TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel
Spring 2019

Lecture 4:
Subword Modeling and Contextualized Word Embeddings
Roadmap

- intro (1 lecture)
- **deep learning for NLP** (5 lectures)
  - structured prediction: sequence labeling, syntactic and semantic parsing, dynamic programming (4 lectures)
  - generative models, latent variables, unsupervised learning, variational autoencoders (2 lectures)
- Bayesian methods in NLP (2 lectures)
- Bayesian nonparametrics in NLP (2 lectures)
- review & other topics (1 lecture)
Today

• modeling subword structure in words
• contextualized word embeddings
Recap

• on Monday we briefly reviewed some models and loss functions for word embeddings
Other Work on Word Embeddings

• using subword information (e.g., characters) in word embeddings

• multiple embeddings for a single word type corresponding to different word senses

• tailoring embeddings using particular resources or for particular NLP tasks
Other Work on Word Embeddings

• using subword information (e.g., characters) in word embeddings

• multiple embeddings for a single word type corresponding to different word senses

• tailoring embeddings using particular resources or for particular NLP tasks
Subword Modeling for Word Embeddings

• Using word embeddings has limitations:
  – closed vocabulary (100k-300k words is typical)
  – large number of parameters! (100k * 300)
  – for morphologically-rich languages, using a separate vector for each word type is “obviously” wrong

• Solution: character-level modeling
  – open vocabulary, fewer parameters, often similar or better performance
Early Neural Methods

morphological analyzer + recursive neural network:

Figure 1: Morphological Recursive Neural Network. A vector representation for the word “unfortunately” is constructed from morphemic vectors: \(un_{\text{pre}}, fortunate_{\text{stm}}, ly_{\text{suf}}\). Dotted nodes are computed on-the-fly and not in the lexicon.

Luong et al. (2013): Better Word Representations with Recursive Neural Networks for Morphology
unsupervised morphological analysis & vector addition:

\[
\text{im} \rightarrow \text{per} + \text{fect} + \text{ion} \rightarrow \text{perfect} + \text{ly}.
\]

We include the surface form of a word as a factor itself. This accounts for noncompositional constructions (\text{greenhouse} = \text{greenhouse} + \text{green} + \text{house}), and makes the approach more robust to noisy morphological segmentation. This strategy also overcomes the order-invariance of additive composition (\text{hangover} \neq \text{overhang}).

Botha & Blunsom (2014): *Compositional Morphology for Word Representations and Language Modelling*
visualization of learned morpheme vectors:

Botha & Blunsom (2014): *Compositional Morphology for Word Representations and Language Modelling*
• 2013-2014: morphological analyzers + define composition function on morphemes + learn embeddings for morphemes

• today, researchers use one of the following:
  – RNNs on character sequences (Ling et al., 2015; Ballesteros et al., 2015)
  – CNNs on character sequences (dos Santos and Zadrozny, 2014; Zhang et al., 2015; Kim et al., 2016)
  – represent words as bags of character $n$-grams, learn embeddings for character $n$-grams
Bidirectional LSTM over Characters

Ling et al. (2015): Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation
Convolutional Neural Network over Character Sequence

Kim et al. (2016): Character-Aware Neural Language Models
Convolutional Neural Networks

• convolutional neural networks (CNNs) use filters that are “convolved with” (matched against all positions of) the input

• informally, think of convolution as “perform the same operation over multiple parts of the input in some systematic order”

• CNNs are often used in NLP to convert a word or sentence into a feature vector
Filters

