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

Phong Ha[1] **Shanshan Zhang**[1] **Nemanja Djuric**[2] **Slobodan Vucetic**[1]
[1]Department of Computer and Information Sciences, Temple University
[2]Uber ATG
{phongtheha, zhang.shanshan, nemanja, vucetic}@temple.edu

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.

1 Introduction

Inability to represent rare and unseen words, which is referred to as the out-of-vocabulary (OOV) problem, limits usefulness of the standard embedding approaches to real-world applications. For example, in the e-recruitment domain there is an extremely large number of ways one can write a job title for the same job type. LinkedIn users who are software engineers may describe themselves as *software developer*, *python guru*, *sw engi.*, *sw developper* or *sw and web application developer*, which are caused by different variants, abbreviations, misspellings, or compoundings. In this case, standard embedding for a finite set of job titles is not helpful when recommending jobs to users with rare or unseen job titles.

An embedding model that can effectively handle the OOV problem should preserve both semantic and syntactic characteristics of words and phrases. Recently proposed **mimicking** approach is a promising solution to this problem, with two representative algorithms being Mimick (Pinter et al., 2017) and GWR (Kim et al., 2018). 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. While Mimick and GWR used BiLSTM and CNN character-level models, respectively, similar approaches were recently proposed with several other architectures (Zhao et al., 2018; Schick and Schütze, 2019).

In order to provide high-quality embeddings useful for downstream applications, it is crucial for the model to preserve semantic characteristics of words and phrases, while also being robust to syntactic and word-form changes. However, this may not always be the case for the vanilla mimicking method. To see why, let us first denote the word embedding model as W and the character embedding model as G in the mimicking mechanism. Let us take a peek at model W produced by training on career profile corpus, described in detail in Section 5. The top 5 neighbors for word *java developer* in the vector space induced by W are *software developer*, *software engineer*, *web developer*, *sr java developer*, and *programmer*. After fitting G to mimic embeddings from W, the top 5 neighbors of *java developer* in the vector space induced by G become *java developeur*, *javadeveloper*, *javva developer*, *javapython developer*, *javas developpper*, many of which are rare phrases that were not placed in the neighborhood by W. This is helpful when dealing with the OOV and rare words such as *javas developpper*, but might negatively affect

Proceedings of the 28th International Conference on Computational Linguistics, pages 1204–1213
Barcelona, Spain (Online), December 8-13, 2020

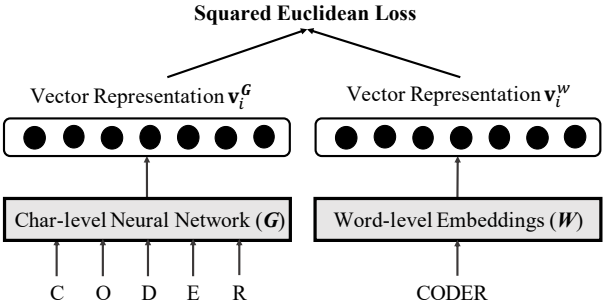

Figure 1: General Mimicking Framework

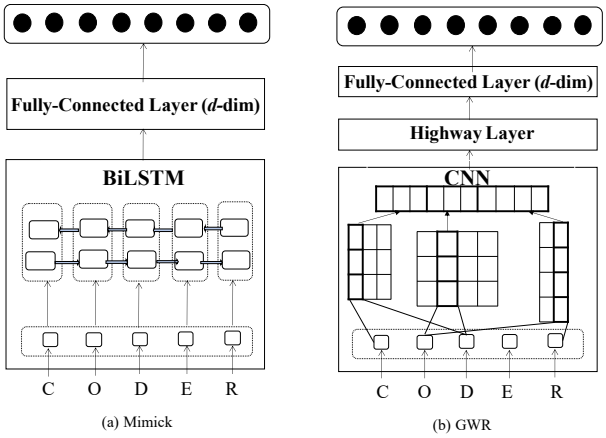

Figure 2: Different architectures for character-level models G

the quality of the embedding produced by G as the word-form similarities become too influential in the training of the model, resulting in G failing to preserve semantic similarities of words with the same meaning but different spelling. In the above example, *software engineer* and *web developer* no longer have similar embeddings with *java developer*, even though they refer to similar occupations. Ideally, model G should retain ability to map semantically related words into the same neighborhood, something that W does well. In addition, G should also be able to differentiate between words with similar spelling that are disparate semantically (e.g., *java cafe expert* should not be in the same neighborhood with *java developer* or *java expert*).

