Analyzing Software using Deep Learning

Token Vocabulary and Code Embeddings

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2022

Overview

- **Token vocabulary problem**
- **Pre-trained token embeddings**
- **Joint embedding space for NL & PL**

Recommended papers:

- ”Distributed representations of words and phrases and their compositionality”, NIPS, 2013
- ”Big Code != Big Vocabulary - Open-Vocabulary Models for Source Code”, ICSE, 2020
- ”Deep Code Search”, ICSE, 2018

Tokens: Building Blocks of Code

- Source code = Sequence of tokens
- Reasoning about large code snippets: Need to reason about tokens first

```
// From Angular.js
browserSingleton.startPoller(100,
    function(delay, fn) {
        setTimeout(delay, fn);
    });
```

Kinds of Tokens

Two categories of tokens

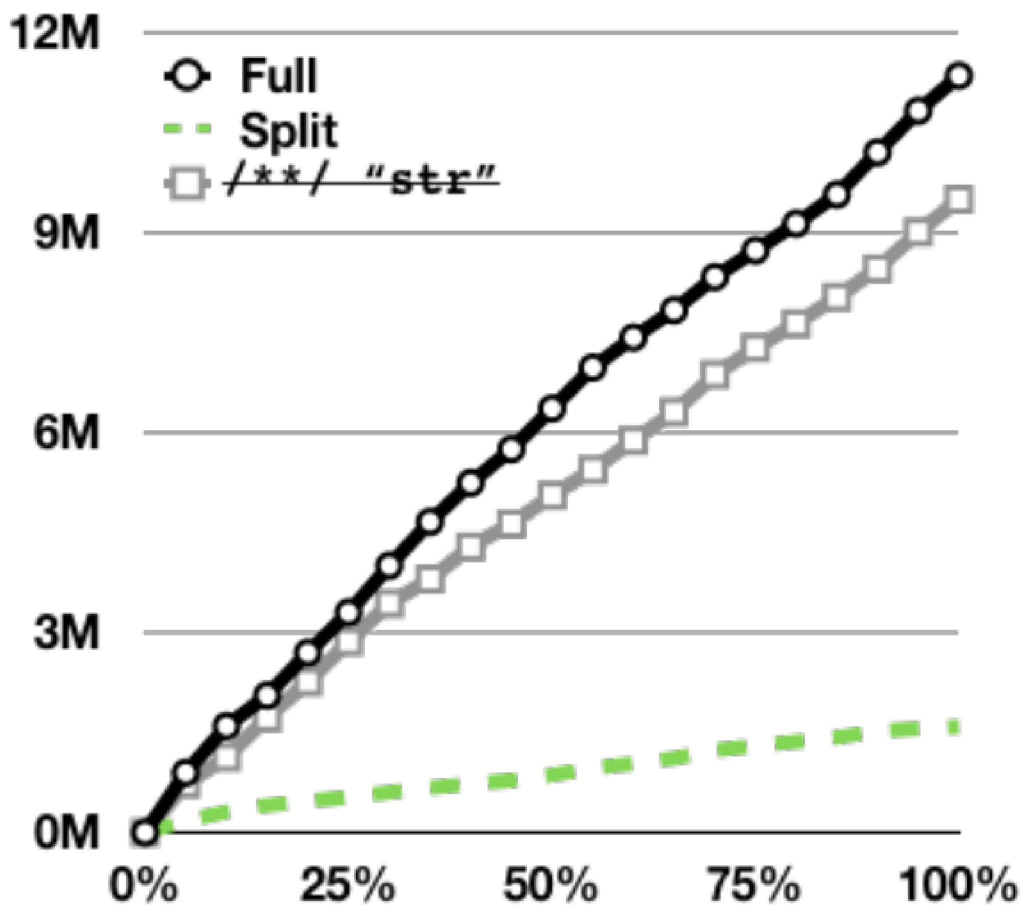
- Fixed by programming language
  - Operators, parentheses, keywords, etc.
- Chosen by developers
  - Identifiers, literals

Vocabulary Problem

- **Large code corpus:**
  Huge number of tokens
- **Difficult to represent and reason about**
- **Relevant for**
  - Models that take code as an input
  - Models that produce code as an output

Vocabulary Problem (2)

