Analyzing Software using Deep Learning

Token Vocabulary and Code Embeddings (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

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:

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

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

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

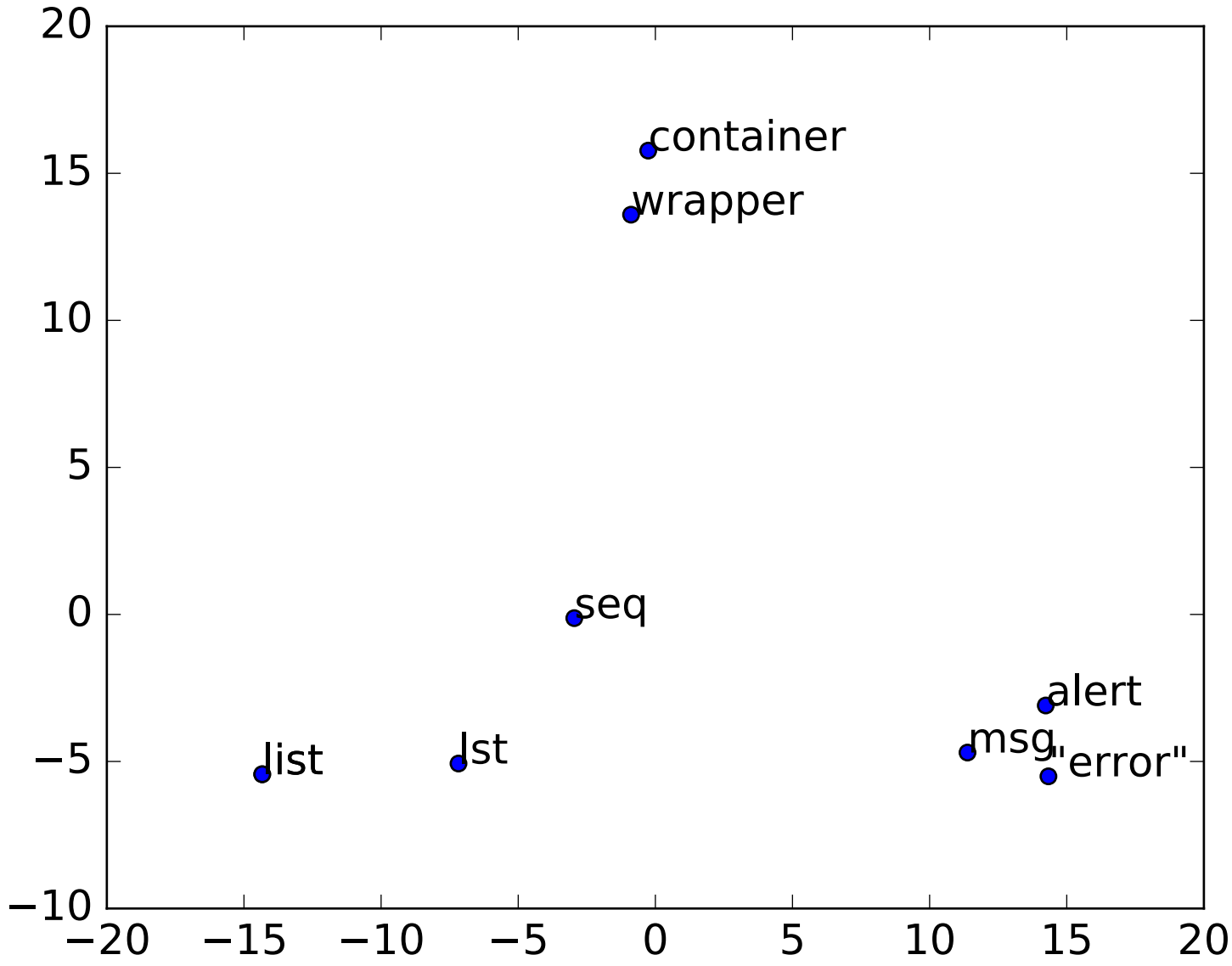
$E(\text{')'}) = [0, 0, 1, 0]$

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

