Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Improving Word Embeddings with Convolutional Feature Learning and Subword Information

Shaosheng Cao* and **Wei Lu**
Singapore University of Technology and Design
8 Somapah Road, Singapore 487372
{shaosheng_cao, luwei}@sutd.edu.sg

Abstract

We present a novel approach to learning word embeddings by exploring subword information (character n-gram, root/affix and inflections) and capturing the structural information of their context with convolutional feature learning. Specifically, we introduce a convolutional neural network architecture that allows us to measure structural information of context words and incorporate subword features conveying semantic, syntactic and morphological information related to the words. To assess the effectiveness of our model, we conduct extensive experiments on the standard word similarity and word analogy tasks. We showed improvements over existing state-of-the-art methods for learning word embeddings, including *skipgram*, *GloVe*, *char* n*-gram* and *DSSM*.

1. Introduction

Traditionally, words are represented using count-based vectors in a sparse high dimension space where each word in the vocabulary is represented by a single dimension. In contrast, word embeddings are low dimensional distributed representations of words in a real-valued vector space. Such dense representations can be learned from data, and they have proven to be effective in improving the performance on many natural language processing (NLP) tasks, *e.g.*, chunking, named entity recognition, and language modelling (Collobert and Weston 2008; Turian, Ratinov, and Bengio 2010; Socher et al. 2013). Thus the quest to find robust word embedding is of particular interest in the NLP community.

Most existing approaches regard words as individual atomic units when learning their embeddings. By doing so, the resulting word embeddings lack linguistic information that might be captured by the subword features. Subword information, such as roots, affixes, character n-grams and inflections, typically conveys useful features that captures derivational and/or inflectional morphological information that can be exploited in the embedding learning process. Additionally, important structural information such as word order needs to be captured to enrich the syntactic properties of the resulting word embeddings (Ling et al. 2015). It remains an active research topic in search of the effective techniques to represent linguistic units and to capture contextual/structural information in word embedding learning.

Similar to previous work (Bian, Gao, and Liu 2014; dos Santos and Zadrozny 2014; Chen et al. 2015; Cotterell and Schütze 2015; Bojanowski et al. 2016), we integrate the subword information in the first feature layer in our model training. In contrast, we improve upon the existing literature by explicitly adding derivational and inflectional morphology features to our subword feature layer. Different from popular approaches, *e.g.*, *continuous bag-of-words* and *skipgram*, that capture contextual information as a meta-prediction task, we make use of convolutional neural networks (CNNs) to capture the structural information of the context words when learning the word embeddings.

In this paper, we propose a novel approach that focuses on fusing fruitful subword information and learning convolutional features that improve the quality of word embeddings. We evaluate our approach on different word similarity and word analogy tasks using standard benchmark datasets and our model can outperform other state-of-the-art word embedding learning methods.

2. Related Work

In this section, we discuss related work on word embedding learning and the applications of convolutional feature learning.

2.1 Word Embeddings

The task of learning word embeddings has received a significant amount of interest recently. Bengio *et al.* (2006) proposed a neural network-based language model which can yield word embeddings. Improved approaches were later introduced to mitigate the high computational cost of neural language models (Morin and Bengio 2005; Mnih and Hinton 2009). Mikolov *et al.* (2010) introduced a recurrent neural network approach to learning word embeddings by capturing long distance relationship among words. Subsequently, the *skipgram* model was introduced and has since gained its popularity due to the open release of its implementation in the `word2vec` package[1] (Mikolov et al. 2013a; 2013b). Thereafter, *CWINDOW* and *Structured Skip-Ngram* were proposed as variant implementations of word2vec to

*Currently with AI Department, Ant Financial Services Group.

[1]The package also contains an alternative model called *CBOW*.

capture the structured information of context words (Ling et al. 2015). Song *et al.* (2014) introduced an alternative Deep Structured Semantic Model (DSSM) to learn word emebeddings via a deep fully-connected neural network argumented by a cosine similarity function to construct the loss function. In contrast to the prediction-based learning methods described above, *GloVe* (Pennington, Socher, and Manning 2014) showed that count-based methods with dimensionality reduction using global word co-occurrence information can achieve similar results.

Other than improving the techniques to model the contextual information, previous works have also leveraged existing lexical resources to improve word embeddings, such as *RCM* (Yu and Dredze 2014), knowledge embedding (Wang et al. 2014) and *SensEmbed* (Iacobacci, Pilehvar, and Navigli 2015). Without additional resources, previous works have also utilized character-level information to improve the word embeddings, *e.g.*, *CWE* (dos Santos and Zadrozny 2014) and *CharWNN* (Chen et al. 2015), morphological information (Cotterell and Schütze 2015). Most recently, Bojanowski *et al.* (2016) improved the original word2vec by introducing character n-gram information in addition to the surface word representations.

