Knowledge-Based Systems 210 (2020) 106529

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description

Jiao Yin [a,b], MingJian Tang [c], Jinli Cao [a,*], Hua Wang [d]

[a] Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC, 3083, Australia
[b] School of Artificial Intelligence, Chongqing University of Arts and Sciences, Chongqing, 402160, China
[c] Huawei Technologies Co. Ltd, Shenzhen, 518129, China
[d] Institute for Sustainable Industries & Liveable Cities, Victoria University, Melbourne, VIC, 3083, Australia

ARTICLE INFO

Article history:
Received 25 May 2020
Received in revised form 13 September 2020
Accepted 12 October 2020
Available online 15 October 2020

Keywords:
Transfer learning
Exploitability prediction
ExBERT
Software vulnerability

ABSTRACT

Thousands of software vulnerabilities are archived and disclosed to the public each year, posing severe cybersecurity threats to the whole society. Predicting the exploitability of vulnerabilities is crucial for decision-makers to prioritize their efforts and patch the most critical vulnerabilities. Software vulnerability descriptions are accessible features in early stage and contain rich semantic information. Therefore, descriptions are wildly used for exploitability prediction in both industry and academia. However, comparing with other corpora, the size of vulnerability description corpus is too small to train a comprehensive Natural Language Processing (NLP) model. To gain a better performance, this paper proposes a framework named ExBERT to accurately predict if a vulnerability will be exploited or not. ExBERT essentially is an improved Bidirectional Encoder Representations from Transformers (BERT) model for exploitability prediction. First, we fine-tune a pre-trained BERT using collected domain-specific corpus. Then, we design a Pooling Layer and a Classification Layer on top of the fine-tuned BERT model to extract sentence-level semantic features and predict the exploitability of vulnerabilities. Results on 46,176 real-word vulnerabilities have demonstrated that the proposed ExBERT framework achieves 91.12% on accuracy and 91.82% on precision, outperforming the state-of-the-art approach with 89.0% on accuracy and 81.8% on precision.

1. Introduction

1.1. Motivations

A vulnerability is a weakness in a system, for example, in procedures, design or implementation, which might be exploited to cause loss or harm [1]. Historical vulnerability information is of great significance for vulnerability assessment, patching and exploitability prediction. Therefore, many companies and organizations spend a lot of resources to collect and maintain vulnerability databases. Among them, two of the most famous databases are the Common Vulnerabilities and Exposures (CVE)[1] maintained by The MITRE Corporation[2] and the National Vulnerability Database (NVD)[3] preserved by National Institute of Standards and Technology (NIST), U.S. Department of Commerce.[4]

More than 130,000 vulnerabilities have been disclosed to public through open-source databases by the end of 2019. Leyla Bilge and Tudor Dumitras pointed out that once a vulnerability is disclosed, the opportunity of being exploited increases by 5 orders of magnitude [2]. Facing an increasing number of vulnerabilities, it is impossible to patch every vulnerability timely. Considering writing, testing and installation costs of patching software vulnerabilities, the prediction of vulnerability exploitability is of importance for decision-makers to prioritize their efforts and patch the most critical vulnerabilities before being exploited [3]. This paper focuses on the prediction challenge to protect the loss of both companies and organizations.

1.2. Challenges

1.2.1. Limitations of common vulnerability scoring system (CVSS)

In industry, CVSS [4] is used as the de facto standard for assessing the severity of security vulnerabilities by a wide range of networking and software vendors, such as Cisco, Oracle and Microsoft. CVSS provides a list of metrics whose values are assigned by a panel of analysts and security experts based on the disclosed information of vulnerabilities [5]. Using a carefully

* Corresponding author.
E-mail address: j.cao@latrobe.edu.au (J. Cao).

1 https://cve.mitre.org/
2 https://www.mitre.org/
3 https://nvd.nist.gov/vuln/data-feeds
4 https://www.nist.gov/

https://doi.org/10.1016/j.knosys.2020.106529
0950-7051/

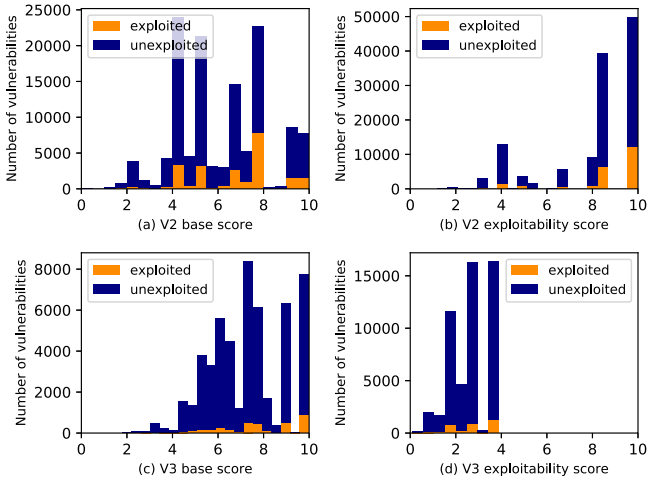

Fig. 1. The distribution of four scored CVSS metrics.

designed formula, CVSS combines the values of all metrics into a final score ranging from 0.0 to 10.0 to represent the overall risk of a given vulnerability, where 10.0 is the most severe level.

We visualize the distribution of four scored metrics of CVSS, namely, (a) V2 base score, (b) V2 exploitability score, (c) V3 base score and (d) V3 exploitability score on all existing vulnerabilities in NVD as of 2019 in Fig. 1, where V2 and V3 represent two different versions of CVSS. The dark orange bars stand for the number of exploited vulnerabilities while the navy bars are unexploited vulnerabilities. In the CVSS scoring system, the bigger the score, the higher the risk. Therefore, unexploited vulnerabilities should be in low risks and low scores. However, as shown in Fig. 1, none of the four CVSS metrics is reliable for indicating the exploitability of vulnerabilities. For example, in subplot (b), many unexploited vulnerabilities with low risks are scored 10.

Previous studies also pointed out that CVSS scores are very poor indicators for predicting the exploitability of vulnerabilities [6–8]. Besides, highly specialized knowledge is required to calculate CVSS scores [9]. Another concern on CVSS is the time delay between the publication of vulnerabilities and their available CVSS scores [10]. Therefore, better methods are expected for inventors and decision-makers to predict the exploitability of vulnerabilities.

1.2.2. Limitation of previous studies

In academic community, researchers have been trying to build more accurate exploitability prediction models, leveraging the emerging machine learning and deep learning techniques [11–13]. To reduce dependence on domain knowledge and time delay problems existing on CVSS, an increasing number of scholars assess vulnerabilities and predict exploitability by mining vulnerability descriptions [7,12,14–16]. A vulnerability description is a brief paragraph for each vulnerability, which contains abundant details such as the vulnerability type, names of affected products and vendors, a summary of affected versions, the impact, the access that an attacker requires to exploit the vulnerability and the important code components or inputs that are involved [17]. Based on description contents, it is straightforward that descriptions contain valuable information for exploitability prediction.

Although some results were reported in previous studies, problems are still existing in the following aspects.

(i) Fail to consider the polysemy problem in NLP and special technical terms in cybersecurity. For example, there are two sentences: sentence A ‘A buffer overflow in lsof allows local users to obtain root privilege’ and sentence B ‘Roses will not root in such acid soil’. The meanings of word ‘root’ in sentence A and B are different. Besides, the word ‘lsof’ in sentence A is a term in cybersecurity which means ‘list open files’. Terms like ‘lsof’ are essential to capture the semantic meaning in vulnerability descriptions and should not be ignored. In previous description-based exploitability prediction studies, researchers used the Term Frequency–Inverse Document Frequency (TF–IDF) algorithm [7,14], rule-based statistical method [16,18,19] and word-embedding [14,15] techniques to extract semantic features from descriptions. These feature extraction methods use identical representation or method to deal with the same word or phrase. Therefore, they cannot capture the polysemy of words within different contexts. Besides, cybersecurity domain corpus contains many low-frequency domain-specific words, such as package names, tool names, variable names and other technical terms. None of the previous studies has addressed this problem.

(ii) Fail to consider the dependencies between features extracted from descriptions when choosing classifiers. As sequential texts, dependencies exist in vulnerability descriptions. For example, the meaning of ‘root’ in phrase ‘root privilege’ depends on the word ‘privilege’. Existing studies applied Support Vector Machine (SVM) [7,14,18], Random Forest (RF) [14,20], Naive Bayes [16] and fully-connected Neural Networks (denoted as DenseNN) [15] as the classifiers for exploitability prediction. These classifiers treat features as individuals with no dependency. Therefore, they fail to capture dependencies within each description.

