Is Deep Learning Better than Traditional Approaches in Tag Recommendation for Software Information Sites?

Pingyi Zhou[a], Jin Liu[a,b,*], Xiao Liu[c], Zijiang Yang[d], John Grundy[e]

[a] *School of Computer Science, Wuhan University, Wuhan, China.*
[b] *Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.*
[c] *School of Information Technology, Deakin University, Geelong, Australia.*
[d] *Department of Computer Science, Western Michigan University, Kalamazoo, Michigan, USA.*
[e] *Faculty of Information Technology, Monash University, Melbourne, Australia.*

Abstract

Context: Inspired by the success of deep learning in other domains, this new technique been gaining widespread recent interest in being applied to diverse data analysis problems in software engineering. Many deep learning models, such as CNN, DBN, RNN, LSTM and GAN, have been proposed and recently applied to software engineering tasks including effort estimation, vulnerability analysis, code clone detection, test case selection, requirements analysis and many others. However, there is a perception that applying deep learning is a "silver bullet" if it can be applied to a software engineering data analysis problem. **Object:** This motivated us to ask the question as to whether deep learning is better than traditional approaches in tag recommendation task for software information sites. **Method:** In this paper we test this question by applying both the latest deep learning approaches and some traditional approaches on tag recommendation task for software information sites. This is a typical Software Engineering automation problem where intensive data processing is required to link disparate information to assist developers. Four different deep learning approaches – TagCNN, TagRNN, TagHAN and TagRCNN – are implemented and compared with

*Corresponding author
Email address: jinliu@whu.edu.cn (Jin Liu)
[1] zhou_pinyi@whu.edu.cn
[2] jinliu@whu.edu.cn
[3] xiao.liu@deakin.edu.au
[4] zijiang.yang@wmich.edu
[5] john.grundy@monash.edu

three advanced traditional approaches – EnTagRec, TagMulRec, and FastTagRec. **Results:** Our comprehensive experimental results show that the performance of these different deep learning approaches varies significantly. The performance of TagRNN and TagHAN approaches are worse than traditional approaches in tag recommendation tasks. The performance of TagCNN and TagRCNN approaches are better than traditional approaches in tag recommendation tasks. **Conclusion:** Therefore, using appropriate deep learning approaches can indeed achieve better performance than traditional approaches in tag recommendation tasks for software information sites.

Keywords: Deep Learning, Data Analysis, Tag Recommendation, Software Information Site, Software Object

1. Introduction

Deep learning has been proven to be effective for some software engineering tasks, such as API embedding representation [1], modeling source code [2], code clone detection [3], semantically enhanced software traceability [4], and predicting semantically linkable knowledge in developer online forums [5]. Various deep learning models have been proposed, including commonly used ones such as Convolutional Neural Networks (CNN) [6], Deep Belief Networks (DBN) [7], Recurrent Neural Networks (RNN) [8], Long Short Term Memory Networks (LSTM) [9], Gated Recurrent Unit Neural Networks (GRU) [10], and Generative Adversarial Networks (GAN) [11]. Each model has its own typical application scenarios and requires different amounts of training time. Thus choosing an appropriate model and balancing efficiency and effectiveness are a key challenge to apply deep learning successfully for a software engineering problem [2, 5, 12]. Nevertheless, given the huge increase in recent papers on the topic, there are increasing numbers of researchers who seem to believe that deep learning is superior to more traditional techniques for data analysis and even regard it as perhaps one of Brooks' "silver bullets" for solving Software Engineering problems [13].

In a recent ASE'16 paper [5], Xu et al. applied CNN to predict semantically linkable knowledge in developer online forums. A question and its answers in a software information site such as `StackOverflow` can be considered as a knowledge unit. T-

wo knowledge units are linkable if they are semantically related. Since the problem of predicting semantically linkable knowledge units can be reduced to a multiclass classification problem, the deep learning model CNN is a good choice and indeed it achieves good performance. However, in an FSE'17 paper [12], Fu and Menzies repeated the study using support vector machine (SVM) with differential evolution (DE). DE is a more traditional hyper-parameter optimization method. The empirical study showed that SVM with DE method achieves similar performance with CNN, but its training time is 84 times faster than that of CNN. Based on their result, they concluded that traditional methods can be as good as, or even better than, the best current deep learning approaches for some data intensive software engineering problems.

However, a single case study is not general enough to determine the pros and cons of applying deep learning in software engineering. In this paper we conduct further comparative evaluation on tag recommendation for a software information site which is a typical Software Engineering problem requiring intensive data processing. Software information sites [14], [15], [16] offer indispensable platforms for software developers to search solutions, share experience, offer help and learn new techniques [17], [18], [19], [20, 21]. The contents posted on these software information sites, such as a question with answers in a developer Q&A community (e.g. `StackOverflow`[6]) and a project in an open source software community (e.g. `GitHub`[7]), are regarded as software objects [14], [15]. As the software information sites evolve, the number of software objects grows significantly. To facilitate search and classification, it is a common practice for software developers to add tags. High quality tags are expected to be concise and can describe the most important features of the software objects.

Unfortunately tagging is inherently a distributed and uncoordinated process [14]. The quality of tags depends not only on a developer's understanding of the software object but also on the developer's language skills and preferences. As a result, the number of different tags grows rapidly along with continuous addition of software objects. For example, there are more than 20 million questions and 46 thousand tags in

[6]http://www.stackoverflow.com

[7]http://github.com

`StackOverflow`. Among such a large number of different tags, many of them are different but redundant. it is becoming harder to maintain software information sites.

In this paper, we apply 4 different contemporary deep learning approaches to the tag recommendation task. These approaches are called TagCNN, TagRNN, TagHAN and TagRCNN respectively. TagCNN, TagRNN, TagHAN and TagRCNN are based on Convolutional Neural Networks (CNN) [22], Recurrent Neural Networks (RNN) [23], Hierarchical Attention Networks (HAN) [24], and Recurrent Convolutional Neural Networks (RCNN) [25], respectively. These four deep learning models have been widely used for many disparate text classification tasks. Three traditional approaches – EnTagRec [15], TagMulRec [26], and FastTagRec [27] that have previously been proposed for tag recommendation tasks – are used as comparison approaches. Our research focuses on whether there is a deep learning approach that can achieve better performance than traditional approaches for tag recommendation for software information sites. Through our extensive experiments we demonstrate a valid comparison between deep learning and traditional approaches in tag recommendation tasks. Our initial hypothesis was that traditional approaches have been well tuned for tag recommendation and thus should be able to beat the four deep learning methods. To our surprise, our empirical study shows that we actually have found two better approaches than traditional approaches.

The main contributions of this paper include:

- With the growing trend towards applying deep learning techniques in software engineering, we conduct an empirical study to examine whether deep learning is really superior to traditional approaches for a representative data intensive software engineering task – tag recommendation for software information sites. Our experiments on a broad range of datasets containing a total of 11,352,714 software objects demonstrate the effectiveness of some deep learning methods. The result shows that appropriate deep learning methods can indeed outperform well-tuned traditional methods in tag recommendation for software information sites.

- We tackle the problem of automated tag recommendation in large software infor-

mation sites. We propose 4 different new deep learning methods for this problem – TagCNN, TagRNN, TagHAN and TagRCNN – and compare their efficiency and effectiveness against three traditional approaches – EnTagRec, TagMulRec, and FastTagRec. Our comprehensive experiments demonstrate that our deep learning methods TagCNN and TagRCNN are better than these traditional approaches.

- We identify some guidelines for other software engineering tool developers from our experiences in the tag recommendation task that we help will be useful in developing their own deep learning-based solutions for data intensive software engineering tasks.

The rest of this paper is organized as follows: Section 2 presents the background and related works on tag recommendation for software information site and deep learning. Section 3 explains in detail the four deep learning methods. Section 4 describes the experimental settings of our study, including research questions, datasets, experimental design, and evaluation measures, and also the experimental results and limitations. In Section 5, we share some of the important lessons that we learned in implementing these deep learning methods, and discuss threats to the validity of our study. Finally Section 6 concludes the paper.

2. Background and Related Work

2.1. Tag Recommendation for Software Information Sites

Tags provide a type of metadata to search, describe, identify, bookmark, classify, and organize software objects in software information sites [16]. They are widely used in the developer Q&A and open source communities. A software object in developer Q&A community, such as `StackOverflow`, includes title, body, tags, comments and so on. A software object in open source community, such as `Freecode`, includes software project name, project description, tags and so on. When a developer posts a question in a Q&A community or shares a project in an open source community, software information sites usually require developers to classify their contents with

multiple tags at the time of posting. For example, `StackOverflow` suggests that developers attach at least three but no more than five tags per posting. `Freecode` allows developers to create more than ten tags for each posting.

Since software developers are free to choose tags, the words used for tags are arbitrary. Tags that are intended to represent the same meaning frequently use different words such as spaces vs. no spaces, upper cases vs. lower cases, acronym vs. full spelling, hyphens vs. no hyphens. Such phenomenon makes it difficult for software developers to search for existing tags, thus become more likely to use their own wording, which leads to even more synonymous tags with different spelling. Automated tag recommendation can alleviate the problem by recommending tags that have been used for semantically similar software objects.