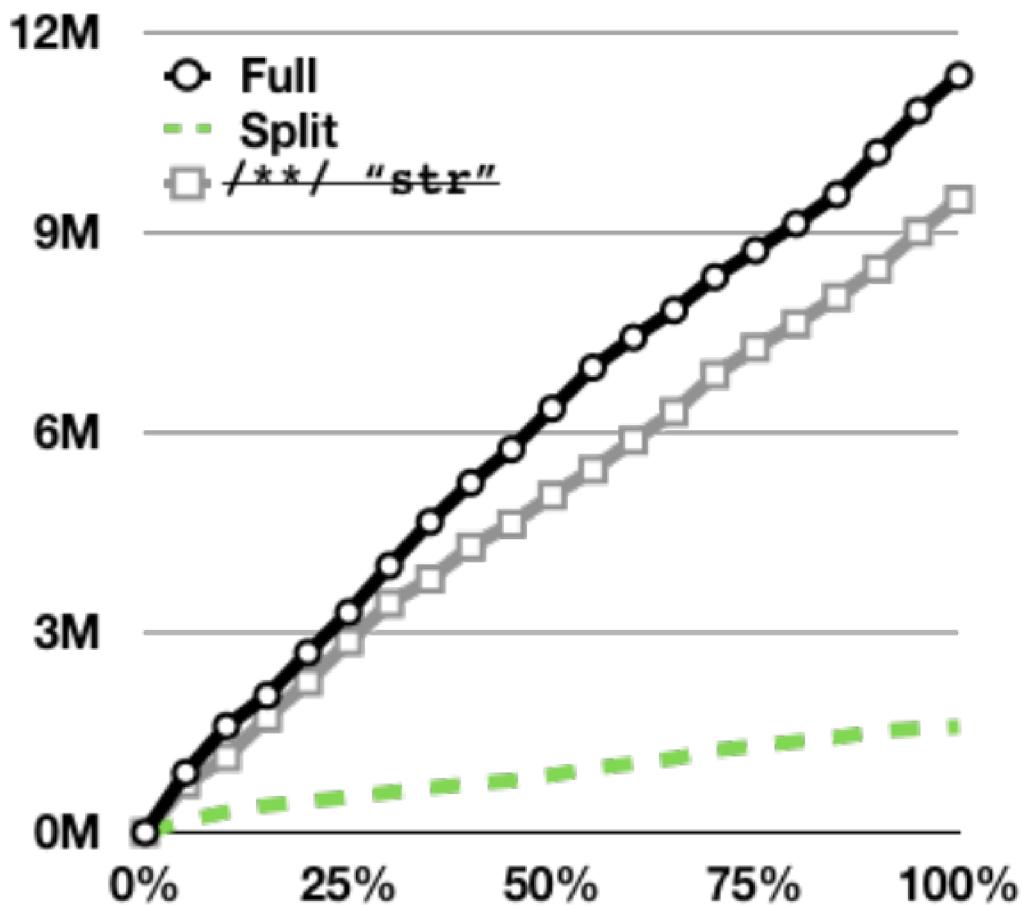
Size of vocabulary for 14k projects



"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)

Vocabulary Problem (2)

Size of vocabulary for 14k projects

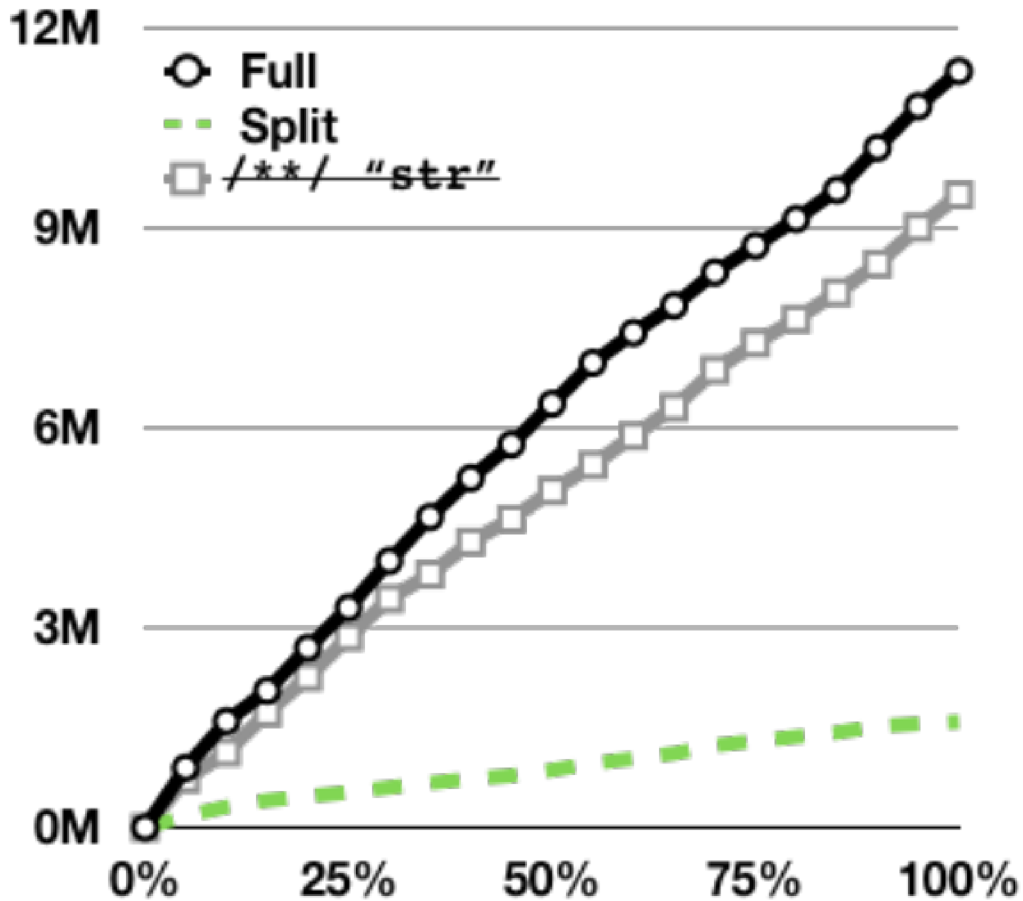


Almost 12 million tokens!

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)

Vocabulary Problem (2)