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

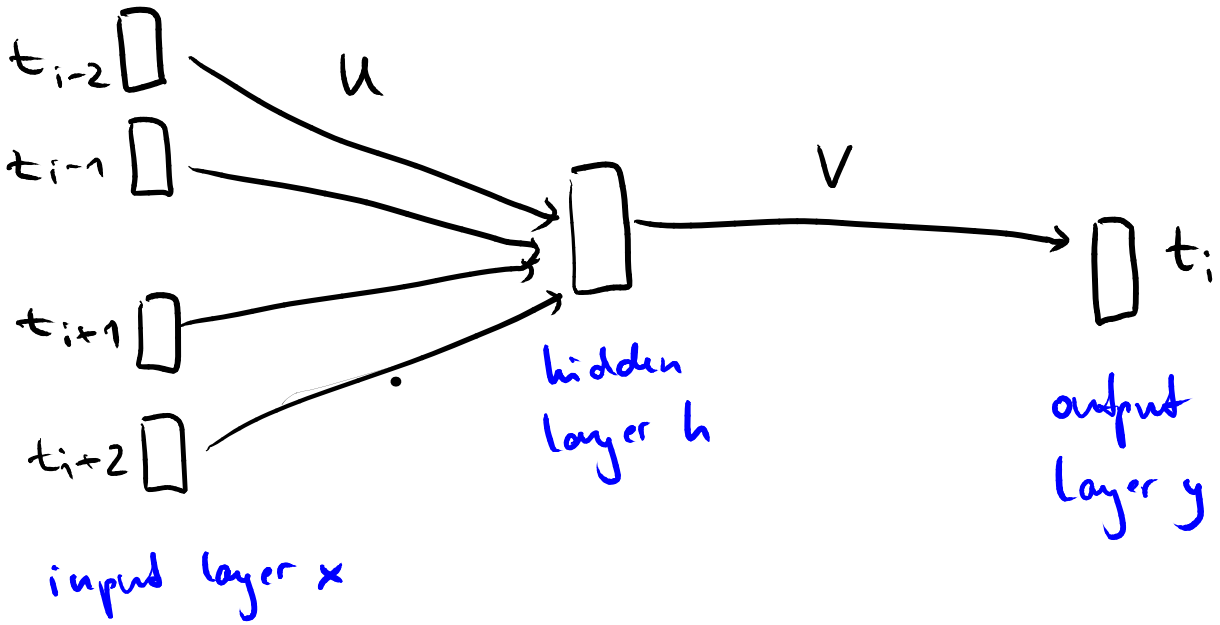
- **Popular technique for learning embeddings (originally, for natural languages)**
- **Learn embeddings from context in which a word occurs**
  - ”You shall know a word by the company it keeps”
  - Context: Surrounding words in sentences

Variant 1: Continuous Bag of Words (CBOW)

Predict token from context

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

$i - \frac{k}{2}, \ldots, i + \frac{k}{2}$ (without i)

