A Simple and Efficient Method to Generate Word Sense Representations

Luis Nieto Piña and **Richard Johansson**
Språkbanken, Department of Swedish, University of Gothenburg
Box 200, SE-40530 Gothenburg, Sweden
{luis.nieto.pina, richard.johansson}@svenska.gu.se

Abstract

Distributed representations of words have boosted the performance of many Natural Language Processing tasks. However, usually only one representation per word is obtained, not acknowledging the fact that some words have multiple meanings. This has a negative effect on the individual word representations and the language model as a whole. In this paper we present a simple model that enables recent techniques for building word vectors to represent distinct senses of polysemic words. In our assessment of this model we show that it is able to effectively discriminate between words' senses and to do so in a computationally efficient manner.

1 Introduction

Distributed representations of words have helped obtain better language models (Bengio et al., 2003) and improve the performance of many natural language processing applications such as named entity recognition, chunking, paraphrasing, or sentiment classification (Turian et al., 2010; Socher et al., 2011; Glorot et al., 2011). Recently, the Skip-gram model (Mikolov et al., 2013a; Mikolov et al., 2013b) was proposed, which is able to produce high-quality representations from large collections of text in an efficient manner.

Despite the achievements of distributed representations, polysemy or homonymy are usually disregarded even when word semantics may have a large influence on the models. This results in several distinct senses of one same word sharing a representation, and possibly influencing the representations of words related to those distinct senses under the premise that similar words should have similar representations. Some recent attempts to address this issue are mentioned in the next section.

We present a simple method for obtaining sense representations directly during the Skip-gram training phase. It differs from most previous approaches in that it does not need to create or maintain clusters to discriminate between senses, leading to a significant reduction in the model's complexity. It also uses a heuristic approach to determining the number of senses to be learned per word that allows the model to use knowledge from lexical resources but also to keep its ability to work withouth them. In the following sections we look at previous work, describe our model, and inspect its results in qualitative and quantitative evaluations.

2 Related Work

One of the first steps towards obtaining word sense embeddings was that by Reisinger and Mooney (2010). The authors propose to cluster occurrences of any given word in a corpus into a fixed number K of clusters which represent different word usages (rather than word senses). Each word's is thus assigned multiple prototypes or embeddings.

Huang et al. (2012) introduced a neural language model that leverages sentence-level and document-level context to generate word embeddings. Using Reisinger and Mooney (2010)'s approach to generate multiple embeddings per word via clusters and training on a corpus whose words have been substituted by its associated cluster's centroid, the neural model is able to learn multiple embeddings per word.

Neelakantan et al. (2014) tried to expand the Skip-gram model (Mikolov et al., 2013a; Mikolov et al., 2013b) to produce word sense embeddings using the clustering approach of Reisinger and Mooney (2010) and Huang et al. (2012). Notably, Skip-gram's architecture allows the model to, given a word and its context, select and train a word sense embedding jointly. The authors

Proceedings of Recent Advances in Natural Language Processing, pages 465–472,
Hissar, Bulgaria, Sep 7–9 2015.

also introduced a *non-parametric* variation of their model which allows a variable number of clusters per word instead of a fixed K.

Also based on the Skip-gram model, Chen et al. (2014) proposed to maintain and train context word and word sense embeddings conjunctly, by training the model to predict both the context words and the senses of those context words given a target word. To avoid using cluster centroids to represent senses, the number of sense embeddings per word and their initial values are obtained from a knowledge network.

Our system for obtaining word sense embeddings also builds upon the Skip-gram model (which is described in more detail in the next section). Unlike most of the models described above, we do not make use of clustering algorithms. We also allow each word to have its own number of senses, which can be obtained from a dictionary or using any other heuristic suitable for this purpose. These characteristics translate into *a*) little overhead calculations added on top of the initial word-based model; and *b*) an efficient use of memory, as the majority of words are monosemic.

3 Model Description

3.1 From Word Forms to Senses

The distributed representations for word forms that stem from a Skip-gram (Mikolov et al., 2013a; Mikolov et al., 2013b) model are built on the premise that, given a certain target word, they should serve to predict its surrounding words in a text. I.e., the training of a Skip-gram model, given a target word w, is based on maximizing the log-probability of the context words of w, $c_1, \ldots, c_n$:

$$\sum_{i=1}^{n} \log p(c_i|w). \tag{1}$$

The training data usually consists of a large collection of sentences or documents, so that the role of target word w can be iterated over these sequences of words, while the context words c considered in each case are those that surround w within a window of a certain length. The objective then becomes maximizing the average sum of the log-probabilities from Eq. 1.