In order to better capture the syntactic and semantic similarities between words, we developed an iterative mimicking framework that strikes a good balance between word-level and character-level representations of words. In the proposed 2-mode framework, W training is re-initialized by G embeddings, then G is fine-tuned by mimicking W embeddings, and the process is repeated several times. As we demonstrate by experiments, the resulting procedure produces improved character-level embeddings on both common and rare and OOV words. We refer to the resulting approach as Iterative Mimicking (IM). Our contributions are summarized below:

- We propose a framework that iteratively refines W and G models. The final char-level model G is used to assign vectors to any input sequence (such as OOV words);
- We show that the IM approach is superior to the state-of-the-art baselines through intrinsic and extrinsic evaluations on five word similarity tasks;
- We illustrate effectiveness of the IM approach on a task of job title normalization, which is an important problem in the e-recruiting domain.

2 Related Work

In this section, we discuss existing approaches capable of generating representations for OOV words.

Aggregation of *n*-grams or morphemes. FastText (Bojanowski et al., 2016) algorithm directly learns embeddings of *n*-grams with a Word2Vec objective (CBOW or SkipGram). Then, embedding for an OOV word is simply aggregated as the average of the word's *n*-gram embeddings. In Charagram (Wieting et al., 2016), a nonlinear aggregation function is used instead of averaging *n*-gram embeddings. Aggregation over morphemes is discussed in (Qiu et al., 2014).

Character-aware modules in deep NNs. Stacking deep NNs on top of character-level modules is a popular recent trend. For example, character BiLSTMs or CNNs are co-trained with a language model in ELMo (Peters et al., 2018) and charCNN (Kim et al., 2016). Similar to mimicking, the character-aware modules in those methods are able to assign vectors to OOV words. However, they are computationally expensive due to a need to train a heavyweight language model on a large corpus to be effective. There are also approaches that co-train the character-aware modules with auxiliary supervised tasks, such as part-of-speech tagging (Santos and Zadrozny, 2014), text classification (Zhang et al., 2015) and machine translation (Luong and Manning, 2016), but the resulting embeddings are not necessarily reusable for other downstream tasks.

Mimicking. This approach allows learning lightweight character embedding models in a two-step manner (Pinter et al., 2017; Kim et al., 2018; Schick and Schütze, 2019; Zhao et al., 2018). It was shown that it yields superior performance to other baselines on multiple intrinsic and extrinsic tasks. Our proposed method is generalizing this line of research, as discussed in the following section.

3 Methodology

We first introduce the original **mimicking** mechanism used in Mimick (Pinter et al., 2017) and GWR (Kim et al., 2018). We then describe our proposed iterative mimicking (IM) mechanism, which is a generalization of the original approaches.

3.1 Mimick and GWR

Mimick and GWR methods 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 $\{(w_i, \mathbf{v}_i^W)|i, ..., M\}$, where $w_i \in V$ is a word and $\mathbf{v}_i^W \in \mathbb{R}^d$ is its vector representation. Word embedding models are good at preserving semantic similarities, where semantically similar words are assigned similar word embeddings.

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).

Figure 1 illustrates the general mimicking framework, where G is a black-box character-level neural network whose architecture can be customized differently depending on the task. (Pinter et al., 2017) defines G to be a Bidirectional LSTM (as shown in Figure 2a), while (Kim et al., 2018) takes G to be a Convolutional Neural Network (CNN) followed by a Highway layer (see Figure 2b), similar to the architecture in (Kim et al., 2016). G takes word w, which is a sequence of characters and generates a d-dimensional vector $\mathbf{v}_w^G$ as output. Note that d matches the dimensionality of the embedding vectors produced by W. To train G, we minimize the squared Euclidean distance between $\mathbf{v}_i^G$ and $\mathbf{v}_i^W$ for $w_i \in V$,

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

After the training phase completes, we can use the learned model G to generate vectors for any input character sequence.

3.2 Iterative Mimicking

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 in Section 5, the neighbors of *java developer* are mostly spelling variants of job titles such as *javadeveloper* or *javapython developer*. Similarly, when the model is trained on a Twitter data set, neighbors of the word *food* are *foods*, *fooooood*, and *frood*. This result is clearly suboptimal, and is consistent with earlier findings (Pinter et al., 2017).