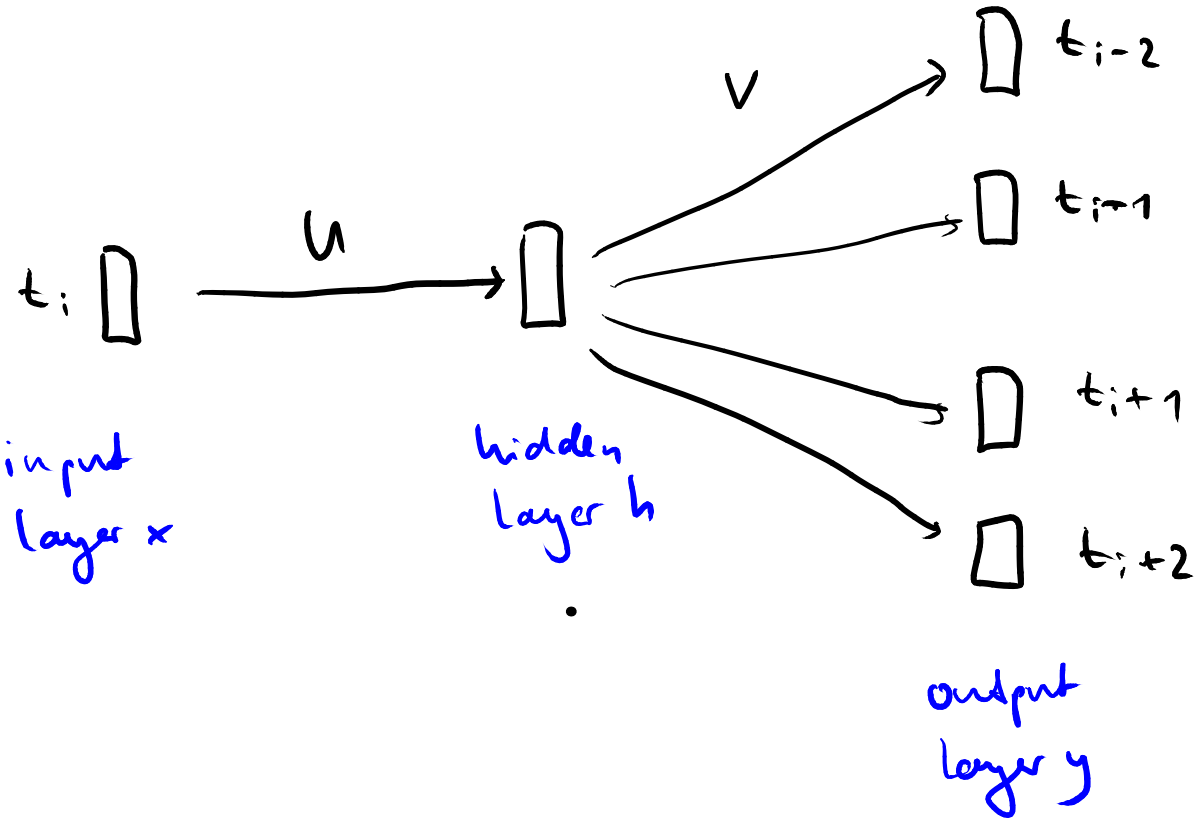


$$y = \text{softmax}(V \cdot h)$$

(here: context of size $k = 4$)

Variant 2: Skip-gram

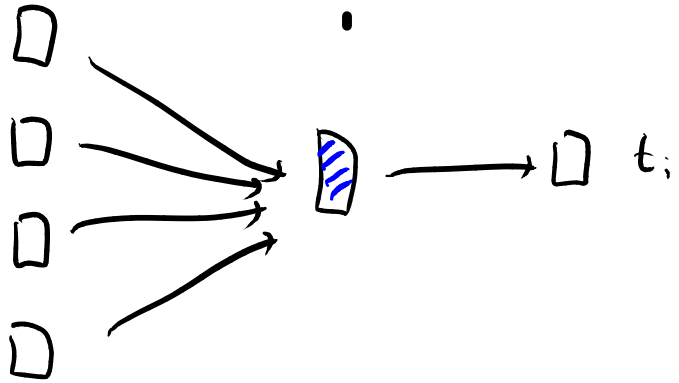
Predict context from token

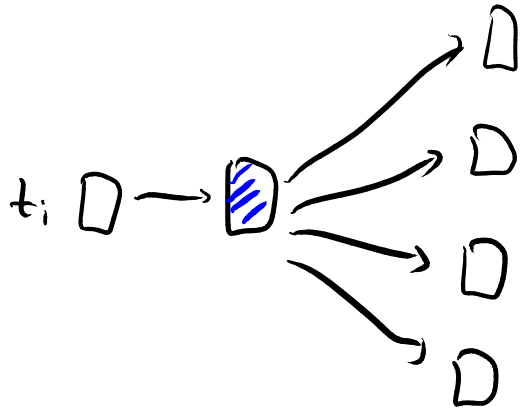


$$h = U \cdot x$$

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

Getting the Embedding





Once the network has become good at its task (through training):

