SN Computer Science (2021) 2:69
https://doi.org/10.1007/s42979-020-00427-1

ORIGINAL RESEARCH

Sentence Embedding Models for Similarity Detection of Software Requirements

Souvick Das[1] · **Novarun Deb**[2] · **Agostino Cortesi**[3] · **Nabendu Chaki**[4]

Received: 11 August 2020 / Accepted: 11 December 2020 / Published online: 2 February 2021

Abstract

Semantic similarity detection mainly relies on the availability of laboriously curated ontologies, as well as of supervised and unsupervised neural embedding models. In this paper, we present two domain-specific sentence embedding models trained on a natural language requirements dataset in order to derive sentence embeddings specific to the software requirements engineering domain. We use cosine-similarity measures in both these models. The result of the experimental evaluation confirm that the proposed models enhance the performance of textual semantic similarity measures over existing state-of-the-art neural sentence embedding models: we reach an accuracy of 88.35%—which improves by about 10% on existing benchmarks.

Keywords Requirements similarity · Sentence embedding · Semantic similarity

Introduction

Context

Software requirement elicitation is mostly performed using natural language, due to the need of effective communication with the client and the use of documents that accurately describe the application scenarios.

The use of natural language processing (NLP) in various aspects of Requirements Engineering (RE) heavlily contributes to speed up the software production process and to improve the quality of the resulting systems [9]. NLP allows machines to understand and extract patterns from text data by applying various techniques such as semantic textual similarity, information retrieval, document classification, entity recognition and so on. Some of the advantages of using NLP in RE are as follows:

1. Support to functional and non-functional requirements classification.
2. Provision of design models for model-driven verification.
3. Reusability of software components using similarity measures.

In fact, software engineers are required to understand, analyze, and validate the requirements manually to generate the requirement specification document. Requirements engineers need to extract all related software requirements before preparing the final specifications. This also requires the classification of functional and non-functional requirements which is also a tedious job. Several classification algorithms (like [1, 33]) can be used to achieve classification of FRs and NFRs. Some of the researches are based

This article is part of the topical collection "Applications of Software Engineering and Tool Support" guest edited by Nabendu Chaki, Agostino Cortesi and Anirban Sarkar.

✉ Souvick Das
souvik.cmsa019@gmail.com

Novarun Deb
novarun_deb@iiitvadodara.ac.in

Agostino Cortesi
cortesi@unive.it

Nabendu Chaki
nabendu@ieee.org

[1] Department of Computer Science and Engineering, University of Calcutta, Kolkata 700106, India

[2] Indian Institute of Information Technology, Vadodara, Gandhinagar, Gujarat 382028, India

[3] Department of Environmental Science, Informatics, and Statistics, Ca' Foscari University, 30172 Venezia, Italy

[4] Department. of Computer Science and Engineering, University of Calcutta, Kolkata 700106, India

on clustering algorithms to classify FRs and NFRs [12]. On the other hand, Knauss et. al. [25] presents a socio-technical model for requirements classification. All these approaches are mostly dependent on good sentence embedding mechanisms. For example, a domain-specific corpus on computer science articles was used to generate embedding of words and capture the context of a word to improve comprehensibility [24]. NLP can be used in RE for finding relationships among multiple functional and non-functional requirements which are essential to build and analyze design models at different phases [29]. Several works (like [15, 22]) show reusable software engineering can be facilitated by finding the semantic similarity between natural language requirements.

Research Problem

Leveraging the benefits of NLP in requirements engineering requires mechanisms that are purely dependent on rich sentence (or word) embedding models. State-of-the-art pre-trained models represent generic, common sense knowledge and fail to achieve high accuracy in specific domains. In spite of having pre-trained word embedding models on *Stack Overflow* posts ([4, 11]), there is a significant lack of domain-specific pre-trained models in Requirements Engineering which could enhance the processing of natural language requirements documents. This is the main research problem being addressed in this paper. We intend to develop pre-trained sentence embedding models that are specifically designed for processing natural language requirements with high accuracy.

Contribution

In this paper, we propose two new domain-specific models which are specially designed for analyzing software requirements in natural language.

The first model, called *PUBER*, is based on the BERT [10] architecture and trained on the PURE [13] dataset, which contains 79 natural language requirements documents.

The second model, called *FiBER*, is more powerful than the first one. The original pre-trained BERT model is trained on Wikipedia articles and Book Corpus [36]. The proposed *FiBER* model utilizes the pre-trained BERT model to find the cosine-similarity between pairs of natural language requirements. This composite architecture is then fine-trained again on the PURE requirements dataset. This allows *FiBER* to utilize a larger vocabulary to understand natural language appropriately and be able to grasp the semantics of natural language requirements documents simultaneously.

Both the models are able to generate fixed-length sentence embeddings which are essential for many different NLP tasks.

Results

To evaluate the effectiveness of the two proposed domain-specific models in the domain of Requirements Engineering, we compared them with state-of-the-art sentence embedding models. The evaluation results show that our *PUBER* model achieves an accuracy of 79.23% for identifying semantically similar sentences and 79.52% for identifying dissimilar sentence pairs. *FiBER*, on the other hand, achieves 91.40% and 85.30% accuracy for similar and dissimilar sentences, respectively, i.e., an overall accuracy of 88.35% which improves by about 10% on existing benchmarks.

Organization of the Paper

The rest of the paper is organized as follows. Section "Related Work" presents the state-of-the-art sentence embedding models as well as several researches where RE community integrated NLP techniques with requirements engineering. In Section "Domain Special Sentence Embedding Models", we introduce our embedding models. Section "Experimental Evaluation" shows the results of an experimental analysis when comparing different sentence embedding models to find semantic textual similarity on requirements datasets. Section "Conclusion" concludes.

