Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Lifelong Domain Word Embedding via Meta-Learning

Hu Xu[1], **Bing Liu**[1], **Lei Shu**[1] and **Philip S. Yu**[1,2]
[1]Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA
[2]Institute for Data Science, Tsinghua University, Beijing, China
{hxu48, liub, lshu3, psyu}@uic.edu

Abstract

Learning high-quality domain word embeddings is important for achieving good performance in many NLP tasks. General-purpose embeddings trained on large-scale corpora are often sub-optimal for domain-specific applications. However, domain-specific tasks often do not have large in-domain corpora for training high-quality domain embeddings. In this paper, we propose a novel *lifelong learning* setting for domain embedding. That is, when performing the new domain embedding, the system has seen many past domains, and it tries to expand the new in-domain corpus by exploiting the corpora from the past domains via meta-learning. The proposed meta-learner characterizes the similarities of the contexts of the same word in many domain corpora, which helps retrieve relevant data from the past domains to expand the new domain corpus. Experimental results show that domain embeddings produced from such a process improve the performance of the downstream tasks.

1 Introduction

Learning word embeddings [Mnih and Hinton, 2007; Mikolov *et al.*, 2013a; Mikolov *et al.*, 2013b; Pennington *et al.*, 2014] has received a great deal of attention due to its success in numerous NLP applications, e.g., named entity recognition [Sienčnik, 2015], sentiment analysis [Maas *et al.*, 2011] and syntactic parsing [Durrett and Klein, 2015]. The key to the success of word embeddings is that a large-scale corpus can be turned into a huge number (e.g., billions) of training examples. Two implicit assumptions are often made about the effectiveness of embeddings to down-stream tasks: 1) the training corpus for embedding is available and much larger than the training data of the down-stream task; 2) the topic (domain) of the embedding corpus is closely aligned with the topic of the down-stream task. However, many real-life applications do not meet both assumptions.

In most cases, the in-domain corpus is of limited size, which is insufficient for training good embeddings. In applications, researchers and practitioners often simply use some general-purpose embeddings trained using a very large general-purpose corpus (which satisfies the first assumption) covering almost all possible topics, e.g., the GloVe embeddings [Pennington *et al.*, 2014] trained using 840 billion tokens covering almost all topics/domains on the Web. Such embeddings have been shown to work reasonably well in many domain-specific tasks. This is not surprising as the meanings of a word are largely shared across domains and tasks. However, this solution violates the second assumption, which often leads to sub-optimal results for domain-specific tasks, as shown in our experiments. One obvious explanation for this is that the general-purpose embeddings do provide some useful information for many words in the domain task, but their embedding representations may not be ideal for the domain and in some cases they may even conflict with the meanings of the words in the task domain because words often have multiple senses or meanings. For example, we have a task in the programming domain, which has the word "Java". A large-scale general-purpose corpus, which is very likely to include texts about coffee shops, supermarkets, the Java island of Indonesia, etc., can easily squeeze the room for representing "Java"' context words like "function", "variable" or "Python" in the programming domain. This results in a poor representation of the word "Java" for the programming task.

To solve this problem and also the limited in-domain corpus size problem, cross-domain embeddings have been investigated [Bollegala *et al.*, 2015; Yang *et al.*, 2017; Bollegala *et al.*, 2017] via transfer learning [Pan and Yang, 2010]. These methods allow some in-domain words to leverage the general-purpose embeddings in the hope that the meanings of these words in the general-purpose embeddings do not deviate much from the in-domain meanings of these words. The embeddings of these words can thus be improved. However, these methods cannot improve the embeddings of many other words with domain-specific meanings (e.g., "Java"). Further, some words in the general-purpose embeddings may carry meanings that are different from those in the task domain.

In this paper, we propose a novel direction for domain embedding learning by expanding the in-domain corpus. The problem in this new direction can be stated as follows:

Problem statement: We assume that the learning system has seen n domain corpora in the past: $D_{1:n} = \{D_1, \ldots, D_n\}$, when a new domain corpus D_{n+1} comes with a certain task, the system automatically generates word embeddings for the $(n+1)$-th domain by leveraging some useful information or knowledge from the past n domains.

This problem definition is in the *lifelong learning* (LL) setting, where the new or $(n+1)$-th task is performed with the help of the knowledge accumulated over the past n tasks [Chen and Liu, 2016]. Clearly, the problem does not have to be defined this way with the domains corpora coming in a sequential manner. It will still work as long as we have n existing domain corpora and we can use them to help with our target domain embedding learning, i.e., the $(n+1)$-th domain.

The main challenges of this problem are 2-fold: 1) how to automatically identify relevant information from the past n domains with no user help, and 2) how to integrate the relevant information into the $(n+1)$-th domain corpus. We propose a meta-learning based system L-DEM (Lifelong Domain Embedding via Meta-learning) to tackle the challenges.

