Below and beyond the word level:
Subword embeddings and embeddings for phrases, sentences, documents...

VL Embeddings

Uni Heidelberg

SS 2019
Below and beyond words

- We can learn semantic representations for words

- But what about other linguistic units?
  - characters
  - morphemes
  - phrases
  - sentences
  - paragraphs
  - documents
Subword embeddings

- **Motivation:**
  High-quality representations for rare or unknown words for
  - morphologically rich languages
  - low-resourced languages
  - languages with no clear word boundaries
  - noisy text (learner language, user-generated content)
  - text from new domains (with many unknown words)
Subword embeddings

- **Motivation:**
  High-quality representations for rare or unknown words for
  - morphologically rich languages
  - low-resourced languages
  - languages with no clear word boundaries
  - noisy text (learner language, user-generated content)
  - text from new domains (with many unknown words)

- **FastText:**
  - word embeddings enriched with subword information
Subword embeddings

- **Motivation:**
  High-quality representations for rare or unknown words for
  - morphologically rich languages
  - low-resourced languages
  - languages with no clear word boundaries
  - noisy text (learner language, user-generated content)
  - text from new domains (with many unknown words)

- **FastText:**
  - word embeddings enriched with subword information

Why not training representations for subword units directly?
Subword embedding types

- Character-based embeddings (characters or char-ngrams)
  - Ling et al. 2015; Luong and Manning 2016; Chiu and Nichols 2016
- Phonemes and Graphemes
  - Chaudhary et al. 2018
- Morphemes
  - Luong et al., 2013; Botha and Blunsom, 2014; Cotterell and Schütze, 2015; Chaudhary et al. 2018
- Byte-pair encoding
  - Sennrich et al. 2016; Heinzerling and Strube 2018
- Compound embeddings
  - Do et al. 2017
Character-based embeddings

- Based on
  - recurrent neural networks (RNN) (Ling et al. 2015)
  - convolutional neural networks (CNN) (Chiu and Nichols, 2016)

from Ling et al. (2015)
Character-based embeddings

- Often used in combination with word embeddings, e.g. for
  - POS/NER tagging (e.g. dos Santos and Zadrozny 2014; dos Santos et al. 2015; Ma and Hovy 2016; Lample et al. 2016)
  - dependency parsing (e.g. Ma et al. 2018)
  - text normalisation (Watson et al. 2018)
  - ...
Byte-pair encoding (BPE)

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
  e.g.: 1. iteration: \( \text{t h} \rightarrow \text{th} \)
    2. iteration: \( \text{th e} \rightarrow \text{the} \)
Byte-pair encoding (BPE)

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: t h → th
    2. iteration: th e → the

  aaabdaaabac
Byte-pair encoding (BPE)

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: t h → th
    2. iteration: th e → the

aaabdaaabac   Z=aa
Byte-pair encoding (BPE)

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: t h → th
          2. iteration: th e → the

    aaabdaaabac       Z=aa
    ZabdZabac
Byte-pair encoding (BPE)

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: t h $\rightarrow$ th
    2. iteration: th e $\rightarrow$ the

    aaabdaaabac   Z=aa
    ZabdZabac     Y=ab
Byte-pair encoding (BPE)

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: t h → th
          2. iteration: th e → the

Examples:
- aaabdaaabac  Z=aa
- ZabdZabac    Y=ab
- ZYdZYYac
Byte-pair encoding (BPE)

• Variable-length encoding: text as a sequence of symbols
  • iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: t h → th
          2. iteration: th e → the

  aaabdaaabac  Z=aa
  ZabdZabac     Y=ab
  ZYdZYac       X=ZY

Example from: https://howlingpixel.com/i-en/Byte_pair_encoding

• Parameter o: number of merge operations
  • o determines if resulting encoding mostly creates short character sequences (e.g. o = 1000) or if it includes symbols for many frequently occurring words, e.g. o = 30,000.
**Byte-pair encoding (BPE)**

- Variable-length encoding: text as a sequence of symbols
  - iteratively merge most frequent symbol pair into a new symbol
    - e.g.: 1. iteration: \( t \ h \rightarrow th \)
    - 2. iteration: \( th \ e \rightarrow the \)

\[
\begin{align*}
\text{aaabdaaabac} & \quad Z=aa \\
\text{ZabdZabac} & \quad Y=ab \\
\text{ZYdZYac} & \quad X=ZY \\
\text{XdXac}
\end{align*}
\]