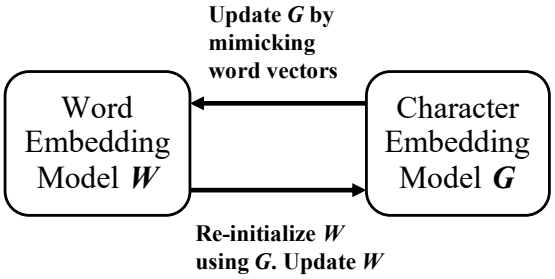

Figure 3: Illustration of Iterative Mimicking (IM)

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 $\mathbf{v}_i^W$ with the current $\mathbf{v}_i^G$ 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. 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 $N = 1$.

In the following sections we evaluate the proposed framework on several different challenges. First, we compare IM to baselines on tasks of word similarity. Then, we consider job title normalization, a critical task in e-recruiting domain. We evaluate the method on a real-world, large-scale data set obtained from a major professional network.

4 Word Similarity

We first perform intrinsic evaluation of embedding models on word similarity tasks. 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.

4.1 Experimental Setup

Following the related work (Faruqui et al., 2016), we used five standard word similarity data sets for English language: RG-65, Simlex (SL), WordSim-353 (WS), Stanford's RareWords (RW) and MEN-3000.

We compared the following embedding models: (1) **FastText**. Embedding of a word is averaged embedding of the word's *n*-grams. (2) **Mimick**. We implemented the original Mimick algorithm by using the same char-BiLSTM architecture as in the original paper (Pinter et al., 2017), except that we set the hidden dimension in the char-BiLSTM to 300. (3) **Mimick+IM**. We used the same char-BiLSTM as in Mimick, but updated it with the proposed IM mechanism. (4) **GWR**. We implemented the original GWR algorithm and used the same architecture for char-CNN as in the original paper (Kim et al., 2018). (5) **GWR+IM**. We used the same char-CNN as in GWR, but updated it with the proposed IM mechanism.

Note that an alternative to the (Iterative) Mimicking models is to simply train a character-level neural network directly on the corpus in a Skip-Gram Word2Vec fashion (skipping W altogether). However, in our experiments on both NLP and the career data sets, we find that the character-level neural network when trained this way is very unstable. Without the help of the word-level information the character-level model is barely able to learn any word semantics on its own, it mostly picks up the word-form similarities. Furthermore, it is prone to being collapsed into a single point, where it would produce very similar vectors for any arbitrary input. For that reason we excluded this baseline from the analysis.

To train FastText and the word embedding model W in the four mimicking models we used two different data sets: (1) Text8[1], which contains the first 100 billion bytes of Wikipedia; (2) Twitter set (Wang et al., 2015), which contains 4 million tweets about different topics. Twitter corpus is used to evaluate the effect of training on informal and noisy sentences. Once a model is trained we generate embeddings for both words in a word-pair, and then calculate cosine similarity between the two embeddings. Finally, we calculated the Pearson correlation between cosine similarities and similarity scores provided by human annotators. We repeated each experiment 3 times. The averaged correlation score is reported.

Hyperparameters: All models were trained from scratch. We set the dimensionality of embeddings to 100 for all five models. For FastText, we choose *n*-grams with *n* ranging from 2 to 5 so that we can assign vectors to any input. We instantiated the word-level model W as SkipGram in models (2)-(5). For both FastText and SkipGram we set the context window to 5. We only trained on words occurring more than 5 times. We ran FastText for 10 epochs. In Mimick and GWR, we trained SkipGram for 10 epochs, followed by 100 training epochs of character-level model G. For IM models we ran $N = 5$ iterations, during each we trained G for 20 epochs and W for 2 epochs. Thus, all models were trained for an equal number of epochs to ensure fair comparison.

4.2 Results

Table 1: Pearson correlation on 5 word similarity tasks

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 1 shows the Pearson correlation for the 5 models on the two data sets. Most models performed better when trained on the Text8 corpus, due to its more diverse and formal vocabulary. We can observe that GWR outperformed Mimick on all data sets, showing that character-level CNN is superior to LSTM both in performance and training time. 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.