Tag recommendation has been a popular research topic in the fields of social network and data mining [28, 29, 30, 31, 32]. Automatic tag recommendation in software engineering was first proposed by Al-Kofahi et al. in 2010 [16]. Al-Kofahi et al. proposed a method called TAGREC to automatically recommend tags for work items in IBM Jazz. TAGREC was based on the fuzzy set theory and considered the dynamic evolution of a system. Later a method called TAGCOMBINE [14] was proposed to automatically recommend tags for software objects in software information sites. It consists of three components: a multi-label ranking component, a similarity based ranking component, and a tag-term based ranking component. The multi-label ranking approach adopted by TAGCOMBINE limits its application to relatively small datasets. For a large-scale software information site such as `StackOverflow`, TAGCOMBINE has to train more than forty thousand binary classifier models and the size of each training set has more than ten million objects.

A more recent approach called EnTagRec [15] outperforms TAGCOMBINE in terms of $Recall$ and $Precision$ metrics. EnTagRec consists of two components: Bayesian inference component and Frequentist inference component. However, EnTagRec is not scalable as well, as it also utilizes all information in software information sites to recommend tags for a software object. Lately, a state-of-the-art tags recommendation method TagMulRec was proposed by Zhou et al. in 2017 [26]. For a given software object, TagMulRec prunes the large-scale categories (tags) into a much smaller set

of target category candidates for similarity distance computation. In addition, neither TAGCOMBINE nor EnTagRec adapts to the dynamic evolution of software information sites. In contrast, TagMulRec is scalable and is able to handle continuous updates in the software information sites. However, TagMulRec only utilizes a small portion of software information sites that is most relevant to a given software object. Recently, an advanced method called FastTagRec [27] outperforms TagMulRec in terms of efficiency and effectiveness. FastTagRec exploits a neural network approach that is based on single-hidden layer neural network and the rank of constraint of words. In order to avoid the limitation of TagMulRec, FastTagRec utilizes shared parameters among features and tags to utilize all information in software information sites.

In this paper, we only recommend tags which already exist in software information sites. All of the methods used to recommend tags in this paper are based only on the textual content of the software object, thus avoiding the ”cold-start problem” [33] and ”content-based tag recommendation” core issues in the machine learning community. We will consider these issues in our future works.

2.2. Deep Learning

Deep learning is an increasingly popular technique from machine learning building upon the concept of artificial neural networks. Feedforward neural networks represent a traditional neural network structure and lay the foundation for deep learning model structures [34]. Based on feedforward neural networks, deep learning model structures feature more complex network connections in a larger number of layers. However, the number of parameters in a fully connected feedforward neural network grows extremely large as the width and depth of the network increase. To address this limitation, researchers and industrial practitioners have proposed various deep learning model structures targeting different types of practical problems. For example, convolutional neural networks (CNN) have been the dominant approach for computer vision (CV) tasks [35]. For natural language process (NLP) tasks, recurrent neural networks (RNN) are particularly well suited for processing sequential text data [36]. Hierarchical attention networks (HAN) [24] can select qualitatively informative words and sentences based on hierarchical structure (words form sentences, sentences form a document) in

text classification task. However, RNN aims to put more weight on recent information, where later words are more dominant than earlier ones. Recurrent convolutional neural network (RCNN) [25] is proposed to address this limitation for text classification tasks. However, all of the above models depend on word vector models, that can translate a word into a numerical vector to provide input translation. The widely-used word vector methods include Word2Vec [37], FastText [38] and Glove [39]. Word2Vec is a shallow with two-layer neural networks model architecture which includes the continuous bag-of-words (CBOW) architecture and the continuous skip-gram architecture to produce a distributed representation of words. The FastText method is based on the CBOW model of the Word2Vec method. Glove method utilizes global word-word co-occurrence statistics from a corpus to produce a vector space with meaningful substructure.

Deep learning has made great progress in natural language processing, machine vision and other fields. Recently, deep learning has also been demonstrated in its effectiveness when applied to various types of recommender systems. These include video recommendation [40], App recommendation [41], news recommendations [42], etc. Many different types of deep learning techniques (CNN, RNN, HAN, Restricted Boltzmann Machine, Generative Adversarial Networks, Deep Reinforcement Learning, and etc) are applied to these recommendation systems [43, 44, 45, 46, 47, 48]. For comparison purpose, in this paper, we have applied popular deep learning models (CNN, RNN, HAN and RCNN) to the software object tag recommendation in software information sites. Comprehensive experimental results have been demonstrated for comparison purposes.

2.3. Deep learning in software engineering

Such state-of-the-art deep learning algorithms have recently seeded promising new methods in the software engineering field [1, 2, 3, 4, 5]. Since deep learning techniques have been successfully applied in various domains such as computer vision and natural language processing, great attention has been attracted from researchers and industrial practitioners in software engineering [49, 50, 51, 5, 26]. Various deep learning models have been applied to solve different software engineering tasks, such as API embedding representation, modeling source code, code clone detection, semantically enhanced

software traceability, predicting semantically linkable knowledge in developer online forums and so on. Deep learning models play two different categories of roles in these software engineering tasks. In the first category, deep learning models are used as a pre-processor. For example, Nguyen et al. proposed utilizing word2vec, a kind of deep learning model, to represent API as vectors for API usages and applications [1]. In the second category, deep learning models are used as a solution. For example, Xu et al. adopt neural language model and convolutional neural networks (CNN) to capture word and document-level semantics of knowledge units [5].

3. Our Approach

In this section, we first formally present the tag recommendation problem. We then propose four deep learning methods–TagCNN, TagRNN, TagHAN and TagRCNN– to solve the same problem.

3.1. Problem Formulation

A software information site is a set $S = \{o_1, \ldots, o_n\}$, where $o_i (1 \leq i \leq n)$ denotes a software object. For a developer Q&A site, the attributes of o_i consist of an identifier id, a body b, a title tt, and a set of tags T. For an open source site, the attribute of o_i consists of project name na, project description d, a set of tags T. If we treat the combination of the title tt and body b of a software object in a Q&A site as a project description d. We can assume that any software object o_i contain a description d and a set of tags T. These tags in a software information site S is a set $\mathcal{TA} = \{t_1, \ldots, t_m\}$ and the tags associated with an object o_i is a subset of $\mathcal{TA}$. The research question in tag recommendation task is the following: given a large set of existing software objects that are attached with tags, how to automatically recommend a set of appropriate tags for a new software object o_i.

3.2. TagCNN

We firstly propose a new deep learning-based TagCNN method for tag recommendation, as depicted in Figure 1. TagCNN is based on CNN [52, 53], a technique that has been proven successful for text classification domains elsewhere. This suggests

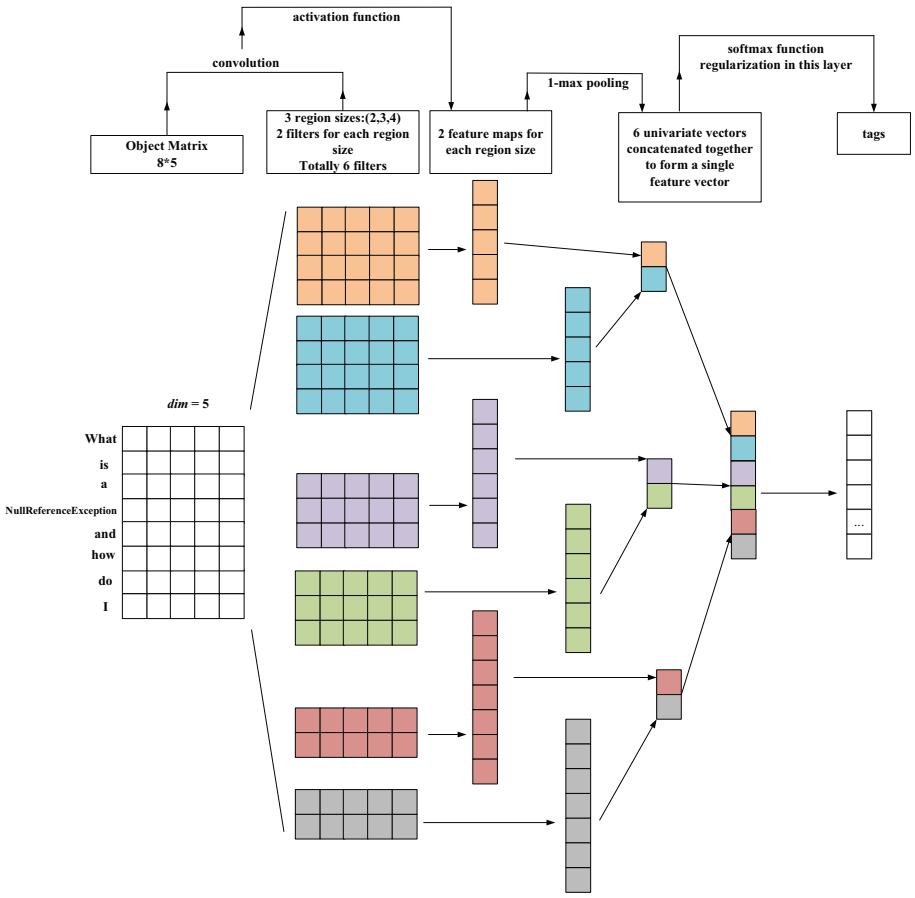

Figure 1: The Overall Architecture of TagCNN.

it could be a good choice for the tag recommendation domain. The major steps of TagCNN include the following:

1. Given a software object o_i, let $x_i \in \mathbb{R}^{dim}$ be the dim-dimensional word vector corresponding to the i-th word in the description $o_i.d$. A description of length n is represented as

$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n, \tag{1}$$

where $\oplus$ is the concatenation operator, and $x_{i:i+j}$ refers to the concatenation of words $x_i, x_{i+1}, \cdots, x_{i+j}$. It can be represented by a $n * dim$ matrix in Figure 1. These word vectors were trained by Mikolov'method [37].

