基于深度学习的系统日志异常检测研究

王易东[1]，刘培顺[1]，王彬[2]

（1. 中国海洋大学信息科学与工程学院，山东 青岛 266100；
2. 中国海洋大学继续教育学院，山东 青岛 266100）

摘　要：系统日志反映了系统运行状态，记录着系统中特定事件的活动信息，快速准确地检测出系统异常日志，对维护系统安全稳定具有重要意义。提出了一种基于 GRU 神经网络的日志异常检测算法，基于 log key 技术实现日志解析，利用执行路径的异常检测模型和参数值的异常检测模型实现日志异常检测，具有参数少、训练快的优点，在取得较高检测精度的同时提升了运行速度，适用于大型信息系统的日志分析。

关键词：日志异常检测；深度学习；GRU 神经网络

中图分类号：TP390

文献标识码：A

doi: 10.11959/j.issn.2096−109x.2019055

Research on system log anomaly detection based on deep learning

WANG Yidong[1], LIU Peishun[1], WANG bin[2]

1. College of Information Science and Engineering, Ocean University of China, Qingdao 266100, China
2. School of Continuing Education, Ocean University of China, Qingdao 226100, China

Abstract: The system log reflects the running status of the system and records the activity information of specific events in the system. Therefore, the rapid and accurate detection of the system abnormal log is important to the security and stability of the system. A log anomaly detection algorithm based on GRU neural network is proposed. Log parsing is implemented based on log key technology. Log anomaly detection is realized by using anomaly detection model of execution path and anomaly detection model of parameter value. The system has the advantages of less parameters and faster training. It improves the running speed while achieving higher detection accuracy, and is suitable for log analysis of large information systems.

Key words: log anomaly detection, deep learning, GRU neural network

收稿日期：2019−03−12；**修回日期：**2019−04−30

通信作者：刘培顺，liups@ouc.edu.cn

基金项目：国家重点研发计划基金资助项目（No.2016YFF0806200）

Foundation Item: The National Key Research and Development Program of China (No.2016YFF0806200)

论文引用格式：王易东, 刘培顺, 王彬. 基于深度学习的系统日志异常检测研究[J]. 网络与信息安全学报, 2019, 5(5): 105-118.

WANG Y D, LIU P X, WANG B. Research on system log anomaly detection based on deep learning[J]. Chinese Journal of Network and Information Security, 2019, 5(5): 105-118.

1 引言

在异常检测领域中，系统日志异常检测一直是一个研究热点。系统日志作为具有多种自由格式的非结构化数据集，和文本分析、统计学、机器学习等学科都有着较为紧密的结合。多年来，各国研究人员将不同领域的方法应用到日志异常检测，并取得了大量出色研究成果。Xu[1]等利用抽象语法树（AST，abstract syntax tree）和主成分分析（PCA，principal component analysis）方法处理解析后的日志特征集，降低了待分析特征集的复杂度，得到了有效的异常检测结果。不过该方法需要预先获取程序源代码和日志的种类，不能作为一种通用的日志异常检测方法。Yu[2]等提出基于 Workflow 监控的异常检测系统——CloudSeer，它可以通过检查交错日志序列中的错误信息来获取执行异常。该方法在一定程度上解决了日志的并发性问题，但它借助自动机组实现，其中某些特定的规则只适用于云基础设施中的日志异常检测。

近些年，深度学习蓬勃发展，不断开创新的应用模式，尤其在 NLP（natural language processing）领域进展巨大，大量 NLP 相关任务的最佳模型均在其基础上建立。Du[3]等将系统日志建模为自然语言序列，提出了一种基于 LSTM 的深度神经网络模型——DeepLog。该模型从正常的日志数据中学习日志规则，当检测到的日志偏离正常规则时，即认定其为异常。实验结果表明，该方法在多个大型数据集上取得了较高的检测精度，总体性能优于其他基于传统数据挖掘的日志异常检测方法。然而在检测效率方面，该方法仍有一定提升空间。

本文在 N-gram 语言模型[4]的基础上，结合循环神经网络提出了一种基于 GRU（gated recurrent unit）神经网络模型的日志异常检测算法。针对复杂的非结构化日志，首先提取 log key，将日志解析为结构化序列，然后使用解析得到的序列训练 GRU 神经网络模型用以检测异常，具有参数少、训练快的优点，在取得较高检测精度的同时提升了运行速度。

2 基于 GRU 的日志异常检测算法

2.1 基于 logkey 的日志解析方法

系统日志数据是一种非结构化文本数据，可以直接从日志文件中获取，在对日志数据进行分析之前，通常需要先将其解析为结构化数据。每个日志条目由常量和变量两部分组成，常量是指由系统程序源码中的 print 语句直接打印出的消息，变量则是常量以外的部分，通常是时间戳或参数值。例如，mysql 中的日志“*InnoDB*：*Initializingbuffer*，*size*=256.0*M*”，常量部分为“*InnoDB*：*Initializingbuffer*，*size*=”，“256.0*M*”则是变量，表示数据库缓冲池大小。所有相似日志条目中的公共常量消息叫作 logkey，可以用来表示日志消息类型。正常日志的输出会遵从一定的流程和顺序，通常称为日志的执行路径（execution path），logkey 序列能够表示日志的执行路径，因此从日志中提取 logkey 是一种有效的日志解析方法。参数值是日志中非常有价值的一类信息，反映了系统的健康状态和性能，某些参数值还可以作为特定执行序列的标识，如 HDFS 日志中的 block_id，基于此能够从多线程并发任务中提取出特定模块的日志序列。

基于 logkey 的异常检测方法，首先需要构造一个包含系统中所有 logkey 的集合，作为 logkey 表（vocabulary）。用 $K=\{k_1,k_2,\cdots,k_n\}$ 表示 logkey 表，则日志解析得到的 logkey 序列中的每一个 logkey 都可以在 K 中找到。对于一个 logkey 序列，序列索引号 t 位置处的值依赖于它有限的历史序列。构建一个大小为 h 的窗口 $W=\{w_{t-h},w_{t-h+1},\cdots,$

$w_{t-1}\}$ 表示 t 位置之前的 logkey 序列，则 t 位置可能输出集合 K 中的任意一个 logkey。因此可以将基于 logkey 的日志异常检测转化为一个多分类问题，K 中每个 logkey 作为一个单独的类，通过 t 位置的历史 logkey 序列，计算出 K 中每一类 logkey 出现在 t 位置的条件概率，用 $P(y_t = x_i \mid w)$ 表示。对 K 中 logkey 出现的概率按照从高到低进行排序，如果新生成的日志条目的 logkey 恰好是 K 中概率最高的几个 logkey 之一，则将其视为正常，否则代表系统出现异常。

当前业内存在多种日志解析工具，Spell[5]是较为先进的一种，它基于 LCS 思想设计，由 MIT 的 logPAI 团队[6-7]开源实现，能够对日志进行在线解析。本文使用 Spell 从日志数据中解析出 logkey 和参数值，作为日志异常检测的基础。

