Learning Sense Embeddings from Definitions in Dictionaries

Anonymous ACL submission

Abstract

We introduce a method for learning to embed word senses as defined in a given set of given dictionaries. In our approach, sense definition pairs, <*word*, *definition*> are transformed into low-dimension vectors aimed at maximizing the probability of reconstructing the definitions in an autoencoding setting. The method involves automatically training sense autoencoder for encoding sense definitions, automatically aligning sense definitions, and automatically generating embeddings of arbitrary description. At run-time, queries from users are mapped to the embedding space and re-ranking is performed on the sense definition retrieved. We present a prototype sense definition embedding, *SenseNet*, that applies the method to two dictionaries. Blind evaluation on a set of real queries shows that the method significantly outperforms a baseline based on the *Lesk algorithm*. Our methodology clearly supports combining multiple dictionaries resulting in additional improvement in representing sense definitions in dictionaries.

1 Introduction

In many natural language processing (NLP) systems, texts are represented by word embeddings, and an increasing number of methods have been proposed to embed words and senses to low-dimensional dense vectors. For example, word2vec and GloVe learn these vectors from a large corpus, while the works published by Hill et al. (2016) and Bosc and Vincent (2018) learn from dictionaries.

Word embeddings such as word2vec and GloVe typically represent each word form as a single vector. However, the vector of an ambiguous word may be dominated by its most frequent senses (Hedderich et al., 2019). Additionally, word-based reverse dictionary systems such us *OneLook* [1] and *WantWords* [2] suffer from overwhelming users with unrelated words. It would be beneficial if the systems only provide the most related words and definitions. However, the model adopted by *WantWords* map sense queries into the vector space of word embeddings (Zheng et al., 2020) instead of sense embeddings. These queries could be answered more precisely if they were mapped to and searched in the space of sense embeddings.

Consider the query *"pale brownish color like sand"* which is submitted to a reverse dictionary system. The best answer for this query is probably not only the target words *"sandy"* and *"flaxen"*, which are returned by the systems such as OneLook and WantWords, but rather the senses *"sandy: of hair color; pale yellowish to yellowish-brown"* and *"flaxen: of hair color; pale yellowish to yellowish-brown"*. A good response of such systems should not contain unrelated senses of the target words such as *"sandy: abounding in sand"* but rather the most related senses. The definition of a sense can be retrieved by embedding the sense definitions and the given query. Intuitively, by autoencoding the sense definitions, we can represent definitionay word senses (i.e., definitions) as vectors.

We present a prototype system, *SenseNet*, that automatically learns to embed definitions from multiple dictionaries into a vector space expected to reflect the semantic meaning of the senses and support sense-based NLP tasks. An example *SenseNet* session where the top 3 most relevant senses retrieved for the query *"pale brownish color like sand"* is shown in Figure 1. *SenseNet* has embedded the query in the space of the sense embeddings and find thesee neighbor senses. *SenseNet* learns this effective embeddings automatically during training by autoencoding a collection of definitions in the given dictionaries. We describe the *SenseNet* training process in more detail in Section 3.

At run-time, *SenseNet* generates effective embeddings for each word sense by training. Due

[1] https://onelook.com/reverse-dictionary
[2] https://wantwords.thunlp.org/

to the nature of encoding definitions, *SenseNet* is inherently more suitable and promising for sense-related tasks such as reverse dictionary. Alternatively, the sense embeddings can be used to disambiguate the senses of a set of synonyms and to integrate information from multiple dictionaries.

The rest of the article is organized as follows. We review the related work in the next section. Then in Section 3, we present our method for automatically learning to embed sense definitions into vectors and computing sense embeddings using these vectors. As part of our evaluation, we compare the alignment of sense definitions across two dictionaries, done using *SenseNet* with what is done using a LESK-inspired baseline over a set of random selected senses and queries (Section 4) and Section 5. Finally, we conclude with some future research directions in Section 6.