2.2 Convolutional Feature Learning

Convolutional neural networks have been widely used in various fields, including computer vision, speech, as well as NLP (LeCun and Bengio 1995; LeCun et al. 1998; Kalchbrenner, Grefenstette, and Blunsom 2014). For example, Shen *et al.* (2014) reported that convolution-based architectures can extract more information from *short texts* as compared to fully-connected structures in an information retrieval task. Meng *et al.* (2015) improved upon the neural network joint model in machine translation by introducing different convolutional neural network architectures to encode the source and target languages. Sun *et al.* (2015) proposed to model structural information of both context mentions and words by convolutional tensors. Fang *et al.* (2015) utilized convolution-pooling layers for measuring the similarities between an image and its caption.

Existing literature have demonstrated the potential benefits of using convolutional feature learning to extract structural information from short texts.

3. Model

Figure 1 illustrates our model for learning word embeddings. On the left, we first construct the input representation for each context word at the subword feature layer based on the character trigram features, the root and affix features and the inflection features. Next, we set a sliding window of size n to learn convolutional features from local word n-grams of context words, where n=3. Thereafter, a max-pooling layer is utilized to extract most informative features from among convolutions.

On the right, we construct the input representations for the current word, together with randomly selected words which serve as negative samples, using the same features described above for context words representation. Next, we feed each of them into a fully-connected non-linear projection layer to learn lower-dimensional representations and the final word embeddings are then obtained at the word embedding layer.

Using the max pooling layer (left) and word embedding (right), we define an objective function which maximizes the similarity between the context and the current word while minimizing the similarity between the context and all other negatively sampled words. We use back-propagation to learn model parameters.

3.1 Extracting Subword Features

The subword feature layer aims to encode useful linguistic information beyond the atomic surface word representations. We exploit the following features that captures varying granularity of the subword information. Figure 2 presents an example of the subword features and how the overlapping feature space highlights the similarity between the related words.

- Instead of using a surface word index, Huang *et al.* (2013) used letter trigrams for word hashing. As such, the dimension of the vocabulary table is reduced. Additionally, the trigrams capture certain morphological information when similar words access overlapping feature space. For example, in Figure 2(a), "*absorptive*" and "*absorption*" both share the features "#ab", "abs", "bso", "sor", "orp", "rpt". The letter trigrams shared between the words indicate that the words can potentially be related to each other. Letter trigrams are often generalized as character n-grams in many works (Cavnar, Trenkle, and others ; Bojanowski et al. 2016). We limit the order of n-grams to three to avoid sparsity issues with higher order n-grams.
- Bian *et al.* (2014) showed that root and affix information can be very useful for understanding the semantics and derivative morphology of a word. For example, in Figure 2(b), "absorbent", "absorbable", "absorbed" and "absorbingly" all have the same root "sorb". The various affixes provide derivational morphology that change the meaning of the words yet they remain semantically similar because of the root word.
- Different from prior work that conflates derivational and inflectional morphology, we explicitly handle inflectional forms (*i.e.*, non-derivational) of words by mapping them to a normalized root.[2] For nouns, we map them to their singular forms, *e.g.*, *"liquids"* gets normalized to *"liquid"*. For verbs, we map them to their infinitive forms, *e.g.*, *"absorbing", "absorbs"* and *"absorbed"* gets normalized to *"absorb"*.

Besides the above features, it is also possible to extract additional subword features. We can even extract certain task-specific or domain-specific features when we have such knowledge available. In this paper, we focus on using the above well-motivated features and focus the rest of the paper on learning general word embeddings.

[2]This is similar to the effects of lemmatization in studies like Trask *et al.* (2015), often information from derivational morphology is removed in lemma-based embeddings.