We propose to modify this model to include a sense s of the word w. Note that Eq. 1 equals

$$\log p(c_1, \ldots, c_n|w) \tag{2}$$

if we assume the context words c_i to be independent of each other given a target word w. The notation in Eq. 2 allows us to consider the Skip-gram as a Naïve Bayes model parameterized by word embeddings (Mnih and Kavukcuoglu, 2013). In this scenario, including a sense would amount then to adding a latent variable s, and our model's behaviour given a target word w is to select a sense s, which is in its turn used to predict n context words $c_1, \ldots, c_n$. Formally:

$$\begin{aligned} p(s, c_1, \ldots, c_n|w) = \\ p(s|w) \cdot p(c_1, \ldots, c_n|s) = \\ p(s|w) \cdot p(c_1|s) \ldots p(c_n|s). \end{aligned} \tag{3}$$

Thus, our training objective is to maximize the sum of the log-probabilities of context words c given a sense s of the target word w plus the log-probability of the sense s given the target word:

$$\log p(s|w) + \sum_{i=1}^{n} \log p(c_i|s). \tag{4}$$

We must now consider two distinct vocabularies: V containing all possible word forms (context and target words), and S containing all possible senses for the words in V, with sizes $|V|$ and $|S|$, resp. Given a pre-set $D \in \mathbb{N}$, our ultimate goal is to obtain $|S|$ dense, real-valued vectors of dimension D that represent the senses in our vocabulary S according to the objective function defined in Eq. 4.

The neural architecture of the Skip-gram model works with two separate representations for the same vocabulary of words. This double representation is not motivated in the original papers, but it stems from `word2vec`'s code[1] that the model builds separate representations for context and target words, of which the former constitute the actual output of the system. (A note by Goldberg and Levy (2014) offers some insight into this subject.) We take advantage of this architecture and use one of these two representations to contain senses, rather than word forms: as our model only uses target words w as an intermediate step to select a sense s, we only do not need to keep a representation for them. In this way, our model builds a representation of the vocabulary V, for the context words, and another for the vocabulary S of senses, which contains the actual output. Note that the

[1] `http://code.google.com/p/word2vec/`

representation of context words is only used internally for the purposes of this work, and that context words are word forms; i.e., we only consider senses for the target words.

3.2 Selecting a Sense

In the description of our model above we have considered that for each target word w we are able to select a sense s. We now explain the mechanism used for this purpose. The probability of a context word c_i given a sense s, as they appear in the model's objective function defined in Eq. 4, $p(c_i|s)$, $\forall i \in [1, n]$, can be calculated using the *softmax* function:

$$p(c_i|s) = \frac{e^{v_{c_i}^\intercal \cdot v_s}}{\sum_{j=1}^{|V|} e^{v_{c_j}^\intercal \cdot v_s}} = \frac{e^{v_{c_i}^\intercal \cdot v_s}}{Z(s)}, \quad (5)$$

where v_{c_i} (resp. v_s) denotes the vector representing context word c_i (resp. sense s), $v^\intercal$ denotes the transposed vector v, and in the last equality we have used $Z(s)$ to identify the normalizer over all context words. With respect to the probability of a sense s given a target word w, for simplicity we assume that all senses are equally probable; i.e., $p(s|w) = \frac{1}{K}$ for any of the K senses s of word w, senses(w).

Using Bayes formula on Eq. 3, we can now obtain the posterior probability of a sense s given the target word w and the context words $c_1, \ldots, c_n$:

$$p(s|c_1, \ldots, c_n, w) = \frac{p(s|w) \cdot p(c_1, \ldots, c_n|s)}{\sum_{s_k \in \text{senses}(w)} p(s_k|w) \cdot p(c_1, \ldots, c_n|s_k)} = \frac{e^{(v_{c_1} + \cdots + v_{c_n}) \cdot v_s} \cdot Z(s)^{-n}}{\sum_{s_k \in \text{senses}(w)} e^{(v_{c_1} + \cdots + v_{c_n}) \cdot v_{s_k}} \cdot Z(s_k)^{-n}}. \quad (6)$$