(iii) No publicly accessed unified datasets exist in exploitability prediction community. Publicly accessed unified datasets are a very important factor for developing machine learning or deep learning algorithms. Although researchers collect data from the same open-source dataset, i.e. NVD and ExploitDB,[5] the chosen vulnerabilities, the exploitability status at a specific time and the total amount of vulnerabilities vary in different studies. To the best of our knowledge, no publicly accessed unified dataset exists for exploitability prediction so far. The major reason is that vulnerability-related databases are dynamically increasing and the exploitability of vulnerabilities also changes over time. Researchers always try to obtain the latest data to support their research. As a result, all existing studies use self-collected datasets when predicting exploitability of vulnerabilities [7,14–16,18].

1.3. Contributions

To achieve better prediction performance, we propose a novel exploitability prediction framework ExBERT based on transfer learning [21]. It can not only learn sentence-level semantic features from descriptions but also capture the long dependencies within inputs using an Long-Short Term Memory (LSTM) classifier. Compared with previous works, our main contributions can be summarized as follows.

(i) We are the first to apply transfer learning to BERT model to extract changeable token embeddings from vulnerability descriptions. Inspired by transfer learning, we fine-tune a pre-trained BERT model using vulnerability descriptions instead of training a BERT from scratch, to overcome the insufficiency of domain corpus. BERT model provides different token embeddings for the same token according to its context, which means it can understand word polysemy. Besides, we adopt wordpiece tokenization algorithm to deal with low-frequency cybersecurity technical terms to gain a better semantic feature.

(ii) We design a Pooling Layer on top of the fine-tuned BERT to extract comprehensive sentence-level semantic features instead of using token-level features directly. The concept of pooling

[5] https://www.exploit-db.com/

layer is originally used in Convolutional Neural Networks (CNN) to reduce dimensionality and extract high-level features. In this paper, we introduce a Pooling Layer to extract sentence-level semantic features from token-level features extracted from the fine-tuned BERT model.

(iii) We are the first to apply LSTM model as the classifier in exploitability prediction, instead of using SVM, RF, CNN, et al. Descriptions are sequential inputs with long term dependencies, Recurrent Neural Networks (RNN) and its variation LSTM are proven good at capturing the long term dependencies of sequential data and dealing with natural language [22]. Experimental results show that our methods using LSTM classifier can achieve better performance in exploitability prediction.

The rest of the paper is organized as follows. Section 2 introduces related work. Section 3 formulates the exploitability prediction problem and clarifies its optimization objective. Section 4 elaborates the proposed ExBERT framework, followed by experimental results and analysis in Section 5. Finally, a conclusion is provided in Section 6.

2. Related works

We briefly review and present the most related techniques in this section.

2.1. Transfer learning

Transfer learning is an emerging learning strategy in machine learning (ML) that focuses on learning and storing knowledge gained from one domain and applying it to a different but related domain [23,24]. A Domain $\mathcal{D}$ consists of two components: a feature space χ and a marginal probability distribution $P(X)$, where $X = x_1, \ldots, x_n \in \chi$. In general, if two domains are different, then they may have different feature spaces χ or different marginal probability distributions $P(X)$. Given a specific domain, $D = \{\chi, P(X)\}$, a task $\tau = \{\mathcal{Y}, f(\cdot)\}$ consists of two components: a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$, which can be written as $P(y|x)$. Labels can be $y = \{0, 1\}$ in binary classification. The predictive function $f(\cdot)$ is not observed, but can be estimated from the training data $\{x_i, y_i\}$ $(x_i \in X, y_i \in \mathcal{Y})$.

We further denote $D_s = (x_{s_1}, y_{s_1}), \ldots, (x_{s_n}, y_{s_n})$ as the source domain, where $x_{s_i} \in \chi_s$ and $y_{s_i} \in \mathcal{Y}_s$ representing features and corresponding labels. Similarly, the target domain is denoted as $D_t = \{(x_{t_1}, y_{t_1}), \ldots, (x_{t_m}, y_{t_m})\}$, where $x_{t_i} \in \chi_t$ and $y_{t_i} \in \mathcal{Y}_t$. In general, $0 < n \ll m$.

Given all the defined notations, transfer learning is formally defined as [23]:

Definition 1. Transfer Learning: Given a source domain $\mathcal{D}_s$ and a learning task τ_s, a target domain $\mathcal{D}_t$ and a learning task τ_t, transfer learning aims to help improving the learning of the target predictive function $f_t(\cdot)$ in $\mathcal{D}_t$ using the knowledge learnt in $\mathcal{D}_s$ and τ_t, where $\mathcal{D}_s \neq \mathcal{D}_t$ or $\tau_s \neq \tau_t$.

2.2. BERT model

In 2018, Devlin and Chang et al. at Google AI Language proposed an NLP model named BERT for language understanding [25]. It has caused a stir in the deep learning community by achieving the state-of-the-art results on eleven NLP tasks instead of on a single task or domain, including Natural Language Inference (NLI) and Question Answering (SQuAD v1.1). BERT is developed based on a popular Transformer model [26] and it trains a multi-layer Transformer in a bidirectional way.

To enable bidirectional training, BERT defined two training tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). The training process of MLM is shown in Fig. 2.

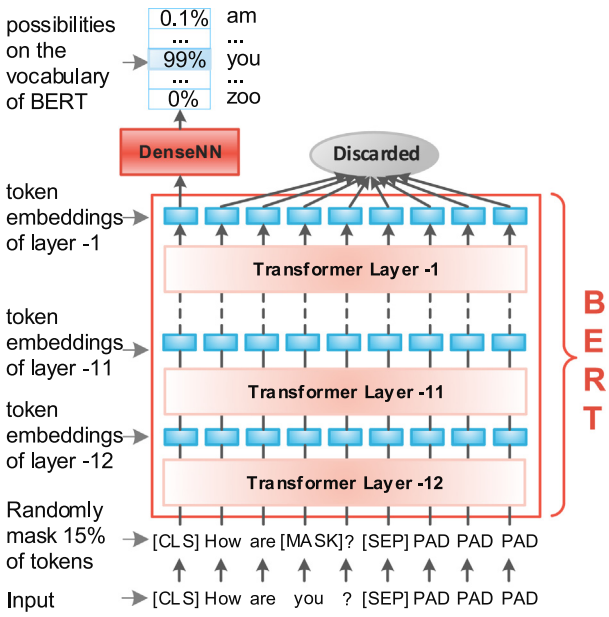

Fig. 2. The training process of Masked Language Model.

Given an input sentence, 15% tokens are randomly replaced with a [MASK] token, then train the BERT model to predict the masked tokens. Tokens [CLS] and [SEP] in Fig. 2 are manually added as the beginning marker and ending marker of an input sentence. BERT consists of 12 stacked Transfer layers, named as Transfer Layer -1, ..., -11, -12. The output of each Transfer Layer corresponding to the token embeddings of the input tokens. For example, the output of the first node in each Transfer Layer is the token embedding of [CLS]. In the original paper [25], only the token embedding corresponding to [CLS] in Transfer Layer -1 is used to predict the value of the masked token, other token embeddings are discarded as shown in Fig. 2. Then, a DenseNN is used as a classifier on top of the [CLS] token embedding to calculate all possibilities on the vocabulary of BERT.

Similarly, the NSP training process is shown in Fig. 3. Given two sentences, [CLS] is added as the beginning of the first sentence and two [SEP] tokens are added at the end of these two sentences. BERT is trained to predict if Sentence B is the subsequent sentence of Sentence A. Also, in NSP task, only the token embedding of [CLS] is used to do classification, other token embeddings are discarded.

Because BERT has become ubiquitous recently for NLP tasks and our implementation is effectively identical to the original paper except for some tiny data pre-processing, we omit an exhaustive description of the model architecture and refer readers to [25] as well as its GitHub resources.[6]

Although the performance of BERT is outstanding, we cannot apply it directly to exploitability prediction. For one thing, training BERT from scratch is slow and resource-consuming. It is infeasible and unnecessary for most downstream applications to train such a comprehensive NLP model from scratch. For another, the corpora size in most downstream applications including exploitability prediction are too small to train such a complicated NLP model. For example, the total words in descriptions collected from published vulnerabilities between 1999 and 2019 is only about 5 million. Fortunately, Google has released several versions of pre-trained models, which are well trained on BooksCorpus

[6] https://github.com/google-research/bert

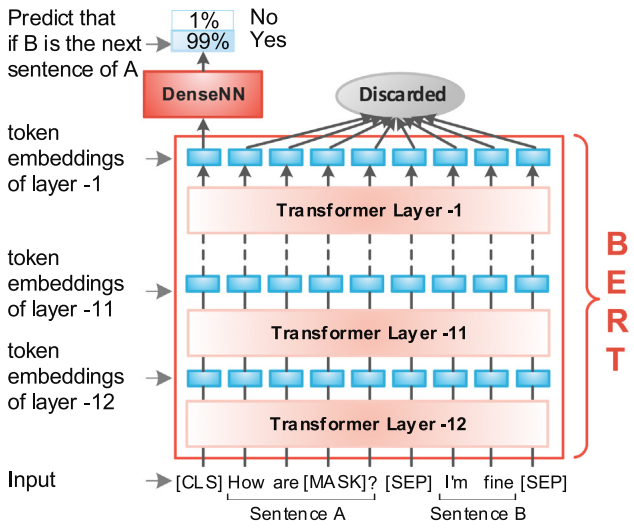

Fig. 3. The training process of Next Sentence Prediction.

