

Lifelong Representation Learning for NLP Applications

by

Hu Xu

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

Defense Committee:

Prof. Philip S. Yu, Chair and Advisor

Prof. Bing Liu, Co-advisor

Prof. Piotr Gmytrasiewicz

Prof. Natalie Parde

Prof. Sihong Xie, Department of Computer Science and Engineering, Lehigh University

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my Ph.D. advisors, Prof. Philip S. Yu and Prof. Bing Liu, for their guidance and support throughout my Ph.D. study and research. It has been my privilege to work with you at different aspects of my Ph.D. journey. Your invaluable suggestions, guidance and your passion for research not only help me with my past academic achievements but also will influence my professional career in the future. Besides my advisors, I would like to thank Prof. Piotr Gmytrasiewicz and Prof. Natalie Parde, for your valuable time on my dissertation. I am grateful to Prof. Sihong Xie from Lehigh University, for the mentorship and support of my early years of a research career and enlightening me on the first glance of research. Last but not least, none of this could have happened without my family. I am grateful for my parents, for their unconditional love, support and encouragement of my Ph.D. study.

HX

TABLE OF CONTENTS

CHAPTER

PAGE

LIST OF TABLES

<u>TABLE</u>	<u>PAGE</u>
--------------	-------------

LIST OF FIGURES

FIGURE

PAGE

LIST OF ABBREVIATIONS

LL	Lifelong Learning
CNN	Convolutional Neural Network
BERT	Bidirectional Encoder Representations from Trans- formers
AE	Aspect Extraction
ASC	Aspect Sentiment Classification
CER	Complementary Entity Recognition
RRC	Review Reading Comprehension
RCRC	Review Conversational Reading Comprehension
R-GCN	Relational Graph Convolutional Network
UMGR	User Memory Graph Reasoner

SUMMARY

Representation learning lives at the heart of deep learning for natural language processing (NLP). Traditional representation learning (such as softmax-based classification, pre-trained word embeddings, and language models, graph representations) focuses on learning general or static representations with the hope to help any end task. As the world keeps evolving, emerging knowledge (such as new tasks, domains, entities or relations) typically come with a small amount of data with shifted distributions that challenge the existing representations to be effective. As a result, how to effectively learn representations for new knowledge becomes crucial. Lifelong learning is a machine learning paradigm that aims to build an AI agent that keeps learning from the evolving world, like humans' learning from the world. This dissertation focuses on improving representations on different types of new knowledge (classification, word-level, contextual-level, and knowledge graph) for a myriad of NLP end tasks, ranging from text classification, sentiment analysis, question answering to the more complex dialogue system. With the help of lifelong representation learning, tasks are greatly improved beyond general representation learning.

CHAPTER 1

INTRODUCTION

Deep learning (DL) has gained significant improvements over the past a few years [?]. The core driving force behind deep learning is its capability or capacity to learn knowledgable features or representations automatically from large-scale data. This significantly reduces the need of asking humans to curate better features manually. As a result, the learned representation gives the model a great advantage to concur with the uncertainty during testing from the unknown world, which can be found in many applications of computer vision and natural language processing. The key advantage of deep learning over traditional machine learning models is that the parameter-intensive DL models can consume much more data than traditional ML models to obtain more general representation by inferring the features on-the-fly as a form of reasoning. These learned features, in the end, boost the performance of many tasks.

Following this advantage, how to smartly consume more data to learn general features and avoid specific features is essential for DL models. Researchers start to pre-train DL models with the hope to encode all features of the world into parameters of DL models. Examples can be found in the large-scale pre-training on ImageNet dataset [?, ?, ?] in computer vision, or pre-trained word embeddings or language models [?, ?] in natural language processing (NLP).

1.1 Motivation

Going beyond the classic deep learning approach, simply aggregating existing data into a DL model may not be enough. Looking forward, the world keeps evolving and yields new data for new tasks, which probably are long-tailed or heavily-tailed. This greatly challenges the existing learned representations. The existing approach may represent the majority general features well and assume they are generally good to any new knowledge. However, it lacks enough capability to represent the vast kinds of specific features that are required each (new) task. To make the learning effective in the long-term, an AI agent must be able to adapt to the changes in the world. In contrast, we humans are very sensitive to the changes in the world and the wide spectrum of novel details by having a focus and learning new knowledge and updating our understanding of the world. We never use our 6-year-old understanding of the world to solve the problems now.

1.2 Research Objectives

Motivated by this observation, an AI agent needs to learn representations as to the way humans do in a new machine learning paradigm or a problem called **lifelong learning**, which aims to build AI agents from a sequence of tasks online.

lifelong learning (LL) assumes the learning tasks come in a sequence $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n$, where the new or $(n + 1)$ -th task is performed with the help of the knowledge accumulated over the past n tasks [?, ?, ?].

Note that this definition does not specify or constrain the forms or types of each task. To name a few, a task can be any learning task, ranging from learning for a new class, learning

for a new domain, learning in a heterogeneous form of a new task, to new concept or relation for accumulating the knowledge. As such, we can see that the problem or concept of lifelong learning can be applied to a vast amount of concrete machine learning tasks.

As a result, this dissertation focuses on a wide range of machine learning tasks and their usage in NLP applications. We aim to cover major types of machine learning tasks in NLP and provide its applications to concrete datasets with experimental results and discussions on the role of lifelong learning for their improvements in performance.

1.3 Outlines

To make a more clear distinction between different forms of learning tasks and NLP applications, this dissertation is organized by separating lifelong learning tasks and NLP applications and avoid following the structure of each original publication.

I first address open-world learning problems on classification tasks in Chapter ??, where traditional classifiers can easily make mistakes on unseen classes that appear during testing or inference. This is because most existing classifiers must classify an example from an unseen class to one of existing pre-defined classes during training. I further extend this problem to a dynamic classification task, where some unseen classes can be added to or removed from the set of existing classes while still keeping rejecting the rest unseen classes. We use a meta-learning approach to address this problem into a very general comparison-based classifier. As a result, it avoids learning a classifier overfitting to a particular set of classes.

In Chapter ??, I switch to classic representation learning problems in NLP. I first focus on learning word embedding and propose a problem of learning domain word embeddings. In

this problem, each word has its domain representation. However, emerging domains typically do not have enough corpus to train fully-fledged embeddings. By applying lifelong learning into word embeddings, I allow corpus-level sharing of knowledge amongst existing domains. As such, I first describe how to obtain domain-specific word embeddings from a small domain corpus in a lifelong learning fashion and show the performance domain-specific embeddings compare to general-purpose embeddings. Second, I explore the usage of domain-specific word embeddings and focus on how to leverage both general-purpose embeddings and domain-specific embeddings together.

In Chapter ??, I switch to contextualized word representation, where each word is strongly tied to its context in a document. This yields a better representation of the meaning of a word in a sentence or paragraph. Given the expensive training of contextualized word representation, I switch to a different style of lifelong learning and focuses on how to obtain domain contextualized word representation via a sequence of different types of learning tasks. I discuss two types of learning tasks: post-training and pre-tuning. On one hand, post-training is a learning task intended to address the shifts of distributions such as domains. This ended with a huge gap between an end task and a general-purpose pre-trained contextualized word representation. Pre-tuning, on the other hand, aims to solve the discrepancy between a pre-trained contextualized word representation and end tasks. Given existing pre-trained models aim to cover a wide range of end-tasks, the learned representation is not optimal for each end task. The proposed pre-tuning task mimics the formulation of an end task with only unlabeled data, which shortens the gap between a pre-trained model and an end task.

Further, in Chapter ??, I move towards graph representation learning. A graph is a natural way for sharable and interpretable knowledge for humans. It can be used for both feature augmentations and reasoning. However, the existing approach of graph representation learning mostly assumes a static graph, where the knowledge and reasoning upon knowledge are never changed. Lifelong learning is ideal for graph reasoning as it can keep updating graphs and reasoning policy. Thus, we rename the term graph as memory graph, indicating the graph is dynamic that can maintain and update reasoning based on newly added knowledge.

Lastly, in Chapter ??, I target the usage of lifelong learning over a wide spectrum of NLP tasks. I first describe the task in text classification, including product classification (using methods in Chapter ??) and review classification (using methods in Chapter ??). Then I focus on tasks in aspect-based sentiment analysis (ABSA). I first describe the usage of domain word embeddings (from Chapter ??) and contextualized word representation (from Chapter ??) for aspect extraction. Later I discuss the contextualized word representation and the lifelong training algorithm of hard examples for aspect sentiment classification. Next, I go through the question-answering problem and focus on machine reading comprehension (MRC) and its novel application to reviews. Lastly, I discuss the usage of lifelong learning for conversational AI. I first describe the usage of pre-tuning for conversational review reading comprehension (CRC). Then focus on lifelong graph reasoning for conversational recommendation with dynamic graph reasoning (using the method in Chapter ??).

CHAPTER 2

LIFELONG CLASSIFICATION

The content of this chapter was previously published as “Open-world Learning and Application to Product Classification” in WWW 2019 [?], DOI: <https://doi.org/10.1145/3308558.3313644>, with an arXiv version ¹ [?].

Classification is a well-known and classic problem and the deep learning variant of the classification task typically leverages an activation function that can compute a categorical distribution over a set of classes (e.g., the softmax function). This type of classification is called *closed-world classification* because the classes seen in testing must have appeared in training. However, this assumption is often violated in real-world applications. For example, new topics often emerge on a social media site and new categories of products appear daily on e-commerce website. A model that cannot detect new/unseen topics or products is hard to function well in such open environments. This is where lifelong learning can be applied to the existing classification problem.

2.1 Motivation

Generally speaking, an AI agent working in the real world must be able to recognize the classes of things that it has seen/learned before and detect new things that it has not seen and learn to accommodate the new things. This learning paradigm is called *open-world learning*

¹<https://arxiv.org/abs/1809.06004>

(OWL) [?, ?, ?]. This is in contrast with the classic supervised learning paradigm which makes the *closed-world assumption* that the classes seen in testing must have appeared in training. With the ever-changing Web, the popularity of AI agents such as intelligent assistants and self-driving cars that need to face the real-world open environment with unknowns, OWL capability is crucial.

For example, with the growing number of products sold on Amazon from various sellers, it is necessary to have an open-world model that can automatically classify a product based on a set S of product categories. An emerging product not belonging to any existing category in S should be classified as “unseen” rather than one from S . Further, this unseen set may keep growing. When the number of products belonging to a new category is large enough, it should be added to S . An open-world model should easily accommodate this addition with a low cost of training since it is impractical to retrain the model from scratch every time a new class is added. As another example, the very first interface for many intelligent personal assistants (IPA) (such as Amazon Alexa, Google Assistant, and Microsoft Cortana) is to classify user utterances into existing known domain/intent classes (e.g., Alexa’s skills) and also reject/detect utterances from unknown domain/intent classes (that are currently not supported). But, with the support to allow the 3rd-party to develop new skills (Apps), such IPAs must recognize new/unseen domain or intent classes and include them in the classification model. These real-life examples present a major challenge to the maintenance of the deployed model.

2.2 Open-world Learning

Most existing solutions to OWL are built on top of closed-world models [?, ?, ?], e.g., by setting thresholds on the logits (before the softmax/sigmoid functions) to reject unseen classes which tend to mix with existing seen classes. One major weakness of these models is that they cannot easily add new/unseen classes to the existing model without re-training or incremental training (e.g., OSDN [?] and DOC [?]). There are incremental learning techniques (e.g., iCaRL [?] and DEN [?]) that can incrementally learn to classify new classes. However, they miss the capability of rejecting examples from unseen classes. This section proposes to solve OWL with both capabilities in a very different way via meta-learning.

Problem Statement: At any point in time, the learning system is aware of a set of seen classes $S = \{c_1, \dots, c_m\}$ and has an OWL model/classifier for S but is unaware of a set of unseen classes $U = \{c_{m+1}, \dots\}$ (any class not in S can be in U) that the model may encounter. The goal of an OWL model is two-fold: (1) classifying examples from classes in S and reject examples from classes in U , and (2) when a new class c_{m+1} (without loss of generality) is removed from U (now $U = \{c_{m+2}, \dots\}$) and added to S (now $S = \{c_1, \dots, c_m, c_{m+1}\}$), still being able to perform (1) without re-training the model.

Related Work

Open-world learning has been studied in text mining and computer vision (where it is called open-set recognition) [?, ?, ?]. Most existing approaches focus on building a classifier that can predict examples from unseen classes into a (hidden) *rejection class*. These solutions are built on top of closed-world classification models [?, ?, ?]. Since a closed-world classifier cannot

detect/reject examples from unseen classes (they will be classified into some seen classes), some thresholds are used so that these closed-world models can also be used to do rejection. However, as discussed earlier, when incrementally learning new classes, they also need some form of re-training, either full re-training from scratch [?, ?] or partial re-training in an incremental manner [?, ?].

Our work is also related to class incremental learning [?, ?, ?], where new classes can be added dynamically to the classifier. For example, iCaRL [?] maintains some exemplary data for each class and incrementally tunes the classifier to support more new classes. However, they also require training when each new class is added. Our work is clearly related to meta-learning (or learning to learn) [?], which turns the machine learning tasks themselves as training data to train a meta-model and has been successfully applied to many machine learning tasks lately, such as [?, ?, ?, ?, ?]. Our proposed framework focuses on learning the similarity between an example and an arbitrary class and we are not aware of any open-world learning work based on meta-learning.

The proposed framework is also related to zero-shot learning [?, ?, ?] (in that we do not require training but need to read training examples), k -nearest neighbors (k NN) (with additional rejection capability, metric learning [?] and learning to vote), and Siamese networks [?, ?, ?] (regarding processing a pair of examples). However, all those techniques work in closed-worlds with no rejection capability. Product classification has been studied in [?, ?, ?, ?, ?, ?], mostly in a multi-level (or hierarchical) setting. However, given the dynamic taxonomy in nature, product classification has not been studied as an open-world learning problem.

2.3 L2AC Framework

Two main challenges for solving open-world learning: (1) how to enable the model to classify examples of seen classes into their respective classes and also detect/reject examples of unseen classes, and (2) how to incrementally include the new/unseen classes when they have enough data without re-training the model. As discussed above, existing methods either focus on the challenge (1) or (2), but not both.

To tackle both challenges in a unified approach, I propose an entirely new OWL method based on meta-learning [?, ?, ?, ?, ?]. The method is called *Learning to Accept Classes* (L2AC). The key novelty of L2AC is that the model maintains a dynamic set S of seen classes that allow new classes to be added or deleted with no model re-training needed. Each class is represented by a small set of training examples. In testing, the meta-classifier only uses the examples of the maintained seen classes (including the newly added classes) on-the-fly for classification and rejection. That is, the learned meta-classifier classifies or rejects a test example by comparing it with its nearest examples from each seen class in S . Based on the comparison results, it determines whether the test example belongs to a seen class or not. If the test example is not classified as any seen class in S , it is rejected as unseen. Unlike existing OWL models, the parameters of the meta-classifier are not trained on the set of seen classes but on a large number of other classes which can share a large number of features with seen and unseen classes, and thus can work with any seen classification and unseen class rejection without re-training.

We can see that the proposed method works like the nearest neighbor classifier (e.g., k NN). However, the key difference is that we train a meta-classifier to perform both classification and

rejection based on a learned metric and a learned voting mechanism. Also, k NN cannot do rejection on unseen classes.

As an overview, ?? depicts how L2AC classifies a test example into an existing seen class or rejects it as from an unseen class. The training process for the meta-classifier is not shown, which is detailed in Chapter ?. The L2AC framework has two major components: a ranker and a meta-classifier. The ranker is used to retrieve some examples from a seen class that are similar/near to the test example. The meta-classifier performs classification after it reads the retrieved examples from the seen classes. The two components work together as follows.

Assume we have a set of seen classes S . Given a test example x_t that may come from either a seen class or an unseen class, the ranker finds a list of top- k nearest examples to x_t from each seen class $c \in S$, denoted as $x_{a_{1:k}|c}$. The meta-classifier produces the probability $p(c = 1|x_t, x_{a_{1:k}|c})$ that the test x_t belongs to the seen class c based on c 's top- k examples (most similar to x_t). If none of these probabilities from the seen classes in S exceeds a threshold (e.g., 0.5 for the sigmoid function), L2AC decides that x_t is from an unseen class (rejection); otherwise, it predicts x_t as from the seen class with the highest probability (for classification). We denote $p(c = 1|x_t, x_{a_{1:k}|c})$ as $p(c|x_t, x_{a_{1:k}|c})$ for brevity when necessary. Note that although we use a threshold, this is a general threshold that is not for any specific classes as in other OWL approaches but only for the meta-classifier. More practically, this threshold is pre-determined (not empirically tuned via experiments on hyper-parameter search) and the meta-classifier is trained based on this fixed threshold.

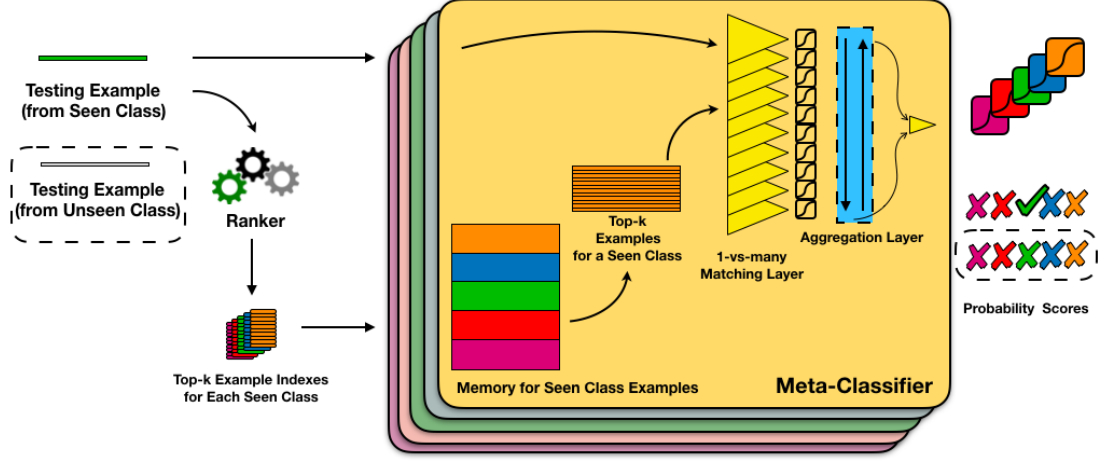


Figure 1. Overview of the L2AC framework

As we can see, the proposed framework works like a supervised lazy learning model, such as the k -nearest neighbor (k NN) classifier. Such a lazy learning mechanism allows the dynamic maintenance of a set of seen classes, where an unseen class can be easily added to the seen class set S . However, the key differences are that all the metric space, voting and rejection are learned by the meta-classifier.

Retrieving the top- k nearest examples $x_{a_{1:k}}$ for a given test example x_t needs a ranking model (the ranker). We will detail a sample implementation of the ranker in Chapter ?? and discuss the details of the meta-classifier in the next section.

Meta-Classifier

Meta-classifier serves as the core component of the L2AC framework. It is essentially a binary classifier on a given seen class. It takes the top- k nearest examples (to the test example

x_t) of the seen class as the input and determines whether x_t belongs to that seen class or not. In this part, we first describe how to represent examples of a seen class. Then we describe how the meta-classifier processes these examples together with the test example into an overall probability score (via a voting mechanism) for deciding whether the test example should belong to any seen class (classification) or not (rejection). Along with that we also describe how a joint decision is made for open-world classification over a set of seen classes. Finally, we describe how to train the meta-classifier via another set of meta-training classes and their examples.

Example Representation and Memory

Representation learning lives at the heart of neural networks. Following the success of using pre-trained weights from large-scale image datasets (such as ImageNet [?]) as feature encoders, we assume there is an encoder that captures almost all features for text classification.

Given an example x representing a text document (a sequence of tokens), we obtain its continuous representation (a vector) via an encoder $h = g(x)$, where the encoder $g(\cdot)$ is typically a neural network (e.g., CNN or LSTM). We will detail a simple encoder implementation in Chapter ??.

Further, we save the continuous representations of the examples into the memory of the meta-classifier. So later, the top- k examples can be efficiently retrieved via the index (address) in the memory. The memory is essentially a matrix $E \in \mathbb{R}^{n \times |h|}$, where n is the number of all examples from seen classes and $|h|$ is the size of the hidden dimensions. Note that we will still use x instead of h to refer to an example for brevity. Given the test example x_t , the meta-classifier first looks up the actual continuous representations $x_{a_{1:k}}$ of the top- k examples for a seen class.

Then the meta-classifier computes the similarity score between x_t and each x_{a_i} ($1 \leq i \leq k$) individually via a 1-vs-many matching layer as described next.

1-vs-many Matching Layer

To compute the overall probability between a test example and a seen class, a 1-vs-many matching layer in the meta-classifier first computes the individual similarity score between the test example and each of the top- k retrieved examples of the seen class. The 1-vs-many matching layer essentially consists of k shared matching networks as indicated by big yellow triangles in ?? . We denote each matching network as $f(\cdot, \cdot)$ and compute similarity scores $r_{1:k}$ for all top- k examples $r_{1:k} = f(x_t, x_{a_{1:k}})$.

The matching network first transforms the test example x_t and x_{a_i} from the continuous representation space to a single example in similarity space. We leverage two similarity functions to obtain the similarity space. The first function is the absolute values of the element-wise subtraction: $f_{\text{abssub}}(x_t, x_{a_i}) = |x_t - x_{a_i}|$. The second one is the element-wise summation: $f_{\text{sum}}(x_t, x_{a_i}) = x_t + x_{a_i}$. Then the final similarity space is the concatenation of these two functions' results: $f_{\text{sim}}(x_t, x_{a_i}) = f_{\text{abssub}}(x_t, x_{a_i}) \oplus f_{\text{sum}}(x_t, x_{a_i})$, where \oplus denotes the concatenation operation. We then pass the result to two fully-connected layers (one with Relu activation) and a sigmoid function:

$$r_i = f(x_t, x_{a_i}) = \sigma\left(W_2 \cdot \text{Relu}(W_1 \cdot f_{\text{sim}}(x_t, x_{a_i}) + b_1) + b_2\right). \quad (2.1)$$

Since there are k nearest examples, we have k similarity scores denoted as $r_{1:k}$. The hyper-parameters are detailed in Chapter ??.

Open-world Learning via Aggregation Layer

After getting the individual similarity scores, an aggregation layer in the meta-classifier merges the k similarity scores into a single probability indicating whether the test example x_t belongs to the seen class. By having the aggregation layer, the meta-classifier essentially has a *parametric voting mechanism* so that it can learn how to vote on multiple nearest examples (rather than a single example) from a seen class to decide the probability. As a result, the meta-classifier can have more reliable predictions, which is studied in Chapter ??.

We adopt a (many-to-one) BiLSTM [?, ?] as the aggregation layer. We set the output size of BiLSTM to 2 (1 per the direction of LSTM). Then the output of BiLSTM is connected to a fully-connected layer followed by a sigmoid function that outputs the probability. The computation of the meta-classifier for a given test example x_t and $x_{a_{1:k}}$ for a seen class c can be summarized as:

$$p(c|x_t, x_{a_{1:k}}) = \sigma(W \cdot \text{BiLSTM}(r_{1:k}) + b). \quad (2.2)$$

Inspired by DOC [?], for each class $c \in S$, we evaluate ?? as:

$$\hat{y} = \begin{cases} \text{reject, if } \max_{c \in S} p(c|x_t, x_{a_{1:k}}) \leq 0.5; \\ \arg \max_{c \in S} p(c|x_t, x_{a_{1:k}}), \text{ otherwise.} \end{cases} \quad (2.3)$$

If none of existing seen classes S gives a probability above 0.5, we *reject* x_t as an example from some unseen class. Note that given a large number of classes, ?? can be efficiently implemented in parallel. We leave this to future work. To make L2AC an easily accessible approach, we use 0.5 as the threshold naturally and do not introduce an extra hyper-parameter that needs to be artificially tuned. Note also that as discussed earlier, the seen class set S and its examples can be dynamically maintained (e.g., one can add to or remove from S any class). So the meta-classifier simply performs open-world classification over the current seen class set S .

