Using Word Embeddings for Text Search and Retrieval

Lecture Notes on Deep Learning

Avi Kak and Charles Bouman

Purdue University

Tuesday 28th April, 2020 11:54

Preamble

In the year 2013, a group of researchers at Google created a revolution in the text processing community with the publication of the *word2vec* neural network for generating numerical representations for the words in a text corpus. Here are their papers on that discovery:

`https://arxiv.org/abs/1301.3781`

`https://arxiv.org/pdf/1310.4546.pdf`

The word-to-numeric representations created by the word2vec network are vectors of real numbers, the size of the vectors being a hyperparameter of the network. These vectors are called word embeddings.

The word embeddings generated by word2vec allow us to establish word similarities on the basis of word contexts.

To elaborate, if two different words, $word_i$ and $word_j$, are frequently surrounded by the same set of other words (called the context words), the word embedding vectors for $word_i$ and $word_j$ would be numerically close.

Preamble (contd.)

What's amazing is that when you look at the similarity clusters produced by word2vec, they create a powerful illusion that a computer has finally solved the mystery of how to automatically learn the semantics of the words.

My goal in this lecture is to introduce you to the word2vec neural network and to then present some of my lab's research on using the Word2vec embeddings in the context of software engineering for automatic bug localization.

However, in order to underscore the importance of word2vec, I am going to start with the importance of text search and retrieval in our lives and bring to your attention the major forums where this type of research is presented.

Outline

Outline

Role of Text Search and Retrieval in our Lives

- A good indicator of the central importance of text search and retrieval is the number of times you google something in a 24-hour period.
- Even without thinking, we now bring up an app on our mobile device or on a laptop to get us instantaneously what it is that we are looking for. We could be looking for how to get to a restaurant, how to spell a long word, how to use an idiom properly, or about any of a very large number of other things.
- You could say that instantaneous on-line search has now become an important part of our reflexive behavior.
- In my own lab (RVL), we now have a history of around 15 years during which we have used text search and retrieval tools to develop ever more powerful algorithms for automatic bug localization.

Text Search and Retrieval and Us (contd.)

- The earliest approaches to text search and retrieval used what are known as the Bag-of-Words (BoW) algorithms. With BoW, you model each document by a histogram of its word frequencies. You think of the histogram as a vector whose length equals the size of the vocabulary. Given a document, you place in each element of this vector the number of times the word corresponding to that element appears in the document.
- The document vector constructed as described above is called the term frequency vector. And the term-frequency vectors for all the documents in a corpus define what is known as the term-frequency matrix.
- You construct a similar vector representation for the user's query for document retrieval. Using cosine distance between the vectors as a similarity metric, you return the documents that are most similar to the query.

Text Search and Retrieval and Us (contd.)

- The BoW based text retrieval algorithms were followed by approaches that took term-term order into account. This was a big step forward. The best of these were based on what is known as the Markov Random Fields (MRF) model.
- In an MRF based framework, in addition to measuring the frequencies of the individual words, you also measure the frequencies of ordered pairs of words. The results obtained with MRF based approach were a significant improvement over those obtained with BoW based methods.
- Most modern approaches to text search and retrieval also include what I called "contextual semantics" in the modeling process. The best-known approach for that is the word2vec neural network that is the focus of this lecture.

Important Forums for Research Related to Text Retrieval

- Before ending this section, I just wanted to mention quickly that the most important annual research forums on general text retrieval are the ACM's SIGIR conference and the NIST's TREC workshops. Here are the links to these meetings:

  `https://sigir.org/sigir2019/`

  `https://trec.nist.gov/pubs/call2020.html`

- Since my personal focus is on text retrieval in the context of extracting information from software repositories, the go-to meeting for me is the annual MSR (Mining Software Repositories) conference:

  `https://2020.msrconf.org/track/msr-2020-papers#event-overview`

  This link is to the list of accepted papers at this year's conference. I chose this link intentionally since it shows our own paper at the top of list. (Nothing wrong with a bit of self promotion! Right?)

