Efficient, Compositional, Order-Sensitive n-gram Embeddings

Adam Poliak[*] and **Pushpendre Rastogi**[*] and **M. Patrick Martin** and **Benjamin Van Durme**
Johns Hopkins University
{azpoliak,vandurme}@cs.jhu.edu, {pushpendre,mmart152}@jhu.edu

Abstract

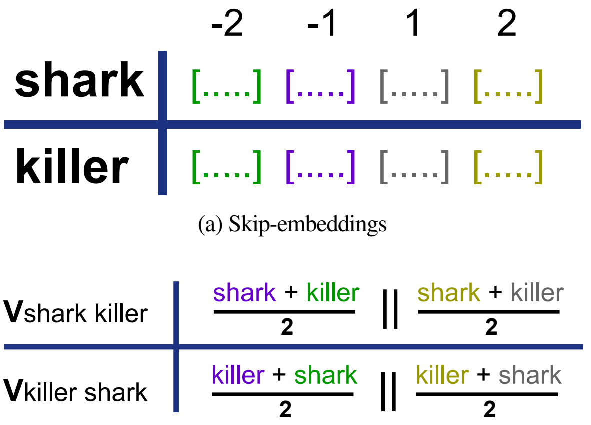
We propose *ECO*: a new way to generate embeddings for phrases that is **E**fficient, **C**ompositional, and **O**rder-sensitive. Our method creates decompositional embeddings for words offline and combines them to create new embeddings for phrases in real time. Unlike other approaches, *ECO* can create embeddings for phrases not seen during training. We evaluate *ECO* on supervised and unsupervised tasks and demonstrate that creating phrase embeddings that are sensitive to word order can help downstream tasks.

1 Introduction

Semantic embeddings of words represent word meaning via a vector of real values (Deerwester et al., 1990). The Word2Vec models introduced by Mikolov et al. (2013a) greatly popularized this semantic representation method and since then improvements to the basic Word2Vec model have been proposed (Levy and Goldberg, 2014; Ling et al., 2015).

Although techniques exist to sufficiently induce representations of single tokens (Mikolov et al., 2013a; Pennington et al., 2014), current methods for creating n-gram embeddings are far from satisfactory. Recent approaches cannot embed n-grams that do not appear during training. For example, Hill et al. (2016) used a heuristic of converting phrases to tokens before learning the embeddings. Additionally, Yin and Schütze (2014) queried sources to determine which phrases to embed.

We propose a new method for creating phrase embeddings on-the-fly. Offline, we compute decomposed word embeddings (Figure 1a) that can be used online to **E**fficiently generate **C**ompositional n-gram embeddings that are sensitive to word **O**rder (Figure 1b). We refer to our method as *ECO*. *ECO* is a novel way to incorporate knowledge about phrases into machine learning tasks. We evaluate our method on different supervised and unsupervised tasks.



(a) Skip-embeddings

(b) *ECO*

Figure 1: (a): Skip-embeddings for each word by generalizing Word2Vec. The numbers refer to the position, relative to the given word, that the individual skip-embedding represents. (b): *ECO*'s efficient heuristic for composing n-gram embeddings.

2 Background

Before introducing our approach for creating decomposed word embeddings to ultimately create n-gram embeddings online, we introduce our notation and provide a brief overview of the Word2Vec model.

Notation We define s to be a sequence of words and s_j to be the j^{th} word of sequence s. Let $|s|$ be the length of the sequence and let S be the set of all sequences. Additionally, let W denote an indexed set of words, w denote a generic word and w_i denote the i^{th} word of W. V and V_{out} denote indexed sets of vectors of length d corresponding to W, i.e. $v \in V$, $v_{\text{out}} \in V_{\text{out}}$, and v_w corresponds to the vector representing word $w \in W$. These two sets of vectors correspond to the input and output representations of a word as described by Mikolov et al. (2013b). The notation $[.,.)$ denotes a set of integers that contain successive integers starting from and including the

[*] denotes equal contribution.

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, pages 503–508,
Valencia, Spain, April 3-7, 2017. ©2017 Association for Computational Linguistics

left and excluding the right argument.