During training, thus, given a target word w and context words $c_1, \ldots c_n$, the most probable sense $s \in$ senses(w) is the one that maximizes Eq. 6. Unfortunately, in most cases it is computationally impractical to explicitly calculate $Z(s)$. From a number of possible approximations, we have empirically found that considering $Z(s)$ to be constant yields the best results; this is not an unreasonable approximation if we expect the context word vectors to be densely and evenly spread out in the vector space. Under this assumption, the most probable sense s of w is the one that maximizes

$$\frac{e^{(v_{c_1} + \cdots + v_{c_n}) \cdot v_s}}{\sum_{s_k \in \text{senses}(w)} e^{(v_{c_1} + \cdots + v_{c_n}) \cdot v_{s_k}}} \quad (7)$$

For each word occurrence, we propose to select and train only its most probable sense. This approach of *hard sense assignments* is also taken in Neelakantan et al. (2014)'s work and we follow it here, although it would be interesting to compare it with a *soft* updates of all senses of a given word weighted by the probabilities obtained with Eq. 6.

The training algorithm, thus, iterates over a sequence of words, selecting each one in turn as a target word w and its context words as those in a window of a maximum pre-set size. For each target word, a number K of senses s is considered, and the most probable one selected according to Eq. 7. (Note that, as the number of senses needs to be informed –using, for example, a lexicon–, monosemic words need only have one representation.) The selected sense s substitutes the target word w in the original Skip-gram model, and any of the known techniques used to train it can be subsequently applied to obtain sense representations. The training process is drafted in Algorithm 1 using Skip-gram with Negative Sampling.

Negative Sampling (Mikolov et al., 2013b), based on Noise Contrastive Estimation (Mnih and Teh, 2012), is a computationally efficient approximation for the original Skip-gram objective function (Eq. 1). In our implementation it learns the sense representations by sampling N_{neg} words from a noise distribution and using logistic regression to distinguish them from a certain context word c of a target word w. This process is also illustrated in Algorithm 1.

4 Experiments

We trained the model described in Section 3 on Swedish text using a context window of 10 words and vectors of 200 dimensions. The model requires the number of senses to be specified for each word; as a heuristic, we used the number of senses listed in the SALDO lexicon (Borin et al., 2013). Note, however, that such a resource is not vital and could be substituted by any other heuristic. E.g., a fixed number of senses per word, as Neelakantan et al. (2014) do in their parametric approach.

As a training corpus, we created a corpus of 1 billion words downloaded from Språkbanken, the Swedish language bank.[2] The corpora are distributed in a format where the text has been tokenized, part-of-speech-tagged and lemmatized.

[2] http://spraakbanken.gu.se

Algorithm 1: Selection of senses and training using Skip-gram with Negative Sampling. (Note that v_x denotes the vector representation of word/sense x.)

Input: Sequence of words $w_1, \ldots, w_N$, window size n, learning rate α, number of negative words N_{neg}
Output: Updated vectors for each sense of words w_i, $i = 1, \ldots, N$

for $t = 1, \ldots, N$ **do**
  $w = w_i$
  $K \leftarrow$ number of senses of w
  context(w) = $\{c_1, \ldots, c_n \mid c_i = w_{t+i},\ i = -n, \ldots, n,\ i \neq 0\}$
  for $k = 1, \ldots, K$ **do**
    $p_k = \frac{e^{(v_{c_1} + \cdots + v_{c_n}) \cdot v_{s_k}}}{\sum_{j=1}^{K} e^{(v_{c_1} + \cdots + v_{c_n}) \cdot v_{s_j}}}$
  $s = \arg\max_{k=1,\ldots,K}\ p_k$
  for $i = 1, \ldots, n$ **do**
    $f = \frac{1}{1+e^{v_{c_i} \cdot v_s}}$
    $g = \alpha(1 - f)$
    $\Delta = g \cdot v_{c_i}$
    $v_{c_i} = v_{c_i} + g \cdot v_s$
    for $j = 1, \ldots, N_{neg}$ **do**
      $d_j \leftarrow$ word sampled from noise distribution, $d_j \neq c_i$
      $f = \frac{1}{1+e^{v_{d_j} \cdot v_s}}$
      $g = -\alpha \cdot f$
      $\Delta = \Delta + g \cdot v_{d_j}$
      $v_{d_j} = v_{d_j} + g \cdot v_s$
    $v_s = v_s + \Delta$

Compounds have been segmented automatically and when a lemma was not listed in SALDO, we used the parts of the compounds instead. The input to the software computing the embeddings consisted of lemma forms with concatenated part-of-speech tags, e.g. *dricka*-verb for the verb 'to drink' and *dricka*-noun for the noun 'drink'.

