Insights into Subword Embeddings

Sebastian Pütz

SFB833 A3
University of Tübingen

December 11, 2019

A different take on composition

- How to compose words?

A different take on composition

- How to compose words?
- What to compose into words!

Outline

1. Background
   - Word embeddings
   - Embeddings with subwords
2. Hashing trick
3. Explicit ngram lookup
4. Introspection & Evaluation
5. Conclusion & Outlook

Background

Background

Word embeddings

- Word representations in low-dimensional vector space.
- Trained unsupervisedly by predicting word-context co-occurrence.
  1. CBOW - given n context words, predict probability of the focus word
  2. skipgram - given a focus word, predict probability of context words

The skipgram algorithm

Mikolov et al. (2013) - word2vec

Skipgram with negative sampling

They$_c$ made$_c$ **some** tasty$_c$ food$_c$

The skipgram algorithm

Mikolov et al. (2013) - word2vec

Skipgram with negative sampling

They$_c$ made$_c$ **some** tasty$_c$ food$_c$

- Each context *word*$_c$ is a positive example for **some**
  - $\rightarrow$ For each *word*$_c$, query the model for $p(positive|\textbf{some}, word_c)$
  - $\rightarrow$ Update *word*$_c$ and **some** to increase probability.

The skipgram algorithm

Mikolov et al. (2013) - word2vec

Skipgram with negative sampling

They$_c$ made$_c$ **some** tasty$_c$ food$_c$

- Each context *word*$_c$ is a positive example for **some**
  - → For each *word*$_c$, query the model for $p(positive|\textbf{some}, word_c)$
  - → Update *word*$_c$ and **some** to increase probability.
- For each positive, draw n random, negative samples
  - → Query for $p(positive|\textbf{some}, random_c)$
  - → Update embeddings to decrease probability.

The skipgram algorithm

Mikolov et al. (2013) - word2vec

Skipgram with negative sampling

They$_c$ made$_c$ **some** tasty$_c$ food$_c$

- Each context $word_c$ is a positive example for **some**
  - → For each $word_c$, query the model for $p(positive|\textbf{some}, word_c)$
  - → Update $word_c$ and **some** to increase probability.
- For each positive, draw n random, negative samples
  - → Query for $p(positive|\textbf{some}, random_c)$
  - → Update embeddings to decrease probability.
- **word** and $word_c$ are distinct vectors.

Structured Skipgram

Ling et al. (2015)

Structured Skipgram

They_{c-1}	**made**	some_{c+1}	tasty_{c+2}	food_{c+3}
They_{c-2}	made_{c-1}	**some**	tasty_{c+1}	food_{c+2}

- Context words are typed by their offset wrt. the focus word.
- Vectors of context words at different offsets are distinct.
  - → They_{c-1} != They_{c-2}
  - → more focused contexts, sparser updates
- Typically perform better on syntactic tasks

Embeddings with subword information

Bojanowski et al. (2017) - fastText

Embeddings with subword information

$$\begin{array}{ccccc} & & <\text{so}_f + & & \\ & & \text{som}_f + & & \\ & & \text{ome}_f + & & \\ & & \text{me}>_f + & & \\ \text{He}_c & \text{made}_c & \textbf{some}_f & \text{tasty}_c & \text{food}_c \end{array}$$

- Ngrams also have embeddings.
- Words are represented by the average of their ngrams
  - → Ngram embeddings are shared across words.
  - → Orthographically similar words get similar representation.
- Known words get an additional, distinct vector

Structured skipgram with subword information

https://github.com/finalfusion/finalfrontier

Structured skipgram with subword information

$$\begin{array}{ccccc} & & \text{<so} + & & \\ & & \text{som} + & & \\ & & \text{ome} + & & \\ & & \text{me>} + & & \\ \text{He}_{c-2} & \text{made}_{c-1} & \textbf{some} & \text{tasty}_{c+1} & \text{food}_{c+2} \end{array}$$

- Combine structured skipgram with subword information
  → Better embeddings for syntactic tasks and broader coverage.

How are ngrams extracted?

How are ngrams extracted?

- Set a minimum and maximum length
  $\rightarrow$ typically 3 and 6
- Bracket words with '<' and '>'
  $\rightarrow$ with minimum length 3, all words will generate ngrams

How are ngrams extracted?

Word	Ngrams	#
a	<a>	1
is	<is + is>+ <is>	3
and	<an + and + nd> + <and + and>+ <and>	6
some	<so + som + ome + me> + <some + some + ome> + <some>	8

→ Examples of extracted ngrams in range 3-6

How are ngrams extracted?

