Auto-Encoding Dictionary Definitions into Consistent Word Embeddings

Tom Bosc
Mila, Université de Montréal
tom.bosc@umontreal.ca

Pascal Vincent
Mila, Université de Montréal, CIFAR,
Facebook AI Research
pascal.vincent@umontreal.ca

Abstract

Monolingual dictionaries are widespread and semantically rich resources. This paper presents a simple model that learns to compute word embeddings by processing dictionary definitions and trying to reconstruct them. It exploits the inherent recursivity of dictionaries by encouraging consistency between the representations it uses as inputs and the representations it produces as outputs. The resulting embeddings are shown to capture semantic similarity better than regular distributional methods and other dictionary-based methods. In addition, the method shows strong performance when trained exclusively on dictionary data and generalizes in one shot.

1 Introduction

Dense, low-dimensional, real-valued vector representations of words known as *word embeddings* have proven very useful for NLP tasks (Turian et al., 2010). They can be learned as a by-product of solving a particular task (Collobert et al., 2011). Alternatively, one can pretrain generic embeddings based on co-occurence counts or using an unsupervised criterion such as predicting nearby words (Bengio et al., 2003; Mikolov et al., 2013). These methods implicitly rely on the distributional hypothesis (Harris, 1954; Sahlgren, 2008), which states that words that occur in similar contexts tend to have similar meanings.

It is common to study the relationships captured by word representations in terms of either *similarity* or *relatedness* (Hill et al., 2016). "Coffee" is related to "cup" as coffee is a beverage often drunk in a cup, but "coffee" is not similar to "cup" in that coffee is a beverage and cup is a container. Methods relying on the distributional hypothesis often capture relatedness very well, reaching human performance, but fare worse in capturing similarity and especially in distinguishing it from relatedness (Hill et al., 2016).

It is useful to specialize word embeddings to focus on either relation in order to improve performance on specific downstream tasks. For instance, Kiela et al. (2015) report that improvements on relatedness benchmarks also yield improvements on document classification. In the other direction, embeddings learned by neural machine translation models capture similarity better than distributional unsupervised objectives (Hill et al., 2014).

There is a wealth of methods that postprocess embeddings to improve or specialize them, such as retrofitting (Faruqui et al., 2014). On similarity benchmarks, they are able to reach correlation coefficients close to inter-annotator agreements. But these methods rely on additional resources such as paraphrase databases (Wieting et al., 2016) or graphs of lexical relations such as synonymy, hypernymy, and their converse (Mrkšić et al., 2017).

Rather than relying on such curated lexical resources that are not readily available for the majority of languages, we propose a method capable of improving embeddings by leveraging the more common resource of monolingual dictionaries.[1] Lexical databases such as WordNet (Fellbaum, 1998) are often built from dictionary definitions, as was proposed earlier by Amsler (1980). We propose to bypass the process of explicitly building a lexical database – during which information is structured but information is also lost – and instead directly use its detailed source: dictionary definitions. The goal is to obtain better representations for more languages with less effort.

The ability to process new definitions is also desirable for future natural language understanding systems. In a dialogue, a human might want to explain a new term by explaining it in his own words,

[1] See Appendix A for a list of online monolingual dictionaries.

and the chatbot should understand it. Similarly, question-answering systems should also be able to grasp definitions of technical terms that often occur in scientific writing.

We expect the embedding of a word to represent its meaning compactly. For interpretability purposes, it would be desirable to be able to generate a definition from that embedding, as a way to verify what information it captured. Case in point: to analyse word embeddings, Noraset et al. (2017) used RNNs to produce definitions from pretrained embeddings, manually annotated the errors in the generated definitions, and found out that more than half of the wrong definitions fit either the antonyms of the defined words, their hypernyms, or related but different words. This points in the same direction as the results of intrinsic evaluations of word embeddings: lexical relationships such as lexical entailment, similarity and relatedness are conflated in these embeddings. It also suggests a new criterion for evaluating word representations, or even learning them: they should contain the necessary information to reproduce their definition (to some degree).

In this work, we propose a simple model that exploits this criterion. The model consists of a definition autoencoder: an LSTM processes the definition of a word to yield its corresponding word embedding. Given this embedding, the decoder attempts to reconstruct the bag-of-words representation of the definition. Importantly, to address and leverage the recursive nature of dictionaries – the fact that words that are used inside a definition have their own associated definition – we train this model with a consistency penalty that ensures proximity of the embeddings produced by the LSTM and those that are used by the LSTM.

Our approach is self-contained: it yields good representations when trained on nothing but dictionary data. Alternatively, it can also leverage existing word embeddings and is then especially apt at specializing them for the similarity relation. Finally, it is also extremely data-efficient, as it permits to create representations of new words in one shot from a short definition.

2 Model

2.1 Setting and motivation

We suppose that we have access to a dictionary that maps words to one or several definitions. Definitions themselves are sequences of words. Our

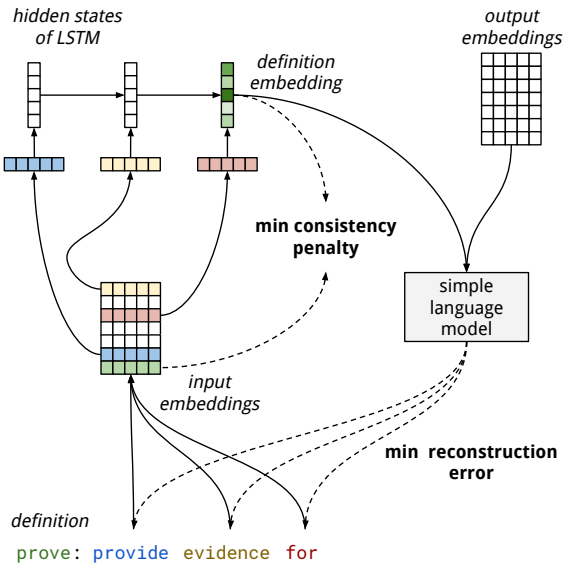

Figure 1: **Overview of the CPAE model.**

training criterion is built on the following principle: we want the model to be able to recover the definition from which the representation was built. This objective should produce similar embeddings for words which have similar definitions. Our hypothesis is that this will help capture semantic similarity, as opposed to relatedness. Reusing the previous example, "coffee" and "cup" should get different representations in virtue of having very different definitions, while "coffee" and "tea" should get similar representations as they are both defined as beverages and plants.

