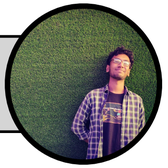
Vocabulary, Code Embeddings and OOV

Dr. Antonio Mastropaolo

Instructor

Mr. Alvi Haque



Teaching Assistant

WILLIAM & MARY

CHARTERED 1693

Spring 2026

Vocabulary, Code Embeddings and OOV

Open-Vocabulary Problem

Any piece of code can be seen as a sequence of tokens

- *Language keywords*
- *Punctuation symbols*
- *Identifiers*
- *Literals*
- *Numerical values*
- *Comments*

Vocabulary, Code Embeddings and OOV

Open-Vocabulary Problem

Any piece of code can be seen as a sequence of tokens

Language keywords
Punctuation symbols
Identifiers
Literals
Numerical values
Comments

Fixed by the programming language

Vocabulary, Code Embeddings and OOV

Open-Vocabulary Problem

Any piece of code can be seen as a sequence of tokens

Language keywords
Punctuation symbols
Identifiers
Literals
Numerical values
Comments

Fixed by the programming language *Usually a small set of tokens*

Vocabulary, Code Embeddings and OOV

Open-Vocabulary Problem

Any piece of code can be seen as a sequence of tokens

Language keywords
Punctuation symbols
Identifiers
Literals
Numerical values
Comments

Fixed by the programming language

Usually a small set of tokens

Chosen by developers

Can be of any size

Vocabulary, Code Embeddings and OOV

Open-Vocabulary Problem

Any piece of code can be seen as a sequence of tokens

Language keywords
Punctuation symbols
Identifiers
Literals
Numerical values
Comments

Fixed by the programming language

f tokens

Can be of any size

Difficult to represent and reason about source code

Vocabulary, Code Embeddings and OOV

Handling the Vocabulary Problem

Vocabulary, Code Embeddings and OOV

Handling the Vocabulary Problem

Abstract tokens

Much smaller vocabulary	Looses valuable information

Vocabulary, Code Embeddings and OOV

Abstract Tokens

raw code

```
public Integer getMinElement(List myList) {
   if(myList.size() >= 0) {
      return ListManager.getFirst(myList);
   }
   return 0;
}
```

abstracted code

```
public TYPE_1 METHOD_1 ( TYPE_2 VAR_1 )
{ if ( VAR_1 . METHOD_2 ( ) >= INT_1 )
{ return TYPE_3 . METHOD_3 ( VAR_1 ) ; }
return INT_1 ; }
```

An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation, Tufano et al., 2019

Vocabulary, Code Embeddings and OOV

Handling the Vocabulary Problem

Abstract tokens

Much smaller vocabulary	Looses valuable information

Consider n most frequent tokens only

Covers a large fraction of all tokens	Out-of-vocabulary problem

Vocabulary, Code Embeddings and OOV

Handling the Vocabulary Problem

Abstract tokens

Much smaller vocabulary	Looses valuable information

Consider n most frequent tokens only

Covers a large fraction of all tokens	Out-of-vocabulary problem

Embed tokens into a vector relying on Sub-tokenization

Constant vector size when code corpus grows	Non-trivial to obtain an effective embedding

Vocabulary, Code Embeddings and OOV

From Tokens to Vectors

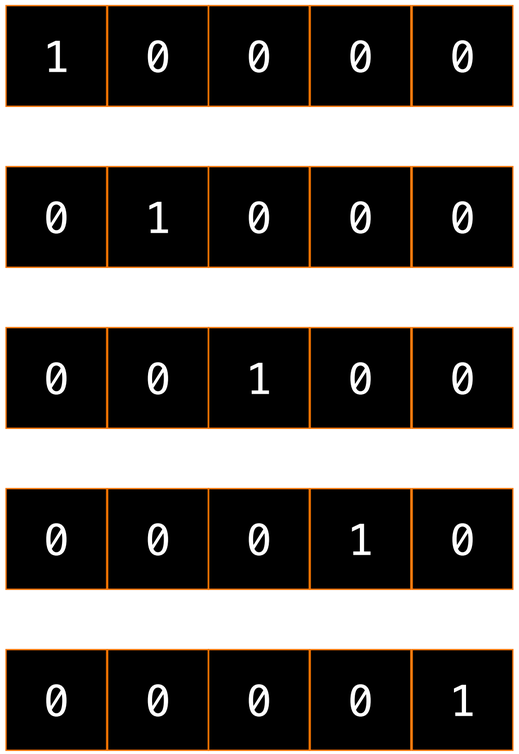
DL models expect vectors as input

How to represent tokens to make these vectors?

