Socialized Word Embeddings

Ziqian Zeng\textsuperscript{1}, Yichun Yin\textsuperscript{2,1}, Yangqiu Song\textsuperscript{1} and Ming Zhang\textsuperscript{2}

\textsuperscript{1}Department of CSE, HKUST, Hong Kong.
\textsuperscript{2}School of EECS, Peking University, China.
zzengae@cse.ust.hk, yichunyin@pku.edu.cn, yqsong@cse.ust.hk, mzhang_cs@pku.edu.cn

Abstract

Word embeddings have attracted a lot of attention. On social media, each user’s language use can be significantly affected by the user’s friends. In this paper, we propose a socialized word embedding algorithm which can consider both user’s personal characteristics of language use and the user’s social relationship on social media. To incorporate personal characteristics, we propose to use a user vector to represent each user. Then for each user, the word embeddings are trained based on each user’s corpus by combining the global word vectors and local user vector. To incorporate social relationship, we add a regularization term to impose similarity between two friends. In this way, we can train the global word vectors and user vectors jointly. To demonstrate the effectiveness, we used the latest large-scale Yelp data to train our vectors, and designed several experiments to show how user vectors affect the results.

1 Introduction

Social media has become one of the major streams of publishing natural language texts on the Web. Users are using social media platforms such as Facebook, Twitter, and Yelp to receive friends’ update about their lives and learn knowledge from friends. A well-known effect of social network is the concept of “homophily,” which has been developed in psychology [Lazarsfeld and Merton, 1954] and observed in social network [McPherson et al., 2001]. It indicates that users being friends tend to share similar opinions or topics. On the other hand, social linguists also found that natural language understanding in a society requires the understanding of the social networks in which the language is embedded. A network could be loose or tight depending on how members interact with each other [Holmes, 2012] and may affect speech patterns adopted by a speaker [Dubois and Horvath, 1998]. People’s social language code can depend on their educations, working classes, ages, genders, and other social groups. For example, some group of people say “I am gonna...” while others say “I am going to...,” and some group of people say “It looks like...” while others say “It looks as if....” Thus, there is a need of development of computational sociolinguistics for social media texts [Nguyen et al., 2016].

Text data representation plays a key role in computational linguistics and natural language processing. Recently, distributed word representation based on neural network language models (NNLMs) [Bengio et al., 2003] has attracted a lot of attention, since such dense word vector representation in high-dimensional (but much lower dimensional than one-hot representation) space can provide reduced computational complexity and improve generalization ability of machine learning models for many downstream tasks [Collobert et al., 2011]. Word embeddings, such as word2vec [Mikolov et al., 2013a], simplifies the NNLM architecture by reducing the latent variables and relaxing the constraint of context words being previous words. Thus, it can adopt efficient training algorithm to train over large-scale corpus and is widely used in many applications, such as information extraction [Turian et al., 2010], sentiment analysis [Tang et al., 2015a], search engine [Mitra and Craswell, 2017], etc. When applying text representation learning to social media texts, a key problem is to handle the aforementioned difference for different social groups of people.

In this paper, we present a socialized word embedding approach to generate social-dependent word embedding vectors for words shown in social media. Our model adopts the simplest but most efficient and effective word embeddings model used in word2vec as a base model. Then we apply personalization to words by incorporating a user vector for each social user. The word embeddings for each social user will not only depend on the global word embeddings, but also the user vector. To incorporate the friend relationship, we add a social regularization term when we train both the global word vectors and local user vectors. To demonstrate our socialized word embeddings, we use the Yelp business review data to train our vectors. We first verify that by incorporating the user vectors into word embedding model, we can significantly reduce the perplexity on test data. Then we use the review rating prediction task to verify that the global word vector plus the local user vector can generate a better representation for the sentiment classification problem. In addition, we test whether we can use the user vectors learned in unsupervised way as attention vectors in deep neural network based models. The code is available at https://github.com/HKUST-KnowComp/SocializedWordEmbeddings.
2 Related Work

In this section, we review our related work in three categories.