Related Work

Representation of words and sentences as vectors in a low dimensional space enables us to incorporate various deep learning NLP techniques to accomplish different challenging tasks. Word and sentence embeddings encode words and sentences in a fixed-length vectors to drastically enhance the performance of the NLP tasks. Word embeddings are considered to be an improvement over traditional bag-of-words model which provides large and sparse word vectors. Word2Vec [23] is the first neural embedding model, developed by Google researchers. Word2Vec represents words as multidimensional array. Two unsupervised algorithms namely Skip-gram and CBoW are used to generate the word embedding. Pennington et al. [26] proposed an unsupervised algorithm(GloVe) for obtaining vector representations of words based on aggregation of global word to word co-occurrence statistics from corpus.

Recently, the search for universal embedding has been gaining importance. Pre-trained embedding on a large corpus can be connected with a variety of downstream tasks such as classification, semantic textual similarity, sentiment analysis and so on to improve performance. This form of learning is referred as Transfer Learning. In [14], authors

have demonstrated that transfer learning can significantly improve the performance of NLP models on specific tasks, such as classification. Setting off the eruption of universal word embedding research, several works improved previous unsupervised approaches (Word2Vec and state of the art contextual word vectors) by incorporating the supervision of semantic or syntactic knowledge. The most noteworthy is FastText [16] and ELMo [27]. FastText's key enhancement over the initial Word2Vec vectors is the integration of character *n*-grams, which enables word representations to be determined for words that did not even exist in the training data ("out-of-vocabulary" words). Within ELMo, each word is given a representation that is a feature of the whole sentence of the corpus to which they belong. The embedding (E) is determined from the internal states of a two-layer bidirectional Language Model (LMo)and, hence, the name ELMo.

In the area of sentence embedding, there is a general perception that the simple technique of directly averaging the word vectors of a sentence (the so-called Bag-of-Word approach [23]) provides a good benchmark for several downstream tasks. However, this approach creates variable length embeddings for sentences of different lengths. In [2], authors have proposed an algorithm which can take any well-known word embedding mechanism to encode a sentence in a linear weighted combination fashion then compute the weighted average of the word vectors. Finally, the projection of the vectors on their first principal component is removed. The first major proposals went further than basic averaging of word vectors, using unsupervised training objectives. Skip-Thought Vectors [18] is an unsupervised learning method for sentence embeddings. It is analogous to word embedding Skip-Gram model. The model consists of an encoder–decoder based on Recurrent Neural Network (RNN) that is trained to recreate the surrounding sentences from the current sentence. An improvement of Skip-Thoughts vectors is presented in Quick-Thoughts Vectors [28]. The strength of the model is its speed of training. For a long period of time, supervised learning of sentence embedding was assumed to generate embedding of lower quality than unsupervised approaches. This assumption was recently reversed, particularly after the InferSent [8] model emerged. Infersent uses Stanford Natural Language Inference [5] (SNLI) corpus to train the classifier on top of a sentence encoder. The authors implement the sentence encoder using bi-directional LSTM coupled with max-pooling operator. In 2018, Cer et al. from Google research proposed Universal Sentence Encoder [7] which has become one of Tensorflow Hub's most downloaded pre-trained text modules, providing versatile sentence embedding models which turn sentences into vector representations. The Universal Sentence Encoder is trained with Deep Averaging Network (DAN) encoder. It is designed for a variety of tasks to understand the natural languages dynamically.

Bidirectional Encoder Representations from Transformers (BERT [10]) is a pre-trained model developed by Google researchers Devlin et al. in 2018. BERT is trained on gigabytes of data from various sources (mostly from Wikipedia and Book Corpus) in an unsupervised fashion. In brief, the training is performed by masking a few words (nearly 15% of the words) in a sentence and allowing the model to predict masked words. As the model is trained to predict, it also learns to generate an efficient internal representation of words as word embedding. The uniqueness of the BERT model is that it explores text representation from both directions to obtain a clearer understanding of meaning of the context and their relationship. BERT has set new state of the art results for several NLP tasks such as question answering, sentence classification, sentence pair regression and so on. To perform sentence pair regression, BERT accepts two sentences separated by special token SEP and applies multi-head attention layers. The output is then passed to a simple regression function to provide the final label. Using this architecture, BERT sets a new benchmark for performance on semantic textual similarity among state-of-the-art models. RoBERTa [20] has shown that minor modifications to pre-training processes can further enhance the efficiency of BERT. The major downside of BERT is that no independent sentence embedding mechanism is assessed. Two major approaches are used to generate sentence embedding:

1. *Averaging Method:* The most popular BERT methods for creating sentence embeddings by simply averaging the word embeddings of all words in one sentence.
2. *CLS vector*: Alternatively, the CLS special token embedding that appears at the beginning of the sentence may be used. ([21, 35]).

The well-known bert-as-a-service[1] repository offers both these options. In [19], authors have proposed another variant of BERT called ALBERT where two-parameter reduction techniques have been introduced to decrease the memory consumption and enhance the training speed of the model. They have also used a self-supervised loss that focuses on modeling inter-sentence coherence. In [34], authors have proposed XLNet which integrates ideas from Transformer-XL, where the model generalized auto-regressive pre-training mechanism. It involves bidirectional learning of contexts by maximizing the expected likelihood over all permutations of the factorization order. In another work [30], authors have come up with an idea to fine-tune BERT model on SNLI dataset. In this work, the modification of the BERT model

[1] https://github.com/hanxiao/bert-as-service/.

