Dict2vec : Learning Word Embeddings using Lexical Dictionaries

Julien Tissier and **Christophe Gravier** and **Amaury Habrard**
Univ. Lyon, UJM Saint-Etienne
CNRS, Lab Hubert Curien UMR 5516
F-42023, Saint-Etienne, France
{firstname.lastname}@univ-st-etienne.fr

Abstract

Learning word embeddings on large unlabeled corpus has been shown to be successful in improving many natural language tasks. The most efficient and popular approaches learn or retrofit such representations using additional external data. Resulting embeddings are generally better than their corpus-only counterparts, although such resources cover a fraction of words in the vocabulary. In this paper, we propose a new approach, Dict2vec, based on one of the largest yet refined datasource for describing words – natural language dictionaries. Dict2vec builds new word pairs from dictionary entries so that semantically-related words are moved closer, and negative sampling filters out pairs whose words are unrelated in dictionaries. We evaluate the word representation obtained using Dict2vec on eleven datasets for the word similarity task and on four datasets for a text classification task.

1 Introduction

Learning word embeddings usually relies on the distributional hypothesis – words appearing in similar contexts must have similar meanings and thus similar representations. Finding such representations for words and sentences has been one hot topic over the last few years in Natural Language Processing (NLP) (Mikolov et al., 2013; Pennington et al., 2014) and has led to many improvements in core NLP tasks such as Word Sense Disambiguation (Iacobacci et al., 2016), Machine Translation (Devlin et al., 2014), Machine Comprehension (Hewlett et al., 2016), and Semantic Role Labeling (Zhou and Xu, 2015; Collobert et al., 2011) – to name a few.

These methods suffer from a classic drawback of unsupervised learning: the lack of supervision between a word and those appearing in the associated contexts. Indeed, it is likely that some terms of the context are not related to the considered word. On the other hand, the fact that two words do not appear together – or more likely, not often enough together – in any context of the training corpora is not a guarantee that these words are not semantically related. Recent approaches have proposed to tackle this issue using an attentive model for context selection (Ling et al., 2015), or by using external sources – like knowledge graphs – in order to improve the embeddings (Wang et al., 2014). Similarities derived from such resources are part of the objective function during the learning phase (Yu and Dredze, 2014; Kiela et al., 2015) or used in a retrofitting scheme (Faruqui et al., 2015). These approaches tend to specialize the embeddings to the resource used and its associated similarity measures – while the construction and maintenance of these resources are a set of complex, time-consuming, and error-prone tasks.

In this paper, we propose a novel word embedding learning strategy, called Dict2vec, that leverages existing online natural language dictionaries. We assume that dictionary entries (a definition of a word) contain latent word similarity and relatedness information that can improve language representations. Such entries provide, in essence, an additional context that conveys general semantic coverage for most words. Dict2vec adds new co-occurrences information based on the terms occurring in the definitions of a word. This information brings us a kind of weak supervision that we can use to improve the embeddings. We can indeed distinguish word pairs for which each word appear in the definition of the other (strong pairs) and pairs where only one appears in the definition

of the other (weak pairs) – each having their own weight as two hyperparameters. Not only this information is useful at learning time to control words vectors to be close for such word pairs, but also it becomes possible to devise a controlled negative sampling. Controlled negative sampling as introduced in `Dict2vec` consists in filtering out random negative examples in conventional negative sampling that forms a (strong or weak) pair with the target word – they are obviously non-negative examples. Processing online dictionaries in `Dict2vec` does not require a human-in-the-loop – it is fully automated. The neural network architecture from `Dict2vec` (Section 3) extends `Word2vec` (Mikolov et al., 2013) approach which uses a Skip-gram model with negative sampling.

Our main results are as follows :

- `Dict2vec` exhibits a statistically significant improvement around 12.5% against state-of-the-art solutions on eleven most common evaluation datasets for the word similarity task when embeddings are learned using the full Wikipedia dump.
- This edge is even more significant for small training datasets (50 millions first tokens of Wikipedia) than using the full dataset, as the average improvement reaches 30%.
- Since `Dict2vec` does significantly better than competitors for small dimensions (in the $[20; 100]$ range) for small corpus, it can yield smaller yet efficient embeddings – even when trained on smaller corpus – which is one of the utmost practical interest for the working natural language processing practitioners.
- We also show that the embeddings learned by `Dict2vec` perform similarly to other baselines on an extrinsic text classification task.

Dict2vec software is an extension and an optimization from the original Word2vec framework leading to a more efficient learning. Source code to fetch dictionaries, train Dict2vec models and evaluate word embeddings are publicly availabe[1] and can be used by the community as a seed for future works.