We chose to compute a single embedding per word in order to avoid having to disambiguate word senses. Indeed, word sense disambiguation remains a challenging open problem with mixed success on downstream task applications (Navigli, 2009). Also, recent papers have shown that a single word vector can capture polysemy and that having several vectors per word is not strictly necessary (Li and Jurafsky, 2015) (Yaghoobzadeh and Schütze, 2016). Thus, when a word has several definitions, we concatenate them to produce a single embedding.

2.2 Autoencoder model

Let $\mathcal{V}^{\mathcal{D}}$ be the set of all words that are *used* in definitions and $\mathcal{V}^{\mathcal{K}}$ the set of all words that are *defined*. We let $w \in \mathcal{V}^{\mathcal{K}}$ be a word and $D_w = (D_{w,1}, \ldots, D_{w,T})$ be its definition, where $D_{w,t}$ is the index of a word in vocabulary $\mathcal{V}^{\mathcal{D}}$. We encode such a definition D_w by processing it with an

LSTM (Hochreiter and Schmidhuber, 1997).

The LSTM is parameterized by Ω and a matrix E of size $|\mathcal{V}^{\mathcal{D}}| \times m$, whose i^{th} row E_i contains an m-dimensional *input embedding* for the i^{th} word of $\mathcal{V}^{\mathcal{D}}$. These input embeddings can either be learned by the model or be fixed to a pretrained embedding. The last hidden state computed by this LSTM is then transformed linearly to yield an m-dimensional *definition embedding* h. Thus the encoder whose parameters are $\theta = \{E, \Omega, W, b\}$ computes this embedding h as

$$h = f_\theta(D_w) = W\,\mathrm{LSTM}_{E,\Omega}(D_w) + b.$$

The subsequent decoder can be seen as a conditional language model trained by maximum likelihood to regenerate definition D_w given definition embedding $h = f_\theta(D_w)$. We use a simple conditional unigram model with a linear parametrization $\theta' = \{E', b'\}$ where E' is a $|\mathcal{V}^{\mathcal{D}}| \times m$ matrix and b' is a bias vector. [2]

We maximize the log-probability of definition D_w under that model:

$$\begin{aligned}\log p_{\theta'}(D_w|h) &= \sum_t \log p_{\theta'}(D_{w,t}|h) \\ &= \sum_t \log \frac{e^{\left\langle E'_{D_{w,t}}, h\right\rangle + b'_{D_{w,t}}}}{\sum_k e^{\left\langle E'_k, h\right\rangle + b'_k}}\end{aligned} \quad (1)$$

where $\langle , \rangle$ denotes an ordinary dot product. We call E' the *output embedding* matrix. The basic autoencoder training objective to minimize over the dictionary can then be formulated as

$$J_r(\theta', \theta) = -\sum_{w \in \mathcal{V}^{\mathcal{K}}} \log p_{\theta'}(D_w|f_\theta(D_w)).$$

2.3 Consistency penalty

We introduced 3 different embeddings: a) *definition embeddings* h, produced by the definition encoder, are the embeddings we are ultimately interested in computing; b) *input embeddings* E are used by the encoder as inputs; c) *output embeddings* E' are compared to definition embeddings to yield a probability distribution over the words in the definition. We propose a soft weight-tying scheme that brings the input embeddings closer to the definition embeddings. We call this term a *consistency* penalty because its goal is to to ensure that the embeddings used by the encoder (input embeddings) and the embeddings produced by the encoder (definition embeddings) are consistent with each other. It is implemented as

$$J_p(\theta) = \sum_{w \in \mathcal{V}^{\mathcal{D}} \cap \mathcal{V}^{\mathcal{K}}} d(E_w, f_\theta(D_w))$$

where d is a distance. In our experiments, we choose d to be the Euclidian distance. The penalty is only applied to some words because $\mathcal{V}^{\mathcal{D}} \neq \mathcal{V}^{\mathcal{K}}$. Indeed, some words are defined but are not used in definitions and some words are used in definitions but not defined. In particular, inflected words are not defined. To balance the two terms, we introduce two hyperparameters $\lambda, \alpha \geq 0$ and the complete objective is

$$J(\theta', \theta) = \alpha J_r(\theta', \theta) + \lambda J_p(\theta).$$

We call the model *CPAE*, for *Consistency Penalized AutoEncoder* when $\alpha > 0$ and $\lambda > 0$ (see Figure 1).[3]

The consistency penalty is a cheap proxy for dealing with the circularity found in dictionary definitions. We want the embeddings of the words in definitions to be compositionally built from their definition as well. The recursive process of fetching definitions of words in definitions does not terminate, because all words are defined using other words. To counter that, our model uses input embeddings that are brought closer to definition embeddings and vice versa in an asynchronous manner.

Moreover, if λ is chosen large enough, then $E_w \approx f_\theta(D_w)$ after optimisation. This means that the definition embedding for w is close enough to the corresponding input embedding to be used by the encoder for producing other definition embeddings for other words. In that case, the model could enrich its vocabulary by computing embeddings for new words and consistently reusing them as inputs for defining other words.

