) \in R^d$，最后送入全连接网络进行分类。

MetaLSTM：此模型来自 Islam[14] 于 2017 年的研究，在 Chen 的基础之上，使用长短期记忆网络 LSTM 替换 RNN，然后对 LSTM 的每个时间步的输出进行全局平均池化，提取所有时间步范围内的时序特征。同时在输入特征部分加入了任务的元信息：用户名、所属作业 ID，任务索引、任务执行时间、重复提交次数、优先级、调度类别。然而针对用户名这种特征，由于没有对应的历史数据，所以无法处理新用户问。同时，虽然有些用户提交的任务确实更有可能执行失败，但如果使用该特征得到的预测结果对任务进行提前终止，会导致某些用户的任务有较高的误杀率。对于任务所属的作业 ID，任务索引号这两个属性，前者会遇到和新用户一样的问题，对于后者不同的作业可能被划分出数量不同的任务，而任务被划分时的顺序可能是随机的，这个索引号并不包含任何任务本身的信息。对于作业执行时间、重复提交次数这两个属性是基于全局信息的，但真实情况下在任务初次提交和未执行完毕时往往无法获得。所以本文只采用了优先级、调度类别这两个任务的元信息，构建该基线模型。

ResNet：根据 2019 年 Fawaz 等人[21] 的对比，此模型为目前最优的多变时间序列分类模型。此模型来自 Wang[8] 等人于 2017 年的研究，它是一种基于 residual network 结构的深度卷积网络，基本结构借鉴了 ResNet 中的残差单元，堆叠了 3 个卷积模块，在每个模块间加入 short cut，以保证深度网络中的梯度流通。每个模块包含 3 个等长卷积操作，每次卷积之后加入 Batch Nornalization 层以防止梯度消失，加速模型收敛。最后进行全局平均值化，获得一个融合了所有时间步信息的特征表示，并送入全连接网络进行分类。相比本文提出的模型，该模型采用一维卷积同时从所有通道中提取时序特征，而本文采用分离式的结构分别用一维卷积提取各通道内的特征。

MCDCNN：此模型来自 Zheng[20] 等人于 2014 年的研究，它采用独立的卷积核处理每一条时间序列。每一条时间序列都会通过两层的卷积模块，每个模块包含一个一维卷积层，以及一个步长为 2 的最大池化层，每经过一个模块时间序列的长度就缩减了一半，以此捕获更长范围的时序依赖关系。所有第二个模块的输出被拼接在一起送入一个全连接网络进行最后的分类。相比本文提出的模型，该模型没有考虑到不同时间序列间的交互性特征，文本提出的通道间多头自注意力模块即是对这一点的改进。

SplitCNN：为了证明本文提出的通道注意力卷积模型 CMSA-CNN 中多头自注意力机制的有效性，我们去掉了 CMSA-CNN 模型中的通道间多头自注意力模块，仅使用通道内一维卷积模块进行各时间序列内部特征的提取。

CMSA-i：该模型是本文提出的通道自注意力卷积模型中的后半部分，也即只使用多头自注意力机制进行通道间特征的提取，而不进行通道内特征提取的一种模型。i 代表使用的自注意力 head 的数量。

CMSA-CNN-i：该模型是本文提出的通道自注意力卷积模型，i 代表使用的自注意力 head 的数量。

4.6.7 实验结果与结论

分别针对上述 7 种模型，进行早期预测，即预测时机选取为 $T_0 = 50$ 个测量周期时；以及晚期预测，即预测时机时选取为 $T_1 = 128$ 个测量周期时。实验结果如表11和表12所示。可以看出在 AUC 以及 F1 指标上本文提出的模型均优于所有基线模型。

表 11 在 50 个测量周期结束时进行任务失败预测的实验结果表

Model	AUC	TPR	FPR	F1	Precision	Recall	Acc
RNN	0.8815	0.8140	0.2074	0.8665	0.9262	0.8140	0.8089
MetaLSTM	0.9663	0.8907	0.0505	0.9344	0.9826	0.8907	0.9047
ResNet	0.9771	0.9220	0.0488	0.9519	0.9837	0.9220	0.9290
MCDCNN	0.9658	0.9014	0.0635	0.9384	0.9784	0.9014	0.9098
SplitCNN	0.9649	0.9136	0.0802	0.9425	0.9733	0.9136	0.9150
CMSA-1	0.9232	0.8665	0.1160	0.9108	0.9598	0.8665	0.8707
CMSA-2	0.9394	0.8762	0.1064	0.9178	0.9634	0.8762	0.8804
CMSA-4	0.9573	0.9104	0.1072	0.9366	0.9645	0.9104	0.9062
CMSA-8	0.9635	0.9106	0.0773	0.9413	0.9741	0.9106	0.9135
CMSA-CNN-1	0.9782	0.9169	0.0348	0.9513	0.9883	0.9169	0.9284
CMSA-CNN-2	0.9784	0.9136	**0.0317**	0.9499	**0.9893**	0.9136	0.9267
CMSA-CNN-4	0.9777	0.9074	**0.0317**	0.9465	0.9892	0.9074	0.9219
CMSA-CNN-8	**0.9801**	**0.9310**	0.0495	**0.9566**	0.9836	**0.931**	**0.9356**

表 12 在 128 个测量周期结束时进行任务失败预测的实验结果表

Model	AUC	TPR	FPR	F1	Precision	Recall	Acc
RNN	0.9013	0.8796	0.1281	0.9300	0.9866	0.8796	0.8789
MetaLSTM	0.9840	0.9432	**0.0098**	0.9703	**0.9990**	0.9432	0.9472
ResNet	0.9904	0.9687	0.0358	0.9824	0.9966	0.9687	0.9683
MCDCNN	0.9833	0.9641	0.0641	0.9787	0.9938	0.9641	0.9617
SplitCNN	0.9791	0.9724	0.1314	0.9799	0.9876	0.9724	0.9635
CMSA-1	0.9687	0.9105	0.0836	0.9493	0.9915	0.9105	0.9110
CMSA-2	0.9777	0.9307	0.0445	0.9620	0.9956	0.9307	0.9328
CMSA-4	0.9862	0.9554	0.0347	0.9756	0.9966	0.9554	0.9562
CMSA-8	0.9880	0.9662	0.0413	0.9809	0.9960	0.9662	0.9656
CMSA-CNN-1	0.9917	0.9557	0.0152	0.9766	0.9985	0.9557	0.9582
CMSA-CNN-2	**0.9918**	0.9738	0.0358	**0.9851**	0.9966	0.9738	**0.9730**
CMSA-CNN-4	0.9871	**0.9812**	0.1770	0.9823	0.9835	**0.9812**	0.9677
CMSA-CNN-8	0.9862	0.9522	0.0304	0.9741	0.9970	0.9522	0.9537

