

Learning to Handle Exceptions

例外処理の学習

Jian Zhang*

SKLSDE Lab, Beihang University,
China

zhangj@act.buaa.edu.cn

Xu Wang*[†]

SKLSDE Lab, Beihang University,
China

wangxu@act.buaa.edu.cn

Hongyu Zhang

The University of Newcastle,
Australia

hongyu.zhang@newcastle.edu.au

Hailong Sun*

SKLSDE Lab, Beihang University,
China

sunhl@act.buaa.edu.cn

Yanjuan Pu

SKLSDE Lab, Beihang University,
China

puyanjuan@nlsde.buaa.edu.cn

Xudong Liu*

SKLSDE Lab, Beihang University,
China

liuxd@act.buaa.edu.cn

- JavaやC#などの最新のプログラミング言語では，堅牢で信頼性の高いソフトウェアを開発するために，**例外処理メカニズム**が存在^[43, 58]

- 例外処理メカニズムとは

```
1 public char[] decrypt(char[] chars) {
2     if (!isEncrypted(chars)) {
3         return chars;
4     }
5     String encrypted = new String(chars, ENCRYPTED_TEXT_PREFIX.length(),
6                                     chars.length - ENCRYPTED_TEXT_PREFIX.length());
7     byte[] bytes;
8     ① try {
9         bytes = Base64.getDecoder().decode(encrypted);
10    }
11    ② catch (IllegalArgumentException e) {
12        throw new ElasticsearchException("unable to decode encrypted data", e);
13    }
14    byte[] decrypted = decryptInternal(bytes, encryptionKey);
15    return CharArrays.utf8BytesToChars(decrypted);
16 }
```

実行時に発生する可能性のある
例外(異常な状態, 予期しない状態)
に対し，そこから**回復するための**
処理を事前に記述^[22, 25]

[43] Robert Miller and Anand Tripathi. 1997. Issues with exception handling in object-oriented systems. In European Conference on Object-Oriented Programming. Springer, 85–103.

[58] Suresh Thummalapenta and Tao Xie. 2009. Mining exception-handling rules as sequence association rules. In 2009 IEEE 31st International Conference on Software Engineering. IEEE, 496–506.

[22] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. 2000. The Java language specification. Addison-Wesley Professional.

[25] Anders Hejlsberg, Scott Wiltamuth, and Peter Golde. 2003. C# language specification. Addison-Wesley Longman Publishing Co., Inc.

- しかし、多くの産業用システム^[10, 18, 21]やOSS^[4, 46]において、
例外処理の記述が不十分
- 原因
開発者は例外処理のエラーが発生するまで、
例外処理が必要かどうかについての知識がほとんどない^[56, 57]
- 先行研究^[19]
開発者を支援するため、例外処理コードの例、例外の種類などの推奨事項を提示
ただし、開発者は例外が発生する可能性のある箇所を自分で考え、
コードのコンテキストに合わせてコード例をカスタマイズする必要

[10] Bruno Cabral and Paulo Marques. 2007. Exception handling: A field study in java and. net. In European Conference on Object-Oriented Programming. Springer, 151–175.

[18] Roberta Coelho, Awais Rashid, Alessandro Garcia, Fabiano Ferrari, Nélío Cacho, Uirá Kulesza, Arndt von Staa, and Carlos Lucena. 2008. Assessing the impact of aspects on exception flows: An exploratory study. In European Conference on Object-Oriented Programming. Springer, 207–234.

[21] Alessandro F Garcia, Cecilia MF Rubira, Alexander Romanovsky, and Jie Xu. 2001. A comparative study of exception handling mechanisms for building dependable object-oriented software. Journal of systems and software 59, 2 (2001), 197–222.

[4] Muhammad Asaduzzaman, Muhammad Ahasanuzzaman, Chanchal K Roy, and Kevin A Schneider. 2016. How developers use exception handling in Java?. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 516–519.

[46] Ana Filipa Nogueira, José CB Ribeiro, and Mário A Zenha-Rela. 2017. Trends on empty exception handlers for Java open source libraries. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 412–416.