[1] https://github.com/TOCOMPLETE

The paper is organized as follows. Section 2 presents related works, along with a special focus on Word2vec, which we later derive in our approach presented in Section 3. Our experimental setup and evaluation settings are introduced in Section 4 and we discuss the results in Section 5. Section 6 concludes the paper.

2 Learning Word Embeddings

2.1 The Neural Network Approach

In the original model from Collobert and Weston (2008), a window approach was used to feed a neural network and learn word embeddings. Since there are long-range relations between words, the window-based approach was later extended to a sentence-based approach (Collobert et al., 2011) leading to capture more semantic similarities into word vectors. Recurrent neural networks are another way to exploit the context of a word by considering the sequence of words preceding it (Mikolov et al., 2010; Sutskever et al., 2011). Each neuron receives the current window as an input, but also its own output from the previous step.

Mikolov et al. (2013) introduced the Skip-gram architecture built on a single hidden layer neural network to learn efficiently a vector representation for each word w of a vocabulary V from a large corpora of size C. Skip-gram iterates over all (target, context) pairs (w_t,w_c) from every window of the corpus and tries to predict w_c knowing w_t. The objective function is therefore to maximize the log-likelihood :

$$\sum_{t=1}^{C} \sum_{k=-n}^{n} \log p(w_{t+k}|w_t) \qquad (1)$$

where n represents the size of the window (composed of n words around the central word w_t) and the probability can be expressed as :

$$p(w_{t+k}|w_t) = \frac{e^{v_{t+k} \cdot v_t}}{\sum_{w \in V} e^{v \cdot v_t}} \qquad (2)$$

with v_{t+k} (resp. v_t) the vector associated to w_{t+k} (resp. w_t).

This model relies on the principle "You shall know a word by the company it keeps" – Firth (1957). Thus, words that are frequent within the context of the target word will tend to have close representations, as the model will update their vectors so that they will be closer. Two main drawbacks can be said about this approach.

First, words within the same window are not always related. Consider the sentence "Turing is widely considered to be the father of theoretical computer science and artificial intelligence."[2], the words (Turing,widely) and (father,theoretical) will be moved closer while they are not semantically related. Second, strong semantic relations between words (like synonymy or meronymy) happens rarely within the same window, so these relations will not be well embedded into vectors.

Bojanowski et al. (2016) use internal additional information from the corpus to solve the latter drawback. They train a Skip-gram architecture to predict a word w_c given the central word w_t and all the n-grams $\mathcal{G}_{w_t}$ (subwords of 3 up to 6 letters) of w_t. The objective function becomes :

$$\sum_{t=1}^{C} \sum_{k=-n}^{n} \sum_{w \in \mathcal{G}_{w_t}} \log p(w_{t+k}|w) \qquad (3)$$

Along learning one vector per word, fastText also learns one vector per n-gram. FastText is able to extract more semantic relations between words that share common n-gram(s) (like fish and fishing) which can also help to provide good embeddings for rare words since we can obtain a vector by summing vectors of its n-grams.

Since Dict2vec is based on the Skip-gram model and fastText demonstrates good results on semantic similarity datasets, we use these two models as baselines for our evaluation.

In what follows, we report related works that leverage external resources in order to address the two raised issues about the window approach.

2.2 Using External Resources

Even with larger and larger text data available on the Web, extracting and encoding every linguistic relations into word embeddings directly from corpora is a difficult task. One way to add more relations into embeddings is to use external data. Lexical databases like WordNet or sets of synonyms like MyThes thesaurus can be used during learning or in a post-processing step to specialize word embeddings. For example, Yu and Dredze (2014) include prior knowledge about synonyms from WordNet and the Paraphrase Database in a joint model built upon Word2vec. Faruqui et al. (2015) introduce a graph-based retrofitting method where they post-process learned vectors with respect to semantic relationships extracted from additional lexical resources. Kiela et al. (2015) propose to specialize the embeddings either on similarity or relatedness relations in a Skip-gram joint learning approach by adding new contexts from external thesaurus or from a norm association base in the function to optimize. Bian et al. (2014) combine several sources (syllables, POS tags, antonyms/synonyms, Freebase relations) and incorporate them into a CBOW model. These approaches have generally the objective to improve tasks such as document classification, synonym detection or word similarity. They rely on additional resources whose construction is a time-consuming and error-prone task and tend generally to specialize the embeddings to the external corpus used. Moreover, lexical databases contain less information than dictionaries (117k entries in WordNet, 200k in a dictionary) and less accurate content (some different words in WordNet belong to the same synset thus have the same definition).

Another type of external resources are knowledge bases, containing triplets. Each triplet links two entities with a relation, for example Paris – is capital of – France. Several methods (Weston et al., 2013; Wang et al., 2014; Xu et al., 2014) have been proposed to use the information from knowledge base to improve semantic relations in word embeddings, and extract more easily relational facts from text. These approaches are focused on knowledge base dependent task.