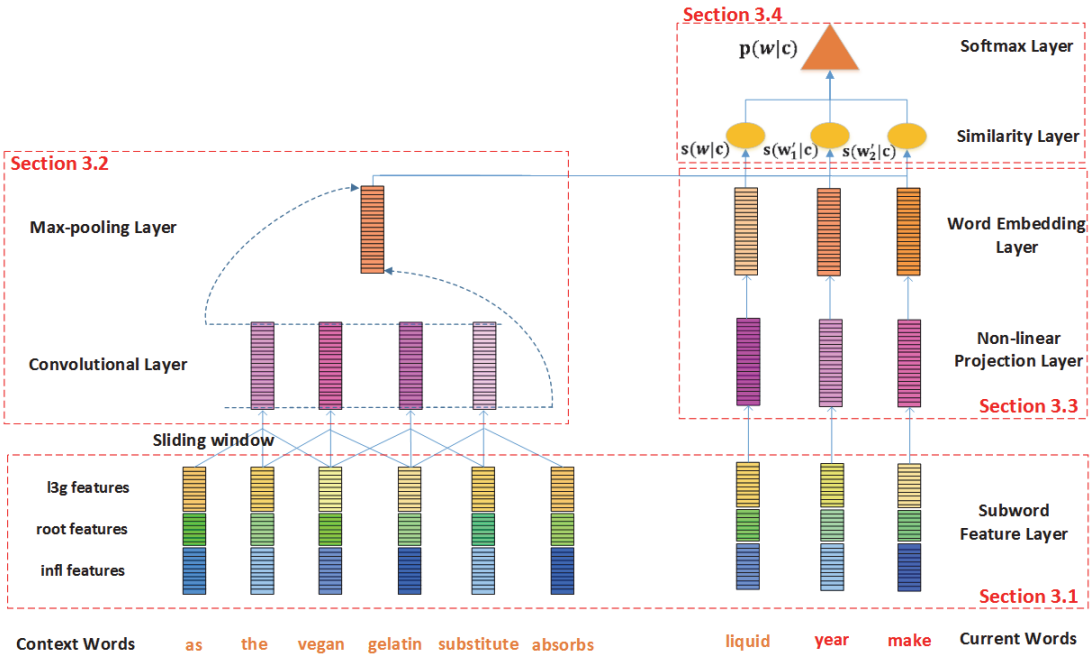
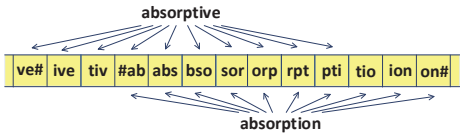
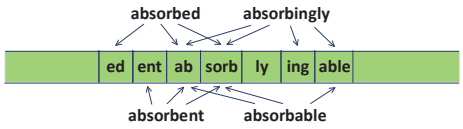


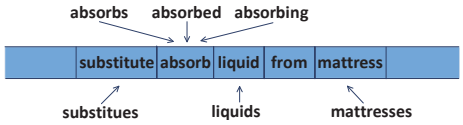
Figure 1: Our model for learning word embeddings. The context words are "*as the vegan gelatin substitute absorbs*", and the current word is "*liquid*". Two example negative samples are "*year*" and "*make*". For subword features, we use different colors to indicate different types of features (see Figure 2 for more details).



(a) Letter Trigrams



(b) Root and Derivational Affixes



(c) Inflectional Affixes

Figure 2: Subword features

3.2 Learning Context Embeddings

There are several variants of the convolution architectures, which are shown effective in modeling word sequences (Shen et al. 2014; Sun et al. 2015; Gao et al. 2014; Yih, He, and Meek 2014). In our model, we use x_i to represent the input features of the i-th context word, where x_i is a d-dimensional column vector. The vector representation of convolutional layer y can be defined as:

$$y_i = \sigma(\varpi \tilde{x}_i + \xi)$$

where y_i is the i-th component of the convolutional layer, and ϖ and ξ are weight parameters. Here $\tilde{x}_i$ is an nd-dimensional column vector, which is defined as follows:

$$\tilde{x}_i = x_{i:i+n-1} = [x_i^T, x_{i+1}^T, \ldots, x_{i+n-1}^T]^T$$

We note that such a representation of convolutional feature learning is able to capture structured context information by preserving certain ordering information amongst the context words. To combine the local information of word n-grams, we use a max-pooling layer, which can be formulated as:

$$c(j) = \max_{i=1,2,\ldots,t-n+1} \{y_i(j)\}$$

where t is the number of context words and c is the output representation from the max-pooling layer, which essentially exploits all the information of convolutional features captured by previous layers. This layer results in a *context embedding*. Here, the j-th entry in the context embedding is obtained by taking the max of the j-th entries of all representations learned from the convolutional layer.

3.3 Learning Current Word Embeddings

While convolutional neural networks can capture word ordering information, it is not necessary to use such an architecture to handle single words. Following the study of DSSM (Song et al. 2014), we utilize a two layered fully-connected neural network to learn current word embeddings

and the negative samples separately from the contextual embedding. Different from their work, our subword feature layer encodes morphosyntactic properties that improves the emedding quality. For the first layer, we have:

$$q = \sigma(\varsigma^1 x + \tau^1)$$

where x contains the subword features of the current word, which is also a d-dimensional vector. ς^1 and τ^1 are the parameters of first fully-connected layer. After this non-linear projection, x is reduced to a vector q. This vector is set as input to another fully connected layer, resulting in w:

$$w = \sigma(\varsigma^2 q + \tau^2)$$

where ς^2 and τ^2 are the parameters of second fully-connected layer. Now this representation w is exactly the current word embeddings. We set the dimensions to be of the same size for both the context embedding and the current word embedding such that we can calculate their similarity.