Vocabulary, Code Embeddings and OOV

One-hot encoding

```
count += n;
```

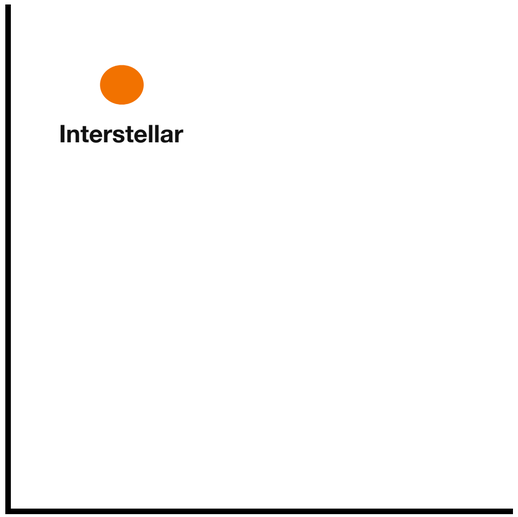


Each token is represented by a vector of size |V|, with V being the vocabulary

Vocabulary, Code Embeddings and OOV

Embedding

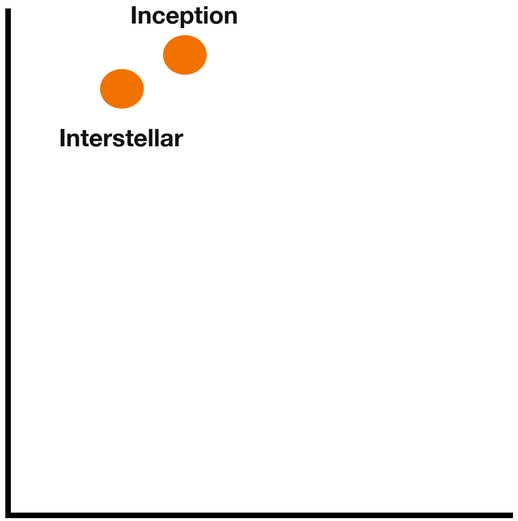
An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



Vocabulary, Code Embeddings and OOV

Embedding

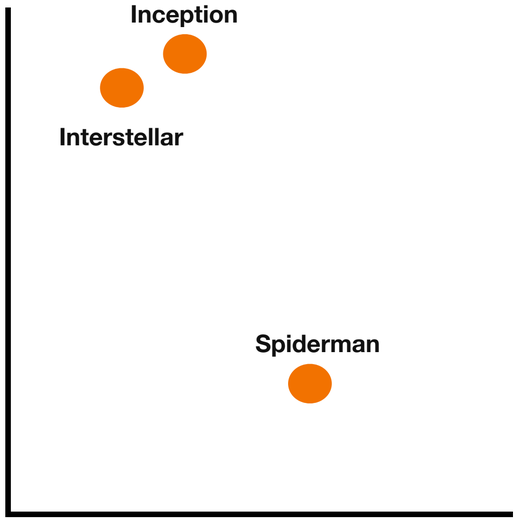
An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



Vocabulary, Code Embeddings and OOV

Embedding

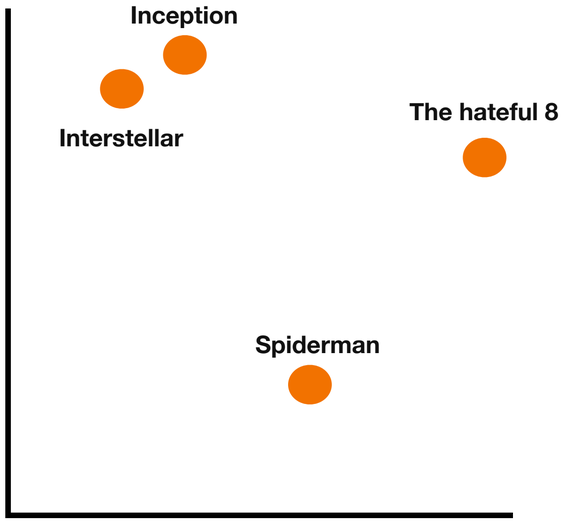
An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