3 Dict2vec

The definition of a word is a group of words or sentences explaining its meaning. A dictionary is a set of tuples (word, definition) for several words. For example, one may find in a dictionary :

> **car**: A road vehicle, typically with four wheels, powered by an internal combustion engine and able to carry a small number of people.[3]

The presence of words like "vehicle", "road" or "engine" in the definition of "car" illustrates the relevance of using word definitions for obtaining weak supervision allowing us to get semantically related pairs of words.

`Dict2vec` models this information by building strong and weak pairs of words (§3.1), in order to provide both a novel positive sampling

[2] https://en.wikipedia.org/wiki/Alan_Turing

[3] Definition from Oxford dictionary.

objective (§3.2) and a novel controlled negative sampling objective (§3.3). These objectives participates to the global objective function of `Dict2vec` (§3.4).

3.1 Strong pairs, weak pairs

In a definition, each word does not have the same semantic relevance. In the definition of "car", the words "internal" or "number" are less relevant than "vehicle". We introduce the concept of strong and weak pairs in order to capture this relevance. If the word w_a is in the definition of the word w_b and w_b is in the definition of w_a, they form a strong pair, as well as the K closest words to w_a (resp. w_b) form a strong pair with w_b (resp. w_a). If the word w_a is in the definition of w_b but w_b is not in the definition of w_a, they form a weak pair.

The word "vehicle" is in the definition of "car" and "car" is in the definition of "vehicle". Hence, (car–vehicle) is a strong pair. The word "road" is in the definition of "car", but "car" is not in the definition of "road". Therefore, (car–road) is a weak pair.

Some weak pairs can be promoted as strong pairs if the two words are among the K closest neighbours of each other. We chose the K closest words according to the cosine distance from a pretrained word embedding and find that using $K = 5$ is a good trade-off between semantic and syntactic extracted information.

3.2 Positive sampling

We introduce the concept of positive sampling based on strong and weak pairs. We move closer vectors of words forming either a strong or a weak pair in addition to moving vectors of words co-occurring within the same window.

Let $\mathcal{S}(w)$ be the set of all words forming a strong pair with the word w and $\mathcal{W}(w)$ be the set of all words forming a weak pair with w. For each target w_t from the corpus, we build $\mathcal{V}_s(w_t)$ a random set of n_s words drawn with replacement from $\mathcal{S}(w_t)$ and $\mathcal{V}_w(w_t)$ a random set of n_w words drawn with replacement from $\mathcal{W}(w_t)$. We compute the cost of positive sampling J_{pos} for each target as follows:

$$J_{pos}(w_t) = \beta_s \sum_{w_i \in \mathcal{V}_s(w_t)} \ell(v_t \cdot v_i) + \beta_w \sum_{w_j \in \mathcal{V}_w(w_t)} \ell(v_t \cdot v_j) \quad (4)$$

where ℓ is the logistic loss function defined by $\ell : x \mapsto \log(1 + e^{-x})$ and v_t (resp. v_i and v_j) is the vector associated to w_t (resp. w_i and w_j).

The objective is to minimize this cost for all targets, thus moving closer words forming a strong or a weak pair.

The coefficients β_s and β_w, as well as the number of drawn pairs n_s and n_w, tune the importance of strong and weak pairs during the learning phase. We discuss the choice of these hyperparameters in Section 5. When $\beta_s = 0$ and $\beta_w = 0$, our model is the Skip-gram model of Mikolov et al. (2013).

3.3 Controlled negative sampling

Negative sampling consists in considering two random words from the vocabulary $\mathcal{V}$ to be unrelated. For each word w_t from the vocabulary, we generate a set $\mathcal{F}(w_t)$ of k randomly selected words from the vocabulary :

$$\mathcal{F}(w_t) = \{w_i\}^k, w_i \in \mathcal{V} \setminus \{w_t\} \quad (5)$$

The model aims at separating the vectors of words from $\mathcal{F}(w_t)$ and the vector of w_t. More formally, this is equivalent to minimize the cost J_{neg} for each target word w_t as follows:

$$J_{neg}(w_t) = \sum_{w_i \in \mathcal{F}(w_t)} \ell(-v_t \cdot v_i) \quad (6)$$

where the notation ℓ, v_t and v_i are the same as described in previous subsection.