Finally, the consistency penalty can be used to leverage pretrained embeddings and bootstrap the learning process. For that purpose, the encoder's input embeddings E can be fixed to pretrained embeddings. These provide targets to the encoder but

[2] We have tried using a LSTM decoder but it didn't yield good representations. It might overfit because the set of dictionary definitions is small. Also, using teacher forcing, we condition on ground-truth words, making it easier to predict the next words. More work is needed to address these issues.

[3] Our implementation is available at https://github.com/tombosc/cpae

also helps the encoder to produce better definition embeddings in virtue of using input embeddings that already contain meaningful information.

To summarize, the consistency penalty has several motivations. Firstly, it deals with the fact that the recursive process of building representation of words out of definitions does not terminate. Secondly, it is a way to enrich the vocabulary with new words dynamically. Finally, it is a way to integrate prior knowledge in the form of pretrained embeddings.

In order to study the two terms of the objective in isolation, we introduce two special cases. When $\lambda = 0$ and $\alpha > 0$, the model reduces to *AE* for *Autoencoder*. When $\alpha = 0$ and $\lambda > 0$, we retrieve *Hill's model*, as presented by Hill et al. (2015).[4] Hill's model is simply a recurrent encoder that uses pretrained embeddings as targets so it only makes sense in the case we use fixed pretrained embeddings.

3 Related work

3.1 Extracting lexical knowledge from dictionaries

There is a long history of attempts to extract and structure the knowledge contained in dictionaries. Amsler (1980) studies the possibility of automatically building taxonomies out of dictionaries, relying on the syntactic and lexical regularities that definitions display. One relation is particularly straightforward to identify: it is the *is a* relation that translates to hypernymy. Dictionary definitions often contain a *genus* which is the hypernym of the defined word, as well as a *differentia* which differentiates the hypernym from the defined word. For example, the word "hostage" is defined as "a prisoner who is held by one party to insure that another party will meet specified terms", where "prisoner" is the genus and the rest is the differentia.

To extract such relations, early works by Chodorow et al. (1985) and Calzolari (1984) use string matching heuristics. Binot and Jensen (1987) operate at the syntactic parse level to detect these relations. Whether based on the string representation or the parse tree of a definition, these rule-based systems have helped to create large lexical databases. We aim to reduce the manual labor involved in designing the rules and directly obtaining representations from raw definitions.

3.2 Improving word embeddings using lexical resources

Postprocessing methods for word embeddings use lexical resources to improve already trained word embeddings irrespective of how they were obtained. When it is used with fixed pretrained embeddings, our method can be seen as a postprocessing method.

Postprocessing methods typically have two terms for trading off conservation of distributional information that is brought by the original vectors with the new information from lexical resources. There are two main ways to preserve distributional information: Attract-Repel (Vulić and Mrkšić, 2017), retrofitting (Mrkšić et al., 2017) and our method control the distance between the original vector and the postprocessed vector so that the new vector does not drift too far away from the original vector. Counter-Fitting (Mrkšić et al., 2016) and dict2vec (Tissier et al., 2017) ensure that the neighbourhood of a vector in the original space is roughly the same as the neighbourhood in the new space.

Finally, methods differ by the nature of the lexical resources they use. To our knowledge, dict2vec is the only technique that uses dictionaries. Other postprocessing methods use various data from WordNet: sets of synonyms and sometimes antonyms, hypernyms, and hyponyms. For instance, Lexical Entailment Attract-Repel (LEAR) uses all of these (Vulić and Mrkšić, 2017). Other methods rely on paraphrase databases (Wieting et al., 2016).

3.3 Dictionaries and word embeddings

We now turn to the most relevant works that involve dictionaries and word embeddings.

Dict2vec (Tissier et al., 2017) combines the word2vec skip-gram objective (predicting all the words that appear in the context of a target word) with a cost for predicting related words. These related words either form *strong pairs* or *weak pairs* with the target word. Strong pairs have a greater influence in the cost. They are pairs of words that are in the neighbourhood of the target word in the

[4] It is not exactly their model as we use Euclidian distance instead of the cosine distance or the ranking loss. They also explore several variants where the input embeddings are learned, which we didn't find to produce any improvement. We haven't experimented with the ranking loss, but the cosine distance does not seem to improve over Euclidian. Finally, they also use a simple encoder that averages word vectors, which we found to be inferior.

original embedding, as well as pairs of words for which the definitions make reference to each other. Weak pairs are pairs of words where only one word appears in the definition of the other. Unlike dict2vec, our method can be used as either a standalone or a postprocessing method (when used with pretrained embeddings). It also focuses on handling and leveraging the recursivity found in dictionary definitions with the consistency penalty whereas dict2vec ignores this aspect of the structure of dictionaries.

Besides dict2vec, Hill et al. (2015) train neural language models to predict a pretrained word embedding given a definition. Their goal was to learn a general-purpose sentence encoder useful for downstream tasks. Noraset et al. (2017) propose the task of generating definitions based on word embeddings for interpretability purposes. Our model unifies these two approaches into an autoencoder. However, we have a different goal: that of creating or improving word representations. Their methods assume that pretrained embeddings are available to provide either targets or inputs, whereas our model is unsupervised, and the use of pretrained embeddings is optional.

Bahdanau et al. (2017) present a related model that produces embeddings from definitions such that it improves performance on a downstream task. By contrast our approach is used either stand-alone or as as a postprocessing step, to produce general-purpose embeddings at a lesser computational cost. The core novelty is the way we leverage the recursive structure of dictionaries.