Table 1
Examples of wordpiece tokenization.

Original words	Tokenized wordpieces
windows	'windows'
linux	'linux'
root	'root'
leaves	'leaves'
spyware	'spy', '##ware'
malware	'mal', '##ware'
vulnerabilities	'vu', '##ln', '##era', '##bilities'
overflow	'over', '##flow'

(800 million words) [27] and English Wikipedia (2500 million words).

Applying transfer learning to BERT is a feasible solution for exploitability prediction. Paper [28] demonstrated that the pretraining-then-fine-tuning paradigm has an even better performance and generalization capability than training from scratch for BERT model in a visualization way. Therefore, we can fine-tune a pre-trained BERT model using cybersecurity domain corpus and transfer the knowledge learnt from the super-large pre-training corpora to the new cybersecurity task.

2.3. Wordpiece tokenization

Wordpiece tokenization was first proposed by Wu et al. in 2016 to handle rare words in Google's Neural Machine Translation System [29]. It divides low-frequency words to a set of common sub-word units (wordpieces) if these words are not in the vocabulary. For example, Table 1 lists some wordpiece tokenized words when adopting the same vocabulary (contains 30,255 tokens) with BERT. We can see that for some common words contained in the vocabulary like 'windows', 'linux', 'root' and 'leaves', the tokenized tokens are the same with the original words. For some low-frequency words which are not in the vocabulary like 'spyware', 'malware', 'vulnerabilities' and 'overflow', they are tokenized into several wordpieces like ['spy', '##ware'], ['mal', '##ware'], ['vu','##ln', '##era', '##bilities'] and ['over', '##flow'] based on the vocabulary. The rest part followed the '##' in a token should be attached to the previous token without space when reversing the tokenization.

Wordpiece tokenization is a powerful tool to deal with rare words or domain-specific terms. Because it can deal with words which are not contained in a vocabulary. The tokenized word is a fixed token combination once the vocabulary is fixed. Many NLP models including BERT can leverage the bidirectional information of a sequence. Therefore, the model can understand the semantic meaning of a fixed token combination after training, which means the tokenized word can still be identified and represented by the corresponding NLP model.

The corpus for exploitability prediction contains around 5 million words, which is too small to train a robust NLP model from scratch. Therefore, we need to apply transfer learning to the pre-trained BERT model and transfer the knowledge learnt from the huge pre-training corpus to the cybersecurity domain. When applying transfer learning, it is important to use the identical wordpiece tokenization method including the same vocabulary as the pre-trained NLP model. Because it will split the downstream-task's corpus into tokens the same way with the pretraining corpus, and the pre-trained semantic knowledge will be inherited by downstream applications. Besides, technology is evolving and new technical terms are emerging every day. Therefore, it is impossible to keep changing the vocabulary of the NLP model. In fact, different vocabulary will only cause different forms of token combinations. As long as keeping consistency, it makes no difference for a machine to understand 'overflow', { 'over', '##flow'} or { 'ov', '##erf', '##low'}. Therefore, instead of building a cybersecurity domain vocabulary for exploitability prediction, we use the identical vocabulary with BERT.

3. Problem formulation

Exploitability prediction is a classification problem in the machine learning domain. The input is a vulnerability description and the output is the likelihood of the input vulnerability being exploitable. As text information, descriptions cannot be used as the input of a classification model directly and it must be transformed into numerical vectors using semantic feature extraction algorithms such as one-hot, Bag-of-Words [30], Skip-gram [30], GloVe [31] and BERT. We formulate the exploitability prediction model as (1),

$$\hat{y} = f(\Theta, x), \tag{1}$$

where x is the input feature extracted from a vulnerability description; Θ is the model parameters and $f(\cdot)$ is the predictive function determined by the predictive model structure and $\hat{y}$ is the predicted probability in a closed interval [0, 1].

We denote the dataset for exploitability prediction as D, Y, where $D = [d^{(1)}, d^{(2)}, \cdots, d^{(m)}]$; $d^{(i)}$ is the ith description of the ith vulnerability; m is the total number of vulnerabilities in D. Accordingly, $Y = [y^{(1)}, y^{(2)}, \cdots, y^{(m)}]$ is the ground truth label vector and $y^{(i)} \in \{1, 0\}$ indicates if the ith sample is exploitable or not. To implement exploitability prediction, D will be transformed into a numerical feature matrix $X = [x^{(1)}, x^{(2)}, \cdots, x^{(m)}]$, where $x^{(i)} \in \mathbb{R}^n$ is the feature extracted from $d^{(i)}$ and n is the feature dimension. According to (1), the predictive output corresponding to X is $\hat{Y} = f(\Theta, X)$, where $\hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \cdots, \hat{y}^{(m)}]$; $\hat{y}^{(i)}$ is the predicted probability corresponding to $x^{(i)}$.

Since exploitability prediction is a binary classification task, according to the conventions in machine learning, the training objective for optimizing the predictive model parameter Θ is to minimize the binary cross-entropy loss function expressed as (2).

$$\begin{aligned} &\min_{\Theta} \; E_{(x,y)\sim(X,Y)}[L_{(x,y)}(\Theta)] \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \bullet log(\hat{y}_i) + (1 - y^{(i)}) \bullet log(1 - \hat{y}_i)]. \end{aligned} \tag{2}$$

4. Methodology

To predict the exploitability of vulnerabilities using descriptions, we propose a framework ExBERT based on transfer learning. This section specifies the architecture of ExBERT and how it works in detail.

4.1. Architecture of exbert

ExBERT mainly consists of two stages, i.e., BERT transfer learning and exploitability prediction application, as shown in Fig. 4. The BERT transfer learning process will finally generate a fine-tuned BERT model, which will be used in the exploitability prediction application stage. Exploitability prediction application process consists of four steps, namely, tokenization, token embedding, sentence embedding and exploitability prediction. Fig. 4 shows the sequential relationship of these steps. Firstly, in the tokenization step, vulnerability descriptions are split into tokens with wordpiece tokenization algorithm. Then, in token embedding step, the tokenized tokens are put into the fine-tuned BERT model. The outputs of the fine-tuned BERT at the lth ($l = -1, -2, \cdots, -L$) layer are the extracted token embeddings, where L is the total number of Transfer layers in BERT and the symbol '-' indicates that we count the layer from the output layer to the input layer. Next, in the sentence embedding step, instead of picking [CLS] token embedding directly as the sentence embedding, we design a Pooling Layer to extract the sentence-level semantic features based on all token embeddings. Finally, a Classification Layer with an LSTM classifier is designed to calculate the predicted likelihood of exploitability $\hat{y}$. We state how the proposed framework works in detail in the rest of this section.

4.2. BERT transfer learning

BERT transfer learning means fine-tuning a pre-trained BERT into a fine-tuned BERT using the collected description corpus. The pre-trained BERT is trained from a randomly initialized BERT on the two pre-training corpora BooksCorpus and English Wikipedia performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total) [25]. Google AI has trained and released many versions of pre-trained BERT to satisfy different applications.

The first thing for BERT transfer learning is to download a proper pre-trained BERT model. We choose the uncased BERT-base[7] as the pre-trained model, which consists of 12 Transfer layers. Then, all vulnerability descriptions in the NVD database from 1999 to 2019 are collected as the domain corpus.

We simplify the input–output relationship of the pre-trained BERT model as formula (3),

$$\boldsymbol{e^{[l]}} = f_{BERT}(\Theta_{BERTpret}, \boldsymbol{t}), \tag{3}$$

where $\boldsymbol{t} = [t_1, t_2, \cdots, t_n]$ is a token list with n tokens which is tokenised from a vulnerability description; $\boldsymbol{e^{[l]}} = [e_1^{[l]}, e_2^{[l]}, \cdots, e_n^{[l]}]$ is the corresponding lth layer's token embeddings extracted from the pre-trained BERT model; $\Theta_{BERTpret}$ is the pre-trained BERT model parameters; $f_{BERT}(\cdot)$ is the forward propagation mapping function between $\boldsymbol{t}$ and $\boldsymbol{e^{[l]}}$, which is decided by the BERT structure; $e_j^{[l]} \in \mathbb{R}^{H_{BERT}^{[l]}}$ is the lth layer's token embedding for the jth token t_j, where $H_{BERT}^{[l]}$ is the hidden size of the lth layer.