However, there is a non-zero probability that w_i and w_t are related. Therefore, the model will move their vectors further instead of moving them closer. With strong/weak word pairs in `Dict2vec`, it becomes possible to better ensure that this is less likely to occur: we prevent a negative example to be a word that forms a weak or strong pair with with w_t. The negative sampling objective from Equation 6 becomes :

$$J_{neg}(w_t) = \sum_{\substack{w_i \in \mathcal{F}(w_t) \\ w_i \notin \mathcal{S}(w_t) \\ w_i \notin \mathcal{W}(w_t)}} \ell(-v_t \cdot v_i) \quad (7)$$

In our experiments, we noticed this method discards around 2% of generated negative pairs. The influence on evaluation depends on the nature of the corpus and is discussed at Section 5.4.

3.4 Global objective function

Our objective function is derived from the noise-contrastive estimation which is a more efficient objective function than the log-likelihood in Equation 1 according to Mikolov et al. (2013). We add the positive sampling and the controlled negative sampling described before and compute the cost for each (target,context) pair (w_t, w_c) from the corpus as follows:

$$J(w_t, w_c) = \ell(v_t \cdot v_c) + J_{pos}(w_t) + J_{neg}(w_t) \tag{8}$$

The global objective is obtained by summing every pair's cost over the entire corpus :

$$J = \sum_{t=1}^{C} \sum_{c=-n}^{n} J(w_t, w_{t+c}) \tag{9}$$

4 Experimental setup

4.1 Fetching online definitions

We extract all unique words with more than 5 occurrences from a full Wikipedia dump, representing around 2.2M words. Since there is no dictionary that contains a definition for all existing words (the word w might be in the dictionary D_i but not in D_j), we combine several dictionaries to get a definition for almost all of these words (some words are too rare to have a definition anyway). We use the English version of Cambridge, Oxford, Collins and dictionary.com. For each word, we download the 4 different webpages, and use regex to extract the definitions from the HTML template specific to each website, making the process fully accurate. Our approach does not focus on polysemy, so we concatenate all definitions for each word. Then we concatenate results from all dictionaries, remove stopwords and punctuation and lowercase all words. For our illustrative example in Section 3, we obtain :

> **car**: road vehicle engine wheels seats small [...] platform lift.

Among the 2.2M unique words, only 200K does have a definition. We generate strong and weak pairs from the downloaded definitions according to the rule described in subsection 3.1 leading to 417K strong pairs (when the parameter K from 3.1 is set to 5) and 3.9M weak pairs.

4.2 Training settings

We train our model with the generated pairs from subsection 4.1 and the November 2016 English dump from Wikipedia [4]. After removing all XML tags and converting all words to lowercase (with the help of Mahoney's script[5]), we separate the corpus into 3 files containing respectively the first 50M tokens, the first 200M tokens, and the full dump. Our model uses additional knowledge during training. For a fair comparison against other frameworks, we also incorporate this information into the training data and create two versions for each file : one containing only data from Wikipedia (corpus A) and one with data from Wikipedia concatenated with the definitions extracted (corpus B).

We use the same hyperparameters we usually find in the literature for all models. We use 5 negatives samples, 5 epochs, a window size of 5, a vector size of 100 (resp. 200 and 300) for the 50M file (resp. 200M and full dump) and we remove the words with less than 5 occurrences. We follow the same evaluation protocol as Word2vec and fastText to provide the fairest comparison against competitors, so every other hyperparameters (K, β_s, β_w, n_s, n_w) are tuned using a grid search to maximize the weighted average score. For n_s and n_w, we go from 0 to 10 with a step of 1 and find the optimal values to be $n_s = 4$ and $n_w = 5$. For β_s and β_w we go from 0 to 2 with a step of 0.05 and find $\beta_s = 0.8$ and $\beta_w = 0.45$ to be the best values for our model. Table 1 reports training times for the three models (all experiments were run on a E3-1246 v3 processor).

	50M	200M	Full
Word2vec	15m30	86m	2600m
fastText	8m44	66m	1870m
Dict2vec	4m09	26m	642m

Table 1: Training time (in min) of Word2vec, fastText and Dict2vec models for several corpus.

4.3 Word similarity evaluation

We follow the standard method for word similarity evaluation by computing the Spearman's rank correlation coefficient (Spearman, 1904) between human similarity evaluation of pairs of words, and

[4] https://dumps.wikimedia.org/enwiki/20161101/
[5] http://mattmahoney.net/dc/textdata#appendixa

the cosine similarity of the corresponding word vectors. A score close to 1 indicates an embedding close to the human judgement.

We use MC-30 (Miller and Charles, 1991), MEN (Bruni et al., 2014), MTurk-287 (Radinsky et al., 2011), MTurk-771 (Halawi et al., 2012), RG-65 (Rubenstein and Goodenough, 1965), RW (Luong et al., 2013), SimVerb-3500 (Gerz et al., 2016), WordSim-353 (Finkelstein et al., 2001) and YP-130 (Yang and Powers, 2006) classic datasets. We follow the same protocol used by Word2vec and fastText by discarding pairs which contain a word that is not in our embedding. Since all models are trained with the same corpora, the embeddings have the same words, therefore all competitors share the same OOV rates.