Vocabulary, Code Embeddings and OOV

Embedding

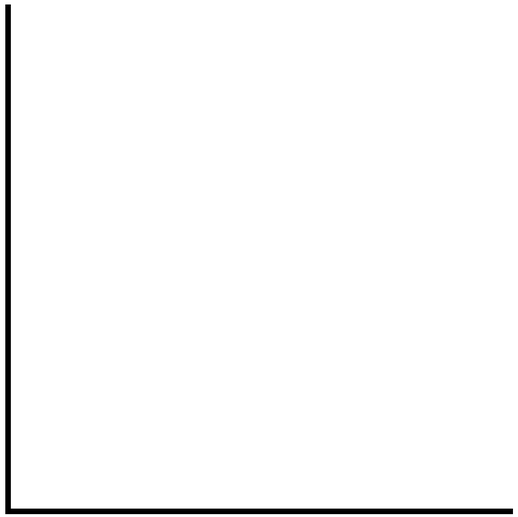
An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



Vocabulary, Code Embeddings and OOV

Embedding

An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



We can represent methods as embeddings

Vocabulary, Code Embeddings and OOV

Embedding

An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. Similar items are mapped closer together based on their relationships or semantic similarity.

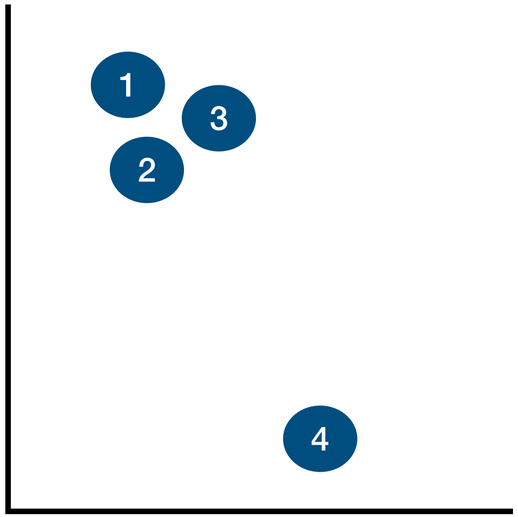
```
1  public String toLowerCase(String input) {
       return input.toLowerCase();
   }

2  public String toUpperCase(String input) {
       return input.toUpperCase();
   }

3  public String trimString(String input) {
       return input.trim();
   }

4  public double computeSquareRoot(double xy) {
       return Math.sqrt(xy);
   }
```

We can represent methods as embeddings

Vocabulary, Code Embeddings and OOV

Embedding

An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



We can represent methods as embeddings

Vocabulary, Code Embeddings and OOV

You are a developer and your boss asks you to improve the codebase.
In particular, she tells you that every time a developer commit the code,
(for the sake of the example let's imagine a method), a script will
automatically check if that method can be a duplicate in the codebase

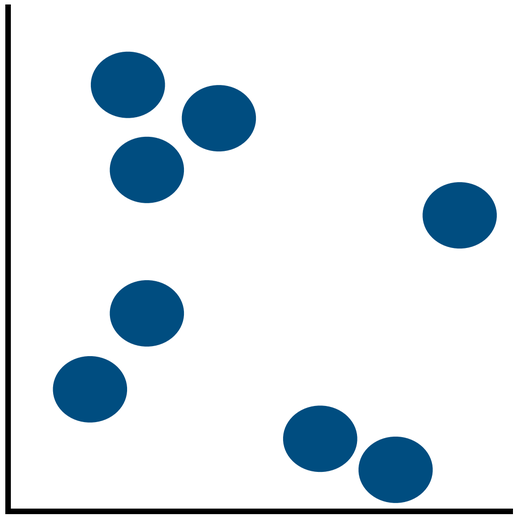
Vocabulary, Code Embeddings and OOV

You are a developer and your boss asks you to improve the codebase.
In particular, she tells you that every time a developer commit the code,
(for the sake of the example let's imagine a method), a script will
automatically check if that method can be a duplicate in the codebase

1. We assume the embedding model (**M**) already exists. Thus, we create N vectors using M and project them onto the vector space where N is the #methods in the codebase.