2. A convolution operation involves a filter $f \in \mathbb{R}^{h \cdot dim}$ that is applied to a region of h words to produce a new feature. For example, a feature c_i is generated from a region of words $x_{i:i+h-1}$ by

$$c_i = tanh(f \cdot x_{i:i+h-1} + b). \tag{2}$$

Here $b \in \mathbb{R}$ is a bias term and $tanh$ is a non-linear hyperbolic tangent function. If h is bigger than n, we apply a matrix zero padding of size $(h - n) * dim$ to input layer.

$$c = tanh(f \cdot (x_{1:n} \oplus zero((h - n) * dim)) + b). \tag{3}$$

Each possible region of words in the description $\{x_{1:h}, x_{2:h+1}, \cdots, x_{n-h+1:n}\}$ apply this filter to produce a feature map

$$c = [c_1, c_2, \cdots, c_{n-h+1}], \tag{4}$$

with $c \in \mathbb{R}^{n-h+1}$.

3. TagCNN applies a 1-max-pooling operation over the feature map and take the maximum value $\hat{c} = max\{c\}$ as the feature corresponding to this particular filter. The first step is to capture the most important feature for each feature map. The pooling process naturally deals with variable description lengths. TagCNN uses multiple filters with different region sizes to get multiple features. These features form the penultimate layer $z = [\hat{c_1}, \hat{c_2}, \cdots, \hat{c_m}]$.

4. Loss function and activation function play different roles in deep learning network structure. Activation function provides the nonlinear modeling ability of the network. Loss function is used to measure the prediction quality of the network model. Sigmoid function maps a real value to an interval of (0, 1), so that it can be used in output layer for two classifications. Softmax function is a generalization of logistic function that squashes (maps) a T-dimensional vector z of arbitrary real values to a T-dimensional vector $\sigma(z)$ of real values in the range (0, 1) that add up to 1. So, softmax function can be used for multi-classification task according the size of $|T|$. Since the number of tags is greater than 2, softmax layer is used as the output layer in our paper. The penultimate layer z is passed to the fully connected softmax layer. TagCNN uses the softmax function to compute the probability distribution over tags. In probability theory, the output of the softmax layer is used to represent a probability distribution over different possible tags [54].

$$P = softmax(W \cdot z + B). \tag{5}$$

Here W is the weight vector and B is a bias term in the fully connected layer. Each dimension of $P \in \mathbb{R}^{\mathcal{TA}}$ represents the probability that the corresponding tag is recommended to the software object. The k tags with highest probability ranking are recommended to the software object o_i.

3.3. TagRNN

We now propose a new TagRNN method that is based on RNN techniques. Recurrent Neural Networks (RNN) [23] are one of the most popular architectures used for NLP-based problems. Again, this suggests they could be a good choice of technique

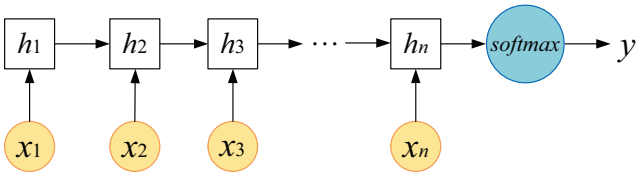

Figure 2: The Overall Architecture of TagRNN.

for the tag recommendation domain. Figure 2 shows the overall TagRNN structure for tag recommendation, which involves the following major steps.

1. The description $o_i.d$ is a text sequence $\{x_1, x_2, \cdots, x_n\}$ in the given software object o_i. n is the length of $o_i.d$ and $x_i \in \mathbb{R}^{dim}$ is the dim-dimensional word vector corresponding to the i-th word in $o_i.d$. We can get the vector representation (embeddings) x_i of the i-th word by the pre-trained word vectors [37].

2. TagRNN is able to process a text sequence of arbitrary length by recursively applying a transition function to its internal hidden state vector $h_i \in \mathbb{R}^H$ of the input sequence. The activation of hidden state h_i at the i-th step is computed as a rectified linear function $Relu$ [55] of the current input x_i and the previous hidden state h_{i-1}

$$h_i = \begin{cases} 0 & i = 0 \\ Relu(U \cdot h_{i-1} + V \cdot x_i + b) & otherwise \end{cases}. \tag{6}$$

   Here $V \in \mathbb{R}^{H*dim}$ is weight vector, $U \in \mathbb{R}^{H*H}$ is a weight parameter and $b \in \mathbb{R}^H$ is a bias term. The output at the last step h_n can be regarded as the representation the description $o_i.d$.

3. TagRNN has a fully connected layer followed by a softmax non-linear layer that predicts the probability distribution over tags.

$$P = softmax(W \cdot h_n + B). \tag{7}$$

   The k tags with highest probability ranking are recommended to the software object o_i.

3.4. TagHAN

We propose a new TagHAN method that is based on HAN models. Hierarchical attention networks (HAN) [24] can select qualitatively informative words and sentences in an NLP analysis task [24]. The overall architecture of TagHAN is given in Figure 3. It contains several parts: a word encoder, a word-level attention layer, a sentence

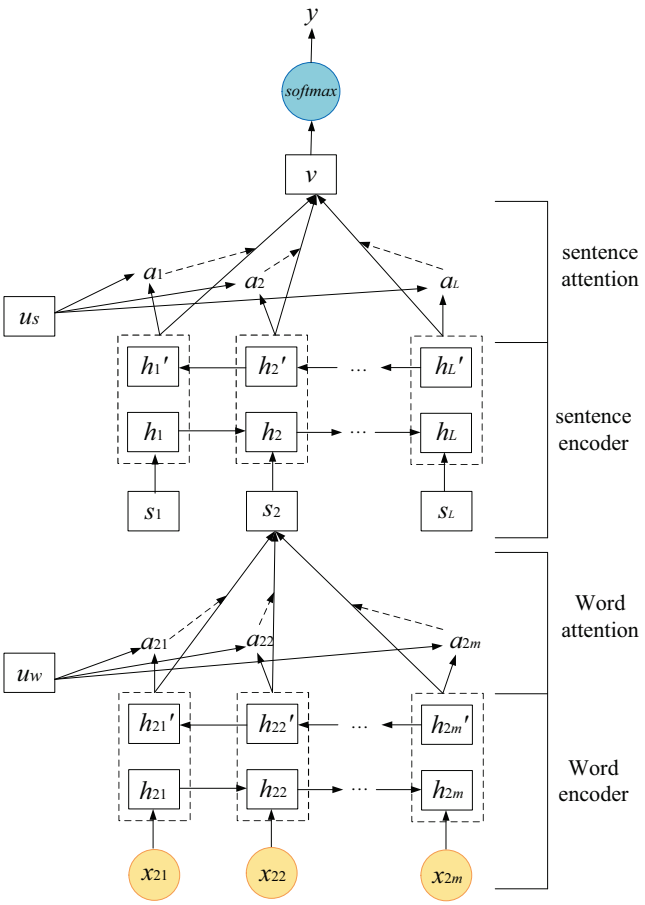

Figure 3: The Overall Architecture of TagHAN.

encoder and a sentence-level attention layer. We describe the details of different components in the following.

3.4.1. GRU-based sequence encoder

We first introduce the gated recurrent unit (GRU) [56] that is the basic unit in word and sentence encoder. Figure 4 shows the structure of GRU. GRU uses a gating mechanism to track the state of sequences. There are two types of gates: the reset gate $r_i (r_i \in \mathbb{R}^H)$ and the update gate $z_i (z_i \in \mathbb{R}^H)$. They together control how information is updated to the state. At the i-th word or sentence, GRU computes the new state as

$$h_i = (1 - z_i) \odot h_{i-1} + z_i \odot \tilde{h_i}. \tag{8}$$

This is a linear interpolation between the previous state $h_{i-1} (h_{i-1} \in \mathbb{R}^H)$ and the current new state $\tilde{h_i} (\tilde{h_i} \in \mathbb{R}^H)$. $\odot$ denotes element-wise multiplication. The gate z_i decides how much past information is kept and how much new information is added. z_i is updated by:

$$z_i = \sigma \left(W_z x_i + U_z h_{i-1} + b_z \right). \tag{9}$$

σ is the sigmoid activation function [57], here $W_z (W_z \in \mathbb{R}^{H*dim})$ is weight vector, $U_z (U_z \in \mathbb{R}^{H*H})$ is a weight parameter, $b_z \in \mathbb{R}^H$ is a bias term and $x_i (x_i \in \mathbb{R}^{dim})$ is the input word vector. The candidate state $\tilde{h_i}$ is computed in a way similar to a RNN:

$$\tilde{h_i} = tanh \left(W_h x_i + r_i \odot (U_h h_{i-1}) + b_h \right), \tag{10}$$

Here r_i is the reset gate which controls how much the past state contributes to the candidate state, $W_h (W_h \in \mathbb{R}^{H*dim})$ is weight vector, $U_h (U_h \in \mathbb{R}^{H*H})$ is a weight

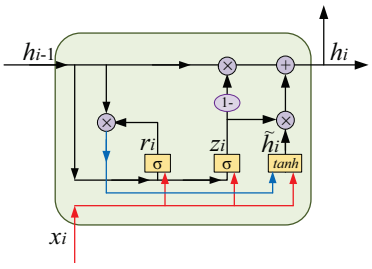

Figure 4: The structure of GRU.

parameter, $b_h \in \mathbb{R}^H$ is a bias term. If r_i is zero vectors, then it forgets the previous state. The reset gate is updated as follows:

$$r_i = \sigma\left(W_r x_i + U_r h_{i-1} + b_r\right). \tag{11}$$

Here $W_r (W_r \in \mathbb{R}^{H*dim})$ is weight vector, $U_r (U_r \in \mathbb{R}^{H*H})$ is a weight parameter, $b_r \in \mathbb{R}^H$ is a bias term.