Outline

Word2vec

- I'll explain the word2vec neural network with the help of the figure shown on the next slide.
- The files in a text corpus are scanned with a window of size $2W + 1$. The word in the middle of the window is considered to be the focus word. And the W words on either side are referred to as the context words for the focus word.
- We assume that the size of the vocabulary is V.
- As a text file is scanned, the V-element long one-hot vector representation of each focus word is fed as input to the neural network.
- Each such input goes through the first linear layer where it is multiplied by a matrix, denoted $W_{V \times N}$, of learnable parameters.

Word2vec (contd.)

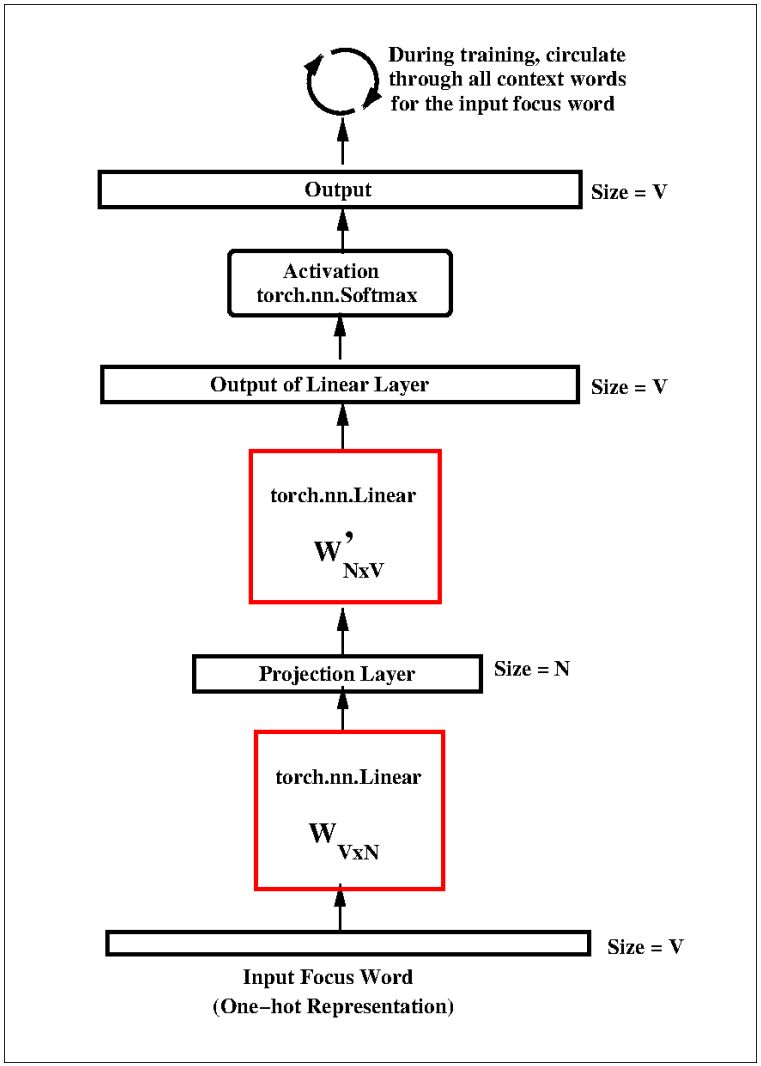

Figure: The SkipGram model for generating the word2vec embeddings for a text corpus.

Word2vec (contd.)

- Overall, in the word2vec network, a V-element tensor at the input goes into an N-element tensor in the middle layer and, eventually, back into a V-element final output tensor.
- Here is a very important point about the first linear operation on the input in the word2vec network: In all of the DL implementations you have seen so far, a `torch.nn.Linear` layer was always followed by a nonlinear activation, but that's NOT the case for the first invocation of `torch.nn.Linear` in the word2vec network.
- The reason for the note in red above is that the sole purpose of the first linear layer is merely to serve as a projection operator.
- But what does that mean?
- To understand what we mean by a projection operator, you have to come to terms with the semantics of the matrix $W_{V \times N}$. And that takes us to the next slide.

Word2vec (contd.)