To deal with the first challenge, for a word in the new domain, L-DEM learns to identify similar contexts of the word in the past domains. Here the context of a word means the surrounding words of that word in a domain corpus. We call such context *domain context* (of a word). For this, we introduce a multi-domain meta-learner that can identify similar (or relevant) domain contexts that can be later used in embedding learning in the new domain. To tackle the second challenge, L-DEM augments the new domain corpus with the relevant domain contexts (knowledge) produced by the meta-learner from the past domain corpora and uses the combined data to train the embeddings in the new domain. For example. for word "Java" in the programming domain (the new domain), the meta-learner will produce similar domain contexts from some previous domains like programming language, software engineering, operating systems, etc. These domain contexts will be combined with the new domain corpus for "Java" to train the new domain embeddings.

The main contributions of this paper are as follows. 1) It proposes a novel direction for domain embedding learning, which is a lifelong or continual learning setting and can benefit down-stream learning tasks that require domain-specific embeddings. 2) It proposes a meta-learning approach to leveraging the past corpora from different domains to help generate better domain embeddings. To the best of our knowledge, this is the first meta-learning based approach to helping domain-specific embedding. 3) It experimentally evaluates the effectiveness of the proposed approach.

2 Related Works

Learning word embeddings has been studied for a long time [Mnih and Hinton, 2007]. Many earlier methods used complex neural networks [Mikolov *et al.*, 2013c]. More recently, a simple and effective unsupervised model called skip-gram (or word2vec in general) [Mikolov *et al.*, 2013b; Mikolov *et al.*, 2013c] was proposed to turn a plain text corpus into large-scale training examples without any human annotation. It uses the current word to predict the surrounding words in a context window. The learned weights for each word are the embedding of that word. Although some embeddings trained using large scale corpora are available [Pennington *et al.*, 2014; Bojanowski *et al.*, 2016], they are often sub-optimal for domain-specific tasks [Bollegala *et al.*, 2015; Yang *et al.*, 2017; Xu *et al.*, 2018a; Xu *et al.*, 2018b]. However, a single domain corpus is often too small for training high-quality embeddings [Xu *et al.*, 2018b].

Our problem setting is related to *Lifelong Learning* (LL). Much of the work on LL focused on supervised learning [Thrun, 1996; Silver *et al.*, 2013; Chen and Liu, 2016]. In recent years, several LL works have also been done for unsupervised learning, e.g., topic modeling [Chen and Liu, 2014], information extraction [Mitchell *et al.*, 2018] and graph labeling [Shu *et al.*, 2016]. However, we are not aware of any existing research on using LL for word embedding. Our method is based on meta-learning, which is very different from existing LL methods. Our work is related to transfer learning and multi-task learning [Pan and Yang, 2010]. Transfer learning has been used in cross-domain word embeddings [Bollegala *et al.*, 2015; Yang *et al.*, 2017]. However, LL is different from transfer learning or multi-task learning [Chen and Liu, 2016]. Transfer learning mainly transfers common word embeddings from general-purpose embeddings to a specific domain. We expand the in-domain corpus with similar past domain contexts identified via meta-learning.

To expand the in-domain corpus, a good measure of the similarity of domain contexts of the same word from two different domains is needed. We use meta-learning [Thrun and Pratt, 2012] to learn such similarities. Recently, meta-learning has been applied to various aspects of machine learning, such as learning an optimizer [Andrychowicz *et al.*, 2016], and learning initial weights for few-shot learning [Finn *et al.*, 2017]. The way we use meta-learning is about domain independent learning [Ganin *et al.*, 2016]. It learns similarities of domain contexts of the same word.

3 Model Overview

The proposed L-DEM system is depicted in Figure 1. Given a series of past domain corpora $D_{1:n} = \{D_1, D_2, \ldots, D_n\}$, and a new domain corpus D_{n+1}, the system learns to generate the new domain embeddings by exploiting the relevant information or knowledge from the past n domains. Firstly, a base meta-learner M is trained from the first m past domains (not shown in the figure) (see Section 4), which is later used to predict the similarities of *domain contexts* of the same words from two different domains. Secondly, assuming the system has seen $n-m$ past domain corpora $D_{m+1:n}$, when a new domain D_{n+1} comes, the system produces the embeddings of the $(n+1)$-th domain as follows (discussed in Section 5): (i) the base meta-learner first is adapted to the $(n+1)$-th domain as M_{n+1} (not shown in the figure) using the $(n+1)$-th domain corpus; (ii) for each word w_i in the new domain, the system uses the adapted meta-learner M_{n+1} to identify every past domain j that has the word w_i with domain context similar to w_i's domain context in the new domain (we simply call such domain context from a past domain *similar domain context*); (iii) all new domain words' similar domain contexts from all past domain corpora $D_{m+1:n}$ are aggregated. This combined set is called the *relevant past knowledge* and denoted by $\mathcal{A}$; (iv) a modified word2vec model that can take both domain corpus D_{n+1} and the relevant past knowledge of $\mathcal{A}$ is applied to produce the embeddings for the $(n+1)$-th new domain. Clearly, the meta-learner here plays a central