- At length 4, each additional character adds 4 new ngrams.
  - → *Universitätsstadt* yields 62 distinct ngrams.
  - → *Eberhard-Karls-Universität* generates 98 distinct ngrams.

Isn't that a lot of ngrams?

Given a large corpus, how to accomodate all the in-vocabulary ngrams?

Hashing trick

The hashing trick

Hashing trick

fastText uses the *hashing trick* to bound memory requirements.

Hashing trick

fastText uses the *hashing trick* to bound memory requirements.

Ingredients:

- Desired number of ngram embeddings
- A fast hashing function: FNV-1a

Hashing trick

fastText uses the *hashing trick* to bound memory requirements.

Ingredients:

- Desired number of ngram embeddings
- A fast hashing function: FNV-1a

Recipe:

- Calculate hash for an ngram.
- Map hash to the ngram embedding space

Hashing trick

Consequences:

- Ngrams don't need to be explicitly stored
- Number of ngram embeddings is independent of the corpus.

Hashing trick

Consequences:

- Ngrams don't need to be explicitly stored
- Number of ngram embeddings is independent of the corpus.

But! the number of ngrams is not independent of corpus size.

$\rightarrow$ Where do the additional ngrams go?

Who is actually tricked?

Who is actually tricked?

Hashing collisions

Collisions

- FNV-1a is not a perfect hashing function.
  - → Hashing collisions happen at random.
  - → Random words share parameters.

Hashing collisions

Collisions

- FNV-1a is not a perfect hashing function.
  - → Hashing collisions happen at random.
  - → Random words share parameters.

Examples	
Hausfriedens**bruchs**	Friede**nsoppo**sition
Jawohl	Pro**fessor**
Recrui**ting-A**bteilung	P**ickel**hauben-Kompanie

→ Taken from TüBa-D/Z with 2^{21} buckets.

Unknown ngrams

Unknown ngrams

- FNV-1a has an answer for every piece of data
  - $\rightarrow$ Out-of-vocabulary ngrams get mapped to random buckets.

Unknown ngrams

Unknown ngrams

- FNV-1a has an answer for every piece of data
  - → Out-of-vocabulary ngrams get mapped to random buckets.

Known	Unknown
Tsunamei	Multim**ediav**orführungen
B**irthd**ay	H**olzpul**t
N**otst**and	Voka**lakrob**at

→ Taken from TüBa-D/Z with 2^{21} buckets.

The hashing trick in real-life

Data:

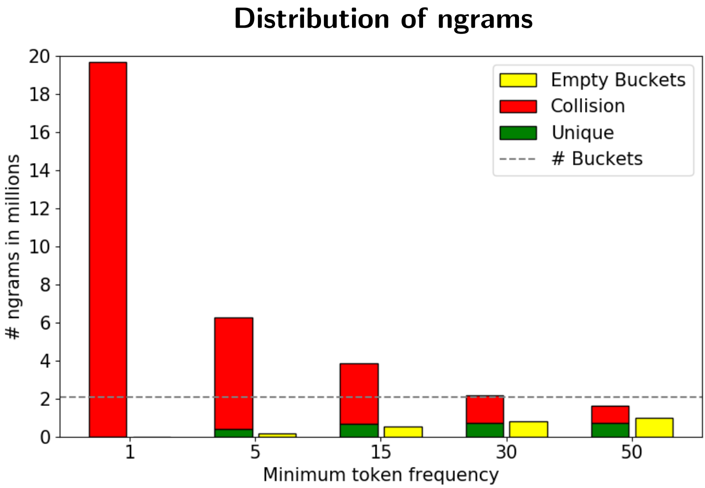
- Non-webcrawled part of TüBa-D/DP corpus (de Kok and Pütz, 2019)
  - $\rightarrow$ **TWE**: **T**AZ[1] + **W**ikipedia[2] + **E**uroparl
  - $\rightarrow$ 1.3 billion tokens, 12.9 million types
  - $\rightarrow$ 19.7 million distinct ngrams
- Only ngrams of in-vocabulary tokens are considered

Parameters:

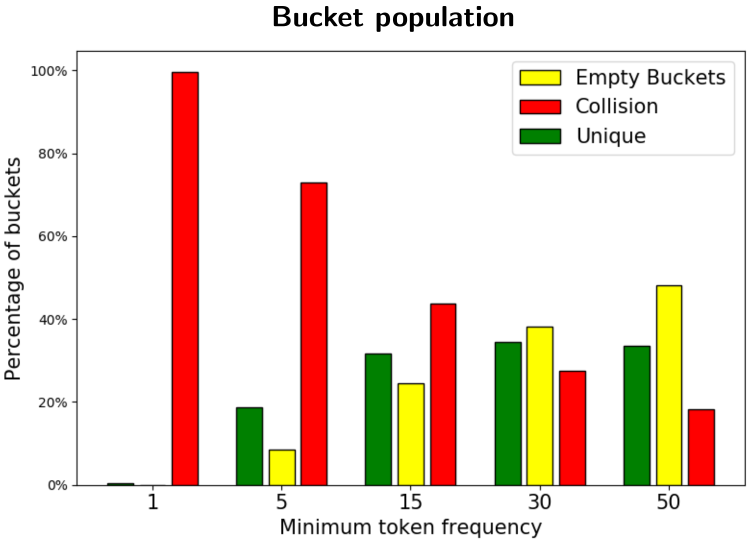
- 2^{21} buckets $\approx$ 2.1 million ngram embeddings
  - $\rightarrow$ closest power of 2 to the default fastText number

[1] 20 years of newspaper text
[2] January '19 dump

Ngram distribution



Bucket population



The hashing trick in words

- For minimum frequency 30:
  - → less than 60000 buckets are missing
  - → 38% are wasted
  - → 27% hold multiple, random ngrams
  - → 35% cleanly map to a single ngram

The hashing trick in words

- For minimum frequency 30:
  - → less than 60000 buckets are missing
  - → 38% are wasted
  - → 27% hold multiple, random ngrams
  - → 35% cleanly map to a single ngram
- TWE has 19.7 million distinct ngrams
  - → Processing the corpus means retrieving an embedding for every ngram
  - → With 2^{21} available buckets, that means more than **9** ngrams per embedding

FNV-1a?

- Is the FNV-1a algorithm just bad at hashing?

FNV-1a?

- Is the FNV-1a algorithm just bad at hashing? **No!**
  - → $\left(\frac{\mathbf{b-1}}{\mathbf{b}}\right)^{\mathbf{n}}$ given *b* buckets and *n* items is the probability of an empty bucket
  - → empirical 38% vs. expected 36%

How does that even work?

- **How does that even work?**

How does that even work?

- **How does that even work?**
  - → Random initialization is near-zero
  - → model has a lot of redundancy.

Do we need the hashing trick?

Do we need the hashing trick to bound memory?

Do we need the hashing trick?

Do we need the hashing trick to bound memory?

- On TWE with mininum frequency 30
  - $\rightarrow$ No, the number of ngram embeddings grows by only 2.7%.

Do we need the hashing trick?

Do we need the hashing trick to bound memory?

- On TWE with mininum frequency 30
  - $\rightarrow$ No, the number of ngram embeddings grows by only 2.7%.
- For larger vocabularies
  - $\rightarrow$ Filter ngrams by frequency.
  - $\rightarrow$ Set vocabulary size instead of number of buckets.

Two approaches

Two approaches to explicit storage:

1. After training, build an index with actually trained ngrams.
   - $\rightarrow$ Possibly downsizes the model.
   - $\rightarrow$ No more lies about unknown ngrams.
   - $\rightarrow$ In-vocab collisions remain.

Two approaches

Two approaches to explicit storage:

1. After training, build an index with actually trained ngrams.
   - $\rightarrow$ Possibly downsizes the model.
   - $\rightarrow$ No more lies about unknown ngrams.
   - $\rightarrow$ In-vocab collisions remain.
2. Train embeddings with a proper hash-table.
   - $\rightarrow$ No wasted space, no trade-off between collisions and space.
   - $\rightarrow$ No lies about unknown ngrams.
   - $\rightarrow$ No collisions.

Comparison: Explicit storage vs. Hashing trick

Hyperparamters	
Corpus	TWE
Dimensions	300
Context Window	10
Negative Samples	5
Ngram Range	3-6

All embeddings were trained with finalfrontier[3], using structured skipgram.