2.1 Personalized and Socialized Language Models

Language model is a fundamental natural language processing problem and has been well studied for many years [Jurafsky and Martin, 2008]. It is natural to extend language models to be personalized or socialized since we know that every person in the world has his/her own speech or writing patterns and can be affected by others. Here we distinguish personalized and socialized language models to be whether they consider the social relationships of users on social media. Personalized language models were mainly applied to Web search [Croft et al., 2001; Song et al., 2010; Sontag et al., 2012] or collaborative search (where user groups are clustered based on user behaviors instead of explicit connections) [Sun et al., 2005; Teevan et al., 2009; Xue et al., 2009]. Socialized language models have recently been developed, which were also applied to search problems, but to social media text search [Vosecky et al., 2014; Huang et al., 2014; Yan et al., 2016]. The socialization of language models can incorporate social friends’ information as a smoothing factor to improve the language sparsity problem.

2.2 Distributed Word Representation

NNLM has attracted more attention recently to generate distributed word representations. It was first successfully trained with large corpus by Bengio et al. (2003) to obtain representations of words. Later, it was further scaled up and studied [Morin and Bengio, 2005; Mnih and Hinton, 2008; Collobert et al., 2011] and pushed to limit to train on Web-scale corpus [Józefowicz et al., 2016]. Word embedding, i.e., word2vec [Mikolov et al., 2013b; Mikolov et al., 2013a], simplifies the NNLM problem and has been shown to be efficient for training over very large-scale corpus. Inspired by word2vec, many alternatives of embedding approaches have been proposed [Pennington et al., 2014; Levy and Goldberg, 2014] and a comprehensive study has shown that word2vec can be very powerful when all the models are tuned on the same corpus with best hyper-parameters [Levy et al., 2015]. Recently, a personalized NNLM has been proposed [Wen et al., 2013] which trains an NNLM over a global data set and adapts that language model to small data for each person. User embeddings were also introduced into word embeddings [Yu et al., 2016; Song and Lee, 2017] while the impact of socialization still remains unclear.

2.3 Multi-task Learning on Social networks

Our work is also related to multi-task learning [Caruana, 1997]. Multi-task learning is a learning setting where different learning tasks are performed simultaneously. Multi-task learning can be naturally applied to social media applications, since the task related to each person can be personalized. For example, social media topic classification and sentiment classification can be personalized [Hu et al., 2013; Song et al., 2013; Wu and Huang, 2016]. Recently, deep learning based models also adopt a separate learning mechanism to model different users along global text representation to improve sentiment classification results [Tang et al., 2015c; Tang et al., 2015b], which can be regarded as a multi-task learning. Moreover, attention models can be enabled with multi-tasks (each task is a user attention) [Chen et al., 2016]. Compared to all the above multi-task learning approaches which need supervision for all the tasks, we are unsupervised learning. Thus, our socialized word embeddings can be used for many down-stream tasks.

3 Socialized Word Embedding

In this section, we introduce our socialized word embedding model and show the optimization algorithm.

3.1 Personalization

To train the word embeddings, we consider the continuous bag-of-words (CBOW) model [Mikolov et al., 2013a] as our base model. We denote the training corpus as \( \mathcal{W} \). Different from original word embedding training, we need to distinguish the training sets for different users. Suppose we have \( N \) users \( u_1, \ldots, u_N \). Since our word embedding training only considers a word \( w_j \)’s context, i.e., \( C(w_j) = \{w_j-s, \ldots, w_j+s\} \) where \( s \) is the half window size, we can aggregate all documents as a corpus \( \mathcal{W}_i \) for user \( u_i \). We also represent \( u_i \)’s neighborhood set as \( N_i = \{u_{i+1}, \ldots, u_{i+L} \} \), where \( N_i \) is the number of neighborhoods.