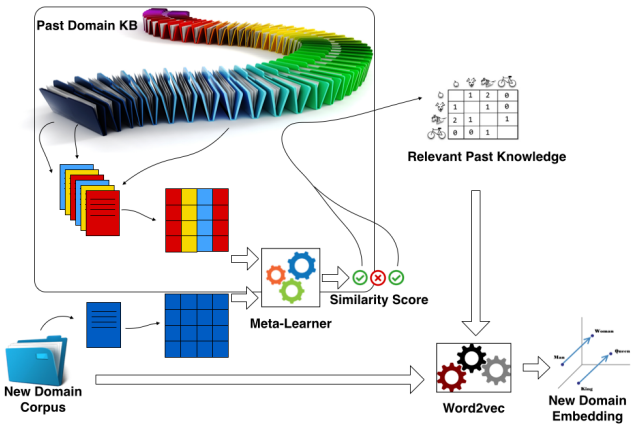

Figure 1: Overview of L-DEM.

role in identifying relevant knowledge from past domains. We propose a pairwise model as the meta-learner.

To enable the above operations, we need a knowledge base (KB), which retains the information or knowledge obtained from the past domains. Once the $(n+1)$-th domain embedding is done, its information is also saved in the KB for future use. We discuss the detailed KB content in Section 5.1.

4 Base Meta-Learner

This section describes the base meta-learner, which identifies similar domain contexts. The input to the meta-learner is a pair of word feature vectors (we simply call them *feature vectors*) representing the domain contexts of the same word from two similar / non-similar domains. The output of the meta-learner is a similarity score of the two feature vectors.

4.1 Training Examples

We assume the number of past domains is large and we hold out the first m domains, where $m \ll n$, as the domains to train and test the base meta-learner. In practice, if n is small, the m domains can be sampled from the n domains. The m domains are split into 3 disjoint sets: training domains, validation domains, and testing domains.

To enable the meta-learner to predict the similarity score, we need both positive examples (from similar domains) and negative examples (from dissimilar domains). Since each past domain can be unique (which makes it impossible to have a positive pair from two similar domains), we sub-sample each domain corpus D_j into 2 sub-corpora: $D_{j,k} \sim P(D_i)$, where $1 \leq j \leq m$ and $k = \{1, 2\}$. This sampling process is done by drawing documents (each domain corpus is a set of documents) uniformly at random from D_j. The number of documents that a domain sub-corpus can have is determined by a pre-defined sub-corpus (file) size (explained in Section 6). We enforce the same file size across all sub-corpora so feature vectors from different sub-corpora are comparable.

Next, we produce feature vectors from domain sub-corpora. Given a word $w_{i,j,k}$ (instance of the word w_i in the domain sub-corpus $D_{j,k}$), we choose its co-occurrence counts on a fixed vocabulary V_{wf} within a context window (similar to word2vec) as the word $w_{i,j,k}$'s feature vector $\mathbf{x}_{w_{i,j,k}}$. The fixed vocabulary V_{wf} (part of the KB used later, denoted as $\mathcal{K}.V_{wf}$) is formed from the top-f frequent words over m domain corpora. This is inspired by the fact that an easy-to-read dictionary (e.g., Longman dictionary) uses only a few thousand words to explain all words of a language. A pair of feature vectors $(\mathbf{x}_{w_{i,j,k}}, \mathbf{x}_{w_{i,j,k'}})$ with $k \neq k'$, forms a postive example; whereas $(\mathbf{x}_{w_{i,j,k}}, \mathbf{x}_{w_{i,j',k}})$ with $j \neq j'$ forms a negative example. Details of settings are in Section 6.

4.2 Pairwise Model of the Meta-learner

We train a small but efficient pairwise model (meta-learner) to learn similarity score. Making the model small but high-throughput is crucial. This is because the meta-learner is required in a high-throughput inference setting, where every word from a new domain needs to have context similarities with the same word from all past domains.