2.2　基于执行路径的异常检测模型

日志异常检测和单词序列预测同属于一类序列预测问题，将日志看作一种特殊的自然语言，logkey 序列相当于一个句子，序列中的每个 logkey 都可以看作一个单词，因此可以采用单词序列预测[4]的建模方法对日志异常检测问题进行建模。基于文献[3]的思路，本文设计并实现了一种基于 GRU 的神经语言模型，该模型能够通过序列的长期依赖检测异常。与文献[3]中构建的 LSTM 模型不同，本文模型使用 GRU 作为循环单元，具有参数少、训练快的优点，在取得较高检测精度的同时提升了运行速度。

基于 GRU 的执行路径异常检测模型分为输入层（input layer）、嵌入层（embedding layer）、隐藏层（hidden layer）和输出层（output layer）。如图 1 所示，模型每一个时间步（timestep）的输入为 log key w_{t-i}，它和上一个时间步的记忆状态共同计算得到当前状态。

输入层：首先在输入层对 logkey 表中的元素进行 one-hot 编码（one-hot encoding）。logkey 表 $K=\{k_1,k_2,\cdots,k_n\}$ 中的每一个元素都可以编码为 1 个 n 维向量，编码结果为 $\{1,0,\cdots,0\}$，$\{0,1,\cdots,0\}$，$\cdots\{0,0,\cdots,1\}$。每个 n 维向量中，1 所在的位置即为其索引位置。对于一个窗口长度的 logkey 序列，经过输入层 one-hot 编码，得到一个索引序列参与后续计算。

嵌入层：若 logkey 表中元素的个数比较多，编码后的向量往往很长且相当稀疏，这将对计算资源造成极大的浪费。因此模型在输入层和隐藏层之间加入了一个嵌入层，用来将 one-hot 向量转化为嵌入向量。嵌入层维护了一个权重矩阵，它直接查找 one-hot 向量中维度值为 1 的位置在矩阵中对应的权重值作为嵌入向量，此时权重矩阵相当于一个查找表（lookuptable）。如图 1 中所示，权重矩阵的行数等于 logkey 表中元素个数 n，列数等于输入元素的特征数 m。假设输入序列长度 l 为 3，有效值所在的位置分别为 1、3、17，相应从权重矩阵中取出第 1、3、17 行向量组成一个 $3\times m$ 的矩阵，即为嵌入层的输出。

隐藏层：该层是 GRU 网络模型的核心部分，在隐藏层中，每个 GRU 节点都是一个记忆块（memory block），如图 1 所示，记忆块之间相互连接，构成一个完整的循环神经网络。单个 GRU 记忆块的构造如图 2 所示，其中包含两个门：更新门和重置门。更新门接收当前输入和上一时刻的隐藏层输出，决定其中有多少信息需要继续传递。通过 Sigmoid 激活函数处理输入信息，更新门得到 1 个介于 0 和 1 之间的结果，其计算公式如下。