Table 1 Comparison of sentence embedding models

Parameters	Models							
	Weighted sum of vectors	Skip-thoughts	InferSent	Google's USE	BERT	RoBERTa	SBERT	XLNet
Learning method	Unsupervised	Unsupervised	Supervised	Unsupervised	Unsupervised	Unsupervised	Unsupervised	Unsupervised
Architecture	Feed-forward Neural network model (Skip-gram or CBoW)	GRU (Gated Recurrent Units) or LSTM (Long Short-Term Memory)	Bi-directional LSTM with Softmax classifier	Transformer or Deep Averaging Network	Bi-directional Transformer	Bi-directional Transformer	Fine-tuned BERT on SNLI with softmax classifier	Transformer architecture with recurrence
Trained dataset	Wikipedia	Can be trained on any text corpus	Trained on GloVe or FastTextSNLI	Wikipedia, web news,web Q/A	Wikipedia and Book Corpus	Wikipedia, Book Corpus, CommonCrawl News dataset and text corpus	Stanford Natural Language Inference Dataset	Wikipedia, Book Corpus, Common Crawl, Giga5, Clueweb etc.
Order of words	Not Considered	Considered	Considered	Considered	Considered	Considered	Considered	Considered
Semantic relation between texts	Not needed	Needed	Needed	Needed	Needed	Needed	Needed	Needed
STS Benchmark scores [6]	70	72.1	80.1	87.21	90	92.4	79.19	91.8

includes the inclusion of siamese and triplet network structures to produce semantically relevant sentence embeddings. Another major contribution of the paper is the generation of sentence embeddings that are compatible with cosine similarity measurements. In Table 1, we have summarized sentence embedding models based on different parameters. These parameters include the model architecture, leaning methods, training datasets for each of the models and so on. The parameter "Order of Words" specifies whether a particular model is aware of the orderings of words (uni-directional or bi-directional) or not. Another parameter "Semantic Relationship between Texts" specifies whether the model is context sensitive or context free. STS [6] Benchmark comprises a selection of the English datasets used in the Semantic Textual Similarity (STS) tasks. The datasets consist of text from image captions, headlines and articles from news and user forums. In Table 1, we have presented the similarity score measured for different models for STS datasets.

Current state-of-the-art features a range of well-known sentence embedding models, widely used for a number of NLP applications in different domains. Requirements Engineering is not unique in incorporating NLP for developing potential solutions of specific problems. However, this domain still lacks a rich domain-specific sentence embedding model which is the basic foundation for most NLP tasks.

Domain-Specific Sentence Embedding Models

In this section, we introduce two domain-specific sentence embedding models—namely *PUBER* and *FiBER*—for finding the similarity (and dissimilarity) between pairs of natural language requirements sentences. Both these models use the BERT neural network architecture [10].

Figure 1 depicts the architecture of the BERT model, which is originally trained on the Wikipedia data and Book Corpus.

Our first model *PUBER* uses the BERT architecture to generate a pre-trained model from the PURE dataset. On the other hand, *FiBER* uses the pre-trained BERT model to derive the cosine similarity between pairs of natural language requirements sentences. This composite architecture is then fine-trained using the PURE dataset. The main objective of this work is to train and build vocabulary for our models to leverage the benefits of NLP in the Requirements Engineering domain.

PUBER

The *PUBER* model is built on the same architecture of the BERT sentence embedding model as presented in Fig. 1. However, the model is trained on the PURE dataset

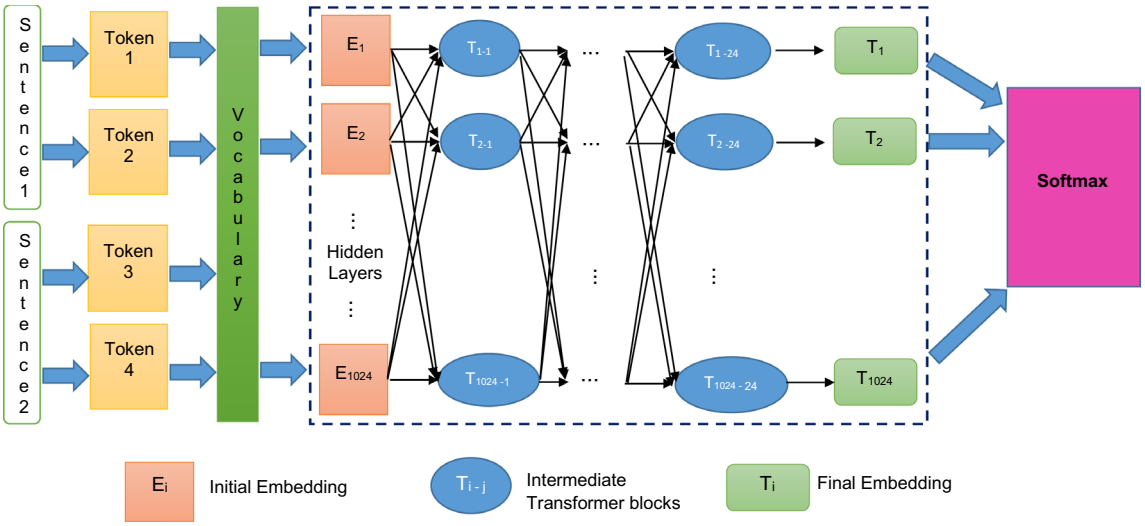


Fig. 1 BERT architecture

consisting of 34,268 unlabeled sentences. A distinctive feature of the BERT architecture is its uniformity across different tasks. There is minimal difference between the pre-trained model architecture and the architecture required for performing several downstream tasks like semantic similarity, classification, sentiment analysis, etc. *PUBER*'s workflow model contains the multi-layer bidirectional transformer encoder of the BERT architecture. Additionally, the cosine similarity is evaluated between sentence embeddings. The model is shown in Fig. 2.