We run each experiment 3 times and report in Table 2 the average score to minimize the effect of the neural network random initialization. We compute the average by weighting each score by the number of pairs evaluated in its dataset in the same way as Iacobacci et al. (2016). We multiply each score by $1,000$ to improve readability.

4.4 Text classification evaluation

Our text classification task follows the same setup as the one for fastText in Joulin et al. (2016). We train a neural network composed of a single hidden layer where the input layer corresponds to the bag of words of a document and the output layer is the probability to belong to each label. The weights between the input and the hidden layer are initialized with the generated embeddings and are fixed during training, so that the evaluation score solely depends on the embedding. We update the weights of the neural network classifier with gradient descent. We use the datasets AG-News [6], DBpedia (Auer et al., 2007) and Yelp reviews (polarity and full)[7]. We split each datasets into a training and a test file. We use the same training and test files for all models and report the classification accuracy obtained on the test file.

4.5 Baselines

We train Word2vec[8] and fastText[9] on the same 3 files and their 2 respective versions (A and B) described in 4.2 and use the same hyperparameters also described in 4.2 for all models. We train Word2vec with the Skip-gram model since our method is based on the Skip-gram model. We also train GloVe with their respective hyperparameters described in Pennington et al. (2014), but the results are lower than all other baselines (weighted average on word similarity task is 350 on the 50M file, 389 on the 200M file and 454 on the full dump) so we do no report GloVe's results.

We also retrofit the learned embeddings on corpus A with the Faruqui's method to compare another method using additional resources. The retrofitting introduces external knowledge from the WordNet semantic lexicon (Miller, 1995). We use the **Faruqui's Retrofitting**[10] with the WN_{all} semantic lexicon from WordNet and 10 iterations as advised in the paper of Faruqui et al. (2015). Furthermore, we compare the performance of our method when using WordNet additional resources instead of dictionaries.

5 Results and model analysis

5.1 Semantic similarity

Table 2 (top) reports the Spearman's rank correlation scores obtained with the method described in subsection 4.3. We observe that our model outperforms state-of-the-art approaches for most of the datasets on the 50M and 200M tokens files, and almost all datasets on the full dump (this is significant according to a two-sided Wilcoxon signed-rank test with $\alpha = 0.05$). With the weighted average score, our model improves fastText's performance on raw corpus (column A) by 28.3% on the 50M file, by 17.7% on the 200M and by 12.8% on the full dump. Even when we train fastText with the same additional knowledge as ours (column B), our model improves performance by 2.9% on the 50M file, by 5.1% in the 200M and by 11.9% on the full dump.

We notice the column B (corpus composed of Wikipedia and definitions) has better results than the column A for the 50M ($+24\%$ on average) and the 200M file ($+12\%$ on average). This demonstrates the strong semantic relations one can find in definitions, and that simply incorporating definitions in small training file can boost the performance of the embeddings. Moreover, when the training file is large (full dump), our supervised method with pairs is more efficient, as the boost

[6] https://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

[7] https://www.yelp.com/dataset_challenge

[8] https://github.com/dav/word2vec

[9] https://github.com/facebookresearch/fastText

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

	50M							200M							Full						
		w2v		FT		our			w2v		FT		our			w2v		FT		our	
	oov	A	B	A	B	A	B	oov	A	B	A	B	A	B	oov	A	B	A	B	A	B
MC-30	0%	697	847	722	823	840	**859**	0%	742	830	795	814	**854**	827	0%	809	826	831	815	**860**	847
MEN-TR-3k	0%	692	753	697	**767**	733	762	0%	734	758	754	**772**	752	768	0%	733	728	752	751	**756**	755
MTurk-287	0%	657	**688**	657	685	665	682	0%	642	**671**	671	661	667	666	0%	660	656	**672**	671	661	660
MTurk-771	0%	596	677	597	692	685	**713**	0%	628	669	632	675	682	**704**	0%	623	620	631	638	**696**	694
RG-65	0%	714	865	671	842	824	**875**	0%	771	842	755	829	857	**877**	0%	787	802	817	820	**875**	867
RW	36%	375	420	442	**512**	475	489	16%	377	408	475	**507**	467	478	2%	407	427	464	468	**482**	476
SimVerb	3%	165	371	179	374	363	**432**	0%	183	306	206	329	377	**424**	0%	186	214	222	233	**384**	379
WS353-ALL	0%	660	739	657	739	738	**753**	0%	694	734	701	735	**762**	758	0%	705	721	729	723	756	**758**
WS353-REL	0%	619	**700**	623	696	679	688	0%	665	706	644	685	**710**	699	0%	664	681	687	686	702	**703**
WS353-SIM	0%	714	**797**	714	790	774	784	0%	743	**792**	758	792	784	787	0%	757	767	775	779	781	**781**
YP-130	3%	458	679	415	674	666	**696**	0%	449	592	509	639	616	**665**	0%	502	475	533	553	**646**	607
W.Average		453	562	467	582	564	**599**		471	533	503	563	569	**592**		476	488	508	512	**573**	570
AG-News		**874**	871	868	871	871	866		**886**	882	880	881	880	880		885	885	**887**	887	881	884
DBPedia		936	942	942	944	**944**	944		952	956	957	958	**960**	959		966	966	967	967	968	**969**
Yelp Pol.		808	835	821	**842**	832	834		837	855	852	859	856	**859**		865	867	872	874	**876**	875
Yelp Full		451	469	460	**473**	471	472		477	491	488	495	499	**501**		506	506	512	514	516	**518**