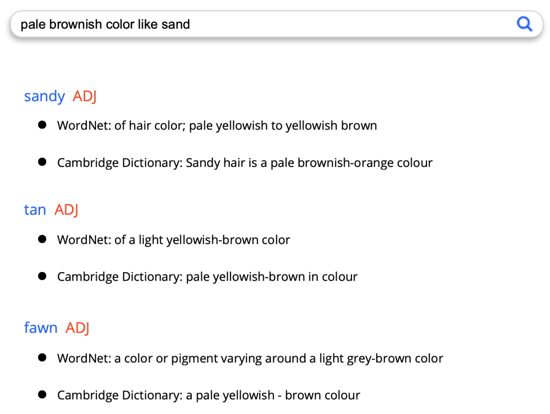

Figure 1: A screenshot of the system retrieves the senses related to the query "pale brownish color like sand".

2 Related Work

Word embedding has been an area of active research. The most influential works in word embedding research are word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014), which involve capturing semantic and even syntactic information of a word given its context. Typical word embedding methods attempt to learn the word representations in an unsupervised manner from a large corpus. Mikolov et al. (2013) proposed an influential paradigm of unsupervised learning, Skip-gram, to make the word2vec model consisting of a shallow neural network represent the word by its contexts. Training word2vec starts with randomly initialized vectors for each word in its vocabulary and uses the vectors to predict which words appear in the context window of a word. In our work we address an aspect of word embedding that has been addressed until recently.

More specifically, we focus on representing each sense of a word as different vectors. Representing senses as vectors has been become more and more active topic of word embeddings research. The body of the sense representation research most closely related to our work focuses on inducing senses and unsupervisedly learning the sense representations based on raw text corpora. (e.g., (Erk and Padó, 2008) and (Van de Cruys et al., 2011)). An interesting approach presented by (Liu et al., 2015) describes how to obtain context-sensitive word representations for each of word types by combining word embeddings and latent topics. In general, unsupervised learning of sense representation uses web corpora as training data and assumes there are underlying senses or word topics in the corpora. In contrast, we will show how to utilize dictionary definitions as a sense inventory and derive sense representations on top of word-based embeddings.

There are various NLP tasks that are context-sensitive, and hence the works utilized or provided contextual word representation, or say contextual embeddings, achieved state-of-the-art performance (Liu et al., 2020). ELMo (Peters et al., 2018) trained a language model adopting a Bi-directional LSTM (BiLSTM) to transform the fixed representation of a word into contextual embeddings with its left and right contexts. For example, consider the sentence *"These flowers generally grow on river banks and near streams."*, the representation of the word *"banks"* used here should reflect the meaning of *"sloping land"* instead of the dominant meaning *"financial institution"*. However, a sequence of words needs to be passed into the model to acquire contextual embeddings. It makes context embeddings infeasible to derive representations for chosen senses.

Recently, various pre-trained transformer-based language models have been proposed for extracting context embeddings used in downstream tasks. Devlin et al. (2018) describe a method to train BERT (Bidirectional Encoder Representations from Transformers) on two unsupervised learning tasks, masked language model (MLM) and next sentence prediction (NSP). These two tasks are used to enable BERT to *understand* natural language texts. RoBERTa (Liu et al., 2019) improves

BERT by removing NSP and applying dynamic masking to MLM on a larger and longer training corpus. Additionally, ALBERT (Lan et al., 2019) reduces the training cost of BERT by applying two parameter-reduction techniques however the time cost of inference remains the same.

In a study more closely related to our work, Lauly et al. (2014) introduce an autoencoder to learn multilingual word representations, where the autoencoder is used to reconstruct the bag-of-words of a given sentence from the encoded representations of its translation. For example, the phrase *"le chien a jappe ´"* is encoded into a vector, then the vector is passed to the autoencoder to reconstruct the bag-of-words of the phrase *"the dog barked"*. The main difference from our current work is that in Lauly et al. (2014), the reconstruced output is the translation of the input sentence, while we reconstruct the bag-of-words of the input.

More recently, Tissier et al. (2017) present a framework named *dict2vec* for automatically learning word embeddings, with the goal of gaining semantic information from dictionaries that can produce better word representations. Furthermore, Bosc and Vincent (2018) proposed a model, CPAE, with consistency penalty as part of the loss fuction to constrain the embeddings generated from dictionaries with pre-trained word embeddings. Our approach, methodology, and evaluation are substantially different. For example, while training of CPAE assumes a single word vector can capture polysemy, we address the problem of learning effective embeddings for each sense of words.

