G-07

BI-LSTM を用いた実行時ログの異常検出

Anomaly detection in runtime logs using BI-LSTM

信藤 海南登 †　　中本 幸一 †
Minato Shindo　Yukikazu Nakamoto

1. はじめに

システムの不具合やプログラムエラーを特定するのは容易ではない．それらを解決するのに有効的な情報としてシステムの実行時ログが考えられる．しかし，システムが大規模になるほどログから不具合を見つけるには時間を要する．そのため，機械学習の教師あり学習である Bidirectional Long Short Term Memory(以降 BI-LSTM と呼ぶ) を用い，不具合の一例としてデッドロックを取り上げ，プログラム修正に必要な手がかりを開発者に提示できるようにした．開発時間の短縮や運用時の不具合の解決に役立たせたいと考えている．

2. 背景

システムの不具合の特定やエラーの解決をするためにログデータは有効的である．しかし，システムが大規模ほどログデータは複雑かつ大量であるため，人間の目で不具合を特定することは時間がかかり，また困難な場合もある．そのため，開発時間の短縮や不具合の見落としを減らしていくために検出を自動化する必要がある．その実現のため本研究では株価予測や感情分析などで用いられる BI-LSTM を用いて不具合の特定を行う．BI-LSTM は過去から未来から予測を行う LSTM と違い，過去から未来と未来から過去から予測を行うことができる．それにより一方向 LSTM よりも文脈の理解ができるため，昨今文章自動作成ツールとしても活用されている．

2.1 関連研究

2017 年，DeepLog[1] と呼ばれるログアナライザが開発された．DeepLog は過去から未来の一方向の LSTM を扱っている．ログの出現パターンを学習後，システムの不具合箇所やエラーの根本原因を見つけることができる．2021 年，LogBERT[2] と呼ばれる BERT を用いたログ分析手法が登場した．LogBERT は，DeepLog の一方向の LSTM とは違い，過去から未来と未来から過去のログから推論することができるため DeepLog よりも F 値 (後に説明する再現率と適合率の調和平均) を上回ることに成功している．そのため，本稿において過去から未来と未来から過去の双方向から予測できる BI-LSTM を用いることによりプログラムの不具合を発見することができるのではないかと考えた．

† 兵庫県立大学，University of Hyogo

3. データの前処理と学習方法

3.1 5 人の哲学者問題について

5 人の哲学者とは，並行処理の例である．5 人の哲学者とその間に 5 本の箸があり，2 本の箸を確保できたら食事ができる．哲学者は必ず左から右の順で箸を持ち，食事が終わると左から右の順に箸を離す．哲学者が箸を使用している場合，他の哲学者はその箸が解放されるまで確保することはできない．本研究では，それぞれ哲学者の左側にある箸を哲学者の番号としている．そのため，哲学者は自身に割り当てられた番号の箸を手にする．このプロセスは，テーブルに設置してある食料がなくなるまで繰り返される．

3.2 デッドロック

デッドロックとは，複数のプロセスが結果待ちとなり待機状態に入ったままプログラムが動かなくなる現象である．そのため，デッドロックが起こらないようにコーディングするのが定石である．デッドロックの条件を下記に示す．

- 循環待機
- 相互排除
- 横取り不能
- 確保と待機

上記の条件を全て満たした場合，デッドロックが成立する．

図 1 は哲学者をスレッドで実現したプログラムでデッドロックが起きた時の実行ログである．図 1 のデッドロックの要因は，5 人が左手に箸を一本ずつ所持しているからである．そのため，お互いに箸が解放されるのを待ち合っているためこれ以上動くことができない．本研究ではある時点でデッドロックが起きている箇所を検出することを目標としている．また，それらを検知することによりデッドロックのリスクを減らせるのではないかと考えた．

```
The philosopher 2 grabs the left chopstick No.2. time : 0.000117
The philosopher 0 grabs the left chopstick No.0. time : 0.000117
The philosopher 4 grabs the left chopstick No.4. time : 0.000119
The philosopher 1 grabs the left chopstick No.1. time : 0.000121
The philosopher 3 grabs the left chopstick No.3. time : 0.000121
```

図 1 デッドロックが起きた時の実行ログ

3.3 データの前処理

本研究で使用するデータは，5 人の哲学者問題のプログラム

作成後，実行時のログを取り扱った．ログメッセージから固定部分と可変部分を切り分ける．固定部分をログキー (またはログタイプ) と呼ぶ．また，可変部分をパラメータと呼ぶ．本研究では，ログキーを用いて分析を行う．

