Information Sciences 448–449 (2018) 36–52

Contents lists available at ScienceDirect

Information Sciences

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

Learning to answer programming questions with software documentation through social context embedding

Jing Li[a,*], Aixin Sun[a], Zhenchang Xing[b]

[a] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[b] College of Engineering and Computer Science, Australian National University, Australia

A R T I C L E I N F O

Article history:
Received 31 January 2017
Revised 3 January 2018
Accepted 9 March 2018
Available online 13 March 2018

Keywords:
Community-based question answering
Software documentation
Social context
Neural network

A B S T R A C T

Official software documentation provides a comprehensive overview of software usages, but not on specific programming tasks or use cases. Often there is a mismatch between the documentation and a question on a specific programming task because of different wordings. We observe from Stack Overflow that the best answers to programmers' questions often contain links to formal documentation. In this paper, we propose a novel *deep-learning-to-answer* framework, named `QDLinker`, for answering programming questions with software documentation. `QDLinker` learns from the large volume of discussions in community-based question answering site to bridge the semantic gap between programmers' questions and software documentation. Specifically, `QDLinker` learns question-documentation semantic representation from these question answering discussions with a four-layer neural network, and incorporates semantic and content features into a learning-to-rank schema. Our approach does not require manual feature engineering or external resources to infer the degree of relevance between a question and documentation. Through extensive experiments, results show that `QDLinker` effectively answers programming questions with direct links to software documentation. `QDLinker` significantly outperforms the baselines based on traditional retrieval models and Web search services dedicated for software documentation retrieval. The user study shows that `QDLinker` effectively bridges the semantic gap between the intent of a programming question and the content of software documentation.

1. Introduction

For most programming languages and software packages, there exist comprehensive language specifications, Application Programming Interface (API) documentation, and tutorials. Such official documentation[1] provides information about functionality, structure, and parameters, but not on specific issues or specific usage scenarios [31,42]. On the other hand, programmers often face very specific issues which are not explicitly stated in software documentation. For many such issues, software documentation does serve as a good reference for why the issues happen and how to address them. However, it is challenging to use a question as a keyword query to search for relevant software documents. This is because the software

* Corresponding author.
E-mail addresses: jli030@e.ntu.edu.sg (J. Li), axsun@ntu.edu.sg (A. Sun), zhenchang.xing@anu.edu.au (Z. Xing).

[1] The term 'software documentation' refers to the collection of documents consisting of language specification, API documentation, and official tutorial.

https://doi.org/10.1016/j.ins.2018.03.014
0020-0255/© 2018 Elsevier Inc. All rights reserved.

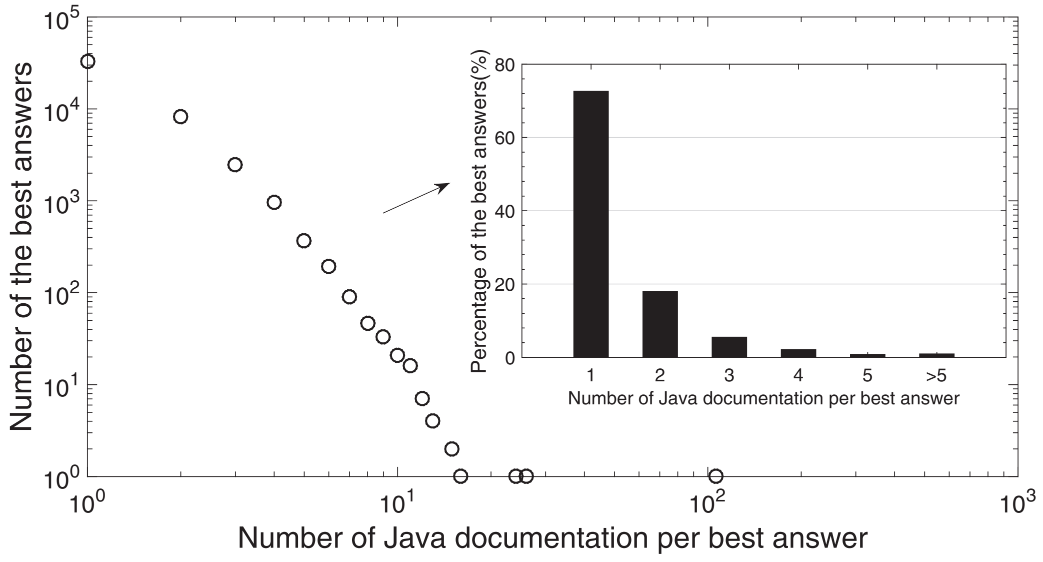


Fig. 1. Distribution of links to Java documentation per best answer, among 45,288 best answers from Stack Overflow. The absolute numbers are plotted in log scale, and the percentages are plotted in bar chart.

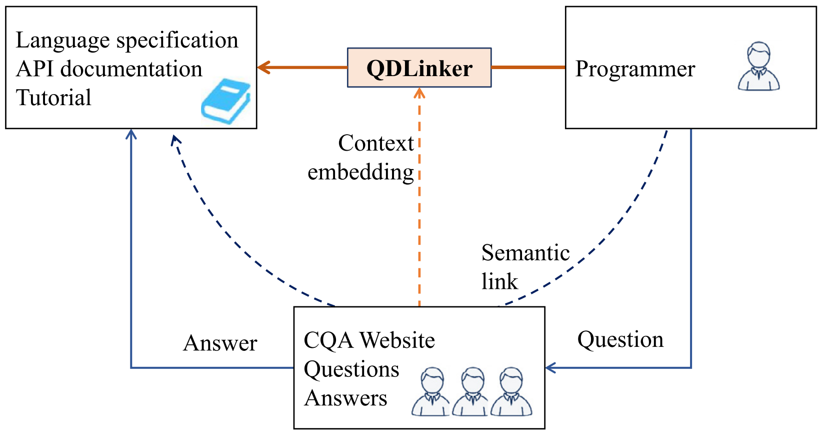


Fig. 2. Overview of `QDLinker`. It directly links programmer's question to formal documentation through embedding the semantic context in CQA. Before `QDLinker`, the semantic links between questions and software documentation are established through the CQA community.

documentation and question are often in different wordings; one is for generic reference and the other is from a specific usage scenario in practice.

With the emergence of Web 2.0 in modern software development, the behavior of developers is changed, in relation to how they search for crowd-generated knowledge to fulfill their needs [21,22,25]. The mismatch between the needs of documentation consumers and the knowledge provided, leads to the overwhelming discussions accumulated at various Community-based Question Answering (CQA) websites such as Quora[2] and Stack Overflow[3]. In these discussions, the community users often refer to software documentation when answering programming questions. From Stack Overflow, we collected 45,288 best answers each contains at least one link to Java official documentation. Fig. 1 plots the distribution of the number of links to Java documentation per best answer, which obeys a power-law distribution. It shows that 72.6% of best answers have exactly one link to Java documentation and fewer than 10% have more than three links. This distribution suggests that for many Java programming questions, there exists a Java official document as a good reference to address the question. The large volume of discussions also create the '*semantic link*' between programmers' questions and software documentation, through the community of programmers, illustrated in Fig. 2.

Posting questions and waiting for answers from other programmers may take much time. The immediate question is: *can we answer a programmer's question by providing a link to the most relevant software documentation?* In this research, we aim

[2] https://www.quora.com/.
[3] http://stackoverflow.com/.

to build an answering system where the questions are from programmers in natural language and the answers are the links to official documentation, illustrated in Fig. 2. This system will provide convenience not only for documentation consumers but also the companies that provide technical support.

However, understanding programming questions to build an effective answering system is not trivial. First of all, mapping question-answer pairs into a discriminative feature space is a critical step. A widely adopted approach is to encode question-answer pairs using various features, *e.g.,* lexical, linguistic, and syntactic features [36,37,53,61,67]. These hand-crafted features may heavily depend on external resources at the loss of generality. Besides, many existing knowledge bases are about lexical knowledge or about open domain facts. A typical example is WordNet [29], a lexical knowledge base for general English language, which may not be suitable to build answer systems for technical questions about programming. As shown by the analysis above, taking the advantage of neural networks to learn semantic representation of question-documentation pair seems to be more appropriate for our task. Neural networks have been proved to be powerful tools in many fields, such as machine transliteration [7], computer vision [50], electromagnetic theory [18], wire coating analysis [30], and bioinformatics [40]. Note that, our task cannot be addressed by search engines for source code [13,35]. Code search system cannot well answer queries in natural language, especially when the queries do not contain any code snippets or API-like terms.

In this paper, we propose a novel *deep-learning-to-answer* framework named `QDLinker`, to answer programming questions with software documentation through social context embedding. *Social context* of a link to software documentation refers to the surrounding words of the link, when community users use it to answer questions in CQA. `QDLinker` embeds social contexts in a latent space, and uses a four-layer Deep Neural Network (DNN) to learn semantic representations of question-documentation pairs. The learned semantic representations and simple content features are then passed to a learning-to-rank schema to train a ranker. Compared to prior work on software text retrieval [67], our approach does not require manual feature engineering or hand-coded resources beyond the pre-trained word vectors. The architecture we proposed is beneficial not only to learn a ranker in training phase, but also to automatic feature extraction for the newcoming query-documentation pairs in online phase. Moreover, our approach takes into account documentation content and social context simultaneously, for its effectiveness in bridging the semantic gap between programming questions and software documentation.

We conducted extensive experiments on Stack Overflow dataset to evaluate the effectiveness of `QDLinker`. Empirical results show that `QDLinker` outperforms three baseline methods which are based on traditional retrieval models. Through a user study with 25 natural language queries collected from test dataset, we show that `QDLinker` significantly outperforms a commercial search engine. In short, our empirical results show that `QDLinker` can effectively bridge the semantic gap between questions and software documentation. In this paper, we make the following contributions:

- We propose `QDLinker`, a novel framework for answering programming questions with software documentation through social context embedding. It leverages the content in official sites and social contexts in CQA to learn semantic representations of question-documentation pairs and answers programming questions in natural language.
- We conduct a large-scale automatic evaluation, to evaluate the performance of `QDLinker` against three baseline methods. The empirical evaluation reveals that our approach can effectively answer Java technical questions against the traditional retrieval models.
- We conduct a user study to compare the software documentation retrieval performances of `QDLinker` and Google search. The results show that `QDLinker` significantly outperforms Google search in the retrieval task.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 details our approach `QDLinker`. Section 4 presents the empirical evaluation. Section 5 presents the user study. Finally, we conclude the paper in Section 6.