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{1}$$

其中，x_t 和 h_{t-1} 分别为当前输入和上一个 GRU 块（记忆块）隐藏层的输出，W_z 和 U_z 为更新门的线性变换参数。

重置门则控制对历史信息的遗忘程度，其表达式形式与更新门相同，只是线性变换参数及其作用不同。

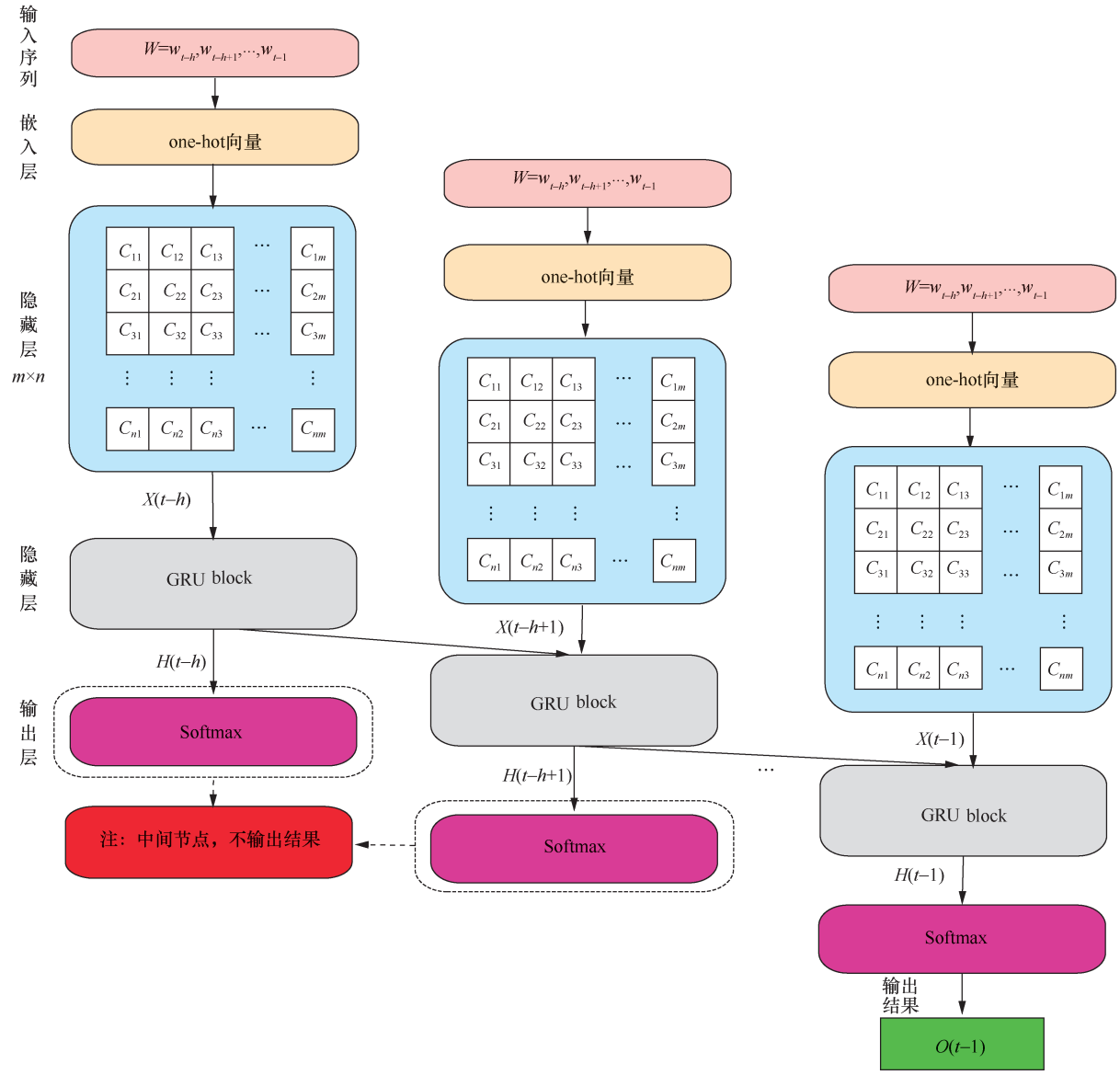


图 1　执行路径异常检测模型结构

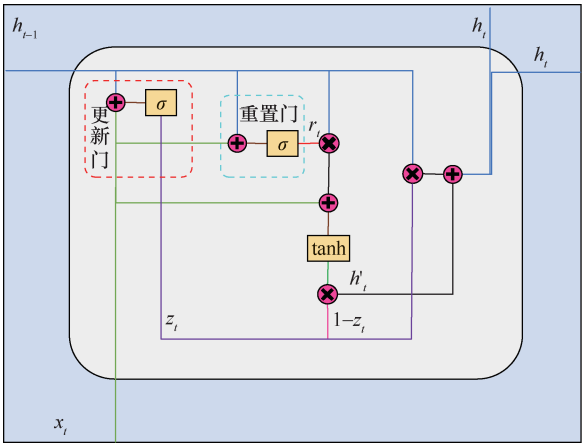


图 2　单个 GRU 块内部结构

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2}$$

借助重置门，可以得到未被遗忘的历史记忆。将其与当前输入通过 tanh 激活函数计算，即可得到 GRU 单元的当前记忆内容。

$$h_t' = \tanh(W_h x_t + r_t \odot U_h h_{t-1} + b_h) \tag{3}$$

输出层：数据通过更新门，从当前记忆和历史记忆中收集信息，两者被保留的信息相加，就得到了当前 GRU 单元在时间序列上的输出。

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t' \tag{4}$$

当前 GRU 块的输出在时间序列上作为下一个 GRU 块的输入，而序列中最后一个 GRU 块的输出则作为整个模型输出层的输入。模型的输出层采用了一个 Softmax 多类分类器，通过 Softmax 函数计算得到一个 n 维向量，每一维度的值代表 logkey 表中的每个元素出现在当前位置的概率，所有概率之和为 1。计算过程的数学形式如下。

$$o_t = \text{softmax}(Vh_t + b_V) \tag{5}$$

由此可得出每个 logkey 出现的概率。

$$p(x_t|C) = \frac{e^{(h_t V'(x_t))}}{\sum_{x_i \in V} e^{(h_t V'(x_i))}} \tag{6}$$

其中，c 表示输入的历史序列，x_t 代表目标 logkey，$V'(x)$ 为 Softmax 层中的输出词向量（对应 embedding 层中的输入词向量）。

将 logkey 表按照输出的概率值从大到小排列，选取其中前 b 个组成集合。若系统当前时刻输出日志的 logkey 存在于集合中，则认为该日志是正常的，否则视为异常。

为了提升性能，模型中的 GRU 网络被设计为多层的实现形式。上层 GRU 的隐藏层输出作为下层 GRU 的输入，层与层之间使用 dropout 方法对数据进行正则化。图 3 给出了一个双层 GRU 语言模型的结构，图中虚线箭头表示使用 dropout 的连接。如果应用场景需要更高的检测精度，模型的层数可以视实际情况增加。

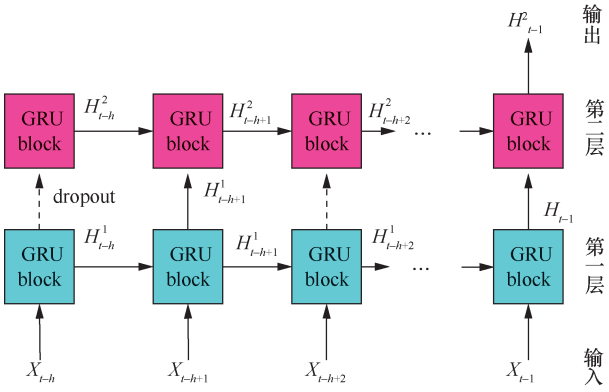


图 3　双层 GRU 模型结构

模型共包括 4 个参数：b、h、L、u。其中，b 表示模型预测结果的可信范围，若观测值存在于前 b 个预测值组成的集合中，则认为其正常，b 的默认值设为 9。h 为滑动窗口大小，即输入序列长度，默认取 10。L 为 GRU 模型的层数，默认采用两层 GRU。u 为单个 GRU 块包含的存储单元数量，默认值为 64。下文设置了多组对比实验，通过控制变量法测试单个参数值变化对模型性能的影响。

2.3　基于参数值的异常检测模型

在一般场景下，使用基于执行路径的日志异常检测模型可以检测出系统中大多数异常。然而，在某些场景下产生的异常，如系统遭受拒绝服务攻击（denial of service attack）导致的运行速度变慢，往往体现在系统日志参数值的变化上。本节构建了一个参数值异常检测模型，通过日志中参数值的变化趋势来检测系统异常状态。

每个日志条目由时间戳、log key 和若干个参数组成，其中，相邻时间戳的差值代表两条日志生成的时间间隔，能够衡量系统的性能，也可将其看作一个日志参数。从日志条目中提取时间戳差值和参数值，构造出一个参数值向量，对于具有相同 log key 的日志条目，将其参数值向量按时间顺序生成一个序列，则不同的 log key 将生成多个参数值向量序列。每个参数值向量序列都可以看作一个单独的时间序列，因此参数异常检测问题被转化为多变量时间序列预测问题，通过预测值和实际值的对比，判断系统是否发生异常。

参数值异常检测模型由三大模块组成，分别为分类模块、预测模块和判断模块。分类模块根据 log key 对输入的参数值向量进行分类，结果传输到预测模块。预测模块依然采用 GRU 网络建模，针对每个具有不同 log key 的参数值向量序列，分别为其构建一个独立的 GRU 网络。每个 GRU 网络都可以看作一个 N–1 的 GRU 时间序

列模型，它在每个 timestep 输入为当前时刻的参数值向量，然后模型根据由各个时刻参数值向量组成的参数值向量序列预测出下一时刻的参数值向量。

GRU 时间序列网络由输入层、隐藏层、输出层 3 层构成。输入层首先需要对数据进行预处理，使其适配 GRU 网络。预处理方式包括删除与预测无关的参数、将字符型数据编码为整数（如日志项 connecting to node1，其中，node1 为参数项，可将其编码为整数处理）、输入数据进行归一化（normalization）处理。数据归一化是机器学习中一项重要的预处理工作，具体做法是采用某种算法处理数据，将其限制在模型所需要的范围内。本文采用离差标准化（min-max normalization）作为模型的归一化方法，该方法基于样本数据的最大值和最小值对数据进行标准化，使处理后的数据特征分布在[0,1]范围内。其数学形式如下。

$$x^{*}=\frac{x-x_{\min}}{x_{\max}-x_{\min}} \tag{7}$$

归一化后的数据输入隐藏层中参与后续处理。隐藏层的架构与 2.2 节描述执行路径异常检测模型类似，隐藏层节点之间相互连接形成一个完整的循环神经网络。

在隐藏层后加入一个输出层，输出结果为一个实值向量，即根据历史序列预测出的参数值向量。

GRU 时间序列模型采用均方误差（MSE mean-square error，）损失函数来衡量预测值与真实值之间的差异。假设模型预测得到的参数值向量为 $\boldsymbol{o}=\{o_1,o_2,o_3\}$，实际参数值向量为 $\boldsymbol{p}=\{p_1,p_2,p_3\}$，则它们的均方误差为

$$MSE_{(\boldsymbol{o},\boldsymbol{p})}=\frac{\sum_{i=1}^{3}(o_i-p_i)^2}{3} \tag{8}$$