After transfer learning, parameters of BERT are updated from the pre-trained state $\Theta_{BERTpret}$ to the fine-tuned state Θ_{BERTft}. Compared with training a BERT model from scratch, applying transfer learning to BERT model can not only maintain the high performance brought by the comprehensive model, but also avoid the extremely high training cost and the lack of domain data.

[7] https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

4.3. Exploitability prediction application

4.3.1. Tokenization via wordpiece tokenization

Tokenization via Wordpiece Tokenization is a data preprocessing process for NLP. Descriptions D are tokenized into token lists $T = [\boldsymbol{t^{(1)}}, \boldsymbol{t^{(2)}}, \cdots, \boldsymbol{t^{(m)}}]$, where $\boldsymbol{t^{(i)}} = [t_1^{(i)}, t_2^{(i)}, \cdots, t_n^{(i)}]$ is a token list split from description $d^{(i)}$; n is the pre-set max sequence length for tokenized descriptions. The notation $t_j^{(i)}$ means the jth token split from the ith description $d^{(i)}$, where $i \in \{1,2, \cdots, m\}$ and $j \in \{1, 2, \cdots, n\}$.

4.3.2. Token embedding by fine-tuned BERT

The output of tokenization is the input of token embedding. When inputting a token list $\boldsymbol{t}$, the fine-tuned BERT will output different level of token embeddings through different BERT Transfer Layers. For example, the lth layer's token embeddings extracted from the fine-tuned BERT is expressed as below:

$$\boldsymbol{e^{[l]}} = f_{BERT}(\Theta_{BERTft}, \boldsymbol{t}). \tag{4}$$

Similar with Eq. (3), $\boldsymbol{t} = [t_1, t_2, \cdots, t_n]$ is a token list with n tokens, $\boldsymbol{e^{[l]}} = [e_1^{[l]}, e_2^{[l]}, \cdots, e_n^{[l]}]$ is the corresponding lth layer's token embeddings extracted from fine-tuned BERT model. $e_j^{[l]} \in \mathbb{R}^{H_{BERT}^{[l]}}$ is the lth layer's token embedding for the jth token t_j, where $H_{BERT}^{[l]}$ is the hidden size of the lth layer of BERT. Especially, the embedding of the first token [CLS] is $e_1^{[-1]}$. Generally speaking, the closer l is to the output layer, the more specific and application-relevant the corresponding token embeddings are. In the original paper, they always extract the last layer's token embedding for [CLS] $e_1^{[-1]}$ as the semantic feature to do classification task.

4.3.3. Sentence embedding by pooling layer

In order to leverage the abundant semantic information contained in all token embeddings and more layers, instead of using $e_1^{[-1]}$ only like the original paper, we design a Pooling Layer for ExBERT to obtain the sentence-level embeddings on top of the fine-tuned BERT. The input of the Pooling Layer is the token embeddings $\boldsymbol{e^{[l]}}$ extracted from fine-tuned BERT. The output of the Pooling Layer is decided by the adopted pooling strategy. Common pooling strategies for NLP tasks include 'Mean' and 'Max'. Additionally, paper [25] has demonstrated that the token embedding for [CLS] $e_1^{[l]}$ can be used as the sentence embedding for downstream tasks directly. Therefore, we define the output of the Pooling Layer as a piecewise function formulated in (5),

$$e_{sent}^{[l]} = \begin{cases} mean(e^{[l]}), & pooling\ strategy\ is\ Mean \\ max(e^{[l]}), & pooling\ strategy\ is\ Max \\ e_1^{[l]}, & pooling\ strategy\ is\ [CLS] \end{cases}, \tag{5}$$

where l could be any single layer in BERT or a combination of several layers; $e_{sent}^{[l]}$ means the extracted sentence-level embedding.

4.3.4. Exploitability prediction in classification layer

Following the Pooling Layer is a Classification Layer, which contains a hidden layer and an output layer. We adopt a LSTM layer instead of a DenseNN as the hidden layer to capture the dependencies amongst extracted sentence embeddings. The output of the hidden layer is formulated in (6),

$$a_h = f_{LSTM}(\Theta_{LSTM}, e_{sent}), \tag{6}$$

where $a_h \in \mathbb{R}^{H_h}$ is the activation result and H_h is the number of hidden nodes in the hidden LSTM layer; Θ_{LSTM} is the parameters for the LSTM layer; $f_{LSTM}(\cdot)$ is the mapping functions for LSTM.

The final output layer is a DenseNN layer with one sigmoid node and the final exploitability prediction result is calculated as below,

$$\hat{y} = \sigma(Wa_h + b), \tag{7}$$

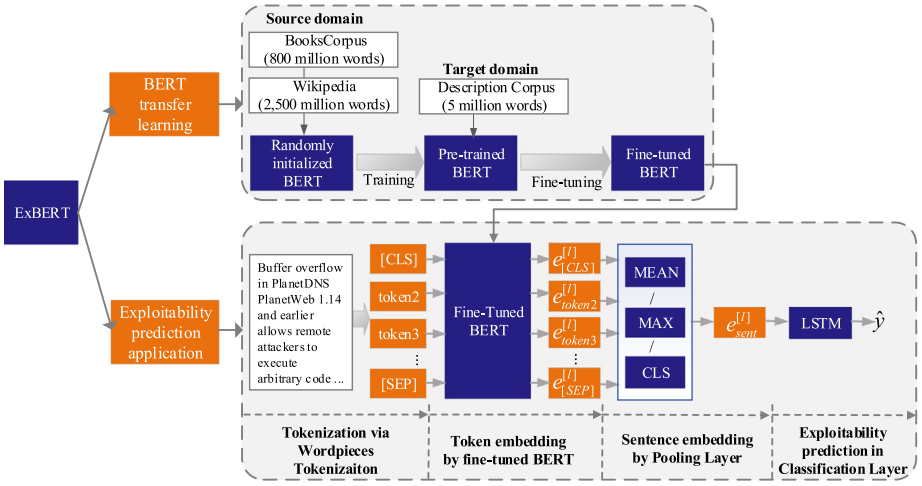

Fig. 4. The architecture of the proposed framework ExBERT.

where weights W and bias b are the parameters of the final DenseNN layer, σ is the sigmoid activation function and $\hat{y}$ is the final possibility of vulnerability exploitability.

5. Experiments and results

All experiments in this work are implemented on an Ubuntu 18.04 operating system with two NVIDIA GeForce RTX 2080 Ti GPUs. The deep learning framework used is Keras[8] with the TensorFlow backend.

5.1. Dataset collection and experimental setting

The datasets used in this work are downloaded from NVD and ExploitDB, containing all vulnerabilities and exploits published between 1999 and 2019. Different vulnerabilities and exploits are identified by CVE-IDs and EDB-IDs accordingly. A CVE-ID is a unique vulnerability identifier, while an EDB-ID is a unique identification of an exploit archived by the ExploitDB database. Most studies including this work mark a vulnerability as exploitable when it has a corresponding 'proof-of-concept' exploit identified by an EDB-ID. As shown in Fig. 5, we obtain 'CVE-ID/Description' pairs from NVD database and 'EDB-ID/CVE-ID' pairs from ExploitDB website. The two databases are integrated into 'Description/ Exploitability' pairs through CVE-ID matching. If a CVE-ID can be found in the 'EDB-ID/CVE-ID' pairs, the corresponding exploitability is set to '1' and the vulnerability is exploitable. Otherwise, the corresponding exploitability is set to '0' and the vulnerability is unexploitable. Finally, the collected dataset is denoted as $\{D, Y\}$. In total, we download 123,254 'CVE-ID/Description' pairs from the NVD website and 41,365 'EDB-ID/CVE-ID' pairs from the ExploitDB website.

After data de-duplication and integration, we finally get 118,209 samples. Among them, 23,073 (19.52%) unique vulnerabilities have corresponding exploits which are exploitable. We randomly selected 23,073 non-exploitable vulnerabilities, along with all exploitable vulnerabilities, to form a balanced dataset with 46,176 vulnerabilities. The dataset is divided to a training set, a validation set and a test set according to a ratio of 70%: 15%: 15%. The training set is used to train classifiers; validation set is used to select hyper-parameters and test set is used to evaluate the final performance of ExBERT.