2. Related work

Question Retrieval. Question retrieval has attracted much attention in recent years [4,8,10,16]. Different retrieval models have been employed in the task, including the Okapi model [16], the translation model [63], the language model [8], and the vector space model [16,17]. In addition, question category information has also been exploited for question retrieval [4]. Xue et al. [57] proposed a translation-based language model that combines the translation model and the language model for question retrieval. Yen et al. [61] developed a question classifier, which is trained to categorize the answer type of a given question and instructs the context-ranking model to re-rank the passages retrieved from the initial retrievers.

For the word mismatch problem among similar questions, existing solutions can be broadly grouped into three approaches to bridge the lexical gap. One approach is to use manual rules or templates. For example, Berger et al. [2] proposed a statistical lexicon correlation method to bridge the lexical chasm. The second approach is to use external knowledge databases such as Wikipedia and WordNet. The method by Zhou et al. [65] using semantic relations extracted from Wikipedia for question retrieval is an example. Burke et al. [3] proposed a model to rank frequently asked questions using combined similarities. The similarities are computed by conventional vector space models with semantic similarities based on WordNet. The third approach is to use deep representation. Zhou et al. [64,66] proposed a neural network architecture to learn the semantic representations of question-answer pairs. Nassif et al. [33] presented a neural-based model with stacked bidirectional Long Short-Term Memory (LSTM) and Multi-Layer Perceptron (MLP) for similar question retrieval. Different

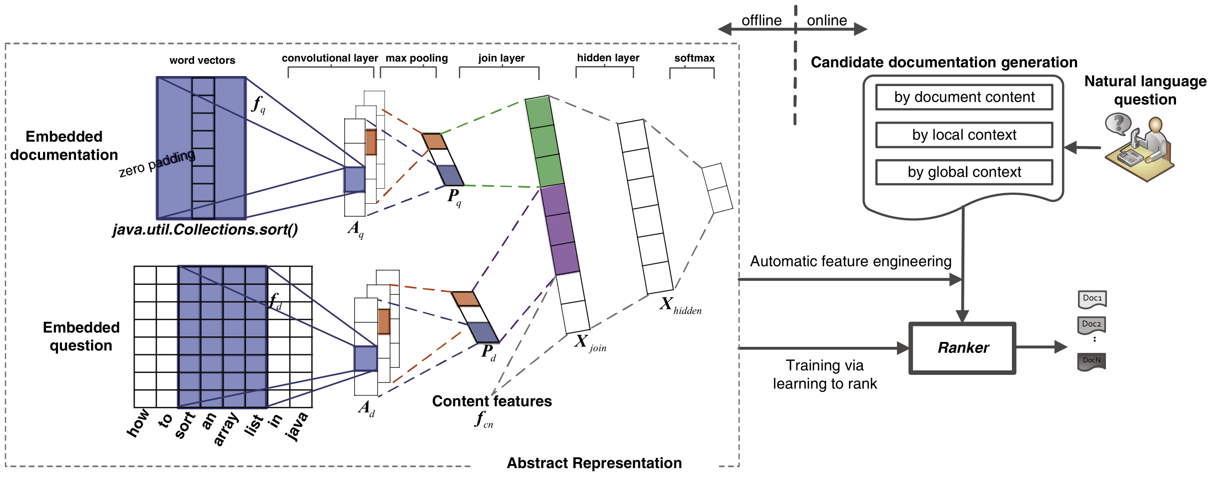


Fig. 3. The architecture of our proposed approach. Offline phase aims to learn abstract representation for question-documentation pair, and to learn a ranker with the abstract representations. Given a new question, online phase aims to rank its candidate documents.

from the prior studies, we aim to directly link questions to their relevant software documents, rather than retrieving similar questions.

Answer Selection. Given a thread containing a question and a list of answers, many studies aim to automatically rank the answers according to their relevance to the question [44,49,51]. Sun et al. [49] used dependency relations between the matched question terms and the answer target as additional evidences to rank passages. Sakai et al. [44] proposed an approach to build answer selection system involving multiple answer assessors and graded-relevance information retrieval metrics. Yao et al. [60] proposed a family of algorithms to jointly detect the high-quality questions, and to help users to identify a useful answer that would gain much positive feedback from site users. Hou et al. [15] and Nicosia et al. [34] proposed automatic answer selection algorithms based on the position of the answer in the thread and the context of an answer in a thread, respectively. Instead of selecting answers, in our task, we attempt to distill the software documentation in answers.

CQA Semantic Representation. Most relevant to our work is the study on semantic representation of CQA. In recent years, deep neural networks have been used to learn higher-level semantic representations of question-answer pairs [9,23,33,46,47]. Tan et al. [52] developed hybrid models to match passage answers to questions accommodating their complex semantic relations. Severyn et al. [46] proposed a convolutional neural network architecture, which maps queries and documents to their distributed vectors, for reranking pairs of short texts. Yan et al. [58] proposed a deep neural network to learn how a query and its context are related to candidate reply. Nassif et al. [33] presented a neural-based model with stacked bidirectional LSTMs and MLP to learn semantic relatedness between questions and answers. Singh et al. [48] proposed a system using semantic keyword search in combination with traditional text search techniques to find similar questions with answers for unanswered questions.

Social Context. Yang et al. [59] investigated user preference and social contexts in point of interest (POI) recommendation. They developed a deep neural architecture that jointly learns the embeddings of users and POIs to predict both user preference and POIs. Bagci et al. [1] built a graph model of location-based social networks (LBSNs) for personalized recommendations. This graph model took into account social contexts such as current social relations, personal preferences and current location of the user. Li et al. [26] proposed WisLinker framework to recommend web resources using social contexts. Rohani et al. [11,43] proposed an approach to address cold-start problem in academic social networks by incorporating social context features. Coined as an enhanced content-based algorithm using social networking (ECSN), the proposed algorithm considers the submitted ratings of faculty mates and friends besides user's own preferences. Rohani et al. [43] proposed an approach to solve cold-start problem by incorporating social context features.

In our proposed `QDLinker`, we learn the semantic representations of programming questions and the social contexts of software documentation. In our case, the social contexts, *i.e.*, how the APIs are used in different scenarios, cannot be obtained through the content of software documentation itself.

3. Deep learning to answer

In this section, we first give an overview of the proposed framework `QDLinker`, and then detail the core modules in `QDLinker` in Sections 3.2–3.4. The input to the framework, *i.e.*, the word embedding, is presented in Section 3.1.

As shown in Fig. 3, the `QDLinker` framework consists of three core modules: *candidate documentation generation*, *a four-layer neural network*, and *learning a ranker*. Given a programming question in natural language, *candidate documenta-*

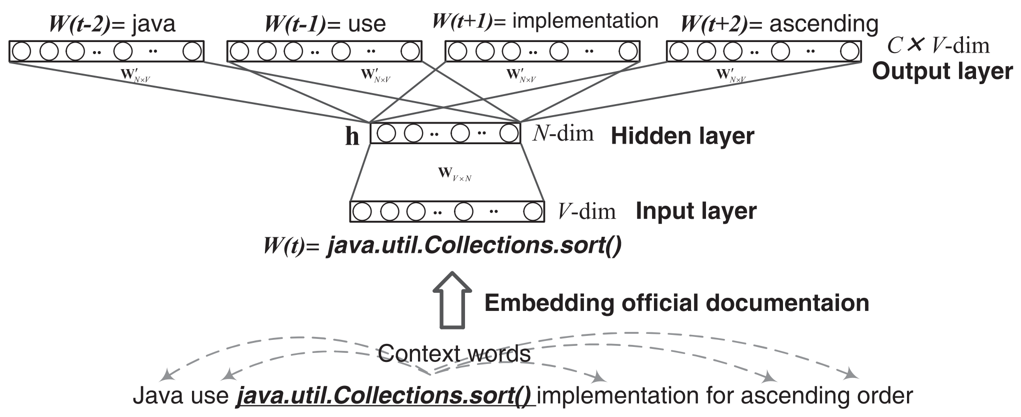


Fig. 4. Training word embedding example with the skip-gram model.

tion generation returns a small set of software documents which are considered relevant to the question. The DNN module learns the semantic representations of query-documentation pairs in a latent space, and generates latent features for the ranker module. The features by DNN are fully automatic without human intervention. Nevertheless, the network does allow handcrafted features to be inserted in the join layer, illustrated in Fig. 3. The learned features are then fed to a learning-to-rank schema to train a ranker, to pick up relatively relevant software documents among the candidates. Note that our approach cannot be formulated as an end-to-end problem (*i.e.*, directly minimizing a ranking cost function) because we need to automatically extract query-documentation representations for newcoming candidate documents in online phase. Thus, the DNN only serves as extracting the final representations of query-documentation pairs. Therefore, the architecture in Fig. 3 is beneficial not only to learn a ranker in training phase, but also to automatic feature extraction for newcoming query-documentation pairs in online phase.

3.1. Social context embedding

As shown in Fig. 3, `QDLinker` is built on pre-trained word vectors, or word embeddings. Traditionally, language models represent each word as a feature vector using one-hot representation, where a vector element is 1 if the word is observed and 0 otherwise [12]. Recently, neural language models have been proposed to generate low-dimensional, distributed embeddings of words [6,54]. These models take the advantage of word order in text documents and capture both syntactic and semantic relationships between words. Mikolov's continuous bag-of-words and skip-gram language models [27,28] are among the most widely used models.

Stack Overflow is a destination with rich source of information about API usages and bug descriptions. Thus, in our implementation, we use the crowd-generated content on Stack Overflow to learn embeddings of words and links to software documentation.

As shown in Fig. 4, a community user of Stack Overflow created a link to API documentation `java.util.Collections.sort()` in an answer. Fig. 4 illustrates the training procedure with the skip-gram model when it reaches the link. This sentence creates a context for the link `java.util.Collections.sort()` through the surrounding words. We build two vocabularies: one for English words, and the other for links to software documentation. In simple words, each link to software documentation is treated as an ordinary term in the word sequence, and a word vector is learned for each link that is mentioned on Stack Overflow. Note that, we learn word embedding for each link as a term, and the words in the anchor text of the link are not used in our training.

We define that w_t is the only word on the input layer. N is the hidden layer size. V is the vocabulary size. C is the number of words in the context. $\boldsymbol{x} \in \mathbb{R}^V$ is the one-hot encoded vector for w_t, which means only one out of V units will be 1 and all other units are 0. The output of hidden layer can be written as

$$\boldsymbol{h} = \boldsymbol{W}^T\boldsymbol{x} = \boldsymbol{V}_{w_t}^T \tag{1}$$

where $\boldsymbol{W} \in \mathbb{R}^{V\times N}$ is the input-hidden weight matrix. $\boldsymbol{V}_{w_t}$ is the vector representation of the input word w_t.