The proposed pairwise model has only four layers. One shared fully-connected layer (with l_1-norm) is used to learn two continuous representations from two (discrete) input feature vectors. A matching function is used to compute the representation of distance in a high-dimentional space. Lastly, a fully-connected layer and a sigmoid layer are used to produce the similarity score. The model is parameterized as follows:

$$\sigma\big(\boldsymbol{W}_2 \cdot \text{abs}\big((\boldsymbol{W}_1 \cdot \frac{\mathbf{x}_{w_{i,j,k}}}{|\mathbf{x}_{w_{i,j,k}}|_1}) - (\boldsymbol{W}_1 \cdot \frac{\mathbf{x}_{w_{i,j',k'}}}{|\mathbf{x}_{w_{i,j',k'}}|_1})\big) + b_2\big), \quad (1)$$

where $|\cdot|_1$ is the l_1-norm, $\text{abs}(\cdot)$ computes the absolute value of element-wise subtraction $(-)$ as the matching function, $\boldsymbol{W}$s and b are weights and $\sigma(\cdot)$ is the sigmoid function. The majority of trainable weights resides in $\boldsymbol{W}_1$, which learns continuous features from the set of f context words. These weights can also be interpreted as a general embedding matrix over V_{wf}. These embeddings (not related to the final domain embeddings in Section 5.2) help to learn the representation of domain-specific words. As mentioned earlier, we train the base meta-learner M over a hold-out set of m domains. We further fine-tune the base meta-learner using the new domain corpus for its domain use, as described in the next section.

5 Embedding Using Past Relevant Knowledge

We now describe how to leverage the base meta-learner M, the rest $n - m$ past domain corpora, and the new domain corpus D_{n+1} to produce the new domain embeddings.

5.1 Identifying Context Words from the Past

When it comes to borrowing relevant knowledge from past domains, the first problem is what to borrow. It is well-known that the embedding vector quality for a given word is determined by the quality and richness of that word's contexts. We call a word in a domain context of a given word a *context word*. So for each word in the new domain corpus, we should borrow all context words from that word's similar domain contexts. The algorithm for borrowing knowledge is described in Algorithm 1, which finds relevant past knowledge $\mathcal{A}$ (see below) based on the knowledge base (KB) $\mathcal{K}$ and the new domain corpus D_{n+1}.

The KB $\mathcal{K}$ has the following pieces of information: (1) the vocabulary of top-f frequent words $\mathcal{K}.V_{wf}$ (as discussed

in Section 4.1), (2) the base meta-learner $\mathcal{K}.M$ (discussed in Section 4.2), and (3) domain knowledge $\mathcal{K}_{m+1:n}$. The domain knowledge has the following information: (i) the vocabularies $V_{m+1:n}$ of past $n-m$ domains, (ii) the sets of past word domain contexts $C_{m+1:n}$ from the past $n-m$ domains, where each C_j is a set of key-value pairs $(w_{i,j}, \mathcal{C}_{w_{i,j}})$ and $\mathcal{C}_{w_{i,j}}$ is a list of context words[1] for word w_i in the j-th domain, and (iii) the sets of feature vectors $E_{m+1:n}$ of past $n-m$ domains, where each set $E_j = \{\mathbf{x}_{w_{i,j,k}} | w_i \in V_j \text{ and } k = \{1,2\}\}$.

The relevant past knowledge $\mathcal{A}$ of the new domain is the aggregation of all key-value pairs $(w_t, \mathcal{C}_{w_t})$, where $\mathcal{C}_{w_t}$ contains all similar domain contexts for w_t.

Algorithm 1 retrieves the past domain knowledge in line 1. Lines 2-4 prepare the new domain knowledge. The BuildFeatureVector function produces a set of feature vectors as $E_{n+1} = \{\mathbf{x}_{w_{i,n+1,k}} | w_i \in V_j \text{ and } k = \{1,2\}\}$ over two sub-corpora of the new domain corpus D_{n+1}. The ScanContextWord function builds a set of key-value pairs, where the key is a word from the new domain $w_{i,n+1}$ and the value $\mathcal{C}_{w_{i,n+1}}$ is a list of context words for the word $w_{i,n+1}$ from the new domain corpus. We use the same size of context window as the word2vec model.

Adapting Meta-learner

In line 5, AdaptMeta-learner adapts or fine-tunes the base meta-learner $\mathcal{K}.M$ to produce an adapted meta-learner M_{n+1} for the new domain. A positive tuning example is sampled from two sub-corpora of the new domain $(\mathbf{x}_{w_{i,n+1,1}}, \mathbf{x}_{w_{i,n+1,2}})$ in the same way as described in Section 4.1. A negative example is exampled as $(\mathbf{x}_{w_{i,n+1,1}}, \mathbf{x}_{w_{i,j,2}})$, where $m+1 \leq j \leq n$. The initial weights of M_{n+1} are set as the trained weights of the base meta-learner M.