3.4 Conditional Probability Based on Softmax

Following (Song et al. 2014), to measure the similarity between current word and context words, *i.e.*, $p(\mathbf{w}|\mathbf{c})$, we define the *softmax function* with the *similarity layer* as follows:

$$p(\mathbf{w}|\mathbf{c}) = \frac{\exp(\gamma \cdot s(w,c))}{\exp(\gamma \cdot s(w,c)) + \sum_{\mathbf{w}'_j \in \mathbb{W}} \exp(\gamma \cdot s(w'_j,c))}$$

where $s(w,c)$ is the similarity function between the current word and its context words and γ is a temperature parameter that can magnify the influence of $s(\cdot,\cdot)$. We select negative samples $\mathbf{w}'_j$ from the collection of words $\mathbb{W}$ and calculate their similarity with context words.[3]

We define the loss function as the negative log-likelihood for each pair of current word **w** and its surrounding context words **c**. Following previous work (Gutmann and Hyvärinen 2012; Mnih and Kavukcuoglu 2013; Mikolov et al. 2013b; Cao, Lu, and Xu 2015), we use negative sampling when training our model.

The loss function associated with a particular $(\mathbf{w}, \mathbf{c})$ pair is given as:

$$l(\mathbf{w}, \mathbf{c}; \theta) = -\log p(\mathbf{w}|\mathbf{c}) = \log(1 + \sum_j \exp(-\gamma \cdot \Delta_j(w,c)))$$

where $\Delta_j(w,c)$ is the similarity difference between the positive sample and the j-th negative sample:

$$\Delta_j(w,c) = s(w,c) - s(w'_j,c)$$

and θ consists of the neural network parameters:

$$\theta = \{\varpi, \xi, \varsigma^1, \tau^1, \varsigma^2, \tau^2\}$$

We define the similarity function $s(\cdot,\cdot)$ as the cosine similarity between two vectors.

The overall loss function defined over the corpus is:

$$L(\theta) = \sum_{(\mathbf{w},\mathbf{c}) \in \mathcal{D}} \log(1 + \sum_j \exp(-\gamma \cdot \Delta_j(w,c)))$$

where $\mathcal{D}$ refers to the entire training corpus.

[3]The *negative samples* here refer to words that are randomly selected from the vocabulary where the current word is excluded.

4. Experiments

In this section, we describe the benchmarks and baseline algorithms, and then present and discuss the performance of our model.

4.1 Tasks

We conduct experiments on three tasks, word visualization, word similarity and word analogy, to assess the efficacy of the word embeddings trained using our approach. Our source codes will be released at `http://shelson.top`.

4.1.1 Word Visualization An intuitive way to understand the performance of a word embedding model is to visualize the resulting embeddings. We project our word emebeddings into a 2-dimensional space using the t-SNE toolbox (Van der Maaten and Hinton 2008). Words that share similar semantics will appear closer to each other in the 2-dimensional space allowing us to make qualitative assessment of the resulting word embeddings.

4.1.2 Word Similarity We evaluated our word embeddings using several word similarity datasets, *viz.*, WordSim353 (Finkelstein et al. 2001), MEN (Bruni et al. 2012), MT (Radinsky et al. 2011), Rel122 (Szumlanski, Gomez, and Sims 2013) and RG (Rubenstein and Goodenough 1965). Each dataset contains pairs of words together with their similarity scores. We use Spearman's rank correlation coefficient (Zar 1972) to evaluate the performance for each baseline algorithm.

$$\rho = 1 - \frac{6 \cdot \sum_{i=1}^{\varphi} (\beta_i - \hat{\beta}_i)^2}{\varphi(\varphi^2 - 1)}$$

where β_i is the benchmark similarity score between the two word embeddings in the i-th instance, and $\hat{\beta}_i$ is the cosine similarity between two words in the i-th instance based on the embeddings produced by each algorithm; φ refers to the total number of instances.

4.1.3 Word Analogy To assess the capability of our word embeddings for semantic deduction, we evaluate them using the word analogy task. This task provides three words **a**, **b**, **u** and requires a fourth word to be generated to satisfy the statement "***a** is to **b** that is similar to **u** is to **v***". For example, given "*China* is to *Beijing* as *Japan* is to _____", the model should generate "*Tokyo*" as the correct answer.

To find the most suitable word **v** based on word embeddings, Mikolov *et al.* (2013) introduced the 3CosAdd function that finds the optimal embedding v^* for the word $\mathbf{v}^*$ using the following:

$$v^* = \underset{v}{\operatorname{argmax}} \cos(v,b) - \cos(v,a) + \cos(v,u)$$

where $cos(\cdot,\cdot)$ refers to the cosine similarity function. Alternatively, Levy *et al.* (2014) proposed the 3CosMul function:

$$v^* = \underset{v}{\operatorname{argmax}} \frac{\cos(v,b) \cdot \cos(v,u)}{\cos(v,a) + 0.001}$$

where a, b, u and v are the embeddings of word **a**, **b**, **u** and **v**, and the objective is to find the best embedding of v^* as word $\mathbf{v}^*$.