1) 和经典任务失败预测算法的对比： Chen 的 RNN 模型在所有的对比模型中的表现最差。如图32所示，在 AUC 指标上，本文提出的模型分别在早期和晚期预测上比 RNN 高 9.86、9.05 个百分点，平均提高 9.5 个百分点。当加入了任务的元信息作为特征，并采用在提取长期依赖特征方面更优秀的 LSTM 以及融合全局特征的 Global average pooling 层之后，预测效果有了显著的提高：对于 AUC 指标，MetaLSTM 模型相较 RNN 分别在早期预测和晚期预测实验中提升了 8.5 和 8.2 个百分点，但仍然和本文提出的单纯基于时序信息的模型 CMSA-CNN 分别模型差了 1.38 和 0.78 个百分点，平均差了 1.1 个百分点，如表11以及表12中 MetaLSTM 数据所示。这说明虽然元信息能够辅助预测，但是这类基于循环神经网络统一处理每一个时间步输入的模型，并没有充分提取到各个监测指标组成的多变量时间序列的关键性特征。下面以 Islam 采用的 LSTM 为例进行分析：基于 LSTM 模型在每一个时刻同时输入所有变量，由于不同变量在相同的时刻受到过去的影响程度可能不同，而 LSTM 在每一步都是通过将输入向量乘以矩阵的形式计算门控单元的输出，公式4.10展示了输入 $x_t = (cpu_t, ram_t)$ 时的遗忘门的输出。

$$\begin{aligned} f_t &= \sigma([cpu_t, ram_t] \times \begin{bmatrix} w_{11}, w_{12} \\ \\ w_{21}, w_{22} \end{bmatrix}) \\ &= \sigma([w_{11}cpu_t + w_{21}ram_t, w_{12}cpu_t + w_{22}ram_t]) \\ &= (f_1, f_2) \end{aligned} \tag{4.10}$$

遗忘门输出的两个维度 f_1, f_2 分别同时受到该时刻输入的两个维度 cpu_t, ram_t 的影响，所以在控制不同维度的信息流动时会受到其他维度的影响，可能不利于提取不同通道内的时序特征。

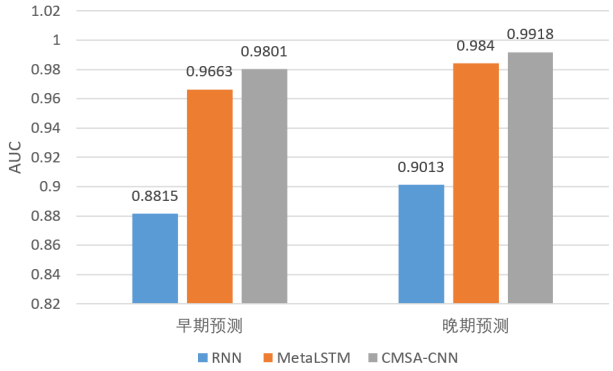


图 32 与经典任务失败预测算法的 AUC 对比图

2）和先进的多变量时间序列分类算法对比：可以看到 ResNet 仅次于本文提出的模型是表现第二好的模型。如图34所示，相比本文的模型，在 AUC 指标上，ResNet 在早期和晚期预测上分别差了 0.30 和 0.14 个百分点，平均相差 0.2 个百分点。主要原因在于 ResNet 试图同时处理通道间和通道内特征，所以每一个通道的卷积核的时间跨度是相同的，这就造成各自通道内在时间维度变化的差异性丢失，如图33所示。

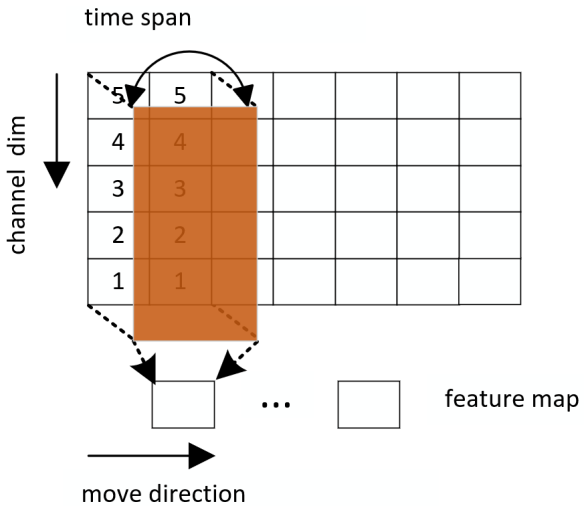


图 33 ResNet 卷积核局限性示意图

然而 ResNet 通过堆叠不同尺度的卷积结构，采用较深的网络层数等策略弥补了这一弱点，相对其他模型仍有优势。对于 MCDCNN 模型，相比本文的模型在 AUC 指标上，早期和晚期预测实验中分别差了 1.43 和 0.85 个百分点，平均差了 1.1 个百分点。该模型也采用通道分离式的结构，使用一维卷积和局部最大池化提取各个通道内的时序特征，但是没有考虑通道之间存在的相关性，比如一个执行成功的任务在运行时可能发

如下的现象：某段时间内 I/O 访问较频繁，之后的一段时间 CPU 有较高的占用率。这种不同变量间的交互性特征无法通过单纯的分离式卷积捕捉到。

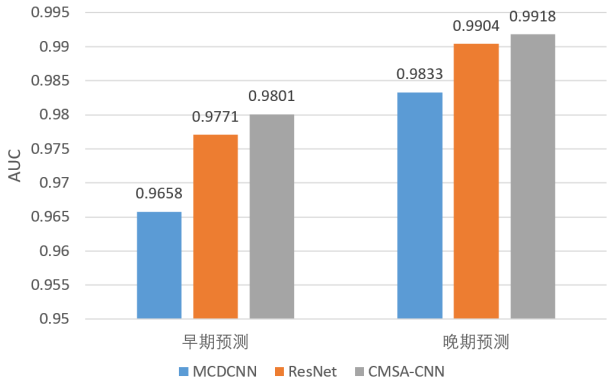


图 34　与先进的多变量时间序列分类算法的 AUC 对比图

3）模型简化分析：如图35所示，当本文提出的 CMSA-CNN 模型去掉前端的通道内一维卷积模块变为 CSMA 之后，AUC 下降较多，这主要是由于本文的采用的通道间多头自注意力模块依赖于通道分离式卷积操作的特征提取能力，如果没有较好的提取出每个通过各自的时序变化特征，直接在通道间计算相关性特征的话，单纯的靠 Attention 机制，无法有效的拟合数据。同样的由于当 CMSA-CNN 模型去掉后端的通道间自注意力模块变为 SplitCNN 之后在 AUC 上也有很大下降。由于 SplitCNN 相比 MCDCNN 在结构上少了每层卷积之后的局部最大池化，为了辅助通道间特征的提取而选择通过最后的全局平均池化来提取全局视角下的通道内特征，所以在捕捉局部的显著时序特征时稍弱于 MCDCNN。此外采用全局平均池化也是为了比局部池化更进一步的缩减参数规模，避免 CMSA-CNN 在增加注意力模块之后参数过多。

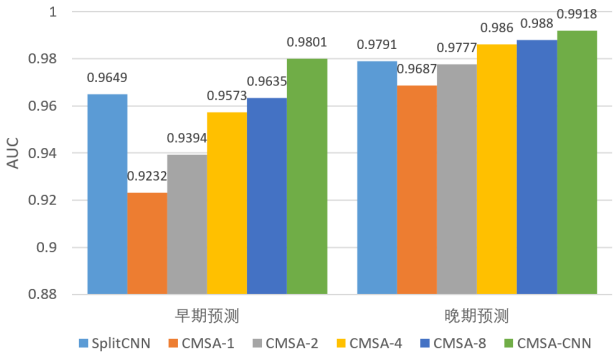


图 35　与简化模型的 AUC 对比图

4）预测时机的对比：对于预测时机 T 的不同造成的 AUC 变化，如图36所示。相比在 T_0 时刻进行的早期预测，可以看到不管是本文提出的 CMSA-CNN 还是 ResNet 等其他基线模型，在 AUC 指标上都低于在 T_1 时刻进行的的晚期预测，这说明早期预测确实会因为无法收集足够的监测指标数据，从而导致预测性能的降低。

