Improving Word Embeddings through Iterative Refinement of Word- and Character-level Models

Phong Ha, Shanshan Zhang, Nemanja Djuric, Slobodan Vucetic
Department of Computer and Information Sciences, Temple University

Abstract

Embedding of rare and out-of-vocabulary (OOV) words is an important open NLP problem. A popular solution is to train a character-level neural network to reproduce the embeddings from a standard word embedding model. The trained network is then used to assign vectors to any input string, including OOV and rare words. We enhance this approach and introduce an algorithm that iteratively refines and improves both word- and character-level models. We demonstrate that our method outperforms the existing algorithms on 5 word similarity data sets, and that it can be successfully applied to job title normalization, an important problem in the e-recruitment domain that suffers from the OOV problem.

Mimick

Recently proposed **Mimicking** (Pinter et al., 2017; Kim et al., 2018) approach is a promising solution to the problem of Out-of-Vocabulary (OOV). Their main idea is to train a character-level embedding neural network (NN) that can reconstruct, or mimic, an embedding from word-level embedding model. The trained character-level model is then used to generate both semantically and syntactically relevant embeddings for arbitrary character sequences.

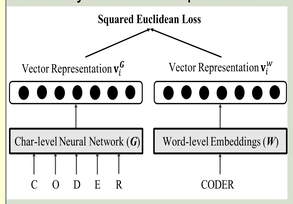

Figure 1. General Mimicking Framework

Mimick (Pinter et al., 2017) and GWR (Kim et al., 2018) are the 2 representatives of this method. They share the same idea, where a character-level embedding model G is learned to mimic a pretrained word-level embedding model W.

Word embedding model (W): Given a corpus D and a vocabulary V, word embedding models such as SkipGram and CBOW output a vector for each word in V, resulting in an embedding set $\{(w_i, vW_i)|i, \dots, M\}$, where $w_i \in V$ is a word and $v^W{}_i \in Rd$ is its vector representation.

Character embedding model (G): Mimick and GWR expands the fixed-vocabulary semantic space by training a character neural network G to mimic the embeddings from W. Generator G provides mapping from a character sequence to a vector in the same space as W, while preserving syntactic similarities(Pinter et al., 2017)

To train G, we minimize the squared Euclidean distance between $v^G{}_i$ and $v^W{}_i$ for $w_i \in V$

$$\mathcal{L} = \frac{1}{|V|}\sum_{i=1}^{|V|} \|\mathbf{v}_i^G - \mathbf{v}_i^W\|_2^2.$$

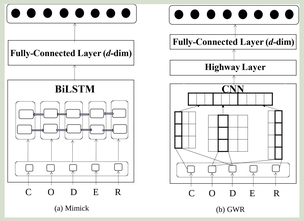

Figure 2. Mimick and GWR Architectures

Iterative Mimick

When G is fit on W, we observed that the syntactic similarities dominate relationships between the words, while semantic similarities weaken as compared to the embeddings produced by W. For example, for our e-recruiting data set discussed in more detail below, the neighbors of are mostly spelling variants of job *java developer* titles such as *javadeveloperor, javapython, developer*. Similarly, when the model is trained on a Twitter data set, neighbors of the word are *foods, fooooood*, and *frood*. This result is clearly suboptimal, and is consistent with earlier findings (Pinter et al., 2017)

To address this issue, we propose an Iterative Mimicking (IM) mechanism, illustrated in Figure 3. In IM, the word embedding model W and character embedding model G alternately influence each other in multiple iterations, a procedure we refer to as retrofitting. After fitting G on W once, IM continues with training W by re-initializing vector $v^W{}_i$ with the current Gi for each $w_i \in V$. Thanks to different initialization, the training of W results in a different local minimum that should better represent the morphological information captured by G. The retrofitting will not only correct the vectors that G wrongly placed in the semantic space, but also enhance the learning of W by exploiting knowledge about syntactically similar words learned by G.

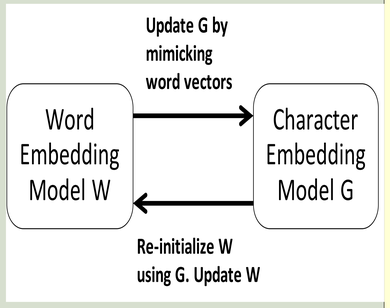

Figure 3. Iterative Mimick Framework

Upon retraining W, IM proceeds by updating G based on the updated W, and the entire iterative process is repeated several times. The algorithm terminates after N iterations, after the gap between word-embedding space W and character-embedding space produced by G is sufficiently small, or after the embedding by G stabilizes. During inference, we may use G to generate vectors for any input string and discard W. Mimick (Pinter et al., 2017) and GWR (Zhao et al.,2018) are special cases of our algorithm, where the number of iterations N=1

Job Title Normalization

Job Title Normalization is a prime example of the OOV problem due to the unstructured nature of job titles. Given data set $D = \{(t_i, di)|i = 1, \dots, N\}$, where t_i is job title and d_i is job description. Both are free-form text entered by a user of a professional network such as LinkedIn or Indeed. Let us also suppose there is a job title taxonomy $O = \{o_1, \dots, om\}$, which is a finite set denoting m distinct job categories. An example of such a taxonomy is the Standard Occupational Classification (SOC), whose O*NET-SOC2010 release contains 1,100 job titles (Elias et al., 2010). Then, the task is to match (or normalize) job title $t \in T$ to one job category $o \in O$ from the taxonomy.