• for now, think of a filter as a vector in the word embedding space

• the filter matches a particular region of the space

• “match” = “has high dot product with”
Convolution

\[ x = \textit{not that great} \]

\[ x = \begin{bmatrix} 0.4 & \ldots & 0.9 & 0.2 & \ldots & 0.7 & 0.3 & \ldots & 0.6 \end{bmatrix}^\top \]

vector for \textit{not} \hspace{1cm} vector for \textit{that} \hspace{1cm} vector for \textit{great}

consider a single convolutional filter \( w \in \mathbb{R}^d \)
Convolution

compute dot product of filter and each word vector:

\[ x = \textit{not that great} \]

\[ w \]

\[ x = [0.4 \ldots 0.9 \ 0.2 \ldots 0.7 \ 0.3 \ldots 0.6]^\top \]

vector for not \quad vector for that \quad vector for great

\[ c_1 = w^\top x_{1:d} \]
ConvoluMon

compute dot product of filter and each word vector:

\[ \mathbf{x} = \text{not that great} \]

\[ \mathbf{x} = \begin{bmatrix} 0.4 & \ldots & 0.9 & 0.2 & \ldots & 0.7 & 0.3 & \ldots & 0.6 \end{bmatrix}^\top \]

- vector for not
- vector for that
- vector for great

\[ c_1 = \mathbf{w}^\top \mathbf{x}_{1:d} \]

\[ c_2 = \mathbf{w}^\top \mathbf{x}_{d+1:2d} \]
ConvoluMon

compute dot product of filter and each word vector:

\[ \mathbf{x} = \textit{not that great} \]

\[ \mathbf{x} = \begin{bmatrix} 0.4 & \ldots & 0.9 & 0.2 & \ldots & 0.7 & 0.3 & \ldots & 0.6 \end{bmatrix}^\top \]

vector for \textit{not}  \hspace{1cm} \text{vector for} \ \textit{that}  \hspace{1cm} \text{vector for} \ \textit{great}  

\[ c_1 = \mathbf{w}^\top \mathbf{x}_{1:d} \]

\[ c_2 = \mathbf{w}^\top \mathbf{x}_{d+1:2d} \]

\[ c_3 = \mathbf{w}^\top \mathbf{x}_{2d+1:3d} \]
ConvoluMon

\[ x = \text{not that great} \]

\[ x = [0.4 \ldots 0.9 \ 0.2 \ldots 0.7 \ 0.3 \ldots 0.6]^\top \]

vector for not vector for that vector for great

\[ c_1 = w^\top x_{1:d} \]

\[ c_2 = w^\top x_{d+1:2d} \]

\[ c_3 = w^\top x_{2d+1:3d} \]

Note: it’s common to add a bias \( b \) and use a nonlinearity \( g \):

\[ c_1 = g \left( w^\top x_{1:d} + b \right) \]
Convolution

$x = \text{not that great}$

$x = [0.4 \ldots 0.9 \ 0.2 \ldots 0.7 \ 0.3 \ldots 0.6]^{\top}$

\[ c_1 = w^{\top} x_{1:d} \]

\[ c_2 = w^{\top} x_{d+1:2d} \]

\[ c_3 = w^{\top} x_{2d+1:3d} \]

c = “feature map” for this filter,

has an entry for each position in input (in this case, 3 entries)
$x = \textit{not that great}$

how do we convert this into a fixed-length vector? use \textbf{pooling}:

max-pooling: returns maximum value in $c$

average pooling: returns average of values in $c$

\[
\begin{align*}
    c_1 &= w \times x_{1:d} \\
    c_2 &= w^\top x_{d+1:2d} \\
    c_3 &= w^\top x_{2d+1:3d}
\end{align*}
\]

$c = \text{“feature map” for this filter,}

has an entry for each position in input (in this case, 3 entries)
Pooling

\[ \mathbf{x} = \textit{not that great} \]

how do we convert this into a fixed-length vector? use \textbf{pooling}:

- max-pooling: returns maximum value in \( \mathbf{c} \)
- average pooling: returns average of values in \( \mathbf{c} \)

\[ c_1 = \mathbf{w} \cdot x_{1:d} \]
\[ c_2 = \mathbf{w}^\top x_{d+1:2d} \]

then, this single filter \( \mathbf{w} \) produces a single feature value (the output of some kind of pooling).

in practice, we use many filters of many different lengths (e.g., \( n \)-grams rather than words).
**Convolutional Neural Networks**