In our CBOW based model, given a sequence of training words, the first objective is to maximize the log-likelihood:

\[
\mathcal{J}_1 = \sum_{i}^{N} \sum_{w_j \in \mathcal{W}_i} \log P(w_j|C(w_j, u_i)),
\]

which is performed over all users \( u_1, \ldots, u_N \). Different from CBOW model, here we specify that the context words are user-dependent. This means for each user \( u_i \), s/he will think about a predicted word given the global words meanings and customize them to his/her own preference. More specifically, suppose we use \( w_j \in \mathbb{R}^d \), where \( d \) is the dimensionality of vector \( w_j \), as the global vector representation of \( w \). We also use a user vector \( u_i \in \mathbb{R}^d \) to represent the user. Then we combine the global word vector and user vector as a new vector of word \( w_j \): \( w_j^{(i)} = w_j + u_i \). If we have a sequence of words \( w_{j-s}^{(i)}, \ldots, w_{j+s}^{(i)} \), then the combined word vectors for user \( u_i \) are represented as \( w_{j-s}^{(i)}, \ldots, w_{j+s}^{(i)} \). An illustration of such composition is shown in Figure 1. The CBOW model is difficult to optimize when the vocabulary size is large, since the computation of \( \log P(w_j|C(w_j, u_i)) \) needs the normalization over all the words in the vocabulary. Thus, originally there are two techniques used for optimizing the problem, hierarchical softmax [Morin and Bengio, 2005; Mnih and Hinton, 2008] and negative sampling [Mikolov et al., 2013a]. Here we use hierarchical softmax as an example. Suppose we have a tree over words, e.g., the Huffman tree [Huffman, 1952] built based on word frequencies. Then let \( l_k^{(w)} \) be Huffman code of \( k \)-th node in the path from root to leaf node identical to \( w_j \), where \( l_k^{(w)} \in \{0, 1\}, k \in \{2, \ldots, L_w\} \), and \( L_w \) is the length of the path. Then
Figure 1: Illustration of Socialized Word Embeddings.

The objective function can be rewritten as:

$$J_1 = -\sum_{i} \sum_{w_j \in \mathcal{W}_i} \sum_{k=2}^{L^w_i} \log P(l^w_{k-i} | \mathbf{X}^{(i)}_{w_j}, \theta_{k-1}^w),$$  

(2)

where $\mathbf{X}^{(i)}_{w_j} = \sum_{w \in \mathcal{C}(w_j)} \mathbf{w}^{(i)} = \sum_{w \in \mathcal{C}(w_j)} (\mathbf{w} + \mathbf{u}_i)$. Thus, hierarchical softmax converts the softmax probability $P(w_j | \mathcal{C}(w_j), u_i)$ of predicting the word $w_j$ based on its contexts $\mathcal{C}(w_j)$ and user vector $u_i$ to be a series of binary classification problems to predict words through the path from root to the node of the word. For each binary classification at node $k$ in the path, we have:

$$\ell(w_j, u_i, l_k) = -\log P(l^w_{k-i} | \mathbf{X}^{(i)}_{w_j}, \theta_{k-1}^w)$$

$$= - (1 - l^w_{k-i}) \cdot \log [\sigma(\mathbf{X}^{(i)}_{w_j}^T \theta_{k-1}^w)]$$

$$- l^w_{k-i} \cdot \log [1 - \sigma(\mathbf{X}^{(i)}_{w_j}^T \theta_{k-1}^w)],$$

(3)

where $\sigma(x) = 1/(1 + \exp(-x))$ is the logistic function. We can see that if the Huffman code $l^w_{k-i} = 1$ we classify the current node $k$ in the path as true and if $l^w_{k-i} = 0$ we classify it to be false. Then we have a series of classification problems following the path. So here in the model, we have a series of “hidden” parameter vectors $\theta_{k-1}$ to perform classification. In the Huffman tree, deeper leaf nodes mean low-frequency words, and thus will follow longer paths.