In contrast to the previous research in word embeddings and sense embeddings, we present a system that automatically learns how to embed senses in the form of definitions in multiple dictionaries, with the goal of providing effective and semantic-rich representation of word senses. We exploit inherent regularity and power of definitions in dictionaries by encoding *sense definitions* into low dimensional dense *vectors* to support sense-related NLP tasks.

3 The *SenseNet*

Representing words (e.g., "apple") as a single vector often does not work very well. Pilehvar (2019) shows that a consistent improvement can be achieved by incorporating more fine-grained representations, sense representations, into a reverse dictionary application. However, word embedding methods typically compress the semantic information of the senses of a polysemy word into a single vector. Unfortunately, the word embeddings may be dominated by the most frequent senses, leaving rare senses under-represented. Such biased embeddings may then hamper the effectiveness of the word representation as a whole. To represent a word and all its word sense, a promising approach is to automatically autoencode sense definitions in multiple dictionaries.

3.1 Problem Statement

We focus on generating sense embeddings using multiple dictionaries. These senses can be used to return relevant words and senses in response to a user query. The returned embeddings can be used to determine similar senses of a given sense directly, or passed on to sophisticated NLP systems utilizing sense embeddings. (e.g., Kartsaklis et al. (2018) and Hedderich et al. (2019)). Thus, it is crucial that each word sense (definition) is represented with a vector reflecting its meaning. At the same time, definitions in two dictionaries of the same meaning (e.g., "apple: fruit with red or yellow or green skin and sweet to tart crisp whitish flesh" in WordNet and "apple: a round fruit with firm, white flesh and a green, red, or yellow skin" in Cambridge Dictionary) should not be represented with two different vectors. Therefore, our goal is to return a set of sense vectors that capture the semantic meaning of the word sense definitions. We now formally state the problem that we are addressing.

Problem Statement: We are given a set of sense definitions in two dictionaries D_1 and D_2 (e.g., WordNet and Cambridge Dictionary). Our goal is to generate sense embeddings and combine D_1 and D_2. For this, we encode the definitions into vectors, and aligning the sense definitions across D_1 to D_2 based on these vectors.

In the rest of this section, we describe our solution to this problem. First, we define a strategy for aligning the sense definitions from two given dictionaries with the same meaning (Section 3.2). This strategy relies on a set of <*word, definition*> pairs (which we will describe in detail in Section 4.1) for training a sense definition autoencoder. In this section, we also describe our method for training the sense autoencoder. Finally, we show how we construct *SenseNet* works as a reverse dictionary to convert a (definition-like) user query into a vector and retrieve relevant words and sense definitions

(Section 3.3).

3.2 Learning to Transform Sense Definitions into Vectors

We attempt to find transformations from sense definitions into effective vectors that capture semantic meaning provided by the dictionaries. Our learning process is shown in Figure 2.

3.2.1 Gathering Senses from Dictionaries

In the first stage of the learning process (Step (1) in Figure 2), we gather a set of pairs of <*word, definition*> that represent senses as natural language texts defined by the dictionaries. For example, the pair <*apple, fruit with red or yellow or green skin and sweet to tart crisp whitish flesh*> is the word *apple* and a sense definition given by *WordNet*.

The input to this stage is the two lexical dictionaries, D_1 and D_2. The set of words and sense definitions provided by these dictionaries constitute the training data. The output of this stage is a set of pairs of <*word, definition*>, are shown in Table 1.

To process the definitions for the learning process, we use a tokenizer to split a definition into words.

3.2.2 Training Sense Autoencoder

In the second stage of the learning algorithm (Step (2) in Figure 2), we train a sense autoencoder to encode sense definitions into sense embeddings.

For this stage of the learning process, we use the collection of <*word, definition*> pairs gathered in the previous step. To update the model by consuming a batch of the pairs, each pair is passed to our model to compute reconstruction error, and the middle hidden states are kept as sense embeddings. We compute consistency penalty (Bosc and Vincent, 2018) using sense embeddings and pre-trained word embeddings. Finally, the loss, a weighted sum over the reconstruction error and the consistency penalty, is backpropagated to update the model parameters. We describe the procedures for updating the model with a batch of <*word, definition*> pairs derived from the previous stage. The procedures are shown in Figure 2.