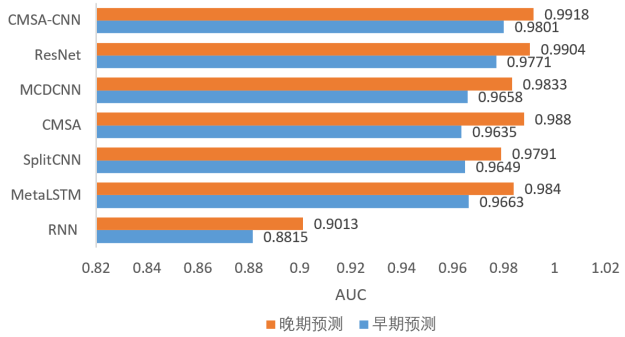
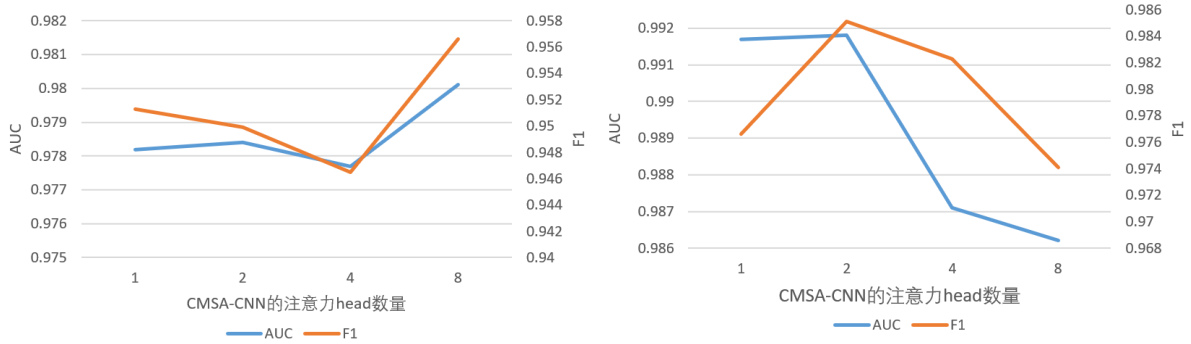


图 36 预测时机对比图

5）注意力 Head 数量研究：最后本文还对多头注意力机制中的 head 的数量进行了研究。如图37(a) 所示，可以看到在早期预测时当 head 数增加时 AUC 和 F1 两个指标都有提升的趋势，说明增加 attention head 的数量，将各通道内特征被投影到更多的空间中，保证了通道间的交互性特征能够被充分提取，一定程度上提升了模型性能。如图37(b) 所示，在晚期预测时 AUC 和 F1 基本在 head 数量为 2 时达到最优，而后效果下降，这可能是由于晚期预测已经收集足够长的时序监测数据，不需要增加 head 数量就能取得不错的效果，而当 head 数量增加之后模型变得复杂，导致训练变得困难造成的。



(a) 早期预测　　(b) 晚期预测

图 37 早期和晚期预测时 AUC 和 F1 随 head 数量变化图

4.7 本章小结

本章针对现有任务失败预测模型以及目前较先进的多变量时间序列分类模型存在的问题，即：现有模型无法同时考虑，不同通道的变量内部的特征具有独立性，以及跨通道的互特具有交互性。提出了基于通道自注意力机制的卷积网络，首先提取通各个道内的特征，然后通过多头自注意力提取跨通道的特征，在集群 trace 数据集上进行的实验中优于所有基线模型。

第五章　基于长短期记忆网络的存储系统 I/O 负载短期时序预测算法

5.1　研究背景

随着存储数据负载量和集群应用程序多样性的不断增加，存储系的统性能成为许多数据密集型应用程序的关键瓶颈[57]。简单地扩展集群无法彻底解决存储瓶颈问题。然而，系统运行过程中产生的大量 I/O trace 数据为我们解决系统性能优化问题提供了新的视角，即“数据驱动”的视角。存储系统的 trace 数据隐含了系统的配置信息，包含了存储系统的空间和时间维度的运行状态。建模系统 I/O 工作负载的特征，然后提前预测工作负载的变化是制定最佳配置策略、执行细粒度负载均衡和提高系统效率的关键。例如，热点数据通常会导致某些服务器处于过载状态，而另一些服务器则处于欠载状态，这样既导致系统整体性能降低，又浪费了硬件资源。基于预测算法，制定合适的策略将数据从这两类服务器间进行迁移无疑会提升集群负载的均衡程度。然而，目前关于 trace 的研究大部分关注计算 trace，I/O trace 方面则较少，尤其是基于开源 I/O trace 的时序预测模型的研究则更加少见。因此如何理解和分析系统产生的 I/O trace 数据，如何建模 I/O trace 的时序特征，如何基于深度学习方法进行工作负载的预测，成为了亟待解决的问题。本文针对上述问题研究基于长短期记忆网络的存储系统 I/O 负载短期时序预测算法。

本章节的贡献如下：

1）设计实现了一种 I/O 工作负载预测算法，它包括工作负载预处理、基于长短时记忆网络的时间序列预测和数据后处理。

2）在实际的存储负载 trace 上进行了实验，和 3 种典型预测方法相比在 MAPE 上有至少 1.1% 的提升。

5.2　问题定义

本文研究的是存储系统的 I/O 访问负载，所以本文的“负载”定义为单位时间内的访问数据量。所以负载预测问题可以转化为单变量的时间序列预测问题，形式化定义为：给定一个存储负载时间序列 $\mathbf{x} = (x_1, \ldots x_i, \ldots x_w)$ 其中工作负载定义为单位时间内的

访问数据量，时间序列预测目标是学习函数 $y = f(x)$,y = $\left(x'_{w+1} \ldots x'_{w+j} \ldots x'_{w+T}\right)$ 其中，x_i 为第 i 时刻的工作负载观测值，x'_i 为第 i 时刻的预测数据，w 为历史观测窗口长度，即使用多少历史数据点来预测未来数据。T 是预测窗口长度，也就是我们想要预测的未来数据点的个数。本文只关注单步预测，因此预测窗口长度 T 被设置为 1。

5.3 相关性分析

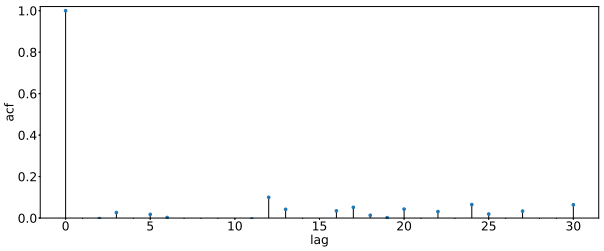
存储系统的 I/O 负载是由外部应用的访问动作产生的，当前时刻的外部访问活动可能受具体访问流程的约束而呈现出一定的顺序性，反应在被访问的存储系统中，其 I/O 负载很可能呈现出一定的时序依赖关系，即当前时刻的 I/O 负载和过去一段时间的负载密切相关。本文使用自相关函数来描述这种时序上的相关关系。自相关描述的是信号与自身延迟的相关性，可以很好地说明 I/O 访问负载中存在的长期依赖关系，其定义为:

$$\rho(t,s) = \frac{E\left(X_t - u_t\right)\left(X_s - u_s\right)}{\sqrt{DX_t \cdot DX_s}} \tag{5.1}$$