[3] https://github.com/finalfusion/finalfrontier

Quick stats

Model	# Embeddings	Size
Buckets 2^{21}	$3.28m$	$3.7GB$
Converted 2^{21}	$2.76m$	$3.2GB$
Explicit	$4.39m$	$5.5GB$
	Mincount 15	

Quick stats

Model	# Embeddings	Size
Buckets 2^{21}	$3.28m$	$3.7GB$
Converted 2^{21}	$2.76m$	$3.2GB$
Explicit	$4.39m$	$5.5GB$
Mincount 15		
Buckets 2^{20}	$1.76m$	$2.0GB$
Converted 2^{20}	$1.59m$	**1.9GB**
Buckets 2^{21}	$2.81m$	$3.2GB$
Converted 2^{21}	$2.01m$	$2.3GB$
Explicit	$2.87m$	$3.3GB$
Mincount 30		

Introspection

Introspection

Introspection

1. How much of the word representation is encoded in ngrams?
2. Do explicit ngram lookups make a difference?

Introspection

- **Known words**: average of ngrams and a distinct word vector
  $\rightarrow$ leave out distinct word vector

Introspection

- **Known words**: average of ngrams and a distinct word vector
  $\rightarrow$ leave out distinct word vector
- Compare known and OOV representation.
- Analyse similarity of known representation with most similar ngram.

Introspection

Model	OOV Sim
Buckets 2^{21}	0.991
Explicit	0.993
Mincount 15	

Introspection

Model	OOV Sim
Buckets 2^{21}	0.991
Explicit	0.993
Mincount 15	
Buckets 2^{20}	0.983
Buckets 2^{21}	0.986
Explicit	0.988
Mincount 30	

Introspection

- Virtually all information of known words is represented by ngram embeddings.
  - $\rightarrow$ Speed-space trade-off by discarding distinct word vectors possible.
    (mincount 15: $-1.1m$, mincount 30: $-0.7m$)
- Slightly more similarity with less collisions.
  - $\rightarrow$ Lower mincount has more collisions, but other interactions are at play.

Introspection

Model	Top Sim
Buckets 2^{21}	0.516
Explicit	0.563
Mincount 15	

Introspection

Model	Top Sim
Buckets 2^{21}	0.516
Explicit	0.563
Mincount 15	
Buckets 2^{20}	0.494
Buckets 2^{21}	0.532
Explicit	0.585
Mincount 30	

$\rightarrow$ Models rely more on single ngrams with clean lookups.

Extrinsic Evaluation

Extrinsic Evaluation

Evaluation

- **Tasks**
  - → Part-of-speech Tagging
  - → Dependency Parsing
- **Data**
  - → TüBa-D/Z r11 (Telljohann et al., 2004)
  - → Random splits: 70% Train, 10% Dev, 20% Val

All models were trained and evaluated with sticker[4].

[4] https://github.com/stickeritis/sticker

Part-of-speech Tagging Setup

Setup

- **Tags:** Concatenation of STTS and UD tags.
  - → Possible to retrieve either tagset by splitting.
  - → Beneficial for dependency parsing as sequence tagging.

Part-of-speech Tagging Setup

Setup

- **Tags:** Concatenation of STTS and UD tags.
  - → Possible to retrieve either tagset by splitting.
  - → Beneficial for dependency parsing as sequence tagging.
- **Model:** 3 stacked Bidirectional LSTMs with 400 hidden units and Residual Connections
- **LR:** 2000 linear warmup steps, followed by plateau scheduler
  - → Stop after 15 epochs without improvement.

Part-of-speech Tagging Results

Model	**Accuracy**
Buckets 2^{21}	99.19
Converted 2^{21}	**99.21**
Explicit	**99.21**
Minimum Count 15	

Part-of-speech Tagging Results

Model	Accuracy
Buckets 2^{21}	99.19
Converted 2^{21}	**99.21**
Explicit	**99.21**
Minimum Count 15	
Buckets 2^{21}	99.18
Converted 2^{21}	99.19
Buckets 2^{20}	99.17
Converted 2^{20}	99.18
Explicit 30	99.19
Explicit 50	99.19
Explicit 125	**99.21**
Minimum Count 30	

Part-of-speech Tagging Results

Discussion:

- Slight advantages for explicit and converted models.
- Much smaller models achieve virtually the same score.
  $\rightarrow$ *Converted* 2^{20} *is only 60% of the size of Buckets* 2^{21}
- Training with bucket embeddings and evaluating with converted embeddings hurts performance.

Dependency Parsing Setup

Setup

- **Tags:** Relative part-of-speech encoding
  - → Part-of-speeches provided through 10-fold jackknifing.
  - → UD version of TüBa-D/Z (Çöltekin et al., 2017)

Dependency Parsing Setup

Setup

- **Tags:** Relative part-of-speech encoding
  - $\rightarrow$ Part-of-speeches provided through 10-fold jackknifing.
  - $\rightarrow$ UD version of TüBa-D/Z (Çöltekin et al., 2017)
- **Model:** 3 stacked Bidirectional LSTMs with 600 hidden units and Residual Connections, input includes POS embeddings.
- **LR:** 2000 linear warmup steps, followed by plateau scheduler
  - $\rightarrow$ Stop after 15 epochs without improvement.