The training time of our model on this corpus was 22 hours. For the sake of time performance comparison, we run an off-the-shelf `word2vec` execution on our corpus using the same parameterization described above; the training of word vectors took 20 hours, which illustrates the little complexity that our model adds to the original Skip-gram.

4.1 Inspection of nearest neighbors

We evaluate the output of the algorithm qualitatively by inspecting the nearest neighbors of the senses of a number of example words, and comparing them to the senses listed in SALDO.

Table 1 shows the nearest neighbor lists of the senses of two words where the algorithm has been able to learn the distinctions used in the lexicon. The verb *flyga* 'to fly' has two senses listed in SALDO: to travel by airplane and to move through the air. The adjective *öm* 'tender' also has two senses, similar to the corresponding English word: one emotional and one physical. The lists are semantically coherent, although we note that they are topical rather than substitutional; this is expected since the algorithm was applied to lemmatized and compound-segmented text and we use a fairly wide context window.

flyg 'flight'	*flaxa* 'to flap wings'
flygning 'flight'	*studsa* 'to bounce'
flygplan 'airplane'	*sväva* 'to hover'
charterplan 'charter plane'	*skjuta* 'to shoot'
SAS-plan 'SAS plane'	*susa* 'to whiz'

(a) *flyga* 'to fly'

kärleksfull 'loving'	*svullen* 'swollen'
ömsint 'tender'	*ömma* 'to be sore'
smek 'caress'	*värka* 'to ache'
kärleksord 'word of love'	*mörbulta* 'to bruise'
ömtålig 'delicate'	*ont* 'pain'

(b) *öm* 'tender'

Table 1: Examples of nearest neighbors of the two senses of two example words.

In a related example, Figure 1 shows the projections onto a 2D space[3] of the representations for the two senses of *åsna*: 'donkey' or 'slow-witted person', and those of their corresponding nearest neighbors.

For some other words we have inspected, we fail to find one or more of the senses. This is typically when one sense is very dominant, drowning out the rare senses. For instance, the word *rock*

[3]The projection was computed using `scikit-learn` (Pedregosa et al., 2011) using multidimensional scaling of the distances in a 200-dimensional vector space.

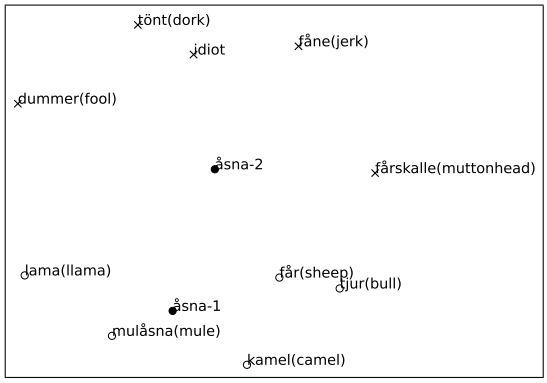

Figure 1: 2D projections of the two senses of *åsna* ('donkey' and 'slow-witted person') and their nearest neighbors.

has two senses, 'rock music' and 'coat', where the first one is much more frequent. While one of the induced senses is close to some pieces of clothing, most of its nearest neighbors are styles of music.

In other cases, the algorithm has come up with meaningful sense distinctions, but not exactly as in the lexicon. For instance, the lexicon lists two senses for the noun *böna*: 'bean' and 'girl'; the algorithm has instead created two bean senses: bean as a plant part or bean as food. In some other cases, the algorithm finds genre-related distinctions instead of sense distinctions. For instance, for the verb *älska*, with two senses 'to love' or 'to make love', the algorithm has found two stylistically different uses of the first sense: one standard, and one related to informal words frequently used in social media. Similarly, for the noun *svamp* 'sponge' or 'mushroom'/'fungus', the algorithm does not find the sponge sense but distinguishes taxonomic, cooking-related, and nature-related uses of the mushroom/fungus sense. It's also worth mentioning that when some frequent foreign word is homographic with a Swedish word, it tends to be assigned to a sense. For instance, for the adjective *sur* 'sour', the lexicon lists one taste and one chemical sense; the algorithm conflates those two senses but creates a sense for the French preposition.