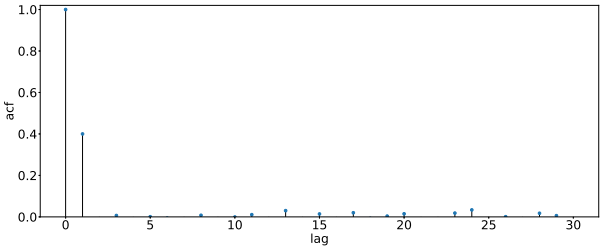
图38显示了三个不同的时间序列的自相关函数。在图38(a) 中，除了滞后时间为 0 时刻的值外，其余值均接近于零，表明相邻时间之间没有明显的相关性（对应横坐标为 1 的位置），滞后时间大于 1 的时刻之间也不存在较大的相关性（对应横坐标为大于 1 的位置），所以其信号更接近白噪声，这是很难预测的。在图38(b) 中，除了滞后时间为 0 和 1 的值外，其余值都接近于零，这意味着相邻时间之间存在很强的相关性（对应横坐标为 1 的位置有较大的取值），而远距离的两时刻之间不存在很强的相关性，此时只使用前一时刻的负载值就可预测出下一时刻的负载。与前两幅图不同的是图38(c) 中值普遍较大，说明相邻时间与较大滞后时刻之间存都在相关性，这就要求在构建预测模型的输入时，应该包含更远距离的负载特征。

5.4 窗口归一化方法

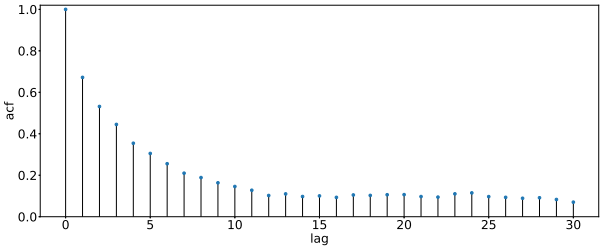
由于负载时间序列存在着很强的波动性，如果直接将原始的负载值输入模型中，在使用一个观测窗口中的数据进行预测时，模型必须学习出位于此负载绝对值附近的时序变化模式，然而处于此绝对值附近的有效点的数量在整个数据集中可能很少，导致深度模型训练时在每个时间点位置上都处于样本量过少的处境中，为了消除这一问题，本



(a) 无明显线性相关性



(b) 相邻时间位置有明显线性相关性



(c) 在大跨度时间范围有明显线性相关性

图 38 相关性分析图

文采用了一种基于窗口的归一化方法：即使用每个观测窗口中第一个时间位置的值作为“参考点”，而后将这个时间窗口内的其他值都转化为相对于这个“参考点”的百分比增量值。从而可以让模型直接学习每个观测窗口内的时序变化模式，简化了原始的问题。计算公式为：

$$x_i = \frac{x_i}{x_{t-w}} - 1, i \in [t-w, t] \tag{5.2}$$

5.5 算法框架

由上述相关性分析可知，I/O 访问负载在时间维度上存在较强的前后依关系，长短期记忆网络（LSTM）是一种适合建模远距离序列依赖关系的神经网络模型，所以，本文基于深度学习模型中的长短期记忆网 (LSTM) 构建一种存储系统 I/O 负载预测模型，

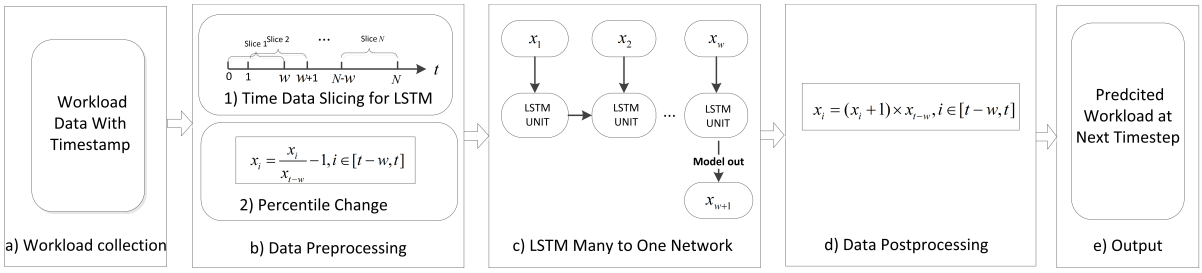


图 39 基于 LSTM 的 I/O 负载预测算法框架图

包括数据预处理、时间序列预测和数据后处理阶段。整体流程如图39所示。

预测算法采用连续滚动预测模式，输入历史观测窗口中的负载数据，经过预处理模块，送入预测模块，得到预测结果，送入后处理模块，输出下一时刻的真实预测负载值，此次预测过程结束。随后收集此次预测对应的真实负载值，历史观测窗口向前移动，窗口前边沿对应的负载加入观测窗口，原始后边沿数据移除窗口，开始下一次预测，预测过程如算法2所示。

算法 2: 连续负载预测算法

Input: storage workload time series $\mathbf{x} = (x_1, \ldots x_i, \ldots x_w)$
Output: the predicted workload $\mathbf{y} = (x'_{w+1}, \ldots x'_{w+j}, \ldots x'_{w+T})$

for $i \leftarrow 1; i <= T; i \leftarrow i+1$ **do**
  $\mathbf{x} \leftarrow$ preprocess input data $\mathbf{x}$;
  $x'_{w+i} \leftarrow$ workload prediction using preprocessed data $\mathbf{x}$;
  $x'_{w+i} \leftarrow$ postprocess the predicted data x'_{w+i};
  $x_{w+i} \leftarrow$ collect the real workload at the predicted time step;
  $\mathbf{x} \leftarrow (x_{i+1}, \ldots x_i, \ldots x_{w+i})$
end
$\mathbf{y} \leftarrow (x'_{w+1}, \ldots x'_{w+j}, \ldots x'_{w+T})$
return y

预处理模块使用固定值的滑动窗口对时序数据进行切片，然后进行放缩变换得到待输入数据。因为我们的目的是学习一个从输入到输出的函数，即：$x_{t+1} = f(x_{t-w} \ldots x_t)$，所以我们使用一个固定长度为 $T+1$ 的时间窗进行时序数据的切片以获得模型训练所需要的输入输出数据，即：

$$\{< (x_{t-w} \ldots x_t), x_{t+1} > | t \in [w+1, N-1]\} \tag{5.3}$$

然而 I/O 负载具有很强烈的波动性，不同输入时间窗内的负载值有很大的差距，这就导致模型收敛困，为了消除不同时间窗内 IO 负载数据的值的差距，使网络更容易学习，本文使用一个基于窗口的归一化方式，如公式5.4所示，使得同一个窗口内的数据

都以第一个数据为参考，将预测幅值转化为预测增长率，从而减缓训练数据的波动。

$$x_i = \frac{x_i}{x_{t-w}} - 1, i \in [t-w, t] \tag{5.4}$$

由于 I/O 负载预测是基于局部性原理而进行的，所以距离预测位置最近的点对要预测的值有着更加重要的影响，所以本文采用多对一的长短期记忆网络结构，使用最后一个时间步的长短期网络的输出作为整个观测窗口的特征，这样可以利用长短期记忆网络的“有偏特性”让提取到的特征中包含更多的“最近”的时序信息。从而让模型更加关注于最近点。具体的，多对一的网络结构即输入和输出上的多对一，该模型尾部使用多层感知机（MLP）将特征向量转化为预测输出值。而后处理过程采用和预处理相反的放缩操作复原数据，最终得到预测值。