3.4.2. Hierarchical Attention

For a software object o, we assume that the description $o.d$ has L sentences s_i and each sentence contains m_i words. x_{ip} represents the p-th word's vector corresponding to the i-th sentence in $o.d$. We can get the vector representation x_{ip} by the pre-trained word vectors [37]. TagHAN projects the description $o.d$ into a vector representation. In the following, we will present how we build the vector progressively from word vectors by using the hierarchical structure.

Word Encoder. We use a bidirectional GRU to get annotations of words by summarizing information from both directions for words. The bidirectional GRU contains the forward GRU which reads the sentence s_i from x_{i1} to x_{im} and a backward GRU which reads from x_{im} to x_{i1} :

$$h_{ip} = \overrightarrow{GRU}(x_{ip}), p \in [1, m], \tag{12}$$

$$(h'_{ip} = \overleftarrow{GRU}(x_{ip}), p \in [1, m]. \tag{13}$$

We obtain an annotation for a given word x_{ip} by concatenating the forward hidden state h_{ip} and back hidden state h'_{ip}, i.e., $\left[h_{ip}, h'_{ip}\right]$, which summarizes the information of the whole sentence centered around x_{ip}.

Word Attention. Not all words contribute equally to the representation of the sentence meaning. Hence, we introduce attention mechanism to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector. Specifically:

$$u_{ip} = tanh\left(W_w \left[h_{ip}, h'_{ip}\right] + b_w\right), \tag{14}$$

$$a_{ip} = \frac{exp(u_{ip}^{\top} u_w)}{\sum_k exp(u_{ip}^{\top} u_w)}, \tag{15}$$

$$s_i = \sum_p a_{ip} \left[h_{ip}, h'_{ip}\right]. \tag{16}$$

That is, we first feed the word annotation $\left[h_{ip}, h'_{ip}\right]$ through one-layer perception to get u_{ip} as a hidden representation of $\left[h_{ip}, h'_{ip}\right]$, then we measure the importance of the word as the similarity of u_{ip} with a word level context vector u_w and get a normalized importance weight a_{ip} through a softmax function. After that, we compute the sentence vector s_i as a weight sum of the word annotations based on the weights.

Sentence Encoder. For a description $o.d$, we can get the vector in a similar way. We use a bidirectional GRU to encode the sentence:

$$h_i = \overrightarrow{GRU}(s_i), i \in [1, L], \tag{17}$$

$$h'_i = \overleftarrow{GRU}(s_i), i \in [L, 1]. \tag{18}$$

We concatenate h_i and h'_i to get an annotation of sentence i, i.e., $[h_i, h'_i]$.

Sentence Attention. To reward sentences that are important to correctly tag a software object, we again use attention mechanism and introduce as sentence level context vector u_s and use the vector to measure the importance of the sentences. Specifically:

$$u_i = tanh\left(W_s \left[h_i, h'_i\right] + b_s\right), \tag{19}$$

$$a_i = \frac{exp(u_i^{\top} u_s)}{\sum_i exp(u_i^{\top} u_s)}, \tag{20}$$

$$v = \sum_i a_i \left[h_i, h'_i\right]. \tag{21}$$

Here, v is the vector that summarizes all the information of sentences in a description of software object.

3.4.3. Tag Recommendation

The vector v is a high level representation of a description of software object and can be used as features for tag recommendation. Finally, TagHAN has a fully connected layer followed by a softmax non-linear layer that predicts the probability distribution over tags.

$$P = softmax(W_c v + b_c). \tag{22}$$

The k tags with highest probability ranking are recommended to the software object o.

3.5. TagRCNN

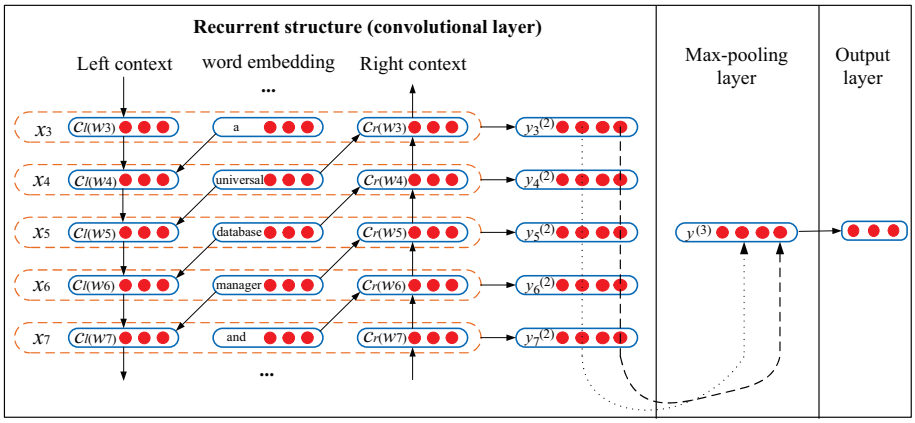

Figure 5: The Overall Architecture of TagRCNN.

Finally, we propose TagRCNN, a new tag recommendation method that is based on RCNN. As RNN aims to put more weight on recent information, where later words are more dominant than earlier ones, a Recurrent Convolutional Neural Network (RCNN) [25] can be used to address this limitation. The overall architecture of TagRCNN is shown in Figure 5. The input of TagRCNN is the description d of a software object o, which is a sequence of words $w_1, w_2, \cdots, w_n$. The output of TagRCNN contains tags. We use $p(t \mid o, \theta)$ to denote the probability of the software object being labeled tag t, where θ is the parameters in the model and o is a given software object. We describe the details of each step in the following.

1. TagRCNN uses a recurrent structure, a bi-directional recurrent neural network, to capture the contexts of a word. TagRCNN combines a word and its context to represent a word. We define $c_l(w_i)(c_l(w_i) \in \mathbb{R}^{|c|})$ as the left context of word w_i and $c_r(w_i)(c_r(w_i) \in \mathbb{R}^{|c|})$ as the right context of word w_i. Both $c_l(w_i)$ and $c_r(w_i)$ are dense vector with $|c|$ float value elements. In the experiments of this paper, $|c|$ is set to 50. The left context $c_l(w_i)$ of word w_i is calculated by Equation 23, where $e(w_{i-1})(e(w_{i-1}) \in \mathbb{R}^{|e|})$ is the word embedding of word w_{i-1}, which is a dense vector with $|e|$ float value elements. $c_l(w_{i-1})$ is the left context of the previous word w_{i-1}. The left context for the first word in any description uses the same shared parameters $c_l(w_1)$. $W^l(W^l \in \mathbb{R}^{|c|\times|c|})$ is a matrix that transforms the hidden layer into next hidden layer and $W^{sl}(W^{sl} \in \mathbb{R}^{|c|\times|e|})$ is a matrix that is used to combine the semantic of the current word with the next word's context. f is a non-linear activation function. The right context $c_r(w_i)$ is get in a similar manner, as shown in Equation 24. The right contexts of the last word in a description share the parameters $c_r(w_n)$

$$c_l(w_i) = f(W^l c_l(w_{i-1}) + W^{sl} e(w_{i-1})). \tag{23}$$

$$c_r(w_i) = f(W^r c_r(w_{i+1}) + W^{sr} e(w_{i+1})). \tag{24}$$

   Here $W^r(W^r \in \mathbb{R}^{|c|\times|c|})$ and $W^{sr}(W^{sr} \in \mathbb{R}^{|c|\times|e|})$ are weight matrixes.

2. TagRCNN defines the representation of word w_i in Equation 25, which is the concatenation of the left context $c_l(w_i)$, the word embedding $e(w_i)$ and the right context $c_r(w_i)$. The recurrent structure can obtain all c_l in a forward scan of a description and c_r in a backward scan of the description.

$$x_i = [c_l(w_i) : e(w_i) : c_r(w_i)]. \tag{25}$$

   After obtaining the representation $x_i(x_i \in \mathbb{R}^{|e|+2|c|})$ of the word w_i, TagRCNN applies a linear transformation together with the tanh activation function to x_i and send the result to the next layer.

$$y_i^{(2)} = tanh(W^{(2)} x_i + b^{(2)}). \tag{26}$$

$y_i^{(2)}(y_i^{(2)} \in \mathbb{R}^{|H|})$ is the latent semantic vector, in which each semantic factor will be analyzed to determine the most useful factor for representing the description. $|H|$ is a hyper-parameter. In our experiments of this paper, $|H|$ is set to 100. $W^{(2)}(W^{(2)} \in \mathbb{R}^{|H|\times(|e|+2|c|)})$ is a weight matrix and $b^{(2)}(b^{(2)} \in \mathbb{R}^{|H|})$ is a bias vector.