Word2Vec Model The popular Word2Vec model consists of four possible models: Continuous Bag-of-Words (CBOW) with hierarchical softmax or negative sampling, and Skip-Gram (SG) with the same choices for optimizing training parameters. CBOW aims to predict a single word w surrounded by the given context while SG tries to predict the context words around w (Rong, 2014). The SG model maximizes the following average log-probability of the sentence averaged over the entire corpus:

$$\frac{1}{|S|}\sum_{s\in S}\frac{1}{|s|}\sum_{j}\sum_{k\in[j-c,0)\cup(0,j+c]}\log p(s_k|s_j), \qquad (1)$$

where c refers to the window size, i.e. half the size of the context. The probability of token s_k given token s_j is computed as the softmax over the inner products of the embeddings of the two tokens:

$$p(s_k|s_j)=\frac{\exp\left(\langle v^{\text{out}}_{s_k},v_{s_j}\rangle\right)}{\exp\left(\sum_{w\in W}\langle v^{\text{out}}_{w},v_{s_j}\rangle\right)}. \qquad (2)$$

3 Possible approaches to embed n-grams

Before introducing *ECO*, we present a discussion of other possible ways to combine unigram embeddings to generate n-gram embeddings. This discussion motivates the need for *ECO* and the issues that our novel approach solves.

Treat n-grams as words The simplest way to create embeddings for phrases would be to treat phrases as single words and run out-of-the-box software to embed those n-grams just like one would for single words. Implementing such an approach would just require changing how one pre-processes text and then running Word2Vec. Yin and Schütze (2014) use external sources to determine common bigrams to embed offline.

This approach can not embed unknown n-grams regardless of whether each of the n-words in the sequence appeared in a training corpus. Since this situation will often occur, especially when increasing the minimum count for words used to learn an unknown embedding, this approach is insufficient and cannot embed n-grams on-the-fly.

Combining individual word embeddings The next plausible approach to create n-gram embeddings would be to combine the individual word embeddings into one new embedding with heuristics such as averaging, adding, or multiplying the word embeddings (Mitchell and Lapata, 2010). Averaging the embeddings, which we use as a baseline for our experiments, can be viewed as

$$v_{[w_1:w_n]}=\frac{v_{w_1}+...+v_{w_n}}{n} \qquad (3)$$

where $v_{[w_1:w_n]}$ is the embedding for a phrase of size n.

However, regardless of how one combines the individual word embeddings, the ordering of words in a phrase is not captured in the new n-gram embedding. For example, with this method, the embeddings for the bigrams `shark killer` and `killer shark` would be the same. Therefore, an ordered approach is needed.

4 The *ECO* Way

We now present our strategy to eliminate the shortcomings of the previously discussed approaches and propose an intuitive method for creating n-gram embeddings.

Skip-Embeddings The Word2Vec model encodes a word w using a single embedding v_w that must maximize the log probability of the tokens that occur around it. This encourages the embedding of a word to be representative of the context surrounding it. However a careful look reveals that the context around a word can be split into multiple categories, specifically that each word has at least $2c$ contexts, one for each position in the window being considered.

Thus, we can parameterize each word w with $2c$ embeddings. For all $i\in[-c{:}c]$ such that $i\neq 0$, v^i_w encodes the context of word w at a specific position, to the left $(-)$ or right $(+)$, from w. With this strategy, instead of having one model with the objective function from (1), we now have $2c$ independent models with their own objective function of

$$\frac{1}{|S|}\sum_{s\in S}\frac{1}{|s|}\sum_{j}\log p(s_k|s_j) \qquad (4)$$

where s_k is the word i positions away from s_j in s. The new probability distribution is now

$$p(s_k|s_j)=\frac{\exp\left(\langle v^{i_{\text{out}}}_{s_k},v^{i}_{s_j}\rangle\right)}{\exp\left(\sum_{w\in W}\langle v^{i_{\text{out}}}_{w},v^{i}_{s_j}\rangle\right)}. \qquad (5)$$

We refer to each of the newly decompositional $2c$ embeddings created per word as skip-embeddings.

Since a skip-embedding only considers a single token separated by i tokens from w, the dimensionality of v^i_w should be kept to $\frac{d}{2c}$ to allow for direct comparison to Word2Vec that uses d dimensional embeddings. Consequently, each skip-embedding is trained with only $\frac{d}{2c}$ parameters.

Another major benefit of this architecture is that the training can run in parallel, since the $2c$ skip embeddings are generated independently. As evidenced in section 5.2, our approach does not sacrifice quality in single word embeddings.