5.6 实验验证

5.6.1 实验环境说明

实验环境见表2和表3所示。Keras[45] 是一个由 python 编写的开源神经网络库，可以作为 Tensorflow[46] 的高阶应用程序接口，本文使用 Keras 进行模型的搭建。使用 python 的统计包 Statsmodels[58] 进行时序相关性分析。

表 13 I/O 负载预测实验环境硬件配置表

硬件	详细描述
CPU	(英特尔)Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
内存	8.00 GB
显卡	NVIDIA GeForce GTX 1060 with Max-Q Design

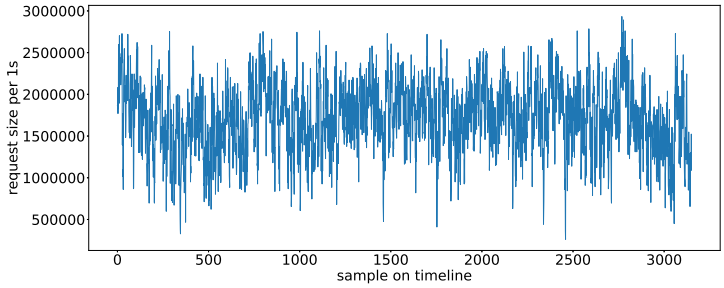
表 14 I/O 负载预测实验环境软件配置表

软件	详细描述
操作系统	Microsoft Windows 10 家庭中文版 (64 位)
Ttensorflow-gpu	1.10.0
Keras	2.2.4
Scikit-learn	0.19.1
Statsmodels	0.8.0

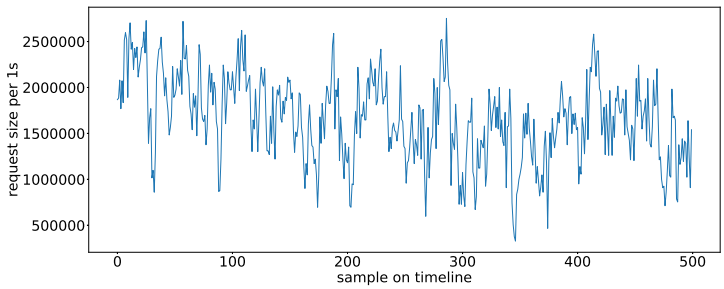
5.6.2 数据集

本文主要研究了一个开源存储系统 trace WebSearch[59]，WebSearch 是由马萨诸塞大学收集并发布的一个服务于搜索引擎的存储系统 I/O trace，它主要记录了每次外部 I/O

访问的时间戳以及对应的读写字节数。在本实验中，我们对 size 字段进行建模，该字段描述了作为每次 I/O 操作访问的字节数。值得注意的是 size 字段既包括读取操作也包括写入操作对应的请求数据量。根据 size 字段和 timestamp 字段，可以建立原始的工作负载时间序列，由于每次访问发生的时间间隔不固定，所以此序列是不等时序列。因此，本文以每秒为标准时间间隔加和 size 字段值，生成以固定时间间隔的 I/O 负载单变量时间序列，作为最终的研究序列。图40展示了此 I/O 负载时间序列。最后，得到了由前



(a) 全部数据



(b) 前 500 个数据

图 40 WebSearch 负载时间序列数据集展示图

2488 个数据点组成的训练集，由接下来的 312 个数据点组成的验证集和由最后 351 个数据点组成的测试集。由于统计特征反映了负载数据的基础特性，因此本文计算了负载数据集的统计特征。如表15所示，可以看到工作负载数据集的平均值是 1711868.02，标准差是 415512.83，最大值是 2932736.00，最小值是 262144.00，说明此负载序列具有非常强的抖动性。

5.6.3 训练方法

本文使用 minibatch 随机梯度下降和 Adam[49] 优化器来训练模型。learning rate 设置为 0.001，batchsize 设置为 64。参数通过标准反向传播来学习，使用均方误差作为目标

表 15 WebSearch 数据集统计特征表

Type	Value
The mean value	1711868.02
The standard deviation	415512.83
The max value	2932736.00
The min value	262144.00

函数，计算公式为：

$$\begin{aligned} j_i =& mse\left(y_i, \hat{y}_i\right) = \left(y_i - \hat{y}_i\right)^2 \\ J(\boldsymbol{\theta}) =& \frac{1}{N}\sum_{i=1}^{N} j_i + \lambda \sum_{\mathbf{k}} \theta_k^2 \end{aligned} \tag{5.5}$$

其中 j_i 为的单个样本产生的损失，y_i 为真实的待预测的负载值，$\hat{y}_i$ 为预测出来的下一时刻负载值。J 为所有样本的损失，N，λ 为正则化项系数，此处采用的是 $L2$ 正则项。

5.6.4 评价指标

为了验证本方法的有效性，我们考虑了三个不同的评价指标，即均方根误差 (RMSE)、平均绝对误差 (MAE) 和平均绝对百分误差 (MAPE)，其中 RMSE 和 MAE 衡量了预测效果的绝对误差，MAPE 则衡量了预测效果的相对误差：

$$\begin{aligned} RMSE &= \sqrt{\frac{1}{N}\left(\sum_i \left(y_i - \hat{y}_i\right)^2\right)} \\ MAE &= \frac{1}{N}\left(\sum_i \left|y_i - \hat{y}_i\right|\right) \\ MAPE &= \frac{1}{N}\left(\sum_i \frac{\left|y_i - y_i\right|}{y_i}\right) \end{aligned} \tag{5.6}$$

5.6.5 对比算法

对于时间序列预测，ARIMA，SVR 和 SimpleRNN 是三种经典方法。因此，为了验证本文提出的方法，我们选择 ARIMA，SVR 和 SimpleRNN 作为对比算法。对于基于 ARIMA 的工作负载预测模型，本文使用 statsmodels 包实现。我们首先使用 Augmented Dickey-Fuller test 对工作负载进行静态分析。然后通过自相关和偏自相关函数来确定参数 p,i,q 三个参数的近似范围。接下来，使用 Akaike 信息准则（AIC）用于查找三个参数的准确值。当我们根据训练集获得正确的 p,i,q 值后，开始拟合 ARIMA 模型然后预测测试集中的未来值。具体的，每做一次预测，就将预测数据的真实值包含到到已知数

据集中，扩展训练数据窗口。然后根据扩展的训练数据拟合新的 ARIMA 模型，直到测试集中没有要预测的数据为止。这样一共拟合的的模型个数和测试集大小相同，每个模型只负责一个点的预测，从而得到整个测试集的预测值。对于 SVR 模型，本文使用径向基函数作为核函数，历史观测窗口 w 在 [5, 10, 15, 20, 25,30,35,40,45,50,55,60] 范围进行搜索，由于此处设置的历史观测窗口搜索范围较大，为了便于确定最优参数组合，本文采用 scikit-learn 中推荐的设置，将径向基函数的参数 $gamma$ 设置为历史观测窗口的倒数，即 $gamma = \frac{1}{w}$，惩罚项系数 C 在 [0.5,1.0,1.5,2.0,2.5,3.0] 范围进行搜索。为了验证 LSTM 在捕获长期依赖方面的效果，本文采用 SimpleRNN 模型进行对比实验，即用 RNN 单元替换 LSTM 单元，保持其它结构不变。