To minimize the objective function $J_1$, stochastic gradient descent (SGD) is applied. The parameter vectors $\theta_{k-1}$, word vectors in the context, and user vectors can be updated as follows:

$$\theta_{k-1}^w := \theta_{k-1}^w - \eta_1 [1 - l^w_{k-i} - \sigma(\mathbf{X}^{(i)}_{w_j}^T \theta_{k-1}^w)] \mathbf{X}^{(i)}_{w_j},$$

$$\mathbf{w} := \mathbf{w} - \eta_1 \sum_{k=2}^{L^w_i} \frac{\partial \ell(w_j, u_i, l_k)}{\partial \mathbf{X}^{(i)}_{w_j}}, \text{ and}$$

$$\mathbf{u}_i := \mathbf{u}_i - \eta_1 c \sum_{k=2}^{L^w_i} \frac{\partial \ell(w_j, u_i, l_k)}{\partial \mathbf{X}^{(i)}_{w_j}},$$

(4)

where $c = 2s$ is the window size and $\eta_1$ is a degenerative learning rate.

**3.2 Social Regularization**

Till now we have introduced how we personalize each user’s word vector by adding a local user vector to the global word vectors. In practice, the social relations are also useful to improve the learning results of user vectors, since some of the users may only publish a few documents. Thus, we propose a social regularization term to all the user vectors:

$$J_2 = \sum_{i \neq j \in \mathcal{N}_i} ||\mathbf{u}_i - \mathbf{u}_j||_2,$$

(5)

where we minimize the $L_2$-norm of the difference between two users with a social relation. The illustration of social regularization is shown in Figure 1.

We can also apply SGD to $J_2$, where we have:

$$\mathbf{u}_i := \mathbf{u}_i - \eta_2 \sum_{j \in \mathcal{N}_i} (\mathbf{u}_j - \mathbf{u}_i),$$

$$\mathbf{u}_j := \mathbf{u}_j - \eta_2 (\mathbf{u}_i - \mathbf{u}_j).$$

Here when we are working with a document published by $u_i$, the user vector $\mathbf{u}_i$ should be updated based on all its friends $u_j$’s vectors while user vector $u_i$ is updated only based on $u_i$.

We can combine the second objective function with the first one and perform SGD alternatively for global word vectors, parameter vectors, and local user vectors. However, as shown in Eqs. (4) and (6), the user vectors will be updated much more times than word vectors. Originally in CBOW optimization, all the global word vectors are not constrained since the scale of word vectors can be bounded by the learning rate (combined with frequencies of words). Here we propose a constraint for user vectors to make the numerical optimization stable. The overall cost function is then:

$$J = J_1 + \lambda J_2$$

$$s.t. \forall \mathbf{u}_i, ||\mathbf{u}_i||_2 \leq r,$$

(7)

where $r$ is the constraint for $u_i$’s $L_2$-norm. $r$ is data dependent and should be tuned with development set. In practice, we solve this by SGD with re-projection [Goodfellow et al., 2016]. The detailed algorithm is shown in Algorithm 1, which is called socialized word embedding algorithm.

### 4 Experiments

In this section, we show the experiments to demonstrate the effectiveness of socialized word embeddings.

**4.1 Datasets**

We use the Yelp Challenge datasets as our evaluation sets. At Yelp, users can write reviews for some businesses, e.g.,

Algorithm 1 Socialized Word Embedding Algorithm.

**Input:** Social media data with N users: $u_1, \ldots, u_N$, where each user has a corpus $W_i = \{d_{i1}, \ldots, d_{iM_i}\}$ where $M_i$ is the number of documents written by $u_i$.

**Initialize:** Maximum iteration $T$, learning rates $\eta_1, \eta_2$, social regularization weight $\lambda$, size of context window $c$, and constraint parameter $r$.

if iteration $t < T$ then
  for all $u_i$ do
    for all $d_j$ in $W_i$ do
      $\theta_{k-1}^{w_j} := \theta_{k-1}^{w_j} - \eta_1 (1 - c^{w_j}) \sigma(X^{w_j})^T \theta_{k-1}^{w_j}) X^{w_j}$
      $w := w - \eta_1 \sum_{k=2}^{L} \frac{\partial \ell(w_j, u_i, d_k)}{\partial \theta_{k-1}^{w_j}} X^{w_j}$
      $u_i := u_i - \eta_1 \sum_{k=2}^{L} \frac{\partial \ell(w_j, u_i, d_k)}{\partial \theta_{k-1}^{w_j}} X^{w_j}$
    end for
    if $||u_i|| > r$ then
      $u_i := \frac{u_i}{||u_i||}$
    end if
  end for
end if

Output: Word embeddings $w_j$ and user embeddings $u_i$.