We apply wordpiece tokenization on all descriptions and listed the distribution of the word counts and token counts in Fig. 6,

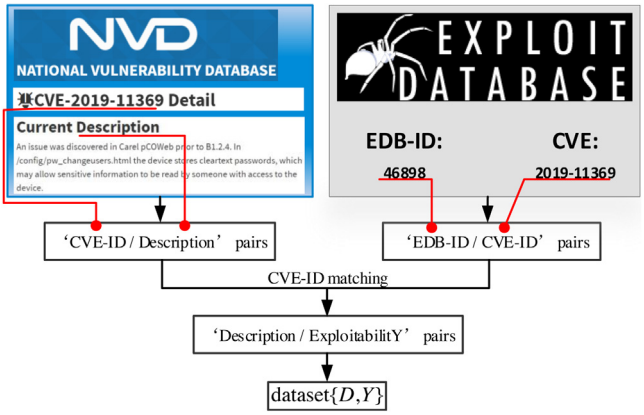

Fig. 5. Dataset collection process.

in which we omit word bins > 300 and token bins > 600 to give more details on the valid data. In fact, 99.16% of descriptions is ≤ 128 words. After tokenization, 99.11% of descriptions is ≤ 254 tokens. Considering the added tokens [CLS] and [SPE], the hyper-parameter 'max sequence length' is set to 256.

The pre-trained BERT model used in this paper is a downloaded BERTbase[9] model, whose Transfer layer L=12 and hidden size $H^{[l]}_{BERT} = 768$. All 118,209 vulnerability descriptions are used as the fine-tuning corpus, and the training epochs are set to 3. It takes about 3 h and 20 min to finish the fine-tuning process on 2 NVIDIA GeForce RTX 2080 Ti GPUs.

5.2. Ablation study

ExBERT is an improved framework based on BERT and we propose three improvements in token embedding stage, Pooling Layer and Classification Layer. Compared with the original BERT model, the proposed improved factors and their possible values are listed as below.

(i) In token embedding stage, the BERT model could be fine-tuned BERT or pre-trained BERT.

(ii) In Pooling Layer, pooling strategy could be Mean, Max or [CLS] and BERT token embedding layer l could be -1, -2, et al.

8 https://keras.io/

9 https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

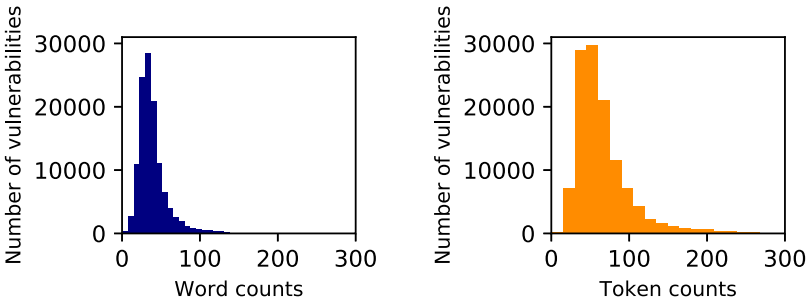

Fig. 6. The word count and token count distributions for descriptions.

Table 2
Word distances in a two-dimensional space.

Word pairs	In vocabulary	Distances (BERT)	Distances (Fine-tuned BERT)
spyware →vulnerabilities	No	11.44	**3.66**
malware →vulnerabilities	No	5.79	**2.41**
spyware →malware	No	5.67	**2.62**
windows→linux	Yes	13.59	**1.61**
linux→root	Yes	7.56	**1.45**
root→leaves	Yes	**1.76**	6.38

(iii) In Classification Layer, the classifier could be DenseNN, RNN and LSTM.

When implementing ExBERT as a whole, those improvements make contributions at the same time. To evaluate the effect and influence of every single factor in ExBERT, we conduct a series of ablation studies and list the results in this section.

5.2.1. Wordpiece tokenization effect analysis

As described in Section 2.3, we adopt wordpiece tokenization method to deal with low-frequency cybersecurity domain-specific words. To verify if the semantic meaning of low-frequency words which are not contained in the vocabulary can be learnt, we select several words in Table 1 to visualize their word embeddings in a two-dimensional space as shown in Fig. 7.

To get the data points in Fig. 7, we first feed each word into the pre-trained BERT and fine-tuned BERT separately. Although 'spyware', 'malware' and 'vulnerabilities' are not contained in the vocabulary, they can be tokenized in token sequences. BERT has the capacity to learn semantic knowledge bidirectionally from a token sequence. Specifically, their word embeddings could be represented by their corresponding [CLS] token embeddings in the last layer of the pre-trained or fine-tuned BERT model. After that, a Principal Component Analysis (PCA) algorithm is used to reduce those token embeddings to a two-dimensional space.

In the cybersecurity domain, both 'spyware' and 'malware' are harmful software that can take advantage of software 'vulnerabilities' to attack a user's system. As shown in Fig. 7, both 'spyware' and 'malware' have a relatively shorter distance to 'vulnerabilities' with fine-tuned BERT model, compared with pre-trained BERT model. This example shows that, after fine-tuning with cybersecurity corpus, the fine-tuned model is indeed better at understanding the semantics in the cybersecurity context than the pre-trained model.

This conclusion also holds for words in the vocabulary, such as 'windows', 'linux', 'root' and 'leaves'. In a general context, 'windows' mean an opening in a wall or roof, so it has a long distance with 'linux', which is an operating system, in a semantic space established by BERT model. However, a fine-tuned BERT model learnt that both 'windows' and 'linux' are operating systems, so they are very close in the fine-tuned cybersecurity semantic space. Similarly, in a general context, both 'root' and 'leaves' are a part of plants, so they are quite close in BERT semantic space. In contrast, 'root' has a much closer relationship with 'linux' than with 'leaves' in the cybersecurity context.

Table 2 lists the Euclidean distances between word pairs in Fig. 7, in which the minimum distance of each word pair is emphasized in bold.

Table 3
Comparison on fine-tuned BERT and pre-trained BERT.

Models	Fine-tuned BERT	Pre-trained BERT	Improvement (%)
Best epoch	**11.63**	53.47	78.24%
Loss	**0.2348**	0.4516	48.01%
Accuracy	**0.9098**	0.7924	14.54%
F1 score	**0.9106**	0.7943	14.71%
Precision	**0.9134**	0.7963	14.53%
Recall	**0.9078**	0.7926	14.53%

5.2.2. Transfer learning effect analysis

To evaluate the effect and influence of transfer learning on the performance of exploitability prediction, we fix pooling strategy as [CLS], BERT token embedding layer $l = -1$ and classifier in Classification Layer as DenseNN. The only variable factor is the BERT model in token embedding stage.

Token Embedding Visualization Comparison. Fig. 8 shows the [CLS] token embeddings of 1000 randomly chosen samples extracted from fine-tuned BERT and pre-trained BERT. The original [CLS] token embedding's dimension is reduced from $H_{BERT}^{(-1)} = 768$ to 2 using Principal Component Analysis (PCA). The horizontal axis is the 1st principal component and the vertical axis is the 2nd principal component of [CLS] token embedding. The dots in 'navy' are non-exploited samples while the dots in 'dark orange' are exploited samples. It is obvious that [CLS] token embeddings extracted from fine-tuned BERT are more promising to separate exploitable and unexploitable vulnerabilities compared with pre-trained BERT.

Classification performance Comparison. To make quantitative comparison, we input the [CLS] token-embeddings extracted from fine-tuned and pre-trained BERT to a DenseNN classifier and compare their classification performance in Fig. 9 and Table 3.

Fig. 9 shows the results of one training process. The left subplots are the results on fine-tuned BERT while the right subplots are on the pre-trained BERT. The x-axes represent the number of training epochs and the y-axes are the metrics specified by the

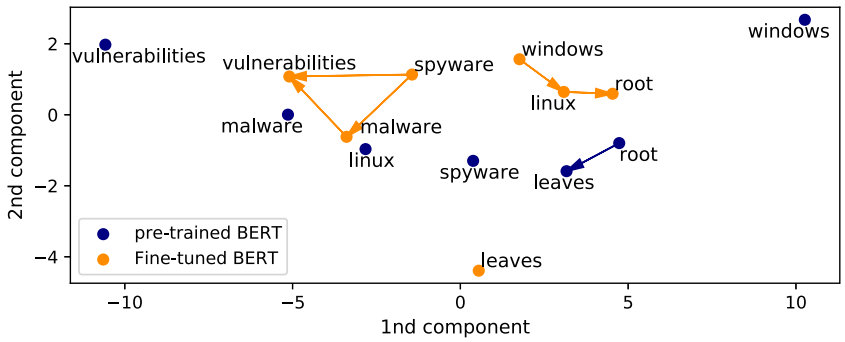

Fig. 7. Word embeddings in a two-dimensional space.