Size of vocabulary for 14k projects

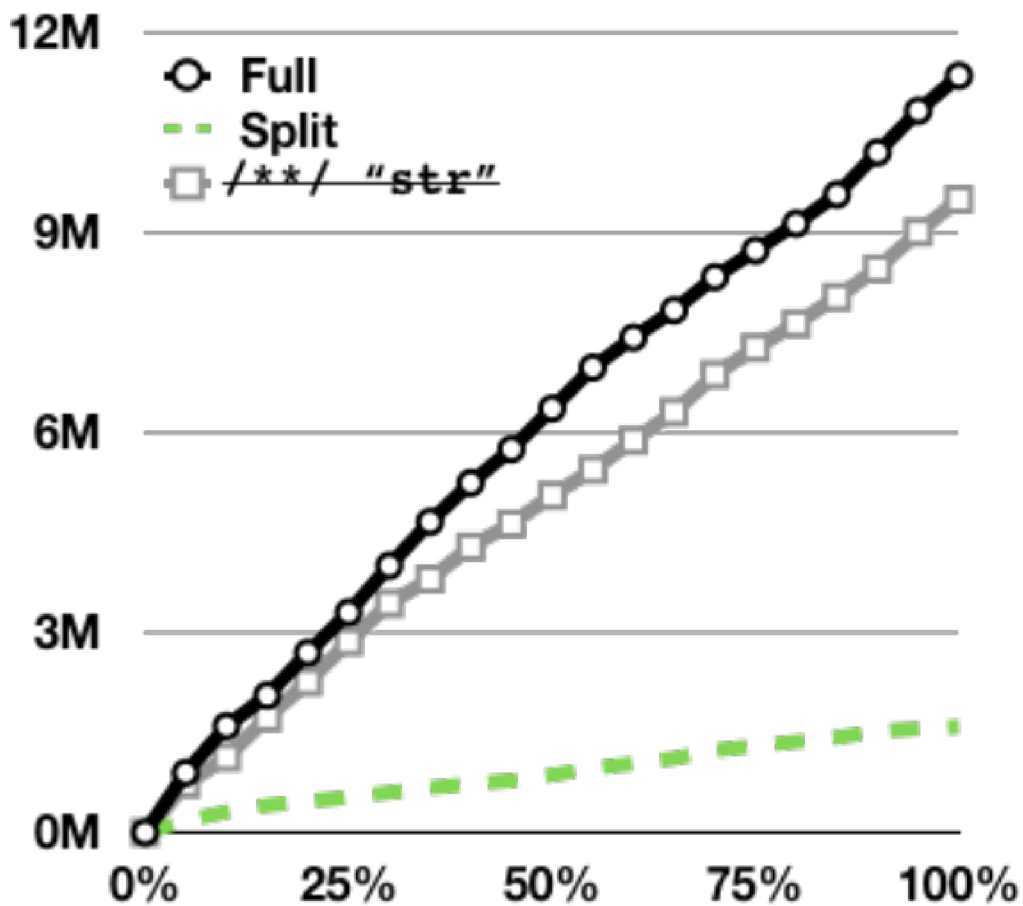


Replacing comments and strings with placeholders

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)

Vocabulary Problem (2)

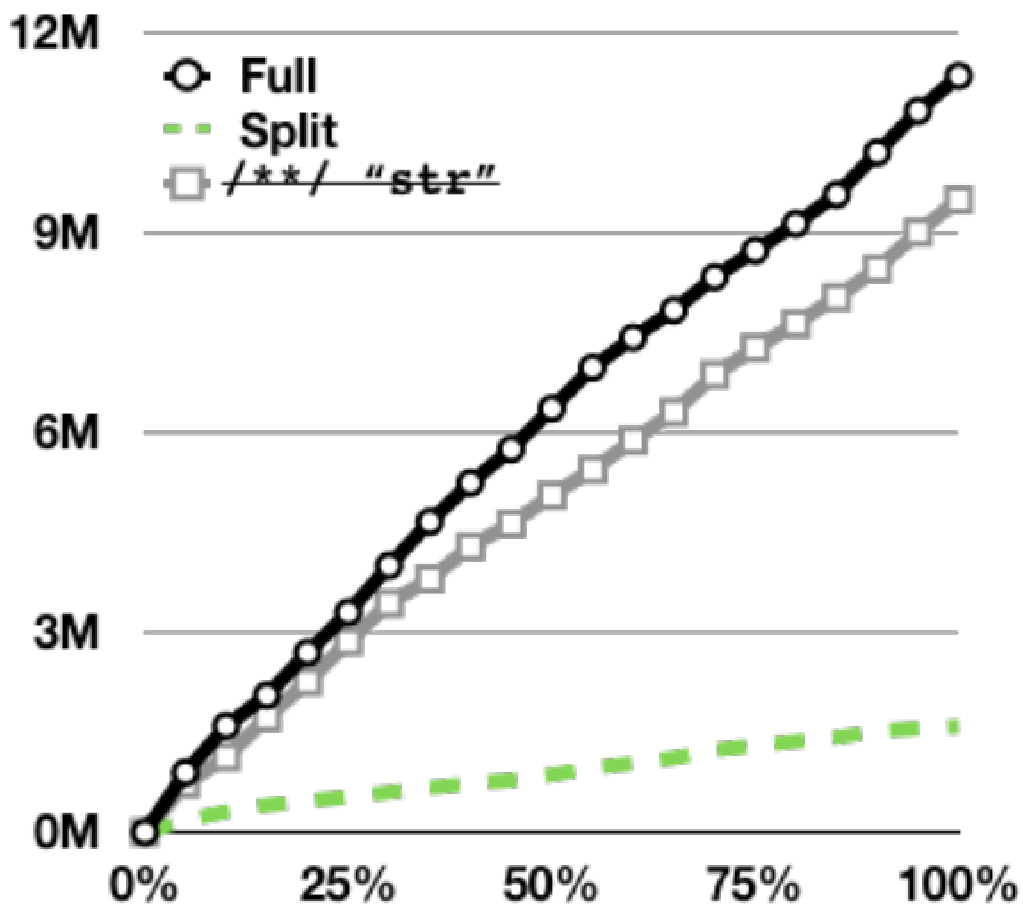
Size of vocabulary for 14k projects



Split identifiers based on camelCase and snake_case

Vocabulary Problem (2)

Size of vocabulary for 14k projects



For all ways of modeling the vocabulary: Linear growth when new projects are added

"Modeling Vocabulary for Big Code Machine Learning" (Babii et al., 2019)