### 4.3 Perplexity

In the first experiment, we test the fitness of test data of different word embeddings. We use perplexity as a measure for the experiments. However, since essentially our model and word2vec are not language models, which do not directly optimize the perplexity, this experiment is only conducted to show insight of different hyper-parameter settings of our model. By definition, perplexity is used to evaluate how good a model is to predict the current word based on several previous words. Since we are using a sliding window of size $s = 5$ to train all the word embeddings, we present the 6-gram perplexity. We train the word embeddings based on the whole training data. To improve efficiency of testing different hyper-parameters, for both development set and test set, we randomly sample one sentence for each user to evaluate the sentence based perplexity.

The results of perplexities are shown in Figure 2. Note that our perplexity values are high if compared with the values normally shown in the literature. There are mainly two reasons. First, our model does not directly optimize perplexity as language models do. Thus, the model may not fit the data very well. Second, the Yelp data we used are noisier than formal language. Thus, the perplexity should be reasonably high. This is also verified in [Yan et al., 2016]. Empirically, although word2vec does not optimize perplexity directly, it has a good trade-off between the training cost and test effectiveness as word representation for other downstream tasks.

We first show the perplexity results with fixed $r$ ($L_2$-norm constraints for user vectors) and varied $\lambda$ (social regularization parameter) in Figures 2(a) and 2(b) on YelpR8 and YelpR9 respectively. They show that for both datasets, when $\lambda = 0$, which is the personalization case, the perplexity can be improved with user vector for each user. Moreover, when increasing the social regularization term, the perplexity can be further improved first, but will show no more benefit when $\lambda$ is too large. This reason may be that, when fixing the user vector size, increasing the social regularization will tend to first refine each use vector by its friend, but eventually impose all the user vector to become as similar as possible, which will again under fit the data.

Then we show the results of varying $r$ with fixed $\lambda$ in Figure 2(c). It shows that when increasing the regularization constraint $r$, the perplexity is first reduced. Then when continuing increasing $r$, again it will make the perplexity worse. If the user vector’s norm becomes too large, it will dominate the word vectors when optimizing the cost function. Thus, in our algorithm, the parameters $r$ and $\lambda$ are coupled. There is no single trend to make the perplexity continues to decrease. We performed grid search in the range of $\{2^{-5}, \ldots, 2^5\}$ over both $r$ and $\lambda$ based on validation set, select the best hyper-parameters, and show the final results on test set in Table 2. From the table we can see that, the improvements of both personalization and socialization are significant.

### 4.4 Sentiment Classification

In this section, we test our socialized word embeddings with a downstream task, rate prediction for Yelp reviews. On Yelp website, a user is allowed to write reviews to businesses. At the same time, the user can provide a rating score to the restaurant, hotel, etc. Users can also follow each other to receive information from friends (some of the friend information is from Facebook or other social networks given its infrastructure). The statistics of our data are shown in Table 1. From the Table we can see that the size of data released by Yelp increases a lot over years. This is one of the largest social network data that are available with a lot of texts. We randomly split the data to be 8:1:1 for training, developing, and testing. All the following results are based on this fixed segmentation.

### 4.2 Experimental Settings

We trained all the word embedding models based on the training data we split out. For downstream applications, we will change the setting according to different intentions we want to test our model. The word embedding model was written with C language based on the original release of word2vec\(^2\). All the experiments were conducted on a SuperMicro server with E5-2640v4 CPUs. To make fair comparison, we set the hyper-parameters for original word2vec to be the same for all the models. For example, the window size was set to be five and the dimension of embeddings was set to be 100. We used CBOW model for all the word embeddings. Empirically, we found Skipgram model had comparable results as CBOW for Yelp datasets.