ログキーをコンピュータが扱いやすいようにするため one-hot ベクトルに変換する．one-hot ベクトルとは，1 つの要素が 1 でその他の要素が全て 0 のベクトルである．例えば，[0， 1， 0， 0， 0] などが考えられる．

3.4 LSTM

LSTM とは，Long Short Term Memory の略である．LSTM は，長期的な時系列の依存関係を学習することができる人工的な RNN のアーキテクチャである．典型的な LSTM は，セル，入力ゲート，出力ゲート，忘却ゲートから構成される．一方向の LSTM は，過去から未来への一方向の層しか持たない．そのため，予測は過去の情報に依存する．

3.5 BI-LSTM

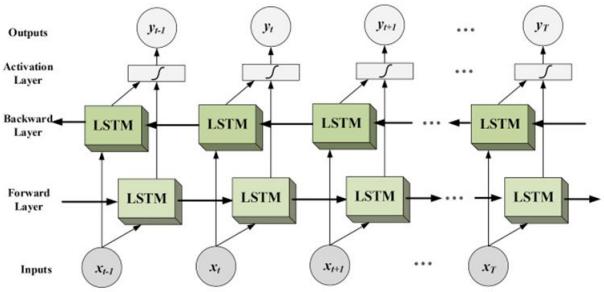


図 2 BI-LSTM のアーキテクチャ [3]

BI-LSTM は，一方向 LSTM と違い過去から未来と未来から過去の二方向から推論を行うため，昨今自然言語処理や音声認識などのアプリケーションに適用されている．BI-LSTM の層は，上記の通り過去から未来と未来から過去の 2 種類の層を持っている．その例を示しているのが図 2 である．BI-LSTM を使う背景は，ログ分析において過去から未来の一方向のみではログメッセージ間の相関関係が崩れる危険があり，前のログメッセージに基づいて次のログメッセージを正確に推論ができないためである [2].

3.6 学習方法，BI-LSTM と LSTM の学習率の比較

本研究で取り扱う学習データは，非デッドロックのログデータである．非デッドロックデータとは，デッドロックが発生していないログデータである．データ数は 1258 データとしている．非デッドロックデータを取り扱う理由は，DeepLog や LogBERT などのアナライザにおいて不具合が生じないログデータを学習データを扱っているためである．学習方法において，3.3 節の入力方法より h × n の行列としている．h は最新のログ履歴であり，次に例を示す．{k1， k2， k3， k4， k5， k6} のログキーの並びがあるとする．h=3 において，入力配列と出力ラベルのペアは，[k1， k2， k3] を入力した場合 k4 が出力ラベルになり，[k2， k3， k4] の入力時 k5 が出力ラベル，[k3， k4， k5] を入力時 k6 が出力ラベルとなる．

実行環境を以下に示す．

- CPU:Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz
- GPU:NVIDIA GeForce RTX 3060
- Python/Keras

LSTM の学習においてエポック数 100，バッチ数 32，LSTM 層の数は 1，LSTM のユニット数は 250 と設定している．その学習率を図 3 に示す．青の線が精度，赤の線が損失を示している．学習時間は 3629 秒であった．また，100 エポック終えての精度は 50%，損失は 9%であった．

BI-LSTM の学習において，エポック数を 100，バッチ数を 32，BI-LSTM の層の数を 1，BI-LSTM のユニット数を 250 に設定している．学習時間は 5638 秒であった．その学習率を図 4 に示す．100 エポック終了時の結果は，精度が約 87%，損失が約 3%の精度となった．

LSTM と BI-LSTM を比較すると，LSTM は BI-LSTM よりも学習時間は短い結果となったが，精度を比較すると約 37%の差が生じた．

このためこれ以下 BI-LSTM について述べる．検証時の h の設定において，h=3~20 でテストをした結果最も良い精度だった h=17 で検証を行なった．h の数を増やすことにより学習率は上昇するが学習時間などの計算時間がかかってしまうことや調査する範囲が広くなるため異常箇所の特定に時間がかかる問題が考えられる．実際に h=1 と h=17 の学習時間を比較した場合，次に述べる実行環境では 1042 秒の差が生じた．そのため，適切に h の数を設定する必要がある．