4.2 Quantitative Evaluation

Most systems that automatically discover word senses have been evaluated either by clustering the instances in an annotated corpus (Manandhar et al., 2010; Jurgens and Klapaftis, 2013), or by measuring the effect of the senses representations in a downstream task such as contextual word similarity (Huang et al., 2012; Neelakantan et al., 2014). However, Swedish lacks sense-annotated corpora as well as word similarity test sets, so our evaluation is instead based on comparing the discovered word senses to those listed in the SALDO lexicon. We selected the 100 most frequent two-sense nouns, verbs, and adjectives and used them as the test set.

To evaluate the senses discovered for a lemma, we generated two sets of word lists: one derived from the lexicon, and one from the vector space. For each sense s_i listed in the lexicon, we created a list L_i by selecting the N senses (for other words) most similar to s_i according to the graph-based similarity metric by Wu and Palmer (1994). Conversely, for each sense vector v_j in our vector-based model, a list V_j was built by selecting the N vectors most similar to v_j, using the cosine similarity. We finally mapped the senses back to their corresponding lemmas, so that the two sets $L = \{L_i\}$ and $V = \{V_j\}$ of word lists could be compared.

These lists were then evaluated using standard clustering evaluation metrics. We used three different metrics:

- *Purity/Inverse-purity F-measure* (Zhao and Karypis, 2001), where each of the lexicon-based lists L_i is matched to the vector-based list V_j that maximizes the F-measure, the harmonic mean of the cluster-based precision and recall:

  $$P(V_j, L_i) = \frac{|V_j \cap L_i|}{|C_j|} \quad R(V_j, L_i) = \frac{|V_j \cap L_i|}{|L_i|}$$

  The overall F-measure is defined as the weighted average of individual F-measures:

  $$F = \sum_i \frac{|L_i|}{\sum_k |L_k|} \max_j F(V_j, L_i)$$

- *B-cubed F-measure* (Bagga and Baldwin, 1998), which computes individual precision and recall measures for every item occurring in one of the lists, and then averaging all precision and recall values. The F-measure is the harmonic mean of the averaged precision and recall.
- *V-measure* (Rosenberg and Hirschberg, 2007), the harmonic mean of the *homogeneity* and the *completeness*, two entropy-based metrics. The homogeneity is defined as the

relative reduction of entropy in V when adding the information about L:

$$h(V, L) = 1 - \frac{H(V|L)}{H(V)}$$

Conversely, the completeness is defined

$$c(V, L) = 1 - \frac{H(L|V)}{H(L)}.$$

Both measures are set to 1 if the denominator is zero.

Table 2 shows the results of the evaluation for nouns, verbs, and adjectives, and for different values of the list size N. As a strong baseline, we also include an evaluation of the sense representations discovered by the system of Neelakantan et al. (2014), run with the same settings as our system. This system is available only in its parametric version. (I.e., the number of senses per word is a fixed parameter.) As the words used in the experiments always have two senses assigned, this parameter is set to 2. This accounts for fairness in the comparison with our approach, which is given the *right* number of senses by the lexicon (and thus in this case also 2). We used the three metrics mentioned above: Purity/Inverse-purity F-measure (*Pu-F*), B-cubed F-measure (B^3-*F*), and V-measure (*V*). As we can see, our system achieves higher scores than the baseline in almost all the evaluations, despite using a simpler algorithm that uses less memory. Only for the V-measure the result is inconclusive for verbs and adjectives; for nouns, and for the other two evaluation metrics, our system is consistently better.

	Pu-F		**B^3-F**		**V**	
N	N-14	ours	N-14	ours	N-14	ours
10	9.4	**10.7**	2.5	**2.8**	8.9	**10.6**
20	9.5	**10.8**	2.1	**2.4**	6.7	**8.9**
40	9.0	**9.9**	1.8	**2.0**	5.1	**7.2**
80	7.8	**8.9**	1.4	**1.7**	4.3	**5.6**
160	7.4	**8.2**	1.3	**1.5**	3.9	**4.7**

(a) Nouns.

	Pu-F		**B^3-F**		**V**	
N	N-14	ours	N-14	ours	N-14	ours
10	9.1	**10.8**	2.0	**2.5**	**11.3**	7.6
20	8.1	**9.3**	1.4	**1.7**	6.7	**7.5**
40	7.3	**8.2**	1.0	**1.3**	4.5	**4.5**
80	7.5	**8.7**	1.0	**1.3**	**3.7**	3.2
160	8.2	**10.3**	1.2	**1.7**	1.2	**1.5**

(b) Verbs.