In Step (1) of the procedures, we use the pre-trained word embeddings, which we will discuss in Section 4., to be the initial representations of the tokenized definitions. The word embeddings are passed to an LSTM, and we keep the final hidden states. The final hidden states are passed to a fully connected neural network with a linear activation function to obtain sense embeddings.

```
procedure TrainingStep(batchWords, batchDefs, alpha,
beta)
(1)     senseEmbeds = EncodeDefinitions(batchDefs)
(2)     re = ReconstructionError(senseEmbeds)
(3)     cp = ConsistencyPenalty(
                batchWords, senseEmbeds)
(4)     loss = alpha * re + beta * cp
(5)     updatedModel = backpropagate(loss)
(6)     return updatedModel
```

Figure 2: Steps for consuming a training batch of senses

In Step (2) of the procedures, we compute the reconstruction error by making the autoencoder generate the definitions passed to the model using the sense embeddings. Intuitively, this process ensures that the sense embeddings are relevant to the definitions.

In Step (3) of the procedures, the consistency penalty is computed by calculating the Euclidean distance between the sense embeddings and the word embeddings of the given word. Along with the training process, the sense embeddings are getting closed to the pre-trained word embeddings in the vector space. This effect is desired since we need to align the senses of the two different dictionaries in the next stage. Though sense embeddings of a given word will be pulled to the same word embeddings, each of the sense embeddings still reflects its respective meaning of sense because of the optimization of the reconstruction error.

In Step (4) of the procedures, we take a weighted sum over the reconstruction error and the consistency penalty as the loss function for the optimization of our model, which can be written as the following equation:

$$L = \alpha L_r + \beta L_c \tag{1}$$

where L_r and α is the reconstruction error and the weight to it, and L_c and β is the consistency penalty and the weight to it.

3.2.3 Aligning Sense Definitions

In the third stage of the learning algorithm (Step (3) in Figure 2), we align the sense definitions given by one of the two dictionaries to those given by the other one using the sense embeddings derived from the previous stage and a heuristic algorithm.

For each <*word, definition*> pair in the training collection, we look up the dictionaries for the part of speech (POS) tags of each sense. The raw POS

word	**definition**
apple	fruit with red or yellow or green skin and sweet to tart crisp whitish flesh
apple	a round fruit with firm, white flesh and a green, red, or yellow skin
baton	a thick, heavy stick used as a weapon by police officers

Table 1: Example of <*word, definition*> pairs for training

tags are normalized by being mapped to the universal POS tagset (Petrov et al., 2011) so that the two dictionaries share the same POS tags. As a result, we extend <*word, definition*> to <*word, POS, definition*> pairs. We then attempt to align the sense definitions provided by one of the dictionaries to the other one given a word w and a POS tag p by the heuristic algorithm describe as the following steps shown in Figure 3.

In Step (1) of the algorithm, there are two sets of sense definitions from the two dictionaries respectively. We make the larger set target sense definitions and the other set source sense definitions.

In Step (2) of the algorithm, we derive <*source sense, target sense*> pairs from the Cartesian product of the source and target sense definitions. We then compute the cosine similarities of sense embeddings of each sense definition pair.

In Step (3) of the algorithm, we sort the pairs by the similarities in decreasing order (Step (3a)). For each <*sourceSense*, *baseSense*> pair of the sorted pairs, we include a pair to aligned sense definitions (Step (3d)) if neither the source sense definition nor the target sense definition has been included (Step (3c)).

Finally, we obtain senses by processing each $< w, p >$ pair where w is the word defined by D_1 or D_2, and p is the POS tag defined by the universal POS tagset.

3.2.4 Generating Sense Embeddings

In the fourth and final stage of the learning algorithm (Step (4) in Figure 2), we generate sense embeddings to represent the senses derived from the previous stage.

We use sense embeddings to represent the sense definitions not align with other ones. However, for a sense definition aligning with the other one, there are two sense embeddings to represent the same word meaning. We address this problem by taking the average over the two sense embeddings.

3.3 Run-Time Sense Embeddings

Once the set of the senses and the sense embeddings are automatically induced and trained, in addition to providing static sense embeddings, *SenseNet* can also compute embeddings for any definition-like sentences or phrases at run time. Intuitively, we can perform reverse dictionary at a sense level using the system.