经过 BPTT 算法训练后的模型即可用来检测日志异常。日志的异常与否通过判断模块进行判定，判断的标准是一个基于训练样本的预测值与真实值之间的误差建模的高斯分布（Gaussian distribution），置信水平的阈值可以根据实验的实际情况调整。在检测阶段，如果预测值与观测值之间的误差处在高斯分布的置信区间内，则认为其为正常，否则视为异常。置信区间（以 98% 为例）用数学公式描述为

$$P\mu-2.33\frac{\sigma}{\sqrt{n}}<M<\mu+2.33\frac{\sigma}{\sqrt{n}}=0.98 \tag{9}$$

其中，μ 和 σ 分别代表 MSE 的均值和标准差，2.33 为显著性水平 98%对应的临界值，$2.33\frac{\sigma}{\sqrt{n}}$ 代表置信区间半径。

图 4 给出了参数值异常检测模型的整体架构以及其中 GRU 时间序列模型的架构。该模型能够检测系统各种性能的异常，如在一段日志时间戳的差值突然增大，可能意味着系统在这段时间中的运行速度变慢；又如日志中表示网络传输时间的参数值大幅度增加，则意味着网络延迟变大，在排除网络环境原因后，要考虑系统遭受地址解析协议（ARP，address resolution protocol）攻击的可能性。

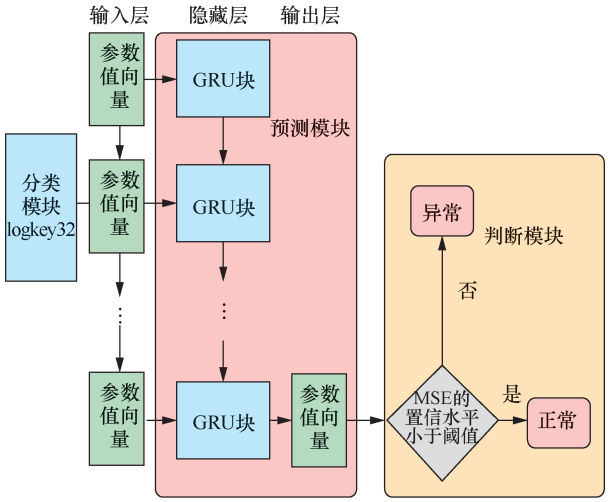


图 4　参数值异常检测模型结构

2.4 模型训练

本文设计的两个 GRU 模型（执行路径异常检测模型、参数值异常检测模型）均采用系统正常执行产生的日志作为训练样本。首先前向计算（参见 2.2 节）得到每个参数的输出值和模型的最终输出，然后对比期望输出与真实输出得到待优化的目标函数，最后通过 BPTT 算法计算各个权重参数的梯度，通过梯度下降法对参数进行更新。

训练时，首先使用窗口对样本数据进行分块，本文模型中使用的窗口为滑动窗口（sliding window）。对于一个 logkey 序列，如$\{x_8,x_2,x_7,x_6,x_1,x_{13}\}$，假设窗口大小$h$=3，步长大小（stepsize）$s$=1，则输入序列和期望输出依次为$\{x_8,x_2,x_7\to x_6\}$，$\{x_2,x_7,x_6\to x_1\}\}$，$x_7,x_6,x_1\to x_{13}\}$。

模型使用交叉熵作为损失函数，t 时刻的损失记为L_t，为了训练模型需要将所有时刻的损失相加并将其最小化。

$$L_{\min_\theta}=\sum_{t=1}^{T}L_t=\sum_{t=1}^{T}-p_t\sum_{i=1}^{n}\log(o_t) \tag{10}$$

其中，θ 为模型中所有参数的集合，o_t 为模型的预测值，p_t 为实际值。

接下来，按照 BPTT 算法，需要计算每个参数的梯度。梯度的计算依赖于模型各部分的误差项δ。各个误差项的计算公式如下。

$$\begin{cases}\delta_o(t)=V(o_t-1)\\ \delta_z(t)=\delta_o(t)\odot\left[s(t-1)-h(t)\right]\odot\sigma\left(x_tU_z+h_{t-1}W_z\right)'\\ \delta_h(t)=\delta_o(t)\odot\left[1-\mathrm{z(t)}\right]\odot[1-h(t)^2]\\ \delta_r(t)=\left[\delta_h(t)\odot\mathrm{s(t-1)}\right]\cdot W_h\end{cases} \tag{11}$$

基于上述各式计算出的模型输出、重置门、当前状态以及更新门的误差信号，利用链式法则计算出各参数梯度。

$$\begin{cases}\nabla V(t)=[o(t)-1]\odot s(t)\\ \nabla c(t)=o(t)-1\\ \nabla W_r(t)=\delta_r(t)\odot s(t-1)\\ \nabla W_h(t)=\delta_h(t)\odot s(t-1)\odot r(t)\\ \nabla W_z(t)=\delta_r(t)\odot s(t-1)\\ \nabla U_r(t)=\delta_r(t)\odot x(t)\\ \nabla U_h(t)=\delta_h(t)\odot x(t)\\ \nabla U_z(t)=\delta_z(t)\odot x(t)\\ \nabla b_r(t)=\delta_r(t)\\ \nabla b_h(t)=\delta_h(t)\\ \nabla b_z(t)=\delta_z(t)\end{cases} \tag{12}$$

模型采用 Adam[8]作为梯度下降优化算法。本文将每次更新学习的样本数量（batch size）设置为 64。在此基础上，每学习一个 mini-batch 的样本数据，模型都会利用平均梯度更新动量。

一阶动量（梯度的均值）：

$$m_t=\beta_1\cdot m_{t-1}+(1-\beta_1)g_t \tag{13}$$

二阶动量（梯度的方差）：

$$V_t=\beta_2\cdot V_{t-1}+(1-\beta_2)g_t^{\,2} \tag{14}$$

其中，g_t 为 t 时间步上的平均梯度。m_0 和 V_0 均初始化为 0，β_1 和 β_2 是 Adam 中的 2 个超参数（hyperparameter），分别控制一阶动量和二阶动量。本文实验中按照文献[8]中的推荐将 β_1 设置为 0.9，β_2 设置为 0.999。将动量引入梯度下降过程既降低了收敛波动性，又提高了训练速度。

学习率（learning rate）是深度学习中另外一个重要的超参数，控制着模型中参数的更新速度，本文实验将初始学习率设为 0.001。若采用固定学习率训练模型，当训练集的损失下降到一定程度时，便停止下降并在一定区间内来回震荡。针对这一问题，本文采用了学习率衰减（learning rate decay）算法，随着时间的推移逐渐减小学习率，学习率的更新方式基于文献[8]中的方法，其数学形式如下。

$$\alpha_t = \alpha \frac{\sqrt{1-\beta_2^{\,t}}}{1-\beta_1^{t}} \tag{15}$$

其中，t 表示 mini-batch 的时间步，每训练一个 mini-batch，t 的值加 1。α_t 表示 t 时刻的学习率。使用学习率衰减可以有效减少参数更新的次数。