Finally, Herbelot and Baroni (2017) also aim at learning representations for word embeddings in a few shots. The method consists of fine-tuning word2vec hyperparameters and can learn in one or several passes, but it is not specifically designed to handle dictionary definitions.

4 Experiments

4.1 Setup

We experiment on English to benefit from the many evaluation benchmarks available. The dictionary we use is that of WordNet (Fellbaum, 1998). WordNet contains graphs of linguistic relations such as synonymy, antonymy, hyponymy, etc. but also definitions. We emphasize that our method trains exclusively on the definitions and is thus applicable to any electronic dictionary.

However, in order to *evaluate* the quality of embeddings on unseen definitions, WordNet relations comes in handy: we use the sets of synonyms to split the dictionary into a train set and a test set, as explained in Section 7. Moreover, WordNet has a wide coverage and high quality, so we do not need to aggregate several dictionaries as done by Tissier et al. (2017). Finally, WordNet is explicitly made available for research purposes, therefore we avoid technical and legal difficulties associated with crawling proprietary online dictionaries.

We do not include part of speech tags that go with definitions. WordNet does not contain function words but contains homonyms of function words. We filter these out.

4.2 Similarity and relatedness benchmarks

Evaluating the learned representations is a complex issue (Faruqui et al., 2016). Indeed, different evaluation methods yield different rankings of embeddings: there is no single embedding that outperforms others on all tasks (Schnabel et al., 2015) and thus no single best evaluation method.

We focus on intrinsic evaluation methods. In particular, we study how different models trade off similarity and relatedness. We use benchmarks which consist of pairs of words scored according to some criteria. They vary in terms of annotation guidelines, number of annotators, selection of the words, etc. To evaluate our embeddings, we score each pair by computing the cosine similarity between the corresponding word vectors. Then the predicted scores and the ground truth are ranked and the correlation between the ranks is measured by Spearman's ρ. We leave aside analogy prediction benchmarks as they suffer from many problems (Linzen, 2016; Rogers et al., 2017).

We adopt one of the methods proposed by Faruqui et al. (2016) and use separate datasets for model selection. We choose the development set to be the development set of SimVerb3500 (Gerz et al., 2016) and MEN (Bruni et al., 2014), the only benchmarks with a standard train/test split.

We justified our emphasis on the similarity relation in Section 1: capturing this relation remains a challenge, and we hypothesize that dictionary data should improve representations in that respect. The model selection procedure reflects that we want embeddings specialized in similarity. To do that, we set the validation loss as a weighted mean which weights SimVerb twice as MEN.

4.3 Baselines

The objective function presented in section 2 gives us 3 different models: CPAE, AE, and Hill's model. The objective of CPAE comprises the sum of the objective of Hill's model and of AE. We compare the CPAE model to both of these to evaluate the individual contribution of the two terms to the performance. In addition, when we use external corpora to pretrain embeddings, we compare these models to dict2vec and retrofitting. The hyperparameter search is described in Appendix C.

The test benchmarks for the similarity relation includes SimLex999 (Hill et al., 2016) and more particularly SimLex333, a challenging subset of SimLex999 which contains only highly related pairs but in which similarity scores vary a lot. For relatedness, we use MEN (Bruni et al., 2014), RG (Rubenstein and Goodenough, 1965), WS353 (Finkelstein et al., 2001), SCWS (Huang et al., 2012), and MTurk (Radinsky et al., 2011; Halawi et al., 2012). The evaluation is carried out by a modified version of the Word Embeddings Benchmarks project.[5] Conveniently, all these benchmarks contain mostly lemmas, so we do not suffer too much from the problem of missing words.[6]

5 Results in the dictionary-only setting

In the first evaluation round, we train models only using a single monolingual dictionary. This allows us to check our hypothesis that dictionaries contain information for capturing the similarity relation between words.

Our baselines are regular distributional models: GloVe (Pennington et al., 2014) and word2vec (Mikolov et al., 2013). They are trained on the concatenation of defined words with their definitions. Such a formatting introduces spurious cooccurences that do not otherwise appear in free text. But these baselines are not designed for dictionaries and cannot deal with their particular structure.

We compare these models to the autoencoder model without (AE) and with (CPAE) the consistency penalty. In this setting, we cannot use Hill's model as it requires pretrained embeddings as targets. We also trained an additional CPAE model with pretrained word2vec embeddings trained on the concatenated definitions. The results are presented in Table 1.

GloVe is outperformed by word2vec by a large margin so we ignore this model in later experiments. Word2vec captures more relatedness than CPAE (+10.7 on MEN-t, +13.5 on MT, +13.2 on WS353) but less similarity than CPAE. The difference in the nature of the relations captured is exemplified by the scores on SimLex333. This subset of SimLex999 focuses on pairs of words that are very related but that can be either similar or dissimilar. On this subset, CPAE fares better than word2vec ($+13.1$).

The consistency penalty improves performance on every dataset. This penalty provides targets to the encoder, but these targets are themselves learned and change during the learning process. The exact dynamics of the system are unknown. It can be seen as a regularizer because it puts strong weight-sharing constraints on both types of embeddings. It also resembles *bootstrapping* in reinforcement learning, which consists of building estimates of values functions on top of over estimates (Sutton and Barto, 1998).

The last model is the CPAE model that uses the word2vec embeddings pretrained on the dictionary data. This combination not only equals other models on some benchmarks but outperforms them, sometimes by a large margin ($+6.3$ on SimLex999 and $+7.5$ on SimVerb3500 compared to CPAE, $+6.1$ on SCWS, $+5.4$ on MT compared to word2vec). Thus, the two kinds of algorithms are complementary through the different relationships that they capture best. The pretraining helps in two different ways, by providing quality input embeddings and targets to the encoder. The pretrained word2vec targets are already remarkably good. That is why the chosen consistency penalty coefficient selected is very high ($\lambda = 64$). The model can pay a small cost and deviate from the targets in order to encode information about the definitions.