Combining Skip-Embeddings After creating skip-embeddings offline, we are ready to embed n-grams on-the-fly, regardless of whether a n-gram appeared in the original training corpus. Although we could concatenate the $2c$ embeddings to create a unigram embedding, instead of creating n-grams embeddings, we average the position specific skip-embeddings of words to create two vectors $v^{\mathbf{L}}_{[w_1:w_n]}$ and $v^{\mathbf{R}}_{[w_1:w_n]}$ that summarize the left and the right context of the n-gram independently. $v^{\mathbf{L}}_{[w_1:w_n]}$ and $v^{\mathbf{R}}_{[w_1:w_n]}$ are computed as follows:

$$v^{\mathbf{L}}_{[w_1:w_n]} = \frac{v^{-1}_{w_1} + ... + v^{n}_{w_n}}{n} \qquad (6)$$

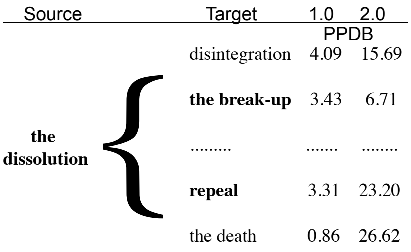
$$v^{\mathbf{R}}_{[w_1:w_n]} = \frac{v^{-n}_{w_1} + ... + v^{-1}_{w_n}}{n} \qquad (7)$$

We then concatenate $v^{\mathbf{L}}_{[w_1:w_n]}$ and $v^{\mathbf{R}}_{[w_1:w_n]}$ to create a single embedding of the entire n-gram. After concatenation, the dimensionality of a *ECO* n-gram embedding is $\frac{d}{c}$.

5 Experiments

Our proposed method decomposes previous word embedding work into $2c$ models as explained in (4) and uses an order-sensitive heuristic (6) (7) to combine skip-embeddings to embed n-grams. Our experiments demonstrate that this novel method retains more semantic meaning than other approaches. We evaluate our n-gram embeddings through both supervised and unsupervised tasks to test how well our technique embeds phrases and words.

Data We extracted over 111 million sentences[1] consisting of over 2 billion words of raw text from English Wikipedia (Ferraro et al., 2014) and ran our *ECO* framework[2] to create skip-embeddings for each word that appeared at least five times in the text. We also ran out-of-the-box Word2Vec on the English Wikipedia dataset as a baseline for *ECO*. For both Word2Vec and *ECO* embeddings, we chose c from $\{2{,}5\}$ and d from $\{100{,}500{,}700\}$. Hill et al. (2016) argue that a dimensionality of 500 is a sufficient comporimise between qualitiy and memory constraints and additionally claim that Faruqui et al. (2015)'s experiments suggest that a dimensionality of 700 yield the best results.



Source	Target	PPDB 1.0	PPDB 2.0
the dissolution	disintegration	4.09	15.69
	the break-up	3.43	6.71
	....	....	....
	repeal	3.31	23.20
	the death	0.86	26.62

Figure 2: Illustration of phrase similarity evaluation data. Bold phrases represent the pair of target phrases that were randomly sampled.

5.1 Phrase Similarity

We compare similarities between source and target phrases extracted from the paraphrase database (PPDB). To create our evaluation set of source and a pair of corresponding target phrases, we randomly sampled source phrases from PPDB that had at least two corresponding target phrases in the database. We then randomly sampled two target phrases for each source phrase (bolded in the figure above). For each tuple consisting of a source phrase and two target phrases, we manually chose which target phrase best captured the meaning of the source phrase or whether both target phrases have the same meaning. This became our gold data. Our evaluation set consists of 279 source phrases: 137 source phrases from PPDB's extra-extra-large phrasal subset and 142 source phrases from PPDB's extra-extra-large lexical subset[3]. Figure 2 illustrates an example from our evaluation dataset.

We use our proposed model to embed the source and target phrases. If the absolute difference between cosine similarities is less than .01, we count the two target phrases as having the same meaning. Otherwise, we choose the target phrase whose embedding had a higher cosine similarity with the embedding of the source phrase. We compare our results with the PPDB1.0 (Ganitkevitch et al., 2013) and PPDB2.0 (Pavlick et al., 2015) similarity scores and the cosine similarity scores computed by the naive approach

[1] We removed sentences with less than 4 tokens.