In this paper, we evaluate our embeddings using both 3CosAdd and 3CosMul on two datasets of analogy questions proposed by Microsoft (Mikolov, Yih, and Zweig 2013) and Google (Mikolov et al. 2013a).

4.2 Experimental Settings

In this section, we describe the baseline systems, the training data and discuss the implementation details. We compare our word embeddings against the following baseline approaches:

- Skipgram[4] is an effective model for learning word embeddings (Mikolov et al. 2013a; 2013b). The model learns word embeddings by maximizing the conditional probability of a context word given the current word.
- Char n-gram[5] (Bojanowski et al. 2016) is a variant based on Skipgram model, which enriches word embeddings with character n-gram information. Fused by character n-grams, morphological information has been incorporated into word embeddings. This model reported good performance on small training datasets.
- GloVe[6] is another state-of-the-art model for learning word embeddings (Pennington, Socher, and Manning 2014). It leverages global count information aggregated from the entire corpus as word-word occurrence matrix to learn word embeddings.
- DSSM[7] is a model that explicitly learns the representations for both current word and its context using standard multi-layer neural networks (Song et al. 2014). In their model, they used letter trigrams as the hashed representation of the input layer. Empirically, their model has been shown to be effective at the word analogy task. This model is the closest to our approach. Unlike their model, we employ a convolutional layer for the context to capture structural context information and used more subword information in our input layer.

We use the publicly available enwik9 corpus as the training data; it consists of the first 1 billion (1×10^9) characters from Wikipedia[8]. We preprocess the corpus by lowercasing all words, keeping only words that contain alphabetical letters, removing the stop-words and selecting the top 50K frequent words.

To make a fair comparison, the context window size t is set to 5 from both sides as discussed in (Song et al. 2014) and we use the same dimension of word embeddings for each baseline algorithm and our model. For our model and DSSM, we set γ as 100, batch size as 1024, learning rate as 0.02, the number of negative samples as 100 and iteration epoch as 20. We train our model on one Nvidia GTX TITAN X GPU card. We use flatcat toolkit[9] (Grönroos et al. 2014) to obtain words' root/affix information. Besides, for every algorithm, we used the same training corpus without shuffling the dataset[10], and we only exploit the current word embeddings – that is, we do not merge them with the embeddings learned from the contexts[11].

[4]https://code.google.com/archive/p/word2vec/
[5]https://github.com/facebookresearch/fastText
[6]http://nlp.stanford.edu/projects/glove/
[7]http://research.microsoft.com/en-us/projects/dssm/
[8]http://mattmahoney.net/dc/enwik9.zip
[9]http://www.cis.hut.fi/projects/morpho/

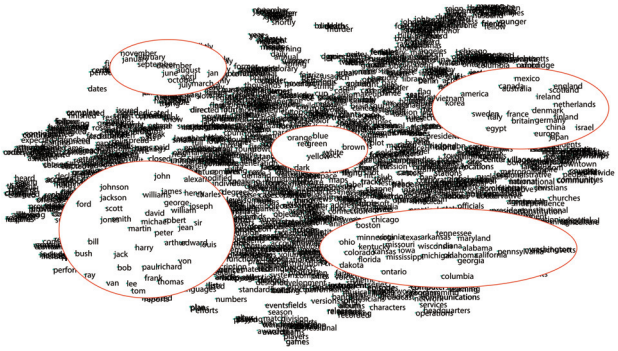


Figure 3: Visualization of word embeddings by t-SNE

Model \ $\rho \times 100$	F. et al. WS353	B. et al. MEN	R. et al. MT	S. et al. Rel122	R. et al. RG
skipgram (neg=10)	63.2	59.8	61.8	53.3	64.0
skipgram (neg=100)	59.5	58.1	60.8	53.8	64.1
char n-gram	59.5	21.7	60.3	51.0	51.3
GloVe	62.6	65.1	60.2	48.8	58.1
DSSM	51.1	41.5	44.1	31.6	37.7
Our Model	**65.7**	**69.0**	**64.8**	**57.6**	**72.7**

Table 1: Performance on word similarity task

4.3 Results

4.3.1 Results of Word Visualization Figure 3 shows the 2-dimensional projections of our word embeddings. We highlighted several regions and magnified them so that we can see them clearly. We observe that one of these clusters comprises common person names. Another cluster contains semantically related words, such as months, colors, countries, as well as states in the US.

This simple visualization provides a qualitative assessment to semantic word clusters learned by our word embeddings.