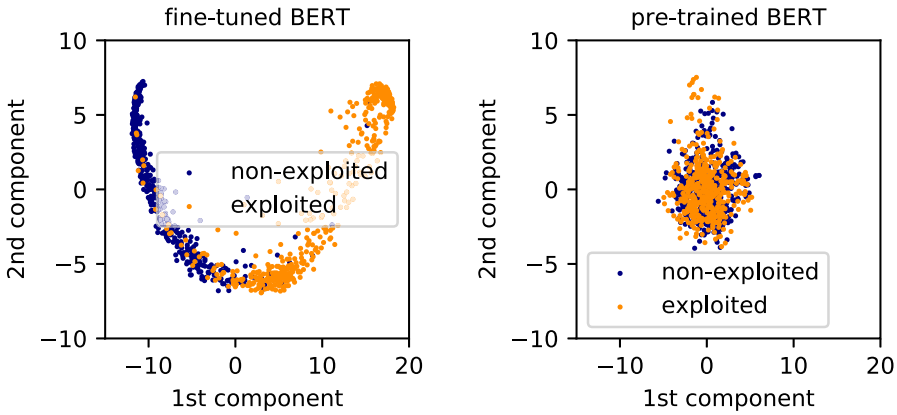

Fig. 8. [CLS] token embeddings visualization comparison.

y-labels. The lines in 'navy' are the results on the training set while the lines in 'dark orange' are on the validation set. Fig. 9 shows that using fine-tuned BERT can achieve approximately 0.9 on the accuracy, F1 score, precision and recall on the validation set, while using the pre-trained BERT can only get around 0.8 on all criteria.

Table 3 shows the performance comparison on the test set. To reduce the influence of randomness, we repeat the training and test experiments 30 times and place the mean values into Table 3. Best epoch is the training epoch number on which the model achieves the smallest validation loss. Results on the test set demonstrate the effectiveness of the transfer learning. The accuracy has improved by 14.54% from 0.7924 to 0.9098 via transfer learning. The F1 score, precision and recall also have an equivalent improvement as shown in Table 3. The average best epoch is decreased by 78.24% from 53.47 to 11.63, which indicates the fine-tuned BERT model are more presentative in the exploitability prediction domain and therefore needs less training epochs to distinguish the exploitable and unexploitable vulnerabilities.

5.2.3. Pooling layer effect analysis

There are two variable factors in Pooling Layer, i.e., pooling strategy and BERT token embedding layer. These two factors are analysed separately below.

Pooling Strategy Comparison. Similarly, to evaluate the effect and influence of pooling strategy on the performance of exploitability prediction, we fix token embedding model as fine-tuned BERT, BERT token embedding layer $l = -1$ and classifier in Classification Layer as DenseNN. The only variable factor is the pooling strategy. Evaluation results are shown in Fig. 10 and Table 4.

Table 4
Comparison on pooling strategies.

Pooling strategies	Mean	Max	[CLS]
Best epoch	16.17	22.53	**11.63**
Loss	0.2386	0.2421	**0.2348**
Accuracy	0.9063	0.9036	**0.9098**
F1 score	0.9070	0.9044	**0.9106**
Precision	0.9106	0.9069	**0.9134**
Recall	0.9034	0.9019	**0.9078**

Fig. 10 shows the results of one training process. All left subplots are the results of 'Mean' pooling strategy, all middle subplots are for 'Max' and all right subplots are for [CLS]. Similarly, all x-axes represent the number of epochs for training and all y-axes are the metrics specified by y-labels. The lines in 'navy' are results on the training set while the lines in 'dark orange' are on the validation set. The results in Table 4 are also the average results of 30 independent experiments to eliminate the influence of randomness.

Besides, Fig. 10 shows that although pooling strategy 'Mean', 'Max' and [CLS] can achieve equivalent good performance on their best epochs, 'Mean' has a more stable performance on both training set and validation set when the training epoch increases. However, both 'Max' and [CLS] will become overfitting when training epoch increases, which means the performance increases on the training set but decreases on the validation set, thus the generalization performance will become worse.

Table 4 indicates all pooling strategies can achieve very good performance. However, [CLS] has a weak advantage.

BERT Token Embedding Layer Comparison. Similarly, to evaluate the effect and influence of BERT token embedding layer on the performance of exploitability prediction, we fix token embedding model as fine-tuned BERT, pooling strategy as [CLS] and classifier

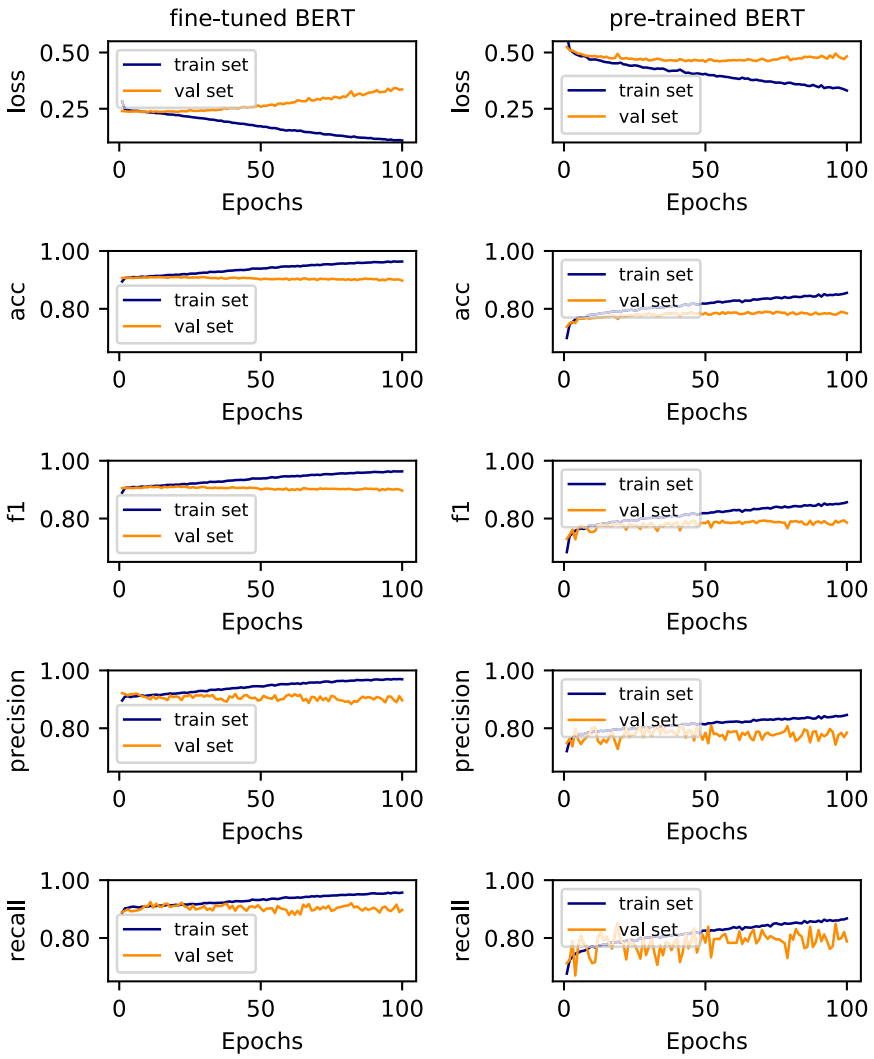

Fig. 9. Training process comparison on fine-tuned BERT and pre-trained BERT.

Table 5
Comparison on BERT token embedding layers.

BERT layers	Best epoch	Loss	Accuracy	F1 score	Precision	Recall
−1	11.63	0.2348	**0.9098**	**0.9106**	0.9134	0.9078
−2	8.13	0.2370	0.9068	0.9076	0.9103	0.9049
−1, −2	10.17	0.2352	0.9092	0.9099	**0.9136**	0.9062
−3, −4	9.10	0.2444	0.9030	0.9039	0.9053	0.9026
−1, −2, −3, −4	**8.10**	0.2343	0.9087	0.9096	0.9112	**0.9079**

Table 6
Classifier comparison results.

Classifiers	Best epoch	loss	Accuracy	F1 score	Precision	Recall
DenseNN	11.63	0.2348	0.9098	0.9106	0.9134	0.9078
RNN	14.73	0.2350	0.9102	0.9109	0.9136	**0.9083**
LSTM	**11.57**	0.2354	**0.9112**	**0.9116**	**0.9182**	0.9051

in Classification Layer as DenseNN. The only variable factor is the BERT token embedding layer in Pooling Layer. Setting BERT token embedding layer $l \in \{-1, -2, [-1, -2], [-3, -4], [-1, -2, -3, 4]\}$, the evaluation results for BERT token embedding Layer are listed in Table 5. When BERT layer contains more than one layer (i.e. $[-1, -2]$), the token embedding for each token is the concatenation of token embeddings in each layer.

Table 5 also presents the average results of 30 independent experiments to reduce the influence of randomness. As shown in Table 5, layer -1 achieves the best accuracy 0.9098 and F1 score 0.9106, layer $[-1, -2]$ gains the best precision 0.9136 and layer $[-1, -2, -3, -4]$ reaches the smallest best epoch 8.1 and the best recall 0.9079. However, generally speaking, different BERT token embedding layers have an equivalent performance on exploitability prediction and none of them has an overwhelming advantage. To decrease the computing complexity caused by concatenating token embedding between different layers, finally, ExBERT fixed l to -1.