Use <u>hidden layer</u> 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`

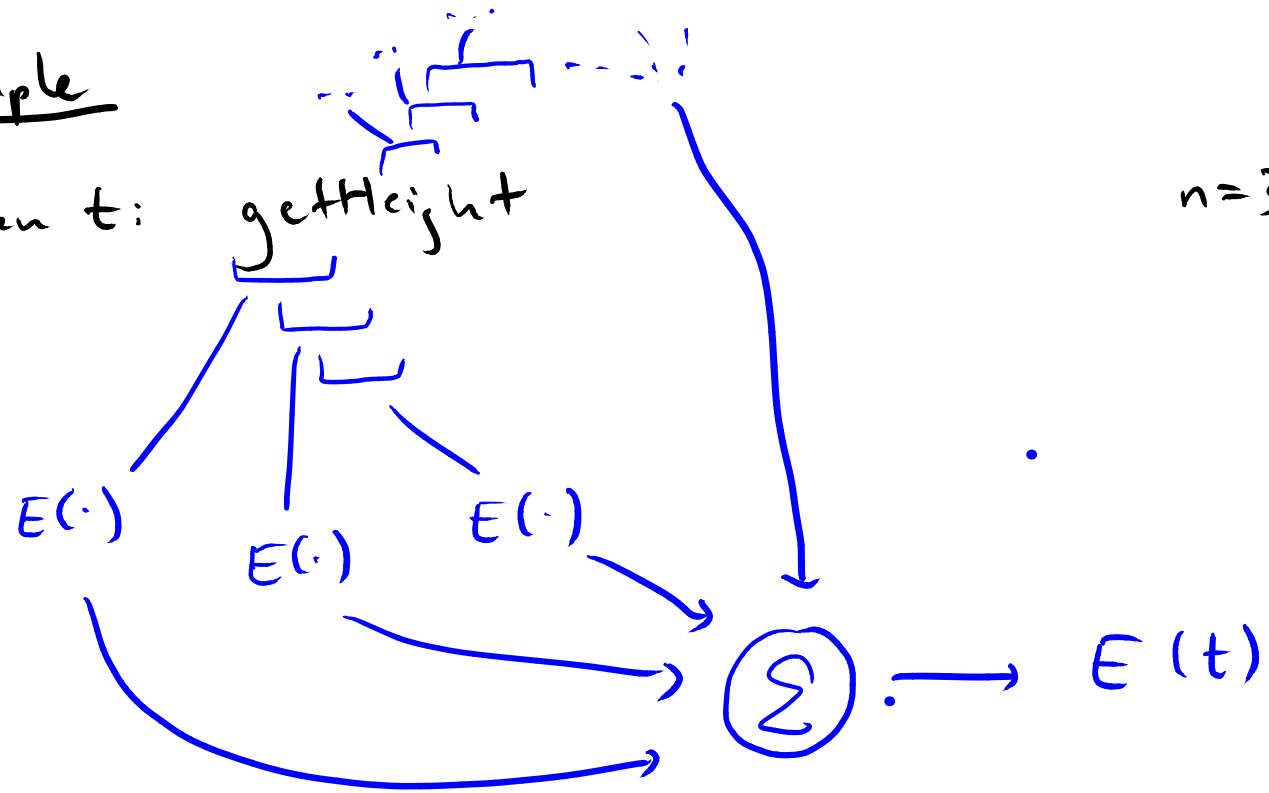
FastText

- **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