Table 2: Spearman's rank correlation coefficients between vectors' cosine similarity and human judgement for several datasets (top) and accuracies on text classification task (bottom). We train and evaluate each model 3 times and report the average score for each dataset, as well as the weighted average for all word similarity datasets.

		50M			200M			Full		
		w2v$_R$	FT$_R$	our$_R$	w2v$_R$	FT$_R$	our$_R$	w2v$_R$	FT$_R$	our$_R$
MC-30	vs self	+13.9%	+9.2%	+1.3%	+5.8%	+4.8%	+3.0%	+5.2%	+2.9%	+1.2%
	vs our	**-7.3%**	**-4.4%**	–	**-3.6%**	**-2.4%**	–	**-1.0%**	**-0.6%**	–
MEN-TR-3k	vs self	+0.9%	-0.7%	-0.1%	+0.7%	-1.9%	+0.4%	+1.4%	-2.8%	+1.6%
	vs our	**-4.2%**	**-7.4%**	–	**-1.3%**	**-1.6%**	–	**-1.7%**	**-3.3%**	–
MTurk-287	vs self	+1.4%	+0.2%	+3.5%	-2.9%	-3.3%	+3.0%	-0.9%	-5.7%	+1.1%
	vs our	**-1.2%**	**-4.0%**	–	**-4.3%**	**-2.7%**	–	**-1.1%**	**-4.1%**	–
MTurk-771	vs self	+8.2%	+4.9%	+1.6%	+6.3%	+4.3%	+1.5%	+4.5%	+1.6%	+0.6%
	vs our	**-7.3%**	**-8.8%**	–	**-3.8%**	**-3.4%**	–	**-6.5%**	**-7.9%**	–
RG-65	vs self	+10.9%	+17.1%	+4.0%	+6.6%	+8.5%	+3.0%	+7.0%	+5.0%	+2.4%
	vs our	**-2.1%**	**-4.9%**	–	**-2.2%**	**-4.4%**	–	**-3.8%**	**-1.9%**	–
RW	vs self	-20.3%	-24.4%	-14.3%	-24.4%	-25.9%	-20.3%	-18.7%	-25.4%	-19.1%
	vs our	**-37.4%**	**-26.9%**	–	**-37.7%**	**-24.6%**	–	**-31.3%**	**-28.2%**	–
Simverb	vs self	+46.0%	+34.0%	+19.8%	+49.7%	+39.8%	+19.9%	+44.6%	+38.7%	+16.7%
	vs our	**-34.4%**	**-28.4%**	–	**-30.5%**	**-23.6%**	–	**-29.9%**	**-19.8%**	–
WS353-ALL	vs self	-4.2%	-10.8%	-1.1%	-3.2%	-8.0%	-1.3%	-4.4%	-10.7%	-2.0%
	vs our	**-13.8%**	**-19.2%**	–	**-11.9%**	**-15.4%**	–	**-10.8%**	**-13.9%**	–
WS353-REL	vs self	-16.1%	-22.7%	-4.9%	-10.4%	-17.9%	-4.5%	-10.7%	-19.7%	-6.0%
	vs our	**-20.9%**	**-27.2%**	–	**-17.7%**	**-25.5%**	–	**-15.5%**	**-21.4%**	–
WS353-SIM	vs self	+4.3%	+0.8%	+3.2%	+2.6%	+0.0%	+3.2%	+0.0%	-3.6%	+2.4%
	vs our	**-3.9%**	**-6.7%**	–	**-2.9%**	**-3.3%**	–	**-3.0%**	**-4.4%**	–
YP-130	vs self	+17.8%	+3.2%	+3.3%	+13.0%	+6.9%	+8.0%	+11.1%	+8.3%	+5.0%
	vs our	**-23.6%**	**-28.2%**	–	**-16.6%**	**-11.7%**	–	**-13.6%**	**-10.7%**	–