Adam 借助学习率、一阶动量和二阶动量，从梯度的均值和方差两个角度出发，自适应地更新权重参数，其参数更新的数学形式如下。

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{(\sqrt{\hat{V}_t} + \varepsilon)} \tag{16}$$

其中，$\hat{m}_t$ 和 $\hat{V}_t$ 分别是偏差修正后的一阶动量和二阶动量，ε 是用于数值稳定的小常数，其值设为 10^{-8}。

使用训练集中的所有数据对模型进行一次完整训练称为 1 个 epoch。本文模型共训练 50 个 epoch，之后，神经网络基本收敛。

由于模型构造了多层 GRU 神经网络，因此模型的误差项会沿时间和空间两个方向反向传播。时间方向是指反向计算每个时刻的误差项，空间方向是指误差项向上层传播。在多层 GRU 中，上层的输出即为当前层输入，由此可得到上层输出的误差。

$$\delta_o^{(l-1)} = \delta_z^{(l)}(t)U_z^{(l)} + \delta_r^{(l)}(t)U_r^{(l)} + \delta_h^{(l)}(t)U_h^{(l)} \tag{17}$$

以此为基础，可以推导出 $\delta_z^{(l-1)}(t)$、$\delta_r^{(l-1)}(t)$、$\delta_h^{(l-1)}(t)$，进而求出相应的梯度，对上层网络进行参数更新。

2.5　模型更新

根据设计思路，模型采用系统正常运行产生的日志进行训练。然而在实际训练中，样本数据往往无法包含所有的 logkey，这样当系统生成的日志中包含不存在于样本数据集中的 logkey 时，就会造成误判。为此，模型构造了一个在线更新（online update）模块，基于线上反馈的假阳性结果实时调整模型的权重参数。

模型更新的方式采用增量更新，即只采用假阳性结果对模型进行更新。假设窗口大小 h=3，输入的历史序列为 $\{x_8, x_2, x_7\}$，系统的实际输出为 x_6 并不符合模型的预测，这时 x_6 即被标记为异常。然而经过人工检测，发现 x_6 并无异常，那么 $\{x_8, x_2, x_7 \to x_6\}$ 即被视为一个假阳性结果。模型使用该假阳性结果更新其权重参数，从而学习到这个新的日志模式。在下一次输入 $\{x_8, x_2, x_7\}$ 时，可以输出 x_6 出现的概率。

与离线训练不同，模型使用 FTRL（followed the regularized leader）算法[9]进行模型更新。该方法针对权重参数的每一维度采用不同的学习率进行学习，且能够产生易于处理的稀疏解，其数学形式如下。

$$w_{t+1} = \mathrm{argmin}_w\left(g_{1:t}w + \frac{1}{2}\sum_{s=1}^{t}\sigma_s w - w_{s2}^2 + \lambda w_1\right) \tag{18}$$

其中，w_{t+1} 为更新后的权重参数，式（18）的第一项为累计梯度和，代表损失函数下降的方向；第二项表示更新结果与现有结果的偏离不要太大；第三项是用来产生稀疏解的正则项。

使用模型更新会大幅度降低模型对某些数据集的误报率，且模型更新和在线检测可以同步进行，保证了模型的检测效率。

3　实验评估

为验证两种基于 GRU 的日志异常检测算法的有效性，本节选取了几个具有典型代表性的日志数据集，在其上进行多角度对比实验并评估两种算法的性能。实验环境在个人笔记本上配置，处理器为 Intel Corei7-6700HQ（2.60 GHz），NVIDIA GeForce GTX 965M GPU（2 GB），16 GB RAM（2 133 MHz），操作系统为 Ubuntu 16.04（64 位）。GRU 网络的搭建和训练基于深度学习框架 keras，tensorflow 作为后端，编程环境为 Python3.6.5。本节实验分为 3 部分。第一部分评

估执行路径异常检测模型的性能，使用本文算法和当前前沿的日志异常检测算法在大型 HDFS 日志数据集上进行对比实验，以多种性能指标来衡量实验结果，综合评估本文算法的性能。然后对模型自身的参数进行调整，得到不同参数下的实验结果，以此研究参数变化对模型性能的影响。第二部分评估参数值异常检测算法的性能，人工构造了一个数据集，验证算法的有效性和对检测性能的提升。第三部分对模型更新模块的性能进行评估，通过对比实验研究使用模型更新和不使用模型更新对训练时间和检测结果的影响。

3.1　执行路径异常检测模型性能评估

实验选取了 3 种当前较为先进的日志异常检测算法：主成分分析[1]（PCA，principal component analysis）、不变量挖掘[2]（IM，invariant mining）和 Deeplog[3]与本文 GRU 算法进行对比。其中，PCA 和 IM 是离线检测算法，这两种方法均使用会话窗口（session windows）对日志进行分块（本文使用滑动窗口），从日志中提取出 logkey，对 logkey 序列执行异常检测。He 等[10]实现了这两种方法，并提供了开源源代码，相关代码可以在 github 上找到。DeepLog 使用 LSTM 神经网络构建模型，能够实现对日志异常的在线检测，本文基于文献[10]中的描述实现了该方法。实验首先对比 PCA、IM、DeepLog、GRU 这 4 种方法的检测精度，然后对比 Deeplog 和 GRU 这 2 种在线检测方法的运行速度，从两方面综合评价本文算法。

对比实验采用的日志数据集为 203 个亚马逊 EC2 节点运行 38.7 h 产生的 HDFS 日志数据集[1]。该数据集中存在 11 175 629 条日志数据，包括 575 062 个事件跟踪（event trace），对应 575 062 个具有不同 block_id 的 HDFS 文件块。所有的 block_id 均由 Hadoop 领域专家标记为正常或异常（之所以能够对超过 50 万个事件跟踪进行标记，是因为大多数事件跟踪是相同且正常的），其中异常数据约占总数据的 2.9%。文献[1]构造了这个数据集，随后在日志异常检测领域被广泛使用[2,3,11]，该数据集可以在 loghub 获取。

由于 PCA 和 IM 均采用会话窗口，故对比实验将使用会话窗口作为异常检测的基准。将 HDFS 数据集按照 block_id 进行分组，可以分为 575 062 个会话，在检测过程中，只要 1 个会话中出现异常日志，该会话即被视为异常。由于 HDFS 日志的规则并不复杂，且数据集中会话存在大量重复，因此本文方法和 DeepLog 方法选取前 1% 日志数据中的正常会话作为训练集训练模型，模型参数 b、h、L、u（各参数的具体含义参见 2.2 节）均采用默认值（b=9，h=10，L=2，u=64）。PCA 和 IM 这两种无监督方法则不需要特定标记的训练集，均按照原文中给出的方法构建模型。HDFS 日志数据中每一个会话的时间跨度都比较大，考虑到 PCA 和 IM 构建模型需要完整的会话，本实验选择整个日志数据集作为 4 种算法的测试集，表 1 给出了训练集和测试集的具体信息。