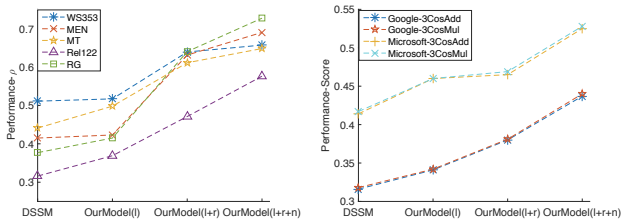
4.3.2 Results of Word Similarity Table 1 shows the empirical results on the word similarity task. We set the hyperparameter for number of negative samples as 10 and 100 for the skipgram model. From the results, the skipgram model yields a higher performance when the number of negative samples is set to 10. We also observe that the skipgram and GloVe model can outperform one another when evaluated on different datasets. The DSSM model does not perform well for the word similarity task, and char n-gram model falls behind skipgram and GloVe. Our model significantly outperforms all baseline approaches on every dataset.

[10]Empirically, we found that shuffling the dataset before training may lead to better results.

[11]As noted by Levy *et al.* (2015), merging the context word embedding and the current word embedding for the same word may lead to better results.

Model	Google		Microsoft	
	3CosAdd	3CosMul	3CosAdd	3CosMul
skipgram (neg=10)	29.1	27.9	22.8	22.6
skipgram (neg=100)	30.4	29.9	22.9	22.6
char n-gram	38.0	38.0	38.7	38.8
GloVe	41.1	33.9	28.0	24.5
DSSM	31.6	31.8	41.4	41.7
Our Model	**43.7**	**44.0**	**52.5**	**52.8**
Our Model (w/o inflection)	38.0	38.1	46.5	46.9

Table 2: Performance on word analogy task



(a) Performance on Word Similarity (b) Performance on Word Analogy

Figure 4: The effects of convolutional learning and importance of subword information

Model	Google		Microsoft	
	3CosAdd	3CosMul	3CosAdd	3CosMul
DSSM	31.6	31.8	41.4	41.7
DSSM (l+r+n)	41.7	41.7	48.5	48.9
Our Model (l+r+n)	**43.7**	**44.0**	**52.5**	**52.8**

Table 3: Performance comparison (on word analogy task) between Our Model (l+r+n) and DSSM (l+r+n) when all subword features are considered

4.3.3 Results of Word Analogy Table 2 describes the performance of each model on word analogy task. Microsoft's DSSM performs better than char n-gram on the Microsoft dataset, while char n-gram is better than DSSM on the Google dataset. Skipgram yields relatively lower results on both datasets, especially when the number of negative samples is set to 10, whereas GloVe has a better performance than skipgram model. Our model consistently outperforms all baseline approaches on both Microsoft and Google dataset.

One important reason is that our model encodes rich subword information, which is helpful for understanding such analogy questions as "*apple* is to *apples* as *pear* is to *pears*". Since we exploit inflection features for words, it might be the case that certain information required for performing the word analogy task might have already be present in the input feature representations. One might hypothesize that adding such features may improve the similarity of two variants of a word, such as "*apple*" and "*apples*". However, we note that the performance of the word analogy task is decided by the difference between word embeddings. For example, we are interested in measuring "vector(*apple*)-vector(*apples*)" and "vector(*pear*)-vector(*pears*)". While adding inflection features can improve the similarities between "*apple*" and "*apples*", it is not clear how this would affect the difference between their learned word vector representations.

To understand whether the improvement our model obtained is simply due to the addition of such inflection features, we performed further experiments by discarding inflection features. The last row of Table 2 reports the result of our model without the inflection features; our model yield poorer results across both datasets when inflection features are excluded. However, our degenerated model still outperforms all other baseline models.

From both Table 1 and Table 2 we can see that while the skipgram model is good at the word similarity task and the DSSM works well on the word analogy task, our model learns embeddings which remain robust for both task across all datasets.

4.4 Discussion

4.4.1 Effect of Convolutional Feature Learning and Importance of Subword Information We conduct experiments to illustrate the effectiveness of convolutional layers versus fully-connected layers and to understand the importance of subword information.

We compare the DSSM approach with a simple version of our model – Our Model (l), which only exploits letter trigrams as input features. The only difference between these two models is that the convolutional layers are used in our model when modeling context, while DSSM consists of fully-connected layers for modeling context. Figure 4 shows that the convolutional feature learning leads to better results, confirming the effectiveness of the convolutional layers in capturing structured context information as compared to fully-connected layers.

Aside from performance gains, Our Model (l) saves roughly 40% running time when compared with the DSSM model. Our Model (l+r) shows the additive results when we extend the letter trigrams with root/affix information and Our Model (l+r+n) yields the best results by further extending the model with the inflection features. These results confirm the importance of subword information when learning word embeddings.