In our evaluations, we use the original implementation proposed by Vaswani et al. [32]. With respect to Fig. 1, we have kept the number of intermediate transformer blocks (T_{i-j}) in each hidden layer to be 24, the number of hidden layers (E_i to T_i) to be 1024 and the number of self-attention heads to be 16 as proposed. We can describe the workflow model of *PUBER* with the help of Fig. 1 and the following steps.

1. The first step in this procedure is to build the domain-specific vocabulary. To build the vocabulary from the alphabet of single byte, we have used the default *WordPiece* embedding with 30,000 token vocabulary. Several characteristics of this vocabulary are presented as follows.
   (a) The classification CLS token is considered as the first token for every sequence.
   (b) Differentiation of sentences is taken care by using SEP token.
   (c) Our domain specific vocabulary is optimized for the PURE dataset. Compared to generic vocabulary trained for English, more requirements-specific words are represented by a single, unsplit token.
2. Once the vocabulary is prepared, we started training the language model. The vocabulary is then used for the word embeddings and masking.
3. As the model is based on BERT, we train it on a task of *Masked Language Modeling* [10] which masks some

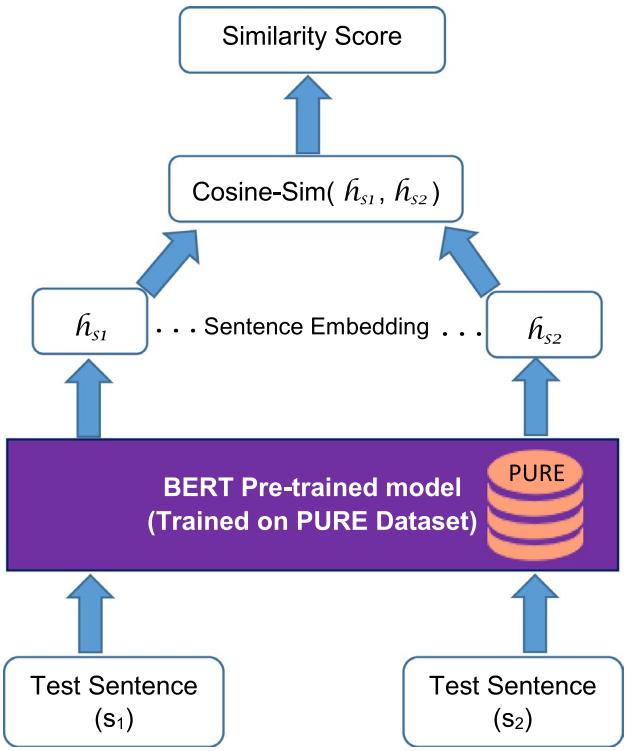


Fig. 2 PUBER Similarity Checking Model

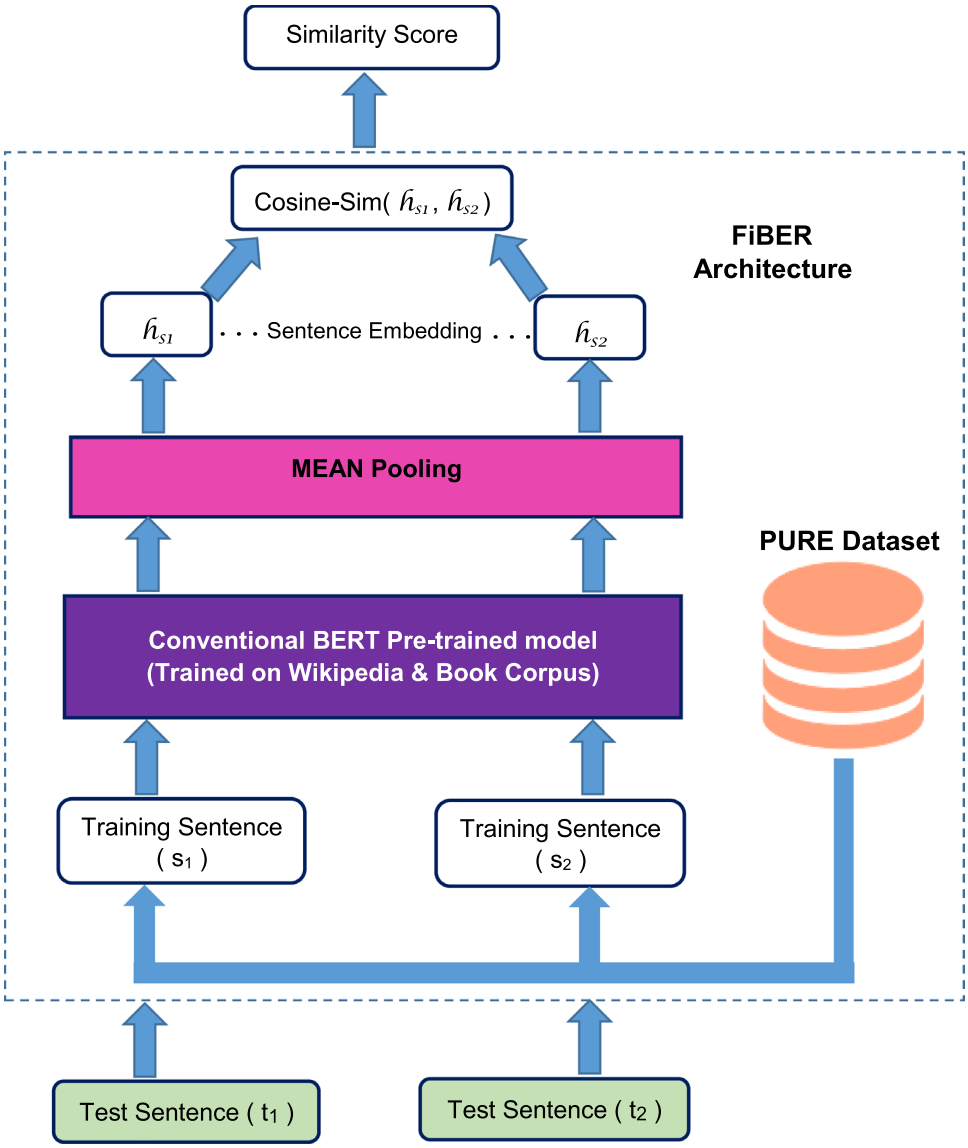


Fig. 3 FiBER similarity checking model

percentage of the input tokens at random, and then predicts those masked tokens.