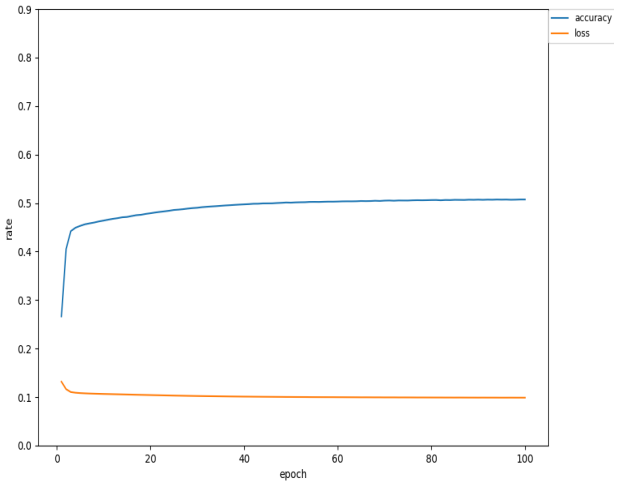


図 3 LSTM における精度と損失

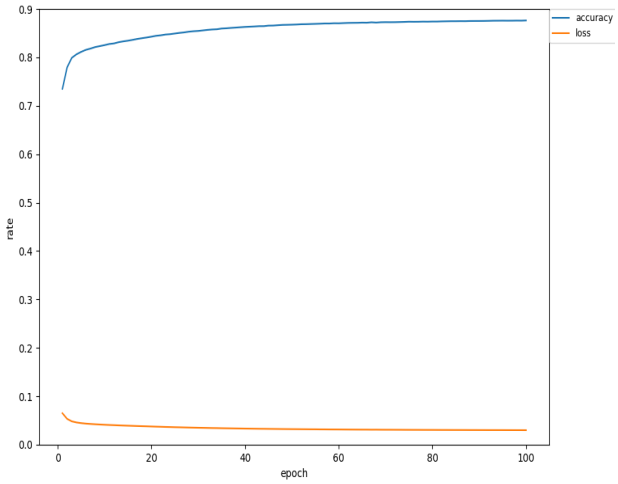


図 4　BI-LSTM における精度と損失

4.　検証

4.1　検知方法

BI-LSTM への入力は 3.6 節の通り h × n の行列である．出力は，次の時間に遷移した時に予測されるログキーの n 次元確率分布である．トップ g 番目までに実際のログキーが存在した場合学習データに沿っている出現パターンと診断する．デッドロックと診断するとは，実際のログキーがトップ g 番目以外の時で，この場合デッドロックとなるログキーが存在しているとみなす．

検証するパラメータを示す．BI-LSTM の層は 1 層, BI-LSTM のユニット数は 250，h の値は 17，g の値は 4 としている．

検証データにおいて，デッドロックデータが 50 と非デッドロックデータが 50 の合計 100 データを検証する．また，本研究では評価をしやすくするため 1 つのログファイルの中に一つでも異常検出された場合，そのファイルをデッドロックが発生したとみなす．

5.　結果と考察

表 1　混同行列

		モデルの予測	
		デッドロック	非デッドロック
実際のクラス	デッドロック	35	15
	非デッドロック	5	45

表 2　4 つの評価指標の結果

適合率	正解率	F 値	再現率
0.88	0.80	0.78	0.70

4.1 節で述べた方法から結果を導き出したのが，表 1 の混同行列である．また，表 1 の混同行列を基に正解率，再現率，適合率，F 値を求めた結果を表 2 に示している．表 1 より非デッドロックの正解率が 90%を達成した．また，デッドロックの正解率は 70%の結果となった．表 2 より 4 つの評価指標のうち最も高い数値だったのは，適合率であった．適合率はデッドロックと診断した中で実際にデッドロックであった割合を示している．このことから 88%正確にデッドロック発生の予測ができている．一方，4 つの中で最も悪い結果は再現率である．数値は 70%である．再現率は実際デッドロックが発生したうちデッドロックと診断できた確率を示している．

6.　今後の課題

再現率の数値が悪かったため，更に精度を上げていけるように工夫をしていきたい．本稿で取り上げた内容では，デッドロックを引き起こした怪しい箇所を示すことができてもその事象が発生した原因を提示することができない．そのため，今後は原因を提示できるような有効的なツールを作成したいと考えている．

また，本稿で紹介した手法を用いて既存システムログの分析において成果を上げたいと考えている．

参考文献

[1] Du, M., Li, F., Zheng, G., Srikumar, V.: DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. (2017).

[2] Haixuan Guo, Shuhan Yuan, Xintao Wu, LogBERT: Log Anomaly Detection via BERT. In: 2021 International Joint Conference on Neural Networks (IJCNN). (2021).

[3] Yugesh Verma, Complete Guide To Bidirectional LSTM, 入手先 : `https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/` 参照 (2022-7-6).