Training of Meta-Classifier

Since the meta-classifier is a general classifier that is supposed to work for any class, training the meta-classifier $p_\theta(c|x_t, x_{a_{1:k}|x_t,c})$ requires examples from another set M of classes called *meta-training classes*.

A large $|M|$ is desirable so that meta-training classes have good coverage of features for seen and unseen classes in testing, which is in a similar spirit to few-shot learning [?]. We also enforce $(S \cup U) \cap M = \emptyset$ in Chapter ??, so that all seen and unseen classes are unknown to the meta-classifier.

Next, we formulate the meta-training examples from M , which consist of a set of pairs (with positive and negative labels). The first component of a pair is a training document x_q from a class in M , and the second component is a sequence of top- k nearest examples also from a class in M .

We assume every example (document) of a class in M can be a training document x_q . Assuming x_q is from class $c \in M$, a positive training pair is $(x_q, x_{a_{1:k}|x_q,c})$, where $x_{a_{1:k}|x_q,c}$ are

top- k examples from class c that are most similar or nearest to x_q ; a negative training pair is $(x_q, x_{a_{1:k}|x_q, c'})$, where $c' \in M, c \neq c'$ and $x_{a_{1:k}|x_q, c'}$ are top- k examples from class c' that are nearest to x_q . We call c' one *negative class* for x_q . Since there are many negative classes $c' \in M \setminus c$ for x_q , we keep top- n negative classes for each training example x_q . That is, each x_q has one positive training pair and n negative training pairs. To balance the classes in the training loss, we give a weight ratio $n : 1$ for a positive and a negative pair, respectively.

Training the meta-classifier also requires validation classes for model selection (during optimization) and hyper-parameters (k and n) tuning (as detailed in Experiments). Since the classes tested by the meta-classifier are unexpected, we further use a set of *validation classes* $M' \cap M = \emptyset$ (also $M' \cap (S \cup U) = \emptyset$), to ensure generalization on the seen/unseen classes.

2.4 Results

We want to address the following Research Questions (RQs) in experiments: **RQ1** - what is the performance of the meta-classifier with different settings of top- k examples and n negative classes? **RQ2** - How is the performance of L2AC compared with state-of-the-art text classifiers for open-world classification (which all need some forms of re-training).

Dataset

We leverage the huge amount of product descriptions from the Amazon Datasets [?] and form the OWL task as the following. Amazon.com maintains a tree-structured category system. We consider each path to a leaf node as a class. We removed products belonging to multiple classes to ensure the classes have no overlapping. This gives us 2598 classes, where 1018 classes have more than 400 products per class. We randomly choose 1000 classes from the 1018 classes

with 400 randomly selected products per class as the *encoder training set*; 100 classes with 150 products per class are used as the (classification) *test set*, including both seen classes S and unseen classes U ; another 1000 classes with 100 products per class are used as the *meta-training set* (including both M and M'). For the 100 classes of the test set, we further hold out 50 examples (products) from each class as test examples. The rest 100 examples are training data for baselines, or seen classes examples to be read by the meta-classifier (which only reads those examples but is not trained on those examples). To train the meta-classifier, we further split the meta-training set as 900 *meta-training classes* (M) and 100 *validation classes* (M'). For all datasets, we use NLTK(<https://www.nltk.org/>) as the tokenizer, and regard all words that appear more than once as the vocabulary. This gives us 17,526 unique words. We take the maximum length of each document as 120 since the majority of product descriptions are under 100 words.

Ranker

We use cosine similarity to rank the examples in each seen (or meta-training) class for a given test (or meta-training) example x_t (or x_q) (Given many examples to process, the ranker can be implemented in a fully parallel fashion to speed up the processing, which we leave to future work as it is beyond the scope of this work.). We apply cosine directly on the hidden representations of the encoder as $\text{cosine}(h_*, h_{a_i}) = \frac{h_* \cdot h_{a_i}}{\|h_*\|_2 \|h_{a_i}\|_2}$, where $*$ can be either t or q , $\|\cdot\|_2$ denotes the l_2 norm and \cdot denotes the dot product of two examples.

Training the meta-classifier also requires a ranking of negative classes for a meta-training example x_q , as discussed in Chapter ?? . We first compute a *class vector* for each meta-training class. This class vector is averaged over all encoded representations of examples of that class.

Then we rank classes by computing cosine similarity between the class vectors and the meta-training example x_q . The top- n (defined in the previous section) classes are selected as negative classes for x_q . We explore different settings of n later.

Evaluation

Similar to [?], we choose 25, 50, and 75 classes from the (classification) test set of 100 classes as the seen classes for three (3) experiments. Note that each class in the test set has 150 examples, where 100 examples are for the training of baseline methods or used as seen class examples for L2AC and 50 examples are for testing both the baselines and L2AC. We evaluate the results on all 100 classes for those three (3) experiments. For example, when there are 25 seen classes, testing examples from the rest 75 unseen classes are taken as from one *rejection class* c_{rej} , as in [?].

Besides using macro F1 as used in [?], we also use weighted F1 score overall classes (including seen and the rejection class) as the evaluation metric. Weighted F1 is computed as

$$\sum_{c \in S \cup \{c_{\text{rej}}\}} \frac{N_c}{\sum_{c \in S \cup \{c_{\text{rej}}\}} N_c} \cdot \text{F1}_c, \quad (2.4)$$

where N_c is the number of examples for class c and F1_c is the F1 score of that class. We use this metric because macro F1 has a bias on the importance of rejection when the seen class set is small (macro F1 treats the rejection class as equally important as one seen class). For example, when the number of seen classes is small, the rejection class should have a higher weight as a classifier on a small seen set is more likely challenged by examples from unseen classes. Further,

to stabilize the results, we train all models with 10 different initializations and average the results.

Hyper-parameters

For simplicity, we leverage a BiLSTM [?, ?] on top of a GloVe [?] embedding (840b.300d) layer as the encoder (other choices are also possible). Similar to feature encoders trained from ImageNet [?], we train classification over the encoder training set with 1000 classes and use 5% of the encoding training data as encoder validation data. We apply dropout rates of 0.5 to all layers of the encoder. The classification accuracy of the encoder on validation data is **81.76%**. The matching network (the shared network within the 1-vs-many matching layer) has two fully-connected layers, where the size of the hidden dimension is 512 with a dropout rate of 0.5. We set the batch size of meta-training as 256.

To answer RQ1 on two hyper-parameters k (number of nearest examples from each class) and n (number of negative classes), we use the 100 validation classes to determine these two hyper-parameters. We formulate the validation data similar to the testing experiment on 50 seen classes. For validation, we select 50 examples for each class. The rest 50 examples from each validation seen class are used to find top- k nearest examples. We perform grid search of averaged weighted F1 over 10 runs for $k \in \{1, 3, 5, 10, 15, 20\}$ and $n \in \{1, 3, 5, 9\}$, where $k = 5$ and $n = 9$ reach a reasonably well weighted F1 (87.60%). Further increasing n gives limited improvements (e.g., 87.69% for $n = 14$ and 87.68% for $n = 19$, when $k = 5$). But a large n significantly increases the number of training examples (e.g., $n = 14$ ended with more than 1 million meta-training examples) and thus training time. So we decide to select $k = 5$ and $n = 9$

Methods	$ S = 25$ (WF1)	$ S = 25$ (MF1)	$ S = 50$ (WF1)	$ S = 50$ (MF1)	$ S = 75$ (WF1)	$ S = 75$ (MF1)
DOC-CNN	53.25(1.0)	55.04(0.39)	70.57(0.46)	76.91(0.27)	81.16(0.47)	86.96(0.2)
DOC-LSTM	57.87(1.26)	57.6(1.18)	69.49(1.58)	75.68(0.78)	77.74(0.48)	84.48(0.33)
DOC-Enc	82.92(0.37)	75.09(0.33)	82.53(0.25)	84.34(0.23)	83.84(0.36)	88.33(0.19)
DOC-CNN-Gaus	85.72(0.43)	76.79(0.41)	83.33(0.31)	83.75(0.26)	84.21(0.12)	87.86(0.21)
DOC-LSTM-Gaus	80.31(1.73)	70.49(1.55)	77.49(0.74)	79.45(0.59)	80.65(0.51)	85.46(0.25)
DOC-Enc-Gaus	88.54(0.22)	80.77(0.22)	84.75(0.21)	85.26(0.2)	83.85(0.37)	87.92(0.22)
L2AC- $n9$ -NoVote	91.1(0.17)	82.51(0.39)	84.91(0.16)	83.71(0.29)	81.41(0.54)	85.03(0.62)
L2AC- $n9$ -Vote3	91.54(0.55)	82.42(1.29)	84.57(0.61)	82.7(0.95)	80.18(1.03)	83.52(1.14)
L2AC- $k5$ - $n9$ -AbsSub	92.37(0.28)	84.8(0.54)	85.61(0.36)	84.54(0.42)	83.18(0.38)	86.38(0.36)
L2AC- $k5$ - $n9$ -Sum	83.95(0.52)	70.85(0.91)	76.09(0.36)	75.25(0.42)	74.12(0.51)	78.75(0.57)
L2AC- $k5$ - $n9$	<u>93.07</u> (0.33)	86.48(0.54)	<u>86.5</u> (0.46)	85.99(0.33)	<u>84.68</u> (0.27)	88.05(0.18)
L2AC- $k5$ - $n14$	93.19 (0.19)	86.91(0.33)	86.63 (0.28)	86.42(0.2)	85.32(0.35)	88.72(0.23)
L2AC- $k5$ - $n19$	93.15(0.24)	86.9(0.45)	86.62(0.49)	86.48(0.43)	85.36 (0.66)	88.79(0.52)

TABLE I. Scores for OWL

for all ablation studies below. Note the validation classes are also used to compute (formulated in a way similar to the meta-training classes) the validation loss for selecting the best model during Adam [?] optimization.

Compared Methods

To the best of our knowledge, DOC [?] is the only state-of-the-art baseline for open-world learning (with rejection) for text classification. It has been shown in [?] that DOC significantly outperforms the methods CL-cbsSVM and cbsSVM in [?] and OpenMax in [?]. OpenMax is a state-of-the-art method for image classification with rejection capability. To answer RQ2, we use DOC and its variants to show that the proposed method has comparable performance with

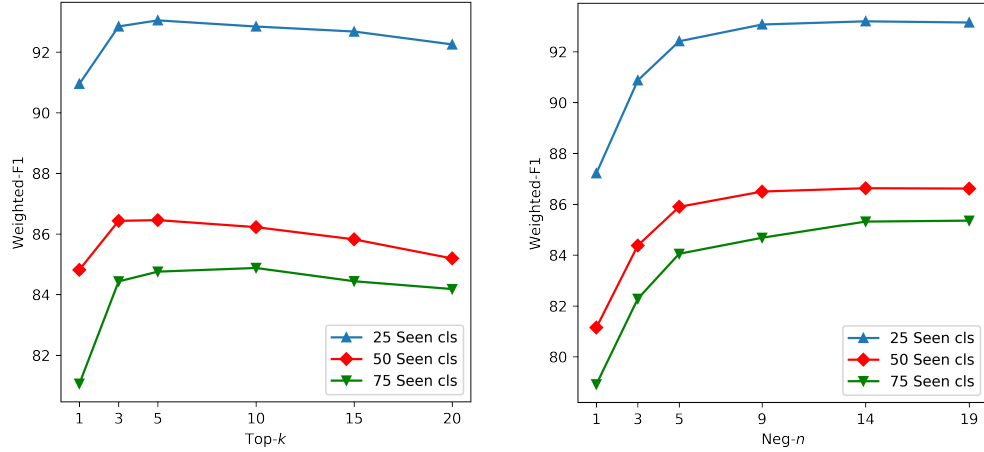


Figure 2. Weighted F1 scores of k and n for OWL

the best open-world learning method with re-training. Note that DOC cannot incrementally add new classes. So we re-train DOC over different sets of seen classes from scratch every time new classes are added to that set.

It is thus actually unfair to compare our method with DOC because DOC is trained on the actual training examples of all classes. However, our method still performs better in general. We used the original code of DOC and created six (6) variants of it.

DOC-CNN: CNN implementation as in the original DOC paper without Gaussian fitting (using 0.5 as the threshold for rejection). It operates directly on a sequence of tokens.

DOC-LSTM: a variant of DOC-CNN, where we replace CNN with BiLSTM to encode the input sequence for a fair comparison. BiLSTM is trainable and the input is still a sequence of tokens.

DOC-Enc: this is adapted from DOC-CNN, where we remove the feature learning part of

DOC-CNN and feed the hidden representation from our encoder directly to the fully-connected layers of DOC for a fair comparison with L2AC.

DOC-*-Gaus: applying Gaussian fitting proposed in [?] on the above three baselines, we have 3 more DOC baselines. Note that these 3 baselines have the same models as above respectively. They only differ in the thresholds used for rejection. Gaussian fitting in [?] is used to set a good threshold for rejection. We use these baselines to show that the Gaussian fitted threshold improves the rejection performance of DOC significantly but may lower the performance of seen class classification. The original DOC is **DOC-CNN-Gaus** here.

The following baselines are variants of L2AC.

L2AC- $n9$ -NoVote: this is a variant of the proposed L2AC that only takes one most similar example (from each class), i.e., $k = 1$, with one positive class paired with $n = 9$ negative classes in meta-training ($n = 9$ has the best performance as indicated in answering RQ1 above). We use this baseline to show that the performance of taking only one sample may not be good enough. This baseline does not have/need the aggregation layer and only has a single matching network in the 1-vs-many layer.

L2AC- $n9$ -Vote3: this baseline uses the same model as L2AC- $n9$ -NoVote. But during the evaluation, we allow a non-parametric voting process (like k NN) for prediction. We report the results of voting over top-3 examples per seen class as it has the best result (ranging from 3 to 10). If the average of the top-3 similar examples in a seen class has example scores with more than 0.5, L2AC believes the testing example belongs to that class. We use this baseline to show that the aggregation layer is effective in learning to vote and L2AC can use more similar examples and

get better performance.

L2AC- $k5-n9$ -AbsSub/Sum: To show that using two similarity functions ($f_{\text{abssub}}(\cdot, \cdot)$ and $f_{\text{sum}}(\cdot, \cdot)$) gives better results, we further perform ablation study by using only one of those similarity functions at a time, which gives us two baselines.

L2AC- $k5-n9/14/19$: this baseline has the best $k = 5$ and $n = 9$ on the validation classes, as indicated in the previous subsection. Interestingly, further increasing k may reduce the performance as L2AC may focus on not-so-similar examples. We also report results on $n = 14$ or 19 to show that the results do not get much better.

Results Analysis

From Table ??, we can see that L2AC outperforms DOC, especially when the number of seen classes is small. First, from ?? we can see that $k = 5$ and $n = 9$ gets reasonably good results. Increasing k may harm the performance as taking in more examples from a class may let L2AC focus on not-so-similar examples, which is bad for classification. More negative classes give L2AC better performance in general but further increasing n beyond 9 has little impact.

Next, we can see that as we incrementally add more classes, L2AC gradually drops its performance (which is reasonable due to more classes) but it still yields better performance than DOC. Considering that L2AC needs no training with additional classes, while DOC needs full training from scratch, L2AC represents a major advance. Note that testing on 25 seen classes is more about testing a model’s rejection capability while testing on 75 seen classes is more about the classification performance of seen class examples. From ??, we notice that L2AC

can effectively leverage multiple nearest examples and negative classes. In contrast, the non-parametric voting of L2AC- $n9$ -Vote3 over top-3 examples may not improve the performance but introduce higher variances. Our best $k = 5$ indicates that the meta-classifier can dynamically leverage multiple nearest examples instead of solely relying on a single example. As an ablation study on the choices of similarity functions, running L2AC on a single similarity function gives poorer results as indicated by either L2AC- $k5$ - $n9$ -AbsSub or L2AC- $k5$ - $n9$ -Sum.

DOC without encoder (DOC-CNN or DOC-LSTM) performs poorly when the number of seen classes is small. Without Gaussian fitting, DOC's (DOC-CNN, DOC-LSTM or DOC-Enc) performance increases as more classes are added as seen classes. This is reasonable as DOC is more challenged by fewer seen training classes and more unseen classes during testing. As such, Gaussian fitting (DOC- $*$ -Gaus) alleviates the weakness of DOC on a small number of seen training classes.

CHAPTER 3

LIFELONG WORD REPRESENTATION LEARNING

The content of this chapter was previously published as “Lifelong Domain Word Embedding via Meta-Learning” in IJCAI 2019 [?] with an arXiv version ¹ [?], and partially as “Double Embeddings and CNN-based Sequence Labeling for Aspect Extraction” in ACL 2018 [?] (DOI: <http://dx.doi.org/10.18653/v1/P18-2094>, with arXiv version ¹ [?]).

Learning word embeddings [?,?,?,?] has received a great deal of attention due to its success in numerous NLP applications, e.g., named entity recognition [?], sentiment analysis [?] and syntactic parsing [?]. 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.

There are 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 downstream 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-

¹<https://arxiv.org/abs/1805.09991>

¹<https://arxiv.org/abs/1805.04601>

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 [?] 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.

3.1 Motivation

Thus, 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.

As such, 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.

To solve this problem and also the limited in-domain corpus size problem, cross-domain embeddings have been investigated [?, ?, ?] via transfer learning [?]. 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.

This chapter is organized in two sections: we first address the problem of learning word embeddings for a new domain in a lifelong learning setting; then we discuss how to leverage both general and domain-specific embeddings.

3.2 Lifelong Domain Word Embeddings

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: $D_{1:n} = \{D_1, \dots, D_n\}$, 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 [?]. The problem does not have to be defined this way with the domains corpora coming sequentially. 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 Embdding 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, the word “Java” in

the programming domain (the new domain), the meta-learner will produce similar domain contexts from some previous domains like a 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.

Related Works

Learning word embeddings has been studied for a long time [?]. Many earlier methods used complex neural networks [?]. More recently, a simple and effective unsupervised model called skip-gram (or word2vec in general) [?, ?] 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 [?, ?], they are often sub-optimal for domain-specific tasks [?, ?, ?, ?]. However, a single domain corpus is often too small for training high-quality embeddings [?].

Our problem setting is related to *Lifelong Learning* (LL). Much of the work on LL focused on supervised learning [?, ?, ?]. In recent years, several LL works have also been done for unsupervised learning, e.g., topic modeling [?], information extraction [?] and graph labeling [?]. 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 [?]. Transfer learning has been used in cross-domain word embeddings [?, ?]. However, LL is different from transfer learning or multi-task learning [?]. 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 [?] to learn such similarities. Recently, meta-learning has been applied to various aspects of machine learning, such as learning an optimizer [?], and learning initial weights for few-shot learning [?]. The way we use meta-learning is about domain-independent learning [?]. It learns similarities of domain contexts of the same word.

3.3 L-DEM Approach

The proposed L-DEM system is depicted in ??. Given a series of past domain corpora $D_{1:n} = \{D_1, D_2, \dots, D_n\}$, 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: (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. The meta-learner here plays a central 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.

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.

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

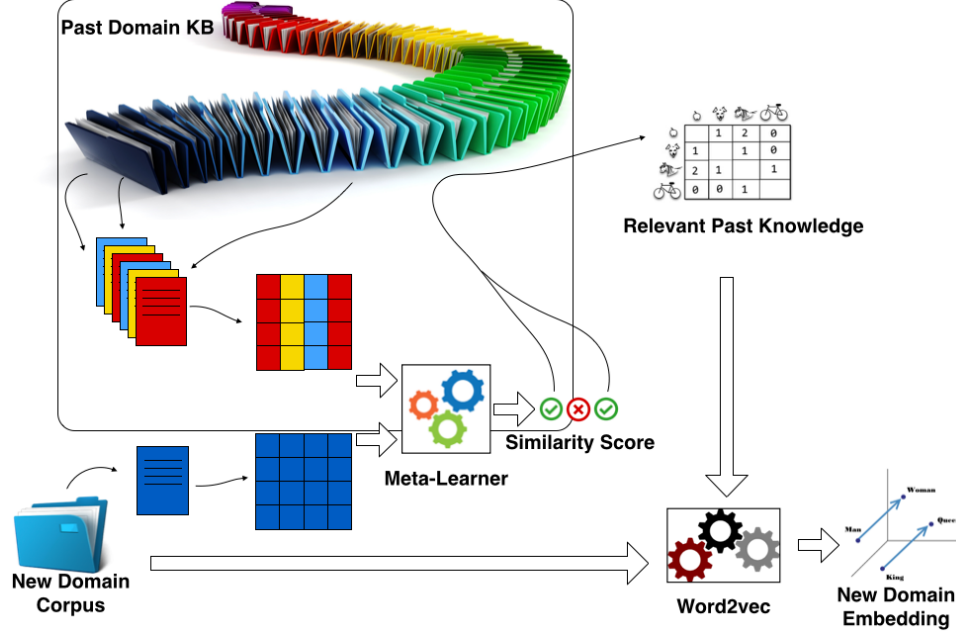


Figure 3. Overview of L-DEM

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. 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 positive 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.

Pairwise Model of the Meta-learner

We train a small but efficient pairwise model (meta-learner) to learn a 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-dimensional space. Lastly, a fully-connected layer and a sigmoid layer are used to produce the similarity score. The model is parameterized as follows:

$$\sigma\left(\mathbf{W}_2 \cdot \text{abs}\left(\left(\mathbf{W}_1 \cdot \frac{\mathbf{x}_{w_{i,j,k}}}{\|\mathbf{x}_{w_{i,j,k}}\|_1}\right) - \left(\mathbf{W}_1 \cdot \frac{\mathbf{x}_{w_{i,j',k'}}}{\|\mathbf{x}_{w_{i,j',k'}}\|_1}\right)\right) + b_2\right), \quad (3.1)$$

where $\|\cdot\|_1$ is the l_1 -norm, $\text{abs}(\cdot)$ computes the absolute value of element-wise subtraction $(-)$ as the matching function, \mathbf{W} s and b are weights and $\sigma(\cdot)$ is the sigmoid function. The majority of trainable weights resides in \mathbf{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 Chapter ??) 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.

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.

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 ??, 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 (We use list to simplify the explanation. In practice, bag-of-word representation should be used to save space.) 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 ?? 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 the 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 $(\mathbf{x}_{w_{i,n+1,1}}, \mathbf{x}_{w_{i,n+1,2}})$. A negative example is sampled 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 .