Dependency Parsing Setup

Setup

- **Tags:** Relative part-of-speech encoding
  - → Part-of-speeches provided through 10-fold jackknifing.
  - → UD version of TüBa-D/Z (Çöltekin et al., 2017)
- **Model:** 3 stacked Bidirectional LSTMs with 600 hidden units and Residual Connections, input includes POS embeddings.
- **LR:** 2000 linear warmup steps, followed by plateau scheduler
  - → Stop after 15 epochs without improvement.
- **Evaluation:** Punctuation is discarded.

Dependency Parsing Results

Model	LAS	UAS
Buckets 2^{21}	93.50	94.89
Converted 2^{21}	93.55	94.91
Explicit	93.48	94.86
Minimum Count 15		

Dependency Parsing Results

Model	LAS	UAS
Buckets 2^{21}	93.50	94.89
Converted 2^{21}	93.55	94.91
Explicit	93.48	94.86
Minimum Count 15		
Buckets 2^{21}	93.42	94.84
Converted 2^{21}	**93.56**	**94.93**
Buckets 2^{20}	93.52	94.92
Converted 2^{20}	93.54	94.91
Explicit 30	93.48	94.86
Explicit 50	**93.56**	94.88
Explicit 125	93.51	94.89
Minimum Count 30		

Dependency Parsing Results

Discussion:

- Converted models beat the corresponding Bucket models.
- 2^{20} buckets performs unexpectedly well: $+0.1$ LAS vs. 2^{21}
- Explicit lookups don't seem to offer improvements.
- No change in accuracy when using converted models to predict.

Conclusion

Conclusion:

- Subword embeddings hold almost the full information.
  - $\rightarrow$ Speed-space tradeoff by discarding word vectors.
- Only small effect on downstream models.
  - $\rightarrow$ Large capacity of the considered models might lower impact of embeddings.
  - $\rightarrow$ More collisions do not hurt performance on the given tasks.
- Converting bucket to explicit models is beneficial.
  - $\rightarrow$ Downsizes the model while improving performance.

Outlook

Outlook:

- Evaluate how explicit lookups interact with pretraining.
- Fine-tune downsized embeddings on downstream tasks.

Thank you

Thank you for your attention!

References I

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association of Computational Linguistics* 5(1):135–146.

Çağrı Çöltekin, Ben Campbell, Erhard Hinrichs, and Heike Telljohann. 2017. Converting the tüba-d/z treebank of german to universal dependencies. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*. pages 27–37.

Daniël de Kok and Sebastian Pütz. 2019. Tüba-d/dp stylebook.

References II

Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, and Greg Corrado. 2013. Efficient estimation of word representations in vector space .

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Viable dependency parsing as sequence labeling. *arXiv preprint arXiv:1902.10505* .

References III

Heike Telljohann, Erhard Hinrichs, Sandra Kübler, and Ra Kübler. 2004. The tüba-d/z treebank: Annotating german with a context-free backbone. In *In Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004*. Citeseer.

Part-of-speech Tagging Results

Model	Accuracy
Buckets 15/21	99.19
Explicit 15/15	**99.21**
Converted 15/21	**99.21**
Buckets 30/21	99.18
Converted 30/21	99.19
Buckets 30/20	99.17
Converted 30/20	99.18
Explicit 30/30	99.19
Expl 30/50	99.19
Expl 30/100	99.14
Expl 30/125	**99.21**

Dependency Parsing Results

Model	**LAS**	**UAS**
Buckets 2^{21}	93.50	94.89
Converted 2^{21}	93.55	94.91
Explicit	93.48	94.86
Minimum Count 15		

Dependency Parsing Results

Model	LAS	UAS
Buckets 2^{21}	93.50	94.89
Converted 2^{21}	93.55	94.91
Explicit	93.48	94.86
Minimum Count 15		
Buckets 2^{21}	93.42	94.84
Converted 2^{21}	**93.56**	**94.93**
Buckets 2^{20}	93.52	94.92
Converted 2^{20}	93.54	94.91
Explicit	93.48	94.86
Minimum Count 30		
Explicit 30/50	**93.56**	94.88
Explicit 30/100	93.52	94.90
Explicit 30/125	93.51	94.89