3. When all of the representation of words are calculated, a max-pooling layer is applied.

$$y_i^{(3)} = \max_{i=1}^{n} y_i^{(2)}. \tag{27}$$

The max function is an element-wise function. The q-th element of $y_i^{(3)}(y_i^{(3)} \in \mathbb{R}^{|H|})$ is the maximum in the q-th element of $y_i^{(2)}$. The pooling layer converts descriptions with various lengths into a fixed-length vector. With the pooling layer, TagRCNN captures the information throughout the entire description of software object. The max-pooling layer attempts to find the most important latent semantic factors in a description. The last part of TagRCNN is an output layer. Similar to traditional neural networks, it is defined as:

$$y^{(4)} = W^{(4)}y^{(3)} + b^{(4)}. \tag{28}$$

Here $y^{(4)}(y_i^{(4)} \in \mathbb{R}^{|\mathcal{TA}|})$ is the output vector. $W^{(4)}(W^{(4)} \in \mathbb{R}^{|\mathcal{TA}|\times|H|})$ is a weight matrix and $b^{(4)}(b^{(4)} \in \mathbb{R}^{|\mathcal{TA}|})$ is a bias vector.

4. TagRCNN applies the softmax function to $y^{(4)}$ to get the probability distribution over tags. The k tags with highest probability ranking are recommended to the software object o.

$$P = \frac{exp(y_i^{(4)})}{\sum_{q=1}^{n} exp(y_q^{(4)})}. \tag{29}$$

4. Experimental Design

In order to investigate whether any of our proposed deep learning-based tag recommendation approaches can outperform three traditional approaches we conducted an empirical study to answer the following two research questions:

- **RQ1:** Which of the four proposed methods – TagCNN, TagRNN, TagHAN and TagRCNN – is better than the three traditional approaches – EnTagRec, TagMulRec, and FastTagRec – for the software site tag recommendation problem?
- **RQ2:** Is the computational cost of the model training for the proposed method acceptable?

4.1. Datasets and Experiment Setup

Our experiments datasets come from ten software information sites. We evaluated our four deep learning methods and three traditional approaches on one large-scale software information site `StackOverflow`, 3 medium-scale software information sites `Askubuntu`, `Serverfault`, `Unix` and 6 small-scale sites `Codereview`, `Freecode`, `Database Administrator`, `Wordpress`, `AskDifferent` and `Software Engineering`.

For `StackOverflow`, we selected software objects posted before July 1st, 2014, the same date setting as used in [26] to facilitate comparison in our empirical study. For the other 9 datasets, we consider all the software objects posted before Dec 31st, 2016. We define a site as a large-scale site if the number of software objects in the site is more then 1 million, as a medium-scale site if the number of software objects in the site is between 100k to 1 million, and as a small-scale site if the number of software objects in the site is less than 100k.

We used the same data preprocessing rules as in [26]. First, there is no conversion operation applied to tags other than that all tags are converted to lowercase. Then we remove rare tags and software objects. A tag is *rare* if its number of appearances is less than or equal to a predefined threshold ts. A software object is removed if all its tags are rare. The threshold value ts 50 was used in prior work [14, 15, 26]. In this paper, we also set the threshold values of ts to 50 for ten software information sites. Table 1 summarizes the number of tags and software objects after removing the rare ones under threshold values 1 and 50. It can be observed from Table 1 that the number of software objects ranges from about 40K to more than ten million for these software information sites. When the threshold value is changed from 1 to 50, 1.811% of the objects are removed and 66.004% of the tags are removed from these sites. Second,

for these remaining software objects in Table 1, we further remove code snippets and screen shots form their descriptions. The two preprocessing steps were also used in prior work [14, 15, 26]. That is, only the text in the description is preserved. After the two preprocessing steps, these preprocessed datasets are used to train and test tag recommendation models.

In order to be consistent with previous researches [14, 15, 26, 27], in our experiments, we also performed a ten-round validation on each dataset to evaluate TagCNN, TagRNN, TagHAN and TagRCNN. For each preprocessed dataset, we randomly selected 10,000 software objects and treated them as a test set V. The remaining software objects in a dataset were treated as a training set and used to recommend tags for the 10,000 selected ones. For an information site, all text in the training set is used to train word vectors by Word2Vec [37]. In our experiments, we set the dimension of word vector to 300 for large-scale site, 200 for medium-scale sites, and 100 for small-scale sites. For an object o with $|T|$ tags in training set, we code tag t_i using one-hot encoding and get $|T|$ training instances $\{(o.d, t_1), ..., (o.d, t_{|T|})\}$. When a new tag is added, we add the size of the output layer for new tags. As the penultimate layer to the output layer is the full connection layer, the model only needs to add these weight vectors associated with new tags.

For the TagCNN method, we set the region sizes to (2, 3, 4, 5), and 25 filters for each region size. For the TagRNN method, the hyper-parameter $|H|$ is set to 128 for small-scale and medium-scale sites and 256 for large-scale site. For the TagHAN method, the hyper-parameter $|H|$ of GRU is set to 50. For the TagRCNN method, the hyper-parameter $|H|$ is set to 100 and the hyper-parameter $|c|$ is set to 50. For the TagCNN, TagHAN and TagRCNN method, in order to reduce memory requirements and speed up the training time, the max sequence length msl is set to 400 for a large-scale site and 600 for medium-scale and small-scale sites. If the word sequence length in the description d of an object o is over the max sequence length msl, we only take the first msl words of the description d. For the TagRNN method, we don't set the max sequence length and use all words in the description of an object as the input of the model.

For each software object $o_i \in V$, we recommend k tags to form a tag set TR_i^k. We

Table 1: Statistics of the ten datasets on different rare tag threshold values

Site Name	ts	#software object	#tags
StackOverflow	1	11203032	44265
	50	11193348	18952
Askubuntu	1	248630	3041
	50	246138	1146
Serverfault	1	232996	3482
	50	231319	1312
Unix	1	104744	2407
	50	103243	770
Codereview	1	39989	909
	50	39811	302
Freecode	1	47978	9018
	50	43644	274
Database Administrator	1	51031	969
	50	50687	293
Wordpress	1	71338	770
	50	70491	403
AskDifferent	1	77978	1049
	50	77503	469
Software Engineering	1	42782	1628
	50	41531	418

repeated the process ten times and compare TagCNN, TagRNN, TagHAN and TagRCNN against TagMulRec. All experiments were conducted on a 64-bit, Intel Core i7 3.6G desktop computer with 64G RAM running Ubuntu 16.04. We used the open source software library `Tensorflow`[8] to implement our methods TagCNN, TagRNN, TagHAN and TagRCNN. These code, parameters and configurations of these methods, and all datasets in experiments can be accessed via the link https://pan.baidu.com/s/1pKCpodP.

[8]http://www.tensorflow.org

4.2. Evaluation Metrics

To be consistent with the experiments reported in [15, 26, 27], we use the top-k prediction recall, the top-k prediction precision, and the top-k prediction $F1\text{-}score$ when evaluating the performance of TagCNN, TagRNN, TagHAN and TagRCNN. The Top-k prediction recall is the percentage of tags selected out of the recommended lists TR_i^k in the software objects true tags. Given a software object o_i, the top-k prediction recall $Recall@k_i$ is computed by Equation 30. Given a set V of software objects, the top-k prediction recall $Recall@k$ is computed by Equation 31.

$$Recall@k_i = \begin{cases} Recall@k_i = \frac{|TR_i^k \cap o_i.T|}{K}, \mid o_i.T \mid > k. \\ Recall@k_i = \frac{|TR_i^k \cap o_i.T|}{|o_i.T|}, \mid o_i.T \mid \leq k. \end{cases} \tag{30}$$

$$Recall@k = \frac{\sum_{i=1}^{|V|} Recall@k_i}{\mid V \mid} \tag{31}$$

The Top-k prediction precision is the percentage of a software objects tags that are in the recommended list TR_i^k. Given a software object o_i, the top-k prediction precision $Precision@k_i$ is defined by Equation 32. Given a set V of software objects, the top-k prediction precision $Precision@k$ is computed by Equation 33.

$$Precision@k_i = \frac{\mid TR_i^k \cap o_i.T \mid}{k} \tag{32}$$

$$Precision@k = \frac{\sum_{i=1}^{|V|} Precision@k_i}{\mid V \mid} \tag{33}$$

However, higher precision rate is not the main target in our scenario. The $Precision@k_i$ is inversely proportional to the k value. The k value indicates the number of tags that we want to recommend to the developer. For example, if a software object has two tags and they are both among the top five tags recommended to the developer (namely the k value is 5), the precision value is 40%. However, if the k value increases to 10, then the precision value will only be 20%. Therefore, lower precision rate in this work does not imply there are many wrong tags but the number of tags recommended to the developer is larger. Clearly, a good precision rate in our work indicates that a reasonable number

of tags has been recommended to the developer, which is much more user friendly than recommending a long list of tags.

The Top-k prediction F1-score combines Top-k prediction recall and Top-k prediction precision. Given a software object o_i, the Top-k prediction F1-score $F1\text{-}score@k_i$ is defined by Equation 34. Given a set V of software objects, the top-k prediction F1-score $F1\text{-}score@k$ is defined by Equation 35.

$$F1 - score@k_i = 2 \cdot \frac{Precision@k_i \cdot Recall@k_i}{Precision@k_i + Recall@k_i} \tag{34}$$

$$F1 - score@k = \frac{\sum_{i=1}^{|V|} F1 - score@k_i}{\mid V \mid} \tag{35}$$

4.3. Experimental Results

In this section, we present our experimental results to answer the aforementioned research questions.