Example from: [https://howlingpixel.com/i-en/Byte_pair_encoding](https://howlingpixel.com/i-en/Byte_pair_encoding)
Byte-pair encoding (BPE)

• **Variable-length encoding**: text as a sequence of symbols
  • iteratively merge most frequent symbol pair into a new symbol
    e.g.: 1. iteration: `t h` → `th`
    2. iteration: `th e` → `the`

  aaabdaaabac
  ZabdZabac
  ZYdZYac
  XdXac

  Z=aa
  Y=ab
  X=ZY

Example from:
https://howlingpixel.com/i-en/Byte_pair_encoding

• **Parameter** $o$: number of merge operations
• $o$ determines if resulting encoding mostly creates short character sequences (e.g. $o = 1000$) or if it includes symbols for many frequently occurring words, e.g. $o = 30,000$
Byte-pair encoding (BPE)

Heinzerling and Strube (2018): Collection of pre-trained subword embeddings in 275 languages

- https://github.com/bheinzerling/bpemb

- Based on Byte-Pair Encoding (BPE)
- Trained on Wikipedia:
  1. iterate over Wikipedia to create byte-pairs
  2. pretrain embeddings for resulting BPE symbol using GloVe
Byte-pair encoding (BPE)

Heinzerling and Strube (2018): Collection of pre-trained subword embeddings in 275 languages

- https://github.com/bheinzerling/bpemb

- Based on Byte-Pair Encoding (BPE)
- Trained on Wikipedia:
  1. iterate over Wikipedia to create byte-pairs
  2. pretrain embeddings for resulting BPE symbol using GloVe

- Advantages of BPE:
  - competitive performance to other types of embeddings for entity typing
  - more compact representations
  - no tokenisation required
Beyond word embeddings: phrase vectors

Mikolov et al. (2013c): Distributed representations of words and phrases and their compositionality

**New York Times** ⇒ newspaper
(not combination of new and york and times)
Beyond word embeddings: phrase vectors

Mikolov et al. (2013c): Distributed representations of words and phrases and their compositionality

New York Times $\Rightarrow$ newspaper
(not combination of new and york and times)

- Goal: Learn vectors that represent phrases instead of words
Beyond word embeddings: phrase vectors

Mikolov et al. (2013c): Distributed representations of words and phrases and their compositionality

**New York Times** ⇒ newspaper
(not combination of new and york and times)

- **Goal**: Learn vectors that represent phrases instead of words
- **Approach**:
  1. find words that occur frequently together, and infrequently in other context
  2. merge those into an atomic representation, e.g.:
     
     New York Times ⇒ New_York_Times

3. train word vectors on the modified corpus
   where phrases are now new atomic words
Phrase Vectors

Evaluation

- Analogical reasoning task:

- Test set with both words and phrases
  Steve Jobs : Apple :: Bill Gates : ?
  - correct if nearest representation to
    \[ \text{vec("Apple")} - \text{vec("Steve Jobs")} + \text{vec("Bill Gates"')} \]
    is \[ \text{vec("?"')} \]
  - 5 different categories of analogies
Phrase Vectors

Evaluation

- Analogical reasoning task:

- Test set with both words and phrases
  Steve Jobs : Apple :: Bill Gates : Microsoft
  - correct if nearest representation to
    \[ \text{vec(“Apple”) - vec(“Steve Jobs”) + vec(“Bill Gates”)} \]
    is \[ \text{vec(“Microsoft”)} \]
  - 5 different categories of analogies
Phrase Vectors

Evaluation

- Analogical reasoning task:


<table>
<thead>
<tr>
<th>Newspapers</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>New York Times</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>San Jose</td>
<td>San Jose Mercury News</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Baltimore</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cincinnati</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baltimore Sun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cincinnati Enquirer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NHL Teams</td>
<td></td>
<td>Montreal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boston</td>
<td>Boston Bruins</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phoenix</td>
<td>Phoenix Coyotes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Montreal Canadiens</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nashville Predators</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portland Trailblazers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBA Teams</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detroit</td>
<td>Detroit Pistons</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oakland</td>
<td>Golden State Warriors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toronto</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memphis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Airlines</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austria</td>
<td>Austrian Airlines</td>
<td>Spain</td>
<td>Spainair</td>
<td></td>
</tr>
<tr>
<td>Belgium</td>
<td>Brussels Airlines</td>
<td>Greece</td>
<td>Aegean Airlines</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Company executives</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steve Ballmer</td>
<td>Microsoft</td>
<td>Larry Page</td>
<td>Google</td>
<td></td>
</tr>
<tr>
<td>Samuel J. Palmisano</td>
<td>IBM</td>
<td>Werner Vogels</td>
<td>Amazon</td>
<td></td>
</tr>
</tbody>
</table>
Phrase Vectors

Evaluation

- Train different SkipGram models with dimensions = 300 and context size=5 on news data
  - Hierarchical Softmax versus Negative Sampling
  - with/without subsampling of frequent tokens

<table>
<thead>
<tr>
<th>Method</th>
<th>Dimensionality</th>
<th>no subsampling [%]</th>
<th>$10^{-5}$ subsampling [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG-5</td>
<td>300</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>NEG-15</td>
<td>300</td>
<td>27</td>
<td>42</td>
</tr>
<tr>
<td>HS-Huffman</td>
<td>300</td>
<td>19</td>
<td>47</td>
</tr>
</tbody>
</table>

Table: Accuracies of SkipGram models on phrase analogy dataset.
Phrase Vectors

Evaluation

- Train different SkipGram models with dimensions $= 300$ and context size $= 5$ on news data
  - Hierarchical Softmax versus Negative Sampling
  - with/without subsampling of frequent tokens

- Maximise accuracy by increasing amount of training data
  $\Rightarrow$ dataset with about 33 billion words
  - Hierarchical Softmax, dimension $= 1000$, context size $= $ entire sentence

- increased accuracy of $72\%$
Phrase Vectors

Evaluation

• Train different SkipGram models with dimensions = 300 and context size=5 on news data
  • Hierarchical Softmax versus Negative Sampling
  • with/without subsampling of frequent tokens

• Maximise accuracy by increasing amount of training data
  ⇒ dataset with about 33 billion words
  • Hierarchical Softmax, dimension = 1000, context size = entire sentence

• increased accuracy of 72%

Best model for analogy task: hierarchical softmax and subsampling of frequent words
Additive compositionality

- Word and phrase representations exhibit a linear structure that makes it possible to perform analogical reasoning using simple vector arithmetics

\[ \text{vec(Berlin)} - \text{vec(Germany)} + \text{vec(France)} = \text{Paris} \]
Additive compositionality

- Word and phrase representations exhibit a linear structure that makes it possible to perform analogical reasoning using simple vector arithmetics

\[ \text{vec(Berlin)} - \text{vec(Germany)} + \text{vec(France)} = \text{Paris} \]

- Word vectors also show additive compositionality:
  - combine words by an element-wise addition of their vector representations, e.g.:
Additive compositionality

• Word and phrase representations exhibit a linear structure that makes it possible to perform analogical reasoning using simple vector arithmetics

$$\text{vec}(\text{Berlin}) - \text{vec}(\text{Germany}) + \text{vec}(\text{France}) = \text{Paris}$$

• Word vectors also show additive compositionality:
  • combine words by an element-wise addition of their vector representations, e.g.:

$$\text{vec}(\text{Vietnam}) + \text{vec}(\text{capital}) = \text{Hanoi}$$
Additive compositionality

- Word and phrase representations exhibit a linear structure that makes it possible to perform analogical reasoning using simple vector arithmetics

\[ \text{vec}(\text{Berlin}) - \text{vec}(\text{Germany}) + \text{vec}(\text{France}) = \text{Paris} \]

- Word vectors also show additive compositionality:
  - combine words by an element-wise addition of their vector representations, e.g.:

\[ \text{vec}(\text{German}) + \text{vec}(\text{airlines}) = \text{Lufthansa} \]
Additive compositionality

- Word and phrase representations exhibit a linear structure that makes it possible to perform analogical reasoning using simple vector arithmetics

\[ \text{vec}(\text{Berlin}) - \text{vec}(\text{Germany}) + \text{vec}(\text{France}) = \text{Paris} \]