4. The final hidden vectors corresponding to the mask tokens are fed into a softmax layer. The training environment created is described as follows.

   (a) Batch size is set to 32.
   (b) Number of training step is considered to be 100,000.
   (c) Learning rate is kept as 2e–5.

The described settings enable us to obtain a bidirectional pre-trained domain-specific language model. Once this model is developed, it is ready to be used for different downstream tasks. In this paper, we have measured the semantic similarity based on cosine similarity between two sentence embeddings. Figure 2 depicts the *PUBER* model which uses the BERT architecture that is trained on the PURE dataset. Essentially, two sentences are passed to our *PUBER* model—say s_1 and s_2. In the next phase, the *PUBER* model provides sentence embeddings for both the sentences—h_{s_1} and h_{s_2}, respectively. Finally, cosine similarity is measured between the two embeddings and a similarity score is evaluated.

FiBER

Model Architecture

The fine-trained *FiBER* model is built by augmenting the pre-trained BERT model with a pooling strategy on top of it. Figure 3 shows the architecture of the *FiBER* transformer model.

(a) At the foundation level, we have the BERT pre-trained model. We already discussed the BERT architecture with 1024 hidden layers, each with 24 transformer blocks (Fig. 1). Every layer does multi-headed attention computations on the word representation of the previous layer. The multi-headed attention computations create a new intermediate representation which is then fed to the next layer of hidden states. We keep the transformer architecture as it is in the original BERT model.

(b) On top of the BERT model, we augment a *MEAN Pooling* component. The pooling mechanisms are essential to get a fixed representation of a sentence. Thus, the transformer model accepts two sentence s_1 and s_2 and generates fixed-sized sentence embeddings—denoted by h_{s_1} and h_{s_2}, respectively. There are several pooling strategies available to perform certain tasks like classification, extraction of word embeddings and sentence embeddings. Four different pooling strategies are described as follows:

1. If the pooling is set to 'None', no pooling is applied. This will result in a [maximum-sequence-length, 1024] encode matrix for a sequence. This mode is useful for solving token level tasks like word embedding. Here, 1024 is the dimension of the encoder.
2. If pooling is set to 'CLS' tokens, only the vector corresponding to first 'CLS' token is retrieved and the output encode matrix will be [batch_size, 1024]. This pooling type is useful for solving sentence-pair classification tasks.
3. If pooling is set to 'MEAN', the embeddings will be the average of the hidden state of encoding layer on the time axis and the output encode matrix will be [batch_size, 1024]. This mode is particularly useful for sentence representation tasks.
4. Finally, if pooling is set to 'MAX', it takes the maximum of hidden state of encoding layers on the time axis. 'MAX' pooling is also useful for sentence representation tasks.

The results of experimental comparison of the three pooling strategies mentioned above are depicted in Table 2. The *FiBER* model exhibits best performance when adopting the 'MEAN' pooling strategy. Thus, we keep our default configuration to 'MEAN' pooling.

(c) To measure the similarity between two test sentences, we need to feed them to the neural network which updates the weights for generating fixed-sized sentence embeddings.

(d) At last, the cosine similarity is measured on these fixed-size sentence embeddings.

Table 2 Performance of *FiBER* for different pooling strategies

Pooling strategy	Average accuracy for test dataset
MAX	84.65
MEAN	88.35
CLS	80.25

Cosine similarity is generally used as a metric that measures the angle between vectors where the magnitude of the vectors are not considered. It could be the case where we work with sentences of uneven lengths. The number of occurrences of a particular word may be more frequent in one sentence than in the other. These are the situations where the semantic similarity between two sentences can be affected if we consider the spatial distance measures. Cosine similarity gives more accuracy for measuring semantic similarity as it measures the angle between two vectors rather than considering the spatial distance.

However, fixed-sized sentence embeddings are compatible for all standard similarity measuring (in terms of angle between vectors or spatial distance) methods like cosine similarity, correlation, Euclidean distance, Jaccard similarity and so on. It is worth mentioning here that we have calculated both cosine similarity and correlation between two fixed-length embeddings to measure the similarity between sentences. Both methods provide almost identical results.

Transformer Model Training

In Fig. 3, we represent training of our transformer model using the PURE dataset sentences within the dotted rectangular block. The PURE dataset contains 79 publicly available natural language requirements documents collected from the Web. It consists of 34,268 sentences. We have used the Cosine loss function [3] for each of the 4 epochs. The cosine loss function constrains the distribution of the features in the same class. It is designed specially for the cosine-similarity measurement. This loss function computes the cosine similarity between the sentence embeddings and minimizes the mean squared error loss. Furthermore, we used a batch size of 32 and the Adam optimizer [17] with learning rate of 3e–5. Finally, we tested our model on 800 pairs of unseen requirements sentences. We evaluated the performance metric, in this case, the cosine similarity between sentence embeddings is computed. We have considered the threshold of similarity metric to be 0.5 which is quite common while measuring cosine similarity.