\(^2\)For a back up of the code, please see: https://github.com/dav/word2vec.
Varying $r$ with fixed $\lambda$ (YelpR8)
Varying $\lambda$ with fixed $r$ (YelpR9)
Varying $r$ with fixed $\lambda = 0.5$

Figure 2: Perplexity results (lower means better) on development set.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>YelpR8</th>
<th>YelpR9</th>
</tr>
</thead>
<tbody>
<tr>
<td>word2vec</td>
<td>45,914.3</td>
<td>57,742.6</td>
</tr>
<tr>
<td>personalization</td>
<td>33,887.2</td>
<td>43,476.8</td>
</tr>
<tr>
<td>socialization</td>
<td>24,301.6</td>
<td>31,074.6</td>
</tr>
</tbody>
</table>

Table 2: Test set perplexity.

<table>
<thead>
<tr>
<th></th>
<th>YelpR8</th>
<th></th>
<th>YelpR9</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#Users</td>
<td>2,125</td>
<td>8,214</td>
<td>3,480</td>
<td>13,288</td>
</tr>
<tr>
<td>#Avg. Reviews</td>
<td>28.5</td>
<td>7.4</td>
<td>28.2</td>
<td>7.4</td>
</tr>
<tr>
<td>#Avg. Friends</td>
<td>135.3</td>
<td>47.9</td>
<td>268.0</td>
<td>130.4</td>
</tr>
</tbody>
</table>

Table 3: Statistics of subsets of one-fifth of training data.

business. We follow the task used in [Tang et al., 2015c; Tang et al., 2015a; Tang et al., 2015b; Chen et al., 2016], which is long document sentiment classification. When user information is desired to be included, previous studies simply preprocess the data and work only on the data containing sufficient user information (e.g., containing only 4,818 users) [Tang et al., 2015b; Chen et al., 2016]. In this work, we are curious about how good it will be if the whole dataset or partial dataset can be used. We follow the semi-supervised learning setting claimed by Turian et al. (2010), where the word embeddings are trained in an unsupervised way with larger corpus, then it can be used for downstream tasks with smaller number of training examples. To test on this task, we adopt the simple linear support vector machines (SVM) as learning machine where the features are average of word embeddings, and select different proportions of data to train the SVM classifiers. To test how significant data selection or preprocessing can affect the final results, we also split the users to be head users and tail users. Here for head users we mean users published a lot of reviews, while tail users publish less. We simply sort all the users and select all the users publish half of the total reviews as head users, and the other users are left as tail users. We randomly select one-fifth of the training data for SVM training for the efficiency of our experiments. The statistics of overall, head, and tail users are shown in Table 3. From the table we can see that head users tend to publish more reviews and have more friends. YelpR9 dataset has larger numbers than YelpR8, especially the average numbers of friends.

We show the results trained based on head and tail subsets in Figure 3. From the figure we can see that, the improvement of both personalization and socialization on head data is more significant than tail data. This means, when there are less reviews for a user, and at the same time the relations link to that use are less, our current algorithm cannot train it very well. On the other hand, the absolute values of accuracy for head are less than tail. It means that, if we randomly sample tail data for annotation, it can get better results. However, in practice, we are more likely to sample head users, e.g., opinion leaders or hubs, in a network to annotation corresponding data. Thus, it may be better to carefully treat different groups of user when we have a real problem which needs us to annotate data for social media. In addition, by comparing YelpR8 and YelpR9, we found that the improvement over YelpR9 is greater than YelpR8 on both head and tail users.

We also show the classification results in Table 4 with 100% of one-fifth training data, with hyper-parameter tuning (including the parameter for SVM) over development set, and tested over test set. It shows again that the personalized vectors have better performance than word2vec, and socialized vectors are better than personalization. The improvement is in the middle of head and tail sets.