Handling the Vocabulary Problem

Abstract tokens

- Much smaller vocabulary
- Looses valuable informa-tion

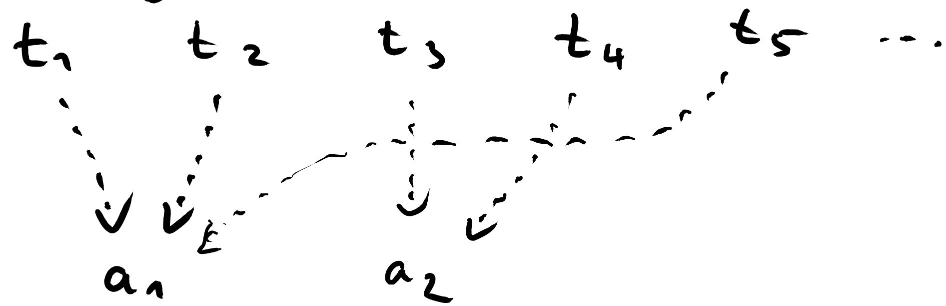
Consider N most frequent tokens only

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

Embed tokens into a vector space

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

Abstracting Token



---→ abstraction

Result:

a_1 a_2 a_2 a_2 a_1 ...

Abstraction by kind of token

```
    operator    operator
if ( file != null ) {
keyword   identifier   literal
    line = file . read ()
}
```

keyword operator identifier operator literal ...

OR:

```
if ( identifier != null ) {
    identifier = identifier . identifier ()
}
```

Consistent Renaming

```
if ( file != null ) {
    line = file.read()
}
```

```
if ( id1 != null ) {
    id2 = id1.id3 ()
}
```

Keeping Top-N Tokens

- **Observation: Vocabulary has a "long-tail" distribution**
  - Few tokens occur frequently
  - Many other tokens occur infrequently
- **Keep only N most frequent tokens**
- **Represent others as special "unknown" token**

Keeping Top-N Tokens (2)

Top-N approach on ≈ 100k JavaScript files

$\lvert\mathcal{V}_{out}\rvert$	Percentage of unique names covered	Percentage of names covered
1,000	0.40	63.19
5,000	1.99	75.07
10,000	3.97	79.48
20,000	7.95	83.82
30,000	11.92	86.38
40,000	15.89	88.16
50,000	19.87	89.56
60,000	**23.84**	**90.74**
70,000	27.81	91.62
80,000	31.79	92.41
90,000	35.76	93.19
100,000	39.74	93.98

”Context2Name: A Deep Learning-Based Approach to Infer Natural Variable Names from Usage Contexts” (Bavishi et al., 2018)

Handling the Vocabulary Problem

Abstract tokens

- Much smaller vocabulary
- Looses valuable informa-tion

Consider N most frequent tokens only

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

Embed tokens into a vector space

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

Overview

- **Token Vocabulary problem**
- **Pre-trained token embeddings** ←
- **Joint embedding space for NL & PL**

Recommended papers:

- ”Distributed representations of words and phrases and their compositionality”, NIPS, 2013
- ”Big Code != Big Vocabulary - Open-Vocabulary Models for Source Code”, ICSE, 2020
- ”Deep Code Search”, ICSE, 2018

From Tokens to Vectors

- **Given a vocabulary of tokens:**
  How to represent a token as a vector?
- **Neural models require vectors as inputs**
- **Need a mapping** $E : V \to \mathbb{R}^k$
  - V .. vocabulary
  - k .. length of vector representation

One-hot Encoding

- **Give each $t \in V$ a unique index**
- **Vector is all zeros, except for the index of t, which is one**

$$E(t)_i = \begin{cases} 1 & \textbf{if index of } t \textbf{ is } i \\ 0 & \textbf{otherwise} \end{cases}$$