Table 3: Percentage changes of word similarity scores for several datasets after the Faruqui's retrofitting method is applied. We compare each model to their own non-retrofitted version (vs self) and our non-retrofitted version (vs our). A positive percentage indicates the level of improvement of the retrofitting approach, while a negative percentage shows that the compared method is better without retrofitting. As an illustration: the +13.9% at the top left means that retrofitting Word2vec's vectors improves the initial vectors output by 13.9%, while the -7.3% below indicates that our approach without retrofitting is better than the retrofitted Word2vec's vectors.

brought by the concatenation of definitions is insignificant ($+1.5\%$ on average).

We also note that the number of strong and weak pairs drawn must be set according to the size of the training file. For the 50M and 200M tokens files, we train our model with hyperparameters $n_s = 4$ and $n_w = 5$. For the full dump (20 times larger than the 200M tokens file), the number of windows in the corpus is largely increased, so is the number of (target,context) pairs. Therefore, we need to adjust the influence of strong and weak pairs and decrease n_s and n_w. We set $n_s = 2, n_w = 3$ to train on the full dump.

The Faruqui's retrofitting method improves the word similarity scores on all frameworks for all datasets, except on RW and WS353 (Table 3). But even when Word2vec and fastText are retrofitted, their scores are still worse than our non-retrofitted model (every percentage on the *vs our* line are negative). We also notice that our model is compatible with a retrofitting improvement method as our scores are also increased with Faruqui's method.

We also observe that, although our model is superior on each corpus size, our model trained on the 50M tokens file outperforms the other models trained on the full dump (an improvement of 17% compared to the results of fastText, our best competitor, trained on the full dump). This means considering strong and weak pairs is more efficient than increasing the corpus size and that using dictionaries is a good way to improve the quality of the embeddings when the training file is small.

The models based on knowledge bases cited in §2.2 do not provide word similarity scores on all the datasets we used. However, for the reported scores, Dict2vec outperforms these models : Kiela et al. (2015) get 0.72 on the MEN dataset while we obtain 0.756; Xu et al. (2014) get 0.683 on the WS353-ALL dataset while we reach 0.758.

5.2 Text classification accuracy

Table 2 (bottom) reports the classification accuracy for the considered datasets. Our model achieves the same performances as Word2vec and fastText on the 50M file and slightly improves results on the 200M file and the full dump. Using supervision with pairs during training does not make our model specific to the word similarity task which shows that our embeddings can also be used in downstream extrinsic tasks.

Note that for this experiment, the embeddings were fixed and not updated during learning (we only learned the classifier parameters) since our objective was rather to evaluate the capability of the embeddings to be used for another task rather than obtaining the best possible models. It is anyway possible to obtain better results by updating the embeddings and the classifier parameters with respect to the supervised information to adapt the embeddings to the classification task at hand as done in Joulin et al. (2016).

5.3 Dictionaries vs. WordNet

		Raw	R_{WN}	R_{dict}
50M	w2v	453	474	479
	FT	467	476	489
	our	569	582	582
200M	w2v	471	488	494
	FT	503	504	524
	our	569	581	587
full	w2v	488	507	512
	FT	508	503	529
	our	571	583	592

Table 4: Weighted average Spearman correlation score of raw vectors and after retrofitting with WordNet pairs (R_{WN}) and dictionary pairs (R_{dict}).

Table 4 reports the Spearman's rank correlation score for vectors obtained after training (Raw column) and the scores after we retrofit those vectors with pairs from WordNet (R_{WN}) and extracted pairs from dictionaries (R_{dict}). Retrofitting with dictionaries outperforms retrofitting with WordNet lexicons, meaning that data from dictionaries are better suited to improve embeddings toward semantic similarities when retrofitting.

	50M	200M	full
No pairs	453	471	488
With WordNet pairs	564	566	559
With dictionary pairs	569	569	571

Table 5: Weighted average Spearman correlation score of Dict2vec vectors when trained without pairs and with WordNet or dictionary pairs.

We also trained Dict2vec with pairs from WordNet as well as no additional pairs during training (in this case, this is the Skip-gram model

from Word2vec). Results are reported in Table 5. Training with WordNet pairs increases the scores, showing that the supervision brought by positive sampling is beneficial to the model, but lags behind the training using dictionary pairs demonstrating once again that dictionaries contain more semantic information than WordNet.