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

```

Retrieving Relevant Past Knowledge

Algorithm ?? 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.

Augmented Embedding Training

We now produce the new domain embeddings via a modified version of the skip-gram model [?] 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 $D_{n+1} = (w_1, \dots, w_T)$, the objective of the skip-gram model maximizes the following log-likelihood:

$$\mathcal{L}_{D_{n+1}} = \sum_{t=1}^T \left(\sum_{w_c \in \mathcal{W}_{w_t}} (\log \sigma(\mathbf{u}_{w_t}^T \cdot \mathbf{v}_{w_c})) + \sum_{w_{c'} \in \mathcal{N}_{w_t}} \log \sigma(-\mathbf{u}_{w_t}^T \cdot \mathbf{v}_{w_{c'}}) \right), \quad (3.2)$$

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; \mathbf{u} and \mathbf{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:

$$\mathcal{L}_{\mathcal{A}} = \sum_{(w_t, \mathcal{C}_{w_t}) \in \mathcal{A}} \left(\sum_{w_c \in \mathcal{C}_{w_t}} (\log \sigma(\mathbf{u}_{w_t}^T \cdot \mathbf{v}_{w_c})) + \sum_{w_{c'} \in \mathcal{N}_{w_t}} \log \sigma(-\mathbf{u}_{w_t}^T \cdot \mathbf{v}_{w_{c'}}) \right). \quad (3.3)$$

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

$$\mathcal{L}'_{D_{n+1}} = \mathcal{L}_{D_{n+1}} + \mathcal{L}_{\mathcal{A}}. \quad (3.4)$$

We use the default hyperparameters of skip-gram model [?] to train the domain embeddings.

3.4 Results

Following [?], we use the performances of down-stream tasks to evaluate the proposed method. We do not evaluate the learned embeddings directly as in [?, ?] 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.

Datasets

We use the Amazon Review datasets from [?], 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.

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

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

TABLE II

F1-score for L-DEM Meta-Learner

building feature vectors. This ends up with 80484 training examples, 6234 validation examples, and 20740 test examples. For comparison, we train an 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 are shown in Table 1. Finally, we empirically set $\delta = 0.7$ as the threshold on the similarity score in Algorithm ??, 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.

Baselines and Our System

	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 III

Accuracy of L-DEM

Unless explicitly mentioned, the following embeddings have 300 dimensions, which are the same size as many pre-trained embeddings (GloVec.840B [?] or fastText English Wiki [?]).

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 downstream task.

fastText: This baseline uses the lower-cased embeddings pre-trained from English Wikipedia using fastText [?]. 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 (<https://code.google.com/archive/p/word2vec/>) based on part of the Google News dataset, which contains 100 billion words.

GloVe.Twitter.27B: This embedding set is pre-trained using GloVe (<https://nlp.stanford.edu/projects/glove/>) 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 the 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-parametric 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.

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 [?] 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 a dropout rate of 0.5 on all layers except the last one and use Adam [?] as the optimizer.

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 are not bad and this domain is more general. Combining all past domain corpora 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.

3.5 Fusion of General and Domain Word Embeddings

The performance gain of domain word embeddings comes from the dense corpus focusing on a particular domain and the feature space dedicated to that particular domain. Although domain word embeddings are good at domain-specific features, many NLP tasks also require good features for general words that are unlikely to be affected by a particular domain too, such as those stop words. As a result, those words are unlikely to be trained well due to the limited corpus of a particular domain, whereas general word embeddings have such an advantage by aggregating corpora from multiple domains together. To the end, for a particular end task, how to leverage the benefits from both types of embeddings is essential for the success of an end task.

3.5.1 – Approach

One simple way is to concatenate the general word embeddings and domain-specific word embeddings. Assume the input is a sequence of word indexes $\mathbf{x} = (x_1, \dots, x_n)$. This sequence gets its two corresponding continuous representations \mathbf{x}^g and \mathbf{x}^d via two separate embedding layers (or embedding matrices) W^g and W^d . The first embedding matrix W^g represents general embeddings pre-trained from a very large general-purpose corpus (usually hundreds of billions of tokens). The second embedding matrix W^d represents domain embeddings pre-trained

from a small in-domain corpus, where the scope of the domain is exactly the domain that the training/testing data belongs to.

We do not allow these two embedding layers trainable because small training examples may lead to many unseen words in test data. If embeddings are tunable, the features for seen words' embeddings will be adjusted (e.g., forgetting useless features and infusing new features that are related to the labels of the training examples). But the embeddings of unseen words from test data still have the old features that may be mistakenly extracted by future layers. As such, we concatenate two embeddings $\mathbf{x}^{(1)} = \mathbf{x}^g \oplus \mathbf{x}^d$ and feed the hidden states to the rest layers of the network for the end task.

3.5.2 – Result

We conducted experiments on two settings, one is in the same setting as for lifelong domain embeddings [?]; the other is for a sequence labeling task in sentiment analysis. We detail the architecture for aspect extraction later in Chapter ??.

L-DEM for Text Classification

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. The results of concatenating general and domain-specific embeddings are shown in Chapter ?. Our method boosts the domain-specific parts of the embeddings further. Note the ideal LL setting is to perform L-DEM on all domain corpora of the pre-trained embeddings.).

	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 IV

Concatenating Word Embeddings

CHAPTER 4

LIFELONG CONTEXTUALIZED REPRESENTATION LEARNING

The part of this chapter was previously published as “BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis” in NAACL 2019 [?], DOI: <http://dx.doi.org/10.18653/v1/N19-1242>, with an arXiv version ¹ [?], together with an arXiv paper “Review Conversational Reading Comprehension” ¹ [?].

Beyond word embeddings that only carry independent word-level features, the meaning (thus features or representations) of a word is also heavily affected by its contexts. As a result, a good representation for an end task may not be only from a word embedding layer, but also from an encoder $E(x)$ that can consume a piece of text and provide representations for each word based on its nearby context in that sequence. To learn such an encoder $E(x)$, researchers need to define a general proxy task that is close to almost all end tasks so to learn features for those end tasks. The proxy task also needs to be self-supervised as the training corpora are unlabeled and can be as large as the corpus for word embeddings.

Language Model is a natural choice for such a proxy task, which aims to generate the rest texts given the input is corrupted from a piece of text. Recent years of representation learning for NLP has a large focus on language models from large-scale unlabeled corpora, such as Elmo [?],

¹<https://arxiv.org/1904.02232>

¹<https://arxiv.org/abs/1902.00821>

GPT/GPT2 [?,?], BERT [?], XLNet [?], RoBERTa [?], ALBERT [?], ELECTRA [?]. The idea behind the progress is that even though the word embedding [?,?] layer (in a typical neural network for NLP) is trained from large-scale corpora, training a wide variety of neural architectures that encode contextual representations only from the limited supervised data on end tasks is insufficient.

BERT [?] is one of the key innovations in the recent progress. The magic behind BERT is the proposed proxy task of masked language model (MLM), which does not aim to generate the next token from the previous token but randomly masking out a portion of tokens from a whole text and task the model to predict. The key benefit of MLM is that it enables a more complex reasoning process of learning and reasoning from the corrupted (masked) input that not only learns from a unidirectional context (e.g., left side of the current token) but from bidirectional contexts. This naturally ended with more deeper reasoning and general features from contexts rather than hard-coded features from a particular piece of text.

4.1 Motivation

Although BERT aims to learn contextualized representations across a wide range of NLP tasks (to be task-agnostic), leveraging BERT alone still leaves the domain challenges unresolved (as BERT is trained on Wikipedia articles and has almost no understanding of the text on a particular domain). As such, BERT only learns features for text in general but largely ignores knowledge for a particular domain. Also, since BERT aims to learn features for almost all end tasks, it introduces another challenge of task-awareness, called the *task challenge*. This challenge arises when the task-agnostic BERT meets the limited number of fine-tuning examples in end

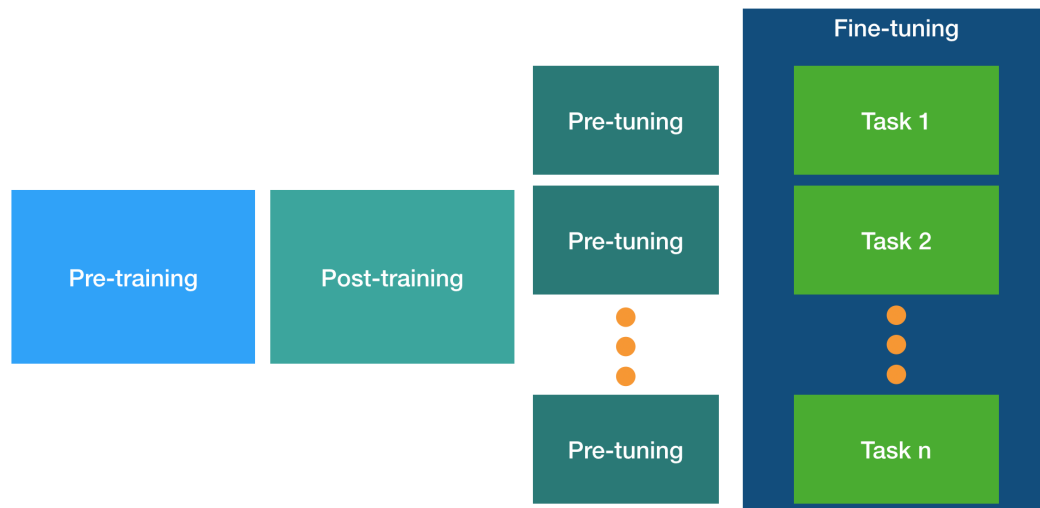


Figure 4. Lifelong stages of training

tasks, which is insufficient to fine-tune BERT to ensure full task-awareness of the system. For example, the end tasks from the original BERT paper typically use tens of thousands of examples to ensure that the system is task-aware. Inspired by these observations, I introduce a lifelong learning style of training.

4.2 Lifelong Training

To address the challenges, I propose a lifelong learning style training by introducing extra training tasks within the well-known pre-training and fine-tuning framework. I explore two (2) training task (or step) into the existing framework: post-training and pre-tuning, as depicted in ??.

Post-training aims to adapt pre-training LM from general text to domain-specific text, whereas pre-tuning aims to adapt pre-trained LM to a particular task.

4.2.1 – Post-training of Language Models

I propose a novel joint post-training technique that takes BERT’s pre-trained weights as the initialization (Due to limited computation resources, it is impractical for us to pre-train BERT directly on reviews from scratch [?].) for basic language understanding and adapts BERT with domain knowledge. I also incorporate tasks from a supervised learning corpus from a machine reading comprehension task (MRC) that carries high-quality QA knowledge annotated by humans. Results show that this task further improves the learned representation. As a result, post-training leverages knowledge from two sources: unsupervised domain reviews and supervised (yet out-of-domain) MRC data.

BERT has two parameter intensive settings:

BERT_{BASE}: 12 layers, 768 hidden dimensions and 12 attention heads (in transformer) with the total number of parameters, 110M;

BERT_{LARGE}: 24 layers, 1024 hidden dimensions and 16 attention heads (in transformer) with the total number of parameters, 340M.

To post-train on domain knowledge, we leverage the two novel pre-training objectives from BERT: masked language model (MLM) and next sentence (The BERT paper refers a sentence as a piece of text with one or more natural language sentences.) prediction (NSP). The former predicts randomly masked words and the latter detects whether two sides of the input are from the same document or not. A training example is formulated as $([CLS], x_{1:j}, [SEP], x_{j+1:n}, [SEP])$,

where $x_{1:n}$ is a document (with randomly masked words) split into two sides $x_{1:j}$ and $x_{j+1:n}$ and [SEP] separates those two.

MLM is crucial for injecting review domain knowledge and for alleviating the bias of the knowledge from Wikipedia. For example, in the Wikipedia domain, BERT may learn to guess the [MASK] in “The [MASK] is bright” as “sun”. But in a laptop domain, it could be “screen”. Further, if the [MASK]ed word is an opinion word in “The touch screen is [MASK]”, this objective challenges BERT to learn the representations for fine-grained opinion words like “great” or “terrible” for [MASK]. The objective of NSP further encourages BERT to learn contextual representation beyond word-level. In the context of reviews, NSP formulates a task of “artificial review prediction”, where a negative example is an original review but a positive example is a synthesized fake review by combining two different reviews. This task exploits the rich relationships between two sides in the input, such as whether two sides of texts have the same rating or not (when two reviews with different ratings are combined as a positive example), or whether two sides are targeting the same product or not (when two reviews from different products are merged as a positive example). In summary, these two objectives encourage to learn a myriad of fine-grained features for potential end tasks.

We let the loss function of MLM be \mathcal{L}_{MLM} and the loss function of next text piece prediction be \mathcal{L}_{NSP} , the total loss of the domain knowledge post-training is $\mathcal{L}_{\text{DK}} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}}$.

To post-train BERT on general QA knowledge, we use SQuAD (1.1), which is a popular large-scale MRC dataset.

We let the loss on SQuAD be \mathcal{L}_{MRC} , which is in a similar setting as the loss \mathcal{L}_{RRC} for RRC. As a result, the joint loss of post-training is defined as $\mathcal{L} = \mathcal{L}_{\text{DK}} + \mathcal{L}_{\text{MRC}}$.

One major issue of post-training on such a loss is the prohibitive cost of GPU memory usage. Instead of updating parameters over a batch, we divide a batch into multiple sub-batches and accumulate gradients on those sub-batches before parameter updates. This allows for a smaller sub-batch to be consumed in each iteration.

Algorithm 2: Post-training Algorithm

Input: \mathcal{D}_{DK} : one batch of DK data;

\mathcal{D}_{MRC} one batch of MRC data;

u : number of sub-batches.

```

1  $\nabla_{\Theta} \mathcal{L} \leftarrow 0$ 
2  $\{\mathcal{D}_{\text{DK},1}, \dots, \mathcal{D}_{\text{DK},u}\} \leftarrow \text{Split}(\mathcal{D}_{\text{DK}}, u)$ 
3  $\{\mathcal{D}_{\text{MRC},1}, \dots, \mathcal{D}_{\text{MRC},u}\} \leftarrow \text{Split}(\mathcal{D}_{\text{MRC}}, u)$ 
4 for  $i \in \{1, \dots, u\}$  do
5    $\mathcal{L}_{\text{partial}} \leftarrow \frac{\mathcal{L}_{\text{DK}}(\mathcal{D}_{\text{DK},i}) + \mathcal{L}_{\text{MRC}}(\mathcal{D}_{\text{MRC},i})}{u}$ 
6    $\nabla_{\Theta} \mathcal{L} \leftarrow \nabla_{\Theta} \mathcal{L} + \text{BackProp}(\mathcal{L}_{\text{partial}})$ 
7 end
8  $\Theta \leftarrow \text{ParameterUpdates}(\nabla_{\Theta} \mathcal{L})$ 

```

Algorithm 1 describes one training step and takes one batch of data on domain knowledge (DK) \mathcal{D}_{DK} and one batch of MRC training data \mathcal{D}_{MRC} to update the parameters Θ of BERT. In line 1, it first initializes the gradients ∇_{Θ} of all parameters as 0 to prepare gradient computation. Then in lines 2 and 3, each batch of training data is split into u sub-batches. Lines 4-7 spread the calculation of gradients to u iterations, where the data from each iteration of sub-batches are supposed to be able to fit into GPU memory. In line 5, it computes the partial joint loss $\mathcal{L}_{\text{partial}}$ of two sub-batches $\mathcal{D}_{\text{DK},i}$ and $\mathcal{D}_{\text{MRC},i}$ from the i -th iteration through forward pass. Note that

the summation of two sub-batches' losses is divided by u , which compensates the scale change introduced by gradient accumulation in line 6. Line 6 accumulates the gradients produced by backpropagation from the partial joint loss. To this end, accumulating the gradients u times is equivalent to computing the gradients on the whole batch once. But the sub-batches and their intermediate hidden representations during the i -th forward pass can be discarded to save memory space. Only the gradients ∇_{Θ} are kept throughout all iterations and used to update parameters (based on the chosen optimizer) in line 8.

Setting for End Tasks

We aim for 3 review-based tasks to evaluate the post-trained BERT: review reading comprehension (RRC), aspect extraction (AE) and aspect sentiment classification (ASC). The details of these 3 tasks are introduced in ???. Here we only discuss the formulation of these 3 tasks under the setting of BERT. The overall illustrations of the formulation are depicted in ???.

Following the success of SQuAD [?] and BERT's SQuAD implementation, we design review reading comprehension as follows. Given a question $q = (q_1, \dots, q_m)$ asking for an answer from a review $d = (d_1, \dots, d_n)$, we formulate the input as a sequence $x = ([CLS], q_1, \dots, q_m, [SEP], d_1, \dots, d_n, [SEP])$, where $[CLS]$ is a dummy token not used for RRC and $[SEP]$ is intended to separate q and d . Let $\text{BERT}(\cdot)$ be the pre-trained (or post-trained as in the next section) BERT model. We first obtain the hidden representation as $h = \text{BERT}(x) \in \mathbb{R}^{r_h * |x|}$, where $|x|$ is the length of the input sequence and r_h is the size of the hidden dimension. Then the hidden representation is passed to two separate dense layers followed by softmax functions: $l_1 = \text{softmax}(W_1 \cdot h + b_1)$ and $l_2 = \text{softmax}(W_2 \cdot h + b_2)$, where $W_1, W_2 \in \mathbb{R}^{r_h}$ and $b_1, b_2 \in \mathbb{R}$. The softmax is applied

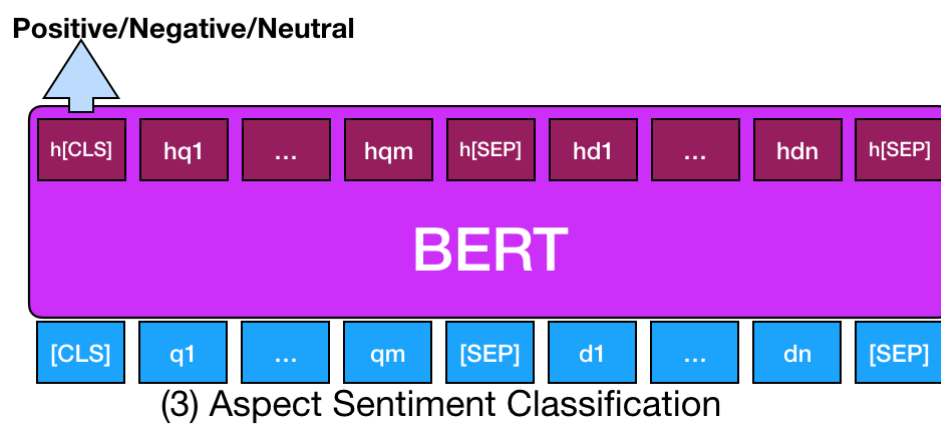
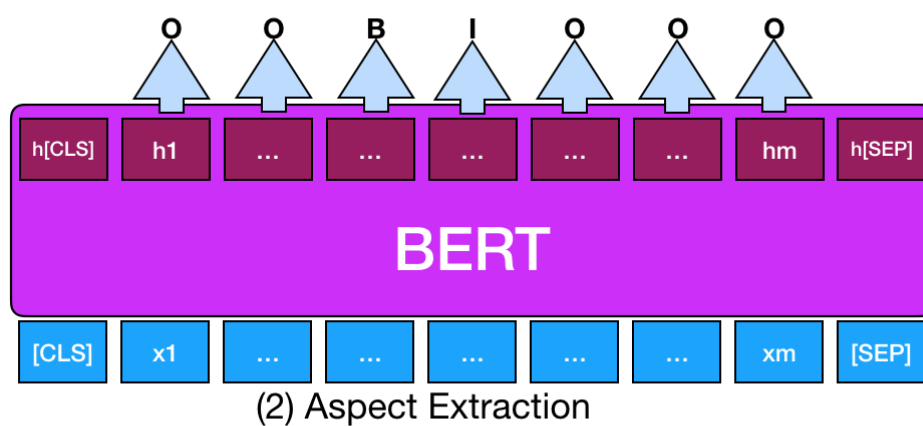
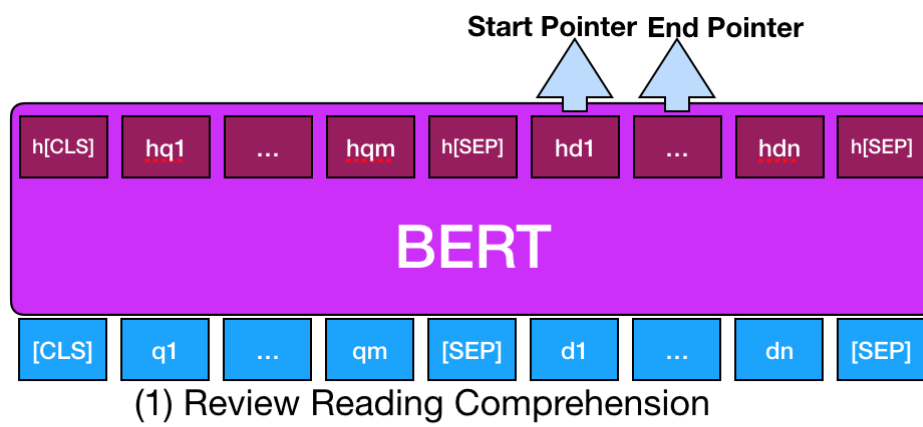


Figure 5. BERT for end tasks

along the dimension of the sequence. The output is a span across the positions in d (after the [SEP] token of the input), indicated by two pointers (indexes) s and e computed from l_1 and l_2 : $s = \arg \max_{\text{Idx}_{[\text{SEP}]} < s < |x|} (l_1)$ and $e = \arg \max_{s \leq e < |x|} (l_2)$, where $\text{Idx}_{[\text{SEP}]}$ is the position of token [SEP] (so the pointers will never point to tokens from the question). As such, the final answer will always be a valid text span from the review as $a = (d_s, \dots, d_e)$.

Training the RRC model involves minimizing the loss that is designed as the averaged cross entropy on the two pointers:

$$\mathcal{L}_{\text{RRC}} = -\frac{\sum \log l_1 \mathbb{I}(s) + \sum \log l_2 \mathbb{I}(e)}{2},$$

where $\mathbb{I}(s)$ and $\mathbb{I}(e)$ are one-hot vectors representing the ground truths of pointers.

As a core task in ABSA, aspect extraction (AE) aims to find aspects that reviewers have expressed opinions on [?]. In supervised settings, it is typically modeled as a sequence labeling task, where each token from a sentence is labeled as one of $\{\underline{B}egin, \underline{I}nside, \underline{O}utside\}$. A continuous chunk of tokens that are labeled as one B and followed by zero or more I s forms an aspect. The input sentence with m words is constructed as $x = ([CLS], x_1, \dots, x_m, [SEP])$. After $h = \text{BERT}(x)$, we apply a dense layer and a softmax for each position of the sequence: $l_3 = \text{softmax}(W_3 \cdot h + b_3)$, where $W_3 \in \mathbb{R}^{3 \times r_h}$ and $b_3 \in \mathbb{R}^3$ (3 is the total number of labels (BIO)). Softmax is applied along the dimension of labels for each position and $l_3 \in [0, 1]^{3 \times |x|}$. The labels are predicted as taking argmax function at each position of l_3 and the loss function is the averaged cross entropy across all positions of a sequence.

As a subsequent task of AE, aspect sentiment classification (ASC) aims to classify the sentiment polarity (positive, negative, or neutral) expressed on an aspect extracted from a review sentence. There are two inputs to ASC: an aspect and a review sentence mentioning that aspect.

Let $x = ([CLS], q_1, \dots, q_m, [SEP], d_1, \dots, d_n, [SEP])$, where q_1, \dots, q_m now is an aspect (with m tokens) and d_1, \dots, d_n is a review sentence containing that aspect. After $h = \text{BERT}(x)$, we leverage the representations of $[CLS]$ $h_{[CLS]}$, which is the aspect-aware representation of the whole input. The distribution of polarity is predicted as $l_4 = \text{softmax}(W_4 \cdot h_{[CLS]} + b_4)$, where $W_4 \in \mathbb{R}^{3 \times r_h}$ and $b_4 \in \mathbb{R}^3$ (3 is the number of polarities). Softmax is applied along the dimension of labels on $[CLS]$: $l_4 \in [0, 1]^3$. Training loss is the cross-entropy on the polarities.

As a summary of these tasks, insufficient supervised training data significantly limits the performance gain across these 3 review-based tasks. Although BERT’s pre-trained weights strongly boost the performance of many other NLP tasks on formal texts, we observe in Chapter ?? that BERT’s weights only result in a limited gain or worse performance compared with existing baselines.

Post-training datasets

For domain knowledge post-training, we use Amazon laptop reviews [?] and Yelp Dataset Challenge reviews (<https://www.yelp.com/dataset/challenge>). For laptop domain, we filtered out reviewed products that have appeared in the validation/test reviews to avoid training bias for test data (Yelp reviews do not have this issue as the source reviews of SemEval are not from Yelp). Since the number of reviews is small, we choose a duplicate factor of 5 (each

review generates about 5 training examples) during BERT data pre-processing. This gives us 1,151,863 post-training examples for laptop domain knowledge.

For the restaurant domain, we use Yelp reviews from restaurant categories that the SemEval reviews also belong to [?]. We choose 700K reviews to ensure it is large enough to generate training examples (with a duplicate factor of 1) to cover all post-training steps that we can afford (discussed in Section) (We expect that using more reviews can have even better results but we limit the number of reviews based on our computational power.). This gives us 2,677,025 post-training examples for restaurant domain knowledge learning.

For general RC knowledge, we leverage SQuAD 1.1 [?] that comes with 87,599 training examples from 442 Wikipedia articles. The experiments are discussed in Chapter ?? when addressing each specific task.

4.2.2 – Pre-tuning for End-tasks

Different from post-training that aims for domain adaption, pre-tuning is preparing a pre-trained (or even post-trained) model for a particular end-task. Pre-tuning is proposed to improve the performance of potential end tasks that are limited by training data. Although BERT is successful on many end tasks, those end tasks typically have thousands of training data. In real-world settings of machine learning, it is often the case humans have limited power to provide enough training data for each end task. This makes the proposed pre-tuning very important because the pre-training stage is targeting a general proxy task, not one particular end task, which leads to a large discrepancy or gap between the pre-training and fine-tuning

task. This is especially true when the masked language model aims to guess the correct tokens for [MASK]s, which are never appear in an end task.

Beyond that, many NLP tasks have more complex formats than the format of MLM (or next sentence prediction (NSP) if the proxy task has that). This difference yields an even larger discrepancy that a limited training data for an end task cannot fine-tune the pre-trained model enough. In my research on pre-tuning, I give a task called review conversational reading comprehension (RCRC), which needs to carry multiple past turns of question answering as input to the model that does not appear in the pre-training stage of BERT. We define the textual format of RCRC in the following way and form a pre-tuning task to improve the supervised learning from limited end task data.

Textual Format

Inspired by the DrQA system [?], we formulate an input example x for both RCRC fine-tuning and pre-tuning (We share the same notation for both tasks for brevity.) as a composition of the context C , the current question q_k , and a review d :

$$([\text{CLS}] [\text{Q}] q_1 [\text{A}] a_1 \dots [\text{Q}] q_{k-1} [\text{A}] a_{k-1} [\text{Q}] q_k [\text{SEP}] d_{1:n} [\text{SEP}]),$$

where past QA pairs $q_1, a_1, \dots, q_{k-1}, a_{k-1}$ in C are concatenated and separated by two tokens [Q] and [A] and then concatenated with the current question q_k as the left side of BERT and the right side is the review document. One can observe that BERT lacks the basic understanding of the RCRC task regarding both the input and output, such as the above input format and textual spans in a review. Limited training data of (RC)₂ may not be sufficient to learn such a

complex input and output. We propose a pre-tuning stage that can enhance the understanding of the input/output before fine-tuning on $(RC)_2$.

Data Formulation for Pre-tuning

We first formulate the data for pre-tuning that aims to address the understanding of the textual format. As we have no annotated data except the limited $(RC)_2$ data, we harvest domain QA pairs and reviews, which are typically organized under an entity (a laptop or a restaurant). The QA pairs and reviews are combined to produce the pre-tuning examples. The process is given in Algorithm ??.

Algorithm 3: Data Generation Algorithm

Input : \mathcal{Q} : a set of QA pairs; \mathcal{R} : a set of reviews; h_{max} : maximum turns in context.

Output: \mathcal{T} : pre-tuning data.