[2] The code and datasets developed are available at `https://github.com/azpoliak/eco`

[3] `http://nlpgrid.seas.upenn.edu/PPDB/eng/ppdb-2.0-xxxl-{lexical,phrasal}.gz`

		PPDB		p=100 w=2		p=100 w=5		p=500 w=2		p=500 w=5		p=700 w=2		p=700 w=5	
	MAJ	1.0	2.0	W2V	*ECO*	W2V	*ECO*	W2V	*ECO*	W2V	*ECO*	W2V	*ECO*	W2V	*ECO*
LEXICAL	43.00	23.24	57.04	54.74	**58.39**†	54.01	**56.20**	55.47	**60.58**†	**56.20**	55.47	**56.93**	55.47	**56.20**	**56.20**
PHRASAL	36.50	24.09	46.72	46.48	**56.34**†	47.89†	**48.59**†	50.70†	**52.82**†	51.41†	**56.34**†	47.18†	**54.93**†	47.89†	**52.82**†
ALL	39.78	24.37	52.33	50.54	**57.35**†	50.90	**52.33**	53.05†	**56.63**†	53.76†	**55.91**†	**55.20**†	**55.20**†	51.97	**54.48**†

Table 1: Accuracy on phrase ranking evaluation. p refers to the number of parameters used to create the word embeddings. **w** refers to window size. W2V refers to word2vec. The best system's scores are in boldface. †denotes improvement to the PPDB2.0 baseline. MAJ refers to the majority choice.

as discussed in section 3. The accuracies reported in Table 1 demonstrate that *ECO* captures semantics on n-grams better than the baseline approach. In all of the configurations, *ECO* outperforms Word2Vec for phrases that are longer than one word.

5.2 Word Embedding Similarity

Although *ECO*'s primary goal is to create n-gram embeddings, it is important for our approach to not sacrifice quality in single word embeddings. Thus, we compare our word embeddings to seven word similarity benchmarks provided by Faruqui and Dyer (2014)'s online system. To evaluate how well *ECO* embeds unigrams, we concatenate v_w^{-1} and v_w^1 for the 5629 words provided by Faruqui and Dyer (2014) and upload our *ECO* word embeddings to Faruqui and Dyer (2014)'s website[4]. We also upload the embeddings we generate by running Word2Vec as our baseline. The scores reported in Table 2 suggest that as the number of parameters increase, *ECO* better retains information for word embeddings than Word2Vec.

		Word2Vec				*ECO*			
		100		700		100		700	
Acronym	Size	2	5	2	5	2	5	2	5
WS-353-SIM	203	0.685	0.696	0.711	0.692	0.611	0.507	**0.725**	0.696
WS-353-REL	252	0.458	**0.478**	0.431	0.444	0.312	0.226	0.430	0.367
MC-30	30	0.659	0.709	0.630	0.664	0.593	0.582	**0.719**	0.710
Rare-Word	2034	0.289	0.306	0.331	0.309	0.307	0.241	**0.360**	0.346
MEN	3000	0.588	0.611	0.591	**0.618**	0.472	0.339	0.542	0.545
YP-130	130	0.206	0.208	0.175	0.246	**0.212**	0.072	**0.186**	0.197
SimLex-999	999	0.305	0.300	**0.363**	0.358	0.228	0.170	0.353	0.320

Table 2: Word Embedding similarity scores form `wordvectors.org`. The left half reports the Word2Vec scores and the right half reports the *ECO* scores. We bold the scores of the best configuration in each row.

5.3 Supervised Scoring Model

Unlike the original paraphrase ranking heuristic, Pavlick et al. (2015) rank paraphrases in a supervised setting. They solicit annotators to rank phrase similarities on an 5-point Likert scale and used a set of 209 features to train a regression. Using their data and features, we add phrase embeddings to the feature set. The scores reflect correlation with human judgements as measured by Spearman's ρ. When using only the features from Pavlick et al. (2015), we report a score of 0.7025. Due to run time constraints, we only include Word2Vec and *ECO* embeddings where $d = 100$. With a window size of 2, *ECO*'s score is 0.729 and Word2Vec's score is 0.622. When increasing the window size to 5, *ECO* scores 0.7156 and Word2Vec's ρ is 0.569. Our results suggest that these features can be useful in improving the quality of existing PPDB resources.

6 Previous work

Due to the popularity of word embeddings and the boost they have provided in supervised (Le and Mikolov, 2014) and unsupervised (Lin et al., 2015) NLP tasks, recent work has focused on how to properly embed sentences and phrases. Yin and Schütze (2014)'s method is similar to the method discussed in Section 3. They use Wiktionary and WordNet to determine the most common bigrams and create embeddings for those. Hill et al. (2016) use reverse dictionaries to determine which phrases define single words and use neural language models to learn a mapping between the phrases and word vectors. Both of these approaches can not generate embeddings for phrases on the fly and require an external corpus.

Recent work has also focused on capturing word order in embeddings. While Yuan et al. (2016) are not concerned with embedding phrases, they point out issues with concatenating or averaging standard word embeddings. They train an LSTM to appropriately incorporate word vectors in the Word Sense Disambiguation task. Their model is sensitive to word order when determining the sense of a specific word. Yuan et al. (2016)'s approach is more computationally intensive than *ECO*. Le and Mikolov (2014)'s Paragraph Vector framework also focus on capturing word order in their embeddings. However, our method is more efficient since *ECO* does not require training the n-gram embeddings.