5.6.6 实验结果与结论

本文首先使用不同的预测模型 ARIMA，SVR 和 SimpleRNN 对测试集进行预测，实验结果如表16所示。然后根据预测结果绘制原始工作负载时间序列和预测时间序列，如图41所示。表16显示本文提出的算法在 MAPE 方面优于所有基线，且在 MAPE 指标上改善至少 1.10%，最多提升 2.46%。相比 SVR 模型，本文提出的模型虽然在 RMSE 和 MAE 上稍逊，但是 MAPE 有了 2.20% 的提升，可能由于 SVR 的间隔边界机制对于拟合高波动高噪声的数据有先天优势，然而 SVR 没有建模时序前后依赖关系，导致在 MAPE 指标上的下降。

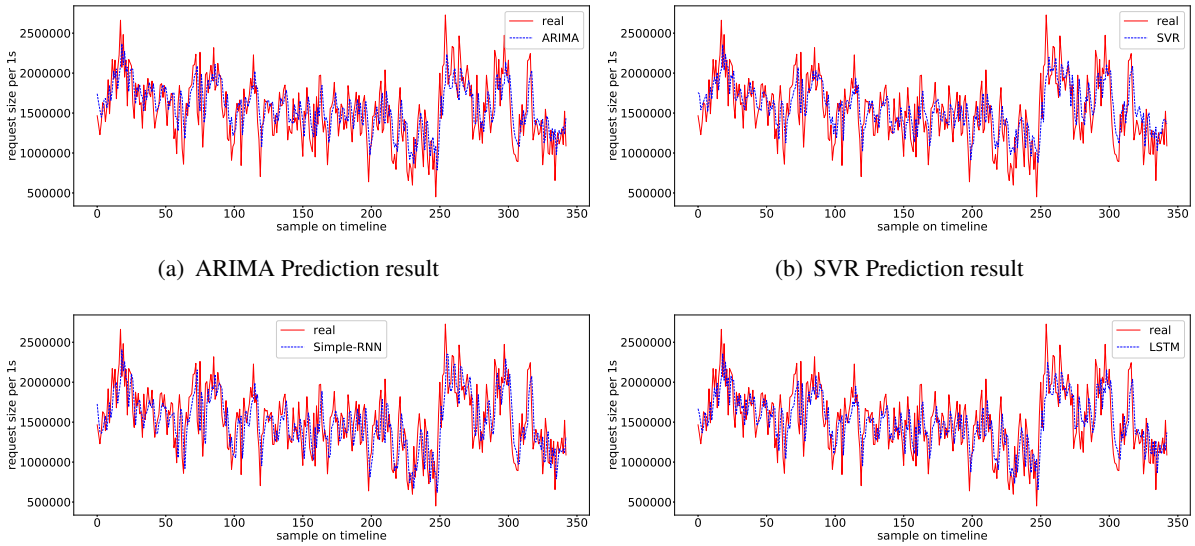
表 16 I/O 负载预测模型预测性能对比表

Model	RMSE	MAE	MAPE
Arima	307629.58(0.89)	250214.06(-0.28)	0.1846(2.46)
SVR	304278.19(-0.20)	248345.99(-1.03)	0.1840(2.20)
SimpleRNN	314267.07(2.98)	258297.24(2.86)	0.1837(1.10)
Ours	304894.71	250915.78	0.1800

[a] data in () means deteriorative percentage compared with our model.

5.6.7 参数敏感性分析

本文还研究了 LSTM 中重要超参数的灵敏度，即隐藏的大小。绘制了隐藏层神经元个数与指标 RMSE，MAE 和 MAPE 之间的关系如图42所示。可以看出，随着隐藏神经元个数的增加，RMSE，MAE 和 MAPE 的指标具有减小然后增加的趋势。当隐藏层神经元在 250 到 260 之间时，三者普遍可以获得更好的结果。因此，简单地增加网络的

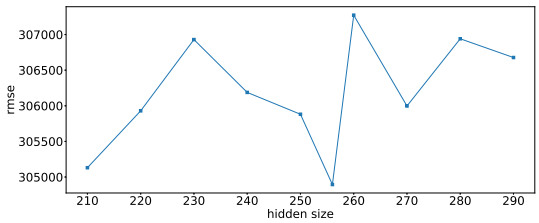


(a) ARIMA Prediction result　(b) SVR Prediction result

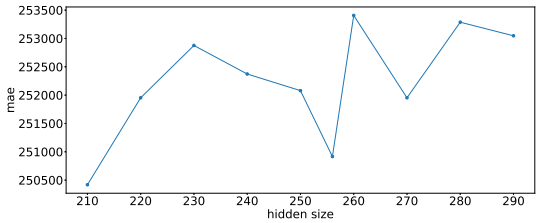
(c) SimpleRNN Prediction result　(d) LSTM Prediction result

图 41 I/O 负载预测模型测试集预测效果对比图

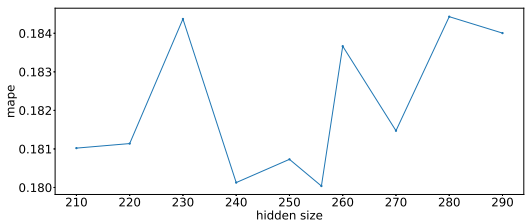
大小以提高模型性能的想法是不合理的。



(a) RMSE change along with hidden size



(b) MAE change along with hidden size



(c) MAPE change along with hidden size

图 42 I/O 负载预测模型参数敏感性分析图

5.7 本章小结

本章节针对现有 I/O 负载点值预测模型的缺点，通过自相关函数作为数学工具，挖掘出 I/O 负载中存在的长期依赖特性，并提出了基于长短期记忆网络的存储系统 I/O 负载短期预测算法，然后在开源 WebSearch trace 数据集上进行了实验，验证了本模型的有效性。

总结与展望

总结

计算机集群系统在运行过程中会产生大量的 trace 数据，这些数据记录了系统在不同时刻的负载信息，基于时间序列挖掘方法采用深度学习模型对集群系统中产生的 trace 数据进行研究，预测系统的未来负载状态，可以为系统资源管理和性能优化提供理论上的指导。本文利用开源的 trace 数据，结合领域特征，在三个场景下研究基于深度学习的时序挖掘算法。取得的主要成果如下：

1）设计实现了特征增强的多任务 LSTM 负载转折点预测算法：设计了四种时序特征用于描述云环境下服务器负载转折点的波动特性，并给出了合理的解释。提出了一个特征增强的多任务 LSTM 模型，将特征序列和原始负载序列分别视为两种不同的任务进行学习，以更好的从特征序列和原始负载序列中提取波动模式；然后采用显式的融合结构，学习特征序列和原始负载序列之间的隐含关系，达到特征增强的效果。在开源的谷歌云服务集群 trace 上进行的实验表明我们的模型在 F1 指标上优于五种基线模型，在 F1 指标上最少平均提升 2 个百分点。然后通过对照实验验证了此模型的特征增强机制以及多任务结构的作用。