We also performed qualitative analysis, where we compared the neighbors of various word embeddings trained on the Twitter corpus, including both common and OOV words. As seen in Table 2, vanilla Mimick relies mainly on similar spelling to produce embeddings, with limited semantic capabilities. While this may be helpful in rare examples such as *cheeseball*, where the neighbors in word2vec are mostly noisy, examples such as *seven* and *manhattan* show that G may often disturb good relative positions between words. When trained in an IM fashion, the model was able to bring more semantically similar words with different spellings into the neighborhood, such as *midtown* and *dumbo* for word *manhattan*. In addition, it also restored the semantic neighbors of *seven*. Results of the Mimick+IM approach show that the model can infer high-quality embeddings for OOV words. For example, for *prelecture* the only relevant neighbor

[1]http://mattmahoney.net/dc/textdata.html, last accessed June 2020.

Table 2: Nearest Neighbors of different embedding models for the Twitter data set

Word	Word2Vec	Mimick	Mimick+IM
manhattan	*nyc, queens, midtown, dumbo, williamsburgbridge*	*manhattans, manhatan, cmanhattan, nyc, manhatten*	*manhattans, midtown, aftersandy, nyc, dumbo, cmanhattan*
seven	*four, eight, three, five, six*	*server, severe, sevens, deven, tern*	*eight, four, sevens, seventy, three*
cheeseball	*eatbar, taboon, india, hahahaha, wildflowers*	*cheeseballs, cheeseboy, cheesehead, cheeses, cheese*	*cheeseballs, cheeseboy, cheesehead, tomato, cheeseburger*
prelecture (OOV)	-	*lecturing, innocence, liban, humanist, preparados*	*lecturer, lecturing, advising, philosophy, discussion*
fishering (OOV)	-	*fishermen, craftbeer, slobbering, gathering, farms*	*fisherman, fishers, fishery, fishes, water, nord*

for Mimick is *lecturing*, while Mimick+IM mapped more semantically similar words such as *lecturer*, *advising*, and *discussion*.

5 Job Title Normalization

We dedicate the rest of the paper to the task of job title normalization, an important problem in online recruiting industry worth astounding $400B (Cohan, 2018). The unstructured nature of job titles and occupations makes job search on these online platforms difficult, as potential matches can be missed due to large variability. For example, although *sw eng.* and *programmer* refer to the same occupation, NLP models may fail to automatically match these two very different strings. This problem is further exacerbated for rare titles, which are even more difficult to model due to their low counts. As seen in Figure 4, titles observed less than 10 times in our data set cover as much as 70% of the total corpus, making handling of rare titles a problem of very significant importance in the e-recruiting domain. To the best of our knowledge, this is the first work to utilize *mimicking*-based approaches to address the problem of job title normalization.

We formally describe the problem of *job title normalization* as follows. We are given a data set $D = \{(t_i, d_i)|i = 1, ..., N\}$, 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 denote the vocabulary of job titles as $\mathcal{T}$, and the vocabulary of words in job descriptions as $\mathcal{V}$. Both $\mathcal{T}$ and $\mathcal{V}$ are infinite sets. Let us also suppose there is a job title taxonomy $\mathcal{O} = \{o_1, \ldots, o_m\}$, 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-SOC 2010 release contains 1,100 job titles (Elias et al., 2010). Then, the task is to match (or normalize) job title $t \in \mathcal{T}$ to one job category $o \in \mathcal{O}$ from the taxonomy. We note that this definition of job normalization corresponds to entity linking (Yamada et al., 2016; Moreno et al., 2017). In our experiments we will evaluate results according to this definition. We also note that an alternative information retrieval definition of job normalization is to find job titles that are most similar to the query job title. We do not show experiments according to this definition.

To classify a job title into a category, we first learn how to embed an input string representing a job title into a vector space using the character-level model *G*. Using the same model we can map all job categories into the same vector space by assigning each category a vector.[2] Then, given a particular job title, we match it to the closest job category in the vector space. Due to the large variety of ways one can write a job title, models that are able to generate high-quality embeddings for rare or OOV titles are preferable. We note that this approach does not require manually labeled training data, unlike the method proposed in (Neculoiu et al., 2016). All that is required is a corpus of (*job titles*, *job description*) pairs to train the embedding models.