Job Title	Job Description
Software Engineer	*Im a python developer building a web application for job recruiters to search multiple job boards at once like kayakcom for recruiters its a highly concurrent application using mongodb for the backend*
Visiting Assistant Professor	*Research in differential algebra and mathematical logic. Taught graduate and undergraduate courses in Logic Model Theory, Theory of Computation.*

Table 1. Example Job Titles and Job Descriptions

We apply our Iterative Mimick algorithm to solve this problem. For each pair (t_i, di), we randomly insert t_i into d_i to train W. Then, once we obtain G from the IM model, we use it to provide each job title t_i and each job category o_i a vector. Each t_i is normalized to the nearest o_i in terms of Euclidean distance. Results are shown in Extrinsinc Evaluation Section

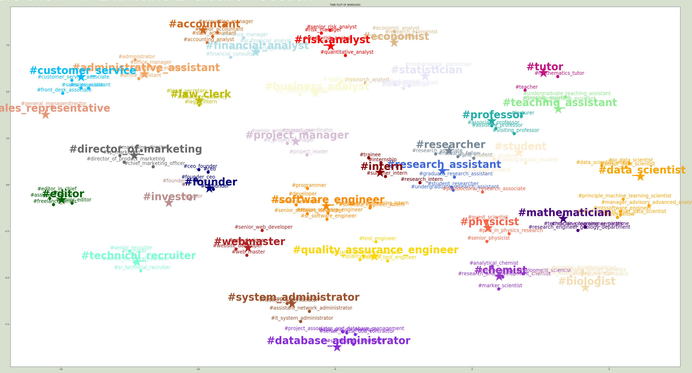

Figure 4. Vector Space of Job Titles

Figure 5. All Job Title Variants are normalized to the category "Software Engineer"

Intrinsic Evaluation

We first evaluate our framework on the task on Word Similarity. Given word pairs and the word similarity judgements by human labelers, the quality of embedding models is measured as the correlation between the human judgment and the cosine similarity of word embeddings.

5 Test sets and 2 Corpus are used in this evaluation. Pearson Correlation is used as the measure. The results are shown in Table 2 below.

Corpus	Model	RG65	SL	WS	RW	MEN
Text8	FastText	0.551	0.273	0.621	**0.422**	0.613
	Mimick	0.418	0.142	0.439	0.206	0.390
	Mimick+IM	0.463	0.238	0.512	0.282	0.474
	GWR	0.572	0.270	0.627	0.276	0.624
	GWR+IM	**0.643**	**0.294**	**0.673**	0.317	**0.664**
Twitter	FastText	0.662	0.196	0.415	**0.277**	**0.628**
	Mimick	0.312	0.061	0.204	0.102	0.316
	Mimick+IM	0.355	0.092	0.242	0.118	0.385
	GWR	0.560	0.202	0.403	0.161	0.578
	GWR+IM	**0.683**	**0.221**	**0.467**	0.189	0.612

Table 2. Pearson Correlation on 5 word similarity tasks

GWR outperformed Mimick on all data sets.
Both Mimick and GWR saw significant improvements through the proposed iterative process on most data sets, confirming our hypothesis that iteratively retrofitting the word- and the character-level models does improve the embedding quality.

Extrinsic Evaluation

6 human judgers were asked to evaluate the quality of the 5 competing models.

Frequency	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
AutoCoder	0.466	0.501	0.537	0.614	0.625	**0.721**	**0.777**	0.732
FastText	0.408	0.419	0.481	0.519	0.581	0.582	0.635	0.611
Mimick	0.371	0.368	0.536	0.513	0.479	0.649	0.67	0.616
Mimick+IM	0.587	**0.665**	0.588	0.557	0.653	0.667	0.749	0.723
GWR+IM	**0.592**	0.623	**0.658**	**0.668**	**0.746**	0.718	0.756	**0.792**

Table 3. Avg. testing scores in different frequency ranges

For each query job title, we listed randomly shuffled best matches from O*NET-SOC taxonomy produced by each model.
Human evaluators were asked to choose the best match.
If a model produced the winning category it received 1 point, otherwise it received 0 points.
If multiple models produced the winning category all of them received 1 point. Results are then averaged.

Judger	#1	#2	#3	#4	#5	#6	Avg.
AutoCoder	0.636	0.679	0.546	0.628	0.540	**0.698**	0.621
FastText	0.552	0.581	0.476	0.518	0.511	0.503	0.524
Mimick	0.501	0.527	0.485	0.571	0.437	0.554	0.513
Mimick+IM	0.673	0.672	0.583	0.631	**0.646**	0.646	0.642
GWR+IM	**0.758**	**0.767**	**0.647**	**0.719**	0.587	0.685	**0.694**

Table 4. Avg. testing scores by 6 different human judgers

As can be seen in the tables, applying IM to the Mimick LSTM model, the quality of the normalization significantly improved, nearly 25% on average.

GWR+IM consistently outperforms other models in most frequency ranges. Our model also outperforms the commercial AutoCoder software in most frequency groups, only losing in T6 and T7.
Given that our model takes much less time to train and does not require manual labor, this is an impressive result.