[56] Hina Shah, Carsten Görg, and Mary Jean Harrold. 2008. Why do developers neglect exception handling?. In Proceedings of the 4th international workshop on Exception handling. 62–68.

[57] Hina Shah, Carsten Görg, and Mary Jean Harrold. 2010. Understanding exception handling: Viewpoints of novices and experts. IEEE Transactions on Software Engineering 36, 2 (2010), 150–161.

[19] Guilherme B de Pádua and Weiyl Shang. 2018. Studying the relationship between exception handling practices and post-release defects. In Proceedings of the 15th International Conference on Mining Software Repositories. 564–575.

目的とアプローチ

目的

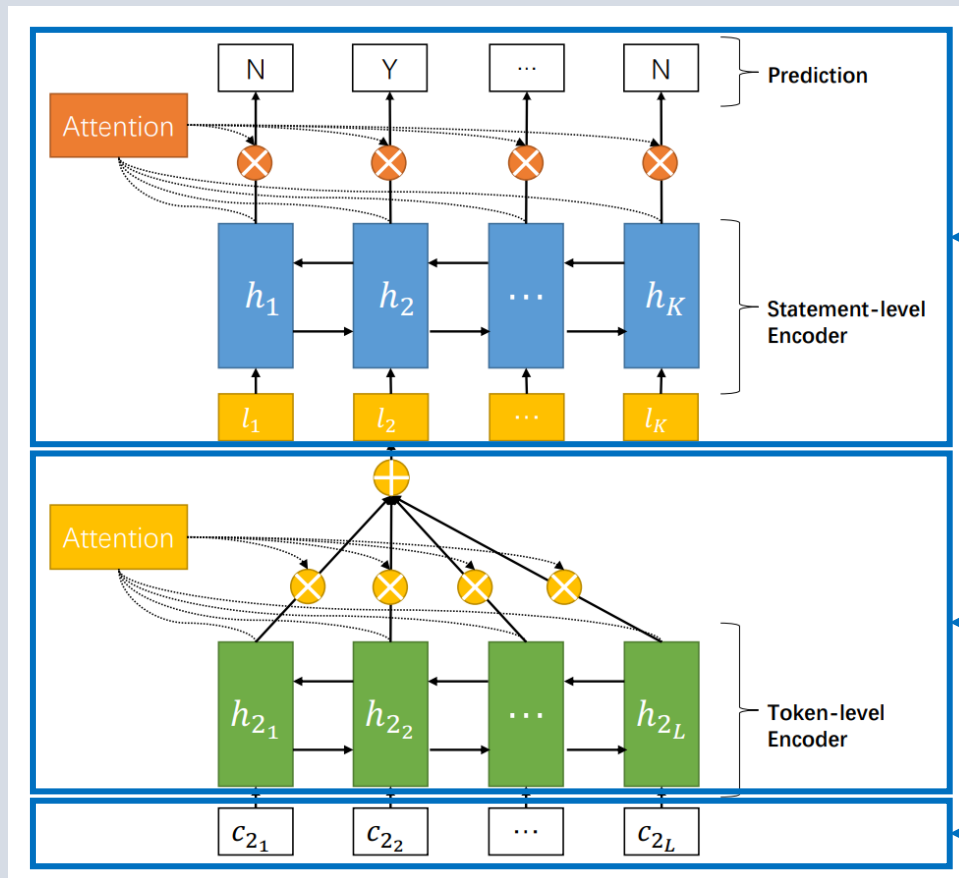
例外処理コード自動生成のための深層学習モデルNexgenを構築

アプローチ

- ・ ソースコード内で、例外処理の必要がある箇所を特定（位置特定タスク）
- ・ 例外に対処するコードを生成（生成タスク）

位置特定タスク

与えられたコード片に対して、文単位で例外処理が必要かどうかをタグ付け



③別のAttention-BiLSTMモデルに、コード片の文ベクトルシーケンスを入力し、
文ごとに例外処理が必要かどうかを予測

②Attention-BiLSTMモデルに、文ごとにトークンシーケンスを入力し、コンテキスト情報を文の特徴ベクトルに埋め込む

①コード片を字句解析し、各文をトークン化

- GitHub上のスター数とフォーク数の合計が上位2,000件のJavaプロジェクトを収集
- 少なくとも1つのtry-catchブロックを含むメソッドを抽出
→ 43万件のコード片を抽出
- 同数のtry-catchブロックを持たないメソッドをランダムサンプリングし、負例とする