Given a definition-like query from a user, *SenseNet* then evaluates a given query using the procedure in Figure 4.

In Step (1), the system encodes the query into a vector using the sense autoencoder described in Section 3.2.2.

In Steps (2a), (2b), and (2c), for each sense induced in Section 3.2.3, we compute the cosine similarity between the query vector and the sense vector derived from Section 3.2.4.

In Step (3), we sort the senses by the similarities computed in the previous step in a decreasing order. Finally, the system can return the senses which are similar to the query.

4 Experiments

SenseNet was designed to learn sense embeddings from multiple dictionaries. As such, *SenseNet* will be trained and evaluated with the dictionaries. Furthermore, since one of the goals of *SenseNet* is to align sense definitions across the dictionaries, we evaluate *SenseNet* on the sense level. Finally, the reverse dictionary is an inherent application of *SenseNet*, we use the task as an extrinsic evaluation of the system.

In this section, we first present the details of training *SenseNet* for the evaluation (Section 4.1). Then, Section 4.2 lists the systems that we use in our comparison. Finally, Section 4.3 introduces the evaluation metrics for the performance of the systems.

4.1 Training *SenseNet*

We used a collection of approximately 203,000 <*word, definition*> pairs for training, obtained from

```
procedure AlignSensesGivenWordAndPos(d1Senses, d2Senses)
(1)     sourceSenses, baseSenses = DetermineBaseAndSource(d1Senses, d2Senses)
(2)     sensePairs = CalculateSimilarities(CartesianProduct(sourceSenses, baseSenses)
(3a)    sortedSensePairs = SortPairsBySimilarity(sensePairs)
         alignedSenses = ∅
(3b)     For each <sourceSense, baseSense> in sortedSensePairs
(3c)         If not (isAligned[sourceSense] or isAligned[baseSense])
                 isAligned[sourceSense] = True
                 isAligned[baseSense] = True
(3d)             alignedSenses += (sourceSense, baseSense)
(4)     return alignedSenses
```

Figure 3: Aligning sense definitions given a word and a POS tag

```
procedure reverseDictionary(userQuery)
(1)     queryEmbeds = SenseAutoEncoder(userQuery)
         results = ∅
(2a)     For each alignedSense in alignedSenses
(2b)         alignedSenseEmbeds = senseEmbeds[alignedSense]
(2c)         similarity = cosineSimilarity(queryEmbeds, alignedSenseEmbeds)
              results += (alignedSense, similarity)
(3)     sortedResults = sortResultsBySimilarity(results)
(4)     return sortedResults
```

Figure 4: Reverse dictionary at run time

POS tag	**# of pairs**
NOUN	83,600
VERB	37,900
ADJ	26,900
ADV	45,00
Total	153,000 (approx.)

Table 2: The Number of Training Pairs from WordNet

POS tag	**# of pairs**
NOUN	28,000
ADJ	11,000
VERB	9,100
ADV	1,400
X	500
NUM	80
PRON	80
DET	60
CCONJ	40
SYM	1
Total	50,000 (approx.)

Table 3: The Number of Training Pairs from CECD

two dictionaries, WordNet and Cambridge Dictionary. Table 1 shows a sample of the word-definition pairs. We obtained WordNet from an open-source library, NLTK (Bird et al., 2009), and obtained Cambridge Dictionary from a public website [3]. For the purpose of Section 3.2.3, we manually build a table to map the POS tags of the two dictionaries to the universal POS tagset. The number of word-definition training pairs in the collection for each of the dictionaries and universal POS tags is shown in Table 2 and Table 3. We used an open-source library, spaCy (Honnibal and Montani, 2017), to tokenize the definitions.

In developing *SenseNet*, we downloaded the word2vec word embeddings from Google [4] as the starting embeddings. As to training parameters, we set the hidden size of the LSTM to 300, the learning rate to 0.0003, the batch size to 32, the number of epochs to 50, the weight to the reconstruction error, α, to 1, and the weight to the consistency penalty, β, to 50. Most of the hyperparameters are referred to the settings presented by Bosc and Vincent (2018). We did not test hyperparameters exhaustively and further fine-tuning may improve the performance of the system. The system is based on an open-source NLP platform of deep learning, AllenNLP (Gardner et al., 2017). We perform the training with a single GPU, GeForce GTX 1080,

[3] https://dictionary.cambridge.org/dictionary/english-chinese-traditional/

[4] https://code.google.com/archive/p/word2vec/

for 50 minutes.