To sum up, dictionary data contains a lot of data relevant to modeling the similarity relationship. Autoencoder based models learn different relationships than regular distributional methods. The consistency penalty is a very helpful prior and regularizer for dictionary data, as it always helps, regardless of what relationship we focus on. Finally, our model can drastically improve embed-

[5] Original project available at https://github.com/kudkudak/word-embeddings-benchmarks, modified version distributed with our code.

[6] Missing words are not removed from the dataset, but they are assigned a null vector.

	Development		Similarity			Relatedness				
	SV-d	MENd	SL999	SL333	SV-t	RG	SCWS	MENt	MT	353
GloVe	12.0	54.8	19.8	-9.1	7.8	57.5	46.8	57.0	49.4	44.4
word2vec	35.2	62.3	34.5	16.0	36.4	65.7	54.5	59.9	56.1	**61.9**
AE	34.9	42.7	35.6	26.8	32.5	64.8	50.2	42.2	38.6	41.4
CPAE ($\lambda = 8$)	42.8	48.5	39.5	29.1	34.8	67.1	54.3	49.2	42.6	48.7
CPAE-P ($\lambda = 64$)	**44.1**	**65.1**	**45.8**	**30.9**	**42.3**	**72.0**	**60.4**	**63.8**	**61.5**	61.3

Table 1: **Positive effect of the consistency penalty and word2vec pretraining.** Spearman's correlation coefficient $\rho \times 100$ on benchmarks. Without pretraining, autoencoders (AE and CPAE) improve on similarity benchmarks while capturing less relatedness than distributional methods. The consistency penalty (CPAE) helps even without pretrained targets. Our method, combined with pretrained embeddings on the same dictionary data (CPAE-P), significantly improves on every benchmark.

dings that were trained on the same data but with a different algorithm.

6 Improving pretrained embeddings

We have seen that CPAE with pretraining is very efficient. But does this result generalizes to other kind of pretraining data? To answer this question, we experiment using embeddings pretrained on the first 50 million tokens of a Wikipedia dump, as well as the entire Wikipedia dump. We compare our method to existing postprocessing methods such as dict2vec and retrofitting, which also aims at improving embeddings with external lexical resources.

Retrofitting, which operates on graphs, is not tailored for dictionary data, which consists in pairs of words along with their definitions. We build a graph where nodes are words and edges between nodes correspond to the presence of one of the words into the definition of another. Obviously, we lose word order in the process.

The results for the small corpus are presented in Table 2. By comparing Table 2 with Table 1, we see that word2vec does worse on similarity than when trained on dictionary data, but better on relatedness. Both dict2vec and retrofitting improve with regards to word2vec on similarity benchmarks and seem roughly on par. However, dict2vec fails to improve on relatedness benchmarks, whereas retrofitting sometimes improves (as in RG, MEN, and MT), sometimes equals (SCWS) and does worse (353).

We do an ablation study by comparing Hill's model and AE with CPAE. Recall that Hill's model lacks the reconstruction cost while AE lacks the consistency penalty. Firstly, CPAE always improves over AE. Thus, we confirm the results of the previous section on the importance of the consistency penalty. In that setting, it is more obvious why this penalty helps, as it now provides pretrained targets to the encoder. Secondly, CPAE improves over Hill on all similarity benchmarks by a large margin ($+12.2$ on SL999, $+13.7$ on SL333, $+16.1$ on SV3500). It is sometimes slightly worse on relatedness benchmarks (-3.3 on MEN-t, -5.6 on MT), other times better or equal. We conclude that both terms of the CPAE objective matter.

We see identical trends when using the full Wikipedia dump. As expected, CPAE can still improve over even higher quality embeddings by roughly the same margins. The results are presented in Appendix D.

Remarkably, the best model among all our experiments is CPAE in Table 1 and uses only the dictionary data. This supports our hypothesis that dictionaries contain similarity-specific information.

7 Generalisation on unseen definitions

A model that uses definitions to produce word representations is appealing because it could be extremely data-efficient. Unlike regular distributional methods which iteratively refine their representation as occurences accumulate, such a model could output a representation in one shot. We now evaluate CPAE in a setting where some definitions are not seen during training.

The dictionary is split into train, validation (for early stopping) and test splits. The algorithm for splitting the dictionary puts words in batches. It ensures two things: firstly, that words which share at least one definition are in the same batch, and secondly, that each word in a batch is associated with all its definitions. We can then group batches to build the training and the test sets such that the

	Development		Similarity			Relatedness				
	SV-d	MEN-d	SL999	SL333	SV-t	RG	SCWS	MEN-t	MT	353
word2vec	21.7	71.1	33.2	6.9	21.2	68.5	**65.8**	71.5	61.1	**65.3**
retrofitting	28.5	**71.6**	36.9	14.1	26.1	**78.9**	65.7	**74.4**	**62.8**	60.7
dict2vec	26.3	63.5	36.2	15.5	22.0	69.2	63.8	63.9	54.9	60.8
Hill	26.9	63.3	27.7	12.9	21.7	72.9	58.4	64.1	54.0	52.3
AE	33.5	47.0	33.1	20.4	32.5	66.0	52.0	46.4	40.2	43.7
CPAE ($\lambda = 4$)	**39.5**	60.8	**39.9**	**26.6**	**37.8**	69.7	59.2	60.8	48.4	55.6

Table 2: **Improving pretrained embeddings computed on a small corpus**. Spearman's correlation coefficient $\rho \times 100$ on benchmarks. All methods use pretrained embeddings. All methods (except maybe Hill) manage to improve the embeddings. Retrofitting outperforms dict2vec and efficiently specializes for relatedness. CPAE outperforms AE and allows to trade off relatedness for similarity.