5.2.4. Classifier effect analysis

Similarly, to evaluate the effect and influence of classifier on the performance of exploitability prediction, we fix token embedding model as fine-tuned BERT, pooling strategy as [CLS] and BERT token embedding layer $l = -1$. The only variable factor is the classifier in Classification Layer. The evaluation results for different classifiers are listed in Table 6, which also presents the average results of 30 independent experiments to reduce the influence of randomness.

As shown in Table 6, classifier RNN achieves the best recall 0.9083 and LSTM obtains the best accuracy 0.9112, F1 score 0.9116 and precision 0.9182. Therefore, LSTM has a weak advantage compared with DenseNN and RNN.

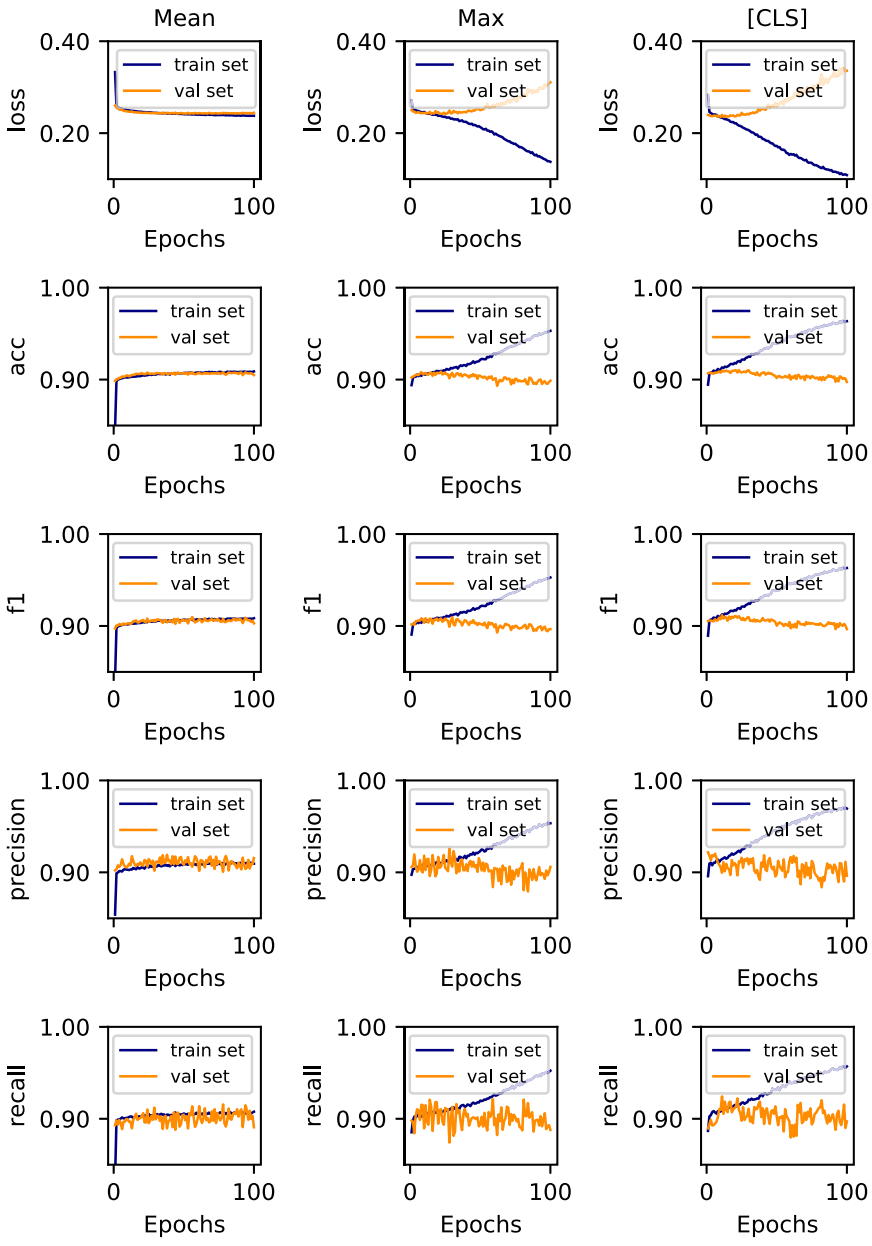

Fig. 10. Training process comparison on pooling strategies.

5.3. Remarks

In Section 5.2, we analyse the effect and influence of various design factors separately. In this section, we put things together and list the exploitability prediction results on ExBERT and BERT as well as 8 variants for both of them (ExBERT_V1 to ExBERT_V8 and BERT_V1 to BERT_V8) in Fig. 11 and Table 7. Obviously, ExBERT and its variants outperformed BERT and its variants with a large margin, improved by almost 14% on accuracy, F1 score, precision and recall.

5.4. Comparison with other similar works

We compare ExBERT with several existing algorithms from the aspects of feature extraction method, classification algorithm and four classification metrics in Table 8. All algorithms listed in Table 8 use vulnerability descriptions from the NVD database as the input feature to predict the exploitability of vulnerabilities. The results of other works are obtained from the original papers. Results in Table 8 show that ExBERT has greatly improved the performance of vulnerability exploitability prediction on accuracy, F1 score, precision and recall.

5.5. Analysis and ponder

Based on results presented in Figs. 9 to 11 and Tables 3 to 7, we can draw the following conclusions.

(1) Our proposed framework ExBERT achieves the state-of-the-art results for exploitability prediction problem. ExBERT can achieve 91.12% on the accuracy, 0.9116 on F1 score, 91.82% on precision and 90.51% on recall.

(2) The dominating factor for performance improvement is BERT fine-tuning via transfer learning other than pooling strategies or classifiers. Figs. 8, 9 and Table 3 have verified that only extracting [CLS] token embedding from the fine-tuned BERT can achieve desirable performance. Keeping other settings identical, extracting features from the pre-trained BERT can only get around 0.8 on all metrics.

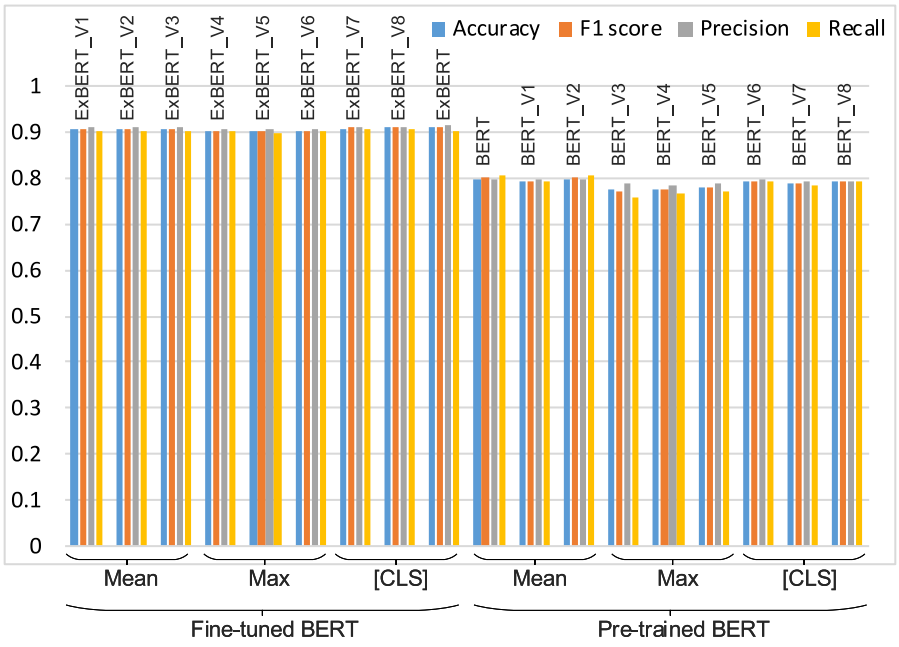

Fig. 11. Performance comparison on ExBERT and BERT as well as their variants.

Table 7
Performance comparison on ExBERT and BERT as well as their variants.