5.1 Experimental Setup

We obtained a list of 1.1 million (*job title*, *job description*) pairs from a large professional social network. An example of such pairs is listed in Table 3. The job titles in the data set encompass a broad range of

[2]The BLS releases a Direct Match Titles file as a part of the SOC taxonomy package, which can be found on https://www.bls.gov/soc/2010/classification. Each category in the taxonomy contains a few Direct Match example titles. In order to strengthen the quality of the category vectors, we assign each Direct Match title a vector using the character-level model and average them to get the final vector for each category in the taxonomy.

Table 3: Example of job title and description

title: software engineer
description: I'm a python developer building a Web application for job recruiters to search multiple job boards at once like Kayak.com. For recruiters, it's a highly concurrent application using MongoDB for the backend.

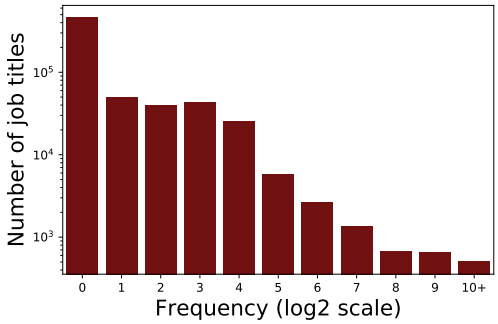

Figure 4: Distribution of job title frequencies (x-axis is rounded logarithm of job title frequencies, while y-axis is number of unique job titles with the corresponding frequency in log-scale

industries and occupations. We process the job titles by replacing the white spaces with underscores and add a # character at the beginning of a job title to distinguish a job title from a word in job descriptions. We remove punctuation and lowercase all job titles and words in the data set. The number of unique job titles is 630,000 and the number of unique words in job descriptions is 920,000. We plot the distribution of job title frequencies in rounded logarithm scale in Figure 4. As we can see, most of the job titles appear only once in D (around 300,000), accounting for nearly 50% of the data set.

Word embedding model W: We preprocess our data set D in order to learn word embedding model W for tokens in both $\mathcal{T}$ and $\mathcal{V}$. We insert each job title t_i into the job description d_i at n random positions where $n = \left\lceil \frac{length(d_i)}{10} \right\rceil$. Thus, we form a new data set $S = \{d'_i | i = 1, ..., N\}$, where d'_i is obtained by adding t_i to d_i. We then feed the sequences S to SkipGram Word2Vec algorithm. As a result, job titles and words in job descriptions are in the same vector space. We note that this job title insertion process results in fundamentally the same embedding as if we applied a joint embedding algorithm such as StarSpace (Wu et al., 2018) on the original (*job title*, *job description*) pairs.

Test job titles: We selected 1,000 testing job titles ranging widely in frequencies. Specifically, we sampled 125 job titles randomly from a frequency range $[2^r, 2^{r+1})$, where r is selected from the set $\{4, 5, 6, 7, 8, 9\}$, resulting in 750 job titles. Another 125 job titles were randomly sampled from range $[1, 2^4)$ and 125 random job titles from range $[2^{10}, \infty)$. Let us denote the testing job titles from the 8 different ranges as $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_8$, where $\mathcal{T}_1$ contains the rarest job titles and $\mathcal{T}_8$ contains the most frequent job titles.

Baselines: We compare the performance of FastText, Mimick, Mimick+IM, and GWR+IM, in addition to AutoCoder, which is a commercial software[3]. It implements a keyword search algorithm, capable of mapping any job title to the O*Net-SOC taxonomy. The software was originally developed for the U.S. Department of Labor. We note that due to the expensive and manual nature of our evaluation process, we decided to only include the original Mimick as a representative of the vanilla mimicking. Difference between Mimick and Mimick+IM should indicate the impact of IM, while comparison of Mimick+IM and GWR+IM should indicate the difference between Mimick and GWR.

Hyperparameters: We set the embedding dimension as 100 for all embedding models. We selected job titles and words with frequency larger than 2^3 to train FastText and SkipGram models. This resulted in a corpus comprising around $80{,}000$ unique job titles and $71{,}000$ unique regular words, used to train the embedding models. Hyperparameters for SkipGram, Mimick, and the IM models were set as in Section 4.