```

1  $\mathcal{T} \leftarrow \{\}$ 
2 for  $(q', a') \in \mathcal{Q}$  do
3    $x \leftarrow [\text{CLS}]$ 
4    $h \leftarrow \text{RandInteger}([0, h_{max}])$ 
5   for  $1 \rightarrow h$  do
6      $q'', a'' \leftarrow \text{RandSelect}(\mathcal{Q} \setminus (q', a'))$ 
7      $x \leftarrow x \oplus [\mathcal{Q}] \oplus q'' \oplus [\text{A}] \oplus a''$ 
8   end
9    $x \leftarrow x \oplus [\mathcal{Q}] \oplus q' [\text{SEP}]$ 
10   $r_{1:m} \leftarrow \text{RandSelect}(\mathcal{R})$ 
11  if  $\text{RandFloat}([0.0, 1.0]) > 0.5$  then
12     $(\_, a) \leftarrow \text{RandSelect}(\mathcal{Q} \setminus (q', a'))$ 
13     $(u, v) \leftarrow (1, 1)$ 
14  end
15  else
16     $a \leftarrow a'$ 
17     $(u, v) \leftarrow (|x|, |x| + |a|)$ 
18  end
19   $l \leftarrow \text{RandInteger}([0, u])$ 
20   $d_{1:n} \leftarrow r_{0:l} \oplus a \oplus r_{l+1:u}$ 
21  if  $u > 1$  then
22     $(u, v) \leftarrow (u + |r_{0:l}|, v + |r_{0:l}|)$ 
23  end
24   $x \leftarrow x \oplus d_{1:n} \oplus [\text{SEP}]$ 
25   $\mathcal{T} \leftarrow \mathcal{T} + (x, (u, v))$ 
26 end

```

The inputs to Algorithm ?? are a set of QA pairs and a set of reviews belonging to the same entity and the maximum number of turns in the context. The output is the pre-tuning data, which is initialized in Line 1. Each example is denoted as $(x, (u, v))$, where x is the input example and (u, v) indicates the boundary (starting and ending indexes) of an answer for the auxiliary objective (discussed in the next section). Given a QA pair (q', a') in Line 2, we first build the left side of input example x in Line 3-9. After initializing input x in Line 3, we randomly determine the number of turns in the context in Line 4 and concatenate \oplus these turns of QA pairs in Line 5-8, where $\mathcal{Q} \setminus (q', a')$ ensures the current QA pair (q', a') is not chosen. In Line 9, we concatenate with the current question q' . Lines 10-23 build the right side of input example x and the answer boundary. In Line 10, we randomly draw a review of r with m sentences. To challenge the pre-tuning stage to discover the semantic relatedness between q' and a' (for the auxiliary objective), we first decide whether to allow the right side of x contains a' (Line 16) for q' or a random (negative/no) answer a in Lines 11-12. We also come up with two indexes u and v initialized in Lines 13 and 17. Then, we insert a into review r by randomly picking one from the $m + 1$ locations in Lines 19-20. This gives us $d_{1:n}$, which has n tokens. We further update u and v to allow them to point to the chunk boundaries of a' . Otherwise, BERT should detect no a' on the right side and point to [CLS] ($u, v = 1$). Finally, examples are aggregated in Line 25. Algorithm ?? is run k times to allow for enough samplings. Following BERT, we still randomly mask some words in each example x but omitted here for brevity.

Auxiliary Objective

Besides the input, we further adapt BERT to the output of RCRC with an auxiliary objective. The design of this auxiliary objective is to mimic a prediction of a textual span in RCRC, which aims to predict the token spans of an answer randomly inserted in the review or *NO ANSWER* if a randomly drawn negative answer appears. The implementation of both the auxiliary objective and the RCRC model is similar to BERT for SQuAD 2.0 [?], so we omit them for brevity. After pre-tuning, we fine-tune using the $(RC)_2$ dataset to show the performance of RCRC. The results of RCRC is discussed in ??.

CHAPTER 5

LIFELONG GRAPH REPRESENTATION LEARNING

Besides classification and word representations that aims to turn unstructured text into a structured form, knowledge graph is an important form of structured data that possess human knowledge. Different from unstructured text, knowledge graph is more interpretable and maintainable by human. Given the discrete nature of the knowledge graph, it enables the dynamic accumulation of structured knowledge for future use, which is the goal of lifelong learning. In this setting, one task in lifelong learning can contribute to the updates of a knowledge graph and the future task can leverage the updated knowledge graph for better reasoning or prediction. Thus, I target a lifelong graph representation learning task, where the model should learn representations for the changes of knowledge graph for better reasoning or prediction.

5.1 Motivation

Existing research on the knowledge graph mostly assumes a static graph. This is because they assume a (factoid) knowledge graph, where knowledge inside is rather stable and seldom change. This is true for most factoid-based knowledge in the world. The changes of the knowledge mostly happen when new events happen and engineers can periodically add new entity or relation to the knowledge graph when enough statistics of data are collected for more reliable updates of knowledge.

However, in contrast, non-factoid knowledge is rather dynamic and needs more updates. Non-factoid knowledge can be that knowledge that does not have agreement among a group of people, but particular to one or a small group of people. There is probably no true or false regarding these kinds of knowledge. These include the experience of a user and their preference or sentiment. This kind of knowledge is rather unstable because even the same person can change their mind quickly and a lifelong learning system should be able to capture such changes quickly.

One important application regarding this kind of knowledge is a recommendation, as almost all recommendation models are user-specific models, which have different predictions for different users. Typically these models aim to learn users and items profile, such as using matrix factorization based on click-through data. Unfortunately, existing recommendation models aim to learn static user and item profiles. These static profiles cannot capture the changes in users' needs. As such, conversational recommendation [?, ?] is a novel type of task that allows using an interactive dialog to collect users' up-to-date preference to update the user profile. Although existing research in conversational recommendation aims to update the user profile in the hidden space, modeling users' profile as a knowledge graph has the following two (2) benefits: (1) it allows for easier updates and maintenance of user profile for long-term use given the semantics of the hidden space is mostly undefined and determined by random initialization; (2) it allows for interpretability given the discrete structure of knowledge graph.

To this end, a user's profile can be ideally represented as a knowledge graph, which requires frequent updates, even in one turn of a dialog between the AI agent and a user. More impor-

tantly, I aim to design a universal knowledge graph that contains almost all knowledge in a recommendation setting, including information from both the users and items. As such, this chapter focuses on designing and maintaining a knowledge graph for recommendation under a conversation setting, where the representation of the knowledge graph needs to be updated frequently for reasoning a better dialog policy.

5.2 Lifelong Knowledge Graph Reasoning

As an example of lifelong representation learning over a dynamic knowledge graph, we demonstrate the usage of dynamic knowledge graph reasoning with an application in dialog system. This task is called conversational recommendation. Conversational recommendation is a combination of dialog system and recommender system and aims to collect a user’s up-to-date knowledge for recommendation. We focus on the graph maintenance aspect of conversational recommendation in this chapter and detail the application side of this task in Chapter ?? . We call the knowledge graph about a user memory because it is the agent’s memory about a user.

Memory-grounded Conversational Recommendation: Given the history of previous items \mathcal{H} (interacted or visited, etc.), candidate items \mathcal{C} for recommendation, and their attributes (values), an agent first (1) constructs a user memory graph $\mathcal{G} = \{(e, r, e') | e, e' \in \mathcal{E}, r \in \mathcal{R}\}$ for user e_u ; then (2) for each turn $d \in D$ of a dialog, the agent updates \mathcal{G} with tuples of preference $\mathcal{G}' \leftarrow \mathcal{G} \cup \{(e_u, r_1, e_1), \dots\}$; (3) performs reasoning over \mathcal{G}' to yield a dialog policy π that either (i) performs more rounds of interaction by asking for more preference, or (ii) predicts optimal (or ground truth) items for recommendations $\mathcal{T} \subset \mathcal{C}$.

Graph Formulation

In this section, we describe the formulation of a user memory graph based on each scenario of dialog (the formulation of a scenario in conversational recommendation can be found in Chapter ??). There are many design choices of constructing a user memory graph and our goal is to model the graph with easy extensibility, maintenance and interpretability for the generation of dialog policy π through the course of a conversation. As a reminder, a user memory graph is denoted as $\mathcal{G} = \{(e, r, e') | e, e' \in \mathcal{E}, r \in \mathcal{R}\}$, which is essentially a heterogeneous graph with typed (or meta) entities and relations.

We first define the entity sets and relations in ?. To illustrate the construction of a user memory graph and its maintenance, we describe an example in ?. Consider a user Bob e_{Bob} , which has a memory $(e_{\text{Bob}}, r_{\text{has_mem}}, e_m)$ (not shown in the figure). This memory entity has a $(e_m, r_{\text{visited}}, e_{\text{Seas}})$ relation to item e_{Seas} (a restaurant). e_{Seas} has values $(e_{\text{Seas}}, r_{\text{has_aspect}}, e_{\text{affordable}})$ and $(e_{\text{Seas}}, r_{\text{has_aspect}}, e_{\text{Japanese}})$. Those two values belong to slots s_{price} and s_{category} , respectively. The values $e_{\text{affordable}}$ and e_{Japanese} are also shared by items e_{Yayoi} and e_{Basil} , respectively. As a result, we can see this user memory graph is highly extendable as new relations or entities can be easily integrated as more experience or preference come from the user. This can be further illustrated in ?, where we add relations about users' sentiment over 3 rounds of interactions. When it comes to the final recommended item e_{Basil} , we can provide explanations that the user is positive on $e_{\text{affordable}}$ and e_{Japanese} , leading to the recommendation e_{Basil} (as in paths $(e_{\text{Bob}} \rightarrow r_{\text{pos_on}} \rightarrow e_{\text{affordable}} \rightarrow r_{\text{has_aspect}} \rightarrow e_{\text{Basil}})$ and $(e_{\text{Bob}} \rightarrow r_{\text{pos_on}} \rightarrow e_{\text{Japanese}} \rightarrow r_{\text{has_aspect}} \rightarrow e_{\text{Basil}})$, respectively). Further, another important explanation is the path $(e_{\text{Bob}} \rightarrow r_{\text{visited}} \rightarrow$

Entity Sets	Explanation
\mathcal{U}	user entities
\mathcal{M}	memory entities
\mathcal{I}	item entities: $\mathcal{C} \cup \mathcal{H}$
\mathcal{S}	slot entities defined in ??
\mathcal{V}	value entities
Relation Types	
$(\mathcal{U}, \text{has_mem}, \mathcal{M})$	a user u has a memory entity m
$(\mathcal{M}, \text{visited}, \mathcal{I})$	a memory m is about an item i
$(\mathcal{I}, \text{has_aspect}, \mathcal{V})$	an item i has a value v
$(\mathcal{V}, \text{is_a}, \mathcal{S})$	a value v belongs to a slot s
$(\mathcal{M}, \text{pos_on}, \mathcal{V}/\mathcal{I})$	m is positive on a value or item
$(\mathcal{M}, \text{neg_on}, \mathcal{V}/\mathcal{I})$	m is negative on a value or item
$(\mathcal{M}, \text{neu_on}, \mathcal{V}/\mathcal{I})$	m is neutral on a value or item

TABLE V

Ontology of Memory Graph

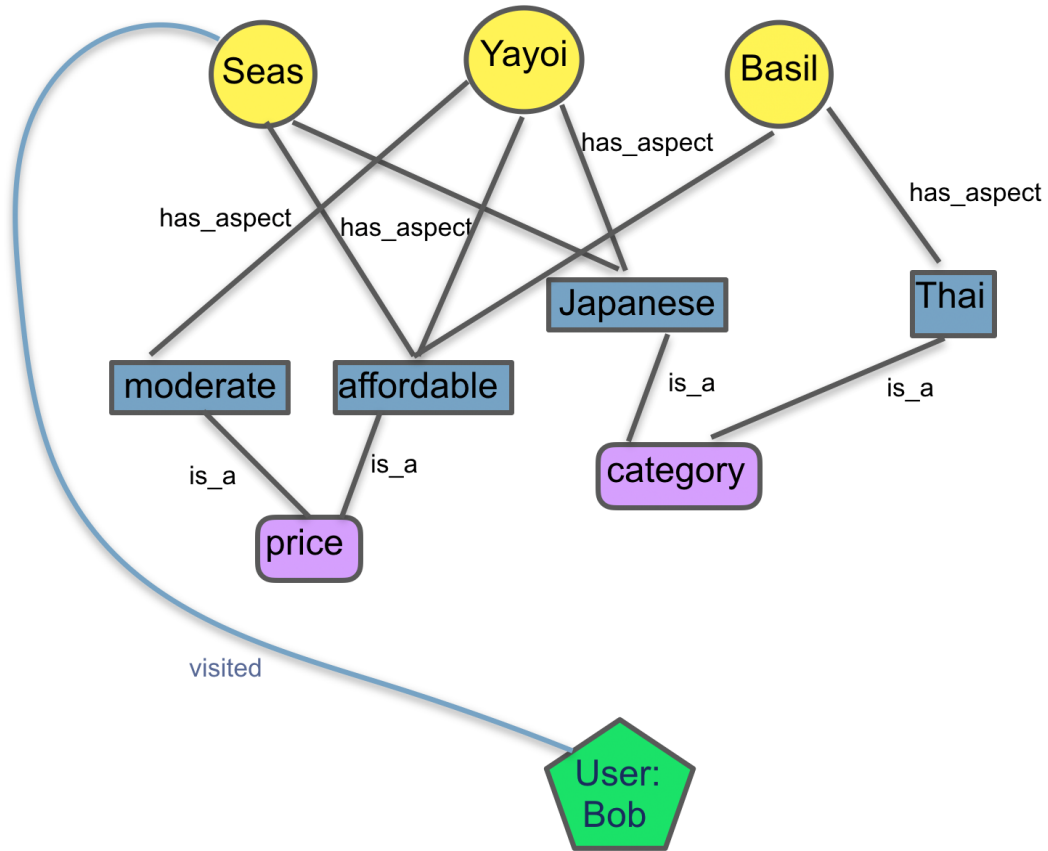


Figure 6. Construction of user memory graph

$e_{\text{Seas}} \rightarrow r_{\text{has_aspect}} \rightarrow e_{\text{affordable}} \rightarrow r_{\text{has_aspect}} \rightarrow e_{\text{Basil}}$) which draws the relevance from a visited item to the current recommendation.

As such, another benefit of formulating such a user memory graph is that all items, slots, values of a generated dialog policy π can be directly mapped to certain (item, slot or value) entities in the user memory graph. This paves the way for reasoning over the user memory graph for explainable dialog policy generation (Chapter ??).

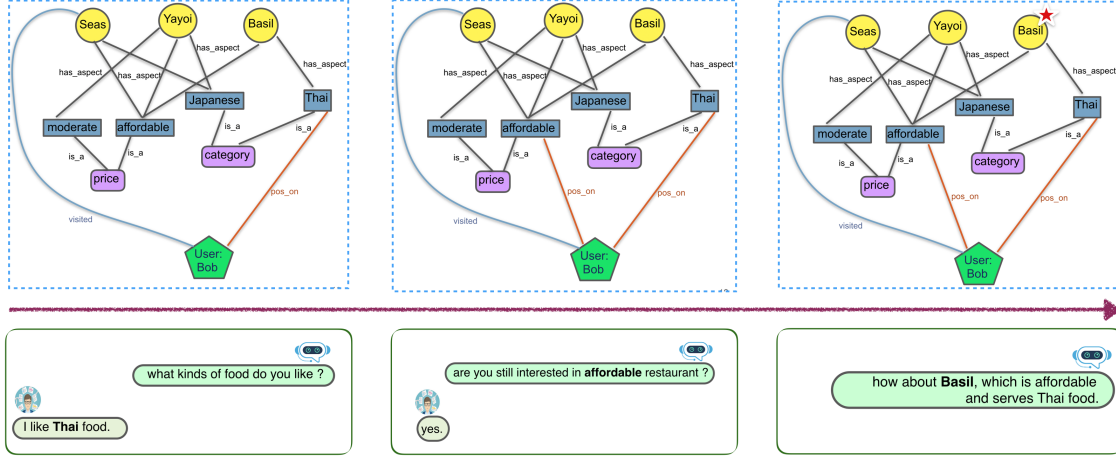


Figure 7. Dynamic updates of memory graph

5.3 Graph Reasoner

We propose a model called User Memory Graph Reasoner (UMGR) to reason the turn-level dialog policy over the user memory graph.

Input: the input of UMGR is the past dialog acts up to the current turn from the user a , the updated user memory graph \mathcal{G}' , which contains all the knowledge about the items their associated values and slots, and visited items. We further accumulate all updates from the user (e.g., via the assumed results from NLU or state tracking) in the form of last 3 types of relations in Table 4 (Similar to visited items, we add a new memory entity for the current dialog and then associate all the new relations to that memory entity.).

Output: UMGR’s output is the dialog policy $\pi = (\hat{y}^{\mathcal{A}}, \hat{y}^{\mathcal{C}}, \hat{y}^{\mathcal{S}}, \hat{y}^{\mathcal{V}})$ for the current turn, where $\mathcal{A}, \mathcal{C}, \mathcal{S}, \mathcal{V}$ indicate the space of dialog acts, candidate items, slots and values, respectively.

The predictions from $\hat{y}^{\mathcal{C}}$, $\hat{y}^{\mathcal{S}}$ and $\hat{y}^{\mathcal{V}}$ essentially provides a ranking over those entity sets. For example, when $\hat{y}^{\mathcal{A}} = \textit{Recommendation}$, the top-1 entity $\arg \max_{e_i \in \mathcal{C}}(\hat{y}^{\mathcal{C}})$ will be provided to the user. Similarly, $\hat{y}^{\mathcal{A}} = \textit{Open Question}$ is related to the top-1 slot $\arg \max_{e_s \in \mathcal{S}}(\hat{y}^{\mathcal{S}})$ and $\hat{y}^{\mathcal{A}} = \textit{Yes/no Question}$ is related to the top-1 value $\arg \max_{e_v \in \mathcal{V}}(\hat{y}^{\mathcal{V}})$. In this way, all arguments of a dialog act can be mapped to certain entities in the user memory graph for a structured explanation instead of decoding from latent space.

To enable the reasoning over a user memory graph on-the-fly, we incorporate a Relational Graph Convolutional Networks (R-GCN) [?] inside UMGR. R-GCN is a GCN [?] with typed relations, where each relation is associated with their weights to enable reasoning over a heterogeneous graph. UMGR first encodes past dialog acts \mathbf{a} and entities $e \in \mathcal{E}$ into hidden dimensions.

$$\begin{aligned} h_a &= \text{LSTM}(W^{\mathcal{A}}(\mathbf{a})), \\ h_j^{(0)} &= W^{\mathcal{E}}(e_j), \end{aligned} \tag{5.1}$$

where $W^{\mathcal{A}}$ and $W^{\mathcal{E}}$ are embedding layers and the past dialog acts are further encoded by an LSTM encoder. We further allow on-the-fly reasoning over (new) items by sharing the embedding weights for different items (as a special entity $\langle \text{ITEM} \rangle$) in $W^{\mathcal{E}}$. Then each entity in the user memory graph is encoded by multiple layers of R-GCN.

$$h_j^{(l+1)} = \text{LeakyReLU}\left(\sum_{r \in \mathcal{R}} \sum_{k \in \mathcal{N}_j^r} \frac{1}{|\mathcal{N}_j^r|} W_r^{(l)} h_j^{(l)}\right), \tag{5.2}$$

where $h_j^{(l)}$ (j can be any type of entity) is the hidden state of entity e_j in the l -th layer of R-GCN. \mathcal{N}_j^r is entity e_j 's neighbor in relation type r and $W_r^{(l)}$ is the weights associated with r in the l -th layer to transform $h_j^{(l)}$. The R-GCN layer updates the hidden states of each entity with the incoming messages in the form of their neighbors' hidden states type-by-type. Then R-GCN sums over all types before passing through the activation. The hidden states from the last layer of R-GCN is pasted into an aggregation layer.

$$h^{\text{ag}} = \frac{1}{|\mathcal{C} \cup \mathcal{S} \cup \mathcal{V}|} \sum_{e_j \in \mathcal{C} \cup \mathcal{S} \cup \mathcal{V}} (W^{\text{ag}} h_j^{(l+1)} + b^{\text{ag}}), \quad (5.3)$$

where W^{ag} and b^{ag} are weights for aggregation layer. The purpose of having an aggregation layer is to leverage the information in the user memory graph for predicting the dialog acts, which is a classification problem. The loss for dialog acts is defined as

$$\begin{aligned} \hat{y}^{\mathcal{A}} &= \text{Softmax}(W^{\mathcal{A}}(h_a \oplus h^{\text{ag}}) + b^{\mathcal{A}}), \\ \mathcal{L}^{\mathcal{A}} &= \text{CrossEntropyLoss}(\hat{y}^{\mathcal{A}}, y^{\mathcal{A}}), \end{aligned} \quad (5.4)$$

where \oplus is the concatenation operation and $y^{\mathcal{A}}$ is the annotated dialog act. Further, all item, slot and value entities are trained by log loss for ranking. For example, the loss for candidate items \mathcal{C} is defined as

$$\begin{aligned} \hat{y}_i &= \text{Sigmoid}(W^{\mathcal{I}} h_i + b^{\mathcal{I}}), \\ \mathcal{L}^{\mathcal{C}} &= \text{LogLoss}(\hat{y}^{\mathcal{C}}, y^{\mathcal{C}}). \end{aligned} \quad (5.5)$$

Similarly, we obtain losses \mathcal{L}_S , \mathcal{L}_V for slot entities S and value entities V , respectively. Finally, the total loss is the sum over all losses for dialog acts, items, slots and values:

$$\mathcal{L} = \mathcal{L}^A + \alpha\mathcal{L}^C + \beta\mathcal{L}^S + \gamma\mathcal{L}^V, \quad (5.6)$$

where α, β and γ are hyper-parameters to align losses of different scales. Note that during training and prediction, all invalid dialog acts (e.g., user dialog acts) and entities (e.g., not appear in a user memory graph) are masked out. As we can see, unlike traditional recommender systems, UMGR does not learn (or “overfit to”) any prior knowledge about users into the weights. Instead, it reasons the dialog policy on-the-fly in each turn based on the updated user memory graph. We will detail the task of conversational recommendation and experiments in the next chapter.

CHAPTER 6

NLP APPLICATIONS

Parts of this chapter were previously published or arXived as the following papers:

- “Double Embeddings and CNN-based Sequence Labeling for Aspect Extraction” in ACL 2018 [?] (DOI: <http://dx.doi.org/10.18653/v1/P18-2094>, with arXiv version <https://arxiv.org/abs/1805.04601> [?]);
- “BERT Post-Training for Review Reading Comprehension and Aspect-based Sentiment Analysis” in NAACL 2019 [?] (DOI: <http://dx.doi.org/10.18653/v1/N19-1242>, with an arXiv version <https://arxiv.org/abs/1904.02232> [?]);
- arXiv paper “Review Conversational Reading Comprehension” <https://arxiv.org/abs/1902.00821> [?];
- arXiv paper “A Failure of Aspect Sentiment Classifiers and an Adaptive Re-weighting Solution” <https://arxiv.org/abs/1911.01460> [?];
- “CER: Complementary Entity Recognition via Knowledge Expansion on Large Unlabeled Product Reviews ” in BigData 2016 [?] (with arXiv version <https://arxiv.org/abs/>

1612.01039 [?]);

- arXiv paper “Supervised Complementary Entity Recognition with Augmented Key-value Pairs of Knowledge ” <https://arxiv.org/abs/1705.10030> [?].

In this chapter, I apply the concept of lifelong representation learning for a wide spectrum of NLP applications. Natural language processing needs lifelong learning because as a form of human knowledge, texts need to be integrated into the agent for understanding and reasoning for agent’s policy.

I will first focus on two (2) tasks in aspect-based sentiment analysis: aspect extraction and aspect sentiment classification. Then I switch to a novel task in my earlier years of PhD study: complementary entity recognition, which aims to identify multiple entities and their relations. Then I switch to review-based question answering. Reading comprehension is an important task in question answering because its good usage of human written texts for answers. I will propose some novel review-based QA tasks, with results indicating the importance of lifelong representation. Lastly, I will switch to the dialog system. I will first talk about the conversational version of QA and then switch to a novel type of dialog system: conversational recommendation, which leverages lifelong graph representation learning for reasoning dialog policy.

6.1 Sentiment Analysis

Sentiment analysis aims to detect people’s polarity from opinion text such as reviews and tweets [?]. More specifically, aspect-based sentiment analysis (ABSA) focuses in fine-grained sentiment analysis over aspects and possibly their categories. ABSA aims to detect the aspects *a*

in opinion texts and their associated polarities (a, p) s. This naturally has two sub-tasks in ABSA: aspect extraction and aspect sentiment classification.

6.1.1 – Aspect Extraction

Aspect extraction is an important task in sentiment analysis [?] and has many applications to its downstream tasks [?]. It aims to extract opinion targets (or aspects) from opinion text. In product reviews, aspects are product attributes or features. For example, from “*Its speed is incredible*” in a laptop review, it aims to extract “speed”.

Aspect extraction has been performed using supervised [?, ?, ?] and unsupervised approaches [?, ?, ?, ?, ?, ?]. Recently, supervised deep learning models achieved state-of-the-art performances [?]. Many of these models use handcrafted features, lexicons, and complicated neural network architectures [?, ?, ?, ?]. Although these approaches can achieve better performances than their prior works, two other considerations are also important. (1) Automated feature (representation) learning is always preferred. How to achieve competitive performances without manually crafting features is an important question. (2) According to Occam’s razor principle [?], a simple model is always preferred over a complex model. This is especially important when the model is deployed in a real-life application (e.g., chatbot), where a complex model will slow down the speed of inference. Thus, to achieve competitive performance whereas keeping the model as simple as possible is important. This paper proposes such a model.

To address the first consideration, we propose a double embeddings mechanism (as discussed in Chapter ??) that is shown crucial for aspect extraction. The embedding layer is the very first layer, where all the information about each word is encoded. The quality of the embeddings