位置特定タスクの評価

提案モデルとベースラインモデルの分類性能を比較

モデル名	説明
CRF (Conditional Random Field)	重み付き無向グラフによる確率モデル
BiLSTM	BiLSTMのみを使ったモデル
BiLSTM-CRF	BiLSTMの各シーケンスデータの出力を，CRFに入力するモデル
Seq2Seq	Attention付きEncoder-Decoderモデル

Models	Precision	Recall	F1-score	Accuracy
CRF	64.7	26.9	38.0	52.4
BiLSTM	73.7	70.6	72.1	67.5
BiLSTM-CRF	51.6	59.2	55.2	54.4
Seq2Seq	77.1	60.0	67.5	48.7
<i>Nexgen</i>	80.9	74.3	77.4	74.7

提案モデルのスコアが最も高い

位置特定タスクの評価 | Attentionの効果

提案モデルの方がベースラインよりも性能が高い理由を理解するため、
Attention層を削除するアブレーションスタディ

Task1				
Description	Precision	Recall	F1-score	Accuracy
– Token Attention	79.3	74.8	76.9	74.1
– Statement Attention	79.5	74.9	76.6	73.8
– Both Attention	78.9	74.3	76.1	73.0
<i>Nexgen</i>	80.9	74.3	77.4	74.7

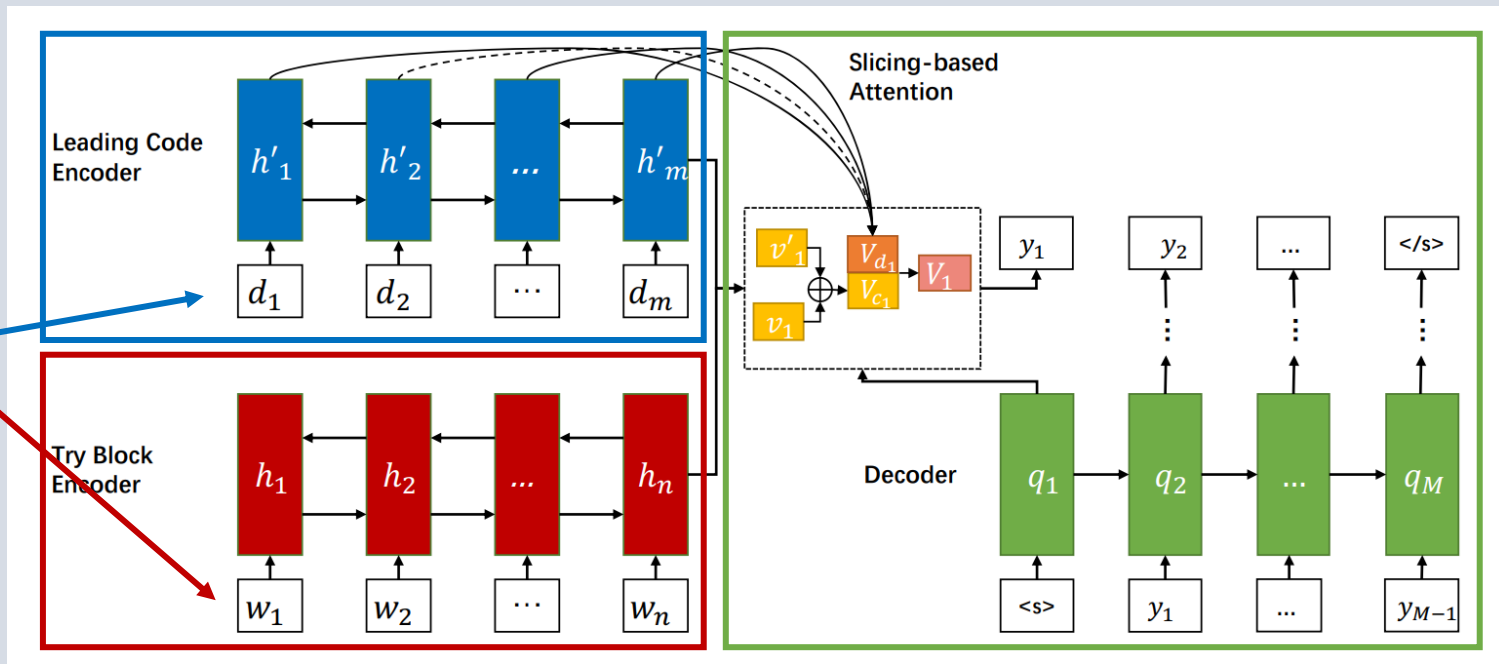
- 各メトリクスで最大2%程度差があるが、ベースラインとの差の方が大きい
- ベースラインとの性能差は、Attentionの有無というより、
ベースラインのどれにも実装されていなかった、
文レベルの特徴ベクトルの埋め込みが効いている