2）设计并实现了基于通道注意力卷积模型的任务失败预测算法：根据分析出的多监测指标时间序列的两种特性：序列内变化的独立性以及序列间的交互性，采用通道分离策略，针对每种监测指标分别通过堆叠的一维卷积模块提取时序特征，然后采用多头注意力机制捕捉不同监测指标间的交互特征。在谷歌云服务集群 trace 上进行了多组实验，在 AUC 指标上相比 Chen[13] 的模型在早期和晚期预测时平均提升 9.5 个百分点，相比 Islam[14] 的模型平均提升 1.1 个百分点，相比目前最好的多变量时间序列分类模型深度残差网络[8] 平均提升 0.2 个百分点。通过对照实验，分别去除分离式的卷积模块和通道间自注意力模块，验证了本文提出的基于通道注意力卷积模型的多变量时间序列分类算法的作用。

3）设计实现了基于长短期记忆网络的存储系统 I/O 负载短期预测算法：该方法包括负载数据预处理、基于长短时记忆网络的时间序列预测和数据后处理。在实际的存储负载 trace 上进行了实验，和 3 种典型预测方法相比在 MAPE 上有至少 1.1% 的提升。

展望

本文基于领域数据存在的时序特性，在三种场景下分别设计实现了三种基于深度学习的负载数据时序挖掘算法，预测效果相比现有模型均有提升。未来可能在以下几个方面进行优化：

1）在工作负载转折点预测问题的研究中，由于目前监测时长最长的谷歌云服务集群 trace 仅提供 29 天，以 5 分钟为基本单位的数据，单个服务器的时序数据总长度不足 10000，这种数据量对于基于深度学习的时间序列挖掘算法来说略显不足。而每个服务器负载时序数据呈现出的时序特征往往很不相同，导致使用多个服务器的数据一起训练模型时效果较差。这就大大限制了本文模型的应用场景。然而最近有研究表明[60] 通过对抗学习（Adversarial Training）能够帮助模型从多条有差异的时间序列中学习到更加鲁棒的特征。所以后续可以研究基于多服务器负载数据的转折点预测算法。

2）本文的第一和第二个工作都是基于谷歌集群进行的，然而国内的阿里巴巴在 2018 年又发布了一个更大规模的集群 trace 数据，数据量超过了谷歌 trace，并且监测粒度为秒级，相比谷歌 trace 的 5 分钟有了较大的提升，未来可以尝试探索该 trace 数据中的时序特征。

参考文献

[1] TOP500 News Team . China Extends Lead in Number of TOP500 Supercomputers, US Holds on to Performance Advantage[EB/OL]. 2019.
https://www.top500.org.

[2] 中国信通院 . 数据中心白皮书（2018）[R]. 北京市海淀区花园北路 : 中国信息通信研究院, 2018.

[3] Reiss C, Wilkes J, Hellerstein J L. Google cluster-usage traces: format + schema[R]. Mountain View, CA, USA : Google Inc., 2011.

[4] Lu C, Ye K, Xu G, et al. Imbalance in the cloud: An analysis on alibaba cluster trace[C] // 2017 IEEE International Conference on Big Data (Big Data). 2017 : 2884 – 2892.

[5] Los Alamos National Laboratory . Operational Data to Support and Enable Computer Science Research[EB/OL]. 2005.
http://institute.lanl.gov/data/fdata/.

[6] Zhao Y, Shen Y, Zhu Y, et al. Forecasting wavelet transformed time series with attentive neural networks[C] // 2018 IEEE International Conference on Data Mining (ICDM). 2018 : 1452 – 1457.

[7] Lin T, Guo T, Aberer K. Hybrid neural networks for learning the trend in time series[C] // Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. 2017 : 2273 – 2279.

[8] Wang Z, Yan W, Oates T. Time series classification from scratch with deep neural networks: A strong baseline[C] // 2017 international joint conference on neural networks (IJCNN). 2017 : 1578 – 1585.

[9] Deng S, Zhang N, Zhang W, et al. Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network[C] // Companion Proceedings of The 2019 World Wide Web Conference. 2019 : 678 – 685.

[10] Xia B, Li T, Zhou Q-F, et al. An effective classification-based framework for predicting cloud capacity demand in cloud services[J]. IEEE Transactions on Services Computing,

2018.

[11] Keogh E, Chu S, Hart D, et al. An online algorithm for segmenting time series[C] // Proceedings 2001 IEEE International Conference on Data Mining : IEEE Computer Society, 2001 : 289 – 296.

[12] Luo L, Chen X. Integrating piecewise linear representation and weighted support vector machine for stock trading signal prediction[J]. Applied Soft Computing, 2013, 13(2) : 806 – 816.

[13] Chen X, Lu C-D, Pattabiraman K. Failure analysis of jobs in compute clouds: A google cluster case study[C] // 2014 IEEE 25th International Symposium on Software Reliability Engineering. 2014 : 167 – 177.

[14] Islam T, Manivannan D. Predicting application failure in cloud: A machine learning approach[C] // 2017 IEEE International Conference on Cognitive Computing (ICCC). 2017 : 24 – 31.

[15] Liu C, Han J, Shang Y, et al. Predicting of job failure in compute cloud based on online extreme learning machine: A comparative study[J]. IEEE Access, 2017, 5 : 9359 – 9368.

[16] El-Sayed N, Zhu H, Schroeder B. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations[C] // 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). 2017 : 1333 – 1344.

[17] Fadishei H, Saadatfar H, Deldari H. Job failure prediction in grid environment based on workload characteristics[C] // 2009 14th International CSI Computer Conference. 2009 : 329 – 334.

[18] Pan X, Tan J, Kavulya S, et al. Ganesha: Blackbox diagnosis of mapreduce systems[J]. ACM SIGMETRICS Performance Evaluation Review, 2010, 37(3) : 8 – 13.

[19] 刘春红 , 韩晶晶 , 商彦磊 . 基于 SVM 分类的云集群失败作业主动预测方法 [J], 2016.

[20] Zheng Y, Liu Q, Chen E, et al. Time series classification using multi-channels deep convolutional neural networks[C] // International Conference on Web-Age Information Management. 2014 : 298 – 310.

[21] Fawaz H I, Forestier G, Weber J, et al. Deep learning for time series classification: a

review[J]. Data Mining and Knowledge Discovery, 2019, 33(4) : 917 – 963.

[22] Dong B, Li X, Wu Q, et al. A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers[J]. Journal of Parallel and distributed computing, 2012, 72(10) : 1254 – 1268.

[23] Gupta S, Dinesh D A. Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks[C] // 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). 2017 : 1 – 6.

[24] Di S, Kondo D, Cirne W. Host load prediction in a Google compute cloud with a Bayesian model[C] // Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. 2012 : 21.

[25] Song B, Yu Y, Zhou Y, et al. Host load prediction with long short-term memory in cloud computing[J]. The Journal of Supercomputing, 2018, 74(12) : 6554 – 6568.

[26] Zhang Z, Tang X, Han J, et al. Sibyl: Host Load Prediction with an Efficient Deep Learning Model in Cloud Computing[C] // Algorithms and Architectures for Parallel Processing - 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part II. 2018 : 226 – 237.

[27] Box G E, Jenkins G M, Reinsel G C, et al. Time series analysis: forecasting and control[M] : John Wiley & Sons, 2015.

[28] Hamilton J D. Time series analysis : Vol 2[M] : Princeton university press Princeton, NJ, 1994.

[29] Walker G T. On periodicity in series of related terms[J]. Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, 1931, 131(818) : 518 – 532.

[30] Slutzky E. The summation of random causes as the source of cyclic processes[J]. Econometrica: Journal of the Econometric Society, 1937 : 105 – 146.