determines how easily later layers (e.g., LSTM, CNN or attention) can decode useful information. Existing deep learning models for aspect extraction use either a pre-trained general-purpose embedding, e.g., GloVe [?], or a general review embedding [?]. However, aspect extraction is a complex task that also requires fine-grained domain embeddings. For example, in the previous example, detecting “speed” may require embeddings of both “Its” and “speed”. However, the criteria for good embeddings for “Its” and “speed” can be different. “Its” is a general word and the general embedding (trained from a large corpus) is likely to have better representation for “Its”. But, “speed” has a very fine-grained meaning (e.g., how many instructions per second) in the *laptop* domain, whereas “speed” in general embeddings or general review embeddings may mean how many miles per second. So using in-domain embeddings is important even when the in-domain embedding corpus is not large. Thus, we leverage both general embeddings and domain embeddings and let the rest of the network to decide which embeddings have more useful information.

To address the second consideration, we use a pure Convolutional Neural Network (CNN) [?] model for sequence labeling. Although most existing models use LSTM [?] as the core building block to model sequences [?,?], we noticed that CNN is also successful in many NLP tasks [?,?,?]. One major drawback of LSTM is that LSTM cells are sequentially dependent. The forward pass and backpropagation must serially go through the whole sequence, which slows down the training/testing process. One challenge of applying CNN on sequence labeling is that convolution and max-pooling operations are usually used for summarizing sequential inputs and the outputs are not well-aligned with the inputs. We call the proposed model Dual Embeddings

CNN (DE-CNN). To the best of our knowledge, this is the first paper that reports a double embedding mechanism and a pure CNN-based sequence labeling model for aspect extraction.

Related Work

Sentiment analysis has been studied at document, sentence and aspect levels [?, ?, ?]. This work focuses on the aspect level [?]. Aspect extraction is one of its key tasks and has been performed using both unsupervised and supervised approaches. The unsupervised approach includes methods such as frequent pattern mining [?, ?], syntactic rules-based extraction [?, ?, ?], topic modeling [?, ?, ?, ?], word alignment [?] and label propagation [?, ?].

Traditionally, the supervised approach [?, ?, ?] uses Conditional Random Fields (CRF) [?]. Recently, deep neural networks are applied to learn better features for supervised aspect extraction, e.g., using LSTM [?, ?, ?] and attention mechanism [?, ?] together with manual features [?, ?]. Further, [?, ?, ?] also proposed aspect and opinion terms co-extraction via a deep network. They took advantage of the gold-standard opinion terms or sentiment lexicon for aspect extraction. The proposed approach is close to [?], where only the annotated data for aspect extraction is used. However, we will show that our approach is more effective even compared with baselines using additional supervision and/or resources.

The proposed embedding mechanism is related to cross domain embeddings [?, ?] and domain-specific embeddings [?, ?]. However, we require the domain of the domain embeddings must exactly match the domain of the aspect extraction task. CNN [?, ?] is recently adopted for named entity recognition [?]. CNN classifiers are also used in sentiment analysis [?, ?]. We adopt CNN for sequence labeling for aspect extraction because CNN is simple and parallelized.

Double Embedding for Sequence Labeling

Following the idea of fusion general and domain-specific embeddings in ??, we have the following CNN-based model for aspect extraction.

The proposed model is depicted in ??. It has 2 embedding layers, 4 CNN layers, a fully-connected layer shared across all positions of words, and a softmax layer over the labeling space $\mathcal{Y} = \{B, I, O\}$ for each position of inputs. Note that an aspect can be a phrase and B, I indicate the beginning word and non-beginning word of an aspect phrase and O indicates non-aspect words.

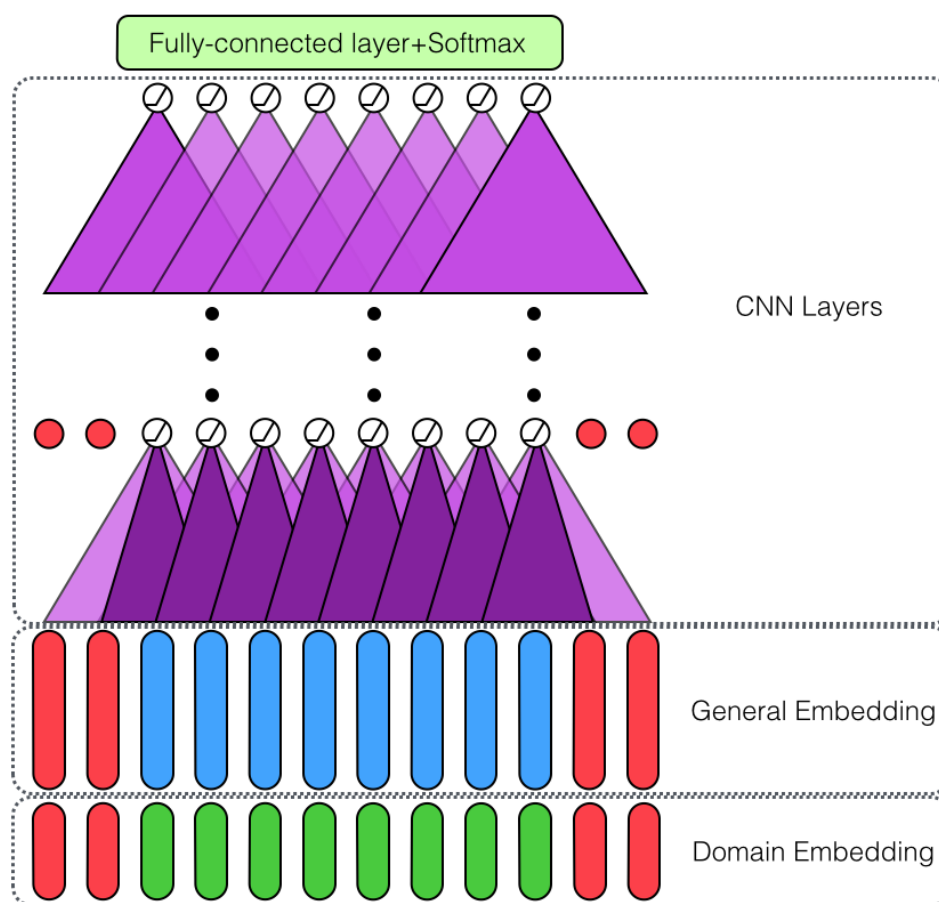


Figure 8. DE-CNN

After leveraging the double embedding layers discussed in Chapter ??, we apply CNN layers for building representations for aspect extraction. A CNN layer has many 1D-convolution

filters and each (the r -th) filter has a fixed kernel size $k = 2c + 1$ and performs the following convolution operation and ReLU activation:

$$x_{i,r}^{(l+1)} = \max \left(0, \left(\sum_{j=-c}^c w_{j,r}^{(l)} x_{i+j}^{(l)} \right) + b_r^{(l)} \right), \quad (6.1)$$

where l indicates the l -th CNN layer. We apply each filter to all positions $i = 1 : n$. So each filter computes the representation for the i -th word along with $2c$ nearby words in its context. Note that we force the kernel size k to be an odd number and set the stride step to be 1 and further pad the left c and right c positions with all zeros. In this way, the output of each layer is well-aligned with the original input \mathbf{x} for sequence labeling purposes. For the first ($l = 1$) CNN layer, we employ two different filter sizes. For the rest 3 CNN ($l \in \{2, 3, 4\}$) layers, we only use one filter size. We will discuss the details of the hyper-parameters in the experiment section. Finally, we apply a fully-connected layer with weights shared across all positions and a softmax layer to compute label distribution for each word. The output size of the fully-connected layer is $|\mathcal{Y}| = 3$. We apply dropout after the embedding layer and each ReLU activation. Note that we do not apply any max-pooling layer after convolution layers because a sequence labeling model needs good representations for every position and max-pooling operation mixes the representations of different positions, which is undesirable (we show a max-pooling baseline in the experiments).

We further conduct BERT model for aspect extraction. We omit the details of BERT for aspect extraction as it is already discussed in Chapter ??.

Description	Training	Testing
	#S./#A.	#S./#A.
SemEval-14 Laptop	3045/2358	800/654
SemEval-16 Restaurant	2000/1743	676/622

TABLE VI

Dataset for AE

Experiments

Following the experiments of a recent aspect extraction paper [?], we conduct experiments on two benchmark datasets from SemEval challenges [?, ?] as shown in Table ?? . The first dataset is from the *laptop* domain on subtask 1 of SemEval-2014 Task 4. The second dataset is from the *restaurant* domain on subtask 1 (slot 2) of SemEval-2016 Task 5. These two datasets consist of review sentences with aspect terms labeled as spans of characters. We use NLTK¹ to tokenize each sentence into a sequence of words.

For the general-purpose embeddings, we use the glove.840B.300d embeddings [?], which are pre-trained from a corpus of 840 billion tokens that cover almost all web pages. These embeddings have 300 dimensions. For domain-specific embeddings, we collect a laptop review corpus and a restaurant review corpus and use fastText [?] to train domain embeddings. The

¹<http://www.nltk.org/>

laptop review corpus contains all laptop reviews from the Amazon Review Dataset [?]. The restaurant review corpus is from the Yelp Review Dataset Challenge¹. We only use reviews from restaurant categories that the second dataset is selected from¹. We set the embedding dimensions to 100 and the number of iterations to 30 (for a small embedding corpus, embeddings tend to be under-fitted), and keep the rest hyper-parameters as the defaults in fastText. We further use fastText to compose out-of-vocabulary word embeddings via subword N-gram embeddings.

Baseline Methods for DE-CNN

We perform a comparison of DE-CNN with three groups of baselines using the standard evaluation of the datasets¹. The results of the first two groups are copied from [?]. The first group uses single-task approaches.

CRF is conditional random fields with basic features¹ and GloVe word embedding [?].

IHS_RD [?] and **NLANGP** [?] are best systems in the original challenges [?,?].

WDEmb [?] enhanced CRF with word embeddings, linear context embeddings and dependency path embeddings as input.

¹<https://www.yelp.com/dataset/challenge>

¹<http://www.cs.cmu.edu/~mehrbod/RR/Cuisines.whl>

¹<http://alt.qcri.org/semeval2014/task4>

¹<http://alt.qcri.org/semeval2016/task5>

¹<http://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html>

LSTM [?,?] is a vanilla BiLSTM.

BiLSTM-CNN-CRF [?] is the state-of-the-art from the Named Entity Recognition (NER) community. We use this baseline¹ to demonstrate that a NER model may need further adaptation for aspect extraction.

The second group uses multi-task learning and also take advantage of gold-standard opinion terms/sentiment lexicon.

RNCRF [?] is a joint model with a dependency tree-based recursive neural network and CRF for aspect and opinion terms co-extraction. Besides opinion annotations, it also uses handcrafted features.

CMLA [?] is a multi-layer coupled-attention network that also performs aspect and opinion terms co-extraction. It uses gold-standard opinion labels in the training data.

MIN [?] is a multi-task learning framework that has (1) two LSTMs for jointly extraction of aspects and opinions, and (2) a third LSTM for discriminating sentimental and non-sentimental sentences. A sentiment lexicon and high precision dependency rules are employed to find opinion terms.

The third group is the variations of DE-CNN.

GloVe-CNN only uses glove.840B.300d to show that domain embeddings are important.

¹<https://github.com/UKPLab/emnlp2017-bilstm-cnn-crf>

Domain-CNN does not use the general embeddings to show that domain embeddings alone are not good enough as the domain corpus is limited for training good general word embeddings.

MaxPool-DE-CNN adds max-pooling in the last CNN layer. We use this baseline to show that the max-pooling operation used in the traditional CNN architecture is harmful to sequence labeling.

DE-OOD-CNN replaces the domain embeddings with out-of-domain embeddings to show that a large out-of-domain corpus is not a good replacement for a small in-domain corpus for domain embeddings. We use all *electronics* reviews as the out-of-domain corpus for the *laptop* and all the Yelp reviews for *restaurant*.

DE-Google-CNN replaces the glove embeddings with GoogleNews embeddings¹, which are pre-trained from a smaller corpus (100 billion tokens). We use this baseline to demonstrate those general embeddings that are pre-trained from a larger corpus performs better.

DE-CNN-CRF replaces the softmax activation with a CRF layer¹. We use this baseline to demonstrate that CRF may not further improve the challenging performance of aspect extraction.

Hyper-parameters of DE-CNN

We hold out 150 training examples as validation data to decide the hyper-parameters. The first CNN layer has 128 filters with kernel sizes $k = 3$ (where $c = 1$ is the number of words

¹<https://code.google.com/archive/p/word2vec/>

¹<https://github.com/allenai/allennlp>

on the left (or right) context) and 128 filters with kernel sizes $k = 5$ ($c = 2$). The rest 3 CNN layers have 256 filters with kernel sizes $k = 5$ ($c = 2$) per layer. The dropout rate is 0.55 and the learning rate of Adam optimizer [?] is 0.0001 because CNN training tends to be unstable.

Results and Analysis of DE-CNN

Table ?? shows that DE-CNN performs the best. The double embedding mechanism improves the performance and in-domain embeddings are important. We can see that using general embeddings (GloVe-CNN) or domain embeddings (Domain-CNN) alone gives an inferior performance. We further notice that the performance on *Laptops* and *Restaurant* domains are quite different. *Laptops* has many domain-specific aspects, such as “adapter”. So the domain embeddings for *Laptops* are better than the general embeddings. The *Restaurant* domain has many very general aspects like “staff”, “service” that do not deviate much from their general meanings. So general embeddings are not bad. Max pooling is a bad operation as indicated by MaxPool-DE-CNN since the max pooling operation loses word positions. DE-OOD-CNN’s performance is poor, indicating that making the training corpus of domain embeddings to be exactly in-domain is important. DE-Google-CNN uses a much smaller training corpus for general embeddings, leading to poorer performance than that of DE-CNN. Surprisingly, we notice that the CRF layer (DE-CNN-CRF) does not help. The CRF layer can improve 1-2% when the laptop’s performance is about 75%. But it doesn’t contribute much when the laptop’s performance is above 80%. CRF is good at modeling label dependences (e.g., label *I* must be after *B*), but many aspects are just single words and the major types of errors (mentioned

Model	Laptop	Restaurant
CRF	74.01	69.56
IHS_RD	74.55	-
NLANGP	-	72.34
WDEmb	75.16	-
LSTM	75.25	71.26
BiLSTM-CNN-CRF	77.8	72.5
RNCRF	78.42	-
CMLA	77.80	-
MIN	77.58	73.44
GloVe-CNN	77.67	72.08
Domain-CNN	78.12	71.75
MaxPool-DE-CNN	77.45	71.12
DE-LSTM	78.73	72.94
DE-ODD-CNN	80.21	74.2
DE-Google-CNN	78.8	72.1
DE-CNN-CRF	80.8	74.1
DE-CNN	81.59	74.37

TABLE VII

F₁ score for AE

later) do not fall in what CRF can solve. Note that we did not tune the hyperparameters of DE-CNN-CRF for practical purposes because training the CRF layer is extremely slow.

One important baseline is BiLSTM-CNN-CRF, which is markedly worse than our method. We believe the reason is that this baseline leverages dependency-based embeddings [?], which could be very important for NER. NER models may require further adaptations (e.g., domain embeddings) for opinion texts.

DE-CNN has two major types of errors. One type comes from inconsistent labeling (e.g., for the restaurant data, the same aspect is sometimes labeled and sometimes not). Another major type of error comes from unseen aspects in test data that require the semantics of the conjunction word “and” to extract. For example, if A is an aspect and when “A and B” appears, B should also be extracted but not. We leave this to future work.

We further conduct experiments for the results of DE-CNN with language model (BERT) based methods.

Hyper-parameters of BERT

We adopt **BERT_{BASE}** (uncased) as the basis for all experiments¹. Since post-training may take a large footprint on GPU memory (as BERT pre-training), we leverage FP16 computation¹ to reduce the size of both the model and hidden representations of data. We set a static loss scale of

¹We expect **BERT_{LARGE}** to have better performance but leave that to future work due to limited computational power.

¹<https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html>

2 in FP16, which can avoid any over/under-flow of floating-point computation. The maximum length of post-training is set to 320 with a batch size of 16 for each type of knowledge. The number of sub-batch u is set to 2, which is good enough to store each sub-batch iteration into a GPU memory of 11G. We use Adam optimizer and set the learning rate to be $3e-5$. We train 70,000 steps for the laptop domain and 140,000 steps for the restaurant domain, which roughly have one pass over the pre-processed data on the respective domain.

Baseline Methods for BERT

BERT leverages the vanilla BERT pre-trained weights and fine-tunes on all 3 end tasks. We use this baseline to answer RQ2 and show that BERT’s pre-trained weights alone have limited performance gains on review-based tasks.

BERT-DK post-trains BERT’s weights only on domain knowledge (reviews) and fine-tunes on the 3 end tasks. We use BERT-DK and the following BERT-MRC to answer RQ3.

BERT-MRC post-trains BERT’s weights on SQuAD 1.1 and then fine-tunes on the 3 end tasks.

BERT-PT (proposed method) post-trains BERT’s weights using the joint post-training algorithm in Chapter ?? and then fine-tunes on the 3 end tasks.

Results of BERT

Domain	Laptop	Rest.
Methods	F1	F1
DE-CNN [?]	81.59	74.37
BERT	79.28	74.1
BERT-DK	83.55	77.02
BERT-MRC	81.06	74.21
BERT-PT	84.26	77.97

TABLE VIII

BERT for AE in F1.

we found that a great performance boost comes mostly from domain knowledge post-training, which indicates that contextualized representations of domain knowledge are very important for AE. BERT-MRC has almost no improvement in restaurant, which indicates Wikipedia may not know aspects of restaurant. We suspect that the improvements on laptop come from the fact that many answer spans in SQuAD are noun terms, which bear a closer relationship with laptop aspects. Errors mostly come from annotation inconsistency and boundaries of aspects (e.g., apple OS is predicted as OS). Restaurant suffers from rare aspects like the names of dishes.

6.1.2 – Aspect Sentiment Classification

As a downstream task of aspect extraction, Asspect-based sentiment classification (ASC) is an important task in fine-grained sentiment analysis. In this subsection, we focus on two

(2) improvements of ASC: (1) using pre-trained / post-trained BERT for ASC; (2) improving the post-trained BERT with the technique of hard example learning. Hard examples are one challenge for ASC because its datasets that are typically rare but very important for learning aspect-level sentiment (e.g., sentences with different polarities for different aspects).

Hard Examples Learning for Aspect Sentiment Classification

Hard examples play an non-neglectable role in many machine learning applications. Many datasets contain a certain number of rare examples that are hard to learn, as can be found in imbalance issues or fairness issues in machine learning ¹. On one hand, the reason is from data collection that can easily and unintentionally bias a dataset. It is very hard, if not impossible, for humans to provide an ideal dataset to a machine learning model. As in the object detection problem [?, ?] in computer vision, it can easily come up with long-tailed hard examples, given it is almost impossible to manually balance objects appear in one image. On the other hand, it is important for machine learning algorithms to avoid such issues.

Aspect-based sentiment classification (ASC) is an important task in detecting the opinion expressed about an aspect (or an opinion target) [?, ?]. However, ASC also suffers from the difficulty of learning from hard examples. For example, “The screen is good but not the battery” requires to detect two fine-grained and *contrastive opinions* within the same sentence: a positive opinion towards “screen” and a negative opinion towards “battery”. We call this type of sentences **contrastive sentence** and [?] found that such sentences are rare but hard to learn from

¹<https://venturebeat.com/2019/01/24/amazon-rekognition-bias-mit/>

existing ASC datasets. But these sentences are extremely important, because without them, the task of ASC turns into detecting sentence-level sentiment without the need to know the referred aspects.

In this paper, instead of manually addressing this issue from data collection, we focus on algorithms that automatically learn from such hard examples. We propose a simple training algorithm called adaptive re-weighting (ARW), which dynamically keeps focusing on hard examples. Since other types of hard examples are hard to identify, we thus use contrastive sentences as the proxy to evaluate ASC on hard examples. We experimentally show that models trained with ARW significantly improves contrastive sentences, while still keep competitive or even better performance on the full set of test examples.

Related Work

Hard example mining is mostly studied in object detection [?, ?], which aims to detect long-tailed and imbalanced classes of sub-regions in one image. In [?], a loss-based weighting is proposed to adjust weights without explicitly re-balance the complex class distribution.

Aspect sentiment classification (ASC) [?] is an important task in sentiment analysis [?, ?]. It is different from document or sentence-level sentiment classification (SSC) [?, ?, ?, ?] as it focuses on fine-grained opinion on each specific aspect. It is either studied as a single task or a joint learning task together with aspect extraction [?, ?, ?]. The problem has been widely dealt with using neural networks [?, ?, ?]. ASC is also studied in transfer learning or domain adaptation, such as leveraging large-scale corpora that are unlabeled or weakly labeled (e.g., using overall rating of a review as the label) [?, ?] and transferring from other tasks/domains [?, ?, ?]. Our

re-weighting method is related to AdaBoost [?], which is a well-known ensemble algorithm that makes predictions collectively via a sequence of weak classifiers. Our work is different as we don't build a sequence of classifiers like AdaBoost but only one classifier. Neither is our model an ensemble model. Our weight updating is also different from AdaBoost as we do it in each epoch of training. We aim to improve the training process of a deep learning model by adaptively discovering incorrect examples (which cover contrastive sentences) and give them higher weights to focus on for subsequent training process. We also notice that AdaBoost is not frequently used in deep learning [?, ?] probably due to the complexity of deep learning models which are not weak learners.

Adaptive Re-Weighting Algorithm

The hardness of an example is highly associated with its rareness in a dataset because those rare examples cannot help each other in learning. As an example, contrastive sentences are rare in ASC datasets (see Experiment). Existing research showed that rare and noisy examples are seldom optimized at the early stage of training (e.g., a few epochs) [?]. This is in contrast to its importance as discussed in introduction. As a result, examples should not be treated equally as the mean of example losses as in most training. Given this unwanted behavior of optimization, a natural idea is to detect them and then increase their contribution to the total loss during training.

Since training in supervised learning has access to ground-truth labels, detecting hard examples naturally means to find examples the current model cannot classify correctly. Assuming we have n training examples. Let incorrect (hard) examples to be those with $y_i \neq \hat{y}_i$ for $i \in [1, n]$,

Algorithm 4: ARW Algorithm

Input : \mathcal{D}_{tr} : training set with n examples;

e : maximum number of epochs.

Output: $p_{\theta}(\hat{y}|\cdot, \cdot)$: a trained model.