Table 3 Comparison of sentence embedding techniques on natural language requirements dataset

Accuracy for Sentence Categories	Models						
	USE	Infersent	FiBER	PUBER	BERT	RoBERTa	DistillBERT
Accuracy for finding similar sentences in percentage	66.58	71.14	91.40	79.23	78.28	73.03	95.94
Accuracy for finding non-similar sentences in percentage	91.86	80.06	85.30	79.52	75.06	80.31	36.48
Average accuracy in percentage	78.50	75.60	88.35	79.37	76.75	76.50	67.62
False positives in percentage	8.39	17.26	14.96	20.20	24.67	19.42	63.25
False negatives in percentage	33.17	20.9	8.59	20.76	21.71	26.96	4.05

Experimental Evaluation

We compare the performance of our two models *PUBER* and *FiBER* with other state-of-the-art sentence embedding models, namely *Universal Sentence Encoder* (USE), *BERT*, *RoBERTa*, *DistilBERT* and *Infersent*. We have evaluated these mechanisms on 800 pairs of software requirements statements to measure semantic textual similarity between them. The dataset consists of pairs of sentences annotated with binary labels—'Yes' and 'No'. The label 'Yes' signifies that the statements are semantically related. The label 'No' signifies the opposite. The dataset is built by manual annotation. The requirements statements are taken from the requirements dataset provided by the OpenScience tera-PROMISE [31] repository. We have presented evaluation results by plotting graphs for different ranges of number of sentences from 100 to 800. The values are listed in Table 3.

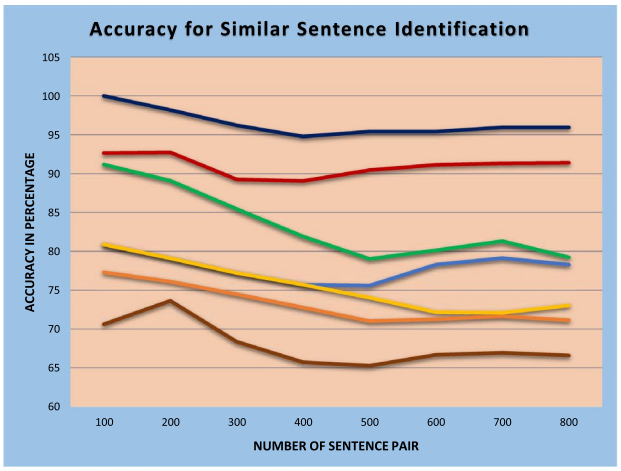
The evaluation result in Fig. 4a shows the accuracy of different approaches in order to identify semantically similar sentences. The figure shows *FiBER* gives over 91.40% of accuracy for identifying the semantically related sentences. Whereas, the *Universal Sentence Encoder* shows quite poor accuracy score of 66.58% for identifying semantically similar sentences. *BERT* achieves 78.28% accuracy score which is better than *RoBERTa*. *Infersent* gives slightly better accuracy score than *Universal Sentence Encoder* but is unable to beat *BERT* or *RoBERTa* for the same scenario. Our another model *PUBER* achieves better accuracy than *BERT* and *RoBERTa* to identify semnatically semantic sentences. The *DistillBERT* is very biased on identifying semantically similar sentence pair and gives 95.94% of accuracy. *DistillBERT* generates almost identical sentence embeddings for every sentence so that it predicts nearly every pair of sentence as semantically similar. This is the reason why it shows a high false positive of 63.25%.

Figure 4b presents the accuracy for different approaches in order to identify dissimilar sentences. It shows that our *FiBER* reaches 85.30% accuracy for identifying non-related sentences from the dataset. The *Universal Sentence Encoder* achieves highest accuracy of 91.86% for the same. *PUBER* achieves better accuracy than *BERT*. *RoBERTa* gives 80.31% of accuracy score for identifying dissimilar sentences. Finally, as we expect, *DistillBERT* gives poor result relative to other approaches because of its high false positives. It only achieves 36.48% of accuracy in order to identify dissimilar sentence pairs.

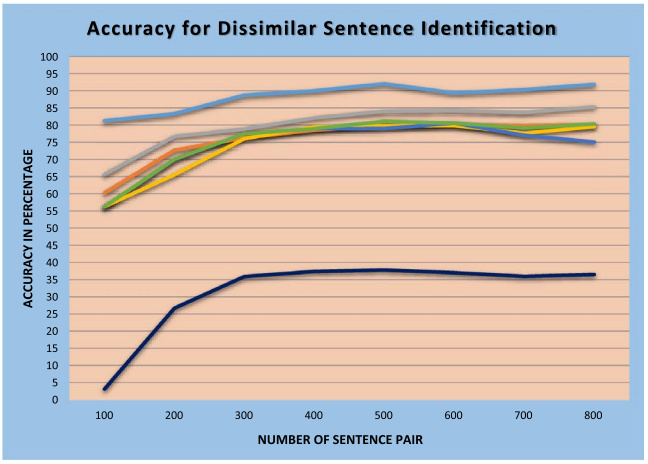
Figure 4c shows how the *FiBER* model outperforms other state of the art sentence embedding methods when applied to a mix of similar and dissimilar sentences. Our fine-trained model achieves an improvement of almost 10% on average over Google's Universal Sentence Encoder and 12% compared to *BERT* or *RoBERTa*. *FiBER* achieves 88.35% accuracy which is highest among all the state-of-the-art sentence embedding models. The *PUBER* model is also slightly better than other sentence embedding models (except *FiBER*). *USE*, *Infersent*, *BERT* and *RoBERTa* show quite similar accuracy scores, whereas *DistillBERT* has the worst accuracy.