- You see, the matrix $W_{V \times N}$ is actually meant to be a stack of the word embeddings that we are interested in. The i^{th} row of this matrix stands for the N-element word embedding for the i^{th} word in a sorted list of the vocabulary.
- Given the one-hot vector for, say, the i^{th} vocab word at the input, the purpose of multiplying this vector with the matrix $W_{V \times N}$ is simply to "extract" the current value for the embedding for this word and to then present it to the neural layer that follows.
- You could say that, for the i^{th}-word at the input, the role of the $W_{V \times N}$ matrix is to project the current value of the word's embedding into the neural layer that follows. It's for this reason that the middle layer of the network is known as the projection layer.
- In case you are wondering about the size of N vis-a-vis that of V, that's a hyperparameter of the network whose value is best set by trying out different values for N and choosing the best.

Word2vec (contd.)

- After the projection layer, the rest of the word2vec network as shown in Slide 12 is standard stuff. You have a linear neural layer with `torth.nn.Softmax` as the activation function.
- To emphasize, the learnable weights in the $N \times V$ matrix W' along with the activation that follows is the only neural layer in word2vec.
- You can reach the documentation on the activation function (`torch.nn.Softmax` through the main webpage for `torch.nn`. This activation function is listed under the category "Nonlinear activations (Other)" at the "`torch.nn`" webpage.
- To summarize, word2vec is a single-layer neural network that uses a projection layer as its front end.
- While the figure on Slide 12 is my visualization of how the data flows forward in a word2vec network, a more common depiction of this network as shown on the next slide.

Word2vec (contd.)

- This figure is from the following publication by Shayan Akbar and myself:

https://engineering.purdue.edu/RVL/Publications/Akbar_SCOR_Source_Code_Retrieval_2019_MSR_paper.pdf

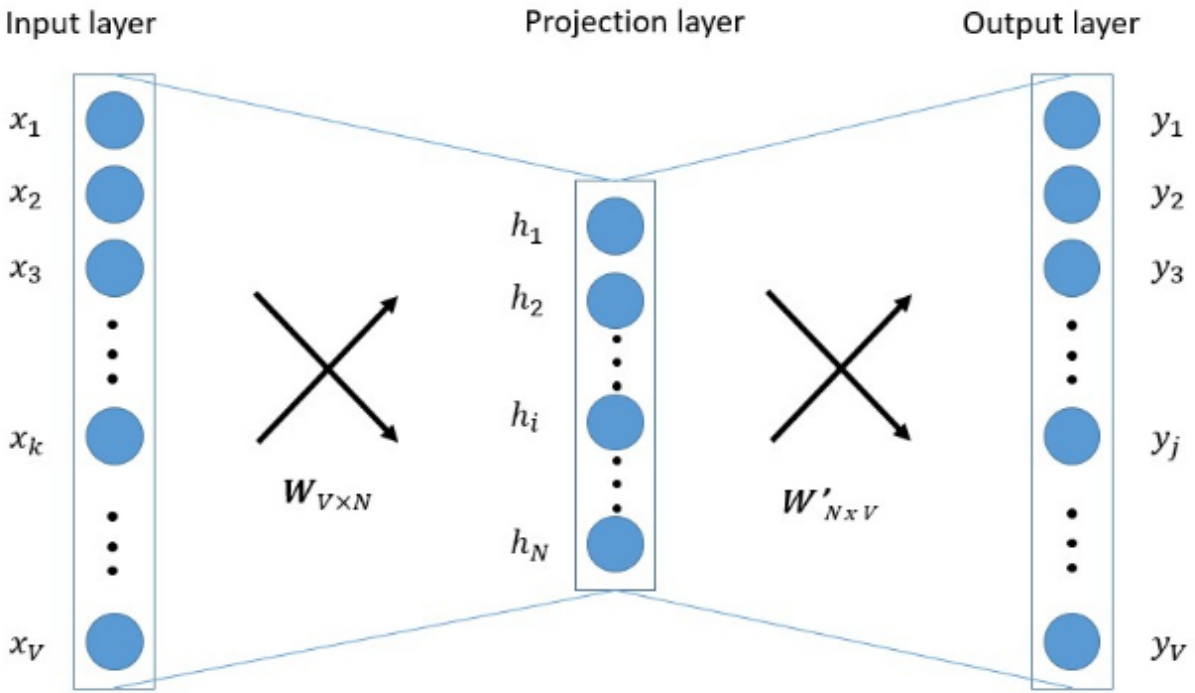

Figure: A more commonly used depiction for the SkipGram model for generating the word2vec embeddings for a vocabulary