- Word vectors also show additive compositionality:
  - combine words by an element-wise addition of their vector representations, e.g.:

\[ \text{vec}(\text{French}) + \text{vec}(\text{actress}) = \text{Juliette Binoche} \]
Beyond Words: Sentence and Document Representations

Le and Mikolov (2014): Distributed Representations of Sentences and Documents

• Paragraph Vector
  • learns fixed-length feature representations from variable-length pieces of texts (sentences, paragraphs, documents)
Beyond Words: Sentence and Document Representations

Motivation

• Standard features for many **text classification** tasks: BoW
  • text is represented by fixed-length vectors of bag-of-words or bag-of-ngrams
  • simple, efficient, hard-to-beat baseline
Beyond Words: Sentence and Document Representations

Motivation

- Standard features for many text classification tasks: BoW
  - text is represented by fixed-length vectors of bag-of-words or bag-of-ngrams
  - simple, efficient, hard-to-beat baseline

- Disadvantages
  - word order is lost (or only preserved for short contexts)
    \[\rightarrow\] semantically different sentences can have the same (or very similar) representations:

  When Mary started singing, everybody went home.

  When everybody went home, Mary started singing.
Beyond Words: Sentence and Document Representations

Motivation

• How can we get meaningful representations for sequences of words?

• Two very simple approaches:

  1. Phrase vectors (Mikolov et al. 2013c)
     ⇒ merge word collocations into a new, atomic string and train embeddings for that new “word”

  2. Combine word vectors by **concatenating** them or by taking the **average** of two vectors, then use resulting vector to predict other words in the context
     (Bengio et al., 2006; Collobert & Weston, 2008; Mnih & Hinton, 2008; Turian et al., 2010; Mikolov et al., 2013a,b)
Beyond Words

Le and Mikolov (2014): Distributed Representations of Sentences and Documents

• Learn representations for whole sentences, paragraphs, documents... ⇒ vector representation is trained to predict words in a paragraph
  1. concatenate paragraph vector with several word vectors from the paragraph
  2. predict the following word in the given context
  3. train both, word and paragraph vectors, using stochastic gradient descent and backpropagation (Rumelhart et al., 1986)

• Paragraph vectors are unique among paragraphs
• Word vectors are shared across all paragraphs
Beyond Words
Le and Mikolov (2014)

Intuition

• Word vectors:
  • contribute to predicting words in sentence context

• Paragraph vectors:
  • contribute to predicting words sampled from whole paragraph
Beyond Words
Le and Mikolov (2014)

Word vector model

Classifier

Average/Concatenate

Word Matrix
Beyond Words
Le and Mikolov (2014)

Paragraph vector model

Classifier

Average/Concatenate

Paragraph Matrix→

Paragraph id

D

W

W

W

on

the

cat

sat
Beyond Words
Le and Mikolov (2014)

- Technical details:
  - Sample fixed-length contexts from a sliding window over the paragraph
  - Paragraph vector is shared across all contexts generated from the same paragraph
  - Word vector matrix is shared across paragraphs
Beyond Words

Le and Mikolov (2014)

- Technical details:
  - Sample fixed-length contexts from a sliding window over the paragraph
  - Paragraph vector is shared across all contexts generated from the same paragraph
  - Word vector matrix is shared across paragraphs

- Training with SGD and backpropagation

- In every iteration
  1. sample a fixed-length context from a random paragraph,
  2. compute the error gradient from the network
  3. use gradient to update parameters of the model
Beyond Words
Le and Mikolov (2014)

- Advantages of the paragraph vectors
  - inherit properties of word vectors
  - sensitive to word order (at least in a small context)
  - less sparse than bag-of-ngram models

- Extension of the model: Distributed bag of words version of Paragraph Vector (PV-DBOW)
  - similar to SkipGram (not shown here, see paper)
References

- Maas, Andrew L., Daly, Raymond E., Pham, Peter T., Huang, Dan, Ng, Andrew Y., and Potts, Christopher. Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, 2011.


References

- Omer Levy and Yoav Goldberg (2014): Dependency-based word embeddings. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 302–308, Baltimore, Maryland, USA
References

- Bich-Ngoc Do, Ines Rehbein and Anette Frank (2017): What do we need to know about an unknown word when parsing German. 1st Workshop on Subword and Character Level Models in NLP. Copenhagen, Denmark.