Retriving Relevant Past Knowledge

Algorithm 1 further produces the relevant past knowledge $\mathcal{A}$ from line 6 through line 16. Line 6 defines the variable that stores the relevant past knowledge. Lines 7-15 produce the relevant past knowledge $\mathcal{A}$ from past domains. The For block handles each past domain sequentially. Line 8 computes the shared vocabulary O between the new domain and the j-th past domain. After retrieving the sets of feature vectors from the two domains in line 9, the adapted meta-learner uses its inference function (or model) to compute the similarity scores on pairs of feature vectors representing the same word from two domains (line 10). The inference function can parallelize the computing of similarity scores in a high-throughput setting (e.g., GPU inference) to speed up. Then we only keep the words from past domains with a score higher than a threshold δ at line 11. Lines 12-14 aggregate the context words for each word in O from past word domain contexts C_j. Line 16 simply stores the new domain knowledge for future use. Lastly, all relevant past knowledge $\mathcal{A}$ is returned.

Algorithm 1: Identifying Context Words from the Past

Input : a knowledge base $\mathcal{K}$ containing a vocabulary $\mathcal{K}.V_{wf}$, a base meta-learner $\mathcal{K}.M$, and domain knowledge $\mathcal{K}_{m+1:n}$;
a new domain corpus D_{n+1}.

Output: relevant past knowledge $\mathcal{A}$, where each element is a key-value pair $(w_t, \mathcal{C}_{w_t})$ and $\mathcal{C}_{w_t}$ is a list of context words from all similar domain contexts for w_t.

1 $(V_{m+1:n}, C_{m+1:n}, E_{m+1:n}) \leftarrow \mathcal{K}_{m+1:n}$
2 $V_{n+1} \leftarrow$ BuildVocab(D_{n+1})
3 $C_{n+1} \leftarrow$ ScanContextWord(D_{n+1}, V_{n+1})
4 $E_{n+1} \leftarrow$ BuildFeatureVector$(D_{n+1}, \mathcal{K}.V_{wf})$
5 $M_{n+1} \leftarrow$ AdaptMeta-learner$(\mathcal{K}.M, E_{m+1:n}, E_{n+1})$
6 $\mathcal{A} \leftarrow \emptyset$
7 **for** $(V_j, C_j, E_j) \in (V_{m+1:n}, C_{m+1:n}, E_{m+1:n})$ **do**
8 $O \leftarrow V_j \cap V_{n+1}$
9 $F \leftarrow \{(\mathbf{x}_{o,j,1}, \mathbf{x}_{o,n+1,1}) | o \in O \text{ and } \mathbf{x}_{o,j,1} \in E_j \text{ and } \mathbf{x}_{o,n+1,1} \in E_{n+1}\}$
10 $S \leftarrow M_{n+1}$.inference(F)
11 $O \leftarrow \{o | o \in O \text{ and } S[o] \geq \delta\}$
12 **for** $o \in O$ **do**
13 $\mathcal{A}[o]$.append$(C_j[o])$
14 **end**
15 **end**
16 $\mathcal{K}_{n+1} \leftarrow (V_{n+1}, C_{n+1}, E_{n+1})$
17 **return** $\mathcal{A}$

5.2 Augmented Embedding Training

We now produce the new domain embeddings via a modified version of the skip-gram model [Mikolov *et al.*, 2013b] that can take both the new domain corpus D_{n+1} and the relevant past knowledge $\mathcal{A}$. Given a new domain corpus D_{n+1} with the vocabulary V_{n+1}, the goal of the skip-gram model is to learn a vector representation for each word $w_i \in V_{n+1}$ in that domain (we omit the subscript $_{n+1}$ in $w_{i,n+1}$ for simplicity). Assume the domain corpus is represented as a sequence of words $D_{n+1} = (w_1, \ldots, w_T)$, the objective of the skip-gram model maximizes the following log-likelihood:

$$\mathcal{L}_{D_{n+1}} = \sum_{t=1}^{T} \Big(\sum_{w_c \in \mathcal{W}_{w_t}} \big(\log \sigma(\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_c}) + \sum_{w_{c'} \in \mathcal{N}_{w_t}} \log \sigma(-\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_{c'}}) \big) \Big), \tag{2}$$

where $\mathcal{W}_{w_t}$ is the set of words surrounding word w_t in a fixed context window; $\mathcal{N}_t$ is a set of words (negative samples) drawn from the vocabulary V_{n+1} for the t-th word; $\boldsymbol{u}$ and $\boldsymbol{v}$ are word vectors (or embeddings) we are trying to learn. The objective of skip-gram on data of relevant past knowledge $\mathcal{A}$ is as follows:

$$\mathcal{L}_{\mathcal{A}} = \sum_{(w_t, \mathcal{C}_{w_t}) \in \mathcal{A}} \Big(\sum_{w_c \in \mathcal{C}_{w_t}} \big(\log \sigma(\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_c}) + \sum_{w_{c'} \in \mathcal{N}_{w_t}} \log \sigma(-\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_{c'}}) \big) \Big). \tag{3}$$

Finally, we combine the above two objective functions as a single objective function:

[1] We use list to simplify the explanation. In practice, bag-of-word representation should be used to save space.