5.4 Positive and negative sampling

For the positive sampling, an empirical grid search shows that a $\frac{1}{2}$ ratio between β_s and β_w is a good rule-of-thumb for tuning these hyperparameters. We also notice that when these coefficients are too low ($\beta_s \leq 0.5$ and $\beta_w \leq 0.2$), results get worse because the model does not take into account the information from the strong and weak pairs. On the other side, when they are too high ($\beta_s \geq 1.2$ and $\beta_w \geq 0.6$), the model discards too much the information from the context in favor of the information from the pairs. This behaviour is similar when the number of strong and weak pairs is too low or too high ($n_s, n_w \leq 2$ or $n_s, n_w \geq 5$).

For the negative sampling, we notice that the control brought by the pairs increases the average weighted score by 0.7% compared to the uncontrolled version. We also observe that increasing the number of negative samples does not significantly improve the results except for the RW dataset where using 25 negative samples can boost performances by 10%. Indeed, this dataset is mostly composed of rare words so the embeddings must learn to differentiate unrelated words rather than moving closer related ones.

5.5 Vector size

In Fig. 1, we observe that our model is still able to outperform state-of-the-art approaches when we reduce the dimension of the embeddings to 20 or 40. We also notice that increasing the vector size does increase the performance, but only until a dimension around 100, which is the common dimension used when training on the 50M tokens file for related approaches reported here.

6 Conclusion

In this paper, we presented `Dict2vec`, a new approach for learning word embeddings using lexical dictionaries. It is based on a Skip-gram model where the objective function is extended by leveraging word pairs extracted from the definitions weighted differently with respect to the strength of the pairs. Our approach shows better results than state-of-the-art word embeddings methods for the word similarity task, including methods based on a retrofitting from external sources. We also provide the full source code to reproduce the experiments.

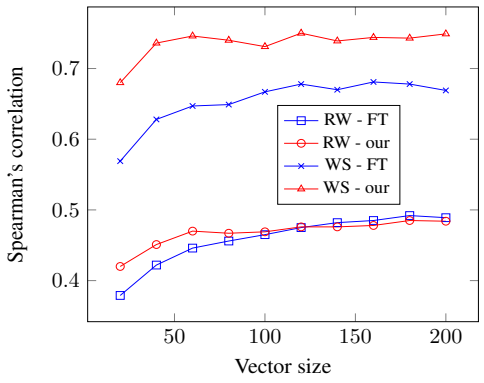


Figure 1: Spearman's rank correlation coefficient for RW-STANFORD (RW) and WS-353-ALL (WS) on the FastText model (FT) and our, with different vector size. Training is done on the corpus A of 50M tokens.

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. *The semantic web* pages 722–735.

Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Knowledge-powered deep learning for word embedding. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pages 132–148.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* .

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

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, pages 160–167.

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.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*. Citeseer, pages 1370–1380.

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

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*. ACM, pages 406–414.

John Rupert Firth. 1957. *Papers in Linguistics 1934-1951: Repr*. Oxford University Press.

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*. ACM, pages 1406–1414.

Daniel Hewlett, Alexandre Lacoste, Llion Jones, Illia Polosukhin, Andrew Fandrianto, Jay Han, Matthew Kelcey, and David Berthelot. 2016. Wikireading: A novel large-scale language understanding task over wikipedia. *arXiv preprint arXiv:1608.03542* .

Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2016. Embeddings for word sense disambiguation: An evaluation study. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. volume 1, pages 897–907.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* .

Douwe Kiela, Felix Hill, and Stephen Clark. 2015. Specializing word embeddings for similarity or relatedness. In *Proceedings of EMNLP*.

Wang Ling, Lin Chu-Cheng, Yulia Tsvetkov, and Silvio Amir. 2015. Not all contexts are created equal: Better word representations with variable attention. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 1367–1372.

Minh-Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *CoNLL*. pages 104–113.

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

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*. volume 2, page 3.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11):39–41.

George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and cognitive processes* 6(1):1–28.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. volume 14, pages 1532–43.

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*. ACM, pages 337–346.

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

Charles Spearman. 1904. The proof and measurement of association between two things. *The American journal of psychology* 15(1):72–101.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. pages 1017–1024.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph and text jointly embedding. In *EMNLP*. Citeseer, pages 1591–1601.

Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. 2013. Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973* .

Chang Xu, Yalong Bai, Jiang Bian, Bin Gao, Gang Wang, Xiaoguang Liu, and Tie-Yan Liu. 2014. Rc-net: A general framework for incorporating knowledge into word representations. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, pages 1219–1228.

Dongqiang Yang and David MW Powers. 2006. *Verb similarity on the taxonomy of WordNet*. Masaryk University.

Mo Yu and Mark Dredze. 2014. Improving lexical embeddings with semantic knowledge. In *ACL (2)*. pages 545–550.

Jie Zhou and Wei Xu. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. In *ACL (1)*. pages 1127–1137.