```

1  $w_{1:n} \leftarrow \frac{1}{n}$ 
2 for  $epoch \in \{1, \dots, e\}$  do
3   for  $(a^b, x^b, y^b, w^b) \in \text{Batchify}(\mathcal{D}_{\text{tr}}, w_{1:n})$  do
4      $l^b \leftarrow \text{CrossEntropy}(p_{\theta}(\hat{y}^b|a^b, x^b), y^b)$ 
5      $L^b \leftarrow \frac{\sum(w^b \cdot l^b)}{\sum w^b}$ 
6     BackProp&ParamUpdate( $L, M$ )
7   end
8    $\hat{y}_{1:n} \leftarrow \arg \max p_{\theta}(\hat{y}_{1:n}|a_{1:n}, x_{1:n})$ 
9    $r \leftarrow \frac{\sum_{i=1}^n (w_i \mathbb{I}[y_i \neq \hat{y}_i])}{\sum_{i=1}^n w_i}$ 
10   $\alpha \leftarrow \log\left(\frac{(1-r)+\epsilon}{r-\epsilon}\right)$ 
11   $w_{1:n} \leftarrow w_{1:n} \exp(\alpha \mathbb{I}[y_{1:n} \neq \hat{y}_{1:n}])$ 
12 end

```

where \hat{y}_i is the prediction of the i -th training example from the current model and y_i is the ground-truth label. Then we associate each example a weight, which decides how much this example contributes to the total loss (e.g., in a batch of optimization). We let $w_{1:n}$ denote the weights associated with n training examples and the total loss L is computed as the weighted sum of the training examples. As deep learning models are typically trained on a batch-by-batch basis, we define the total loss L^b as the loss from a batch. Let l^b be the example-wise losses for examples within a batch. Since a batch is randomly drawn from the training set, we re-normalize the weights w^b for examples in that batch $L^b = \frac{\sum(w^b \cdot l^b)}{\sum w^b}$ to avoid fluctuation caused by randomly drawing examples with weights of different magnitudes.

Given the dynamics of a training process, we aim to design an adaptive weighting function that keeps adjusting the weights. This is because a used-to-be hard example can later be an easy example and vice-versa. At the beginning, we assume an uniform distribution of weights across all training examples $w_{1:n} \leftarrow \frac{1}{n}$. We adjust the weights at the end of training of each epoch because every example has been consumed once. We define an indicator variable $\mathbb{I}[y_i \neq \hat{y}_i]$ to pick the incorrect (hard) examples and estimate the overall weighted error rate $r \in [0, 1]$ to detect whether the current model tends to make more mistakes or not. Note that the reason for using the weighted error rate instead of just the error rate is that the weighted error rate reflects the hardness on optimizing hard examples instead of simply example-level errors. We will detail the formula in the next subsection. For example, when the weighted error rate is high (e.g., > 0.5), instead of increasing the weights for incorrect examples, we probably need to reduce them so as to avoid learning too much noise. Lastly, the weight adjustment for incorrect

examples is determined by the (correct-versus-incorrect) ratio $(\frac{(1-r)+\epsilon}{r-\epsilon})$. So when this value is larger than 1, multiply it to increase the weights; otherwise to decrease the weights. Here we introduce a weight assignment factor ϵ , which is a hyperparameter to control whether the model should favor even more weights (e.g., $\epsilon > 0$) or not (e.g., $\epsilon < 0$).

ARW Algorithm

The proposed ARW algorithm is shown in Algorithm ?? . In Line 1, it initializes the weights of all training examples uniformly. Lines 2-12 pass through the training data epoch-by-epoch and update the example weights at the end of each epoch. Specifically, Line 3 retrieves one randomly sampled batch of aspects a^b , sentences x^b , polarity labels y^b and their (current) corresponding weights w^b . Line 4 makes a forward pass on aspects and sentences $p_\theta(\hat{y}|a^b, x^b)$. Then we compute example-wise loss l^b for each training example in the batch. Line 5 computes the weighted loss and re-normalize these weights throughout the batch to get the total loss L^b . Line 6 does normal backpropagation and parameter updating as in ordinary neural networks training. Line 8 gets the prediction on the training set. Line 9 first discovers the hard examples represented by an indicator variable $\mathbb{I}[y_i \neq \hat{y}_i]$. It then computes the weighted error rate. Line 10 computes the log of the correct-incorrect ratio. $\alpha > 0$ indicates increasing the weights and $\alpha < 0$ means decreasing the weights. Lastly, in Line 11, we only adjust the weights via the indicator variable $\mathbb{I}[y_{1:n} \neq \hat{y}_{1:n}]$ since the weights of correctly classified (easy) examples are always multiply by 1. As a result, Algorithm ?? keeps track of the weights $w_{1:n}$ for all training examples and always focuses on adjusting weights of incorrect examples from contrastive sentences. We also perform a normal validation process after each epoch (omitted in the Algorithm ?? for brevity).

Experiment

Dataset

We adopt the SemEval 2014 Task 4¹ datasets, which contain two domains: *laptop* and *restaurant*. The statistics are shown in ?? . In addition to the *Full Testing Set*, we further form a *Contrastive Test Set* to specifically test aspect-level sentiments. The contrastive test set of laptop is augmented with extra annotated examples from Amazon laptop reviews to ensure enough testing examples.

Results of BERT

We first discuss the results of post-training BERT from Chapter ?? . We have the following state-of-the-art baseline:

MGAN [?] reaches the state-of-the-art ASC on SemEval 2014 task 4. We compute both accuracy and Macro-F1 over 3 classes of polarities, where Macro-F1 is the major metric as the imbalanced classes introduce biases on accuracy. To be consistent with existing research [?], examples belonging to the *conflict* polarity are dropped due to a very small number of examples.

ASC, we observed that large-scale annotated MRC data is very useful. We suspect the reason is that ASC can be interpreted as a special MRC problem, where all questions are about the polarity of a given aspect. MRC training data may help BERT to understand the input format of ASC given their closer input formulation. Again, domain knowledge post-training also helps ASC. ASC tends to have more errors as the decision boundary between the negative and neutral examples is unclear (e.g., even annotators may not sure whether the reviewer shows

¹<http://alt.qcri.org/semeval2014/task4>

	Laptop	Restaurant
Training		
#Sentence	3045	2000
#Aspect	2358	1743
#Positive	987	2164
#Negative	866	805
#Neutral	460	633
#Sent. /w Asp.	1462	1978
#Contrastive Sent.	165	319
%Contrastive Sent.	11.3%	16.1%
Full Testing Set		
#Sentence	800	676
#Aspect	654	622
#Positive	341	728
#Negative	128	196
#Neutral	169	196
#Sent. /w Asp.	411	600
#Contrastive Sent.	38	80
%Contrastive Sent.	9.2%	13.3%
Contrastive Test Set		
#Contrastive Sent.	78	80
#Aspect	203	228
#Positive	72	85
#Negative	71	60
#Neutral	60	83

TABLE IX

Statistics of SemEval14 Task4 with Contrastive sentences

Domain	Laptop		Rest.	
Methods	Acc.	MF1	Acc.	MF1
MGAN [?]	76.21	71.42	81.49	71.48
BERT	75.29	71.91	81.54	71.94
BERT-DK	77.01	73.72	83.96	75.45
BERT-MRC	77.19	74.1	83.17	74.97
BERT-PT	78.07	75.08	84.95	76.96

TABLE X

ASC in Accuracy and Macro-F1(MF1).

no opinion or slight negative opinion when mentioning an aspect). Also, BERT-PT has the problem of dealing with one sentence with two opposite opinions (“The screen is good but not for windows.”). We believe that such training examples are rare.

Next we discuss the results of ARW.

Baselines for ARW

We evaluate all baselines on both accuracy (Acc.) and macro F1 (MF1) and adopt the following baselines: RAM [?]¹, AOA [?], MGAN [?], TNET [?], BERT-DK [?]¹. For the last model, we further challenge it by removing the aspects from the testing examples as there is

¹The first 4 baselines are adopted from <https://github.com/songyouwei/ABSA-PyTorch>.

¹<https://github.com/howardhsu/BERT-for-RRC-ABSA>

no architecture change in doing so. In this way, we want to test the performance of BERT-DK under a setting with no access to aspects.

We use BERT-DK as a base model to compare the following re-weighting schemes.

+Manual Re-weighting. This baseline first counts the number of training examples C_c that are contrastive sentences and gives these examples/sentences the weight $(n - C_c)$ and other examples the weight C_c , where n is the total number of training examples. These weights are re-normalized within a batch. Note that we also experimented with a number of other manual weighting schemes and this method does the best.

+Focal Loss. We compute weights as $(1 - p)^\gamma$ [?], where p is the probability of prediction on the ground-truth label (from softmax) and γ is a hyper-parameter. We use $\gamma = 2.0$ from the original paper that works best for ASC, too.

+ARW. This is the proposed training algorithm. This method discovers all incorrect examples, which include examples from the contrastive sentences set and other examples. We search $\epsilon \in \{-0.2, -0.1, -0.05, 0.0, 0.05, 0.1, 0.2\}$ and use $\epsilon = -0.05$ for results.

+ARW w/ manual initial weighting. We further investigate the use of +Manual Re-weighting’s weighting function as the initial weights and then use ARW for adaptive re-weighting.

Hyper-parameters

For all methods, we use Adam optimizer and set the learning rate to 3e-5. The batch size is set as 32. To perform model selection, we hold out 150 examples from the training set as the validation set. We set the maximum epochs to 12. Lastly, all results are averaged over 10 runs.

Result Analysis

	Laptop		Rest.	
	Acc.	MF1	Acc.	MF1
RAM [?]				
on Full Test Set	74.49	71.35	80.23	70.8
on Contrastive Test Set	41.87	38.65	52.19	55.19
AOA [?]				
on Full Test Set	74.5	-	81.2	-
on Contrastive Test Set	42.86	33.53	42.98	33.66
MGAN [?]				
on Full Test Set	75.39	72.47	81.25	71.94
on Contrastive Test Set	46.8	43.38	53.95	57.64
TNET [?]				
on Full Test Set	76.54	71.75	80.69	71.27
on Contrastive Test Set	49.75	49.86	56.58	58.05
BERT-DK [?]				
on Full Test Set	76.9	73.65	84.21	76.2
on Full Test Set w/o aspect	<u>76.0</u>	<u>73.05</u>	<u>80.03</u>	<u>72.95</u>
on Contrastive Test Set	51.13	50.04	65.53	66.92
BERT-DK	Acc.	MF1	Acc.	MF1
+ Manual Re-weighting				
on Full Test Set	75.41	71.99	84.36	76.35
on Contrastive Test Set	53.45	52.76	68.03	69.51
+ Focal Loss [?]				
on Full Test Set	76.33	73.24	84.57	76.56
on Contrastive Test Set	51.48	50.43	66.4	67.14
+ ARW w/ manual initial weighting				
on Full Test Set	70.08	65.89	84.48	77.41
on Contrastive Test Set	55.37	54.68	75.31	75.81
+ ARW				
on Full Test Set	77.23	73.81	85.35	78.46
on Contrastive Test Set	61.08	60.34	71.84	72.66

TABLE XI

Results of ARW on ASC

From ??, we can see that all existing ASC baselines have significant drops on contrastive test set for both Accuracy (Acc.) and F1 score, indicating the hardness of this testing set. When the aspects are dropped from the input (*on Full Test Set w/o aspect*), the BERT-DK ASC classifier dropped a little and still comparable to other baselines on the full test set.

BERT-DK + ARW outperforms other baselines mostly. If we compare it with *BERT-DK*, it gives nearly 10% of improvement for laptop and 6% for restaurant on the contrastive test set. After examining the errors, we notice that contrastive sentences with *neutral* polarity is harder. This is because there may be no transition, but just one aspect with *pos/neg* opinion and one aspect with no opinion (*neutral*). Some implicit transition word is also hard to learn (e.g., “The screen is great and I can live with the keyboard’s slightly smaller size.”). Manual re-weighting improves the performance on laptop and restaurant by about 3% for the contrastive test sets. *BERT-DK + ARW w/ manual initial weighting* has the best performance on the contrastive test set but not laptop. Focal loss does not perform well. The reason is that the “soft” probability may not explicitly distinguish whether the model is making a mistake on an example or not.

6.2 Complementary Entity Recognition

E-commerce websites (e.g., Amazon.com) contain a huge amount of products reviews and most existing works of sentiment analysis [?] (or opinion mining) on reviews focus on extracting opinion targets (aspects or features) of the reviewed product and the associated opinions [?, ?, ?] (e.g., extract “battery” and *pos* from “It has a good battery”). Besides features about the reviewed product itself (e.g., “battery” or “screen”), one important feature is whether the reviewed product is compatible/incompatible with another product. We call the reviewed

product *target entity* and the other product *complementary entity*. A pair of a target entity and its complementary entity forms a *complementary relation*. They may work together to fulfill some shared functionalities. So, they are usually co-purchased. For example, in ??, we assume there are some reviews of several accessories (on the left) talking about compatibility issues. We consider these accessories as the target entities and they have some complementary entities (on the right side) mentioned in reviews. The target entities are one *micro SD card*, one *tablet stand* and one *mouse*; the complementary entities are one *Nikon DSLR*, one *iPhone*, one *Samsung Galaxy S6* and one *MS Surface Pro*. An arrow pointing from a target entity to a complementary entity indicates that they have a complementary relation and shall work together. For example, the *micro SD card* can help the *Samsung Galaxy S6* to expand its memory capacity. Knowing these complementary entities is important because compatible products are preferred over incompatible ones. Thus, recognizing complementary entities is an important task in text mining.

Problem Statement: we study the problem of finding complementary entities from texts (e.g., extracting “Samsung Galaxy S6” from “It works with my Samsung Galaxy S6”). We coin this problem as Complementary Entity Recognition (CER). We observe that compatibility issues are more frequently discussed in reviews of electronics accessories, so we choose reviews of accessories for experiments. To the best of our knowledge, accessory reviews are not well studied before. This section focuses on two lifelong learning settings on CER: we first focus on an unsupervised method that using knowledge expansion over large amount of unlabeled reviews;

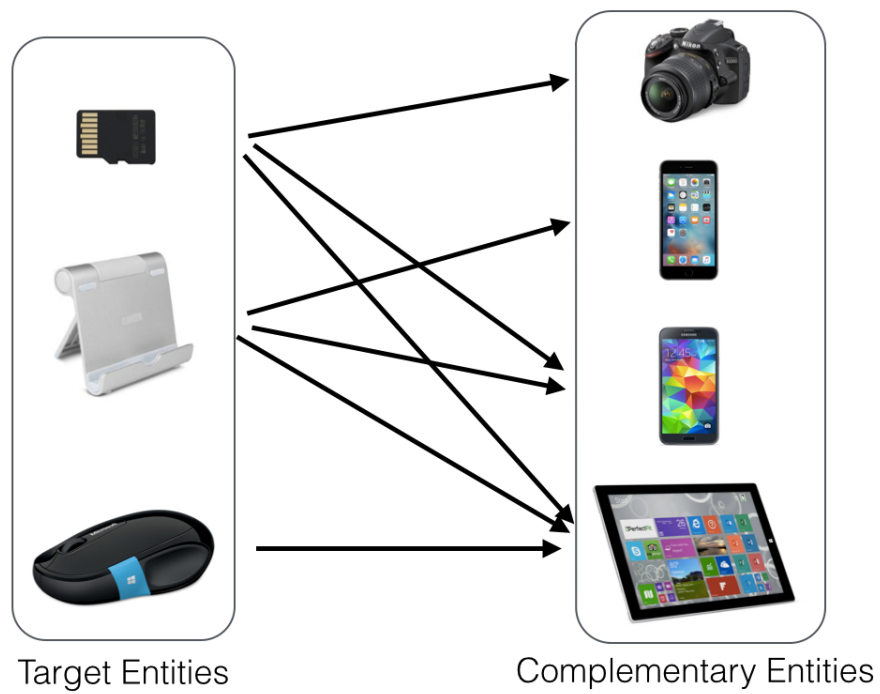


Figure 9. Target entities, their complementary entities and complementary relations.

then we switch to a supervised method of CER by collecting key-value pair of knowledge to enhance the performance of CER.

6.2.1 – Knowledge Expansion on Large Unlabeled Product Reviews

The proposed CER problem has a few challenges and also provides more research opportunities:

- To the best of our knowledge, the linguistic patterns of complementary relations are not studied in computer science. There is no largely annotated dataset for supervised methods. We propose an unsupervised method, which does not require any labeled data to solve this problem (we only annotate a small amount of data for evaluation purposes).
- Similar to the aspect (feature) extraction problem in reviews [?], CER is also a domain-specific problem. We leverage domain knowledge to help the unsupervised method to adapt to different products. This novel product domain knowledge is expanded using a few seed words on a large amount of unlabeled reviews under the same category as the target entity. The idea of using reviews under the same category as the target entity is that the number of reviews for one target entity is small. We observe that products (target entities) under the same category share similar complementary entities (i.e., two different *micro SD cards* may share complementary entities like *phone* or *tablet*). So the domain knowledge expanded on reviews from the same category is larger than that on reviews from a single target entity. Therefore, there is almost no labor-intensive effort to get domain knowledge. Our domain knowledge contains candidate complementary entities and domain-specific verbs.

- Although the problem may be closely related to the well-known Named Entity Recognition (NER) problem on surface [?], recognizing a complementary entity requires more contexts. For example, given a review for a *micro SD card*, we should not treat “Samsung Galaxy S6” in “Samsung Galaxy S6 is great” as a complementary entity. However, we should consider the same entity in “It works with my Samsung Galaxy S6” as a complementary entity. The domain knowledge contains domain-specific verbs, which greatly help to detect the contexts of complementary entities.
- We further notice that some linguistic patterns of complementary relations are similar to other extraction patterns (e.g., patterns for aspect extraction). Candidate complementary entities in the domain knowledge can help to filter out non-complementary entities extracted by similar patterns.

Related Works

The proposed problem is closely related to product recommender systems that are able to separate substitutes and complements [?, ?]. Zheng et al. [?] first propose to incorporate the concepts of substitutes and complements into recommendation systems by analyzing navigation logs. More specifically, predicting complementary relations is pioneered by McAuley et al. [?]. They utilize topic models and customer purchase information (e.g., the products in the “items also viewed” section and the “items also bought” section of a product page) to predict category-level substitutes and complements. However, we observe that purchase information generated by the unknown algorithm from Amazon.com tends to be noisy and inaccurate for complementary entities since co-purchased products may not be complementary to each other.

We demonstrate that their predictions are non-complementary entities for the products that we use for experiments in Section ?? . Also, category-level predictions are not good enough for specific pairs of products (i.e., *DSLR lens* and *webcam* are not complements). Furthermore, their predictions do not provide information about incompatible entities, which are valuable buying warnings for customers. Thus, fine-grained extraction of complementary entities from reviews that express firsthand user experience is important. To the best of our knowledge, the linguistic patterns of complementary relations are not studied in computer science.

The proposed problem is closely related to aspect extraction [?, ?, ?, ?], which is to extract product features from reviews. More specifically, extracting comparable products (i.e, one type of substitutes, or products that can replace each other) from reviews is studied by Jindal and Liu [?]. Recently, dependency paths [?] are used for aspect extraction [?, ?]. Shu et al. [?] use unsupervised graph labeling method to identify entities from opinion targets. However, since aspects are mostly context independent and the same aspect may appear multiple times, aspect extraction in general does not need to extract each occurrence of an aspect (as long as the same aspect can be extracted at least once). In contrast, the CER problem is context dependent and many complementary entities are infrequent (i.e., *Samsung Galaxy S6* is infrequent than the aspect *price*). We use dependency paths to accurately identify each occurrence of complementary entities. Since extracting each complementary entity can be inaccurate, we further utilize domain knowledge to improve the precision.

CER is closely related to Named Entity Recognition (NER) [?] and relation extraction [?]. NER methods utilize annotated data to train a sequential tagger [?, ?]. However, our task is

totally different from NER since we care about the context of a complementary entity and many complementary entities are not named entities (e.g., *phone*). CER is also different from relation extraction $[?, ?, ?, ?]$, which assumes that two entities are identified in advance. In reviews, the target entity is unfortunately missing in many cases (i.e., “Works with my phone”). The proposed method only cares about the relation context of a complementary entity rather than a full relation.

Term Definitions

Our problem is to recognize entities that functionally complement to the reviewed product. There are several definitions involved in this problem.

Target Entity: We define *target entity* e_T as the reviewed product.

We do not extract target entities from reviews but assume that the target entity can be retrieved from the meta data (product title) of reviews. This is because many mentions of the target entity are co-referenced or implicitly assumed in reviews. For example, if the reviewed product is a *tablet stand*, “It works with my Samsung Galaxy S6” uses “It” to refer to the target entity *tablet stand*; “Works well with Samsung Galaxy S6” completely omits the target entity.

Complementary Entity: Given a set of reviews R_T of a target entity e_T , a *complementary entity* e_C is an entity mentioned in reviews that are functionally complementary to the target entity e_T . A target entity has a set of complementary entities: $e_C \in E_C$.

A complementary entity can either be a single noun (e.g., *iPhone*) or a noun phrase (e.g., *Samsung Galaxy S6*). There are two types of complementary entities: a *named entity* or a *general entity*. A named entity is usually a specific product name containing a brand name and a model

name (e.g., *Samsung Galaxy S6* or *Apple iPhone*). A general entity (e.g., *phone* or *tablet*) represents a set of named entities. General entities are informative. For example, in a review of a *tablet stand*, “phone” in “It also works with my phone” is a good assurance for phone owners who want to use this *tablet stand* as a *phone stand*.

Complementary Relation: Each complementary entity $e_C \in E_C$ forms a *complementary relation* (e_T, e_C) with the target entity e_T .

Complementary Entity Recognition: Given a set of reviews R_T for a target entity e_T , the problem of Complementary Entity Recognition (CER) is to identify a set of complementary entities E_C , where each $e_C \in E_C$ has a complementary relation (e_T, e_C) with the target entity e_T .

We do not extract an entity without a complementary context (e.g., “Samsung Galaxy S6” in “Samsung Galaxy S6 is great”, even though *Samsung Galaxy S6* may be a complementary entity).

Domain: We assume that every target entity e_T belongs to a pre-defined *domain* (or *category*) $Dom(e_T) = d \in D$. A *review corpora* $R^{Dom(e_T)}$ is all reviews under the same category as the target entity e_T .

Domain Knowledge: Each domain d has its own *domain knowledge*. We consider two types of domain knowledge: *candidate complementary entity* $e_C^d \in E_C^d$ and *domain-specific verb* $v^d \in V^d$. All target entities e_T under the same domain share the same domain knowledge.

Basic Ideas

The basic idea of the proposed method is to use dependency paths to identify complementary entities. Due to different linguistic patterns, these dependency paths may have different

performance on extraction. Some dependency paths may have high precision but low recall and vice versa. To ensure the quality of extraction, high precision dependency paths are preferred. The idea of using domain knowledge is that high precision dependency paths can expand high quality (precision) domain knowledge on a large amount of unlabeled reviews, which in turn helps low precision but high recall dependency paths to improve their precisions. In the end, the domain knowledge serves as a filter to remove noises in low precision paths. This framework can potentially be generalized to any extraction task when a large amount of unlabeled data is accessible. We describe the proposed method in the following two parts:

Basic Entity Recognition: We analyze the linguistic patterns and leverage multiple dependency paths to recognize complementary entities. The major goal of the basic entity recognition is to get high recall because each complementary entity can be infrequent and we care about each mention of a complementary entity. Due to similarity with other noisy patterns, these paths tend to have a low precision.

Recognition via Domain Knowledge Expansion: We expand the domain knowledge on a large amount of unlabeled reviews using a set of high precision dependency paths to compensate for the low precision (noisy) dependency paths. First, we extract candidate complementary entities for each domain using only verbs *fit* and *work*. Then we use the extracted candidate complementary entities to induce domain-specific verbs (e.g., *insert* for *micro SD card*, or *hold* for *tablet stand*). Finally, we integrate these two types of domain knowledge into the dependency paths of basic entity recognition to improve the precision.

Dependency Paths

In this subsection, we briefly review the concepts used by dependency paths. We further describe how to match a dependency path with a sentence.

Dependency Relation: A *dependency relation* is a typed relation between two words in a sentence with the following format of *attributes*:

$$type(gov, govidx, govpos, dep, depidx, deppos),$$

where *type* is the type of a dependency relation, *gov* is the *governor word*, *govidx* is the index (position) of the *gov* word in the sentence, *govpos* is the POS (Part-Of-Speech) tag of the *gov* word, *dep* is the *dependent word*, *depidx* is the index of the *dep* word in the sentence and *deppos* is the POS tag of the *dep* word. The *direction* of a dependency relation is from the *gov* word to the *dep* word.

A sentence can be parsed into a set of dependency relations through dependency parsing¹ [?, ?]. For example, “It works with my phone” can be parsed into a set of dependency relations in ??, which is further illustrated in ??.

Dependency Segment: A *dependency segment* is an abstract form of a dependency relation. A dependency segment has the following format of attributes, which is similar to a dependency relation:

$$(src, srcpos) \xrightarrow{pathtype} (dst, dstpos),$$

¹We utilize Stanford CoreNLP as the tool for dependency parsing.

ID	Dependency Relation	Syntactic Dependency Relation Type	Explanation
1	<i>nsubj(works, 2, VBZ, It, 1, PRP)</i>	<i>nsubj</i> : nominal subject	Relate the 1st word “It” to the 2nd word “works”
2	<i>root(ROOT, 0, None, works, 2, VBZ)</i>	<i>root</i> : root relation	Relate the 2nd word “works” to the virtual word ROOT
3	<i>case(phone, 5, NN, with, 3, IN)</i>	<i>case</i> : case-marking	Relate the 3rd word “with” to the 5th word “phone”
4	<i>nmod:poss(phone, 5, NN, my, 4, PRP\$)</i>	<i>nmod:poss</i> : possessive nominal modifier	Relate the 4th word “my” to the 5th word “phone”
5	<i>nmod:with(works, 2, VBZ, phone, 5, NN)</i>	<i>nmod:with</i> : nominal modifier via with	Relate the 5th word “phone” to the 2nd word “works”

TABLE XII

Dependency relations.

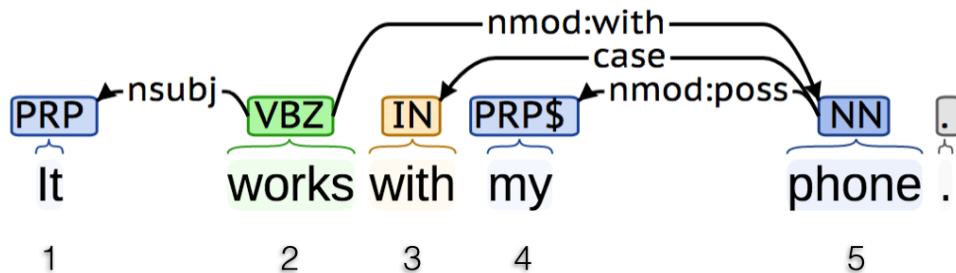


Figure 10. Visualization of dependency relations.

where *src* is the *source word*, *srcpos* is the POS tag of the source word, *dst* is the *destination word*, *dstpos* is the POS tag of the destination word and *pathtype* is the *dependency type* of the segment. Similarly, the *direction* of an segment is from the *src* word to the *dst* word.

Dependency Segment Matching: A dependency segment can have a *dependency segment matching* with a dependency relation. To have such a match, we must ensure that attributes *src*, *srcpos*, *dst*, *dstpos* and *pathtype* in an segment match attributes *gov*, *govpos*, *dep*, *deppos* and *type* in a dependency relation respectively. So the direction of a dependency segment also matches the direction of a dependency relation.

To allow a matching to cover more specific dependency relations, we further define a set of rules when matching the attributes, which are summarized in ?? . Please note that we finally want to extract the complementary entity covered by tag *CETT*. Other kinds of attributes are defined to make the dependency paths more compact.

The segment:

$$("work", V) \xrightarrow{nmod:cmprel} (CETT, N) \quad (6.2)$$

can match the dependency relation 5 in ??. This is because source word "*work*" is the lemmatized governor word "*works*"; *V* covers *VBZ*; *N* covers *NN*; and *nmod:cmprel* covers dependency type *nmod:with*. Since the tag *CETT* as the destination word in the segment covers the dependent word "*phone*" in dependency relation 5, this segment indicates "*phone*" is a possible complementary entity.

Dependency Path: A *dependency path* is a finite sequence of dependency segments connected by a sequence of *src/dst* attributes.

Path Attr.	Value	Rel. Attr.	Value
<i>src/dst</i>	[<i>lem. word</i>]	<i>gov/dep</i>	[<i>specific form</i>]
<i>src/dst</i>	* / <i>CETT</i>	<i>gov/dep</i>	[<i>any word</i>]
<i>srcpos/dstpos</i>	<i>N</i>	<i>gov/dep</i>	<i>NN NNP NNPS NP</i>
<i>srcpos/dstpos</i>	<i>V</i>	<i>gov/dep</i>	<i>VB VBD VBG</i> <i>VCN VBP VBZ</i>
<i>srcpos/dstpos</i>	<i>J</i>	<i>gov/dep</i>	<i>JJ JJR JJS</i>
<i>pathtype</i>	<i>nmod:cmpr</i>	<i>type</i>	<i>nmod:with nmod:for</i> <i>nmod:in nmod:on</i> <i>nmod:to nmod:inside</i> <i>nmod:into</i>

TABLE XIII

Rules of matching attributes of dependency segments and dependency relations

Given different directions of 2 adjacent dependency segments, there are 4 possible types of a connection: $\rightarrow\rightarrow$, $\rightarrow\leftarrow$, $\leftarrow\rightarrow$ and $\leftarrow\leftarrow$.

Dependency Path Matching: A procedure of *dependency path matching* is specified as the following: when matching a dependency path with a sentence, we first check whether there are at least one dependency relations for each segment. If so, we further check whether the two directions of dependency segments for each connection match the directions of two corresponding dependency relations and whether the connected governor/dependent words from two matched dependency relations have the same index (they are the same word in the original sentence).

Finally, after we have a successful dependency path matching, we extract the *gov/dep* in dependency relations labeled as *CETT* by the dependency path.

The following path

$$(*, V) \xrightarrow{nmod:with} (CETT, N) \xrightarrow{nmod:poss} ("my", PRP\$) \quad (6.3)$$

can match the sentence "It works with my phone" since the two segments match dependency relation 5 and 4 respectively. Here wildcard * matches word "works". Further the dependent word "phone" of the dependency relation 5 have the same index (the 5th word described in ??) as the governor word of the dependency relation 4.

Basic Entity Recognition

Syntactic Patterns of Complementary Relation

There are many ways to mention complementary relations in reviews. Complementary relations are usually expressed with or without a preposition. In the first case, the preposition is used to bring out the complementary entity and is usually associated with a verb, a noun, an adjective or a determiner; in the second case without a preposition, reviewers only use transitive verbs to bring out the complementary entities. The verbs used in both cases can either be general verbs such as “fit” or “work”, or domain-specific verbs such as “insert” for *micro SD card* or “hold” for *tablet stand*. Complementary relations can also be expressed through nouns, adjectives or determiners. We discuss the syntactic patterns of complementary relations as the following:

Verb+Prep: The majority of complementary relations are expressed through a verb followed by a preposition. For example, “It works with my phone” falls into this pattern, where the verb “works” and the preposition “with” work together to relate the pronoun “It” to “phone”. The target entity can appear in this pattern either as the subject or as the object of the verb. In the previous example, subject “It” indicates the target entity. In “I insert the card into my phone”, “the card” is the object of the verb “insert”. The target entity can also be implicitly assumed as in “Works with my phone.”

Noun+Prep: Complementary relation can be expressed through nouns. Those nouns typically have opinions. For example, “No problem” in “No problem with my phone” has a positive opinion on “phone”.

Adjective+Prep: Complementary relation can also be expressed through adjectives with prepositions. For example, the adjective “useful” together with the preposition “for” in “It is useful for my phone” expresses a positive opinion on a complementary relation.

Path Type	ID	Path	Example
Verb+Prep	1	$(verb, V) \xrightarrow{nmod:cmprel} (CETT, N)$	It works/V with my phone[CETT].
Noun+Prep	2	$(*, N) \xrightarrow{nmod:cmprel} (CETT, N)$	No problem/N with my phone[CETT].
Adjective+Prep	3	$(*, J) \xrightarrow{nmod:cmprel} (CETT, N)$	It is compatible/J with my phone[CETT].
Determiner+Prep	4	$(*, DT) \xrightarrow{nmod:cmprel} (CETT, N)$	I use this/DT for my phone[CETT].
Verb	5	$(verb, V) \xrightarrow{dobj} (CETT, N) \xrightarrow{nmod:poss} ("my", PRP\$)$	It fits my phone[CETT].