	CC	KSO	CS
10MB	0.832	0.841	0.856
30MB	0.847	0.859	0.876

Table 1: F1-score of positive predictions of the adapted meta-learner on 3 new domains: Computer Components (CC), Kitchen Storage and Organization (KSO) and Cats Supply (CS).

$$\mathcal{L}'_{D_{n+1}} = \mathcal{L}_{D_{n+1}} + \mathcal{L}_{\mathcal{A}}. \quad (4)$$

We use the default hyperparameters of skip-gram model [Mikolov *et al.*, 2013b] to train the domain embeddings.

6 Experimental Evaluation

Following [Nayak *et al.*, 2016], we use the performances of down-stream tasks to evaluate the proposed method. We do not evaluate the learned embeddings directly as in [Mikolov *et al.*, 2013b; Pennington *et al.*, 2014] because domain-specific dictionaries of similar / non-similar words are generally not available. Our down-stream tasks are text classification that usually requires fine-grained domain embeddings.

6.1 Datasets

We use the Amazon Review datasets from [He and McAuley, 2016], which is a collection of multiple-domain corpora. We consider each second-level category (the first level is department) as a domain and aggregate all reviews under each category as one domain corpus. This ends up with a rather diverse domain collection. We first randomly select 56 (m) domains as the first m past domains to train and evaluate the base meta-learner. Then from rest domains, we sample three random collections with 50, 100 and 200 ($n - m$) domains corpora, respectively, as three settings of past domains. These collections are used to test the performance of different numbers of past domains. Due to the limited computing resource, we limit each past domain corpus up to 60 MB. We further randomly selected 3 rest domains (*Computer Components* (CC), *Kitchen Storage and Organization* (KSO) and *Cats Supply* (CS)) as new domains for down-stream tasks. These give us three text classification problems, which have 13, 17, and 11 classes respectively. The tasks are topic-based classification rather than sentiment classification. Since the past domains have different sizes (many have much less than 60 MB) and many real-world applications do not have big in-domain corpora, we set the size of the new domain corpora to be 10 MB and 30 MB to test the performance in the two settings.

6.2 Evaluation of Meta-Learner

We select the top $f = 5000$ words from all 56 domains' corpora as word features. Then we split the 56 domains into 39 domains for training, 5 domains for validation and 12 domains for testing. So the validation and testing domain corpora have no overlap with the training domain corpora. We sample 2 sub-corpora for each domain and limit the size of each sub-corpus to 10 MB. We randomly select 2000, 500, 1000 words from each training domain, validation domain, and testing domain, respectively, and ignore words with all-zero feature vectors to obtain pairwise examples. The testing 1000 words are randomly drawn and they have 30 overlapping words with the training 2000 words, but not from the same domains. So in most cases, it's testing the unseen words in unseen domains. We set the size of a context window to be 5 when building feature vectors. This ends up with 80484 training examples, 6234 validation examples, and 20740 test examples. For comparison, we train a SVM model as a baseline. The F1-score (for positive pairs) of SVM is 0.70, but the F1-score of the proposed base meta-learner model is **0.81**.

To adapt the base meta-learner for each new domain. We sample 3000 words from each new domain, which results in slightly fewer than 6000 examples after ignoring all-zero feature vectors. We select 3500 examples for training, 500 examples for validation and 2000 examples for testing. The F1-scores on the test data is shown in Table 1. Finally, we empirically set $\delta = 0.7$ as the threshold on the similarity score in Algorithm 1, which roughly doubled the number of training examples from the new domain corpus. The size of the context window for building domain context is set to 5, which is the same as word2vec.

	CC(13)	KSO(17)	CS(11)
NE	0.596	0.653	0.696
fastText	0.705	0.717	0.809
GoogleNews	0.76	0.722	0.814
GloVe.Twitter.27B	0.696	0.707	0.80
GloVe.6B	0.701	0.725	0.823
GloVe.840B	0.803	0.758	0.855
ND 10M	0.77	0.749	0.85
ND 30M	0.794	0.766	0.87
200D + ND 30M	0.795	0.765	0.859
L-DENP 200D + ND 30M	0.806	0.762	0.870
L-DEM 200D + ND 10M	0.791	0.761	0.872
L-DEM 50D + ND 30M	0.795	0.768	0.868
L-DEM 100D + ND 30M	0.803	0.773	0.874
L-DEM 200D + ND 30M	**0.809**	**0.775**	**0.883**

Table 2: Accuracy of different embeddings on classification tasks for 3 new domains (numbers in parenthesis: the number of classes)

6.3 Baselines and Our System

Unless explicitly mentioned, the following embeddings have 300 dimensions, which are the same size as many pre-trained embeddings (GloVec.840B [Pennington *et al.*, 2014] or fastText English Wiki[Bojanowski *et al.*, 2016]).