In case of finding semantically similar sentences, although *DistillBERT* shows best results but percentage of false positives (63.25%) is also highest among all the approaches. The consequence of this situation, *DistillBERT* provides the worst outcome on the detection of dissimilar sentences. Google's *Universal Sentence Encoder* performs best for identifying dissimilar sentences whereas the performance for identifying semantically similar sentences is not quite well. The false-negative percentage is also quite high for *USE*—approximately 33%. *BERT* and *RoBERTa* provides almost similar accuracy on an average. On the other hand, our proposed *FiBER* model achieves the highest accuracy on average, and also performs well for both similar and dissimilar sentence recognition. On the other side, false-positive and false-negative percentage levels are also the second lowest for each case—14.96% and 8.59% respectively.

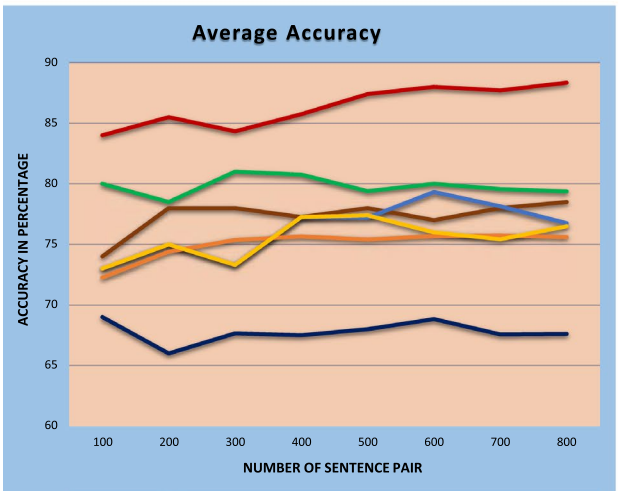
Considering the disjoint vocabulary and the scale of improvement over state-of-the-art well-known models like BERT, Google's Universal Sentence Encoder and Infersent, we conclude that when the Requirements Engineering domain-specific vocabulary and sentence embeddings are the key concern, *FiBER* and *PUBER* perform the best.



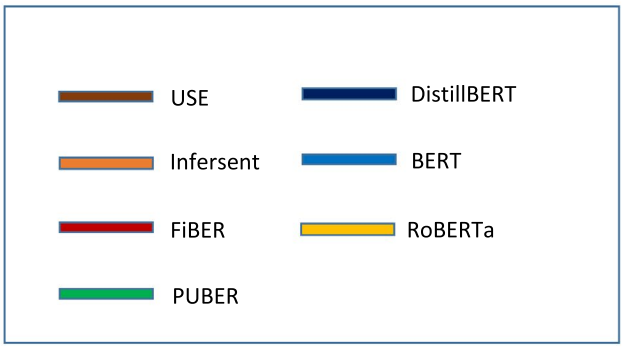
(a) Accuracy of sentence embedding models for similar sentences



(b) Accuracy of sentence embedding models for dissimilar sentences



(c) Average accuracy of sentence embedding models



(d) Legends

Fig. 4 Illustration of performance of different sentence embedding models

Conclusion

The *PUBER* model has a rich word piece vocabulary for Requirements Engineering domain. Since *PUBER* is purely trained on PURE requirements dataset, it does not have rich general English vocabulary. This is why the BERT model has been fine-trained on the PURE dataset to build the enhanced version which we call *FiBER*. The fine-trained model is able to make use of BERT's huge vocabulary and also understand specific words from the Requirements Engineering domain.

Since we have built sentence embedding models for the Software Requirements domain, we can empower different NLP tasks within the Requirements Engineering domain. In the future direction, we aim to apply our model to accomplish several such NLP tasks with our proposed sentence embedding models. These include requirements classification, named entity recognition, and sentiment analysis to understand code quality in code repositories, checking code similarity and so on.

Acknowledgements This work has been partially supported by the Project IN17MO07 "Formal Specification for Secured Software System", under the Indo-Italian Executive Programme of Scientific and Technological Cooperation.