Vocabulary, Code Embeddings and OOV

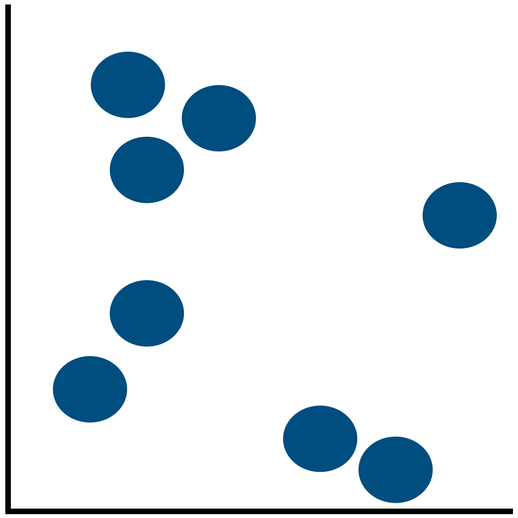
You are a developer and your boss asks you to improve the codebase.
In particular, she tells you that every time a developer commit the code,
(for the sake of the example let's imagine a method), a script will
automatically check if that method can be a duplicate in the codebase



1. We assume the embedding model (**M**) already exists. Thus, we create N vectors using M and project them onto the vector space where N is the #methods in the codebase.

Vocabulary, Code Embeddings and OOV

You are a developer and your boss asks you to improve the codebase.
In particular, she tells you that every time a developer commit the code,
(for the sake of the example let's imagine a method), a script will
automatically check if that method can be a duplicate in the codebase

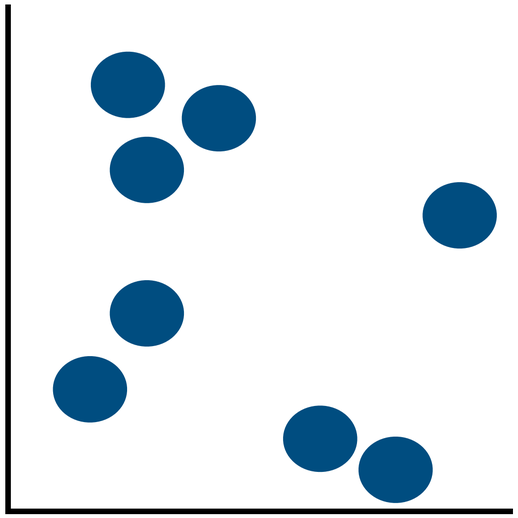


1. We assume the embedding model (**M**) already exists. Thus, we create N vectors using M and project them onto the vector space where N is the #methods in the codebase.

2. We fix a similarity metric (e.g., Cosine similarity) and a threshold, for example 0.9.

Vocabulary, Code Embeddings and OOV

You are a developer and your boss asks you to improve the codebase.
In particular, she tells you that every time a developer commit the code,
(for the sake of the example let's imagine a method), a script will
automatically check if that method can be a duplicate in the codebase



1. We assume the embedding model (**M**) already exists. Thus, we create N vectors using M and project them onto the vector space where N is the #methods in the codebase.

2. We fix a similarity metric (e.g., Cosine similarity) and a threshold, for example 0.9.

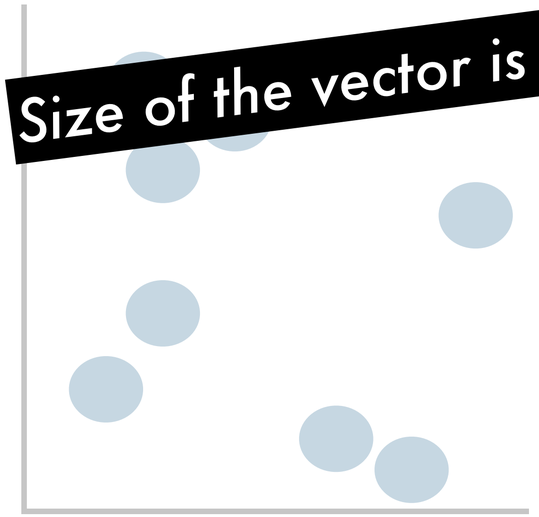
3. Given the method implemented by the developer, we get the embedding using **M** and then, we compute the similarity between the *D_method* (●) and all the methods in the codebase (●)

Vocabulary, Code Embeddings and OOV

You are a developer and your boss asks you to improve the codebase.
In particular, she tells you that every time a developer commit the code, (for the sake of the example let's imagine a method), a script will automatically check if that method can be a duplicate in the codebase

Size of the vector is not the same as the size of the vocabulary, but usually much smaller

1. We assume the embedding … vectors … into the vector space where N … ds in the codebase.



2. We fix a similarity metric (e.g., Cosine similarity) and a threshold, for example 0.9.

3. Given the method implemented by the developer, we get the embedding using **M** and then, we compute the similarity between the *D_method* (●) and all the methods in the codebase (●)

Vocabulary, Code Embeddings and OOV

How to get vector embeddings of tokens?