No Embedding (NE): This baseline does not have any pre-trained word embeddings. The system randomly initializes the word vectors and train the word embedding layer during the training process of the down-stream task.

fastText: This baseline uses the lower-cased embeddings pre-trained from English Wikipedia using fastText [Bojanowski *et al.*, 2016]. We lower the cases of all corpora of down-stream tasks to match the words in this embedding.

GoogleNews: This baseline uses the pre-trained embeddings from word2vec [2] based on part of the Google News dataset, which contains 100 billion words.

[2] https://code.google.com/archive/p/word2vec/

GloVe.Twitter.27B: This embedding set is pre-trained using GloVe[3] based on Tweets of 27 billion words. This embedding is lower-cased and has 200 dimensions.
GloVe.6B: This is the lower-cased embeddings pre-trained from Wikipedia and Gigaword 5, which has 6 billion tokens.
GloVe.840B: This is the cased embeddings pre-trained from Common Crawl corpus, which has 840 billion tokens. This corpus contains almost all web pages available before 2015. We show that the embeddings produced from this very general corpus are sub-optimal for our domain-specific tasks.
New Domain 10M (ND 10M): This is a baseline embedding pre-trained only from the new domain 10 MB corpus. We show that the embeddings trained from a small corpus alone are not good enough.
New Domain 30M (ND 30M): This is a baseline embedding pre-trained only from the new domain 30 MB corpus. We increase the size of the new domain corpus to 30 MB to see the effect of the corpus size.
200 Domains + New Domain 30M (200D + ND 30M): The embedding set trained by combining the corpora from all past 200 domains and the new domain. We use this baseline to show that using all past domain corpora may reduce the performance of the down-stream tasks.
L-DENP 200D + ND 30M: This is a Non-Parametric variant of the proposed method. We use TFIDF as the representation for a sentence in past domains and use cosine as a non-parametic function to compute the similarity with the TFIDF vector built from the new domain corpus. We report the results on a similarity threshold of 0.18, which is the best threshold ranging from 0.15 to 0.20.
L-DEM Past Domains + New Domain (L-DEM [P]D + ND [X]M): These are different variations of our proposed method L-DEM. For example, "L-DEM 200D + ND 30M" denotes the embeddings trained from a 30MB new domain corpus and the relevant past knowledge from 200 past domains.

6.4 Down-stream Tasks and Experiment Results

As indicated earlier, we use classification tasks from 3 new domains ("Computer Components", "Cats Supply" and "Kitchen Storage and Organization") to evaluate the embeddings produced by our system and compare them with those of baselines. These 3 new domains have 13, 17 and 11 classes (or product types), respectively. For each task, we randomly draw 1500 reviews from each class to make up the experiment data, from which we keep 10000 reviews for testing (to make the result more accurate) and split the rest 7:1 for training and validation, respectively. All tasks are evaluated on accuracy. We train and evaluate each task on each system 10 times (with different initializations) and average the results.

For each task, we use an embedding layer to store the pre-trained embeddings. We freeze the embedding layer during training, so the result is less affected by the rest of the model and the training data. To make the performance of all tasks consistent, we apply the same Bi-LSTM model [Hochreiter and Schmidhuber, 1997] on top of the embedding layer to learn task-specific features from different embeddings. The input size of Bi-LSTM is the same as the embedding layer and the output size is 128. All tasks use many-to-one Bi-LSTMs for classification purposes. In the end, a fully-connected layer and a softmax layer are applied after Bi-LSTM, with the output size specific to the number of classes of each task. We apply dropout rate of 0.5 on all layers except the last one and use Adam [Kingma and Ba, 2014] as the optimizer.

[3]https://nlp.stanford.edu/projects/glove/

	CC(13)	KSO(17)	CS(11)
GloVe.840B&ND 30M	0.811	0.78	0.885
GloVe.840B&L-DEM 200D+30M	**0.817**	**0.783**	**0.887**

Table 3: Results of concatenated embeddings with GloVe.840B

Table 2 shows the main results. We observe that the proposed method L-DEM 200D + ND 30M performs the best. The difference in the numbers of past domains indicates more past domains give better results. The GloVe.840B trained on 840 billion tokens does not perform as well as embeddings produced by our method. GloVe.840B's performance on the CC domain is close to our method indicating mixed-domain embeddings for this domain is not bad and this domain is more general. Combining all past domain corpora together with the new domain corpus (200D + ND 30M) makes the result worse than not using the past domains at all (ND 30M). This is because the diverse 200 domains are not similar to the new domains. The L-DENP 200D + ND 30M performs poorly indicating the proposed parametric meta-learner is useful, except the CC domain which is more general.