生成タスク

与えられたコード片に対して，例外に対処するコードを生成

③それより前のコードブロックをAttention-BiLSTMに入力

- ①コード片を，
- 例外処理が必要なコードブロック
 - それより前のコードブロック
- に分割
(次スライドでねらいを説明)



- ②例外処理が必要なコードブロックを BiLSTMに入力
- ④②のBiLSTMの状態と，
③のAttentionの重みづけをもとに，
Decoderからトークンを1つずつ出力

生成タスク

例外処理が必要なコードブロックと、それより前のコードブロックに分けて学習する理由

```
1 public char[] decrypt(char[] chars) {  
2     if (!isEncrypted(chars)) {  
3         return chars;  
4     }  
5     String encrypted = new String(chars, ENCRYPTED_TEXT_PREFIX.length(),  
6                                     chars.length - ENCRYPTED_TEXT_PREFIX.length());  
7     byte[] bytes;  
8     bytes = Base64.getDecoder().decode(encrypted);  
9     byte[] decrypted = decryptInternal(bytes, encryptionKey);  
10    return CharArrays.utf8BytesToChars(decrypted);  
11 }
```

- 例外処理が必要なコードブロック → 例外処理の内容に最も関連

- それより前のコードブロック

- 変数encryptedの内容によって例外が発生するので、encryptedに関連するコードも重要
- ただし、無関係のコードも多く含まれておりノイズになる可能性
→ Attentionでマスキングする必要

⇒ 分けたほうが学習しやすい

生成タスクの評価

データセットは位置特定タスクと同様．ベースラインモデルと分類性能を比較

モデル名	説明
N-gram	N-gram(この論文ではN = 6)単位で次に生成するトークンを予測する確率モデル
LSTM	LSTMのみを使ったモデル
Seq2Seq	Attention付きEncoder-Decoderモデル

・ 評価指標：BLEU

N-gram(この論文ではN = 4トークンで計算)単位での正解データと生成データの重複率

Models	BLEU	Accuracy
N-gram	5.0	0.0
LSTM	29.4	8.1
Seq2Seq	42.6	20.7
<i>Nexgen</i>	46.7	22.6

提案モデルのスコアが最も高い

位置特定タスクと同様に，Attention層を削除するアブレーションスタディを行って，提案モデルの方がベースラインよりも性能が高い理由を分析

Task2		
Description	BLEU	Accuracy
– Leading Attention	45.4	22.1
– Slicing Attention	44.2	22.0
– Both Attention	30.2	14.5
<i>Nexgen</i>	46.7	22.6

- Attentionを全く使わないモデルとの性能差が大きい
 - ⇒ 入力されるコード片には例外処理に無関係のコードが多く含まれており，
Attentionを使うことで，こうしたノイズをマスキングできている

目的

例外処理コード自動生成のための深層学習モデルNexgenを構築

アプローチ

- ・ ソースコード内で，例外処理の必要がある箇所を特定（位置特定タスク）
- ・ 例外に対処するコードを生成（生成タスク）

結果

- ・ 位置特定タスク，生成タスクの両方で，ベースラインを上回るスコアを達成
- ・ 提案モデルが一般的な深層学習モデルと比べて効果的である理由を，アブレーションスタディによって確認