- “convolutional layer” = set of filters that are convolved with the input vector (whether $\mathbf{x}$ or hidden vector)

- could be followed by more convolutional layers, or by a type of pooling

- filters of varying n-gram lengths commonly used (1- to 5-grams)

- CNNs commonly used for character-level processing; filters look at character n-grams
Convolutional Neural Network over Characters

Kim et al. (2016): Character-Aware Neural Language Models
1. What dimension are the character embeddings?
1. What dimension are the character embeddings? 4
2. How many character 4-gram filters are there?
1. What dimension are the character embeddings? 4
2. How many character 4-gram filters are there? 5
3. Why do different filter lengths lead to different lengths of feature maps?
• what about simpler methods?
• add or average vectors for character $n$-grams in the word:
  – word space (Schutze, 1993)
  – deep structured semantic models (Huang et al., 2013)
  – charagram (Wieting et al., 2016)
  – fastText (Bojanowski et al., 2017)
DSSM (Microsoft Research, 2013-2016)

Tri-letter: a scale-able word representation

- Tri-letter based Word Hashing of “cat”
  - -> #cat#
  - Tri-letters: #-c-a, c-a-t, a-t-#

- Compact representation
  - $|\text{Voc}|$ (500K) $\rightarrow$ $|\text{TriLetter}|$ (30K)
  - Generalize to unseen words
  - Robust to misspelling, inflection, etc.

\[
x(\text{cat}) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}
\]

The index of word cat in the vocabulary

\[
f(\text{cat}) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 1 \\ 1 \\ \vdots \\ 0 \end{bmatrix}
\]

Indices of #-c-a, c-a-t, a-t-# in the letter-tri-gram list, respectively.

Huang et al. (2013): Learning Deep Structured Semantic Models for Web Search using Clickthrough Data
DSSM (Microsoft Research, 2013-2016)

Word hashing by n-gram of letters

- Collision:
  - What if different words have the same word hashing vector?
  - Statistics
    - 22 out of 500K words collide
    - Collision Example: #bananna# <-> #bannana#

<table>
<thead>
<tr>
<th>Vocabulary size</th>
<th>Unique tri-letter observed in voc</th>
<th>Number of Collisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>40K</td>
<td>10306</td>
<td>2</td>
</tr>
<tr>
<td>500K</td>
<td>30621</td>
<td>22</td>
</tr>
</tbody>
</table>

Huang et al. (2013): *Learning Deep Structured Semantic Models for Web Search using Clickthrough Data*
“Charagram” Embeddings

• to embed a character sequence (word or sentence), sum embeddings for character $n$-grams

• only parameters to learn are embeddings for character $n$-grams

Wieting et al. (2016): Charagram: Embedding Words and Sentences via Character $n$-grams
Wieting et al. (2016): Charagram: Embedding Words and Sentences via Character n-grams

Charagram Embeddings

- Faster convergence to strong performance than character LSTM or CNN
Charagram Word Embeddings

• we used all 122,610 character $n$-grams observed in training set ($2 \leq n \leq 4$), including spaces

• we trained on paraphrase pairs from the Paraphrase Database

Wieting et al. (2016): Charagram: Embedding Words and Sentences via Character n-grams
For words in training set:

<table>
<thead>
<tr>
<th>word</th>
<th>nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>refunding</td>
<td>refunds, refunded, refund, repayment, reimbursement, rebate, repay reimbursements, reimburse, repaying, repayments, rebates, rebating</td>
</tr>
<tr>
<td>professors</td>
<td>professor, professorships, professorship, teachers, professorial, teacher prof., teaches, lecturers, teachings, instructors, headteachers</td>
</tr>
<tr>
<td>huge</td>
<td>enormous, tremendous, large, big, vast, overwhelming, immense, giant formidable, considerable, massive, huger, large-scale, great, daunting</td>
</tr>
</tbody>
</table>

Wieting et al. (2016): *Charagram: Embedding Words and Sentences via Character n-grams*
For words in training set:

<table>
<thead>
<tr>
<th>word</th>
<th>nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>refunding</td>
<td>refunds, refunded, refund, repayment, reimbursement, rebate, repay reimbursements, reimburse, repaying, repayments, rebates, rebating</td>
</tr>
<tr>
<td>professors</td>
<td>professor, professorships, professorship, teachers, professorial, teacher prof., teaches, lecturers, teachings, instructors, headteachers</td>
</tr>
<tr>
<td>huge</td>
<td>enormous, tremendous, large, big, vast, overwhelming, immense, giant formidable, considerable, massive, huger, large-scale, great, daunting</td>
</tr>
</tbody>
</table>

For words not in training set:

<table>
<thead>
<tr>
<th>word</th>
<th>nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>vehicals</td>
<td>vehical, vehicles, vehicels, vehicular, cars, vehicle, automobiles, car</td>
</tr>
<tr>
<td>journeying</td>
<td>journey, journeys, voyage, trip, roadtrip, travel, tourney, voyages, road-trip</td>
</tr>
<tr>
<td>babyyyyyy</td>
<td>babyyyyyyyy, baby, babys, babe, baby.i, babydoll, babycake, darling</td>
</tr>
</tbody>
</table>

Wieting et al. (2016): *Charagram: Embedding Words and Sentences via Character n-grams*
fastText

• like word2vec, but represents a word as the sum of its character $n$-gram embeddings and an embedding for the word itself

Bojanowski et al. (2017): *Enriching Word Vectors with Subword Information*
fastText

- better data efficiency than word2vec:

Figure 1: Influence of size of the training data on performance. We compute word vectors following the proposed model using datasets of increasing size. In this experiment, we train models on a fraction of the full Wikipedia dump.

Bojanowski et al. (2017): *Enriching Word Vectors with Subword Information*
• if you’re just encoding text (rather than generating), you can use neural architectures like these to capture subword information

• for generation, it’s trickier:
  – character RNNs are fine for generating words, but not sentences (very long sequences and long-distance dependencies)

• simple, data-driven segmentation methods have emerged as the standard way to handle this
Data-Driven Segmentation

• Most popular methods:
  – Byte pair encoding (BPE)
  – SentencePiece’s unigram LM

SentencePiece

SentencePiece is an unsupervised text tokenizer and detokenizer mainly for Neural Network-based text generation systems where the vocabulary size is predetermined prior to the neural model training. SentencePiece implements subword units (e.g., byte-pair-encoding (BPE) [Sennrich et al.]) and unigram language model [Kudo.]) with the extension of direct training from raw sentences. SentencePiece allows us to make a purely end-to-end system that does not depend on language-specific pre/postprocessing.

This is not an official Google product.

Technical highlights
Data-Driven Segmentation

• Most popular methods:
  – Byte pair encoding (BPE)
  – SentencePiece’s unigram LM

• these are easy and fast to use & work well

• they permit unbounded vocabularies with a relatively small number of parameters
Byte Pair Encoding (BPE)  
(Gage, 1994)

• simple data compression technique

• iteratively replaces most frequent pair of bytes in a sequence with a single, unused byte

• Sennrich et al. (2016) adapted BPE for characters and character sequences
Byte Pair Encoding (BPE)

- **merge**: a rule that combines two consecutive units into a single unit
- initially, units are characters
- after merges, units become character sequences
- greedy algorithm:
  - merge two units with the largest unit bigram count, produce merged unit
  - replace all instances of that 2-unit sequence with the merged unit, recompute counts
Byte Pair Encoding (BPE)

• Sennrich et al. use BPE based on word counts from a corpus
  – sentences are not used; all that’s needed are word types and their counts

  – special treatment for end-of-word symbol $<\text{/w}>$
    (an unseen initial step merges final character in each word with $<\text{/w}>$)