On the output layer, each output is computed using the hidden-output matrix:

$$p\left(w_{c,j} = w_{O,c}\middle|w_t\right) = \frac{\exp\left(u_{c,j}\right)}{\sum_{j'=1}^{V}\exp\left(u_{j'}\right)} \tag{2}$$

where w_t is the input word. $w_{c,\,j}$ is the j-th word on the c-th panel of the output layer. $w_{O,\,c}$ is the actual c-th word in the output context word. $u_{c,\,j}$ is the net input of the j-th unit on the c-th panel of the output layer,

$$u_{c,j} = \boldsymbol{V'}_{w_j}^{T} \cdot \boldsymbol{h},\ for\ c = 1, 2, \ldots, C \tag{3}$$

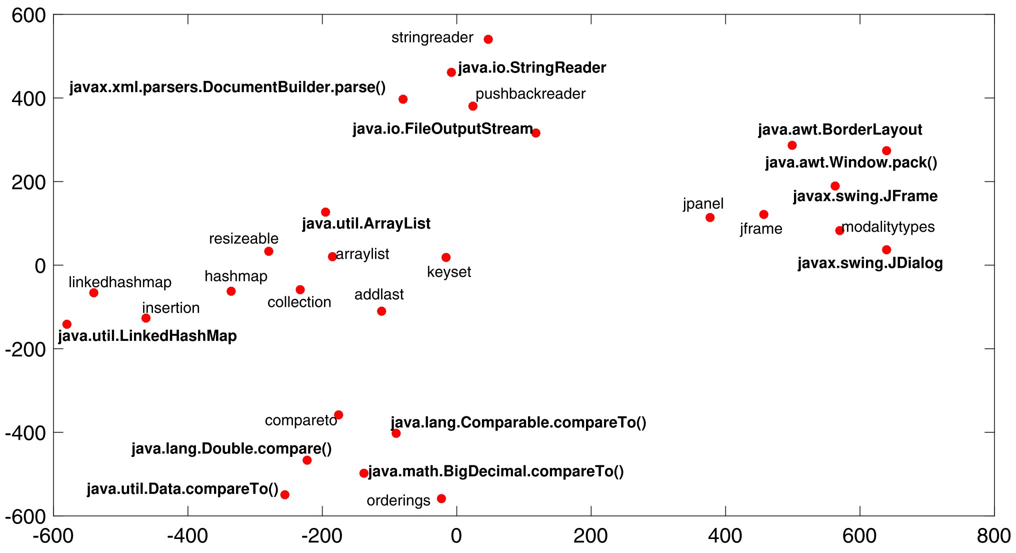


Fig. 5. A 2D projection of embedding natural language terms and API documentation using PCA (API documentation in bold font and natural language terms in non-bold font).

where $\boldsymbol{V'}^{T}_{w_j}$ is the output vector of the j-th word in the vocabulary, w_j and $\boldsymbol{V'}^{T}_{w_j}$ is taken from a column of the hidden-output weight matrix, $\boldsymbol{W'}$.

When training the skip-gram model to predict C context words, the loss function is written as

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,C}|w_t) \\ &= -\log \prod_{c=1}^{C} \frac{\exp\left(u_{c,j_c^*}\right)}{\sum_{j'=1}^{V} \exp\left(u_{j'}\right)} \end{aligned} \tag{4}$$

where j_c^* is the index of the actual c-th output context word in the vocabulary.

Fig. 5 illustrates a 2-D projection of vectors of natural language terms and API documentation using principal component analysis (PCA). In the embedding space, the vectors of terms and API documentation with the same intent have the shortest distance. For example, the term “arraylist” is close to API documentation `java.util.Arraylist`. API documentation `java.awt.Window.pack()` is close to `java.swing.JFrame`. Observe that there are four clusters in Fig. 5: java I/O, java.awt layout, java collection, and java compare to. For intra-cluster instances, they have similar programming function or purpose.

3.2. Candidate documentation generation

Given a programming question as a query, candidate generation selects a subset of software documents that are relevant to the question. We use three methods to select candidates.

Document Content. The content of software documentation reflects its relevance to a given query. In our implementation, we build a search engine for software documentation using Apache Lucene. Specifically, stopword removal and stemming are performed as preprocessing, and for each query, the search engine returns top 10 most relevant results based on the BM25 scoring function.

Local Context. Stack Overflow is a popular CQA site where developers ask questions and share knowledge about software development and maintenance. The discussions on Stack Overflow provide enriching context to mine usage scenarios of software documentation. When a software document appears in a discussion thread, its surround texts reflect its relevance to the question.

Definition 1 (Local Context)**.** If a software document is mentioned in a best answer, the texts of the question (title and body) and the best answer are regarded as the local context of the software document.

Local context is defined based on the consideration that the quality of best answer is better than other answers in the discussion thread, to avoid including too much noise. The body of the best answer is the immediate context where a software document is mentioned. On the other hand, question title and body often describe the programming issues and reflect the relevance between the problem and the software documentation mentioned in its best answer.

Note that, each mention of a software document has its own local context. If a software document is mentioned multiple times, multiple local contexts are extracted. We collect all local contexts of the mentioned software documents in our corpus. Given a query, we use Lucene to retrieve the most relevant local contexts, then pick the top 10 unique software documents as candidates.

Global Context. As aforementioned, a software document may be mentioned in multiple best answers and has multiple pieces of local contexts. For example, `java.util.ArrayList` was mentioned 906 times in our dataset.

Definition 2 (Global Context)**.** The global context of a software document is the collection of all its local contexts.

We build up a corpus through collecting global contexts for all software documents. Then we obtain the vector of a software document by social context embedding described in Section 3.1. Following [55], we use bag-of-words model to average out the vectors of the individual words in a query. Given a query, we retrieve top 10 software documents based on the cosine similarity between the average vector and software documentation vector.

3.3. Four-layer deep neural network

Deep neural network with multiple layers has demonstrated its effectiveness in capturing semantical and higher-level discriminative information from the input data [24]. As shown in Fig. 3, our DNN has four layers: *convolutional layer, join layer, hidden layer*, and *output layer*.

3.3.1. Convolutional layer

For a natural language query with many words in a sentence, prior studies [19,62] have shown that the simple bag-of-words model is unable to capture complex semantics of a sentence. Convolutional neural network can capture long-range dependencies and learn to correspond to the internal syntactic structure of sentences. Thus, we use one convolutional layer as the first layer in our approach.

Convolution Operation. Let $\boldsymbol{X}_d$ and $\boldsymbol{X}_q$ be the software documentation vector and query vector, respectively. Suppose that there are s words in the query and let $\boldsymbol{X}_q^i \in \mathbb{R}^k$ be the i-th k-dimensional word vector corresponding to the i-th word in the query. More formally, the convolution operation $*$ between two vectors $\boldsymbol{X}_q \in \mathbb{R}^{sk}$ and $\boldsymbol{f}_q \in \mathbb{R}^{mk}$ (called a filter of size m) results in a vector $\boldsymbol{c}_q \in \mathbb{R}^{s-m+1}$ where each component is as follows:

$$c_q^j = (\boldsymbol{X}_q * \boldsymbol{f}_q)_j = \boldsymbol{f}_q^T \cdot \boldsymbol{X}_q^{[j:j+m-1]} + b_{qc} \tag{5}$$

where $j = 1, \ldots, s-m+1$ and $\boldsymbol{X}_q^{[j:j+m-1]}$ represents the concatenation of word vectors $\boldsymbol{X}_q^j, \boldsymbol{X}_q^{j+1}, \ldots, \boldsymbol{X}_q^{j+m-1}$, $b_{qc} \in \mathbb{R}$ is a bias term. Thus, this filter $\boldsymbol{f}_q$ is applied to each possible window of words in the query to produce a feature map:

$$\boldsymbol{c}_q = [c_q^1, c_q^2, \ldots, c_q^{s-m+1}] \tag{6}$$

where $\boldsymbol{c}_q \in \mathbb{R}^{s-m+1}$. Similarly, we can utilize filter $\boldsymbol{f}_d$ to produce documentation feature map $\boldsymbol{c}_d$.

So far we have described the convolution layer with a single filter. Our model applies a set of filters that work in parallel to generate multiple feature maps. Let n be the number of filters. Given filters $\boldsymbol{F}_q^{n\times mk}$ and $\boldsymbol{F}_d^{n\times mk}$, the convolution operations produce two feature maps $\boldsymbol{C}_q \in \mathbb{R}^{n\times(s-m+1)}$ and $\boldsymbol{C}_d \in \mathbb{R}^{n\times(s-m+1)}$, respectively.

Activation Function. To allow the neural network to learn non-linear decision boundaries, each convolutional layer is followed by a non-linear activation function applied element-wise to the output of the convolution operations. Sigmoid, hyperbolic tangent *tanh*, and a rectified linear (*ReLU*) are among the most common choices for activation functions. In particular, it is reported that rectified linear unit has significant benefits over sigmoid and *tanh* functions [32]. Thus, in our implementation, we use *ReLU* as the activation function. The output of activation layer can be written as

$$\boldsymbol{A}_q = ReLU(\boldsymbol{C}_q) = max(0, \boldsymbol{C}_q) \tag{7}$$

$$\boldsymbol{A}_d = ReLU(\boldsymbol{C}_d) = max(0, \boldsymbol{C}_d) \tag{8}$$

where $\boldsymbol{A}_q \in \mathbb{R}^{n\times(s-m+1)}$ and $\boldsymbol{A}_d \in \mathbb{R}^{n\times(s-m+1)}$.

Pooling. The output from activation function is then passed to the pooling layer, whose goal is to aggregate the information and reduce the representation. As mentioned above, there are n filters. The pooling operation is applied on every filter. Taking the pooling of $\boldsymbol{A}_q \in \mathbb{R}^{n\times(s-m+1)}$ as an example, the output of pooling $\boldsymbol{P}_q \in \mathbb{R}^n$ can be written as

$$\boldsymbol{P}_q = \begin{bmatrix} pool\left(\boldsymbol{A}_q^1\right) \\ \cdots \\ pool\left(\boldsymbol{A}_q^n\right) \end{bmatrix} \tag{9}$$

The pooling operation maps the feature map to a single value, formally: $pool(\boldsymbol{A}_q^i) : \mathbb{R}^{1\times(s-m+1)} \to \boldsymbol{P}_q^i : \mathbb{R}$. There are a few common choices for the *pool*() operations: *average, max* and *L2-norm*. Average pooling was often used in the past but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice. In our approach, we use $1 - max$ pooling strategy, which extracts a scalar with the maximum value for each feature map.