*4.3.1. **RQ1:** Which of the four proposed methods – TagCNN, TagRNN, TagHAN and TagRCNN – is better than the three traditional approaches – EnTagRec, TagMulRec, and FastTagRec, for the software site tag recommendation problem?*

Motivation. Our four approaches (TagCNN, TagRNN, TagHAN and TagRCNN) are based on four different deep learning models that are very different from the three traditional approaches EnTagRec, TagMulRec, and FastTagRec. The answer to this research question would shed light on whether any of the chosen contemporary deep learning techniques can improve the performance of our software site tag recommendation task.

Approach. For each preprocessed dataset, we applied our four approaches and the three traditional approaches to train tag recommendation models respectively and then used the test set V to evaluate the performance of these models. We compared the $Recall@k$, $Precision@k$ and $F1\text{-}score@k$ (k = 5 and 10) metrics of different approaches.

Results. Tables 2, 3, and 4 give the results in terms of $Recall@k$, $Precision@k$ and $F1\text{-}score@k$ (k = 5 and 10) metrics, respectively. In these tables, Column 1 lists the names of the software information sites and Column 2, 3, 4 lists the results of the

three traditional approaches EnTagRec, TagMulRec, and FastTagRec. For the approach EnTagRec, we list the experimental results on small-scale sites. Columns 5, 6, 7 and 8 list the results of our four approaches TagCNN, TagRNN, TagHAN and TagRCNN. The highest scores for each row in these tables are marked in bold.

It can be observed that *TagCNN and TagRCNN outperform traditional approaches* in terms of $Recall@k$, $Precision@k$, $F1\text{-}score@k$ on all the k settings in all the ten software information sites. For one large-scale and three medium-scale software information sites, TagRCNN almost achieved the best performance. For six small-scale software information sites, TagCNN achieved the best performance. Wilcoxon signed-rank test [58, 59] confirms that the performance improvements of TagCNN and TagRCNN methods are statistically significant (p-value < 0.001).

However, the *performance of TagRNN and TagHAN approaches are not as good as these traditional approaches* on some software information sites. For the TagRNN method, we assume that the reason for the lack of good results is that TagRNN aims to put more weight on recent information, where later words are more dominant than earlier ones. Because TagHAN method is based on the hierarchical structure of description text. For the TagHAN method, we assume that the reason for the failure to achieve good results is that the length of the description text in most software objects is short. For TagCNN and TagRCNN methods, we assume that the reason for the good results is that both of two methods contain the convolutional layer and max-pooling layer. The convolutional operation can eliminate bias and the max-pooling operation can fairly determine discriminative phrases in a text [25].

In addition, to evaluate the K most popular tags method which is a obvious method, we randomly extract 100k software objects with tags from stack-overflow data set. The example experimental results on the K (K = 5 and 10) most popular tags is shown in Tables 5. The experimental results shows that the performance of the K most popular tags method is poor compared with other methods.

Our experimental results show that TagCNN and TagRCNN, which are based on deep learning models CNN and RCNN respectively, significantly outperform the three traditional approaches EnTagRec, TagMulRec, and FastTagRec on all ten software information sites. TagRNN and TagHAN, which are based on deep learning models RNN and HAN respectively, are not as good as these traditional approaches on some of the software information sites.

Table 2: Four deep learning approaches vs. three traditional approaches in terms of Recall@k (k = 5 and 10)

	Recall@5						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	0.640	-	0.698	0.711	0.672	0.731	**0.877**
Askubuntu	0.603	-	0.684	0.656	0.603	0.624	**0.849**
Serverfault	0.622	-	0.666	0.708	0.633	0.650	**0.851**
Unix	0.604	-	0.627	0.793	0.580	0.617	**0.795**
Codereview	0.718	0.707	0.758	**0.956**	0.459	0.571	0.765
Freecode	0.659	0.641	0.588	**0.902**	0.332	0.374	0.756
Database Administrator	0.666	0.672	0.692	**0.958**	0.625	0.667	0.805
Wordpress	0.605	0.801	0.632	**0.873**	0.622	0.646	0.797
AskDifferent	0.708	0.878	0.689	**0.947**	0.639	0.663	0.838
Software Engineering	0.594	0.564	0.582	**0.936**	0.439	0.501	0.700
	Recall@10						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	0.749	-	0.774	0.804	0.765	0.822	**0.924**
Askubuntu	0.721	-	0.770	0.791	0.714	0.737	**0.898**
Serverfault	0.716	-	0.753	0.828	0.740	0.756	**0.896**
Unix	0.682	-	0.722	**0.899**	0.690	0.727	0.848
Codereview	0.788	0.773	0.820	**0.980**	0.539	0.675	0.814
Freecode	0.758	0.751	0.692	**0.949**	0.416	0.484	0.816
Database Administrator	0.778	0.794	0.816	**0.978**	0.742	0.777	0.884
Wordpress	0.725	0.864	0.765	**0.945**	0.731	0.760	0.863
AskDifferent	0.827	0.940	0.815	**0.978**	0.741	0.768	0.897
Software Engineering	0.704	0.697	0.708	**0.961**	0.535	0.599	0.764

4.3.2. ***RQ2:*** *Is the computational cost of the model training for the proposed method acceptable?*

Motivation. The four deep learning-based models (TagCNN, TagRNN, TagHAN and TagRCNN) and the three models based on traditional approaches (EnTagRec, TagMulRec, and FastTagRec) all need to be trained before they can be used for tag recom-

Table 3: Four deep learning approaches vs. three traditional approaches in terms of Precision@k (k = 5 and 10)

	Precision@5						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	0.343	-	0.386	0.396	0.305	0.301	**0.487**
Askubuntu	0.271	-	0.346	0.329	0.311	0.311	**0.441**
Serverfault	0.305	-	0.344	0.369	0.343	0.335	**0.466**
Unix	0.294	-	0.309	0.399	0.303	0.302	**0.416**
Codereview	0.377	0.371	0.398	**0.520**	0.256	0.281	0.421
Freecode	0.383	0.379	0.343	**0.530**	0.211	0.218	0.478
Database Administrator	0.313	0.324	0.332	**0.479**	0.315	0.318	0.404
Wordpress	0.265	0.348	0.278	**0.397**	0.283	0.285	0.374
AskDifferent	0.372	0.365	0.357	**0.506**	0.342	0.344	0.450
Software Engineering	0.253	0.245	0.252	**0.436**	0.217	0.221	0.325
	Precision@10						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	0.205	-	0.217	0.232	0.182	0.171	**0.279**
Askubuntu	0.166	-	0.198	0.203	0.188	0.187	**0.233**
Serverfault	0.179	-	0.198	0.222	0.205	0.200	**0.245**
Unix	0.169	-	0.182	**0.231**	0.184	0.183	0.222
Codereview	0.211	0.207	0.218	**0.268**	0.153	0.170	0.224
Freecode	0.245	0.239	0.219	**0.297**	0.138	0.151	0.258
Database Administrator	0.188	0.194	0.201	**0.246**	0.190	0.189	0.222
Wordpress	0.163	0.186	0.173	**0.219**	0.170	0.172	0.203
AskDifferent	0.222	0.201	0.216	**0.263**	0.202	0.203	0.240
Software Engineering	0.153	0.151	0.157	**0.225**	0.135	0.136	0.177

Table 4: Four deep learning approaches vs. three traditional approaches in terms of F1−score@k (k = 5 and 10)

	F1−score@5						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	0.444	-	0.476	0.509	0.420	0.426	**0.626**
Askubuntu	0.374	-	0.437	0.439	0.410	0.415	**0.581**
Serverfault	0.403	-	0.435	0.485	0.445	0.442	**0.602**
Unix	0.395	-	0.397	0.531	0.398	0.406	**0.546**
Codereview	0.494	0.485	0.502	**0.674**	0.329	0.376	0.543
Freecode	0.485	0.476	0.434	**0.668**	0.258	0.275	0.586
Database Administrator	0.426	0.438	0.449	**0.639**	0.419	0.430	0.538
Wordpress	0.368	0.484	0.386	**0.546**	0.389	0.396	0.509
AskDifferent	0.488	0.514	0.471	**0.660**	0.446	0.453	0.586
Software Engineering	0.355	0.340	0.352	**0.595**	0.290	0.307	0.443
	F1−score@10						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	0.310	-	0.329	0.360	0.294	0.283	**0.429**
Askubuntu	0.270	-	0.303	0.323	0.297	0.299	**0.370**
Serverfault	0.287	-	0.304	0.350	0.321	0.316	**0.385**
Unix	0.271	-	0.282	**0.367**	0.291	0.292	0.352
Codereview	0.333	0.325	0.335	**0.421**	0.239	0.271	0.351
Freecode	0.364	0.360	0.332	**0.453**	0.208	0.230	0.392
Database Administrator	0.201	0.310	0.323	**0.393**	0.303	0.304	0.355
Wordpress	0.266	0.305	0.283	**0.356**	0.276	0.280	0.328
AskDifferent	0.350	0.330	0.342	**0.415**	0.317	0.321	0.378
Software Engineering	0.252	0.248	0.257	**0.365**	0.216	0.221	0.287

Table 5: The K most popular tags method in terms of $Recall@k$, $Precision@k$, and $F1\text{-}score@k$ (K = 5 and 10)

Dataset	Recall@5	Precision@5	F1−score@5
StackOverflow	0.096	0.054	0.067
Dataset	Recall@10	Precision@10	F1−score@10
StackOverflow	0.138	0.078	0.097

mendation. For training these deep learning-based models, we first train word vectors through Word2Vec as the input of these models. The cost of training word vectors is included into the computational cost of these deep learning-based models training. Training of EnTagRec, TagMulRec, FastTagRec, TagCNN, TagRNN, TagHAN and TagRCNN, is done only once and off-line. After model training, using these models to recommend tags takes negligible time on-line. Understanding the training time cost helps us to better understand the scalability of each approach.