Outline

The Softmax Activation Function

- An important part of education is to see how the concepts you have already learned relate to the new concepts you are about to learn. The following comment is in keeping with that precept.
- As you saw in the Lecture on Recurrent Neural Networks (see Slide 22 of that lecture), the *activation* function `LogSoftmax` and the *loss* function `NLLLoss` are typically used together and their joint usage amounts to the same thing as using the *loss* function `CrossEntropyLoss` that I had presented previously in the lecture on Object Detection and Localization (see Slide 8 of that lecture).
- In the sense mentioned above, we can say that `NLLLoss`, `LogSoftmax`, and `CrossEntropyLoss` are closely related concepts, despite the fact that two of them are loss functions and one an activation function.
- On the next slide, I'll add to the mix of those three related concepts the activation function `Softmax`.

The Softmax Activation Function (contd.)

- The Softmax activation function shown below looks rather similar to the cross-entropy loss function presented on Slide 8 of the lecture on Object Detection:

$$Softmax(x_i) \;=\; \frac{e^{x_i}}{\sum_j e^{x_j}}$$

yet the two functions carry very different meanings, which has nothing to do with the fact that the cross-entropy formula requires you take the negative log of a ratio that looks like what is shown above.

- The cross-entropy formula presented in the Object Detection lecture focuses specifically on just that output node that is supposed to be the true class label of the input notwithstanding the appearance of all the nodes in the denominator that is used for the normalization of the value at the node that the formula focuses on.

- On the other hand, the Softmax formula shown above places equal focus on all the output nodes. That is, the values at all the nodes are normalized by the same denominator.

The Softmax Activation Function (contd.)

- The best interpretation of the formula for Softmax shown on the previous slide is that it converts all the output values into a probability distribution.
- As to why, the value of the ratio shown in the formula is guaranteed to be positive, is guaranteed to not exceed 1, and the sum of the ratios calculated at all the output nodes is guaranteed to sum to 1 exactly.
- That the output of the activation function can be treated as a probability is important to word2vec because it allows us to talk about each output as being the conditional probability of the corresponding word in the vocab being the context word for the input focus word.
- To elaborate, let w_i represent the i^{th} row of the W_{VxN} matrix of weights for the first linear layer and let w'_j represent the j^{th} column of the W'_{NxV} matrix of weights for the second linear layer in the word2vec network.

The Softmax Activation Function (contd.)

- If we use x_j to denote the output of the second linear layer (that is, prior to its entering the activation function) at the j^{th} node, we can write

$$x_j = w'^{T}_{j} w_i$$

- In light of the probabilistic interpretation given to the output of the activation function, we now claim the following: If we let $p(j|i)$ be the conditional probability that the j^{th} vocab word at, obviously, the j^{th} output node is a context word for the i^{th} focus word, we have

$$p(j|i) = \frac{e^{w'^{T}_{j} w_i}}{\sum_k e^{w'^{T}_{k} w_i}}$$

- The goal of training a word2vec network is to maximize this conditional probability for the actual context words for any given focus word.

The Softmax Activation Function (contd.)

- That takes us to the following issues:
  1. How to best measure the loss between the true conditional probability for the context words and the current estimates for the same; and
  2. How to backpropagate the loss?
- These issues are addressed in the next section.

Outline

Training for Word2vec

- As you already know, the word2vec network is training by scanning the text corpus with a window of size $2W + 1$, where W is typically between 5 and 10, for each focus word, testing the output against the one-hot representations for the $2W$ context words for the focus word.
- That raises the issue of how to present the context words at the output for the calculation of the loss.
- In keeping with the conditional probability interpretation of the forward-projected output as presented in the last section, the target output could be a V-element tensor that is a $2W$-version of a one-hot tensor: A V-element tensor in which just those elements are 1 that correspond to the context words, with the rest of the elements being 0s.