3.3.2. Join layer

Inspired by Yu et al. [46,62], we also add simple content features $\boldsymbol{f}_{cn}$ to our model. $\boldsymbol{f}_{cn}$ contains two word overlap features: word overlap count, and word overlap count weighted by IDF (inverse document frequency). Note that both features do not require any linguistic annotation or pre-processing. The output of join layer $\boldsymbol{X}_{join} \in \mathbb{R}^{2n+2}$ can be expressed as follows:

$$\boldsymbol{X}_{join} = [\boldsymbol{P}_d; \boldsymbol{P}_q; \boldsymbol{f}_{cn}] \tag{10}$$

3.3.3. Hidden layer

DNNs could use the intermediate layers to build up multiple layers of abstraction. These multiple layers of abstraction seem likely to give deep networks a compelling advantage in learning to solve complex pattern recognition problems [45]. In our architecture, the hidden layer is a fully-connected layer with parameters $\boldsymbol{W}_h$ and $\boldsymbol{b}$. The output of hidden layer can be represented as

$$\boldsymbol{X}_{hidden} = ReLU(\boldsymbol{W}_h \cdot \boldsymbol{X}_{join} + \boldsymbol{b}_h) \tag{11}$$

3.3.4. Output layer

The output of hidden layer $\boldsymbol{X}_{hidden}$ is passed to a fully connected softmax layer. It computes the probability distribution over the class labels:

$$p(y = j|\boldsymbol{X}_{hidden}; \boldsymbol{W}_s, \boldsymbol{b}_s) = softmax_j(\boldsymbol{W}_s \cdot \boldsymbol{X}_{hidden} + \boldsymbol{b}_s) \tag{12}$$

where $\boldsymbol{W}_s$ and $\boldsymbol{b}_s$ are the weight vector and the bias of softmax classifier, respectively. Our model is trained to minimize the cross-entropy cost function:

$$\mathcal{L} = -log\prod_{i=1}^{N} p(y_i|\boldsymbol{X}_q^i, \boldsymbol{X}_d^i) + \lambda\|\theta\|_2^2 \tag{13}$$

where θ contains all parameters and we use $L2 - norm$ regularization.

$$\theta = \{\boldsymbol{F}_q, \boldsymbol{b}_{qc}, \boldsymbol{F}_d, \boldsymbol{b}_{dc}, \boldsymbol{W}_h, \boldsymbol{b}_h, \boldsymbol{W}_s, \boldsymbol{b}_s\} \tag{14}$$

In our problem setting, for a given question-documentation pair as an input instance, softmax layer outputs probabilities for two classification labels: positive and negative. Fig. 6 shows an example of question-documentation pair extracted from discussions on Stack Overflow. Together with the question, each link to documentation mentioned in the question's best answer forms a positive question-documentation pair instance.

For training the DNN, we use the links from the best answers of the training questions to form positive instances, and use randomly selected links to form negative instances. We use backpropagation algorithm to compute the gradients and use Adam update rule [20] to update the parameters of the network. To mitigate the overfitting issue, we augment the cost function with $L2 - norm$ regularization for the parameters of the network.

3.4. Learning a ranker

In particular, $\boldsymbol{X}_{hidden}$ can be thought of as a final abstract representation of a query-documentation pair, obtained by a series of transformations from the input layer through a series of layers. In our approach, we consider $\boldsymbol{X}_{hidden}$ as features to feed to a learning-to-rank schema. The learning-to-rank schema can leverage multiple features for ranking and can automatically learn the optimal way of combining these features.

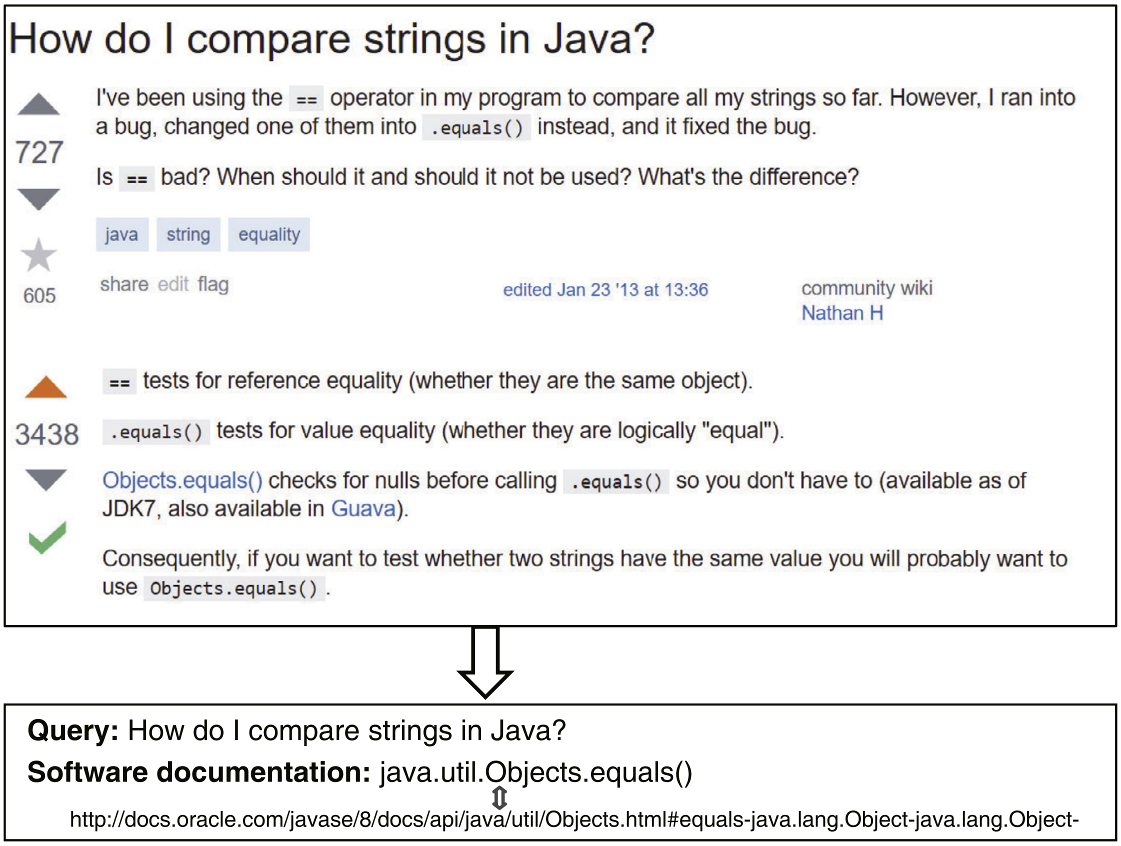
Our goal is to build a ranking model which facilitates each query q and its candidate list $D = \{d_1, d_2, \ldots, d_n\}$ to generate the optimal ranking. More formally, the task is to learn a scoring function $F(q, d)$:

$$F(q, d) = \sum_{k=1}^{K} \omega_i \cdot \phi_i(q, d) \tag{15}$$

where each feature $\phi_i(q, d) \in \boldsymbol{X}_{hidden}$ measures a specific relationship between the query and a candidate software document. ω_i is the weight of the i-th feature (among the total K features), and is learned during the training process. In our task, the optimization procedure of learning-to-rank tries to find the scoring function that ranks the most relevant software document to the query at the top among all candidates. We train the ranking model using LambdaMART [56], a boosted tree version of LambdaRank [38], that won the Yahoo! Learning to Rank Challenge.

3.5. Summary and complexity analysis

To summarize, we present a flow diagram of `QDLinker` in Fig. 7 and the pseudocode in Algorithms 1 and 2. Observe from Fig. 7, our framework contains both offline phase and online phase. The offline phase first learns abstract representation of question-documentation pair and then learns a ranker based on the positive and negative instances. Here an instance is a question-documentation pair. The offline phase is listed in Algorithm 1. Given a new question, the online phase first



How do I compare strings in Java?

I've been using the `==` operator in my program to compare all my strings so far. However, I ran into a bug, changed one of them into `.equals()` instead, and it fixed the bug.

Is `==` bad? When should it and should it not be used? What's the difference?

java string equality

share edit flag edited Jan 23 '13 at 13:36 community wiki Nathan H

`==` tests for reference equality (whether they are the same object).

`.equals()` tests for value equality (whether they are logically "equal").

Objects.equals() checks for nulls before calling `.equals()` so you don't have to (available as of JDK7, also available in Guava).

Consequently, if you want to test whether two strings have the same value you will probably want to use `Objects.equals()`.

Query: How do I compare strings in Java?

Software documentation: java.util.Objects.equals()

http://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#equals-java.lang.Object-java.lang.Object-

Fig. 6. An example of extracting query-documentation pair from discussion thread on Stack Overflow.