4.2 Systems Compared

Recall that *SenseNet* starts with a collection of word-definition pairs, and aligns senses from different dictionaries. The output of *SenseNet* is a set of senses and sense embeddings. In this study, we compared two systems with different methods to align sense definitions and perform reverse dictionary.

Aligning Sense Definitions

- **LESK** adopts Lesk algorithm (Lesk, 1986) to compute the similarity between two sense definitions when aligning sense definitions. More specifically, we transform the sense definitions into vectors by one-hot encoding and regard the inner product of these vectors as the similarities.
- **SenseNet** aligns sense definitions as we described in Section 3.2.3.

Reverse Dictionary

- **LESK** computes the similarity between a given query and a sense consisting of aligned sense definitions by taking the inner product of the one-hot encoding vector of the query and the vectors derived from performing an *"OR"* operation on the one-hot encoding vectors of the sense definitions.
- **SenseNet** performs reverse dictionary as we described in Section 3.3.

4.3 Evaluation Metrics

Methods for aligning senses need to be compared based on the quality of the induced senses. This quality can be quantified using two metrics, recall, and precision. For the evaluation of reverse dictionary, we compute two metrics, top-K precision and Mean Reciprocal Rank (MRR), to measure the relevance by inspecting the K senses returned by the various systems that we compare (Section 4.2) for each query that we consider. We will describe these four metrics in detail below.

Aligning Sense Definitions

- **recall**: The percentage of the senses from a ground truth have been induced by a system that aligns sense definitions. The recall can be defined as the following equation:

$$\text{recall} = \frac{n}{M} \tag{2}$$

  where n is the number of induced senses are in the ground truth, and M is the number of the senses in the ground truth.

- **precision**: The percentage of the senses induced by the system are in the ground truth. The precision can be defined as the following equation:

$$\text{precision} = \frac{n}{N} \tag{3}$$

  where N is the number of the senses induced by the system.

Reverse Dictionary

- **top-K precision**: The percentage of senses relevant to a query among the top K senses returned by a reverse dictionary system for the query. The top-K precision can be defined as the following equation:

$$\text{top-K precision} = \frac{n}{K} \tag{4}$$

  where n is the number of the returned senses which are relevant to the query.

- **MRR**: The average of the reciprocal rank values over all evaluated queries. The MRR can be defined as the following equation:

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_{q_i}} \tag{5}$$

  where N is the number of the evaluated queries, and r_{q_i} is the highest rank of a sense returned by a system that is judged relevant for the i-th query.

5 Evaluation Results

In this section, we report the results of the experimental evaluation using the methodology described in the previous section. First, in Section 5.1 we report the results of our evaluation of aligning sense definitions of the polysemous 12 words evaluated by Yarowsky (1992). In Section 5.2 we present the results of an extrinsic evaluation, reverse dictionary, which totaled 96 queries evaluated by a human judge.

System	Recall	Precision
SenseNet	0.81	0.88
Lesk	0.61	0.68

Table 4: Alignment recall and precision of **SenseNet** and **Lesk**

System	MTP-10	MRR
SenseNet	0.21	0.57
Lesk	0.07	0.29

Table 5: Mean top-10 precision (MTP-10) and MRR of **SenseNet** and **Lesk**

5.1 Results from the Alignment Evaluation

During the first evaluation, the 12 polysemous words were evaluated. We manually induced 115 senses as the ground truth, and automatically induced senses using the two systems for comparison.

Table 4 shows the evaluation results. As we can see, *SenseNet* outperforms the baseline, *Lesk*, by 33% on the recall and 29% on the precision. This indicates that our system can capture the semantic meaning of sense definitions so it performs better than *Lesk* which computes sense similarities on a discrete word level.