	Similarity			Relatedness			
	999	333	SV-t	353	MEN	SCWS	MT
All	36.5	27.7	36.9	43.7	48.8	53.2	41.5
Train	40.0	28.5	36.7	46.9	53.4	57.1	42.9
Test	27.5	25.4	38.5	42.5	44.1	40.3	39.1

Table 3: **One pass generalisation.** Spearman's correlation coefficient $\rho \times 100$ on benchmarks. The model is CPAE (without pretrained embeddings). All: all pairs in the benchmarks. Train: pairs for which both words are in the training or validation set. Test: pairs which contain at least one word in the test set. Correlation is lower for test pairs but remains strong ($\rho > 0.3$): the model has good generalisation abilities.

test set does not contain synonyms of words from the other sets. We sort the batches by the number of distinct definitions they contain. We use the largest batch returned by the algorithm as the training set: it contains mostly frequent and polysemous words. The validation and the test sets, on the contrary, contain many multiword expressions, proper nouns, and rarer words. More details are given in Appendix B.1.

We train CPAE only on the train split of the dictionary, with randomly initialized input embeddings. Table 3 presents the same correlation coefficients as in the previous tables but also distinguishes between two subsets of the pairs: the pairs for which all the definitions were seen during training (*train*) and the pairs for which at least one word was defined in the test set (*test*). Unfortunately, there are not enough pairs of words which both appear in the test set to be able to compute significant correlations. On small-sized benchmarks, correlation coefficients are sometimes not significant so we do not report them (when p-value > 0.01).

The scores of CPAE on the test pairs are quite correlated with the ground truth: except on SimLex999 and SCWS, there is no drop in correlation coefficients between the two sets. The scores of Hill's model follow similar trends, but are lower on every benchmark so we do not report them. This shows that recurrent encoders are able to generalize and produce coherent embeddings as a function of other embeddings in one pass.

8 Conclusion and future work

We have focused on capturing the similarity relation. It is a challenging task which we have proposed to solve using dictionaries, as definitions seem to encode the relevant kind of information.

We have presented an alternative for learning word embeddings that uses dictionary definitions. As a definition autoencoder, our approach is self-contained, but it can alternatively be used to improve pretrained embeddings, and includes Hill's model (Hill et al., 2015) as a special case.

In addition, our model leverages the inherent recursivity of dictionaries via a consistency penalty, which yields significant improvements over the vanilla autoencoder.

Our method outperforms dict2vec and retrofitting on similarity benchmarks by a quite large margin. Unlike dict2vec, our method can be used as a postprocessing method which does not require going through the original pretraining corpus, it has fewer hyperparameters, and it generalises to new words.

We see several directions for future work.

Firstly, more work is needed to evaluate the representations on other languages and tasks.

Secondly, solving downstream tasks requires representations for the inflected words as well. We have set aside this issue by focusing on benchmarks involving lemmas. To address it in furture work, we might want to split word representations into a lexical and a morphological part. With such a split representation, we could postprocess only the lexical component, and all the words, whether inflected or not, would benefit from this. This seems desirable for postprocessing methods in general and would make them more suitable for synthetic languages.

Thirdly, dictionary defines every sense of words, so we could produce one embedding per sense (Chen et al., 2014) (Iacobacci et al., 2015). This requires potentially complicated modifications to our model as we would need to disambiguate senses inside each definition. However, some class of words might benefit a lot from such representations, for example words that can be used as different parts of speech.

Lastly, a more speculative direction could be to study iterative constructions of the set of embeddings. As our algorithm can generalize in one shot, we could start the training with a small set of words and their definitions and iteratively broaden the vocabulary and refine the representations without retraining the model. This could be useful in discovering a set of semantic primes from which one can define all the other words (Wierzbicka, 1996).

Acknowledgements

We thank NSERC and Facebook for financial support, Calcul Canada for computational ressources, Dzmitry Bahdanau and Stanisław Jastrzębski for contributing to the code on which the implementation is based, the developers of Theano (Theano Development Team, 2016), Blocks and Fuel (van Merriënboer et al., 2015) as well as Laura Ball for proofreading.

References

Robert Alfred Amsler. 1980. The structure of the merriam-webster pocket dictionary.

Dzmitry Bahdanau, Tom Bosc, Stanislaw Jastrzebski, Edward Grefenstette, Pascal Vincent, and Yoshua Bengio. 2017. Learning to compute word embeddings on the fly. *CoRR*, abs/1706.00286.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Jean-Louis Binot and Karen Jensen. 1987. A semantic expert using an online standard dictionary. In *Proceedings of the 10th international joint conference on Artificial intelligence-Volume 2*, pages 709–714. Morgan Kaufmann Publishers Inc.

Elia Bruni, Nam-Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *J. Artif. Intell. Res.(JAIR)*, 49(2014):1–47.

Nicoletta Calzolari. 1984. Detecting patterns in a lexical data base. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, pages 170–173. Association for Computational Linguistics.

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.

Martin S Chodorow, Roy J Byrd, and George E Heidorn. 1985. Extracting semantic hierarchies from a large on-line dictionary. In *Proceedings of the 23rd annual meeting on Association for Computational Linguistics*, pages 299–304. Association for Computational Linguistics.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Manaal Faruqui, Jesse Dodge, Sujay K. Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. 2014. Retrofitting Word Vectors to Semantic Lexicons. *arXiv:1411.4166 [cs]*. ArXiv: 1411.4166.

Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. 2016. Problems with evaluation of word embeddings using word similarity tasks. *arXiv preprint arXiv:1605.02276*.