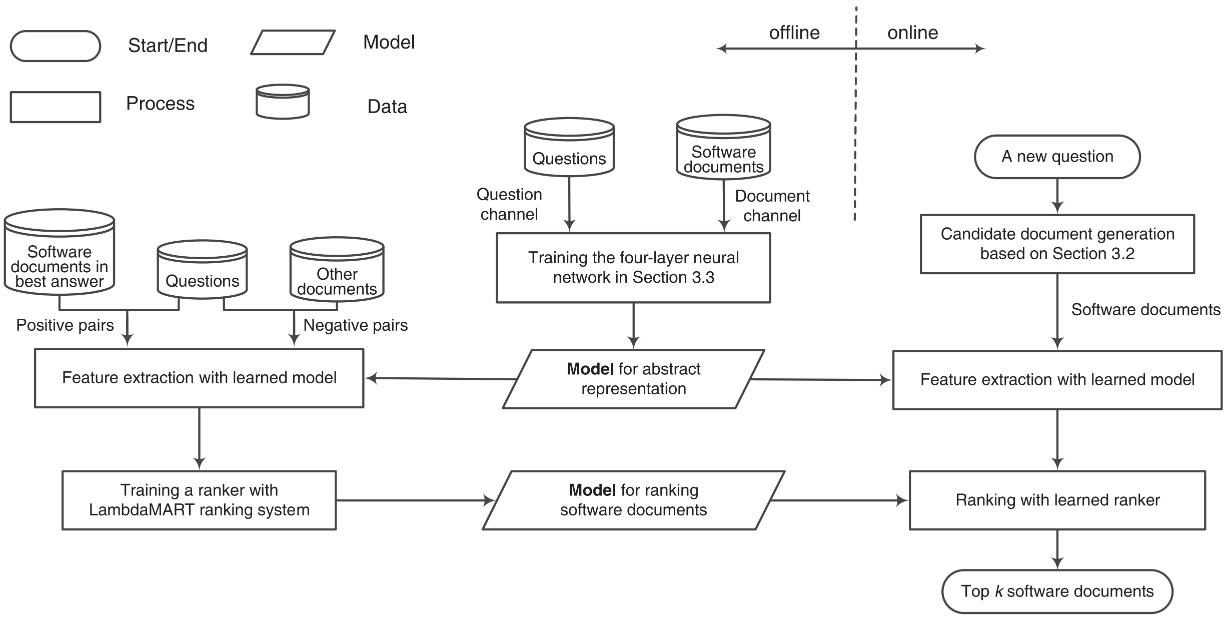

Start/End
Model
Process
Data
offline
online
Questions
Software documents
Question channel
Document channel
A new question
Candidate document generation based on Section 3.2
Software documents in best answer
Questions
Other documents
Training the four-layer neural network in Section 3.3
Software documents
Positive pairs
Negative pairs
Feature extraction with learned model
Model for abstract representation
Feature extraction with learned model
Training a ranker with LambdaMART ranking system
Model for ranking software documents
Ranking with learned ranker
Top k software documents

Fig. 7. The flow diagram of the proposed `QDLinker`. The middle part is to learn a model for extracting abstract representation for question-documentation pair. The left part aims to learn a ranker. The right part is to answer a new question with the learned models.

Algorithm 1: Pseudocode for offline phase of the proposed QDLinker.

```
   Input: N training instances of question-document pairs
   Output: Feature vector function $V_\theta(q,d)$ and ranking function $F$
   // Phase 1: Learning abstract representation
1  Initialize all parameters $\theta$;
2  foreach epoch in $epoch_{max}$ do                          // iterate through epoches
3      Sample a minin-batch from N pairs;
4      Clear gradients $d\theta \leftarrow 0$;
5      Computing $\mathcal{L}$ based on Equation (13);
6      Update $\theta \leftarrow \theta - \frac{\partial \mathcal{L}}{\partial \theta} \cdot lr$ ;
7      return $V_\theta(q,d)$ ;                 // return feature vector function for q-d pair
   // Phase 2: Learning a ranker
8  Set number of trees M, number of leaves per tree L;
9  foreach m in M do                                          // iterate trees
10     foreach n in N do                                      // iterate through training pairs
11         Calculating feature vector for current n via $V_\theta(q,d)$ in Phase 1;
12         Calculating the λ-gradients for each q-d pair;     // more detials [56]
13         Calculating the second-order derivative using the λ-gradients;
14     Building a regreesion tree with L terminal nodes;
15     Update F function;
16     return F ;                                             // Final ranking function F
```

Algorithm 2: Pseudocode for online phase of the proposed QDLinker.

```
   Input: A new question $q_{new}$
   Output: top-k software documents
1  Retrieving candidate documents for $q_{new}$ based on Section 3.2;
2  Extracting features for these question-document pairs using $V_\theta(q,d)$ in Phase 1 of Algorithm 1;
3  Ranking these candidate documents using F in Phase 2 of Algorithm 1;
4  return top-k software documents ;                         // Answers to the issued question
```

generates the k candidate software documents based on Section 3.2. Then these documents are ranked by the learned models in offline phase. Accordingly, the online phase is listed in Algorithm 2.

We first detail the parameter size of our framework, then deduce the time and space complexity. Eqs. (14) and (15) show all parameters that QDLinker would learn. Now we consider one query-document pair as shown in Fig. 3. For convolutional layer, QDLinker has $2n \times mk + 2n$ parameters, where n is the number of filters and m is the window size of each filter, k is the dimension of word embedding, and number 2 indicates query channel and document channel. For join layer, there is no parameter. For hidden layer, there are $(2n+2) \times h + h$ parameters, where h is the number of neruons in hidden layer. For output layer, there are $h \times 2 + 2$ parameters. For ranker layer, there are h weights because QDLinker uses the $\boldsymbol{X}_{hidden}$ as the final abstract representation.

Given a new question, we assume that QDLinker generates κ candidate software documents. Now considering a question-documentation pair, the total time and space complexity of convolutional layer are both $O(\alpha \cdot m^2 \cdot \beta)$ [5,14], where α and β are the number of input nodes and the number of output nodes respectively, m is the window size of each filter. Hidden layer and output layer are fully connected layers. The time and space complexity of this linear projection are both $O(\alpha\beta)$. For the ranker, the time complexity is $O(\kappa^2)$ [56]. Thus, the time complexity of QDLinker is $O(\kappa\alpha m^2\beta + \kappa^2)$, and the space complexity is $O(\alpha m^2\beta)$.

4. Empirical evaluation

We now evaluate the effectiveness of QDLinker by measuring its accuracy on linking questions on Stack Overflow to software documentation. Our evaluation assumes that the software documentation mentioned in a question's best answer is the most relevant to the question.

4.1. Experimental setting

Data Collection. In our evaluation, we focus on Java software documentation which consists of Java Standard Edition API documentation, Java tutorials, and language specifications. Usually, a programming question is expressed in natural language

Table 1
Summary of the number of threads used in each task.

Data	#Discussion threads
Word embeddings	24,217
Train	10,649
Development	1,000
Test	1,693

thus it is similar to the discussions on Stack Overflow. We therefore use the data collected from Stack Overflow in our experiments.

We extract discussion threads from the datadump archive[4] that satisfy the following criteria: (i) The score of question is greater than 0. This condition guarantees that at least one developer has voted the question to be a 'useful question'. (ii) The question has an answer which is accepted as the best answer, and the score of the best answer is greater than 0, and (iii) The best answer must contain at least one link to the above listed Java documentation. Based on the above criteria, we collect 30,272 discussion threads from the data dump released on August 2015. We randomly select 24,217 discussion threads (account for 80%) as training data and the remaining 6055 threads (account for 20%) as test data.

Model Training. We learn word embeddings from the training data, *i.e.,* the 24,217 discussion threads. For each discussion thread, we extract text from question title, question body, and all answers whose scores are greater than 0. We use the skip-gram model implemented in word2vec[5]. The context window size is set to 10 and the minimal word frequency is 5. Recall that each link to a software document is also treated as a word (or term, see Fig. 4). Based on this condition, we have 1520 distinct links to Java documentation in the training data.

Next is to train the DNN and the ranker. Note that some discussion threads cannot be used to extract query-documentation pairs for training DNN because the links to software documentation in best answers are filtered out when the minimal word frequency is set to 5. Finally, we have 10,649 discussion threads for training DNN and the ranker, 1000 discussion threads used for development set, and 1693 discussion threads used for test. Table 1 summarizes the dataset in our experiments.

We empirically set the hyperparameters based on the development set. The number of filters in convolutional layer is 64, and the size of filter is set to 2. The size of hidden layer is set to 64. The dimensionality of pre-trained word vectors is 200. $L2-norm$ term is set to $1e-5$ and the learning rate is $1e-3$.

Performance Measures. We use the following five performance metrics in our evaluation:

- Precision at k, $P@k = \frac{|D_k \cap D_g|}{k}$, is the fraction of relevant documentation links to the query question among the top k ranked results. D_k denotes the set of top-k ranked links to software documentation and D_g is the set of ground-truth links (*i.e.,* links to software documentation in the question's best answer).
- Recall at k, $R@k = \frac{|D_k \cap D_g|}{|D_g|}$ is the fraction of ground-truth links in the top-k results.
- Hit rate at k, denoted by $HR@k$, is 1 if $|D_k \cap D_g| > 0$ and 0 otherwise.
- Mean average precision, MAP, is the mean of average precision (AP) over a set of test queries.
- Mean reciprocal rank, $MRR(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{rank_j}$, is the average reciprocal rank of the results over a set of test queries Q. In the equation, $rank_j$ denotes the rank position of the first relevant document for the $j-th$ test query.

Baseline Methods. In order to validate the effectiveness of the proposed method, we evaluate the following three baseline methods in our experiments.

- OfficialCn: This is the baseline model which selects candidate software documentation by content (see Section 3.2). BM25 [41] scoring function is used to rank the candidates.
- LocalCx: This baseline ranks candidates based on local context (see Section 3.2). This model indexes the local contexts in discussion threads and retrieves candidates using BM25 scoring function.
- GlobalCx: This baseline ranks candidates based on global context (see Section 3.2). This model represents natural language words and software documentation as vectors in shared embedding space [27].

4.2. Performance comparison

Tables 2 and3 report the linking performance by different methods on all evaluation metrics. From the results, `QDLinker` significantly outperforms all baselines on all metrics. The improvement is statistically significant based on t-test with $p < 0.05$. Observe that $P@10$ is very small for all methods including `QDLinker`. On Stack Overflow, more than 70% of the best answers contain only one link to software documentation (reported in Section 1, Fig. 1). As the result, $|D_{10} \cap D_g| = 1$ in most cases. Thus, the ideal value of $P@10$ is slightly above 0.1, `QDLinker` achieves a very good result of 0.0919 in this sense.

[4] https://archive.org/details/stackexchange.

[5] https://code.google.com/archive/p/word2vec/.

Table 2
Performance (*P@k*, *R@k*, *MAP* and *MRR*) for different methods. The best performance is highlighted in bold face. † indicates that the differences between the result of `QDLinker` and other models are statistically significant with $p < 0.05$ under *t*-test.

Method	P@1	P@2	P@5	P@10	R@1	R@2	R@5	R@10	MAP	MRR
OfficialCn	0.1347	0.1087	0.0749	0.0511	0.1147	0.1797	0.3053	0.4108	0.2234	0.2267
LocalCx	0.1630	0.1337	0.0932	0.0657	0.1401	0.2253	0.3876	0.5361	0.2896	0.2956
GlobalCx	0.1536	0.1234	0.0875	0.0628	0.1300	0.2002	0.3548	0.5005	0.2564	0.2614
`QDLinker`	**0.1875†**	**0.1576†**	**0.1272†**	**0.0919†**	**0.1708†**	**0.2734†**	**0.5012†**	**0.6847†**	**0.3461†**	**0.3584†**

Table 3
Performance(*HR@k*) for different methods.