We also compare Our Model and DSSM when all the subword features (*i.e.*, l+r+n) are considered. Table 3 shows the results on word analogy task. Our Model (l+r+n) performs better than DSSM (l+r+n) on such a task. Such results demonstrate the effectiveness of the convolutional feature learning. We also conduct experiments on word similarity task. Our Model (l+n+r) consistently outperforms DSSM (l+r+n).

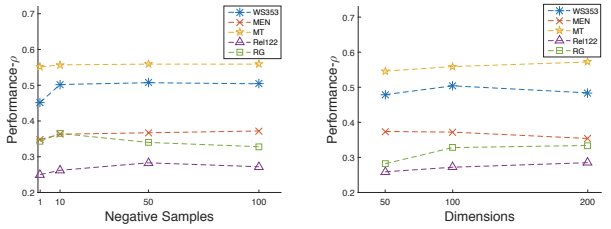
4.4.2 Performance v.s. Training Set Size To understand the robustness of our model, we conducted experiments on a smaller training corpus, enwik8[12], which is a collection of the first 100 million (1×10^8) characters in wikipedia (*i.e.*, the first 10% of enwik9).

From Table 4, we see that our model (enwik8) trained on a smaller training corpus performs relatively well on the Microsoft dataset as compared to the model trained on enwik9. Subword information plays a crucial role in mitigat-

[12]http://mattmahoney.net/dc/enwik8.zip

	Google		Microsoft	
Model	3CosAdd	3CosMul	3CosAdd	3CosMul
skipgram (neg=100)	6.2	6.1	6.6	6.5
char n-gram (epoch=1)	13.0	12.3	17.2	16.3
char n-gram (epoch=5)	36.7	36.7	41.5	42.0
GloVe	13.9	9.3	9.3	6.2
DSSM	28.2	28.2	37.4	37.7
Our Model	**37.7**	**37.7**	**50.0**	**50.2**
Our Model (enwik9)	43.7	44.0	52.5	52.8

Table 4: Performance on word analogy task trained on enwik8



(a) Performance over negative samples (b) Performance over dimensions

Figure 5: Performance over negative samples or dimensions on word similarity task

ing the lack of of training data. The same robustness to data sparsity is evident when we see that DSSM that uses letter trigrams outperforms the skipgram model when trained on the enwik8 corpus. Our model has the capability to integrate heterogeneous subword information to overcome the insufficient data issue, thus leading to an improved performance.

Interestingly, to overcome the data sparsity issue, the performance of the char n-gram model can be improved when the training epoch is enlarged within a certain range. The best performing results are achieved when the number of epoch is set to 5. We also change the training epoch hyperparameter of GloVe, but there is almost no improvement.

4.4.3 Performance v.s. Dimensions or Negative Samples We explore the relation between the performance of our model on the word similarity task with varying number of negative samples and the dimensions. Figure 5 shows how the performance fluctuates on the word similarity task when different negative samples and different dimensions are used.

We observe that the performance of our model degrades when the number of negative samples is either too small or too large. Our model obtains optimal results when the number of negative samples is set to a value between 50 and 100. Regarding the dimension of the word embeddings, the best performance can be obtained when the dimension is set to a value between 100 and 200.

5. Conclusion

In this paper, we proposed a novel approach for learning word embeddings. The model is able to integrate various subword information effectively into the learning process of word embeddings, and at the same time can capture certain structured information from the context. Particularly, we illustrated the effectiveness of convolutional feature learning and importance of subword information when learning word embeddings, and present a comprehensive analysis on the performance with varying hyperparameters. We demonstrated that our model outperforms state-of-art word embedding approaches and empirically investigated the effects of several variables that can affect the quality of word embeddings. In the future, we will explore other subword information, and investigate its integration with alternative neural architectures that are capable of capturing structured contextual information. We are also interested in investigating methods for effectively exploiting subword information for learning word embeddings for certain languages other than English (such as Chinese).

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and Liling Tan for his helpful suggestions. This work is supported by MOE Tier 1 grant SUTDT12015008.

References

Bengio, Y.; Schwenk, H.; Senécal, J.-S.; Morin, F.; and Gauvain, J.-L. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*. 137–186.

Bian, J.; Gao, B.; and Liu, T.-Y. 2014. Knowledge-powered deep learning for word embedding. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 132–148.

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Bruni, E.; Boleda, G.; Baroni, M.; and Tran, N.-K. 2012. Distributional semantics in technicolor. In *ACL*, 136–145.

Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*, 891–900.

Cavnar, W. B.; Trenkle, J. M.; et al. N-gram-based text categorization. *Ann Arbor MI* 48113(2):161–175.

Chen, X.; Xu, L.; Liu, Z.; Sun, M.; and Luan, H. 2015. Joint learning of character and word embeddings. In *IJCAI*, 1236–1242.