- **Length k of vectors equals vocabulary size $|V|$**

Example

$V = \{ \text{if}, (,), \text{id} \}$

$E(\text{'if'}) = [1, 0, 0, 0]$

$E(\text{'('}) = [0, 1, 0, 0]$

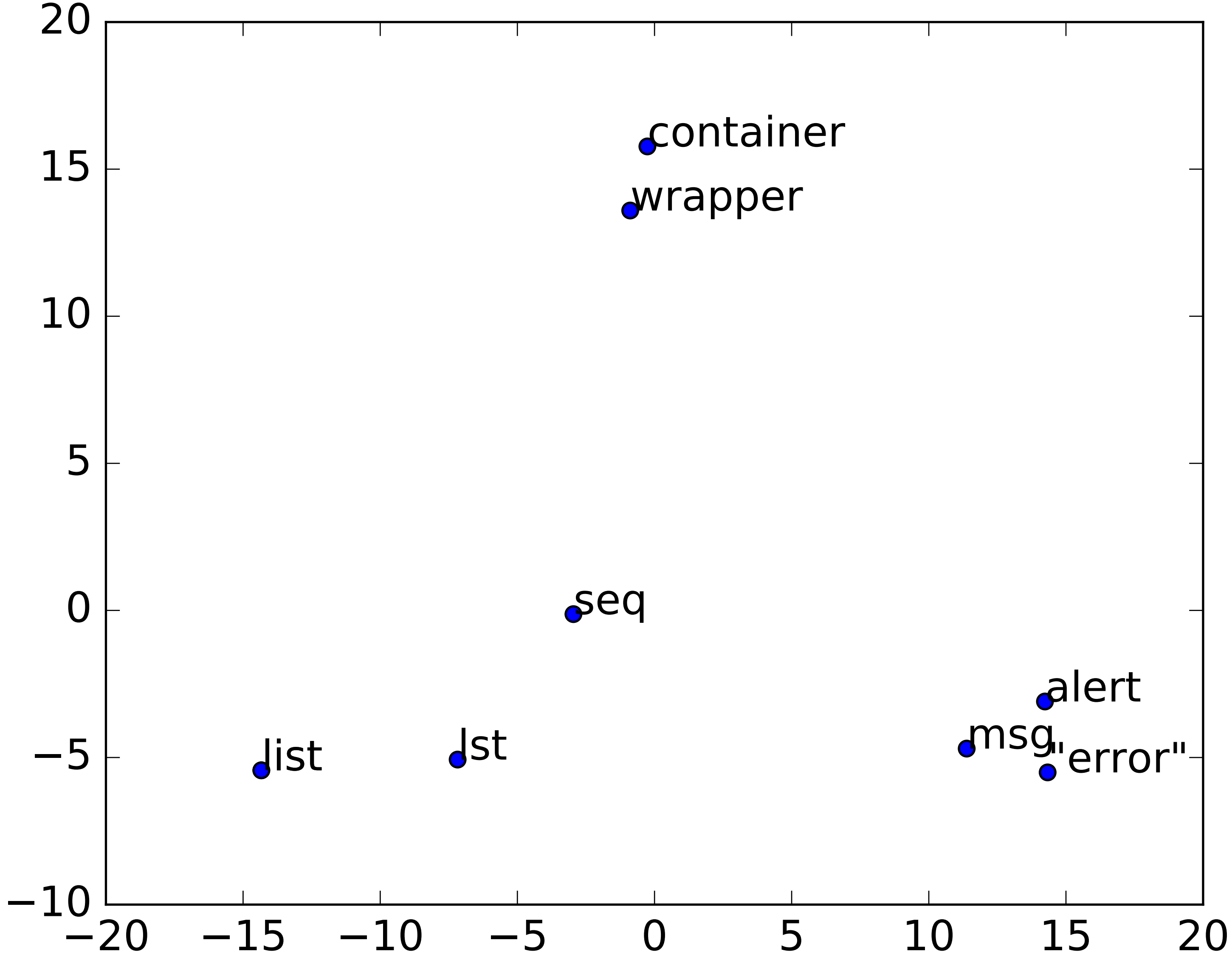
$E(\text{')'}) = [0, 0, 1, 0]$

$E(\text{'id'}) = [0, 0, 0, 1]$

Token Embeddings

- **Map tokens to a vector space**
  - Semantically similar tokens have a similar vector representation
  - Size k of vectors is much smaller than $|V|$

Example: Token Embeddings



End-to-End vs. Pre-trained

How to get vector embeddings of tokens?

- Option 1: Learn embedding function E jointly with the rest of the model
  - Embeddings fit the ultimate application
- Option 2: Pre-train a separate embedding model E
  - Powerful model designed just for this purpose

End-to-End vs. Pre-trained

How to get vector embeddings of tokens?

- Option 1: Learn embedding function E jointly with the rest of the model
  - Embeddings fit the ultimate application
- Option 2: Pre-train a separate embedding model E
  - Powerful model designed just for this purpose

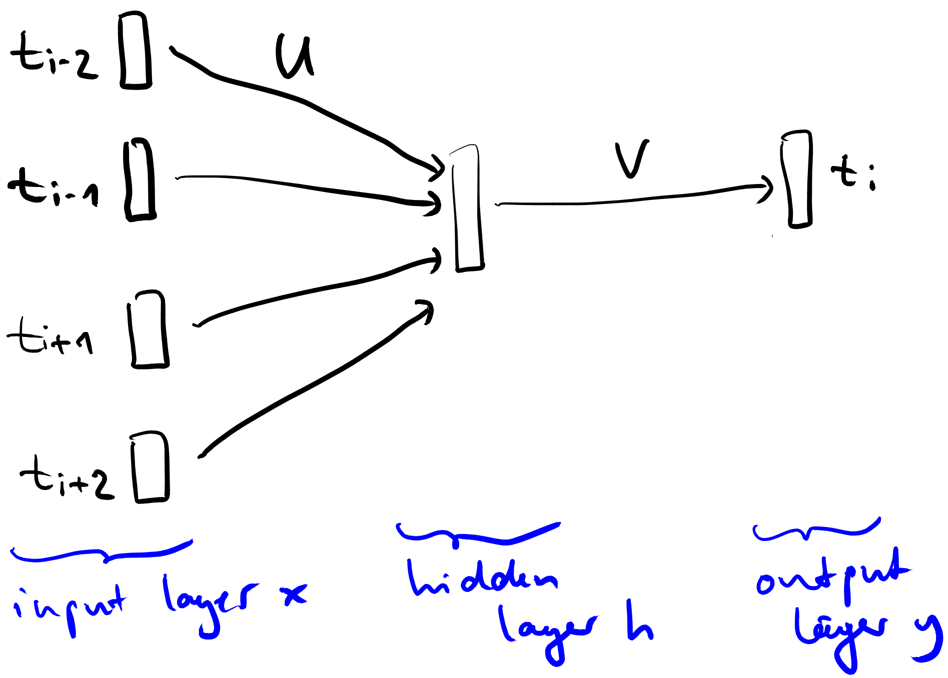
Focus for rest of this lecture

Word2vec

- **Popular technique for learning embeddings (originally, for natural languages)**
- **Learn embeddings from context in which a word occurs**
  - ”You shall know a word by the company it keeps”
  - Context: Surrounding words in sentences

Variant 1: Continuous Bag of Words (CBOW)