表 1　　训练集和测试集信息

	训练集（本文方法）	测试集
正常会话数	4 855	558 224
异常会话数	0	16 838
Logkey 数	29	29

算法的性能通过假阳性（FP，false positives）、假阴性（FN，false negatives）、准确率（accuracy）、精确率（precision）、召回率（recall）、F 值（F-measure）来衡量。将正确检测出一个异常定义为一个正类，则 FP 和 FN 分别代表异常日志和正常日志的误报数量；准确率（$\frac{TP+TN}{TP+TN+FP+FN}$）表示被正确分类的正常日志和异常日志占总日志的百分比。

精确率（$\frac{TP}{TP+FP}$）表示检测到的异常中为真异常的比例；召回率（$\frac{TP}{TP+FN}$）表示检测到的异常占数据集中总异常的百分比；F 值（$\frac{2\cdot precision\cdot recall}{precision+recall}$）是精确率和召回率的加权调和平均，是一种基于精确率和召回率的综合评价指标。表 2 给出了 4 种算法的 FP、FN 值和准确率，其中 PCA 算法的假阳性最少，不过相应得到了较多的假阴性，说明其更容易倾向于将日志判别为正常；IM 算法的假阳性和假阴性比较多，但其总体准确率优于 PCA；本文执行路径异常检测算法和 DeepLog 同时取得了较少的假阳性和假阴性，准确率达到 99.74%和 99.75%，性能显然优于其他两者。

表 2　PCA、IM、DeepLog、本文方法性能比较结果

算法	假阳性	假阴性	准确率
PCA	281	5 429	99.00%
IM	2 131	1 226	99.42%
DeepLog	839	615	99.75%
GRU	857	624	99.74%

图 5 通过精确率、召回率、*F* 值 3 个指标进一步对 3 种算法进行比较。可以看出，DeepLog 方法取得了最高的召回率和 *F* 值，精确率虽略低于 PCA 方法，但 PCA 方法取得较高精确率的代价是较低的召回率。本文方法的各项指标略低于 DeepLog 方法，但明显高于其他两者。

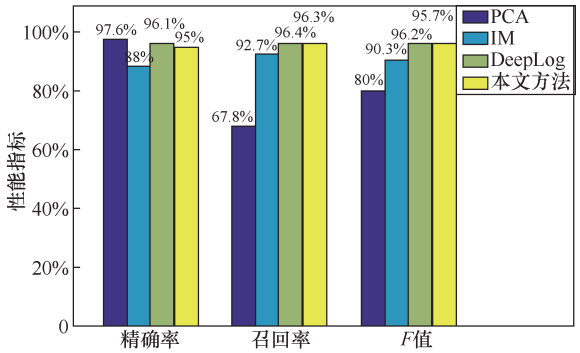


图 5　PCA、IM、DeepLog、本文方法性能比较

接下来，对 DeepLog 和本文方法的运行速度进行单独对比，运行速度通过平均每条日志所需的检测时间来衡量。表 3 展示了 2 种方法在 HDFS 日志测试集上的运行速度。

表 3　DeepLog 和本文方法运行速度对比

方法	日志条数	总时间/s	平均时间/s	时间成本
DeepLog	11 175 629	11 399	1.02	1.00
GRU（本文方法）	11 175 629	9 499	0.85	0.833

可以看到，本文算法和 DeepLog 取得的检测精度相当，但运行速度方面本文算法有较大的领先，相比于 DeepLog 约提升了 16.7%。

为研究模型参数变化对检测性能的影响，本文设计了基于控制变量法（control variates）的实验，当研究一个参数时，控制其余参数不变。模型的参数可以分为两种。一种是 GRU 层数（*L*）和 GRU 存储单元数（*u*）这种 GRU 网络本身的结构参数。图 6 显示了 *L* 和 *u* 对模型性能的影响。

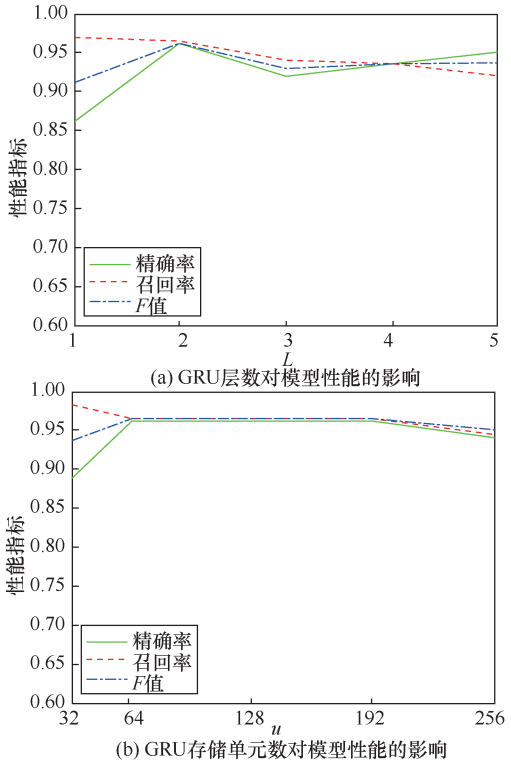
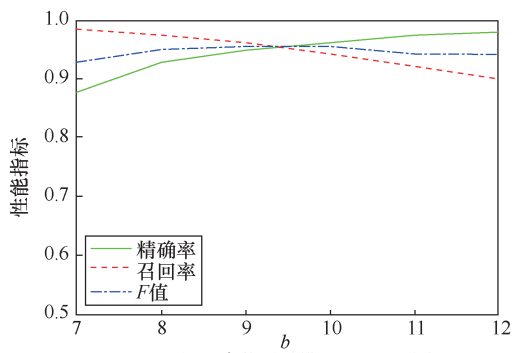


图 6　GRU 层数和 GRU 存储单元数对模型性能的影响

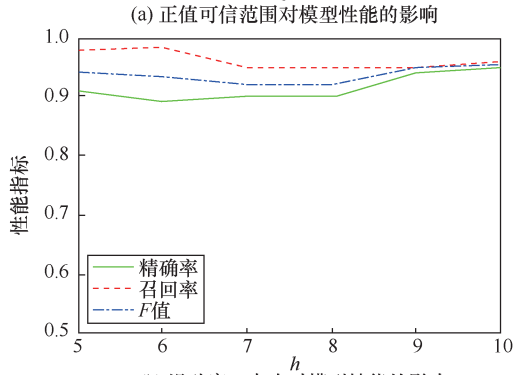
当 L=2 时，模型性能达到最佳，当 L=3 时，精确率和召回率均大幅度下降，检测精度降低，之后随着 GRU 层数增多，检测精度逐渐回升。由于训练样本过小，当 GRU 层数设置太大时，容易产生过拟合（over-fitting）现象，且过多的 GRU 层数会增大计算量；导致模型训练时间增加，因此 GRU 层数设置为 2 层较为合适。

u 表示 GRU 存储单元的个数，当其较小时，模型欠拟合（under-fitting）导致精确率较低；当将其增大到 64 时，模型性能总体上趋于稳定，随着其继续增大变化并不明显，；当其过大时，模型性能开始逐渐下降，可能是出现了过拟合。