Christiane Fellbaum. 1998. *WordNet*. Wiley Online Library.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM.

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. Simverb-3500: A large-scale evaluation set of verb similarity. *arXiv preprint arXiv:1608.00869*.

Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. 2012. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1406–1414. ACM.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Aurélie Herbelot and Marco Baroni. 2017. High-risk learning: acquiring new word vectors from tiny data. *arXiv preprint arXiv:1707.06556*.

Felix Hill, Kyunghyun Cho, Sébastien Jean, Coline Devin, and Yoshua Bengio. 2014. Embedding word similarity with neural machine translation. *CoRR*, abs/1412.6448.

Felix Hill, Kyunghyun Cho, Anna Korhonen, and Yoshua Bengio. 2015. Learning to understand phrases by embedding the dictionary. *arXiv preprint arXiv:1504.00548*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2016. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

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.

Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2015. Sensembed: Learning sense embeddings for word and relational similarity. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 95–105.

Douwe Kiela, Felix Hill, and Stephen Clark. 2015. Specializing word embeddings for similarity or relatedness. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2044–2048.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Jiwei Li and Dan Jurafsky. 2015. Do multi-sense embeddings improve natural language understanding? *EMNLP 2015*.

Tal Linzen. 2016. Issues in evaluating semantic spaces using word analogies. *arXiv preprint arXiv:1606.07736*.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. 2015. Blocks and fuel: Frameworks for deep learning. *CoRR*, abs/1506.00619.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Nikola Mrkšić, Ivan Vulić, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gašić, Anna Korhonen, and Steve Young. 2017. Semantic specialisation of distributional word vector spaces using monolingual and cross-lingual constraints. *arXiv preprint arXiv:1706.00374*.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–148, San Diego, California. Association for Computational Linguistics.

Roberto Navigli. 2009. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, 41(2):10.

Roberto Navigli and Simone Paolo Ponzetto. 2012. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250.

Thanapon Noraset, Chen Liang, Larry Birnbaum, and Doug Downey. 2017. Definition Modeling: Learning to Define Word Embeddings in Natural Language. In *AAAI*, pages 3259–3266.

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.

Olivier Picard, Alexandre Blondin-Massé, Stevan Harnad, Odile Marcotte, Guillaume Chicoisne, and Yassine Gargouri. 2009. Hierarchies in dictionary definition space. *arXiv preprint arXiv:0911.5703*.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346. ACM.

Anna Rogers, Aleksandr Drozd, and Bofang Li. 2017. The (Too Many) Problems of Analogical Reasoning with Word Vectors. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 135–148.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Magnus Sahlgren. 2008. The distributional hypothesis. *Italian journal of linguistics*, 20(1):33–54.

Tobias Schnabel, Igor Labutov, David M Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings.

Richard S Sutton and Andrew G Barto. 1998. Introduction to reinforcement learning.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2017. Dict2vec: Learning Word Embeddings using Lexical Dictionaries. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pages 254–263.

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.

Ivan Vulić and Nikola Mrkšić. 2017. Specialising word vectors for lexical entailment. *arXiv preprint arXiv:1710.06371*.

Anna Wierzbicka. 1996. *Semantics: Primes and universals: Primes and universals*. Oxford University Press, UK.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Charagram: Embedding words and sentences via character n-grams. *arXiv preprint arXiv:1607.02789*.

Yadollah Yaghoobzadeh and Hinrich Schütze. 2016. Intrinsic subspace evaluation of word embedding representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 236–246, Berlin, Germany. Association for Computational Linguistics.

A Lists of electronic dictionaries

Electronic monolingual dictionaries are rather widespread compared to machine-readable lexical databases. Wiktionary[7] is a collaborative online dictionary, where 79 languages have more than 10,000 entries and 42 languages have more than 100,000 entries. BabelNet (Navigli and Ponzetto, 2012) aligns Wikipedia and WordNet to automatically obtain definitions ("glosses" in BabelNet terminology).

Alternatively, Wikipedia provides a list of monolingual dictionaries.[8] One can also translate "dictionary" into the desired language and look up this term. For example, there are dictionaries for Friulian and Frisian, which are minority languages spoken by less than a million speakers.[9]

B Data

The dump of Wikipedia that we have mentioned in Section 6 is the dump from the 14th June 2014. Although this dump is not available anymore online, results should be replicable with newer dumps.

B.1 Split dictionary setting

We briefly describe the algorithm used to split the dictionary. If we had used a regular dictionary instead of WordNet, we could have randomly distributed words into a train, validation and test split. However, by doing so, we would have put synonyms in two different splits, which could be a weak form of a test-set leak. On the other hand, WordNet has sets of synonyms, called synsets, where two words in a synset necessarily share at least one definition.

We create batches of words which definitions do not overlap across batches with the following algorithm: First, we create maps from words to their definitions and their converse. We will create batches indexed by i, and we now describe how to build the batch i. We create a set of definitions that is initialized as a singleton containing a random definition, $C_i = \{d\}$. We instantiate another set $S_i = \emptyset$ that will contain all the words which definitions overlap with at least one other word of the set. We iterate over the set C_i and pop a definition. Then, we go through all the words that possess that definition and add them to S_i, while we also add all the other definitions of these words to C_i. We keep iterating over C_i until it is empty. Then we start again the process on a new batch with a new $C_{i+1} = \{d'\}$ where d' has never been added to a C_i before.

Then, we order the batches S_i by the number of words they contain. We choose the largest batch to be the training set. It can be seen as a large subset of the vocabulary that is "central", in a way. By construction, it contains highly polysemous words and frequent words, as shown in Table 5.