Word2vec

Popular technique for learning embeddings (original for natural language, but used also on code)

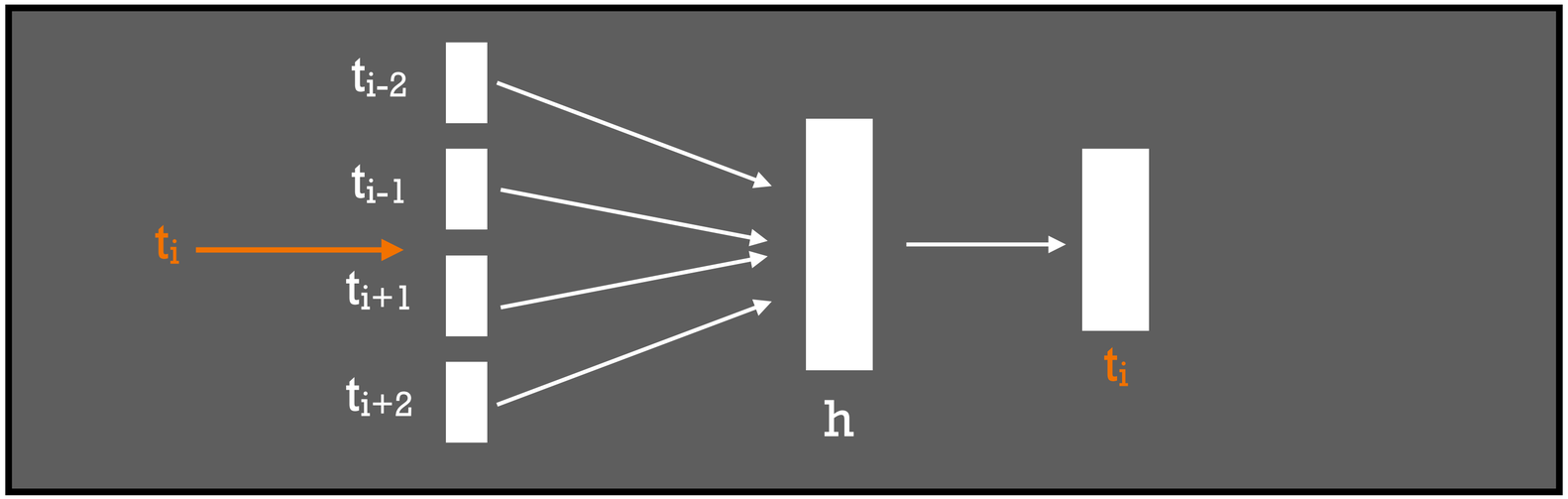
Learning embeddings from the context in which a word occurs (i.e., surrounding words in sentences)

If two words/tokens appear in similar contexts, they are likely to be similar

Vocabulary, Code Embeddings and OOV

How to get vector embeddings of tokens?

Word2vec



Vocabulary, Code Embeddings and OOV

How to get vector embeddings of code tokens?

CodeBERT:
A Pre-Trained Model for Programming and Natural Languages

Zhangyin Feng[1]*, **Daya Guo**[2]*, **Duyu Tang**[3], **Nan Duan**[3], **Xiaocheng Feng**[1]
Ming Gong[4], **Linjun Shou**[4], **Bing Qin**[1], **Ting Liu**[1], **Daxin Jiang**[4], **Ming Zhou**[3]
[1] Research Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China
[2] The School of Data and Computer Science, Sun Yat-sen University, China
[3] Microsoft Research Asia, Beijing, China
[4] Microsoft Search Technology Center Asia, Beijing, China
{zyfeng,xcfeng,qinb,tliu}@ir.hit.edu.cn
guody5@mail2.sysu.edu.cn
{dutang,nanduan,migon,lisho,djiang,mingzhou}@microsoft.com

Abstract

We present CodeBERT, a *bimodal* pre-trained model for programming language (PL) and natural language (NL). CodeBERT learns general-purpose representations that support downstream NL-PL applications such as natural language code search, code documentation generation, etc. We develop CodeBERT with Transformer-based neural architecture, and train it with a hybrid objective function that incorporates the pre-training task of replaced token detection, which is to detect plausible alternatives sampled from generators. This enables us to utilize both "*bimodal*" data of NL-PL pairs and "*unimodal*" data, where the former provides input tokens for model training while the latter helps to learn better generators. We evaluate CodeBERT on two NL-PL applications by fine-tuning model parameters. Results show that CodeBERT achieves state-of-the-art performance on both natural language code search and code documentation generation. Furthermore, to investigate what type of knowledge is learned in CodeBERT, we construct a dataset for NL-PL probing, and evaluate in a zero-shot setting where parameters of pre-trained models are fixed. Results show that CodeBERT performs better than previous pre-trained models on NL-PL probing.[1]