<table>
<thead>
<tr>
<th></th>
<th>YelpR8 dev</th>
<th>YelpR8 test</th>
<th>YelpR9 dev</th>
<th>YelpR9 test</th>
</tr>
</thead>
<tbody>
<tr>
<td>word2vec</td>
<td>57.02</td>
<td>58.43</td>
<td>58.81</td>
<td>59.83</td>
</tr>
<tr>
<td>personalization</td>
<td>57.25</td>
<td>58.66</td>
<td>59.15</td>
<td>60.11</td>
</tr>
<tr>
<td>socialization</td>
<td>57.43</td>
<td>58.84</td>
<td>59.42</td>
<td>60.30</td>
</tr>
</tbody>
</table>

Table 4: SVM classification accuracy (in %) on test data (trained with one-fifth training data).

4.5 User Vectors for Attention

Finally, we test our user vectors with a deep learning setting. Deep learning has been heavily used in the community since its state-of-the-art performance. For document level sentiment classification on Yelp data, the most recent work [Chen et al., 2016] shows that by using a user attention vector, the results can be improved significantly. In this experiment, we show an interesting setting to use our user vectors as “fixed” attention vectors. Then we show how good they are com-
pared to the baseline without attention, and the approach with user attention trained with supervised learning. The attention mechanism and architecture simply follow [Chen et al., 2016]. This experiment shows how good our unsupervised learning of user vectors are compared with supervised learning. All the training settings and parameters follow the best one shown in [Chen et al., 2016] and their released software.

We use the one-fifth training set in the previous experiment of YelpR8 data as our training data, and test on both development and test sets, which have same scales with the sets used in [Tang et al., 2015b; Chen et al., 2016]. We compared with different deep learning architectures, i.e., convolutional neural networks (CNN) and long short term memory (LSTM) recurrent neural networks (see [Goodfellow et al., 2016] for more details), and their hierarchical versions (HCNN and HLSTM) [Tang et al., 2015b; Chen et al., 2016]. For hierarchical versions, the attention can be attended to both word and sentence levels. The results are shown in Table 5. We can see that our user vectors are very effective compared with the trained user vectors. We can improve CNN with about 1.2 points on test set while trained attention improves 1.6 points. For other deep learning architectures, we can consistently improve the corresponding deep learning algorithms without attention, while are very competitive to the supervised learning results.

Table 5: Comparison of our model and other baseline methods (in accuracy) on user attention based deep learning for sentiment analysis. All the models are trained with the randomly chosen one-fifth of the training sets same as Table 4.

<table>
<thead>
<tr>
<th>Method</th>
<th>CNN</th>
<th>LSTM</th>
<th>HCNN</th>
<th>HLSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev Test</td>
<td>Dev Test</td>
<td>Dev Test</td>
<td>Dev Test</td>
</tr>
<tr>
<td>Without attention</td>
<td>61.14%</td>
<td>62.37%</td>
<td>62.78%</td>
<td>64.23%</td>
</tr>
<tr>
<td>Trained attention</td>
<td>62.61%</td>
<td>63.99%</td>
<td>63.86%</td>
<td>65.30%</td>
</tr>
<tr>
<td>Fixed user vector as attention</td>
<td>62.15%</td>
<td>63.55%</td>
<td>63.22%</td>
<td>64.76%</td>
</tr>
</tbody>
</table>

Figure 3: SVM classification accuracy (in %) on development sets. All of the results are based on ten trials of random sampling from randomly selected one-fifth training set.

5 Conclusion

In this paper, we present a socialized word embedding algorithm to learn a set of global word vectors and a set of local user vectors from social media. A simple but effective social regularization is imposed to our model and we have shown that both the user vectors themselves for personalization and the social regularization can improve the downstream tasks. We use three sets of experiments to demonstrate the effectiveness of our user embeddings. One important future work is how to improve the performance over tail users who publish less text on the social media and have less friends.

Acknowledgements

This paper was supported by China 973 Fundamental R&D Program (No.2014CB340304) and National Natural Science Foundation of China (NSFC Grant Nos. 61472006 and 91646202). Yangqiu Song was also supported the LORELEI Contract HR0011-15-02-0025 with the US Defense Advanced Research Projects Agency (DARPA). Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. We also thank the anonymous reviewers for their valuable comments and suggestions that help improve the quality of this manuscript.

References