	6	$(("it"/"this", DT) \xleftarrow{nsubj} (verb, V) \xrightarrow{dobj} (CETT, N))$	It fits iPhone[CETT].

TABLE XIV

Summary of dependency paths

Determiner+Prep: Determiner “this” in “I use this for my phone” refers to the target entity. It is associated with the preposition “for” in dependency parsing.

Verb: Complementary relation can be expressed only through verbs without using any preposition. For example, in “It fits my phone”, subject “It” is related to the object “phone” via only the transitive verb “fits”. This pattern has low precision on extraction since almost every sentence has a subject, a verb and an object. We improve the precision of this pattern using the domain knowledge.

Dependency Paths for Extraction

According to the discussed patterns, we implement dependency paths, which are summarized in ???. For patterns with a preposition (e.g., Verb+Prep, Noun+Prep, Adjective+Prep,

Determiner+Prep), we use dependency type *nmod:cmpr* to encode all prepositions, because *cmpr* represents *with, for, in, on, to, inside* and *into*. Then type *nmod:cmpr* can relate verbs, nouns, adjectives or determiners to the complementary entities. As shown in Example 1 and 2, *nmod:cmpr* can match *nmod:with* and relates the verb “works” to the complementary entity “phone” for dependency relation 5 in ??. This path is defined as Path 1 in ??.

For pattern Verb, we use dependency type *dobj* to relate a verb to the complementary entity. Since this pattern tends to have low precision, we further constrain the pattern by connecting a *nsubj* relation or a *nmod:poss* relation, as described in Path 5 or Path 6 respectively in ??. For example, “It fits iPhone” has the following two dependency relations: *nsubj*(“fits”, VBZ, 2, “It”, PRP, 1) and *dobj*(“fits”, VBZ, 2, “iPhone”, NNP, 3). Path 6 can match these two dependency relations separately and then check the two “fits”s have the same index 2 in these two dependency relations. So “iPhone” tagged as *CETT* can be extracted.

Finally, these paths may appear multiple times in a sentence. So multiple complementary entities in a sentence can be extracted. For example, “It works with my phone, laptop and tablet” has 3 complementary entities. It has the following 3 dependency relations: *nmod:with*(“works”, VBZ, 2, “phone”, NN, 5), *nmod:with*(“works”, VBZ, 2, “laptop”, NN, 7) and *nmod:with*(“works”, VBZ, 2, “tablet”, NN, 9). So Path 1 can have 3 matches to extract “phone”, “laptop” and “tablet”.

Please note that ?? does not list all possible dependency paths. For example, complementary entities can also serve as the subject of a sentence: “My phone likes this card”. We simply demonstrate typical dependency paths and new dependency paths can be easily added into the system to improve the recall.

Post-processing

Since a dependency relation can only handle the relation between two individual words, a complementary entity (labeled by *CETT*) extracted from Subsection B can only contain a single word. In reality, many complementary entities are named entities that represent product names such as “Samsung/NNP Galaxy/NNP S6/NNP”. Dependency relations usually pick a single noun (e.g., “S6”) and relate it with other words in the phrase via other dependency relations (e.g., type *compound*). We use the regular expression pattern $\langle N \rangle \langle N|CD \rangle^*$ to chunk a single noun into a noun phrase¹. This pattern means one noun (*N*) followed by 0 to many nouns or numbers. Nouns and numbers (model number) are typical POS tags of words in a product name.

Recognition via Domain Knowledge Expansion

Using the paths defined tends to have low precision (noisy) of extractions since syntactic patterns may not distinguish a complementary relation from other relations. For example, Path 6 can match any sentence with type *dobj*. A sentence like “It has fast speed” uses type *dobj* to bring out “speed”, which is a feature of the target entity itself. To improve the precision, we incorporate category-level domain knowledge (candidate complementary entities and domain-specific verbs) into the extraction process. Those knowledge can help to constrain possible choices of *CETT* and *verb* in dependency paths.

We mine domain knowledge from a large amount of unlabeled reviews under the same category. We get those two types of domain knowledge by bootstrapping them only from general

¹We implement the noun phrase chunker via NLTK: <http://www.nltk.org/>

Type	ID	Path	Example
CCE	7	$(\text{"fit"/"work"}, V) \xrightarrow{\text{nmod:cmprcl}} (CETT, N) \xrightarrow{\text{nmod:poss}} (\text{"my"}, PRP\$)$	It works with my phone[CETT].
DSV	8	$(\text{verb}, V) \xrightarrow{\text{nmod:cmprcl}} (CETT, N) \xrightarrow{\text{nmod:poss}} (\text{"my"}, PRP\$)$	I insert[verb] the card into my phone[CETT].
	9	$(\text{"this"}, DT) \xleftarrow{\text{dobj}} (\text{verb}, V) \xrightarrow{\text{nmod:poss}} (\text{"my"}, PRP\$)$	This holds[verb] my phone[CETT] well.

TABLE XV

Summary of dependency paths for extraction

verb *fit* and *work*. We randomly select 6000 reviews for each domain (category) to accumulate enough knowledge (knowledge from reviews of a single target entity may not be sufficient). One important observation is that products under the same domain share similar complementary entities and use similar domain-specific verbs. For example, all *micro SD cards* have *camera*, *camcorder*, *phone*, *tablet*, etc. as their complementary entities and use verbs like *insert* to express complementary relations. But these complementary entities and domain-specific verbs do not make sense for category *tablet stand*. To ensure the quality of the domain knowledge, we utilize several high precision dependency paths. These paths have low recall, so applying them directly to the testing reviews of the target entity has poor performance. High precision paths can leverage big data to improve the precision of other paths.

Exploiting Candidate Complementary Entities

Knowing category-level candidate complementary entities is important for extracting complementary entities for a target entity under that category. For example, the sentences “It works

in iPhone”, “It works in practice” and “It works in 4G” have similar dependency relations *nmod:in*(“works”, VBZ, 2, “iPhone”/“practice”/“4G”, NN, 4). But only the first sentence has a mention of a complementary entity; the second sentence has a common phrase “in practice” with a preposition “in”; the third sentence expresses an aspect of the target entity. The key idea is that if we know that *iPhone* is a potential complementary entity under the category of *micro SD card* and “practice” and “4G” are not, we are confident to extract “iPhone” as a complementary entity.

We use Path 7 to extract candidate complementary entities as described in ?? . It has high precision because given a verb like “fit” or “work”, a preposition that relates to another entity and the possessive pronoun “my”, we are confident that the entity modified by “my” is a complementary entity. Lastly, all extracted complementary entities are stored as domain knowledge for each category.

Exploiting Domain-Specific Verbs

Similarly, knowing category level domain-specific verbs is also important. This is because each category of products may have its own domain verbs to describe a complementary relation. If we only use general verbs (e.g., *fit* and *work*), we may miss many complementary entities that are brought out via domain-specific verbs (e.g., *insert* for *micro SD card* or *hold* for *tablet stand*), and this leads to poor recall rate. In contrast, if we consider all verbs into the paths without distinguishing them, we may bring in lots of noisy false positives. For example, if the target entity is a *tablet stand*, “It holds my tablet” and “It prevents my finger going numb” have similar dependency relations (*dobj*(“holds”/“prevents”, VB, 2, “tablet”/“finger”, NN, 4)). The former

one has a complementary entity since “holds” indicates a functionality that a *tablet stand* can have. The latter one does not have one. So if we know *hold* (we lemmatize the verbs) is a domain-specific verb under the category of *tablet stand* and “prevents” is not, we are more confident to get rid of the latter one. Therefore, we design dependency paths to extract high quality domain-specific verbs. This time, candidate complementary entities can help to identify whether a verb has a semantic meaning of *complement*. So we leverage the domain knowledge extracted in Subsection A to extract domain-specific verbs. In the end, we get domain-specific verbs from general seed verbs *fit* and *work*.

Path 8 and 9 in ?? are used to get verbs in pattern Verb+Prep and Verb respectively. These paths also have high precision because given possessive modifier “my” modifying a complementary entity or determiner “this” indicating a target entity it is almost certain that the verb between them indicates a complementary relation. Then we keep the words tagged by *verb* more than once (to reduce the noise) and store them as domain knowledge. Please note that we do not further expand domain knowledge to avoid reducing the quality of domain knowledge.

Entity Extraction using Domain Knowledge

We use the same dependency paths to perform extraction. But this time we utilize the knowledge of candidate complementary entities and domain-specific verbs under the same category as the target entity. During matching, we look up candidate complementary entities and domain-specific verbs for tags *CETT* and *verb* respectively. But there is an exception for *CETT*. Since a named entity as a complementary entity may rarely appear again in a large amount of reviews, we ignore such a check if the word covered by *CETT* can be expanded into a noun

Product	Revs.	Sents.	Rel.	Revs. w/ Rels.
Stylus	216	892	165	116
Micro SD Card	216	802	193	149
Mouse	216	1158	221	136
Tablet Stand	218	784	154	115
Keypad	114	618	113	76
Notebook Sleeve	109	405	125	84
Compact Flash	113	347	99	82

TABLE XVI

Statistics of the CER dataset

phrase (more than 1 word) during post-processing. Furthermore, we notice that knowledge about target entities is also useful. For example, “I insert this card into my phone” uses “this” to bring out the target entities, which may indicate nearby entities are complementary entities. However, knowledge about a target entity may be expanded on reviews of that target entity (test data) rather than reviews under the same category because target entities are not the same under the same category.

Experimental Results

Dataset

Product	NP Chunker			OpenNLP			UIUC NER			CRF			Sceptre		
	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	$\mathcal{P}@25$		
Stylus	0.21	0.96	0.35	0.03	0.13	0.05	0.41	0.21	0.28	0.69	0.46	0.55	0.04		
Micro SD Card	0.26	0.99	0.41	0.04	0.14	0.07	0.34	0.39	0.36	0.85	0.47	0.6	0.16		
Mouse	0.22	0.98	0.36	0.1	0.4	0.15	0.3	0.26	0.28	0.65	0.4	0.49	0.16		
Tablet Stand	0.25	0.97	0.4	0.06	0.21	0.09	0.82	0.16	0.27	0.73	0.44	0.55	0.04		
Keypad	0.2	0.98	0.33	0.05	0.21	0.08	0.4	0.25	0.31	0.63	0.24	0.35	0.04		
Notebook Sleeve	0.33	0.97	0.5	0.05	0.1	0.06	0.79	0.26	0.4	0.64	0.26	0.37	0.0		
Compact Flash	0.3	0.95	0.46	0.06	0.16	0.09	0.56	0.36	0.44	0.77	0.33	0.46	0.04		
	"My" Entity			CER			CER1K+			CER3K+			CER6K+		
	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1
Stylus	0.5	0.54	0.52	0.35	0.89	0.5	0.89	0.64	0.75	0.88	0.69	0.77	0.86	0.71	0.78
Micro SD Card	0.63	0.51	0.56	0.39	0.8	0.52	0.81	0.64	0.71	0.79	0.66	0.72	0.8	0.67	0.73
Mouse	0.54	0.37	0.44	0.35	0.91	0.5	0.69	0.69	0.69	0.66	0.7	0.68	0.66	0.72	0.69
Tablet Stand	0.58	0.43	0.49	0.41	0.84	0.55	0.68	0.39	0.5	0.75	0.69	0.72	0.75	0.72	0.74
Keypad	0.54	0.46	0.5	0.33	0.92	0.49	0.66	0.67	0.66	0.67	0.73	0.7	0.69	0.82	0.75
Notebook Sleeve	0.69	0.38	0.49	0.46	0.71	0.56	0.93	0.5	0.65	0.93	0.65	0.76	0.92	0.66	0.77
Compact Flash	0.75	0.61	0.67	0.46	0.88	0.6	0.86	0.63	0.73	0.86	0.68	0.76	0.85	0.7	0.77

TABLE XVII

Comparison of different methods for CER

Category	1K(s)	3K(s)	6K(s)	Candidate Complementary Entity	Domain-Specific Verbs
Cat:Stylus	1.16	4.53	7.49	ipad 2, tablet, iPhone, Samsung Galaxy 2	scratch, match, press, draw, sketch, sign
Cat:Micro SD Card	1.23	3.67	5.58	laptop, psp, galaxy s4, Galaxy tab	add, insert, plug, transfer, store, stick
Cat:Mouse	1.61	5.1	7.71	Macbook pro, laptop bag, MacBook Air	move, rest, carry, connect, click
Cat:Tablet Stand	1.51	4.08	6.93	Nook, ipad 2, Kindle Fire, Galaxy tab, fire	rest, insert, stand, support, hold, sit
Cat:Keypad	1.25	2.93	6.17	MacBook, MacBook pro, Mac	hook, connect, go, need, use, fit, plug
Cat:Notebook Sleeve	1.11	2.79	5.46	backpack, Macbook pro, Lenovo x220	show, scratch, bring, feel, protect
Cat:Compact Flash	1.49	3.29	6.45	dslr, Canon rebel, Nikon d700	load, pop, format, insert, put

TABLE XVIII

Running time of expanding domain knowledge

We select reviews of 7 products that have frequent mentions of complementary relations from the Amazon review datasets [?]. We choose accessories because compatibility issues are more frequently discussed in accessory reviews. The products are *stylus*, *micro SD card*, *mouse*, *tablet stand*, *keypad*, *notebook sleeve* and *compact flash*. We select nearly 220 reviews for the first 4 products and 110 reviews for the last 3 products. We select 50% reviews of the first 4 products as the training data for Conditional Random Field (CRF) (one supervised baseline). The remaining reviews of the first 4 products and all reviews of the last 3 products are test data. We split the training/testing data for 5 times and average the results. We label complementary entities in each sentence. The whole datasets are labeled by 3 annotators independently. The initial agreement is 82%. Then disagreements are discussed and final agreements are reached. The

statistics of the datasets¹ can be found in ???. We observe that more than half of the reviews have at least one mention of complementary entities and more than 10% sentences have at least one mention of complementary entities.

We also utilize the category information in the meta data of each review to group reviews under the same category together. Then we randomly select 1000 (1K), 3000 (3K), 6000 (6K) reviews from each category and use them for extracting domain knowledge. We choose different scales of reviews to see the performance of CER under the help of different sizes of domain reviews and the scalability of the running time of domain knowledge expansion.

Compared Methods and Evaluation

Since the proposed problem is novel, there are not so much existing baselines that can directly solve the problem. Except for CRF, we compare existing trained models or unsupervised methods with the proposed methods.

NP Chunker: Since most product names are Noun Phrases (NP), we use the same noun phrase chunker ($\langle N \rangle \langle N|CD \rangle^*$) as the proposed method to extract nouns or noun phrases and take them as names of complementary entity. This baseline is used to illustrate a close to random results.

OpenNLP NP Chunker: We utilize the trained noun phrase chunking model from OpenNLP¹ to tag noun phrases. We only consider chunks of words tagged as *NP* as predictions of complementary entities.

¹The annotated dataset is available at <https://www.cs.uic.edu/~hxu/>

¹<https://opennlp.apache.org/>

UIUC NER: We use UIUC Named Entity Tagger [?] to perform Named Entity Recognition (NER) on product reviews. It has 18 labels in total and we consider entities labeled as *PRODUCT* and *ORG* as complementary entities. We use this baseline to demonstrate the performance of a named entity tagger.

CRF: We retrain a Conditional Random Field (CRF) model using 50% reviews of the first 4 products. We use BIO tags. For example, “Works with my Apple iPhone” should be trained/predicted as “Works/O with/O my/O Apple/B iPhone/I”. We use MALLET¹ as the implementation of CRF.

Sceptre: We also retrieve the top 25 complements for the same 7 products from Sceptre [?] and adapt their results for a comparison. Direct comparison is impossible since their task is a link prediction problem with different labeled ground truths. We label and compute the precision of the top 25 predictions and assume annotators have the same background knowledge for both datasets. We observe that the predicted products are mostly non-complementary products (e.g., *network cables*, *mother board*) and all 7 products have similar predictions.

“My” Entity: This baseline extracts complementary entities by finding all nouns/noun phrases modified by word “my” via dependency type *nmod:poss* (e.g., “It works with my phone”). The word “my” usually indicates a product already purchased, so the modified nouns/noun phrases are highly possible complementary entities. We use path

$$(CETT, N) \xrightarrow{nmod:poss} (“my”, PRP\$)$$

¹<http://mallet.cs.umass.edu/>

to extract complementary entities and use the same post-process step as CER/CER1K/3K/6K+.

CER: This method uses all paths without using any domain knowledge.

CER1K+, CER3K+, CER6K+: These methods incorporate domain knowledge extracted from 1000/3000/6000 domain reviews respectively.

We perform our evaluation on each mention of complementary entities and compute precision and recall of extraction. We first count the true positive tp , the false positive fp and the false negative fn of each prediction. For each sentence, one extracted complementary entity that is contained in the annotated complementary entities from the sentence is considered as one count for tp ; one extracted complementary entity that are not found contributes one count to fp ; any annotated complementary entity that can not be extracted contributes one count to fn . We run the system on an i5 laptop with 4GB memory. The system is implemented using Python. All reviews are preprocessed via dependency parsing [?].

Result Analysis

?? demonstrates results of different methods. We can see that CER6K+ performs well on all products. It significantly outperforms CER for each product. This shows that domain knowledge can successfully reduce the noise and improve the precision. More importantly, we notice that using just 3K reviews already gets good performance. This is important for categories with less than 6K reviews. We notice that the F1-scores of CER are close or worse than baselines such as CRF or “My” Entity. The major reason of its low precisions is that Path 5 and Path 6 in ?? can introduce many false positives as we expected. Please note that removing Path 5 and 6 can increase the F1-score of CER. But to have a fair comparison with

CER1K/3K/6K+ and demonstrate the room of improvement, we keep noisy Path 5 and 6 in CER. “My” Entity has better precision but lower recall than those of CER baselines since not all complementary entities are modified by “my”. CRF performs relatively good on these products. But the performance drops for the last 3 products because of the domain adaptation problem. In reality, it is impractical to have training data for each product. Sceptre performs poorly, we guess the reason is that products in “Items also bought” are noisy for training labels. The overall recall of UIUC NER is low because many complementary entities (e.g., general entities like *tablet*) are not named entities. Please note that the information of domain knowledge (or unlabeled data) may help other baselines, but all those baselines may not be able to adopt domain knowledge easily. The running time of all testing is short (less than 1 seconds), so we omit the discussion here.

Next, we demonstrate the running time of domain knowledge expansion and samples of domain knowledge in ???. We observe that expanding knowledge is pretty fast and scalable as the size of reviews grow. We can see that for each category most entities and verbs are reasonable based on our common sense. For example, for category *Cat:Stylus*, the system successfully detects capacitive screen devices as its candidate complementary entities and most drawing actions as domain-specific verbs.

Case Studies

We notice that category-level domain knowledge is useful for extraction. Knowing candidate complementary entities can successfully remove many words that are not complementary entities or even entities. In the reviews of *micro SD card*, many features such as *speed*, *data*, etc. are mentioned; also, common phrases like “in practice”, “in reality”, “in the long run” are

also mentioned. Handling these cases one-by-one is impractical since identifying different types of false positive examples needs different techniques to identify. But knowing candidate complementary entities can easily remove those false positives.

Domain-specific verbs such as *draw*, *insert* and *hold* are successfully mined for *stylus*, *micro SD card* and *tablet stand* respectively. Taking *tablet stand* for example, the significant improvement of the precision of CER1K/3K/6K+ comes from taking *hold* as a domain-specific verb. Reviewers are less likely to use general verbs such as *fit* or *work* for *tablet stand*. The reason could be that a *tablet* is loosely attached to a *tablet stand*. So people tend to use “It holds tablet well” a lot. However, this sentence has a *doobj* relation that usually relates a verb to an object, which can appear in almost any sentence. Knowing *hold* is a domain-specific verb is important to improve the precision. The major errors come from parsing errors since reviews are informal texts.

6.2.2 – Augmented Key-value Pairs of Knowledge

This subsection focuses on supervised method of complementary entity recognition. A traditional supervised method like Conditional Random Field (CRF) may have good precision on such extraction yet suffer from low recall due to unseen context words appear in the test data but not in the training data. To solve this problem, instead of using supervised method, [?] uses an unsupervised method by leveraging manually-crafted high precision dependency rules [?, ?, ?, ?] to expand (bootstrap) context words as knowledge on a large amount of unlabeled data and combine those context words with another set of manually-crafted high recall dependency rules for CER. However, crafting dependency rules for both context words and CER can be

time-consuming and such rules may be domain dependent and subject to change for new domains.

To benefit from both the supervised and unsupervised methods, we consider to automatically learn patterns for both CER and knowledge expansion (of context words) from training data and expand context word knowledge on unlabeled data. So when making predictions on the test data, the model can leverage more contextual knowledge from unlabeled data to make better predictions. Or put it another way, we wish the prediction behavior of a supervised model can change after training when it sees more unlabeled data. This framework is inspired by the lifelong sequence labeling method proposed in [?,?]. However, we do not expand knowledge for lifelong learning here and we make one step further: we automatically learn knowledge-based features and knowledge (or context words) as key-value pairs rather than manually crafting them. We use CRF as the base learner and augment CRF with knowledge-based features (a modified dependency relation) that are automatically learned from the training data. The augmented CRF is called Knowledge-based CRF (KCRF).

The proposed method has the following steps:

Pre-training We first train a CRF as a traditional sequence labeling training process using hand-crafted features, including primitive features (defined later) such as dependency relation based features. Then we automatically select from those primitive features as knowledge-based features to build a group of key-value pairs as initial knowledge, where keys are selected feature types and values are feature values (e.g., context words) appear in the training data.

Knowledge-based Training Then we train a Knowledge-based CRF (KCRF) based on the initial

knowledge. So KCRF knows which features (as keys) can be used to expand knowledge (get more values for the same key).

Knowledge Expansion We expand the values in initial knowledge by iteratively collecting reliable knowledge from reliable predictions on plenty of unlabeled reviews. Experiments demonstrate that the expanded knowledge is effective in predicting test data.

Preliminaries

We briefly review the terms used throughout this paper. We use dependency relations as the major type of knowledge-based features since a dependency relation associates one word (current word) with another word (context word), which can be viewed as a piece of context knowledge.

Dependency Feature: A *dependency feature* for the n -th word is a simplified dependency relation with the following attributes:

$$(role, type, gov/dep, govpos/deppos),$$

where *role* can be either “GOV” or “DEP” indicating whether the n -th word is a governor word or a dependent word; *type* is the type of the original dependency relation; *gov/dep* is the other word associated with the n -th word via the original dependency relation and *govpos/deppos* is the POS tag of the other word.

Note here we omit the n -th word, its POS-tag in a dependency relation and define them as separate features since they are the same for all dependency features of the n -th word.

Primitive Feature: A *primitive feature* can be either a dependency feature or a current word feature (taking current word as a feature). Primitive features are used to generate knowledge base.

Knowledge Base: A *knowledge base* is a set of key-value pairs $(k, v) \in KB$, where k is the *knowledge type* and v is the *knowledge value* belonging to that type. The same k may have multiple knowledge values. We further define separate knowledge bases KB^{t_o} for each tag $t_o \in T$, where T is the set of output labels for sequence labeling and $KB = \{KB^{t_o} | t_o \in T\}$.

Knowledge-based Feature: A *knowledge-based feature* is defined based on a knowledge type k in a knowledge base. We use $d \in D$ to denote an index about a knowledge-based feature in a feature vector x_n . So $x_{n,d} = 1$ indicates that the d -th feature of the n -th word is a knowledge-based feature of type k_d with some (k_d, v) found in KB . We use $K = \cup_{t_o \in T} K^{t_o}$ to denote all knowledge types in KB .

A primitive feature can generate a knowledge-based feature in the form of (k, v) . Current word feature has a corresponding knowledge type $k = [\text{WORD}]$ and takes the current word as the knowledge value v (e.g., we use $([\text{WORD}], \text{"phone"})$ to indicate "phone" is in the knowledge base as type $[\text{WORD}]$). Dependency features have a corresponding knowledge type $k = [\text{role}, \text{type}, \text{govpos}/\text{deppos}]$, which is similar to a dependency feature. The *gov/dep* part (the other word related to the current word in a dependency relation) is considered as the knowledge value $v = \text{gov}/\text{dep}$. For example, if $K = \{([\text{WORD}], [\text{DEP}, \text{nmod:with}, \text{VBZ}], [\text{GOV}, \text{nmod:poss}, \text{PRP\$}])\}$ and we have knowledge value "phone" and "works" for the first two types, we may have $KB = \{([\text{WORD}], \text{"phone"}), ([\text{DEP}, \text{nmod:with}, \text{VBZ}], \text{"works"})\}$. We

describe how to automatically obtain all knowledge types K and how to get initial knowledge values in the next section.

Pre-training

The role of pre-training is to identify knowledge types K and initial knowledge values. It is important to obtain useful knowledge types and reliable knowledge values because some knowledge types may not help the prediction task and wrong knowledge values may be harmful to the performance of predictions. Fortunately, a trained CRF model can tell us which features are more useful for prediction and need to be enhanced with knowledge. The basic idea is to perform a traditional CRF training using primitive features and select knowledge-based features K and initial knowledge values based on the weights λ of primitive features in the trained CRF model.

Let x'_n denote a feature vector of the n -th word in an input sequence for pre-training. We use $r \in R$ to denote an index about a primitive feature so $x'_{n,r} = 1$ means the n -th word has a primitive feature (e.g., $WORD = \text{"phone"}$ or $(DEP, nmod:with, works, VBZ)$) indexed by r . We distinguish different feature functions according to the value of y_n and the primitive features indexed by r in x'_n . We care about the following type of feature function, which is a multiplication of 2 indicator functions:

$$f_r^{t_o}(y_n, x'_n) = \mathbb{I}(y_n = t_o) \mathbb{I}(x'_{n,r}), \quad (6.4)$$

It turns all combinations of primitive features $r \in R$ and tag set T into $\{0, 1\}$. Further we use a similar notation $\lambda_r^{t_o}$ for the corresponding weight. A positive weight $\lambda_r^{t_o} > 0$ indicates a primitive feature indexed by r has positive impact on predicting tag t_o ; while a negative weight $\lambda_r^{t_o} < 0$ indicates a primitive feature indexed by r has negative impact on predicting tag t_o .

After training CRF using primitive features, we obtain the weights $\lambda_r^{t_o}$ for $r \in R$ and $t_o \in T$, which are very important to know which primitive features are more useful for prediction and need to be expanded. We use entropy to measure the usefulness of a primitive feature. We compute the probability of each tag t_o for r :

$$p^r(t_o) = \frac{\exp(\lambda_r^{t_o})}{\sum_{l=1}^{|T|} \exp(\lambda_r^{t_l})}. \quad (6.5)$$

Based on Equation ??, we compute the entropy for a primitive feature indexed by r :

$$H(r) = - \sum_{o=1}^{|T|} p^r(t_o) \log p^r(t_o). \quad (6.6)$$

The intuition of using entropy is that a salient primitive feature should favor some tags over the others so it has low entropy. We select primitive features that attain the maximum probability for tag t_o and have entropies below δ (We set $\delta = 0.3$ for $|T| = 2$):

$$R^{t_o} = \{r | H(r) < \delta \wedge t_o = \arg \max_{t_l} p^r(t_l)\}. \quad (6.7)$$

We obtain a set of primitive features indexed by R^{t_o} and use it to generate (k, v) for tag t_o since each primitive feature can be interpreted as a (k, v) . We group the same k under R^{t_o} to form the set K^{t_o} and use the associated v as initial knowledge value.

Knowledge-based Training

We train KCRF using knowledge-based features in this section. A knowledge-based feature simply tells whether a feature found in an example with a specified knowledge type has some values found in the current knowledge base (or KB). We use x_n to denote the feature vectors with knowledge-based features for the n -th word and use $d \in D$ to denote a knowledge-based feature indexed by d in x_n . So $x_{n,d} = 1$ indicates that the d -th feature is a knowledge-based feature and the n -th word has a knowledge with type k_d and initial knowledge value v found in KB :

$$x_{n,d} = \mathbb{I}((k_d, v) \in KB). \quad (6.8)$$

For example, if $([DEP, \text{nmod:with}, VBZ], \text{"works"}) \in KB$, the word "phone" has a dependency relation knowledge-based feature with type $k = [DEP, \text{nmod:with}, VBZ]$ and $v = \text{"works"}$ and current word knowledge-based feature $k = [WORD]$ and $v = \text{"phone"}$. We denote the trained KCRF as c and its parameters λ^c . It predicts on x and generates probabilities $p(y|x; \lambda^c)$ for $y \in \mathcal{Y}$.

Knowledge Expansion

We perform sequence labeling on a large amount of unlabeled reviews under the same category as the target entity to expand the KB using c . We assume that target entities under

the same category share similar knowledge. Here the key point is to ensure the quality of the expanded knowledge since it is very easy to have harmful knowledge from unlabeled reviews without human supervision. We aggregate knowledge from *reliable predictions* on those reviews. To obtain a reliable prediction for the n -th word, we marginalize over $y_{1:N}$ of other positions except n as:

$$p(y_n|x; \lambda^c) = \sum_{y_1} \cdots \sum_{y_{n-1}} \sum_{y_{n+1}} \cdots \sum_{y_N} p(y_{1:N}|x; \lambda^c). \quad (6.9)$$

Then if a tag t_o attains the maximum probability that is larger than a threshold: $\max_{t_o} (p(y_n = t_o|x; \lambda^c)) > \delta'$, we consider it as a reliable prediction for tag t_o at position n . The knowledge-based features k_d and potential knowledge values associated with such a reliable prediction are considered as candidate knowledge. We use cKB^{t_o} as the set of candidate knowledge for tag t_o . We further prune the knowledge since similar knowledge may appear in the knowledge base of another tag so this can make candidate knowledge from reliable predictions not reliable.

Algorithm 5: Knowledge Expansion

Input : (c, KB) , with $KB = \{KB^{t_o} | t_o \in T\}$, $U = \{u_1, \dots, u_{|U|}\}$ **Output**: (c, KB) , with updated KB