[4] `http://wordvectors.org/`

Ling et al. (2015)'s work on structured Word2Vec is most similar to ours. However, instead of decomposing Word2Vec into $2c$ models with the same number of parameters, Ling et al. (2015) combine the contexts into one large model, creating a single model with $2c$ parameters. Even though Ling et al. (2015) incorporate positional information into the Word2Vec models, their approach cannot be used to create efficient, compositional, and order-sensitive n-gram embeddings.

7 Conclusion

We investigated a general view of Word2Vec based upon creating multiple separate, skip-embeddings per word, where each skip-embedding is individually much smaller in size in comparison to the single Word2Vec word embedding. Our method allows us to efficiently compose embeddings for n-grams that were not seen during training of the skip-embeddings while maintaining order sensitivity. Our experiments also demonstrated that averaging skip-embeddings for creating n-gram embeddings that preserve order-sensitive information is useful for NLP tasks while using the same number of parameters as the word2vec method. In comparison to previous approaches (Le and Mikolov, 2014; Yuan et al., 2016), our method is computationally efficient. This tradeoff between efficiency, both in terms of the number of parameters stored and learnt, computations performed, and order sensitivity is unique to our proposed model.

In future work, we will investigate other heuristics for combining skip-embeddings into n-gram embeddings. Additionally, we hope to use similar techniques as *ECO* to embed full sentences and documents in real time. Finally, we plan to explore tensor factorization methods (Cotterell et al., 2017) to incorporate morphology, syntactic relations, and other linguistic structures into *ECO* n-gram embeddings.

Acknowledgements

We thank Courtney Napoles for assistance with the PPDB dataset and Ellie Pavlick for providing the feature set for the PPDB evaluation. We also thank colleagues, particularly Huda Khayrallah and David Russel, and anonymous reviewers for helpful discussion and feedback. This work was supported in part by the JHU Human Language Technology Center of Excellence (HLTCOE), and DARPA DEFT, agreement number FA8750-13-2-001. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes. The views and conclusions contained in this publication are those of the authors and should not be interpreted as representing official policies or endorsements of DARPA or the U.S. Government.

References

Ryan Cotterell, Adam Poliak, Benjamin Van Durme, and Jason Eisner. 2017. Explaining and generalizing skip-gram through exponential family principal component analysis. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, Spain, April.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Manaal Faruqui and Chris Dyer. 2014. Community evaluation and exchange of word vectors at wordvectors.org. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 19–24, Baltimore, USA, June. Association for Computational Linguistics.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2015. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615, Denver, Colorado, May–June. Association for Computational Linguistics.

Francis Ferraro, Max Thomas, Matthew R. Gormley, Travis Wolfe, Craig Harman, and Benjamin Van Durme. 2014. Concretely Annotated Corpora. In *4th Workshop on Automated Knowledge Base Construction (AKBC)*, Montreal, Canada, December. Advances in Neural Information Processing Systems 27.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, Atlanta, Georgia, June. Association for Computational Linguistics.

Felix Hill, KyungHyun Cho, Anna Korhonen, and Yoshua Bengio. 2016. Learning to understand phrases by embedding the dictionary. *Transactions of the Association for Computational Linguistics*, 4:17–30.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1188–1196, Beijing, China, June.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27*, pages 2177–2185. Montreal, Canada, December.

Chu-Cheng Lin, Waleed Ammar, Chris Dyer, and Lori Levin. 2015. Unsupervised pos induction with word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1311–1316, Denver, Colorado, May–June. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W 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*, pages 1299–1304, Denver, Colorado, May–June. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, Lake Tahoe, Nevada, USA, December.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1439.

Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2015. Ppdb 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 425–430, Beijing, China, July. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October. Association for Computational Linguistics.

Xin Rong. 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Wenpeng Yin and Hinrich Schütze. 2014. An exploration of embeddings for generalized phrases. In *Proceedings of the ACL 2014 Student Research Workshop*, pages 41–47, Baltimore, Maryland, USA, June. Association for Computational Linguistics.

Dayu Yuan, Julian Richardson, Ryan Doherty, Colin Evans, and Eric Altendorf. 2016. Semi-supervised word sense disambiguation with neural models. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1374–1385, Osaka, Japan, December. The COLING 2016 Organizing Committee.