[31] Drucker H, Burges C J, Kaufman L, et al. Support vector regression machines[C] // Advances in neural information processing systems. 1997 : 155 – 161.

[32] Bengio Y, Delalleau O, Le Roux N. The curse of dimensionality for local kernel machines[J]. Techn. Rep, 2005, 1258.

[33] Fischer T, Krauss C. Deep learning with long short-term memory networks for financial market predictions[J]. European Journal of Operational Research, 2018, 270(2): 654–669.

[34] Hossain M, Rekabdar B, Louis S J, et al. Forecasting the weather of Nevada: A deep learning approach[C] // 2015 international joint conference on neural networks (IJCNN). 2015: 1–6.

[35] Bengio Y, Simard P, Frasconi P, et al. Learning long-term dependencies with gradient descent is difficult[J]. IEEE transactions on neural networks, 1994, 5(2): 157–166.

[36] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735–1780.

[37] Amiri M, Khanli L M. Survey on prediction models of applications for resources provisioning in cloud[J]. J. Network and Computer Applications, 2017, 82: 93–113.

[38] Duggan M, Shaw R, Duggan J, et al. A multitime-steps-ahead prediction approach for scheduling live migration in cloud data centers[J]. Softw., Pract. Exper., 2019, 49(4): 617–639.

[39] Zhang G, Bao W, Zhu X, et al. A server consolidation method with integrated deep learning predictor in local storage based clouds[J]. Concurrency and Computation: Practice and Experience, 2018, 30(23).

[40] Moreno-Vozmediano R, Montero R S, Huedo E, et al. Efficient resource provisioning for elastic Cloud services based on machine learning techniques[J]. Journal of Cloud Computing, 2019, 8(1): 5.

[41] JoSEP A D, KAtz R, KonWinSKi A, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4).

[42] Wilder J W. New concepts in technical trading systems[M]: Trend Research, 1978.

[43] Fortier, M . TA-Lib: Technical Analysis Library[EB/OL]. 2005. http://institute.lanl.gov/data/fdata/.

[44] Christ M, Braun N, Neuffer J, et al. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)[J]. Neurocomputing, 2018, 307: 72–77.

[45] Gulli A, Pal S. Deep Learning with Keras[M]: Packt Publishing Ltd, 2017.

[46] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning[C] // 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016 : 265 – 283.

[47] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python[J]. Journal of machine learning research, 2011, 12(Oct) : 2825 – 2830.

[48] Sebastio S, Trivedi K S, Alonso J. Characterizing machines lifecycle in google data centers[J]. Performance Evaluation, 2018, 126 : 39 – 63.

[49] Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[50] Vapnik V. The nature of statistical learning theory[M] : Springer science & business media, 2013.

[51] Hongyan T, Ying L, Tong J, et al. Time Series Based Killer Task Online Recognition Service: A Google Cluster Case Study[C] // 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE). 2016 : 136 – 145.

[52] Normalization B. Accelerating deep network training by reducing internal covariate shift[J]. CoRR.–2015.–Vol. abs/1502.03167.–URL: http://arxiv. org/abs/1502.03167, 2015.

[53] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C] // Advances in neural information processing systems. 2017 : 5998 – 6008.

[54] Cui Y, Chen Z, Wei S, et al. Attention-over-attention neural networks for reading comprehension[J]. arXiv preprint arXiv:1607.04423, 2016.

[55] Paulus R, Xiong C, Socher R. A deep reinforced model for abstractive summarization[J]. arXiv preprint arXiv:1705.04304, 2017.

[56] Huang J, Ling C X. Using AUC and accuracy in evaluating learning algorithms[J]. IEEE Transactions on knowledge and Data Engineering, 2005, 17(3) : 299 – 310.

[57] Liu G, Shen H, Wang H. Towards long-view computing load balancing in cluster storage systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(6) : 1770 – 1784.

[58] Seabold S, Perktold J. Statsmodels: Econometric and statistical modeling with python[C]

// Proceedings of the 9th Python in Science Conference : Vol 57. 2010 : 61.

[59] Bates K, McNutt B. UMass Trace Repository[R]. Redwood City, CA 94062-1645 : Storage Performance Council, 2002.

[60] Feng F, Chen H, He X, et al. Enhancing stock movement prediction with adversarial training[C] // Proceedings of the 28th International Joint Conference on Artificial Intelligence. 2019 : 5843 – 5849.

[61] Zhang G, Bao W, Zhu X, et al. A server consolidation method with integrated deep learning predictor in local storage based clouds[J]. Concurrency and Computation: Practice and Experience, 2018, 30(23) : e4503.

[62] Hallac D, Vare S, Boyd S, et al. Toeplitz inverse covariance-based clustering of multivariate time series data[C] // Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017 : 215 – 223.

[63] Jiang G, Chen H, Yoshihira K. Efficient and scalable algorithms for inferring likely invariants in distributed systems[J]. IEEE Transactions on knowledge and data engineering, 2007, 19(11) : 1508 – 1523.

[64] Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.

[65] Mason K, Duggan M, Barrett E, et al. Predicting host CPU utilization in the cloud using evolutionary neural networks[J]. Future Generation Computer Systems, 2018, 86 : 162 – 173.

[66] Peng C, Li Y, Yu Y, et al. Multi-step-ahead Host Load Prediction with GRU Based Encoder-Decoder in Cloud Computing[C] // 2018 10th International Conference on Knowledge and Smart Technology (KST). 2018 : 186 – 191.

[67] Yu Y, Jindal V, Yen I-L, et al. Integrating clustering and learning for improved workload prediction in the cloud[C] // 2016 IEEE 9th International Conference on Cloud Computing (CLOUD). 2016 : 876 – 879.

[68] Zhang W, Li B, Zhao D, et al. Workload prediction for cloud cluster using a recurrent neural network[C] // 2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI). 2016 : 104 – 109.

[69] Barati M, Sharifian S. A hybrid heuristic-based tuned support vector regression model for cloud load prediction[J]. The Journal of Supercomputing, 2015, 71(11) : 4235 – 4259.

[70] Yang Q, Zhou Y, Yu Y, et al. Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing[J]. The Journal of Supercomputing, 2015, 71(8) : 3037 – 3053.

[71] Barroso L A, Hölzle U, Ranganathan P. The datacenter as a computer: Designing warehouse-scale machines[J]. Synthesis Lectures on Computer Architecture, 2018, 13(3) : i – 189.

[72] Ruan L, Xu X, Xiao L, et al. A Comparative Study of Large-Scale Cluster Workload Traces via Multiview Analysis[C] // 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). 2019 : 397 – 404.

攻读硕士学位期间取得的学术成果

论文成果：

[1] Ruan Li, **Bai Yu**，Xiao Limin, Cloud Server Load Turning Point Prediction Based on Feature Enhanced Multi-Task LSTM，International Conference on Algorithms and Architectures for Parallel Processing 2019. (学生一作，CCF-C 已录用，2019 年 12 月 9 到 11 日在澳大利亚墨尔本召开)

[2] Ruan Li, **Bai Yu**, Xiao Limin, Xv Rongbin, Zhang Yiyang, He Shuibing, You Hongtao, Workload Time Series Prediction in Storage Systems: A Deep Learning Based Approach, The 2nd Industry/University Joint International Workshop on Data Center Automation, Analytics, and Control (DAAC), 2018. (学生一作)