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

Cotterell, R., and Schütze, H. 2015. Morphological word-embeddings. In *NAACL*.

dos Santos, C. N., and Zadrozny, B. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 25th international conference on Machine learning*, 1818–1826.

Fang, H.; Gupta, S.; Iandola, F.; Srivastava, R. K.; Deng, L.; Dollár, P.; Gao, J.; He, X.; Mitchell, M.; Platt, J. C.; et al.

2015. From captions to visual concepts and back. In *CVPR*, 1473–1482.

Finkelstein, L.; Gabrilovich, E.; Matias, Y.; Rivlin, E.; Solan, Z.; Wolfman, G.; and Ruppin, E. 2001. Placing search in context: The concept revisited. In *WWW*, 406–414.

Gao, J.; Deng, L.; Gamon, M.; He, X.; and Pantel, P. 2014. Modeling interestingness with deep neural networks. In *EMNLP*.

Grönroos, S.-A.; Virpioja, S.; Smit, P.; and Kurimo, M. 2014. Morfessor flatcat: An hmm-based method for unsupervised and semi-supervised learning of morphology. In *COLING*, 1177–1185.

Gutmann, M. U., and Hyvärinen, A. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 307–361.

Huang, P.-S.; He, X.; Gao, J.; Deng, L.; Acero, A.; and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, 2333–2338.

Iacobacci, I.; Pilehvar, M. T.; and Navigli, R. 2015. Sensembed: learning sense embeddings for word and relational similarity. In *ACL*, 95–105.

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A convolutional neural network for modelling sentences. *ACL*.

LeCun, Y., and Bengio, Y. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 1995.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 2278–2324.

Levy, O.; Goldberg, Y.; and Dagan, I. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics* 211–225.

Levy, O.; Goldberg, Y.; and Ramat-Gan, I. 2014. Linguistic regularities in sparse and explicit word representations. In *CoNLL*, 171–180.

Ling, W.; Dyer, C.; Black, A.; and Trancoso, I. 2015. Two/too simple adaptations of word2vec for syntax problems. In *NAACL*, 1299–1304.

Meng, F.; Lu, Z.; Wang, M.; Li, H.; Jiang, W.; and Liu, Q. 2015. Encoding source language with convolutional neural network for machine translation. *ACL*.

Mikolov, T.; Karafiát, M.; Burget, L.; Cernockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *INTERSPEECH*, 3.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Mikolov, T.; Yih, W.-t.; and Zweig, G. 2013. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, 746–751.

Mnih, A., and Hinton, G. E. 2009. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, 1081–1088.

Mnih, A., and Kavukcuoglu, K. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in neural information processing systems*, 2265–2273.

Morin, F., and Bengio, Y. 2005. Hierarchical probabilistic neural network language model. In *Aistats*, 246–252.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP*, 1532–1543.

Radinsky, K.; Agichtein, E.; Gabrilovich, E.; and Markovitch, S. 2011. A word at a time: computing word relatedness using temporal semantic analysis. In *WWW*, 337–346.

Rubenstein, H., and Goodenough, J. B. 1965. Contextual correlates of synonymy. *Communications of the ACM* 627–633.

Shen, Y.; He, X.; Gao, J.; Deng, L.; and Mesnil, G. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, 101–110.

Socher, R.; Bauer, J.; Manning, C. D.; and Ng, A. Y. 2013. Parsing with compositional vector grammars. In *ACL*, 455–465.

Song, X.; He, X.; Gao, J.; and Deng, L. 2014. Unsupervised learning of word semantic embedding using the deep structured semantic model. Technical report, Tech. Rep. MSR-TR-2014-109.

Sun, Y.; Lin, L.; Tang, D.; Yang, N.; Ji, Z.; and Wang, X. 2015. Modeling mention, context and entity with neural networks for entity disambiguation. In *IJCAI*, 1333–1339.

Szumlanski, S. R.; Gomez, F.; and Sims, V. K. 2013. A new set of norms for semantic relatedness measures. In *ACL*, 890–895.

Trask, A.; Michalak, P.; and Liu, J. 2015. sense2vec - A fast and accurate method for word sense disambiguation in neural word embeddings. *CoRR* abs/1511.06388.

Turian, J.; Ratinov, L.; and Bengio, Y. 2010. Word representations: a simple and general method for semi-supervised learning. In *ACL*, 384–394.

Van der Maaten, L., and Hinton, G. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research* 85.

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph and text jointly embedding. In *EMNLP*, 1591–1601.

Yih, W.-t.; He, X.; and Meek, C. 2014. Semantic parsing for single-relation question answering. In *ACL*, 643–648.

Yu, M., and Dredze, M. 2014. Improving lexical embeddings with semantic knowledge. In *ACL*, 545–550.

Zar, J. H. 1972. Significance testing of the spearman rank correlation coefficient. *Journal of the American Statistical Association* 578–580.