Method	HR@1	HR@2	HR@5	HR@10
OfficialCn	0.1347	0.2069	0.3491	0.4663
LocalCx	0.1630	0.2631	0.4341	0.5914
GlobalCx	0.1536	0.2383	0.4109	0.5640
`QDLinker`	**0.1875†**	**0.3057†**	**0.5828†**	**0.8128†**

Table 4
Results on additional content features.

Content features	MAP	MRR
Without $\boldsymbol{f_{cn}}$ features	0.3054	0.3102
With $\boldsymbol{f_{cn}}$ features	0.3461(↑13.32%)	0.3584(↑15.53%)

On *R@k* measure, `QDLinker` significantly outperforms the other baselines for all k values ($k = 1, 2, 5, 10$). It is worth noting that `QDLinker` achieves the highest recall (0.6847) when $k = 10$. In terms of *MAP* measure, `QDLinker` outperforms the three baseline methods OfficialCn, LocalCx and GlobalCx by 34.98%, 19.51% and 54.93%, respectively. On *MRR* measure, `QDLinker` outperforms the three baselines by 37.11%, 21.24% and 58.09%, respectively. Similar observations hold on hit rate measure *HR@k*, reported in Table 3.

Observe from the results that the content-based method OfficialCn delivers the worst performance on all measures. This implies that content-based approach cannot bridge developers' intent and content of software documentation. This observation is consistent with our description in Section 1 that software documentation is prepared to give comprehensive coverage without targeting on specific problems, while the programming questions are encountered in specific programming tasks. Therefore, it is essential to utilize the social context available on Stack Overflow to bridge the semantic gap between programmers' questions and software documentation.

4.3. Impacts of factors on performance

4.3.1. Impact of content features

In our approach, $\boldsymbol{f_{cn}}$ consists of word overlap count and word overlap count weighted by IDF value. Table 4 shows the performance and improvement of considering additional content features $\boldsymbol{f_{cn}}$. More specifically, considering $\boldsymbol{f_{cn}}$ improves *MAP* by 13.32%, and *MRR* by 15.53%.

There are two aspects resulting in the improvement. First, as described above, input of our approach are the pre-trained word vectors. In fact, the word dictionary of our dataset may not cover all the words in English language. The overlap features can provide supplementary information in our approach. Additionally, one of the weaknesses of approaches relying on distributed word vectors is their inability to deal with numbers and proper nouns [46,62]. But when developers issue natural language queries, most of the questions are of type "what", "when", "who" that are looking for answers containing numbers or proper nouns. Thus, the model with $\boldsymbol{f_{cn}}$ features outperforms the one without $\boldsymbol{f_{cn}}$ features on *MAP* and *MRR*.

4.3.2. Dimensionality of word embedding

`QDLinker` takes in pre-trained word vectors in the input layer and feeds into the convolutional layer. We vary the dimensionality of word embedding and evaluate its impact on *MAP* and *MRR*. Fig. 8 reports the performance of `QDLinker` using word embedding in different dimensions (50, 100, 200, 300, 400, 500 and 600). The results indicate that the dimensionality of word embedding has very marginal impact on the performance. One possible reason is that the distributed word vectors varying different dimensions contain enough latent information for building a ranker in our dataset. Thus, when training a ranker, our approach is stable for different dimensions of word embedding.

4.3.3. Impact of layer sizes

As shown in Fig. 3, out architecture needs to set the number of neurons (*i.e.*, layer size) in convolutional layer and hidden layer. Fig. 9 shows the impact of setting different convolutional layer sizes and hidden layer sizes, on the test set. The comparison is based on 200 dimensions of word vectors.

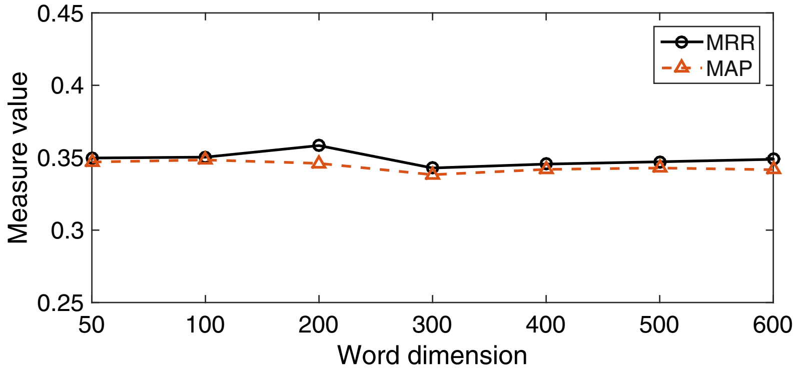


Fig. 8. Impact of dimensionality of word embedding.

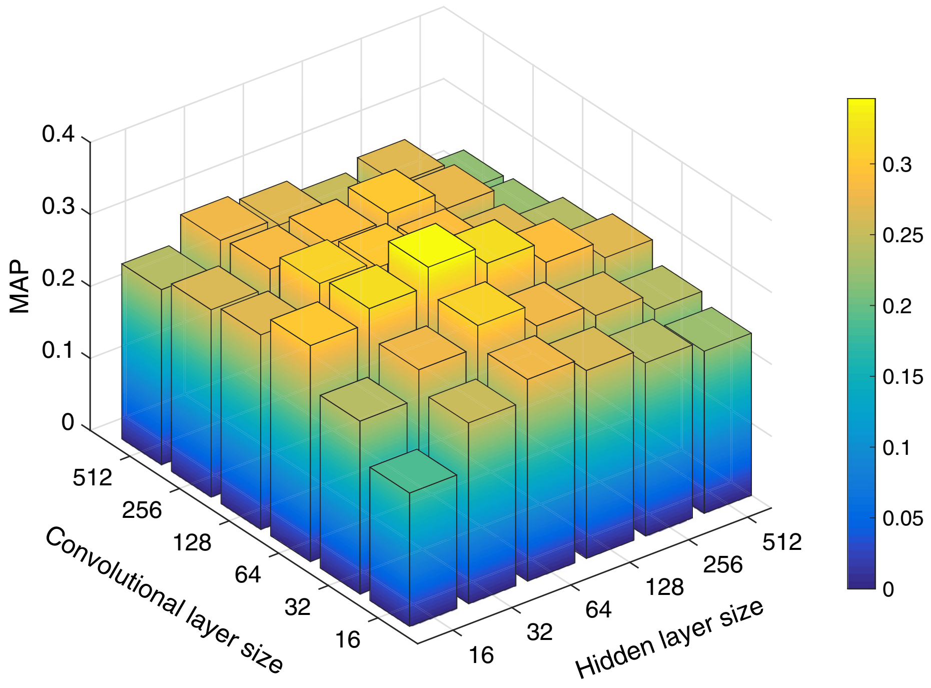


Fig. 9. Performance in MAP with different layer sizes.

We observe that the performance of ranker greatly depends on the combination of convolutional layer size and hidden layer size. In our dataset, we obtain the best performance when the convolutional layer size and hidden layer size are both 64.

When the combination layer sizes are small, the *MAP* is around 0.2 only, much lower than the best performance. Too few neurons in the convolutional and hidden layers will result in underfitting, as the neurons cannot capture enough signals to model complex data. However, larger combination layer sizes does not lead to better ranker performance either. In addition, a large number of neurons in the convolutional and hidden layers increase the training time.

In summary, dimensionality of word vectors has very marginal impact on the performance. However, the sizes of convolutional and hidden layers have significant impact on the performance of our model.

5. User study

To the best of our knowledge, there is no existing work on answering programming questions in natural language. Commercial search engines, *e.g.*, Google and Bing, are tools for daily use in software development. It naturally motivates us to compare the returned results with such search engines. If we can improve the performance of search results on the search engines, it will provide convenience not only for developers but also for the companies that provide documentation support.

In the previous set of experiments, we consider the software documentation mentioned in the best answers as the ground truth to questions. This assumption may ignore the other retrieved software documentation which is relevant to the question but is not mentioned in the best answers. That is, although limiting documentation mention in best answer is a good criterion to control the quality of ground truth, the criterion may exclude the relevant results from our evaluation. In this section, we perform a user study to manually evaluate performance of `QDLinker` against Web search services.

5.1. Evaluation setup

From the test dataset, we randomly select 25 discussion threads and query `QDLinker` using their questions. The 25 questions are listed in Table 5. For each question, we also use Google search engine to retrieve a list of software documentation. Because we focus on official documentation in this study, we restrict the retrieved results by Google by setting the "site" parameter in the search. For example, the second query in Table 5 is extended as"*XML string parsing in Java site:docs.oracle.com/javase*" using Google search engine in August, 2016. Note that, all the three types of Java documentation (language specification, API documentation, and tutorial) are under the same site: docs.oracle.com/javase.

To measure the performance of `QDLinker` and Google, we use three metrics [13,39]. *FR* is the rank of the first relevant result, as most users scan the results from top to bottom. The smaller the number of *FR*, the better the performance. The *P*@5 denotes the precision of the top 5 ranked results. Note that, the judgements of relevance are manually labeled by our annotators. Similarly, the *P*@10 is the precision of the top 10 ranked results.

We recruited two developers to manually annotate the two set of results from Google and `QDLinker`, respectively. Each link to software documentation in result list was marked relevant or irrelevant, indicating whether the developer considered this software documentation is relevant to the question. The annotation was done individually by the two developers and for inconsistent judgments, the two developers reached a consensus through discussion.

5.2. Evaluation results

Table 5 shows the performance comparison of Google search and `QDLinker`. In particular, the symbol "-" in the second column indicates that there is no relevant software documentation returned by Google search in the query. The last row shows the average performance on the three metrics.

Compared with Google search, `QDLinker` achieves better performance on *FR*, *P*@5 and *P*@10. In most cases (20 out of the 25 queries), `QDLinker` is able to recommend relevant software documentation at the first position in the result list. The differences between these two approaches in terms of the three metrics are statistically significant at $p < 0.05$. That is, `QDLinker` provides more relevant software documentation in top 10 results than Google search in our user study.