Approach. For the four deep learning-based models TagCNN, TagRNN, TagHAN and TagRCNN, we record the start time and the end time of the program for training word vectors and models to obtain the training time. For the three traditional models EnTagRec, TagMulRec and FastTagRec, we record the start time and the end time of the program for training models to obtain the training time.

Results. Table 6 gives the training cost of EnTagRec, TagMulRec, FastTagRec, TagCNN, TagRNN, TagHAN and TagRCNN models. Compared with TagMulRec on ten sites, the average training cost of TagRNN is 2.1 times that of TagMulRec, the average training cost of TagHAN is about 5.8 times that of TagMulRec, the average training cost of TagCNN is about 5.7 times that of TagMulRec and the average training cost of TagRCNN is about 19.3 times that of TagMulRec. Compared with four deep learning-based approaches on six small-scale sites, the average training cost of EnTagRec is about 169 times that of TagRNN, 64 times that of TagHAN, 53 times that of TagCNN and 14 times that of TagRCNN. Compared with FastTagRec on ten sits, the average training cost of TagRNN is 9.2 times that of FastTagRec, the average training cost of TagHAN is 25.3 times that of FastTagRec, the average training cost of TagCNN is 27.8 times that of FastTagRec and the average training cost of TagRNN is 101.4 times that of FastTagRec. Because the training cost is a one-time expense, we conclude that the training costs of TagCNN and TagRCNN are acceptable.

Training of tag recommendation models can be done off-line and only needs to be done once. TagCNN and TagRCNN are practical for use on the various-scale datasets

Table 6: Four deep learning approaches vs. three traditional approaches in terms of the training model time

	Training Time(s)						
Site Name	TagMulRec	EnTagRec	FastTagRec	TagCNN	TagRNN	TagHAN	TagRCNN
StackOverflow	41808	-	28783	213303	84433	190156	203770
Askubuntu	908	-	372	5466	1682	4601	5111
Serverfault	246	-	87	1467	537	1392	5879
Unix	621	-	101	3585	1038	2812	8772
Codereview	495	51959	83	2860	857	2329	6956
Freecode	95	55963	48	581	452	1319	1494
Database Administrator	311	73765	32	1776	566	1580	6737
Wordpress	361	91229	34	2059	633	1771	9065
AskDifferent	211	99204	90	1273	227	551	6417
Software Engineering	205	54160	44	1194	441	1295	7751

5. Discussion

In this section, we will first share some of the important lessons that we learned in implementing the work in this paper, followed by the discussion on threats to Validity.

5.1. Implementation of Deep Learning-based Methods

Many typical software engineering problems can be regarded as classification problems. Classification problems include binary classification problems, multiple classification problems and multi-label classification problems. For example, defect prediction [60], code clone detection [3], missing issue-commit link recovery [61] are binary classification problems, where one object is associated to one of the two categories; predicting semantically linkable knowledge in developer online forums [5], and program classification problem [62] are multiple classification problems, where one object is associated to one of the multiple categories (large than two); automated tag recommendation for software information sites [16, 14, 15, 26] is a multi-label classification problem, where one object could be associated with several categories simultaneously. Obviously, the research problem in this paper is a multi-label classification problem. For classification problems, high quality vector representations of the observed object or artifact can significantly improve the deep learning-based model accuracy [37, 63]. However, for a typical software engineering problem that is regarded as classification

problem, how to best represent the software object or artifact with a high quality vector is still a major challenge.

In this paper, for each software object, we utilized four different deep neural network structures (TagCNN, TagRNN, TagHAN and TagRCNN) to obtain different vector representations for use by each different deep learning-based model. TagRNN aims to put more weight on recent information where later words are more dominant than earlier ones, and hence the vector representation of a software object obtained by the TagRNN method is also a representation that puts more weight on recent information. TagHAN method is based on the hierarchical structure of text and the quality of the vector will not be ideal if the text is short. Therefore, the major reason for our failure to achieve a high quality vector representation of a software object with the TagHAN method is that the length of the description text in most software objects is short. For TagCNN and TagRCNN methods, the quality of the vector representation of a software object is high as both of these methods contain the convolutional layer and the max-pooling layer. The convolutional operation can eliminate bias and the max-pooling operation can fairly determine discriminative phrases in a text [25].

Given our experiences in implementing these deep learning methods, we see that a high quality vector representation of a software object or artifact can greatly improve performance of typical software engineering problems which can be formulated as classification problems. Further research is needed in software engineering to improve representation of such software artifacts and relationships for deep learning-based applications [64].

5.2. Threats to Validity

There are several threats that can potentially affect the validity of our experimental results.

Experimental Errors. Threats to internal validity relate to errors in our experimental data, approaches implementation and settings. We have double checked our experimental datasets, approaches implementation or re-implementation and settings, however there could be experimental errors that we did not detect.

Biased Results. Our tag recommendation task assumes that existing tags in a software information site are correct. However, human errors are inevitable. We do apply some filtering rules, such as removing rare tags and software objects, to alleviate the problem. These filtering rules have also been used in past research [14, 15], [26, 27]. However, this issue, such as how to deal with large number of synonymous tags, cannot be completely solved.

Generalizability. In this paper, we evaluated our four approaches and three traditional approaches on ten software information sites. There are more than 11,000,000 software objects in the ten datasets, and the number of software objects in the ten datasets ranges from about 40K to more than ten million. Even so, more case studies are needed to reduce this external validity threat. In the future, more software information sites need to be used.

Evaluation Metrics. In this paper, $Recall@k$, $Precision@k$ and $F1\text{-}score@k$ are used as our evaluation metrics for the algorithms. $Recall@k$ and $Precision@k$ have been used in the past to evaluate the performance of tag recommendation for software information sites [14, 15, 16] and for social media and network [65, 66, 67, 68]. The computational costs of constructing tag recommendation models are also used in this paper. Because of differences in operating systems, hardware, the development environment and others, the computational costs may be not suitability in repeated our experiments.

Scalability. We applied our algorithms to one large software information site, StackOverflow. Further scalability evaluation is needed to determine if both the online tag recommendation speed and offline training costs are acceptable for very large software information sites.

Model Components. We only utilized the text part of the description of software objects. We removed code snippets and screenshots from the description. We will look to utilize these code snippet and screenshot in the description to extend our models in our future work.

6. Conclusion

In this paper, we proposed four new deep learning-based methods TagCNN, TagRNN, TagHAN and TagRCNN for automated tag recommendation for large software information sites. The purpose of our study is to investigate whether there are deep learning methods that can achieve better performance than traditional approaches used for the tag recommendation task for software information sites. Our experimental results have shown that:

- For the tag recommendation task, two of our approaches TagCNN and TagRCNN, which are based on deep learning models CNN and RCNN respectively, achieved significant performance improvement over the three traditional approaches EnTagRec, TagMulRec, and FastTagRec. In contrast, the other two approaches TagRNN and TagHAN, which are based on deep learning models RNN and HAN respectively, were outperformed by these traditional approaches.

- Training of recommendation models can be done off-line and only needs to be done once. Although TagCNN and TagRCNN require longer training time than TagMulRec, the overhead is acceptable for real-world practice.

In summary, based on the results of this study, we believe the use of deep learning can help to find a better approach for some typical Software Engineering problems. Several open challenges remain. Software artifacts and relationships must be processed and represented in a vectorized form suitable for the chosen deep learning algorithm. For large data sets, the deep learning algorithm needs to be scaled across multiple machines and training may be a very time and compute intensive step. Even if this training is done off-line, if retraining is needed often, the scalability of the approach may be in question. As discussed above, we need to apply our new deep learning-based models to further software information sites and extend it to process rich content including code, design models and images, to more precisely determine its performance relative to more traditional data analytical approaches. Finally, we need to develop other deep learning-based models for other data intensive software engineering tasks, such as link recommendation, effort estimate, vulnerability detection and so on, to

determine how these perform in practice.

7. Acknowledgments

The authors would like to acknowledge the support provided by the grands of the National Natural Science Foundation of China (61572374, U1636220,61472423, 61602258), Open Fund of Key Laboratory of Network Assessment Technology from CAS, the Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012).

References

[1] T. D. Nguyen, A. T. Nguyen, H. D. Phan, T. N. Nguyen, Exploring api embedding for api usages and applications, in: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, 2017, pp. 438–449.

[2] V. J. Hellendoorn, P. Devanbu, Are deep neural networks the best choice for modeling source code?, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, 2017, pp. 763–773.

[3] M. White, M. Tufano, C. Vendome, D. Poshyvanyk, Deep learning code fragments for code clone detection, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016, pp. 87–98.

[4] J. Guo, J. Cheng, J. Cleland-Huang, Semantically enhanced software traceability using deep learning techniques, in: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, 2017, pp. 3–14.