5.2 Results from the Reverse Dictionary Evaluation

We now report results from the evaluation of reverse dictionary with 96 queries randomly selected from the book, *Flip Dictionary* (Kipfer, 2001). We present an expert on computational linguistics with these queries along with the senses returned by the compared systems. The expert was asked to select the senses which are relevant to the queries.

Table 5 shows the evaluation results. We can see that *SenseNet* substantially outperforms *Lesk*. As we described in Section 4.2, *Lesk* finds the relevant senses to a query by comparing the words used in the query and the sense definitions. The significant improvements shown in Table 5 indicate that *SenseNet* is a robust sense embedding system. More specifically, *SenseNet* has successfully mapped the queries not from dictionaries to the space of sense embeddings which are trained on dictionaries.

Although embeddings generally reflect meaning better that one-hot representation based on word, word level information is still very useful in reverse dictionary application, since definitions typically were written with a core vocabulary and tends to have a high degree of consistency across dictionaries. Our methods is limited by not taking into account the actural words in a given query and definitions. A combination of vector space similarity and Lesk-like similarity might work better that our current method.

6 Conclusion and Future Work

There are many directions for future research and improvement of our system. For example, consistency penalty could be down-weight as we are computing on sense level rather than word level. Definitions in more dictionaries could be autoencoded and aligned to improve the coverage and quality of *SenseNet*. Additionally, an interesting direction to explore is creating a multilingual vector space of sense embeddings so definitions written in one language can be mapped to senses in another language. For example, the Chinese translation of *"hats worn by bishops"*, *"主教戴的帽子"*, can be mapped to the word *"bishop"* with a sense definition *"a liturgical headdress worn by bishops on formal occasions"*.

In summary, we have introduced a method for learning sense embeddings that improves the ability to represent words at a more fine-grained level using a deep learning model and an aligning algorithm. The method involves training a sense definition autoencoder, aligning sense definitions across dictionaries, generating integrated sense embeddings for more than one dictionaries, and run-time embedding any definition-like queries into vectors. We have implemented and thoroughly evaluated the method as applied to embedding and aligning sense definitions, as well as a reverse dictionary application. In extensive blind evaluations, we have shown that the method substantially outperforms the baseline of representing with one-hot vectors or on the word level.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Tom Bosc and Pascal Vincent. 2018. Auto-encoding dictionary definitions into consistent word embeddings. In *Proceedings of the 2018 Conference on*

Empirical Methods in Natural Language Processing, pages 1522–1532.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Katrin Erk and Sebastian Padó. 2008. A structured vector space model for word meaning in context. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 897–906.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Michael A Hedderich, Andrew Yates, Dietrich Klakow, and Gerard De Melo. 2019. Using multi-sense vector embeddings for reverse dictionaries. *arXiv preprint arXiv:1904.01451*.

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.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Dimitri Kartsaklis, Mohammad Taher Pilehvar, and Nigel Collier. 2018. Mapping text to knowledge graph entities using multi-sense lstms. *arXiv preprint arXiv:1808.07724*.

Barbara Ann Kipfer. 2001. *Flip Dictionary*. Writer's Digest.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Stanislas Lauly, Alex Boulanger, and Hugo Larochelle. 2014. Learning multilingual word representations using a bag-of-words autoencoder. *arXiv preprint arXiv:1401.1803*.

Michael Lesk. 1986. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2015. Learning context-sensitive word embeddings with neural tensor skip-gram model. In *Twenty-fourth international joint conference on artificial intelligence*.

Qi Liu, Matt J Kusner, and Phil Blunsom. 2020. A survey on contextual embeddings. *arXiv preprint arXiv:2003.07278*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

Jeffrey Pennington, Richard Socher, and Christopher D 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.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2011. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*.

Mohammad Taher Pilehvar. 2019. On the importance of distinguishing word meaning representations: A case study on reverse dictionary mapping. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2151–2156.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2017. Dict2vec: Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263.

Tim Van de Cruys, Thierry Poibeau, and Anna Korhonen. 2011. Latent vector weighting for word meaning in context. In *Empirical Methods in Natural Language Processing*.

David Yarowsky. 1992. Word-sense disambiguation using statistical models of roget's categories trained on large corpora. In *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*.

Lei Zheng, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. 2020. Multi-channel reverse dictionary model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 312–319.