  – when segmenting new data, segments words individually (does not use context)
Corpus:  Merges:

low
low
lower
lowest
high
high
higher

Sennrich et al. (2016): *Neural Machine Translation of Rare Words with Subword Units*
Corpus:

low
low
lower
lowest
high
high
high
higher

Merges:

actually the corpus looks like this

Sennrich et al. (2016): *Neural Machine Translation of Rare Words with Subword Units*
the first thing we do is merge word-ending characters with \(</w>\)

*Sennrich et al. (2016): Neural Machine Translation of Rare Words with Subword Units*
Corpus:  

<table>
<thead>
<tr>
<th>Word</th>
<th>Segmentation:</th>
<th>Merges:</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td></td>
<td>w &lt; /w&gt; (2)</td>
</tr>
<tr>
<td>low</td>
<td></td>
<td>r &lt; /w&gt; (2)</td>
</tr>
<tr>
<td>lower</td>
<td></td>
<td>h &lt; /w&gt; (2)</td>
</tr>
<tr>
<td>lowest</td>
<td></td>
<td>t &lt; /w&gt; (1)</td>
</tr>
<tr>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>high</td>
<td></td>
<td></td>
</tr>
<tr>
<td>higher</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sennrich et al. (2016): Neural Machine Translation of Rare Words with Subword Units**

given this set of merges, let’s segment the corpus!
Corpus:  

<table>
<thead>
<tr>
<th>Word</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>lower</td>
<td>low e r</td>
</tr>
<tr>
<td>lowest</td>
<td>low e s t</td>
</tr>
<tr>
<td>high</td>
<td>h i g h</td>
</tr>
<tr>
<td>high</td>
<td>h i g h</td>
</tr>
<tr>
<td>higher</td>
<td>h i g h e r</td>
</tr>
</tbody>
</table>

Sennrich et al. (2016): *Neural Machine Translation of Rare Words with Subword Units*
<table>
<thead>
<tr>
<th>Corpus:</th>
<th>Segmentation:</th>
<th>Merges:</th>
</tr>
</thead>
<tbody>
<tr>
<td>low&lt;/w&gt;</td>
<td>l o w&lt;/w&gt;</td>
<td>w &lt;/w&gt;</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>l o w&lt;/w&gt;</td>
<td>r &lt;/w&gt;</td>
</tr>
<tr>
<td>lower&lt;/w&gt;</td>
<td>l o w e r&lt;/w&gt;</td>
<td>h &lt;/w&gt;</td>
</tr>
<tr>
<td>lowest&lt;/w&gt;</td>
<td>l o w e s t&lt;/w&gt;</td>
<td>t &lt;/w&gt;</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>h i g h&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>h i g h&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>higher&lt;/w&gt;</td>
<td>h i g h e r&lt;/w&gt;</td>
<td></td>
</tr>
</tbody>
</table>

looking at the segmented corpus shows us what merge will occur next
<table>
<thead>
<tr>
<th>Corpus:</th>
<th>Segmentation:</th>
<th>Merges:</th>
</tr>
</thead>
<tbody>
<tr>
<td>low&lt;/w&gt;</td>
<td>low&lt;/w&gt;</td>
<td>w &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>low&lt;/w&gt;</td>
<td>r &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lower&lt;/w&gt;</td>
<td>lower&lt;/w&gt;</td>
<td>h &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lowest&lt;/w&gt;</td>
<td>lowest&lt;/w&gt;</td>
<td>t &lt;/w&gt; (1)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>high&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>high&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>higher&lt;/w&gt;</td>
<td>higher&lt;/w&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**what is the next merge?**  
(what unit bigram appears most often?)