另一种是针对样本数据的参数，包括正常值可信范围（b）和滑动窗口大小（h）。由图 7 可以看到随着 b 的增大，模型的精确率不断增大，而召回率则不断减小，在极端情况下，当 b 的值接近 log key 集的大小时，精确率可以达到 100%，但随之而来的可能是极低的召回率。当 b=9 时，F 值取得最大值，故 b=9 可作为判断异常样本的阈值。



(a) 正值可信范围对模型性能的影响



(b) 滑动窗口大小对模型性能的影响

图 7　正常值可信范围和滑动窗口大小对模型性能的影响

h 表示输入序列的长度，随着 h 的增大，算法精确率先是逐渐增大而后趋于稳定，这说明序列长度较短时，GRU 网络可能无法学习到日志样本中隐藏的规律，而序列较长时增加长度并不能显著提升模型性能，这是因为离预测值较远的数据对预测值的影响较小。

综合来看，本文算法比 PCA 和 IM 两种算法检测性能更好，比 DeepLog 算法检测速度更快，在处理大规模日志数据集时，可节省大量时间，目前日志异常检测对实时性的要求越来越高，更能体现出本文算法的优越性。且本文算法基于每条 logkey 进行检测，相比于基于会话的检测方法适用性更广。和下文中参数值异常检测算法结合使用，可以进一步提升异常检测性能。

3.2　参数值异常检测模型性能评估

对于参数值异常检测算法，选取了 Smartbi（报表工具）的客户端日志作为实验数据集。Smartbi 服务端安装在个人笔记本上，客户端安装在个人台式机上。当客户端执行任务时，会调用服务端数据库中的数据，服务端和客户端之间发生通信。实验在 Smartbi 上设置了一个重复的定时任务，通过控制网络速度模拟系统可能遭受拒绝服务攻击的场景。任务共重复了 600 次，最终采集到 37 287 条客户端日志。图 8 给出了 Smartbi 的部分日志及其解析后得到的参数值向量。

当网络波动时，日志的参数值会表现出异常。出现异常的参数值序列可分为两种：一种是相邻时间戳的差值过大，另一种是请求消耗的时间过大，这两种异常充分反映出网络的波动情况。本实验基于参数值异常检测算法构建模型，将是否能够检测出异常参数值序列作为模型有效性的评估标准。

实验将具有相同 log key 的日志分为一组，每组数据包括训练集、验证集和测试集。首先使用只包含正常数据的训练集训练异常检测模型，然后在验证集上使用训练好的模型生成 MSE 的高

斯分布，最后通过测试集检测模型性能。训练集、验证集和测试集各使用$\frac{1}{3}$的数据，在本实验中，模型成功识别出了测试集中的所有异常日志条目，识别准确率为 100%。图 9 显示了不同 logkey 的参数值向量检测结果，横坐标为 Smartbi 任务的次数序号，纵坐标为每次任务中模型预测值和真实值之间的均方误差，中间的横线表示 98%置信水平对应的置信区间阈值，当预测值和真实值之间的均方误差超过该阈值时，即认为出现异常。图 9（a）为 logkey62（INFO NewPage.show:*）的检测结果，检测出的 3 个异常参数值向量均是因为运行时间过长造成的。图 9（b）为 logkey17（INFO Insight.runRenderMacro：*）的检测结果，可以看出，该 logkey 中的参数值向量都是正常的，这是因为该 log key 并不涉及客户端和服务端之间的通信。

对于执行路径异常和参数值异常都存在的日志数据集，加入参数值异常检测可以大幅度提升检测精度。为证明参数值异常检测算法的优势，本文在 Smartbi 异常日志中人为加入了 5 条异常消息（Error：pleasere-excutethecommand），模拟执行路径异常，然后使用执行路径检测模型和参数值异常检测模型对 Smartbi 异常日志数据进行检测，与仅使用执行路径异常检测模型进行对比。表 4 给出了检测结果（多组 logkey 的总体检测结果），可以看到，单独使用执行路径异常检测模型取得的召回率仅为 22.7%，而同时使用两种模型则准确地检测出了所有异常，取得了 100%的召回率，相比仅使用执行路径异常检测模型有大幅度提升。实验结果表明，对于某些参数值存在异常的日志，由于 logkey 序列并没有发生变化，执行路径异常检测模型无法检测出这类异常，而参数值异常检测模型则可以对不同 logkey 进行分组，通过衡量预测值和真实值之间的均方误差准确地检测出参数值异常，从而提高检测召回率。

表 4　　两种异常检测模型的检测结果对比

模型情况	真阳性	假阴性	召回率
执行路径异常检测模型	5	17	22.70%
同时使用两种模型	22	0	100%

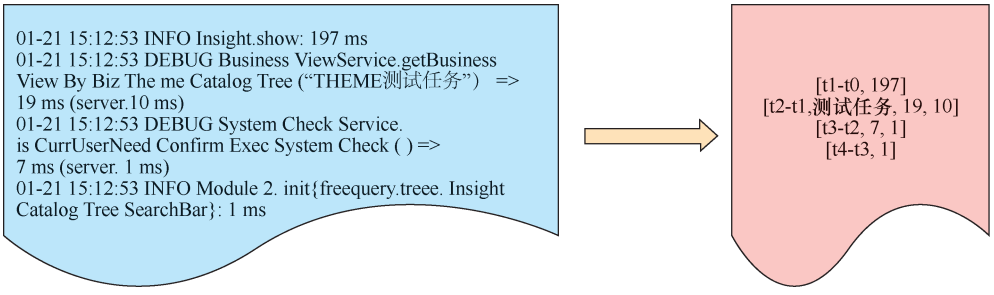


图 8　Smartbi 日志及其解析结果

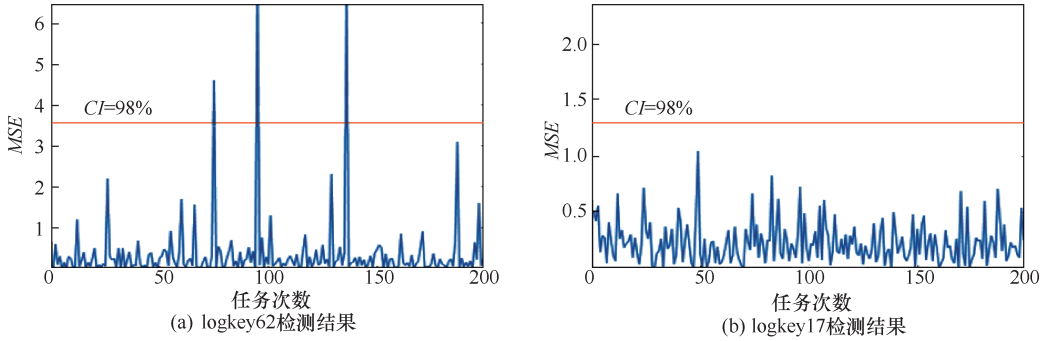


图 9　不同 logkey 下参数值异常检测结果

3.3　基于模型更新的性能提升

本文算法虽然在 HDFS 日志异常检测实验中取得了较好的性能，但在处理一些更加不规则的日志（如系统日志）时，难免会发生训练集无法涵盖所有正常执行路径的情况，当检测阶段出现不包含在训练集中的执行路径时，会引起错误预测，将其识别为异常。模型更新模块可以有效解决该问题，本节设置了是否进行模型更新的对比实验，来验证其有效性。