Calculating the Loss

- A target tensor such as the one described above would look like

$$p_T(j|i) \;=\; \begin{cases} \frac{1}{2W+1} & j \text{ is context word for } i \\ 0 & \text{otherwise} \end{cases}$$

- The calculation of loss now amounts to comparing the estimated probability distribution on Slide 21 against the target probability distribution shown above. This is best done with the more general cross-entropy loss formula shown on Slide 10 of the lecture on Object Detection. With the notation I am using in this lecture, that formula becomes:

$$\begin{aligned} Loss_i(p_T, p) &= -\sum_{j \in context(i)} p_T(j|i) \cdot \log_2 p(j|i) \\ &= -\sum_{j \in context(i)} \frac{1}{2W+1} \cdot \log_2 \left(\frac{e^{w'^T_j w_i}}{\sum_k e^{w'^T_k w_i}} \right) \\ &= \sum_{j \in context(i)} -\log_2 \left(e^{w'^T_j w_i} \right) + \log_2 \left(\sum_k e^{w'^T_k w_i} \right) \qquad \text{ignoring inconsequential terms} \\ &= \sum_{j \in context(i)} -w'^T_j w_i + \log_2 \left(\sum_k e^{w'^T_k w_i} \right) \end{aligned}$$

Computing the Gradients of the Loss

- The subscript i in the notation $Loss_i$ is meant to indicate that the i^{th} of the vocabulary is the focus word at the moment and that the loss at the output is being calculated for that focus word.
- Given the simple form for the loss as shown on the last slide, it is easy to write the formulas for the gradients of the loss with respect to the all the learnable parameters. To see how that would work, here is a rewrite of the equation shown at the bottom of the previous slide:

$$Loss_i = \sum_{j \in context(i)} -w'^{T}_{j} w_i + \log_2 \left(\sum_k e^{w'^{T}_{k} w_i} \right)$$

  where the subscript i on $Loss$ means that this the loss when the word whose position index in the vocab is i is fed into the network.
- As shown on the next slide, the form presented above lends itself simply to the calculation of the gradients of the loss for updating the learnable weights.

The Gradients of the Loss (contd.)

- For arbitrary values for the indices s and t, we get the following expression for the gradients of the loss with respect to the elements of the matrix W:

$$\frac{\partial Loss_i}{\partial w_{st}} = \sum_{j \in context(i)} -\frac{\partial(\vec{w'}_j^T \vec{w}_i)}{\partial w_{st}} + \frac{1}{\sum_k e^{\vec{w'}_k^T \vec{w}_i}} \frac{\partial(\sum_k e^{\vec{w'}_k^T \vec{w}_i})}{\partial w_{st}}$$

  where I have introduced the arrowed vector notation $\vec{w}_i$ so that you can distinguish between the elements of the matrix and the row and column vectors of the same matrix.

- You will end up with a similar form for the loss gradients $\frac{\partial Loss_i}{\partial w'_{st}}$.

- I leave it to the reader to simplify further these expressions. You might find useful the derivations presented in the following paper:

  `https://arxiv.org/abs/1411.2738`

Outline

Using Word2Vec for Text Retrieval

- The numerical word embeddings generated by the word2vec network allow us to establish an easy-to-calculate similarity criterion for the words: We consider two words to be similar if their embedding vectors are close using, say, the Euclidean distance between them.
- Just to show an example of how powerful word2vec is at establishing word similarities (that a lot of people would refer to as "semantic similarities"), the figure at the top on the next slide shows the word cluster discovered by word2vec for software repositories.
- The similarity clusters shown on the next slide were obtained by Shayan Akbar in his research on automatic bug localization. To generate the word embeddings, he downloaded 35,000 Java repositories from GitHub which resulted in 0.5 million software-centric terms. So 500,000 is the size of the vocabulary here. He used $N = 500$ for size of the word embedding vectors.

Using Word2Vec for Text Retrieval (contd.)