Sennrich et al. (2016): Neural Machine Translation of Rare Words with Subword Units
Sennrich et al. (2016): *Neural Machine Translation of Rare Words with Subword Units*

<table>
<thead>
<tr>
<th>Corpus:</th>
<th>Segmentation:</th>
<th>Merges:</th>
</tr>
</thead>
<tbody>
<tr>
<td>low&lt;/w&gt;</td>
<td>low&lt;/w&gt;</td>
<td>w &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>low&lt;/w&gt;</td>
<td>r &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lower&lt;/w&gt;</td>
<td>lower&lt;/w&gt;</td>
<td>h &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lowest&lt;/w&gt;</td>
<td>lowest&lt;/w&gt;</td>
<td>t &lt;/w&gt; (1)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>high&lt;/w&gt;</td>
<td>lo &lt;/w&gt; (4)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>high&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>higher&lt;/w&gt;</td>
<td>higher&lt;/w&gt;</td>
<td></td>
</tr>
</tbody>
</table>

given this new set of merges, let’s re-segment the corpus!
<table>
<thead>
<tr>
<th>Corpus:</th>
<th>Segmentation:</th>
<th>Merges:</th>
</tr>
</thead>
<tbody>
<tr>
<td>low&lt;/w&gt;</td>
<td>lo w&lt;/w&gt;</td>
<td>w &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>lo w&lt;/w&gt;</td>
<td>r &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lower&lt;/w&gt;</td>
<td>lo w e r&lt;/w&gt;</td>
<td>h &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lowest&lt;/w&gt;</td>
<td>lo w e s t&lt;/w&gt;</td>
<td>t &lt;/w&gt; (1)</td>
</tr>
<tr>
<td>highest&lt;/w&gt;</td>
<td>h i g h h&lt;/w&gt;</td>
<td>l o</td>
</tr>
<tr>
<td>highest&lt;/w&gt;</td>
<td>h i g h h&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>higher&lt;/w&gt;</td>
<td>h i g h e r&lt;/w&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Sennrich et al. (2016): *Neural Machine Translation of Rare Words with Subword Units*

note: we will always “back off” to the complete segmentation.
Corpus: | Segmentation: |
---|---|
low</w> | lo w</w> |
low</w> | lo w</w> |
lower</w> | lo we r</w> |
lowest</w> | lo we st</w> |
high</w> | hi gh</w> |
high</w> | hi gh</w> |
higher</w> | hi ghe r</w> |

what is the next merge? (what unit bigram appears most often?)
<table>
<thead>
<tr>
<th>Corpus</th>
<th>Segmentation</th>
<th>Merges</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>lo w</td>
<td>w &lt;w&gt; (2)</td>
</tr>
<tr>
<td>low</td>
<td>lo w</td>
<td>r &lt;w&gt; (2)</td>
</tr>
<tr>
<td>low</td>
<td>lo w</td>
<td>h &lt;w&gt; (2)</td>
</tr>
<tr>
<td>low</td>
<td>lo w</td>
<td>t &lt;w&gt; (1)</td>
</tr>
<tr>
<td>lower</td>
<td>lo we r</td>
<td>l o (4)</td>
</tr>
<tr>
<td>lowest</td>
<td>lo we s t</td>
<td>i g (3)</td>
</tr>
<tr>
<td>highest</td>
<td>h i g h</td>
<td></td>
</tr>
<tr>
<td>highest</td>
<td>h i g h</td>
<td></td>
</tr>
<tr>
<td>higher</td>
<td>h i g h e r</td>
<td></td>
</tr>
<tr>
<td>Corpus:</td>
<td>Segmentation:</td>
<td>Merges:</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>lo w&lt;/w&gt;</td>
<td>w &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>lo w&lt;/w&gt;</td>
<td>r &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lower&lt;/w&gt;</td>
<td>lo we r&lt;/w&gt;</td>
<td>h &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lowest&lt;/w&gt;</td>
<td>lo we st&lt;/w&gt;</td>
<td>t &lt;/w&gt; (1)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>high&lt;/w&gt;</td>
<td>lo          (4)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>high&lt;/w&gt;</td>
<td>ig          (3)</td>
</tr>
<tr>
<td>higher&lt;/w&gt;</td>
<td>higher&lt;/w&gt;</td>
<td></td>
</tr>
<tr>
<td>Corpus</td>
<td>Segmentation</td>
<td>Merges</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>low</td>
<td>lo w</td>
<td>w (2)</td>
</tr>
<tr>
<td>low</td>
<td>lo w</td>
<td>r (2)</td>
</tr>
<tr>
<td>lower</td>
<td>lo w e r</td>
<td>h (2)</td>
</tr>
<tr>
<td>lowest</td>
<td>lo w e s t</td>
<td>t (1)</td>
</tr>
<tr>
<td>highest</td>
<td>h ig h</td>
<td>lo (4)</td>
</tr>
<tr>
<td>highest</td>
<td>h ig h</td>
<td>i g (3)</td>
</tr>
<tr>
<td>higher</td>
<td>h ig h e r</td>
<td>h ig (3)</td>
</tr>
<tr>
<td>Corpus</td>
<td>Segmentation</td>
<td>Merges</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>lo w&lt;/w&gt;</td>
<td>w &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>low&lt;/w&gt;</td>
<td>lo w&lt;/w&gt;</td>
<td>r &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lower&lt;/w&gt;</td>
<td>lo w e r&lt;/w&gt;</td>
<td>h &lt;/w&gt; (2)</td>
</tr>
<tr>
<td>lowest&lt;/w&gt;</td>
<td>lo w e s t&lt;/w&gt;</td>
<td>t &lt;/w&gt; (1)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>hig h&lt;/w&gt;</td>
<td>l o (4)</td>
</tr>
<tr>
<td>high&lt;/w&gt;</td>
<td>hig h&lt;/w&gt;</td>
<td>i g (3)</td>
</tr>
<tr>
<td>higher&lt;/w&gt;</td>
<td>hig h e r&lt;/w&gt;</td>
<td>h i g (3)</td>
</tr>
</tbody>
</table>
Corpus:

<table>
<thead>
<tr>
<th>low&lt;w&gt;</th>
<th>low&lt;w&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>low&lt;w&gt;</td>
<td>low&lt;w&gt;</td>
</tr>
<tr>
<td>lower&lt;w&gt;</td>
<td>lowe r&lt;w&gt;</td>
</tr>
<tr>
<td>lowest&lt;w&gt;</td>
<td>lowe s t&lt;w&gt;</td>
</tr>
<tr>
<td>high&lt;w&gt;</td>
<td>high&lt;w&gt;</td>
</tr>
<tr>
<td>high&lt;w&gt;</td>
<td>high&lt;w&gt;</td>
</tr>
<tr>
<td>higher&lt;w&gt;</td>
<td>hig h e r&lt;w&gt;</td>
</tr>
</tbody>
</table>

Segmentation:

Merges:

w </w> (2)

r </w> (2)

h </w> (2)

t </w> (1)

l o    (4)

i g    (3)

h i g   (3)

w e    (2)

lo we  (2)

lo w<w>(2)

hig h<w>(2)
New Corpus:

low (2x)
lower
lowest
high (2x)
higher
small
smaller
smallest

Merges for New Corpus:

lo
sm
sm a
i g
h i g
e r
smal l

s t
lo w
lo w
hig h
e st
Merges:

- low
- small
- smallest
- low
- lower
- lowest
- high
- higher
- highest
- small
- smaller
- smallest

Application:

- low $\rightarrow$ low
- lower $\rightarrow$ low er
- lowest $\rightarrow$ low est
- high $\rightarrow$ high
- higher $\rightarrow$ hig h er
- highest $\rightarrow$ hig h est
- small $\rightarrow$ smal l
- smaller $\rightarrow$ small er
- smallest $\rightarrow$ small est
• we can limit the vocabulary size of the segmented data by limiting the number of merge

• this can be very helpful for handling an open vocabulary of words while reducing computation (e.g., when using a softmax over the vocabulary)
Stanford Sentiment Treebank
BPE merging on train+dev sets
up to 20k merges (max number found: 15,417)