1 Introduction

Large pre-trained models such as ELMo (Peters et al., 2018), GPT (Radford et al., 2018), BERT and RoBERTa (Liu et al., 2019) have dramatically improved the state-of-the-art on a variety of natural language processing (NLP) tasks. These pre-trained models learn effective contextual representations from massive unlabeled text optimized by self-supervised objectives, such as masked language modeling, which predicts the original masked word from an artificially masked input sequence. The success of pre-trained models in NLP also drives a surge of multi-modal pre-trained models, such as ViLBERT (Lu et al., 2019) for language-image and VideoBERT (Sun et al., 2019) for language-video, which are learned from *bimodal* data such as language-image pairs with *bimodal* self-supervised objectives.

In this work, we present CodeBERT, a *bimodal* pre-trained model for natural language (NL) and programming language (PL) like Python, Java, JavaScript, etc. CodeBERT captures the semantic connection between natural language and programming language, and produces general-purpose representations that can broadly support NL-PL understanding tasks (e.g. natural language code search) and generation tasks (e.g. code documentation generation). It is developed with the multi-layer Transformer (Vaswani et al., 2017), which is adopted in a majority of large pre-trained models. In order to make use of both *bimodal* instances of NL-PL pairs and large amount of available *unimodal* codes, we train CodeBERT with a hybrid objective function, including standard masked language modeling (Devlin et al., 2018) and replaced

Vocabulary, Code Embeddings and OOV

How to get vector embeddings of code tokens?

Generated withthe help of GPT-5.2

CodeBERT:
A Pre-Trained Model for Programming and Natural Languages

Zhangyin Feng[1]*, **Daya Guo**[2]*, **Duyu Tang**[3], **Nan Duan**[3], **Xiaocheng Feng**[1]
Ming Gong[4], **Linjun Shou**[4], **Bing Qin**[1], **Ting Liu**[1], **Daxin Jiang**[4], **Ming Zhou**[3]
[1] Research Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China
[2] The School of Data and Computer Science, Sun Yat-sen University, China
[3] Microsoft Research Asia, Beijing, China
[4] Microsoft Search Technology Center Asia, Beijing, China
{zyfeng,xcfeng,qinb,tliu}@ir.hit.edu.cn
guody5@mail2.sysu.edu.cn
{dutang,nanduan,migon,lisho,djiang,mingzhou}@microsoft.com