6.5 Additional Experiments

Note that we did not compare with the existing transfer learning methods [Bollegala *et al.*, 2017; Bollegala *et al.*, 2015; Yang *et al.*, 2017] as our approaches focus on domain-specific words in a lifelong learning setting, which do not need the user to provide the source domain(s) that are known to be similar to the target domain. One approach to leveraging existing embeddings is to concatenate pre-trained embeddings with domain-specific embeddings[4]. To demonstrate our method further improves the domain-specific parts of the down-stream tasks, we evaluate two methods: (1) GloVe.840B&ND 30M, which concatenates new domain only embeddings with GloVe.840B; (2) GloVe.840B&L-DEM 200D + ND 30M, which concatenates our proposed embeddings with GloVe.840B. As shown in Table 3, concatenating embeddings improve the performance. Our method boosts the domain-specific parts of the embeddings further.

7 Conclusions

In this paper, we formulated a domain word embedding learning process. Given many previous domains and a new domain corpus, the proposed method can generate new domain embeddings by leveraging the knowledge in the past domain corpora via a meta-learner. Experimental results show that our method is highly promising.

[4]Note the ideal LL setting is to perform L-DEM over all domain corpora of the pre-trained embeddings.

Acknowledgments

This work is supported in part by NSF through grants IIS-1526499, IIS-1763325, IIS1407927, CNS-1626432 and NSFC 61672313, and a gift from Huawei Technologies.

References

[Andrychowicz *et al.*, 2016] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, pages 3981–3989, 2016.

[Bojanowski *et al.*, 2016] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv*, 2016.

[Bollegala *et al.*, 2015] Danushka Bollegala, Takanori Maehara, and Ken-ichi Kawarabayashi. Unsupervised cross-domain word representation learning. In *ACL*, pages 730–740, 2015.

[Bollegala *et al.*, 2017] Danushka Bollegala, Kohei Hayashi, and Ken-ichi Kawarabayashi. Think globally, embed locally—locally linear meta-embedding of words. *arXiv:1709.06671*, 2017.

[Chen and Liu, 2014] Zhiyuan Chen and Bing Liu. Topic modeling using topics from many domains, lifelong learning and big data. In *ICML*, pages 703–711, 2014.

[Chen and Liu, 2016] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning*. Morgan & Claypool Publishers, 2016.

[Durrett and Klein, 2015] Greg Durrett and Dan Klein. Neural crf parsing. *arXiv*, 2015.

[Finn *et al.*, 2017] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017.

[Ganin *et al.*, 2016] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, pages 2096–2030, 2016.

[He and McAuley, 2016] R. He and J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Maas *et al.*, 2011] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, 2011.

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

[Mikolov *et al.*, 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[Mikolov *et al.*, 2013c] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, pages 746–751, 2013.

[Mitchell *et al.*, 2018] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, B Yang, J Betteridge, A Carlson, B Dalvi, M Gardner, B Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[Mnih and Hinton, 2007] Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *ICML*, pages 641–648, 2007.

[Nayak *et al.*, 2016] Neha Nayak, Gabor Angeli, and Christopher D Manning. Evaluating word embeddings using a representative suite of practical tasks. *ACL 2016*, pages 19–23, 2016.

[Pan and Yang, 2010] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE TKDE*, pages 1345–1359, 2010.

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

[Shu *et al.*, 2016] Lei Shu, Bing Liu, Hu Xu, and Annice Kim. Lifelong-rl: Lifelong relaxation labeling for separating entities and aspects in opinion targets. In *EMNLP*, pages 225–235, 2016.

[Sienčnik, 2015] Scharolta Katharina Sienčnik. Adapting word2vec to named entity recognition. In *NCCL*, pages 239–243, 2015.

[Silver *et al.*, 2013] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong Machine Learning Systems: Beyond Learning Algorithms. In *AAAI Spring Symposium: LML*, page 05, 2013.

[Thrun and Pratt, 2012] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer, 2012.

[Thrun, 1996] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *NIPS*, pages 640–646, 1996.

[Xu *et al.*, 2018a] Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. Double embeddings and cnn-based sequence labeling for aspect extraction. In *ACL*, pages 148–154, 2018.

[Xu *et al.*, 2018b] Hu Xu, Sihong Xie, Lei Shu, and Philip S. Yu. Dual attention network for product compatibility and function satisfiability analysis. In *AAAI*, pages 6013–6020, 2018.

[Yang *et al.*, 2017] Wei Yang, Wei Lu, and Vincent Zheng. A simple regularization-based algorithm for learning cross-domain word embeddings. In *EMNLP*, pages 2898–2904, 2017.