Compliance with Ethical Standards

Conflict of Interest Statement On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Abad ZSH, Karras O, Ghazi P, Glinz M, Ruhe G, Schneider K. What works better? A study of classifying requirements. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), IEEE; 2017; p. 496–501.
2. Arora S, Liang Y, Ma T. A simple but tough-to-beat baseline for sentence embeddings. In: International Conference on learning representations; 2016; p. 1–16.
3. Barz B, Denzler J. Deep learning on small datasets without pre-training using cosine loss. In: The IEEE Winter Conference on applications of computer vision, 2020; p. 1371–380.
4. Biswas E, Vijay-Shanker K, Pollock L. Exploring word embedding techniques to improve sentiment analysis of software engineering texts. In: 2019 IEEE/ACM 16th International Conference on mining software repositories (MSR), IEEE, 2019; p. 68–78.
5. Bowman SR, Angeli G, Potts C, Manning CD. A large annotated corpus for learning natural language inference. In: Proceedings of the 2015 Conference on empirical methods in natural language processing, association for computational linguistics, Lisbon, Portugal, 2015. ;. 632–42, https://doi.org/10.18653/v1/D15-1075.
6. Cer D, Diab M, Agirre E, Lopez-Gazpio I, Specia L. Semeval-2017 task 1: semantic textual similarity-multilingual and cross-lingual focused evaluation. In: Association for Computational Linguistics, 2017; p. 1–14. https://www.aclweb.org/anthology/S17-2001, arXiv preprint arXiv:1708.00055.
7. Cer D, Yang Y, Kong Sy, Hua N, Limtiaco N, St John R, Constant N, Guajardo-Cespedes M, Yuan S, Tar C, Strope B, Kurzweil R. Universal sentence encoder for Eenglish. In: Proceedings of the 2018 Conference on empirical methods in natural language processing: system demonstrations, association for computational linguistics, Brussels, Belgium, 2018; p. 169–74, https://doi.org/10.18653/v1/D18-2029
8. Conneau A, Kiela D, Schwenk H, Barrault L, Bordes A. Supervised learning of universal sentence representations from natural language inference data. In: Proceedings of the 2017 Conference on empirical methods in natural language processing, association for computational linguistics, Copenhagen, Denmark, 2017; p. 670–80, https://doi.org/10.18653/v1/D17-1070
9. Dalpiaz F, Ferrari A, Franch X, Palomares C. Natural language processing for requirements engineering: the best is yet to come. In: IEEE software, IEEE, 2018;35:115–19.
10. Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019; p. 4171–186, https://doi.org/10.18653/v1/N19-1423
11. Efstathiou V, Chatzilenas C, Spinellis D. Word embeddings for the software engineering domain. In: Proceedings of the 15th International Conference on mining software repositories, 2018; p. 38–41.
12. Eyal Salman H, Hammad M, Seriai AD, Al-Sbou A. Semantic clustering of functional requirements using agglomerative hierarchical clustering. In: Information, Multidisciplinary Digital Publishing Institute, 2018;9: 222.
13. Ferrari A, Spagnolo GO, Gnesi S. Pure: a dataset of public requirements documents. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), IEEE, 2017; p. 502–5.
14. Howard J, Ruder S. Universal language model fine-tuning for text classification. In: Proceedings of the 56th Annual Meeting of the association for computational linguistics (Volume 1: Long Papers), association for computational linguistics, Melbourne, Australia, 2018; p. 328–39, https://doi.org/10.18653/v1/P18-1031
15. Ilyas M, Kung J. A similarity measurement framework for requirements engineering. In: 2009 Fourth International Multi-Conference on computing in the global information technology, IEEE, 2009; p. 31–4.
16. Joulin A, Grave E, Bojanowski P, Mikolov T. Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the association for computational linguistics: volume 2, short papers, association for computational linguistics, Valencia, Spain, 2017;.p. 427–31.
17. Kingma DP, Ba J. Adam: a method for stochastic optimization. In: 3rd International Conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings; 2015.
18. Kiros R, Zhu Y, Salakhutdinov RR, Zemel R, Urtasun R, Torralba A, Fidler S. Skip-thought vectors. In: Advances in neural information processing systems, 2015' p. 3294–302.
19. Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R. Albert: a lite bert for self-supervised learning of language representations. In: arXiv preprint arXiv:1909.11942 2019.
20. Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V. Roberta: A robustly optimized Bert pretraining approach. In: arXiv preprint arXiv:1907.11692 2019.
21. May C, Wang A, Bordia S, Bowman SR, Rudinger R. On measuring social biases in sentence encoders. In: Proceedings of the 2019 Conference of the North American Chapter of the association for computational linguistics: human language technologies, volume 1 (Long and Short Papers), association for computational linguistics, Minneapolis, Minnesota, 2019; p. 622–28, https://doi.org/10.18653/v1/N19-1063
22. Mihany FA, Moussa H, Kamel A, Ezzat E, Ilyas M. An automated system for measuring similarity between software requirements. In: Proceedings of the 2nd Africa and Middle East Conference on software engineering, 2016; p. 46–51.
23. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In: NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing Systemss, Vol. 2. Red Hook, NY: Curran Associates Inc.; 2013. p. 3111–9.
24. Mishra S, Sharma A. On the use of word embeddings for identifying domain specific ambiguities in requirements. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), IEEE, 2019; p. 234–40.
25. Ott D. Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In: International Working Conference on requirements engineering: foundation for software quality, Springer, 2013; p. 50–64.
26. Pennington J, Socher R, Manning CD. Glove: global vectors for word representation. In: Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP), 2014; p. 1532–543.
27. Peters M, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L. Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the association for computational linguistics: human language technologies, volume 1 (Long Papers), association for computational linguistics, New Orleans, Louisiana, 2018; p. 2227–237, https://doi.org/10.18653/v1/N18-1202
28. Quan Z, Wang Z, Le Y, Yao B, Li K, Yin J. An efficient framework for sentence similarity modeling. IEEE/ACM Trans Audio Speech Lang Process. 2019;27:853–65.
29. Rahimi M, Mirakhorli M, Cleland-Huang J. Automated extraction and visualization of quality concerns from requirements specifications. In: 2014 IEEE 22nd International Requirements Engineering Conference (RE), IEEE, 2014; p. 253–62.
30. Reimers N, Gurevych I. Sentence-BERT: sentence embeddings using Siamese BERT-networks. In: Proceedings of the 2019

Conference on empirical methods in natural language processing and the 9th International Joint Conference on natural language processing (EMNLP-IJCNLP), association for computational linguistics, Hong Kong, China, 2019; p. 3982–992, https://doi.org/10.18653/v1/D19-1410
31. Shirabad JS, Menzies TJ. The promise repository of software engineering databases. In: School of information technology and engineering, University of Ottawa, Canada, 2005; vol 24.
32. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: NIPS 2017: 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. Red Hook, NY: Curran Associates Inc. 2017; p. 5998–6008.
33. Winkler J, Vogelsang A. Automatic classification of requirements based on convolutional neural networks. In: 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), IEEE, 2016; p. 39–45.
34. Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov RR, Le QV. Xlnet: generalized autoregressive pretraining for language understanding. In: Advances in neural information processing systems, 2019; p 5754–764.
35. Zhang T, Kishore V, Wu F, Weinberger KQ, Artzi Y. Bertscore: evaluating text generation with Bert. In: 8th International Conference on learning representations, ICLR, 2020; 2020.
36. Zhu Y, Kiros R, Zemel R, Salakhutdinov R, Urtasun R, Torralba A, Fidler S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In: Proceedings of the IEEE International Conference on computer vision, 2015; p. 19–27.