The last column shows web page titles of the top 3 ranked relevant documents by `QDLinker`, which consist of Java API documentation (marked by **A**_), Java language specifications (marked by **S**_) and Java tutorials (marked by **T**_). For example, in Queries 1 and 2, "***S**_Chapter 16. Definite Assignment*"[6], "***A**_javax.xml.parsers. DocumentBuilder. parse()*"[7], "***T**_Branching Statements*"[8] represent a document in language specification, API documentation, and tutorial, respectively. Compared with Google search, we make following observations:

- `QDLinker` can bridge the semantic gap between question and software documentation. For example, query 5 is "*reinitialise transient variable*", and there is no Java documentation which contains all the three keywords. Google search cannot return relevant results in this query. Likewise, the state-of-the-art API usage miner [13] cannot return any API sequences based on code corpus from Github. We manually check our training dataset and find that some community users have implemented the task using the class "`java.io.Serializable`" and method "`readObject`" on Stack Overflow. Thus, `QDLinker` can effectively answer this question because it takes into account the content and context of software documentation simultaneously.
- `QDLinker` can effectively answer complex and bug-like queries. For instance, query 17 "*raster format exception (Y+height)*" and query 19 "*java modifying a class directly, null reference*" are related to program exceptions. Poorer results were obtained from Google for such kind of queries. On the contrary, `QDLinker` provided high quality results for such kind of queries, and an example is the API documentation `java.awt.image.BufferedImage` for query 17.
- `QDLinker` can effectively answer programming questions which are in specific usage scenarios. For instance, query 18 "*rendering combo boxes in a JTable*" is about usage of *combo boxes* in the scenario of "*JTable*" and query 13 "*java executor with no ability to queue tasks*" is about of *Java executor* in the scenario of "*queue tasks*". The official software documentation does not serve as good reference for these queries in specific usage scenarios, while `QDLinker` can provide high quality software documentation for these queries. For example, `java.util.concurrent.ThreadPoolExecutor` is a high-quality API document for query 13.

[6] https://docs.oracle.com/javase/specs/jls/se8/html/jls-16.html.
[7] https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/DocumentBuilder.html#parse-java.io.File-.
[8] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html.

Table 5
Human evaluation for Google search and QDLinker (*BA*: the number of ground truth links in best answer. *FR*: the rank of the first relevant documentation. *P*@5 and *P*@10: precision of the first 5 and 10 results. "**A_**" indicates Java API documentatins. "**S_**" indicates Java language specification. "**T_**" indicates Java tutorial. The boldface documentation indicate the ground truth documentation in best answers. † indicates the differences between QDLinker and Google are significant with $p < 0.05$ under *t*-test).

Query	BA	Google Search			QDLinker			Top 3 relevant documents by QDLinker
		FR	*P*@5	*P*@10	*FR*	*P*@5	*P*@10	
1: java switch error when simplifying code	1	7	0	0.2	1	0.8	0.5	**S_Chapter 16. Definite Assignment; T**_Branching Statements; **S**_Chapter 14.11. The switch Statement
2: XML string parsing in Java	1	1	1	0.9	4	0.4	0.3	**A**_javax.xml.parsers.DocumentBuilder.parse(); **A**_javax.xml.bind.DatatypeConverter; **A**_javax.xml.transform.Transformer
3: PLAF can't change button color	2	1	0.4	0.5	1	0.4	0.3	**A**_javax.swing.UIManager; **T_How to Set the Look and Feel; T**_How to Use Color Choosers
4: match generics with mockito	1	1	0.4	0.5	1	0.6	0.7	**T_Type Erasure; T**_Erasure of Generic Types; **T**_Erasure of Generic Methods
5: reinitialise transient variable	1	–	0	0	1	0.6	0.7	**A_java.io.Serializable; S**_Chapter 8.3.2. Initialization of Fields; **S**_Chapter 14.14. The for Statement
6: write and read multiple byte[] in file	1	2	0.2	0.2	1	0.6	0.5	**A**_java.io.FileInputStream.read(); **A**_java.io.FileOutputStream.write(); **A_java.io.RandomAccessFile**
7: equality of boxed boolean	1	1	0.4	0.5	1	0.6	0.5	**S_Chapter 5.1.7. Boxing Conversion; A**_java.util.Arrays.equals(); **A**_java.lang.Object.equals()
8: resetting and copying two dimensional arrays	1	–	0	0	1	0.4	0.4	**A_java.lang.System.arraycopy(); T**_Arrays; **A**_java.util.Arrays.copyOf()
9: java standard on result of casting a double to an int	2	1	0.8	0. 6	1	0.8	0.7	**S_Chapter 5.1.3. Narrowing Primitive Conversion; A**_java.lang.Number; **A**_java.lang.Double
10: registering and using a custom java.net.URL protocol	5	3	0.2	0.1	1	0.6	0.8	**A_java.net.URLConnection; A**_java.net.URI; **A**_java.net.HttpURLConnection
11: why is the protected method not visible	1	1	0.2	0.1	3	0.4	0.5	**S**_Chapter 5.1.3. Narrowing Primitive Conversion; **S**_Chapter 5.1.2. Widening Primitive Conversion **T_Controlling Access to Members of a Class**;
12: java date formatting ParseException	1	1	0.8	0.6	1	0.6	0.7	**A_java.text.SimpleDateFormat; A**_java.text.DateFormat.parse(); **A**_java.text.DateFormat.format()
13: java executor with no ability to queue tasks	2	–	0	0	1	0.8	0.8	**A**_java.util.concurrent.Executors; **A_java.util.concurrent.ThreadPoolExecutor; A**_java.util.concurrent.ExecutorService
14: how to replace a jPanel based on user clicks	1	2	0.6	0.5	2	0.4	0.6	**T_How to Use CardLayout; T**_How to Use the Focus Subsystem; **T**_How to Write a Mouse Listener
15: ambiguous varargs method call compilation error	1	–	0	0	2	0.4	0.5	**S_Chapter 15.12.2. Compile-Time Step 2: Determine Method Signature; A**_java.lang.invoke.MethodHandle; **S**_Chapter 6.6.1. Determining Accessibility
16: why is there no generic type information at run rime	2	1	0.8	0.7	1	0.6	0.6	**T**_Generic Methods; **T**_Why Use Generics?; **T**_Erasure of Generic Types
17: raster format exception (Y+height)	1	4	0.2	0.3	1	0.4	0.4	**A_java.awt.image.BufferedImage; A**_javax.imageio.ImageIO; **T**_Lesson: Working with Images
18: rendering combo boxes in a JTable	2	1	0.6	0.6	1	0.8	0.7	**T**_How to Use Tables; **T**_How to Write an Item Listener; **T**_How to Use Combo Boxes
19: java modifying a class directly, null reference	1	–	0	0	1	0.6	0.5	**T_Anonymous Classes; S**_Chapter 12.4. Initialization of Classes and Interfaces; **A**_java.lang.NullPointerException
20: windows azure date format to java date	1	–	0	0	1	0.8	0.7	**A_java.text.SimpleDateFormat; A**_java.time.format.DateTimeFormatter; **A**_java.text.DateFormat
21: reading arraylist from a .txt file	1	5	0	0.1	2	0.6	0.6	**A**_java.nio.file.Files.readAllLines(); **A**_java.io.FileInputStream; **A_java.util.Scanner.nextLine()**;
22: close connection and statement finally	1	1	0.8	0.4	1	0.8	0.6	**T_The try-with-resources Statement; A**_java.sql.Statement; **T**_Using Transactions
23: when are java temporary files deleted	2	1	0.4	0.2	1	0.4	0.5	**A_java.io.File.createTempFile(); A_java.io.File.deleteOnExit(); A**_java.nio.file.Files.createTempDirectory()
24: shutdown application gracefully upon power loss	1	–	0	0	1	0.6	0.6	**A_java.lang.Runtime.addShutdownHook(); A**_java.util.concurrent.ExecutorService.shutdownNow() **A**_java.util.Timer
25: get single bytes from multi-byte variable	1	1	0.2	0.1	1	0.4	0.5	**T**_Primitive Data Types;**A_java.nio.ByteBuffer; T**_Variables
average	1.4	> 1.94	0.32	0.284	1.32†	0.576†	0.568†	

6. Conclusion

Developers often encounter questions in specific programming tasks. Although programming languages and software packages are well supported by formal documentation, the documentation aims at comprehensive coverage and not on specific tasks. The semantic gap between the developers' questions and software documentation makes it difficult for developers to search for the most relevant documentation. Utilizing the social context available on Stack Overflow, we built `QDLinker` to bridge the gap between the questions and documentation. Given a programming question, `QDLinker` returns the links to the most relevant documentation. The semantic features between questions and software documentation in `QDLinker` are learned through a four-layer deep neural network. Together with content features, the learned features are fed to a learning-to-rank schema for ranking the most relevant software documentation at top position. Using real questions from Stack Overflow, we show that `QDLinker` effectively locates the most relevant software documentation to questions, and its performance significantly outperforms baseline methods.

The proposed `QDLinker` framework may benefit other software engineering problems. First, considering the ability of bridging the semantic gap between programming questions and software documentation, `QDLinker` could improve official software documentation with the information in questions and answers. Second, current code search does not support natural language. `QDLinker` can be integrated in code search engines to improve code search performance. Third, this work opens several interesting directions for future work with regard to automatic conversation between humans and computers. In the future, we will explore the applications of `QDLinker` to these problems.