	Pu-F		**B^3-F**		**V**	
N	N-14	ours	N-14	ours	N-14	ours
10	6.8	**7.6**	1.4	**1.7**	9.4	**10.7**
20	6.5	**7.6**	1.3	**1.5**	**8.5**	7.2
40	6.4	**7.3**	1.1	**1.3**	5.4	**5.8**
80	6.5	**7.0**	1.0	**1.1**	**5.2**	4.7
160	6.9	**7.5**	1.0	**1.1**	4.1	**4.4**

(c) Adjectives.

Table 2: Evaluation of the senses produced by our system and that of Neelakantan et al. (2014).

5 Conclusions and Future Work

In this paper, we present a model for automatically building sense vectors based on the Skip-gram method. In order to learn the sense vectors, we modify the Skip-gram model to take into account the number of senses of each target word. By including a mechanism to select the most probable sense given a target word and its context, only slight modifications to the original training algorithm are necessary for it to learn distinct representations of word senses from unstructured text.

To evaluate our model we train it on a 1-billion-word Swedish corpus and use the SALDO lexicon to inform the number of senses associated to each word. Over a series of examples in which we analyse the nearest neighbors of some of the represented senses, we show how the obtained sense representations are able to replicate the senses defined in SALDO, or to make novel sense distinctions in others. On instances in which a sense is dominant we observe that the obtained representations favour this sense in detriment of less common ones.

We also give a quantitative evaluation of the sense representations learned by our model using a variety of clustering evaluation metrics, and compare its performance with that of the model proposed by Neelakantan et al. (2014). In most instances of this evaluation our model obtains higher scores than this baseline, despite its relative lower complexity. Our model's low complexity is characterized by *a*) the simple word sense disambiguation algorithm introduced in Section 3.2, which allows us to fit word sense embeddings into Skip-gram's existing architecture with little added computations; and *b*) the flexible number of senses per word, which takes advantage of the monosemic condition of most words to make an efficient use of memory. This low complexity is demonstrated by our training algorithm's small increase in running time with respect to that of the original, word-

based Skip-gram model.

In this work, our use of a lexicon is limited to setting the number of senses of a given word, While this information proves useful for obtaining coherent sense representations, an interesting line of research lies in further exploiting existing knowledge resources for learning better sense vectors. E.g., leveraging the network topology of a lexicon such as SALDO, that links together senses of semantically related words, could arguably help improve the representations for those rare senses with which our model currently struggles, by learning their representations taking into account those of neighbour senses in the network.

Acknowledgments

We would like to thank the reviewers for their constructive comments. This research was funded by the Swedish Research Council under grant 2013–4944, *Distributional methods to represent the meaning of frames and constructions*.

References

Amit Bagga and Breck Baldwin. 1998. Algorithms for scoring coreference chains. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pages 563–566, Granada, Spain.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Lars Borin, Markus Forsberg, and Lennart Lönngren. 2013. SALDO: a touch of yin to WordNet's yang. *Language Resources and Evaluation*, 47:1191–1211.

Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. 2014. A unified model for word sense representation and disambiguation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1025–1035.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics.

David Jurgens and Ioannis Klapaftis. 2013. SemEval-2013 task 13: Word sense induction for graded and non-graded senses. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 290–299, Atlanta, United States.

Suresh Manandhar, Ioannis Klapaftis, Dmitriy Dligach, and Sameer Pradhan. 2010. Semeval-2010 task 14: Word sense induction & disambiguation. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 63–68, Uppsala, Sweden.

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*, pages 3111–3119.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273.

Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.

Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. 2014. Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of EMNLP*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Joseph Reisinger and Raymond J Mooney. 2010. Multi-prototype vector-space models of word meaning. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 109–117. Association for Computational Linguistics.

Andrew Rosenberg and Julia Hirschberg. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural*

Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 410–420, Prague, Czech Republic.

Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics.

Zhibiao Wu and Martha Palmer. 1994. Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 133–138, Las Cruces, United States.

Ying Zhao and George Karypis. 2001. Criterion functions for document clustering: Experiments and analysis. Technical Report TR 01-040, Department of Computer Science, University of Minnesota.