5.2 Evaluation Metrics

Evaluating job title normalization: We asked 6 human evaluators to judge the job title normalization quality of the 5 competing models. All human judgers are computer science graduate students who have broad knowledge about occupation titles and the job market. We designed our evaluation scheme similarly to (Liu et al., 2015). For each query job title, we listed randomly shuffled best matches from O*NET-SOC

[3]https://www.onetsocautocoder.com/plus/onetmatch, last accessed June 2020.

taxonomy produced by each model. Then, 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. The normalization quality of a model is the sum of the points divided by 1000, which is a size of the test set. The evaluation process is manual and very labor intensive, and it took more than 5 days to complete.

We note that there are cases when it is difficult for human evaluators to pick the most related category from the list, especially for rare titles, when all models produce different but similarly good results. For example, job titles can be blended such as *engineer and statistician*. Some models may normalize the job title to *engineers* and others to *statisticians*. Rare job titles may also be ambiguous, such as *energy healing and body work for women*. For such titles, human evaluators were allowed to search on Google to understand their meaning. We also observed that the categories in the O*NET-SOC taxonomy can be very similar. For instance, *acrobat software engineer* was normalized to *software developers-application*, *software developers-systems software*, *computer systems engineers/architects* by different models, all of which are equally acceptable. In such cases, human evaluators were also allowed to choose more than one category as co-winners.

5.3 Results

Table 4: Avg. testing scores of models by different human judgers

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 5: Avg. testing scores of models in different frequency ranges

Frequency	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$	$\mathcal{T}_5$	$\mathcal{T}_6$	$\mathcal{T}_7$	$\mathcal{T}_8$
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 6: Example of job title normalization results from 5 competing models

Raw title	Freq	AutoCoder	FastText	Mimic	Mimic+IM	GWR+IM
art teacher cochair sdh art dept	$\mathcal{T}_1$	Art, Drama, And Music Teachers, Postsecondary	Kindergarten Teachers, Except Special Educations	Education Teachers, Postsecondary	Art, Drama, And Music Teachers, Postsecondary	Art, Drama, And Music Teachers, Postsecondary
econometrics intern	$\mathcal{T}_2$	Not Classified	Economists	Social Sicence Research Assistants	Financial Quantitative Analysts	Economists
consultantowner	$\mathcal{T}_3$	Rehabilitation Counselors	Industrial-Organizational Psychologists	Wine Energy Project Managers	Logistics Managers	Chief Executives
founding board member	$\mathcal{T}_4$	Electrical And Electronic Equipment Assemblers	Fundraisers	Fundraisers	Chief Executives	Chief Executives
social media intern	$\mathcal{T}_7$	Public Relations Specialists	Public Relations Specialists	Advertising And Promotions Managers	Public Relations Specialists	Public Relations Specialists

Table 4 shows the scores given by 6 human evaluators. Five out of 6 human evaluators rated our IM models superior when it comes to normalizing job titles, with GWR+IM being the best model. Experimental results also show that by applying IM to the Mimick LSTM model, the quality of the normalization significantly improved, nearly 25% on average. We also report the average score by the 6 evaluators for each of the frequency ranges in Table 5. As expected, all models performed well on the

Table 7: Nearest job titles of different embedding models

Job titles	Word2Vec	Mimick	Mimick+IM
cto	*chief technology officer* *founder ceo* *ceo* *technical director* *founder and cto*	*tco* *ceo* *ctocoo* *vp ctp* *acting cto*	*ceo* *partner* *founder* *chief technology officer* *coo*
waitress	*waiter* *bartender* *hostess* *server* *barman*	*assistant to the mayor* *media relations assistant* *assistant to head officer of public relations* *catman assistant* *welders assistant*	*barman* *waiter* *front desk runner* *catering assistant* *food server*
sofftware engineeeeeer (OOV)	-	*research member senior software engineer* *researchersoftware engineer* *researcher software engineer* *external rd software engineeer* *researcher software engineering unit*	*c software engineer* *sr software engineer* *software rd engineer* *staff software developer* *engineer developer*
teaching assistant for principles of data science (OOV)	-	*teaching assistant computer science* *teaching assistant data analyst* *teaching assistant computer science dept* *teaching assistant cs* *teaching assistant dept of computer science*	*teaching assistant big data mining* *data science instructor* *instructor data scientist in residence* *teaching assistant dept of computer science* *research associate data science lecturer*

more frequent job titles. When compared to the commercial AutoCoder software, we can see that our best model is comparable on the frequent job titles. AutoCoder outperforms our model on $\mathcal{T}_6$ and $\mathcal{T}_7$ and our model is better on the most frequent group $\mathcal{T}_8$. However, for the less frequent job titles covering groups $\mathcal{T}_1$ to $\mathcal{T}_5$, our approach is significantly more accurate than AutoCoder. This is an impressive result considering that our model is trained in less than an hour on an unsupervised document corpus with minimal manual human efforts aside from evaluations, while AutoCoder is based on multiple years of painstaking tuning and updates.