Predict token from context



$$h = \frac{1}{k} \cdot U \cdot \Big(\sum_j t_j \Big)$$

j: $i-\frac{k}{2}, \ldots, i+\frac{k}{2}$

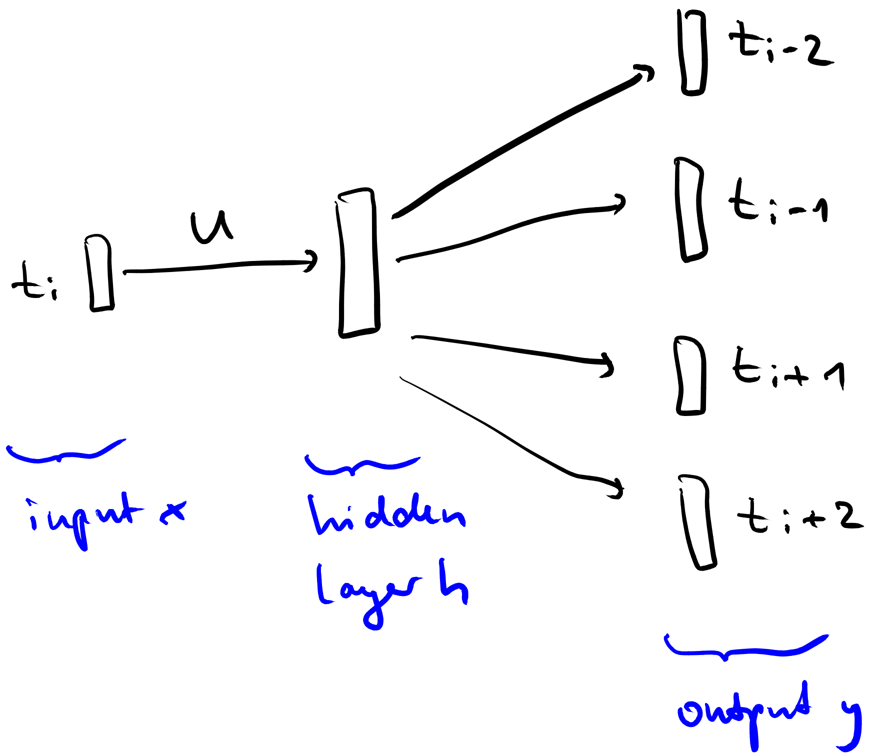
(without i)

$$y = \text{softmax}(V \cdot h)$$

Context size $k = 4$

Variant 2: Skip-gram

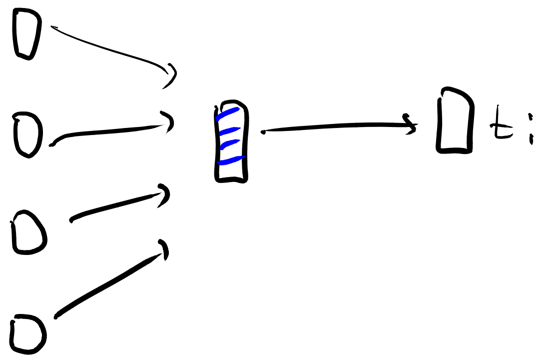
Predict context from token

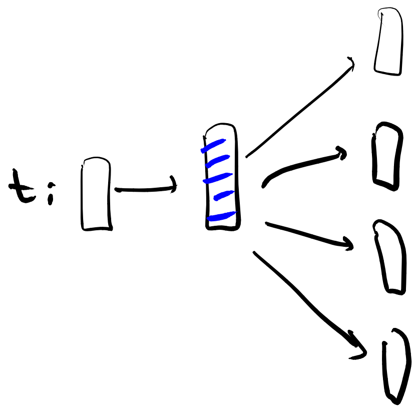


$$h = U \cdot x$$

$$y = \text{softmax}(V \cdot h)$$

Getting the embedding





Once model is good at its task:

Use hidden layer as embedding for t_i

Out-of-Vocabulary Problem

- **During training: Finite set of tokens**
- **During prediction: New tokens may appear**
  - Represented as special “unknown” token
  - Loss of valuable information

Embeddings of Subtokens

- **Idea to address out-of-vocabulary problem:**
  - Learn embedding of subtokens
  - Previously unseen tokens are likely to composable of the subtokens
- **Example**
  - `setHeight` decomposed into subtokens `set` and `Height`

FastText

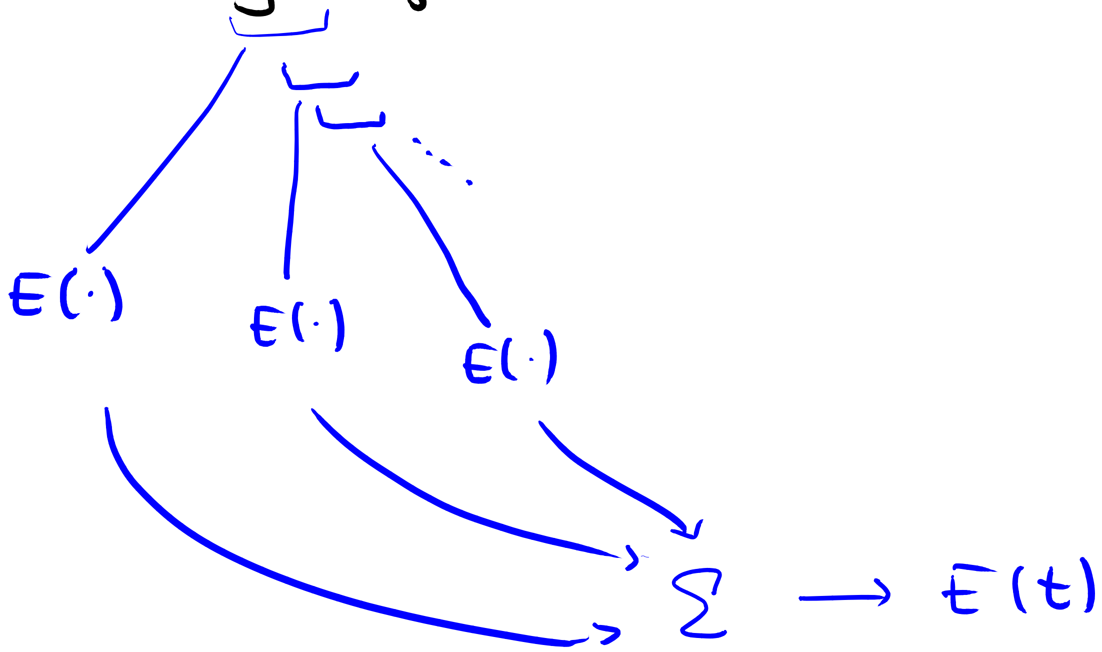
- **Decompose tokens into their character n-grams**
  - n-gram: n consecutive characters
- **Learn embedding for each n-gram**
  - Using Word2vec-like skip-gram model
- $E(t) = \sum_{s \in \text{n-gram sub-tokens of } t} E(s)$