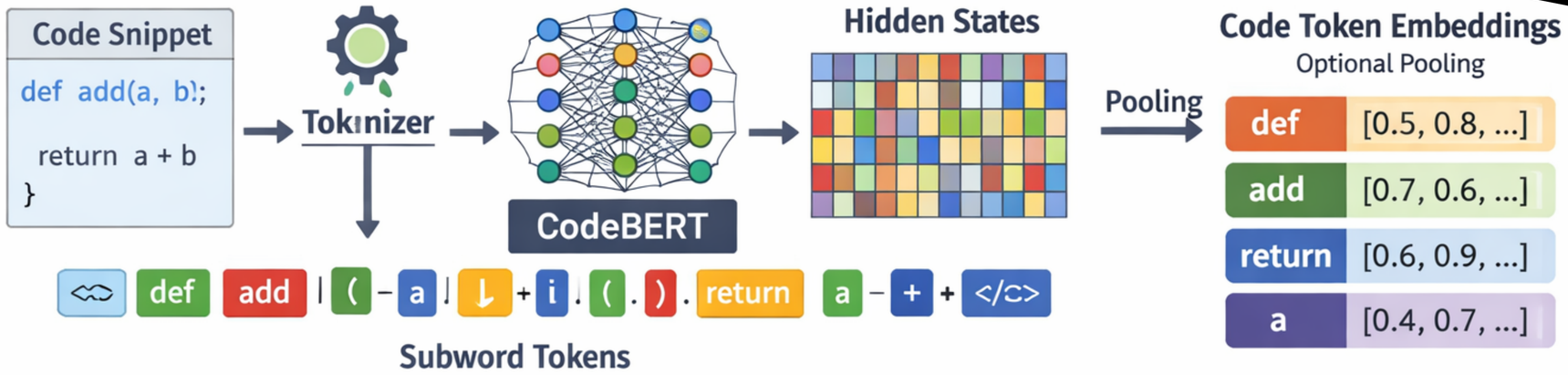
Abstract

We present CodeBERT, a *bimodal* pre-trained model for programming language (PL) and natural language (NL). CodeBERT learns general-purpose representations that support downstream NL-PL applications such as natural language code search, code documentation generation, etc. We develop CodeBERT with Transformer-based neural architecture, and train it with a hybrid objective function that incorporates the pre-training task of replaced token detection, which is to detect plausible alternatives sampled from generators. This enables us to utilize both "*bimodal*" data of NL-PL pairs and "*unimodal*" data, where the former provides input tokens for model training while the latter helps to learn better generators. We evaluate CodeBERT on two NL-PL applications by fine-tuning model parameters. Results show that CodeBERT achieves state-of-the-art performance on both natural language code search and code documentation generation. Furthermore, to investigate what type of knowledge is learned in CodeBERT, we construct a dataset for NL-PL probing, and evaluate in a zero-shot setting where parameters of pre-trained models are fixed. Results show that CodeBERT performs better than previous pre-trained models on NL-PL probing.[1]

1 Introduction

Large pre-trained models such as ELMo (Peters et al., 2018), GPT (Radford et al., 2018), BERT and RoBERTa (Liu et al., 2019) have dramatically improved the state-of-the-art on a variety of natural language processing (NLP) tasks. These pre-trained models learn effective contextual representations from massive unlabeled text optimized by self-supervised objectives, such as masked language modeling, which predicts the original masked word from an artificially masked input sequence. The success of pre-trained models in NLP also drives a surge of multi-modal pre-trained models, such as ViLBERT (Lu et al., 2019) for language-image and VideoBERT (Sun et al., 2019) for language-video, which are learned from *bimodal* data such as language-image pairs with *bimodal* self-supervised objectives.

In this work, we present CodeBERT, a *bimodal* pre-trained model for natural language (NL) and programming language (PL) like Python, Java, JavaScript, etc. CodeBERT captures the semantic connection between natural language and programming language, and produces general-purpose representations that can broadly support NL-PL understanding tasks (e.g. natural language code search) and generation tasks (e.g. code documentation generation). It is developed with the multi-layer Transformer (Vaswani et al., 2017), which is adopted in a majority of large pre-trained models. In order to make use of both *bimodal* instances of NL-PL pairs and large amount of available *unimodal* codes, we train CodeBERT with a hybrid objective function, including standard masked language modeling (Devlin et al., 2018) and replaced



https://arxiv.org/pdf/2002.08155

antoniomastropaolo.com

aura-se-lab.github.io

Vocabulary, Code Embeddings and OOV

Embedding of Sub-tokens

Learn embeddings of subtokens

Main intuition

Previously unseen tokens that would in a different conext result in out-of-vocabulary, are likely to be composable using sub-tokens

For example, an identifier setHeight can be composed by its sub-terms set and Height

Popular sub-tokens are identified and learned as embeddings

Vocabulary, Code Embeddings and OOV

Embedding of Sub-tokens

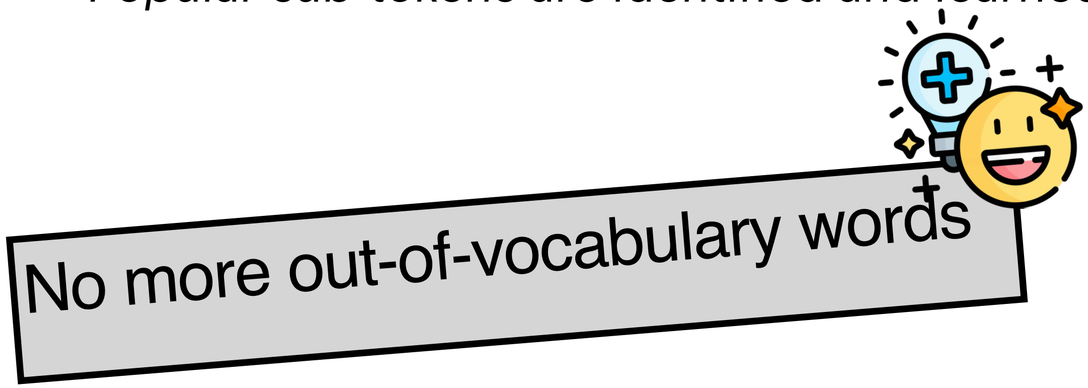
Learn embeddings of subtokens

Main intuition

Previously unseen tokens that would in a different conext result in out-of-vocabulary, are likely to be composable using sub-tokens