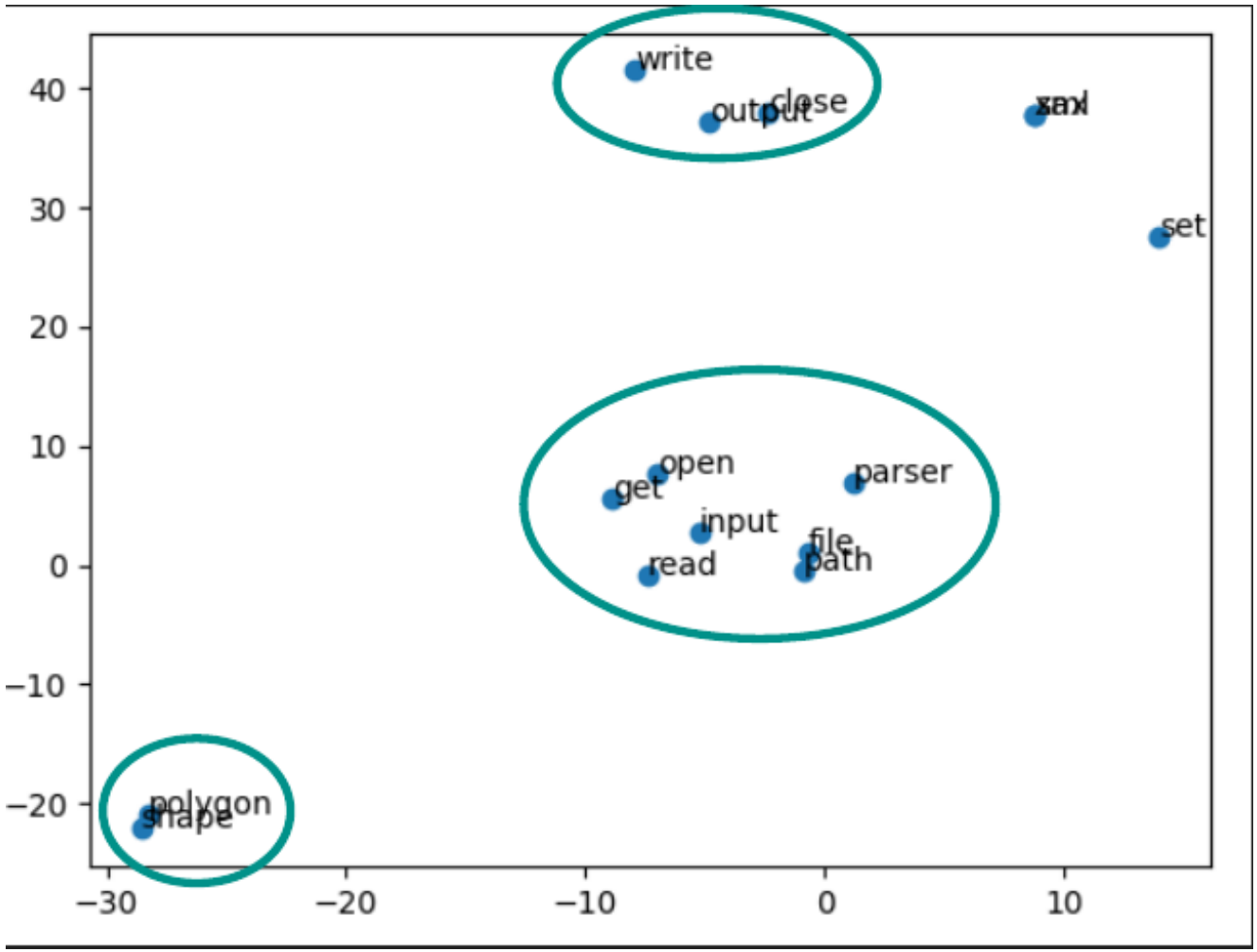

Figure: Some examples of word clusters obtained through the similarity of their embeddings.

Using Word2Vec for Text Retrieval (contd.)

- Show below are additional examples of word similarities discovered through the same word embeddings as mentioned on Slide 29. In this display, we seek the three words that are most similar to the words listed in the top row.
- You've got to agree that it is almost magical that after digesting half a million software-centric words, the system can figure out automatically that "parameter", "param", "method", and "argument" are closely related concepts. The same comment applies to the other columns in the table.

alexnet	delete	rotation	add	parameter
resnet	remove	angle	list	param
lenet	update	rot	set	method
imagenet	copy	lhornang	create	argument

Figure: Additional examples of software-centric word similarities based on learned their embeddings.

Using Word2Vec for Text Retrieval (contd.)

- Now that we know how to establish "semantic" word similarities, the question remains how to use the similarities for improving the quality of retrieval.
- How that problem was solved in the context of retrieval from software repositories is described in our 2019 MSR publication:

https://engineering.purdue.edu/RVL/Publications/Akbar_SCOR_Source_Code_Retrieval_2019_MSR_paper.pdf