Example

token t: getHeight

$n = 3$

i.e. 3-grams



Byte Pair Encoding (BPE)

Compute subtokens from data

- Start with one subtoken per character
- Repeat:
  - Find pair of current subtokens that most frequently appear consecutively
  - Merge pair into a new subtoken
- Result: Ordered list L of merge operations
- Represent a token t by
  - splitting t into characters and
  - merging the characters into subtokens using operations as ordered in L

Handling the Vocabulary Problem

Abstract tokens

- Much smaller vocabulary
- Looses valuable informa-tion

Consider N most frequent tokens only

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

Embed tokens into a vector space

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

Overview

- **Token Vocabulary problem**
- **Pre-trained token embeddings**
- **Joint embedding space for NL & PL**

Recommended papers:

- ”Distributed representations of words and phrases and their compositionality”, NIPS, 2013
- ”Big Code != Big Vocabulary - Open-Vocabulary Models for Source Code”, ICSE, 2020
- ”Deep Code Search”, ICSE, 2018

NL & PL Information

- **Software is not just code**
- **Many natural language artifacts**
- **Applications of reasoning about both PL and NL information**
  - NL-to-code search
  - Predict or check comments
  - Learn from API documentation

Joint Embedding Space

- **How to reason about PL tokens and NL words together?**
- **Idea: Learn embedding that maps both PL tokens and NL words into a single vector space**
  - Goal: Related tokens and words are close-by
  - Model learns how to related PL and NL information to each other

Deep Code Search

NL query → Learning-based Code Search Engine → Code snippet

Deep Code Search

NL query → **Learning-based Code Search Engine** → **Code snippet**

"read an object from an xml"

Deep Code Search

NL query → **Learning-based Code Search Engine** → **Code snippet**

“read an object from an xml”

```
public static < S > S deserialize(Class c, File xml) {
    try {
        JAXBContext context = JAXBContext.newInstance(c);
        Unmarshaller unmarshaller = context.createUnmarshaller();
        S deserialized = (S) unmarshaller.unmarshal(xml);
        return deserialized;
    } catch (JAXBException ex) {
        log.error("Error-deserializing-object-from-XML", ex);
        return null;
    }
}
```

Deep Code Search

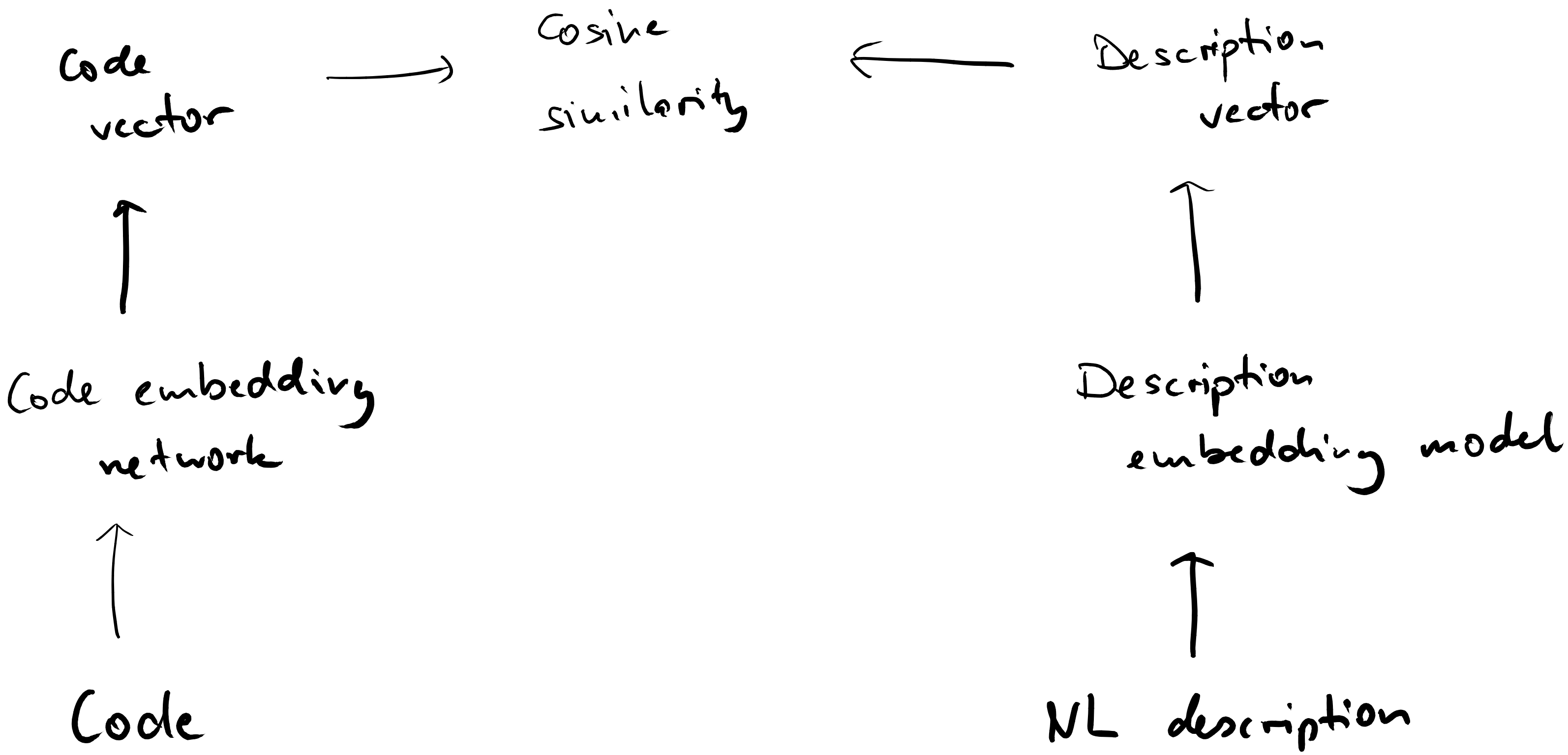
NL query → **Learning-based Code Search Engine** → **Code snippet**