References

[1] H. Bagci, P. Karagoz, Context-aware location recommendation by using a random walk-based approach, Knowl. Inf. Syst. 47 (2) (2016) 241–260.
[2] A. Berger, R. Caruana, D. Cohn, D. Freitag, V. Mittal, Bridging the lexical chasm: statistical approaches to answer-finding, in: Proceedings of the SIGIR, 2000, pp. 192–199.
[3] R.D. Burke, K.J. Hammond, V. Kulyukin, S.L. Lytinen, N. Tomuro, S. Schoenberg, Question answering from frequently asked question files: experiences with the FAQ finder system, AI Mag. 18 (2) (1997) 57.
[4] X. Cao, G. Cong, B. Cui, C.S. Jensen, A generalized framework of exploring category information for question retrieval in community question answer archives, in: Proceedings of the WWW, 2010, pp. 201–210.
[5] Y. Cheng, F.X. Yu, R.S. Feris, S. Kumar, A. Choudhary, S.-F. Chang, An exploration of parameter redundancy in deep networks with circulant projections, in: Proceedings of the ICCV, 2015, pp. 2857–2865.
[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (Aug) (2011) 2493–2537.
[7] T. Deselaers, S. Hasan, O. Bender, H. Ney, A deep learning approach to machine transliteration, in: Proceedings of the Fourth Workshop on Statistical Machine Translation, 2009, pp. 233–241.
[8] H. Duan, Y. Cao, C.-Y. Lin, Y. Yu, Searching questions by identifying question topic and question focus., in: Proceedings of the ACL, 2008, pp. 156–164.
[9] M.J. Er, Y. Zhang, N. Wang, M. Pratama, Attention pooling-based convolutional neural network for sentence modelling, Inf. Sci. (Ny) 373 (2016) 388–403.
[10] A. Figueroa, G. Neumann, Context-aware semantic classification of search queries for browsing community question–answering archives, Knowl. Based Syst. 96 (2016) 1–13.
[11] A. Gani, A. Siddiqa, S. Shamshirband, F. Hanum, A survey on indexing techniques for big data: taxonomy and performance evaluation, Knowl. Inf. Syst. 46 (2) (2016) 241–284.
[12] M. Grbovic, N. Djuric, V. Radosavljevic, F. Silvestri, N. Bhamidipati, Context-and content-aware embedding for query rewriting in sponsored search, in: SIGIR, 2015, pp. 383–392.
[13] X. Gu, H. Zhang, D. Zhang, S. Kim, Deep API learning, in: Proceedings of the FSE, ACM, 2016, pp. 631–642.
[14] K. He, J. Sun, Convolutional neural networks at constrained time cost, in: Proceedings of the CVPR, 2015, pp. 5353–5360.
[15] Y. Hou, C. Tan, X. Wang, Y. Zhang, J. Xu, Q. Chen, Hitszicrc: Exploiting classification approach for answer selection in community question answering, in: Proceedings of the SemEval, 15, 2015, pp. 196–202.
[16] J. Jeon, W.B. Croft, J.H. Lee, Finding similar questions in large question and answer archives, in: Proceedings of the CIKM, 2005, pp. 84–90.
[17] Z. Ji, F. Xu, B. Wang, B. He, Question-answer topic model for question retrieval in community question answering, in: Proceedings of the CIKM, 2012, pp. 2471–2474.
[18] J.A. Khan, M.A.Z. Raja, M.M. Rashidi, M.I. Syam, A.M. Wazwaz, Nature-inspired computing approach for solving non-linear singular Emden–Fowler problem arising in electromagnetic theory, Conn. Sci. 27 (4) (2015) 377–396.
[19] Y. Kim, Convolutional neural networks for sentence classification, arXiv:1408.5882 (2014).
[20] D. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv:1412.6980 (2014).
[21] A.J. Ko, R. DeLine, G. Venolia, Information needs in collocated software development teams, in: Proceedings of the ICSE, 2007, pp. 344–353.
[22] A.J. Ko, B.A. Myers, M.J. Coblenz, H.H. Aung, An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks, IEEE Trans. Softw. Eng. 32 (12) (2006) 971–987.
[23] Y. Kokkinos, K.G. Margaritis, Topology and simulations of a hierarchical Markovian radial basis function neural network classifier, Inf. Sci. (Ny) 294 (2015) 612–627.
[24] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[25] J. Li, L. Bao, Z. Xing, X. Wang, B. Zhou, BPMiner: mining developers' behavior patterns from screen-captured task videos, in: Proceedings of the SAC, ACM, 2016, pp. 1371–1377.
[26] J. Li, Z. Xing, D. Ye, X. Zhao, From discussion to wisdom: web resource recommendation for hyperlinks in stack overflow, in: Proceedings of the SAC, 2016, pp. 1127–1133.
[27] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv:1301.3781 (2013).
[28] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Proceedings of the NIPS, 2013, pp. 3111–3119.
[29] G.A. Miller, Wordnet: a lexical database for english, Commun. ACM 38 (11) (1995) 39–41.
[30] A. Munir, M.A. Manzar, N.A. Khan, M.A.Z. Raja, Intelligent computing approach to analyze the dynamics of wire coating with Oldroyd 8-constant fluid, Neural Comput. Appl. (2017) 1–25, doi:10.1007/s00521-017-3107-4.
[31] S. Nadi, S. Krüger, M. Mezini, E. Bodden, Jumping through hoops: why do java developers struggle with cryptography APIs? in: Proceedings of the ICSE, 2016, pp. 935–946.
[32] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the ICML, 2010, pp. 807–814.

[33] H. Nassif, M. Mohtarami, J. Glass, Learning semantic relatedness in community question answering using neural models, in: Proceedings of Rep4NLP@ACL, 2016, pp. 137–147.
[34] M. Nicosia, S. Filice, A. Barrón-Cedeno, I. Saleh, H. Mubarak, W. Gao, P. Nakov, G. Da San Martino, A. Moschitti, K. Darwish, et al., Qcri: answer selection for community question answeringexperiments for arabic and english, in: SemEval, 15, 2015, pp. 203–209.
[35] H. Niu, I. Keivanloo, Y. Zou, Learning to rank code examples for code search engines, Empir. Softw. Eng. 22 (1) (2017) 259–291, doi:10.1007/s10664-015-9421-5.
[36] D. Palomera, A. Figueroa, Leveraging linguistic traits and semi-supervised learning to single out informational content across how-to community question-answering archives, Inf. Sci. (Ny) 381 (2017) 20–32.
[37] G. Petrosyan, M.P. Robillard, R. De Mori, Discovering information explaining api types using text classification, in: Proceedings of the ICSE, 2015, pp. 869–879.
[38] C. Quoc, V. Le, Learning to rank with nonsmooth cost functions, NIPS 19 (2007) 193–200.
[39] M. Raghothaman, Y. Wei, Y. Hamadi, Swim: Synthesizing what i mean, in: Proceedings of the ICSE, 2016.
[40] M.A.Z. Raja, F.H. Shah, E.S. Alaidarous, M.I. Syam, Design of bio-inspired heuristic technique integrated with interior-point algorithm to analyze the dynamics of heartbeat model, Appl. Soft. Comput. 52 (2017) 605–629.
[41] S. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, M. Gatford, Okapi at TREC–3, in: Overview of the Third Text Retrieval Conference (TREC–3), Gaithersburg, MD: NIST, 1995, pp. 109–126.
[42] M.P. Robillard, R. Deline, A field study of API learning obstacles, Empir. Softw. Eng. 16 (6) (2011) 703–732.
[43] V.A. Rohani, Z.M. Kasirun, S. Kumar, S. Shamshirband, An effective recommender algorithm for cold-start problem in academic social networks, Math. Probl. Eng. 2014 (2014).
[44] T. Sakai, D. Ishikawa, N. Kando, Y. Seki, K. Kuriyama, C.-Y. Lin, Using graded-relevance metrics for evaluating community QA answer selection, in: Proceedings of the WSDM, 2011, pp. 187–196.
[45] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[46] A. Severyn, A. Moschitti, Learning to rank short text pairs with convolutional deep neural networks, in: Proceedings of the SIGIR, 2015, pp. 373–382.
[47] A. Severyn, A. Moschitti, Modeling relational information in question-answer pairs with convolutional neural networks, arXiv:1604.01178 (2016).
[48] P. Singh, E. Simperl, Using semantics to search answers for unanswered questions in q&a forums, in: Proceedings of the WWW, 2016, pp. 699–706.
[49] R. Sun, H. Cui, K. Li, M.-Y. Kan, T.-S. Chua, Dependency relation matching for answer selection, in: Proceedings of the SIGIR, 2005, pp. 651–652.
[50] Y. Sun, Y. Chen, X. Wang, X. Tang, Deep learning face representation by joint identification-verification, in: Advances in Neural Information Processing Systems, 2014, pp. 1988–1996.
[51] M. Surdeanu, M. Ciaramita, H. Zaragoza, Learning to rank answers on large online QA collections., in: Proceedings of the ACL, 8, 2008, pp. 719–727.
[52] M. Tan, C. dos Santos, B. Xiang, B. Zhou, Improved representation learning for question answer matching, in: Proceedings of the ACL, 2016, pp. 464–473.
[53] H. Toba, Z.-Y. Ming, M. Adriani, T.-S. Chua, Discovering high quality answers in community question answering archives using a hierarchy of classifiers, Inf. Sci. (Ny) 261 (2014) 101–115.
[54] J. Turian, L. Ratinov, Y. Bengio, Word representations: a simple and general method for semi-supervised learning, in: Proceedings of the ACL, 2010, pp. 384–394.
[55] T. Van Nguyen, A.T. Nguyen, T.N. Nguyen, Characterizing API elements in software documentation with vector representation, in: Proceedings of the ICSE, 2016, pp. 749–751.
[56] Q. Wu, C.J. Burges, K.M. Svore, J. Gao, Adapting boosting for information retrieval measures, Inf. Retr. Boston 13 (3) (2010) 254–270.
[57] X. Xue, J. Jeon, W.B. Croft, Retrieval models for question and answer archives, in: Proceedings of the SIGIR, 2008, pp. 475–482.
[58] R. Yan, Y. Song, H. Wu, Learning to respond with deep neural networks for retrieval-based human-computer conversation system, in: Proceedings of the SIGIR, 2016, pp. 55–64.
[59] C. Yang, L. Bai, C. Zhang, Q. Yuan, J. Han, Bridging collaborative filtering and semi-supervised learning: A neural approach for poi recommendation, in: Proceedings of the SIGKDD, 2017, pp. 1245–1254.
[60] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, J. Lu, Detecting high-quality posts in community question answering sites, Inf. Sci. (Ny) 302 (2015) 70–82.
[61] S.-J. Yen, Y.-C. Wu, J.-C. Yang, Y.-S. Lee, C.-J. Lee, J.-J. Liu, A support vector machine-based context-ranking model for question answering, Inf. Sci. (Ny) 224 (2013) 77–87.
[62] L. Yu, K.M. Hermann, P. Blunsom, S. Pulman, Deep learning for answer sentence selection, arXiv:1412.1632 (2014).
[63] G. Zhou, L. Cai, J. Zhao, K. Liu, Phrase-based translation model for question retrieval in community question answer archives, in: Proceedings of the ACL, 2011, pp. 653–662.
[64] G. Zhou, T. He, J. Zhao, P. Hu, Learning continuous word embedding with metadata for question retrieval in community question answering, in: Proceedings of the ACL, 2015, pp. 250–259.
[65] G. Zhou, Y. Liu, F. Liu, D. Zeng, J. Zhao, Improving question retrieval in community question answering using world knowledge., in: Proceedings of the IJCAI, 13, 2013, pp. 2239–2245.
[66] G. Zhou, Y. Zhou, T. He, W. Wu, Learning semantic representation with neural networks for community question answering retrieval, Knowl. Based Syst. 93 (2016) 75–83.
[67] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, L. Zhang, Learning to rank for question-oriented software text retrieval (t), in: Proceedings of the ASE, 2015, pp. 1–11.