Example from test set:

writer/director/producer \(\rightarrow\)
write/er/or director/producer

(To recover original text, remove “@@ ”)
Here's a British flick gleefully unconservingly concerned with plausibility, yet just as determined to entertain you.

probably good: “unconcerned” becomes “un concerned”
maybe bad: “gle efully”

It wouldn't be my preferred red way of spending 100 minutes or $7.00.

probably good: “prefer” is related to “preferred”
maybe bad: “spending” is not related to “ending”
• BPE is a useful hack, doesn’t correspond to optimizing any probabilistic objective function

• other related methods have interpretations as probabilistic models

• we will see methods later in the course for unsupervised segmentation using probabilistic modeling and priors related to “minimum description length”
Other Work on Word Embeddings

• using subword information (e.g., characters) in word embeddings

• multiple embeddings for a single word type corresponding to different word senses

• tailoring embeddings using particular resources or for particular NLP tasks
Other Work on Word Embeddings

• using subword information (e.g., characters) in word embeddings

• multiple embeddings for a single word type corresponding to different word senses

• tailoring embeddings using particular resources or for particular NLP tasks
Multisense Word Embeddings

• one embedding for a word type is insufficient
  – due to different senses of a word, different meanings (polysemy, homonymy)

• there has been a lot of work in learning sense-specific word embeddings:
  – use a word sense labeler or cluster word tokens into clusters that capture word sense
  – learn embeddings for each sense/cluster
Multisense Word Embeddings

Huang et al. (2012): Improving Word Representations Via Global Context And Multiple Word Prototypes
• nearest neighbors given context:

<table>
<thead>
<tr>
<th>Context</th>
<th>Nearest Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple is a kind of fruit.</td>
<td>pear, cherry, mango, juice, peach, plum, fruit, cider, apples, tomato, orange, bean, pie</td>
</tr>
<tr>
<td>Apple releases its new ipads.</td>
<td>microsoft, intel, dell, ipad, macintosh, ipod, iphone, google, computer, imac, hardware</td>
</tr>
<tr>
<td>He borrowed the money from banks.</td>
<td>banking, credit, investment, finance, citibank, currency, assets, loads, imf, hsbc</td>
</tr>
<tr>
<td>along the shores of lakes, banks of rivers</td>
<td>land, coast, river, waters, stream, inland, area, coasts, shoreline, shores, peninsula</td>
</tr>
<tr>
<td>Basalt is the commonest volcanic rock.</td>
<td>boulder, stone, rocks, sand, mud, limestone, volcanic, sedimentary, pelt, lava, basalt</td>
</tr>
<tr>
<td>Rock is the music of teenage rebellion.</td>
<td>band, pop, bands, song, rap, album, jazz, blues, singer, hip-pop, songs, guitar, musician</td>
</tr>
</tbody>
</table>

Table 2: Nearest neighbors of words given context. The embeddings from context words are first inferred with the Greedy strategy; nearest neighbors are computed by cosine similarity between word embeddings. Similar phenomena have been observed in earlier work (Neelakantan et al., 2014)

Neelakantan et al. (2014): *Efficient nonparametric estimation of multiple embeddings per word in vector space*

Li & Jurafsky (2015): *Do Multi-Sense Embeddings Improve Natural Language Understanding?*
Multisense Word Embeddings

• limitations:
  – need a way to label senses or cluster word tokens in training data (and for downstream tasks)
  – fragments training data, so more may be needed for estimating word embeddings
  – unlikely to get good clusters for rare word types
  – unable to handle new senses that only appear in test data
  – unclear if sense-specific embeddings are useful for downstream tasks
Do Multisense Embeddings Help on NLP Tasks?

- yes, on some tasks
- but when using powerful neural architectures, multisense embeddings may not be needed
- increasing dimensionality of (single-sense) embeddings achieves some benefit of multisense embeddings
  - high dimensionality also may make it easier for subsequent architectures to extract relevant sense based on context

Li & Jurafsky (2015): Do Multi-Sense Embeddings Improve Natural Language Understanding?