“read an object from an xml”

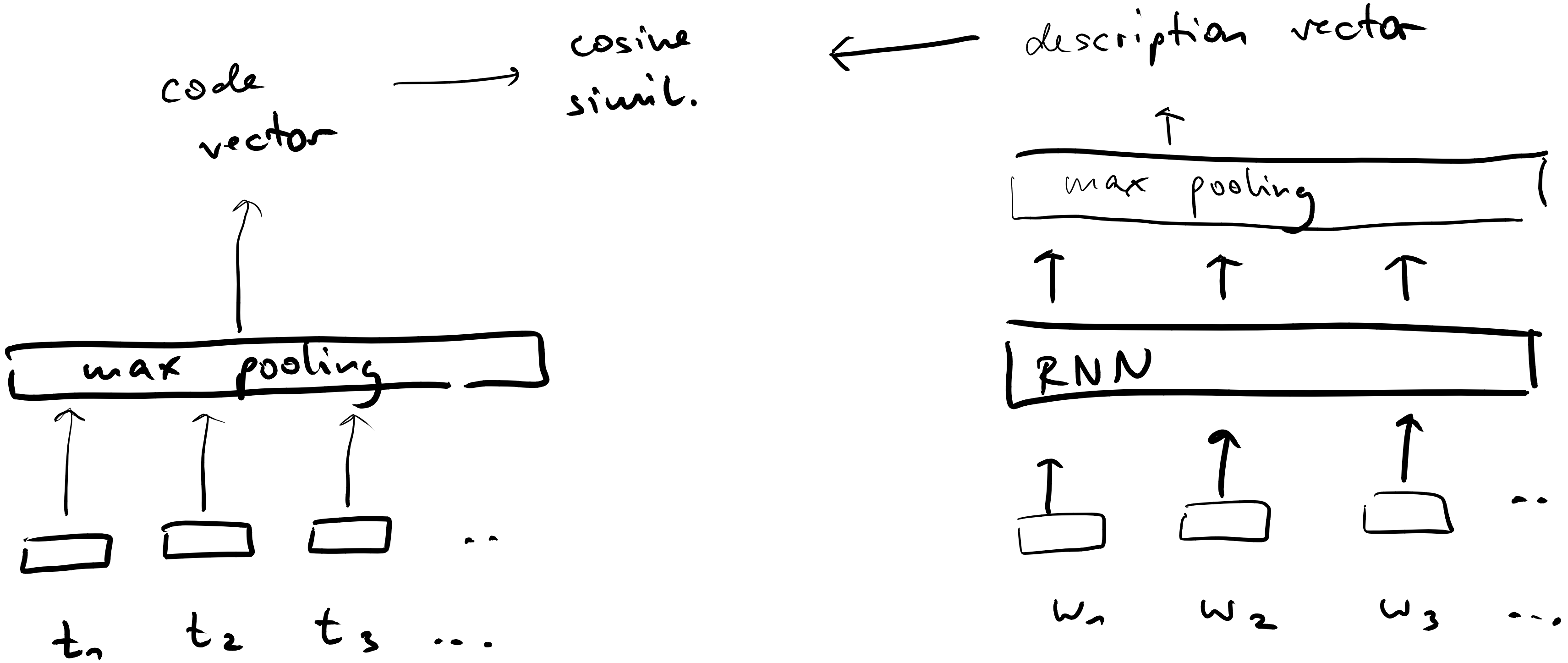
Note: Similar terms, but not the same

```
public static < S > S deserialize(Class c, File xml) {
    try {
        JAXBContext context = JAXBContext.newInstance(c);
        Unmarshaller unmarshaller = context.createUnmarshaller();
        S deserialized = (S) unmarshaller.unmarshal(xml);
        return deserialized;
    } catch (JAXBException ex) {
        log.error("Error-deserializing-object-from-XML", ex);
        return null;
    }
}
```

Overview


Code vector
Cosine similarity
Description vector
Code embedding network
Description embedding model
Code
NL description

Neural model



Code : set of tokens

Description : sequence of words

Training the Model

- **Train with pairs of code snippet c and NL query d**
  - Matching pairs (c, d_+)
  - Non-matching pairs (c, d_-)
- **Loss function:**

$$\mathcal{L}(\theta) = \sum_{<C, D+, D-> \in P} max(0, \epsilon - cos(\boldsymbol{c}, \boldsymbol{d}+) + cos(\boldsymbol{c}, \boldsymbol{d}-))$$

"Deep Code Search" (Gu et al., 2018)

Results

- **Model trained on 18 million Java methods and their comments (as a surrogate for NL queries)**
- **Evaluation with 50 questions from stackoverflow.com**
  - Correct code snippet predicted at position 1 or 2 for most queries