[5] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, S. Li, Predicting semantically linkable knowledge in developer online forums via convolutional neural network, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016, pp. 51–62.

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[7] A.-r. Mohamed, G. E. Dahl, G. Hinton, Acoustic modeling using deep belief networks, IEEE Transactions on Audio, Speech, and Language Processing 20 (1) (2012) 14–22.

[8] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: Acoustics, speech and signal processing (icassp), 2013 ieee international conference on, IEEE, 2013, pp. 6645–6649.

[9] T. N. Sainath, O. Vinyals, A. Senior, H. Sak, Convolutional, long short-term memory, fully connected deep neural networks, in: Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, IEEE, 2015, pp. 4580–4584.

[10] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint arXiv:1412.3555.

[11] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434.

[12] W. Fu, T. Menzies, Easy over hard: a case study on deep learning, in: Joint Meeting on Foundations of Software Engineering, ACM, 2017, pp. 49–60.

[13] F. P. Brooks, No silver bullet, IEEE Computer 20 (4) (1987) 10–19.

[14] X. Xia, D. Lo, X. Wang, B. Zhou, Tag recommendation in software information sites, in: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, 2013, pp. 287–296.

[15] S. Wang, D. Lo, B. Vasilescu, A. Serebrenik, Entagrec: An enhanced tag recommendation system for software information sites., in: ICSME, 2014, pp. 291–300.

[16] J. M. Al-Kofahi, A. Tamrawi, T. T. Nguyen, H. A. Nguyen, T. N. Nguyen, Fuzzy set approach for automatic tagging in evolving software, in: Software Maintenance (ICSM), 2010 IEEE International Conference on, IEEE, 2010, pp. 1–10.

[17] M.-A. Storey, C. Treude, A. van Deursen, L.-T. Cheng, The impact of social media on software engineering practices and tools, in: Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, 2010, pp. 359–364.

[18] A. Begel, R. DeLine, T. Zimmermann, Social media for software engineering, in: Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, 2010, pp. 33–38.

[19] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, S. R. Klemmer, Two studies of opportunistic programming: interleaving web foraging, learning, and writing code, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2009, pp. 1589–1598.

[20] L. Guerrouj, S. Azad, P. C. Rigby, The influence of app churn on app success and stackoverflow discussions, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, 2015, pp. 321–330.

[21] C. Treude, M.-A. Storey, Work item tagging: Communicating concerns in collaborative software development, IEEE Transactions on Software Engineering 38 (1) (2012) 19–34.

[22] N. Kalchbrenner, E. Grefenstette, P. Blunsom, A convolutional neural network for modelling sentences, arXiv preprint arXiv:1404.2188.

[23] P. Liu, X. Qiu, X. Huang, Recurrent neural network for text classification with multi-task learning, arXiv preprint arXiv:1605.05101.

[24] Z. Yang, D. Yang, C. Dyer, X. He, A. J. Smola, E. H. Hovy, Hierarchical attention networks for document classification., in: HLT-NAACL, 2016, pp. 1480–1489.

[25] S. Lai, L. Xu, K. Liu, J. Zhao, Recurrent convolutional neural networks for text classification., in: AAAI, Vol. 333, 2015, pp. 2267–2273.

[26] P. Zhou, J. Liu, Z. Yang, G. Zhou, Scalable tag recommendation for software information sites, in: The 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2017.

[27] J. Liu, P. Zhou, Z. Yang, X. Liu, J. Grundy, Fasttagrec: fast tag recommendation for software information sites, Automated Software Engineering.

[28] B. Sigurbjörnsson, R. Van Zwol, Flickr tag recommendation based on collective knowledge, in: Proceedings of the 17th international conference on World Wide Web, ACM, 2008, pp. 327–336.

[29] S. Rendle, L. Schmidt-Thieme, Pairwise interaction tensor factorization for personalized tag recommendation, in: Proceedings of the third ACM international conference on Web search and data mining, ACM, 2010, pp. 81–90.

[30] D. Yin, Z. Xue, L. Hong, B. D. Davison, A probabilistic model for personalized tag prediction, in: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2010, pp. 959–968.

[31] Q. Wang, L. Ruan, Z. Zhang, L. Si, Learning compact hashing codes for efficient tag completion and prediction, in: Proceedings of the 22nd ACM international conference on Information & Knowledge Management, ACM, 2013, pp. 1789–1794.

[32] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, G. Stumme, Tag recommendations in folksonomies, in: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2007, pp. 506–514.

[33] E. F. Martins, F. M. Belm, J. M. Almeida, M. A. Gonalves, On cold start for associative tag recommendation, Journal of the Association for Information Science and Technology 67 (1) (2016) 83105.

[34] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[35] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[36] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, S. Khudanpur, Recurrent neural network based language model., in: Interspeech, Vol. 2, 2010, p. 3.

[37] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.

[38] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification (2016) 427–431.

[39] J. Pennington, R. Socher, C. Manning, Glove: Global vectors for word representation, in: Conference on Empirical Methods in Natural Language Processing, 2014, pp. 1532–1543.

[40] P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations, in: Proceedings of the 10th ACM Conference on Recommender Systems, ACM, 2016, pp. 191–198.

[41] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al., Wide & deep learning for recommender systems, in: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, ACM, 2016, pp. 7–10.

[42] S. Okura, Y. Tagami, S. Ono, A. Tajima, Embedding-based news recommendation for millions of users, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 1933–1942.

[43] X. He, X. Du, X. Wang, F. Tian, J. Tang, T.-S. Chua, Outer product-based neural collaborative filtering, arXiv preprint arXiv:1808.03912.

[44] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, H. Jing, Recurrent recommender networks, in: Proceedings of the tenth ACM international conference on web search and data mining, ACM, 2017, pp. 495–503.

[45] H. Ying, F. Zhuang, F. Zhang, Y. Liu, G. Xu, X. Xie, H. Xiong, J. Wu, Sequential recommender system based on hierarchical attention networks, in: the 27th International Joint Conference on Artificial Intelligence, 2018.

[46] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted boltzmann machines for collaborative filtering, in: Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 791–798.

[47] X. Cai, J. Han, L. Yang, Generative adversarial network based heterogeneous bibliographic network representation for personalized citation recommendation., in: AAAI, 2018.

[48] I. Munemasa, Y. Tomomatsu, K. Hayashi, T. Takagi, Deep reinforcement learning for recommender systems, in: International Conference on Information and Communications Technology, 2018, pp. 226–233.

[49] Z. Xu, J. Xuan, J. Liu, X. Cui, Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering, in: Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, Vol. 1, IEEE, 2016, pp. 370–381.

[50] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, N. Kapre, Software-specific named entity recognition in software engineering social content, in: Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, Vol. 1, IEEE, 2016, pp. 90–101.

[51] M. Mondal, C. K. Roy, K. A. Schneider, Spcp-miner: A tool for mining code clones that are important for refactoring or tracking, in: Software Analysis, Evo-

lution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, IEEE, 2015, pp. 484–488.

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

[53] Y. Zhang, B. Wallace, A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, arXiv preprint arXiv:1510.03820.

[54] M. A. Nielsen, Neural networks and deep learning, Vol. 25, Determination press USA, 2015.

[55] G. E. Dahl, T. N. Sainath, G. E. Hinton, Improving deep neural networks for lvcsr using rectified linear units and dropout, in: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 8609–8613.

[56] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473.

[57] D. Costarelli, R. Spigler, Approximation results for neural network operators activated by sigmoidal functions, Neural Networks 44 (8) (2013) 101–106.

[58] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect-proneness prediction framework, IEEE Transactions on Software Engineering 37 (3) (2011) 356–370.

[59] T. Zimmermann, N. Nagappan, Predicting defects using network analysis on dependency graphs, in: Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on, IEEE, 2008, pp. 531–540.

[60] X. Yang, D. Lo, X. Xia, J. Sun, Tlel: A two-layer ensemble learning approach for just-in-time defect prediction, Information & Software Technology (2017) 206–220.

[61] Y. Sun, C. Chen, Q. Wang, B. Boehm, Improving missing issue-commit link recovery using positive and unlabeled data, in: Ieee/acm International Conference on Automated Software Engineering, 2017, pp. 147–152.

[62] L. Mou, G. Li, L. Zhang, T. Wang, Z. Jin, Convolutional neural networks over tree structures for programming language processing (2014) 1287–1293.

[63] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, Computer Science.

[64] H. K. Dam, T. Tran, J. Grundy, A. Ghose, Deepsoft: A vision for a deep model of software, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2016, pp. 944–947.

[65] E. Zangerle, W. Gassler, G. Specht, Using tag recommendations to homogenize folksonomies in microblogging environments, in: International Conference on Social Informatics, Springer, 2011, pp. 113–126.

[66] H. Wang, B. Chen, W.-J. Li, Collaborative topic regression with social regularization for tag recommendation., in: IJCAI, 2013.

[67] D. Yang, Y. Xiao, H. Tong, J. Zhang, W. Wang, An integrated tag recommendation algorithm towards weibo user profiling, in: International Conference on Database Systems for Advanced Applications, Springer, 2015, pp. 353–373.

[68] D. Yang, Y. Xiao, Y. Song, J. Zhang, K. Zhang, W. Wang, Tag propagation based recommendation across diverse social media, in: Proceedings of the 23rd International Conference on World Wide Web, ACM, 2014, pp. 407–408.