Although our construction method is very different, it is slightly similar in spirit to the grounding kernel presented by (Picard et al., 2009). They report that words in the grounding kernel are more frequent, as we do for the training set. It is also related to the concept of semantic primes, a subset of the lexicon from which all other words can be defined (Wierzbicka, 1996).

C Hyperparameter search

All embeddings are of size 300 for the small Wikipedia dump experiments and 400 for the large Wikipedia dump experiments.

C.1 GloVe

GloVe is run only on the definition corpus because we found word2vec to be superior. We have fixed the number of epochs to be 50. The window size varies between $\{5, 7, 9, 11, 15, \mathbf{20}\}$ and x_{max} in $\{0, 5, \mathbf{20}, 100\}$, where the selected hyperparameters are bolded.

C.2 Word2vec

We have used gensim implementation.[10]

On definitions, we do a hyperparameter search on the window size in $\{5, \mathbf{15}\}$ and on the downsampling threshold in $\{0, \mathbf{0.1}, 0.01, 0.001, 0.0001\}$, and we also choose from the skip-gram or CBOW variant, where the skip-gram variant is selected.

On the small training corpus, we only use the skip-gram model and choose the number of iterations in $\{5, \mathbf{30}\}$, the window size in $\{3, \mathbf{5}, 7, 10\}$, and the downsampling threshold in $\{0, 0.1, 0.01, \mathbf{0.001}, 0.0001\}$, where the selected hyperparameters are boldened.

[7] https://www.wiktionary.org/

[8] https://en.wikipedia.org/wiki/List_of_dictionaries_by_number_of_words

[9] https://taalweb.frl/wurdboekportaal http://www.arlef.it/grant-dizionari-talian-furlan/htdocs/gdbtf.pl

[10] https://radimrehurek.com/gensim/

	Development		Similarity			Relatedness				
	SV-d	MENd	SL999	SL333	SV-t	RG	SCWS	MENt	MT	353
word2vec	28.2	73.5	36.5	14.2	23.0	76.4	64.2	74.6	64.0	**68.5**
retrofitting	32.8	**74.4**	40.2	20.7	28.2	**84.0**	**65.3**	**77.7**	**65.6**	62.9
dict2vec	36.5	67.9	41.0	24.2	32.0	74.1	61.7	68.1	57.9	64.8
Hill	29.4	62.4	31.9	17.2	20.0	67.1	53.6	62.9	52.6	50.7
AE	33.0	45.6	34.7	24.1	30.3	71.5	49.3	45.5	38.4	42.5
CPAE-P ($\lambda = 16$)	**39.3**	63.1	**45.2**	**31.5**	**37.4**	69.6	59.3	60.7	50.2	58.5

Table 4: **Improving pretrained embeddings computed on a large corpus**. Spearman's correlation coefficient $\rho \times 100$ on benchmarks. Same as Table 2 but word2vec is trained on the entire Wikipedia dump. Dict2vec fails to improve. Retrofitting especially improves relatedness, while CPAE improves similarity.

	# defs	# defs / # words	Avg. counts
Train	36,722	2.87	11.40
Valid	2,109	1.81	4.07
Test	128,223	1.13	1.92

Table 5: **Statistics of the split dictionary**. By construction, the train set contains much more frequent and polysemous words than the train set. The average counts are geometric averages of the smoothed counts of words, computed on the first 50M tokens of the Wikipedia dump.

On the full training corpus, we have used the same hyperparameters except that the number of iteration is reduced to 5, and we have filtered out words that appear less than 50 times.

C.3 AE, CPAE, Hill's model

We train the AE and CPAE models with Adam (Kingma and Ba, 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, a learning rate of $3 \cdot 10^{-4}$, a batch size of 32. In the settings where we use part of the dictionary, we use early stopping: every $2,000$ batches, we compute the mean cost on the validation set and stop after 20000 batches without improvement. On the full dictionary, where we do not have a validation set, we train for 50 epochs.

CPAE and AE models always use a reconstruction cost so $\alpha = 1$. The λ parameter that weights the cost of the consistency penalty varies in $\{1, 2, 4, 8, 16, 32, 64\}$, and the value chosen by the model selection is indicated in the tables.

C.4 Retrofitting

We have used the original implementation.[11] We have tuned the hyperparameter α that controls the proximity of the retrofitted vector to the original vector by grid-search: $\alpha \in \{0.25, 0.5, 1, 2, 4\}$. The selected value is $\alpha = 0.5$ for both the small and the large corpora.

C.5 Dict2vec

We have used the original implementation.[12] Dict2vec has many hyperparameters. We fixed $K = 5$, where the K closest neighbours to each word in the original embeddings are promoted to form a strong pair, as well as $n_s = 4$ and $n_w = 5$, the number of pairs to sample. We run a grid-search over the hyperparameters, the coefficients that weight the strong and weak pairs importance in the auxiliary cost: $\beta_s \in \{0.4, 0.6, 0.8, 1.0, 1.2, 1.4\}$ and $\beta_w \in \{0.0, 0.2, 0.4, 0.6\}$. In the smaller data regime (small dump), $\beta_s = 1.2$ and $\beta_w = 0$, and in the larger data regime (the entire wikipedia dump), the model selection procedure picks $\beta_s = 0.8$ and $\beta_w = 0.2$. When focusing on the similarity relation, it seems better not to use weak pairs, or at least to weight them very low.

D Improving pretrained embeddings on the full Wikipedia dump

The scores of word2vec are not really improved by using the much larger full Wikipedia dump, so we increase the size of the embeddings from 300 to 400. The results are given in Table 4.

The trends are similar to what we have observed on the first 50 million tokens of Wikipedia. However, dict2vec seems a bit better and improves over retrofitting in similarity.

[11] https://github.com/mfaruqui/retrofitting

[12] https://github.com/tca19/dict2vec