BERT	Pooling strategies	Classifiers	Model versions	Best epoch	Loss	Accuracy	F1 score	Precision	Recall
Fine-tuned BERT	Mean	DenseNN	ExBERT_V1	16.17	0.2386	0.9063	0.9070	0.9106	0.9034
		RNN	ExBERT_V2	31.30	0.2391	0.9071	0.9079	0.9103	0.9055
		LSTM	ExBERT_V3	15.47	0.2385	0.9068	0.9076	0.9100	0.9053
	Max	DenseNN	ExBERT_V4	22.53	0.2421	0.9036	0.9044	0.9069	0.9019
		RNN	ExBERT_V5	17.57	0.2429	0.9034	0.9042	0.9074	0.9011
		LSTM	ExBERT_V6	28.27	0.2418	0.9034	0.9042	0.9066	0.9019
	[CLS]	DenseNN	ExBERT_V7	11.63	0.2348	0.9098	0.9106	0.9134	0.9078
		RNN	ExBERT_V8	14.73	0.2350	0.9102	0.9109	0.9136	**0.9083**
		LSTM	**ExBERT**	**11.57**	0.2354	**0.9112**	**0.9116**	**0.9182**	0.9051
Pre-trained BERT	**Mean**	DenseNN	BERT	65.30	0.4389	0.7996	**0.8032**	0.7978	**0.8088**
		RNN	BERT_V1	96.83	0.4510	0.7945	0.7959	0.7994	0.7925
		LSTM	BERT_V2	54.57	**0.4386**	**0.8000**	0.8029	**0.8002**	0.8058
	Max	DenseNN	BERT_V3	90.20	0.4670	0.7758	0.7735	0.7909	0.7573
		RNN	BERT_V4	88.26	0.4750	0.7749	0.7750	0.7837	0.7666
		LSTM	BERT_V5	94.40	0.4648	0.7795	0.7794	0.7889	0.7705
	[CLS]	DenseNN	BERT_V6	53.47	0.4516	0.7924	0.7943	0.7963	0.7926
		RNN	BERT_V7	95.90	0.4535	0.7879	0.7890	0.7939	0.7843
		LSTM	BERT_V8	**38.00**	0.4521	0.7917	0.7935	0.7957	0.7916

Table 8
Performance comparison with other similar works.

Researchers	Feature extraction method	Classifier	Acc	F1	Pre	Recall
Bozorgi [7]	TF–IDF	SVM	0.890	–	–	–
Nazgol Tavabi [14]	TF–IDF/Doc2Vec	SVM/RF	–	0.660	–	–
Zhuobing Han [15]	word embedding	CNN	0.816	0.816	0.818	0.815
Michel Edkrantz [18]	10,000 MCWs	SVM	0.833	–	0.825	0.834
ExBERT	**Fine-tuned BERT**	**LSTM**	**0.911**	**0.912**	**0.918**	**0.905**

(3) As demonstrated in Fig. 11 and Table 7, pooling strategy [CLS] can achieve the best performance on fine-tuned BERT, while 'Mean' is the best for the pre-trained BERT. 'Max' is the worst for both of them. The reason why [CLS] can achieve best results on the fine-tuned BERT is that [CLS] itself is a token embedding extracted from the fine-tuned BERT and it learns some semantic information of the entire sentence during the fine-tuning process.

(4) 'LSTM' is the best classifier for ExBERT, as shown in Tables 6 and 7. This verifies that LSTM is suitable for dealing with sequence inputs by capturing the dependencies within inputs.

6. Conclusion

Thousands of software vulnerabilities are disclosed to public each year and it is crucial for software vendors to prioritize their efforts and patch the most critical vulnerabilities. However, the de facto industry standard CVSS has been proved a poor indicator for vulnerability assessment. In order to find out the most possible exploitable vulnerabilities, we propose an exploitability prediction framework ExBERT to accurately predict if a vulnerability will be exploited or not. Experimental results on

46,176 real-world vulnerabilities show that ExBERT can achieve the state-of-art performance on exploitability prediction.

In our study, we assume that the data distribution and pattern for exploitability prediction are static, so we randomly shuffled the data between 1999 and 2020 to train and validate the proposed framework. Besides, we focus on learning the semantic features in vulnerability descriptions but ignore other source of related information in this framework. For future work, we plan to integrate multi-source factors into consideration, including information on user privilege, user interaction, availability, authentication, confidentiality, severity, et al. In addition, another promising direction is to establish an online learning model to capture and address concept drift problems along the time.

CRediT authorship contribution statement

Jiao Yin: Investigation, Methodology, Software, Visualization, Formal analysis, Writing - original draft, Software. **MingJian Tang:** Conceptualization, Methodology, Writing - review & editing. **Jinli Cao:** Supervision, Resources, Writing - review & editing, Project administration. **Hua Wang:** Supervision, Resources, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The first author was supported by a La Trobe University Postgraduate Research Scholarship and a La Trobe University Full Fee Research Scholarship, she was also supported by the Science and Technology Research Program of Chongqing Municipal Education Commission of China (Grant No. KJQN201901306 and KJQN201801325).

References

[1] M. Tang, M. Alazab, Y. Luo, Big data for cybersecurity: Vulnerability disclosure trends and dependencies, IEEE Trans. Big Data 5 (3) (2019) 317–329.
[2] L. Bilge, T. Dumitraş, Before we knew it: an empirical study of zero-day attacks in the real world, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, ACM, 2012, pp. 833–844.
[3] M. Tang, J. Yin, M. Alazab, J.C. Cao, Y. Luo, Modelling of extreme vulnerability disclosure in smart city industrial environments, IEEE Trans. Ind. Inf. (2020) 1, http://dx.doi.org/10.1109/TII.2020.3022182.
[4] M. Schiffman, A. Wright, D. Ahmad, G. Eschelbeck, The Common Vulnerability Scoring System, National Infrastructure Advisory Council, 2004, pp. 2–20, Vulnerability Disclosure Working Group, Vulnerability Scoring Subgroup.
[5] C. Team, Common vulnerability scoring system v3. 0: Specification document, First. org. (2015) 1–21.
[6] A. Younis, Y.K. Malaiya, I. Ray, Assessing vulnerability exploitability risk using software properties, Softw. Qual. J. 24 (1) (2016) 159–202.
[7] M. Bozorgi, L.K. Saul, S. Savage, G.M. Voelker, Beyond heuristics: learning to classify vulnerabilities and predict exploits, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2010, pp. 105–114.
[8] L. Allodi, F. Massacci, Comparing vulnerability severity and exploits using case-control studies, ACM Trans. Inf. Syst. Secur. 17 (1) (2014) 1–20.
[9] C. Eiram, B. Martin, The Cvssv2 Shortcomings, Faults, and Failures Formulation, Technical report, Forum of Incident Response and Security Teams (FIRST), 2013, pp. 3–13.
[10] J. Ruohonen, A look at the time delays in cvss vulnerability scoring, Appl. Comput. Inform. 15 (2) (2019) 129–135.
[11] M. Alazab, M. Tang, Deep Learning Applications for Cyber Security, Springer, 2019.
[12] F. Liu, X. Zhou, J. Cao, Z. Wang, H. Wang, Y. Zhang, Arrhythmias classification by integrating stacked bidirectional lstm and two-dimensional cnn, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2019, pp. 136–149.
[13] J. Lu, Z. Yan, J. Han, G. Zhang, Data-driven decision-making (d 3 m): Framework, methodology, and directions, IEEE Trans. Emerg. Top. Comput. Intell. 3 (4) (2019) 286–296.
[14] N. Tavabi, P. Goyal, M. Almukaynizi, P. Shakarian, K. Lerman, Darkembed: Exploit prediction with neural language models, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 7849–7854.
[15] Z. Han, X. Li, Z. Xing, H. Liu, Z. Feng, Learning to predict severity of software vulnerability using only vulnerability description, in: 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2017, pp. 125–136.
[16] E.R. Russo, A. Di Sorbo, C.A. Visaggio, G. Canfora, Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities, J. Syst. Softw. 156 (2019) 84–99.
[17] CVE, Cve - frequently asked questions, 2019, https://cve.mitre.org/about/faqs.html#cve_entry_descriptions_created. (Accessed 18 October 2019).
[18] M. Edkrantz, S. Truvé, A. Said, Predicting vulnerability exploits in the wild, in: 2015 IEEE International Conference on Cyber Security and Cloud Computing, IEEE, 2015, pp. 513–514.
[19] Y. Zhang, H. Chen, J. Lu, G. Zhang, Detecting and predicting the topic change of knowledge-based systems: A topic-based bibliometric analysis from 1991 to 2016, Knowl.-Based Syst. 133 (1991) 255–268.
[20] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, P. Shakarian, Proactive identification of exploits in the wild through vulnerability mentions online, in: 2017 International Conference on Cyber Conflict, CyCon US, IEEE, 2017, pp. 82–88.
[21] J. Ma, G. Zhang, J. Lu, A state-based knowledge representation approach for information logical inconsistency detection in warning systems, Knowl.-Based Syst. 23 (2) (2010) 125–131.
[22] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.
[23] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2009) 1345–1359.
[24] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, G. Zhang, Transfer learning using computational intelligence: A survey, Knowl.-Based Syst. 80 (2015) 14–23.
[25] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: The 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019, pp. 4171–4186.
[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.
[27] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, S. Fidler, Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 19–27.
[28] Y. Hao, L. Dong, F. Wei, K. Xu, Visualizing and understanding the effectiveness of bert, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, Hong Kong, 2019, pp. 4143–4152.
[29] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144.
[30] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.
[31] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014, pp. 1532–1543.