For example, an identifier setHeight can be composed by its sub-terms set and Height

Popular sub-tokens are identified and learned as embeddings



- BPE (Byte-Pair-Encoding)
- WordPiece
- Unigram (SentencePiece)

Vocabulary, Code Embeddings and OOV

```
from transformers import RobertaTokenizer, T5ForConditionalGeneration

tokenizer = RobertaTokenizer.from_pretrained('Salesforce/codet5-base')
model = T5ForConditionalGeneration.from_pretrained('Salesforce/codet5-base')

text = "def greet(user): print('hello world')"
```

BPE

Vocabulary, Code Embeddings and OOV

BPE

```
from transformers import RobertaTokenizer, T5ForConditionalGeneration

tokenizer = RobertaTokenizer.from_pretrained('Salesforce/codet5-base')
model = T5ForConditionalGeneration.from_pretrained('Salesforce/codet5-base')

text = "def greet(user): print('hello world')"
tokens = tokenizer.tokenize(text)
encodedTokens = tokenizer.encode(tokens)
print(tokens)
print(encodedTokens)
```

Vocabulary, Code Embeddings and OOV

BPE

```
from transformers import RobertaTokenizer, T5ForConditionalGeneration

tokenizer = RobertaTokenizer.from_pretrained('Salesforce/codet5-base')
model = T5ForConditionalGeneration.from_pretrained('Salesforce/codet5-base')

text = "def greet(user): print('hello world')"
tokens = tokenizer.tokenize(text)
encodedTokens = tokenizer.encode(tokens)
print(tokens)
print(encodedTokens)
```

```
['def', 'Ġgre', 'et', '(', 'user', '):', 'Ġprint', "('", 'hello', 'Ġworld', "')"]
[1, 536, 5174, 278, 12, 1355, 4672, 1172, 2668, 23711, 9117, 6134, 2]
```

Vocabulary, Code Embeddings and OOV

BPE

```
from transformers import RobertaTokenizer, T5ForConditionalGeneration

tokenizer = RobertaTokenizer.from_pretrained('Salesforce/codet5-base')
model = T5ForConditionalGeneration.from_pretrained('Salesforce/codet5-base')

text = "def greet(user): print('hello world')"
tokens = tokenizer.tokenize(text)
encodedTokens = tokenizer.encode(tokens)
print(tokens)
print(encodedTokens)

['def', 'Ġgre', 'et', '(', 'user', '):', 'Ġprint', "('", 'hello', 'Ġworld', "')"]
[1, 536, 5174, 278, 12, 1355, 4672, 1172, 2668, 23711, 9117, 6134, 2]
<s>                                                                  </s>
```

Vocabulary, Code Embeddings and OOV

Deep Learning for SD Automation: Open Vocabulary Problem & Embeddings

Handling the Vocabulary Problem

Abstract tokens

Much smaller vocabulary	Looses valuable information

Consider n most frequent tokens only

Covers a large fraction of all tokens	Out-of-vocabulary problem

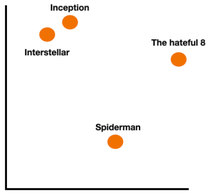
Embed tokens into a vector relying on Sub-tokenization

Constant vector size when code corpus grows	Non-trivial to obtain an effective embedding

Deep Learning for SD Automation: Open Vocabulary Problem & Embeddings

Embedding

An ***embedding*** is a numerical representation of data—such as words, sentences, code, or images, projected onto a continuous vector space. The peculiarity is that, similar entities are mapped closer together based on their meaning or contextual similarity.



Deep Learning for SD Automation: Open Vocabulary Problem & Embeddings

Embedding of Sub-tokens

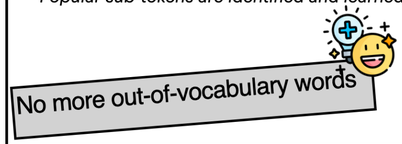
Learn embeddings of subtokens

Main intuition

Previously unseen tokens are likely to be composable using sub-tokens, learn embeddings for sub-tokens!

For example, an identifier setHeight can be composed by its sub-terms set and Height

Popular sub-tokens are identified and learned as embeddings



- BPE (Byte-Pair-Encoding)
- WordPiece
- Unigram (SentencePiece)