Examples of job title normalization can be found in Table 6. We list raw job titles within different frequency ranges, from rare to frequent job titles. We can see that both Mimick+IM and GWR+IM improve the normalization results compared to the baselines in a number of cases across the frequency ranges. For example, for rare titles like *econometrics intern*, the commercial software AutoCoder fails to classify it to any category, while the proposed iterative mimicking models are able to assign a finance-related category for the title. For the title *consultantowner*, GWR+IM is the only model that recognizes *owner* and thus normalizes it to the most relevant category *Chief Executives*. Another interesting example is for the title *founding board member*, which appears in group $\mathcal{T}_4$. The IM-based methods assigned *Chief Executives* category, which is arguably more related to the raw job title than *Electrical And Electric Equipment Assemblers* inferred by AutoCoder or *Fundraisers* by Mimick and FastText approaches.

Finally, we performed a qualitative analysis of the job title embeddings to better understand differences between vanilla Mimick and Iterative Mimick. We list the top 5 nearest neighbors of example job titles based on the embeddings obtained from 3 models, shown in Table 7. As can be seen, although Mimick is able to mimic the nearest neighbors of Word2Vec to some extent, it relies too much on the surface form of words in job titles. By retrofitting Word2Vec and Char-BiLSTM, our proposed Iterative Mimicking was able to generate nearest neighbors with better overall quality than the competing approaches. For example, neighbors of *waitress* inferred by Mimick+IM closely resemble those produced by Word2Vec, and significantly outperform those produced by Mimick. When it comes to OOV titles that Word2Vec cannot handle, for *sofftware engineeeeeer* the proposed method provided much better neighbors than Mimick. When it comes to *teaching assistant for principles of data science*, we can see that neighbors produced by Mimick are reasonable, yet those in Mimick+IM are more fine-grained as they are all instructional positions in the field of data science.

6 Conclusion

In this paper we introduced a lightweight framework that enhances the existing mimicking models by iteratively retrofitting a word-level and a character-level embedding neural networks. We showed that the proposed algorithm can be successfully applied to the NLP task of word similarity, as well as the task of job title normalization, a challenging and very important problem in the e-recruitment domain.

References

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

Peter Cohan. 2018. Hired challenges linkedin in $400b market for talent recruiting.

Peter Elias, Margaret Birch, et al. 2010. Soc2010: revision of the standard occupational classification. *Economic & Labour Market Review*, 4(7):48–55.

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*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *AAAI*, pages 2741–2749.

Yeachan Kim, Kang-Min Kim, Ji-Min Lee, and SangKeun Lee. 2018. Learning to generate word representations using subword information. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2551–2561, Santa Fe, New Mexico, USA, August. Association for Computational Linguistics.

Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval.

Minh-Thang Luong and Christopher D Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

J. Moreno, R. Besançon, R. Beaumont, E. D'hondt, A. L. Ligozat, S. Rosset, X. Tannier, and B. Grau. 2017. Combining word and entity embeddings for entity linking. extended semantic web conference. Portoroz, Slovenia, Jan. Extended Semantic Web Conference, Springer.

Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking word embeddings using subword rnns. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 102–112.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 141–150.

Cicero D Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

Timo Schick and Hinrich Schütze. 2019. Attentive mimicking: Better word embeddings by attending to informative contexts. *arXiv preprint arXiv:1904.01617*.

Haoyu Wang, Eduard Hovy, and Mark Dredze. 2015. The hurricane sandy twitter corpus. In *Workshops at the twenty-ninth AAAI conference on artificial intelligence*.

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

Ledell Yu Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2018. Starspace: Embed all the things! In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ikuya Yamada, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. 2016. Joint learning of the embedding of words and entities for named entity disambiguation. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 250–259, Berlin, Germany, August. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Jinman Zhao, Sidharth Mudgal, and Yingyu Liang. 2018. Generalizing word embeddings using bag of subwords. *arXiv preprint arXiv:1809.04259*.