本实验选用的日志数据集为 708 M 的 BlueGene / L 超级计算机的系统日志[12]，该数据集包含 4 747 963 条日志，其中 348 460 条被标记为异常。文献[11]最早公开了该数据集，现已被广泛应用于日志解析，可从 loghub 下载得到。与 HDFS 日志不同，该数据集中很多日志只在特定时间出现，因此训练集很可能无法涵盖所有的正常执行路径和 logkey，这也是该数据集被选用的原因。

对比实验分为两组，分别使用数据集中前 10%和前 20%的正常日志条目训练模型，其余数据作为测试集。模型更新使用训练好的模型检测异常，每当发现检测到的结果为假阳性时，使用该结果的输入输出序列更新模型；不使用模型更新的情况则只进行异常检测，不对模型做任何增量更新。由于样本数据集中正常日志数据大幅度重复且种类较少，因此本次实验使用单层 GRU 异常检测模型（L=1），窗口大小 h 设定为 3，g 设定为 5。为防止产生欠拟合，存储单元数量设定为 256，多次实验证实该参数设置下异常检测准确率最高。

表 5 给出了使用和不使用模型更新两种情况下的 FP、FN、TP，其中 N 表示不使用模型更新，Y 表示使用模型更新。结合图 10 中的其他性能指标，可以看到在 10%训练数据下，存在较多的假阳性误报，模型的精确率和 F 值非常低；将训练数据扩大到 20%，精确率和 F 值有了一定提升，但提升幅度并不大；在经过模型更新后，检测结果的假阳性大幅度减少，精确率和 F 值显著提升。

表 5　使用模型更新和不使用模型更新性能对比结果

模型情况	假阳性	假阴性	真阳性
N(10%)	804 961	0	155 613
N(20%)	616 215	0	123 539
Y(10%)	20 619	0	155 613
Y(20%)	14 648	0	123 539

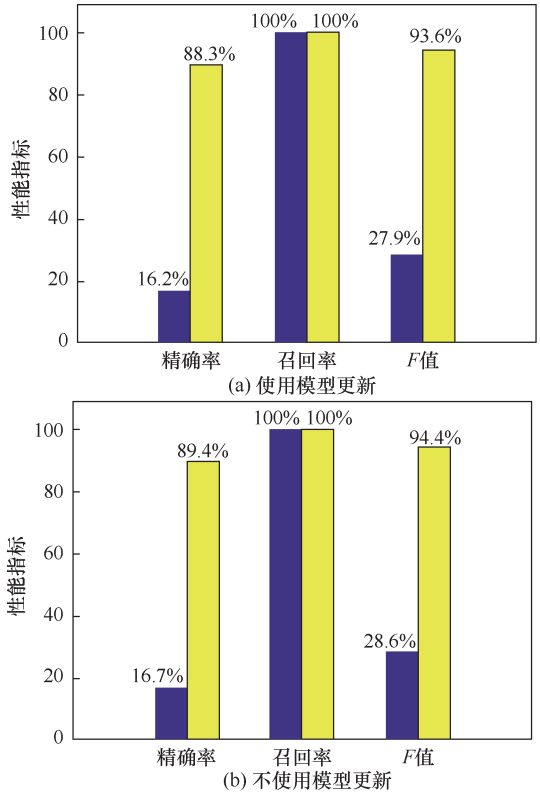


图 10　模型使用模型更新和不使用模型更新性能对比

实验证明了模型更新算法的有效性，经过模型更新，模型的检测精度大大提升。在使用 10%的正常日志作为训练集的情况下，模型更新将模型的精确率提高了 72.1%；在使用 20%的正常日志作为训练集的情况下，模型更新将模型的精确率提高了 72.7%。模型更新和异常检测能够并行

执行，在使用当前权重参数执行异常检测的同时，模型可以进行模型更新，因此，模型更新并不会增加过多时间成本。

4　结束语

当前日志异常检测领域中涉及深度学习的研究相对较少。本文研究针对日志异常检测领域的薄弱点，提出了一种基于 GRU 的日志异常检测算法，使用 Spell 解析日志，从 log key 和参数值 2 个角度构建了 2 个检测模型。模型的训练以 BPTT 算法为基础，使用梯度下降法更新权重参数。在模型的基础上提出一种模型更新策略，使模型可不断学习新的日志规则。实验结果表明，本文算法在 HDFS 大型日志数据集上表现优异，精确率和召回率优于当前前沿的日志异常检测方法。此外，本文针对参数变化对模型性能的影响进行分析，并验证了模型更新策略的有效性。本文为今后相关工作提供了算法参考和模型构建基准，具有一定理论指导意义。面对日志数量巨大，日志规则复杂的现状，本文研究有较高的应用价值。

参考文献：

[1] XU W, HUANG L, FOX A, et al. Detecting large-scale system problems by mining console logs[C]//ACM SIGOPS 22nd symposium on Operating systems principles. 2009: 117-132.

[2] YU X, JOSHI P, XU J, et al. CloudSeer: workflow monitoring of cloud infrastructures via interleaved logs[J]. ACM Sigarch Computer Architecture News, 2016, 44(2):489-502.

[3] DU M, LI F, ZHENG G, et al. Deeplog: anomaly detection and diagnosis from system logs through deep learning[C]//2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 1285-1298.

[4] JURAFSKY D. Speech & language processing[M]. Pearson Education India, 2000:35-61.

[5] DU M, LI F. spell: Streaming parsing of system event logs[C]//IEEE 16th International Conference on Data Mining (ICDM). 2016: 859-864.

[6] ZHU J, HE S, LIU J, et al. Tools and benchmarks for automated log parsing[J]. arXiv preprint arXiv:1811.03509, 2018.

[7] HE P, ZHU J, HE S, et al. An evaluation study on log parsing and its use in log mining[C]//46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). 2016: 654-661.

[8] KINGMA D P, BA J. Adam: a method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[9] MC MAHAN H B, HOLT G, SCULLEY D, et al. Ad click prediction: a view from the trenches[C]//The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013: 1222-1230.

[10] HE S, ZHU J, HE P, et al. Experience report: system log analysis for anomaly detection[C]//IEEE 27th International Symposium on Software Reliability Engineering (ISSRE). 2016: 207-218.

[11] XU W, HUANG L, FOX A, et al. Online system problem detection by mining patterns of console logs[C]//Ninth IEEE International Conference on Data Mining. 2009: 588-597.

[12] OLINER A, STEARLEY J. What supercomputers say: a study of five system logs[C]//37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). 2007: 575-584.

[作者简介]

王易东（1996− ），男，山东济宁人，中国海洋大学硕士生，主要研究方向为信息安全、云计算和大数据。

刘培顺（1975− ），男，山东菏泽人，中国海洋大学讲师，主要研究方向为网络与信息安全。

王彬（1981− ），男，山东沾化人，中国海洋大学实验师，主要研究方向为计算机应用技术。