```

1 do
2   transform each  $u \in U$  into a sequence of knowledge-based feature vectors  $x \in X$  using
    $KB$ 
3   for  $x \in X$  do
4     use  $(c, KB)$  to predict
5     use ?? to compute  $p(y_n | x; \lambda^c)$  for  $n = 1 : N$ 
6     for  $n = 1, \dots, N$  do
7       for  $t_o \in T$  do
8         if  $\max_{t_o} (p(y_n = t_o | x)) > \delta'$  then
9           add associated  $(k, v)$  to  $cKB^{t_o}$  for  $k \in K^{t_o}$ 
10          end
11        end
12      end
13    end
14    for  $t_o \in T$  do
15       $cKB^{t_o} \leftarrow cKB^{t_o} - \cup_{t_l \neq t_o} cKB^{t_l}$ 
16       $KB.KB^{t_o} \leftarrow KB^{t_o} \cup cKB^{t_o}$ 
17    end
18     $cKB \leftarrow \cup_{t_o} cKB^{t_o}$ 
19  while  $cKB \neq \emptyset$ 
20 return  $(c, KB)$ 

```

Algorithm ?? is to maintain high-quality knowledge during expansion. We use U to denote a set of unlabeled sequences and we transform $u \in U$ to knowledge-based feature vectors $x \in X$ based on current knowledge base KB (line ??). We apply KCRF c and current KB on x and get

Product	CRF(-)DR			CRF			CRF-Init			KCRF		
	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1	\mathcal{P}	\mathcal{R}	\mathcal{F}_1
Stylus	0.5	0.54	0.52	0.75	0.50	0.60	0.84	0.64	0.73	0.66	0.84	0.74
Micro SD Card	0.63	0.51	0.56	0.89	0.44	0.59	0.87	0.57	0.69	0.77	0.70	0.74
Mouse	0.54	0.37	0.44	0.80	0.48	0.60	0.75	0.53	0.62	0.68	0.68	0.68
Tablet Stand	0.58	0.43	0.49	0.79	0.40	0.53	0.85	0.46	0.60	0.75	0.65	0.70
Keyboard	0.54	0.46	0.5	0.8	0.42	0.55	0.8	0.34	0.48	0.66	0.72	0.69
Notebook Sleeve	0.69	0.38	0.49	0.90	0.23	0.37	0.91	0.23	0.37	0.77	0.63	0.69
Compact Flash	0.75	0.61	0.67	0.92	0.46	0.62	0.89	0.51	0.65	0.82	0.73	0.77

TABLE XIX

Results of KCRF

reliable prediction in line ??). We add associated knowledge in line ??) and prune it to get reliable knowledge and update KB in line ??-??. The whole process will stop once no reliable knowledge is available. Note that during knowledge expansion, KCRF c itself is never re-trained.

Experimental Results

Dataset

We use the dataset¹ from [?], which includes 7 products. We take 50% reviews of the first 4 products as the training data for all methods that require supervised training. The remaining

¹<https://www.cs.uic.edu/~hxu/>

reviews of the first 4 products (for in-domain test) and all reviews of the last 3 products (for out-of-domain test) are test data. Similar to [?], we also randomly select 6000 unlabeled reviews for each category from [?] and use them as unlabeled reviews to expand knowledge.

Compared Methods

Since this paper proposes a supervised method on CER, we focus on the improvements of KCRF over CRF. We use CRFSuite¹ as the base implementation of CRF.

CRF(-)DR: This is a very basic CRF without dependency relations as features to show that dependency relations are useful features. We use the following features: current word, POS-tags, 4 nearby words and POS-tags on the left and right, number of digits and whether current word has slash/dash.

CRF: This is the baseline with dependency relations as features. It is also the same learner as in the pre-training step of KCRF.

CRF-Init: This baseline uses the trained KCRF and initial KB directly on test data without knowledge expansion on unlabeled data.

KCRF: This is the proposed method that uses trained KCRF and initial KB to expand knowledge on unlabeled reviews under the same category as the target entity. We empirically set $\delta' = 0.8$ as the precisions of most predictions are around 0.8.

Evaluation Methods

¹<http://www.chokkan.org/software/crfsuite/>

We perform evaluation on each mention of complementary entities. We count true positive tp , false positive fp and false negative fn . For each sentence, one recognized complementary entity that is contained in the annotated complementary entities for that sentence is considered as one count for the tp ; one recognized complementary entity that are not found contributes one count to the fp ; any annotated complementary entity that can not be recognized contributes one count to the fn . Then we compute precision \mathcal{P} , recall \mathcal{R} and F1-score $\mathcal{F1}$ based on tp , fp and fn .

Result Analysis

From ??, we can see that KCRF performs well on F1-score. It significantly outperforms other methods on recall, which indicates that the expanded knowledge is helpful. CRF-Init performs better than CRF on most products, which indicates that knowledge-based features are better than primitive features in general. However, we notice that in order to get a higher recall, KCRF sacrifices its precision a lot. So how to further ensure that the expanded knowledge is of high quality to keep high precision is still an open problem.

The performance of KCRF does not drop much for the last 3 products even though we do not have any training data for those products. This is because KCRF can utilize unlabeled data to expand knowledge about the last 3 products separately from the knowledge of the first 4 products. One intuitive example is that “work” can be a frequent general verb knowledge that exists in the training data for some verb related knowledge type. Then later KCRF expands such a verb to other domain-specific verbs like “insert” for Compact Flash that does not have training data.

6.3 Question Answering

In this section, we discuss the usage of post-training to question answering. We focus on a novel review-based task called review reading comprehension (RRC).

6.3.1 – Motivation

Question-answering plays an important role in e-commerce as it allows potential customers to actively seek crucial information about products or services to help their purchase decision making. Inspired by the recent success of machine reading comprehension (MRC) on formal documents, this paper explores the potential of turning customer reviews into a large source of knowledge that can be exploited to answer user questions. We call this problem *Review Reading Comprehension* (RRC). To the best of our knowledge, no existing work has been done on RRC. In this work, we first build an RRC dataset called ReviewRC based on a popular benchmark for aspect-based sentiment analysis. Since ReviewRC has limited training examples for RRC (and also for aspect-based sentiment analysis), we then explore a novel post-training approach on the popular language model BERT to enhance the performance of fine-tuning of BERT for RRC. To show the generality of the approach, the proposed post-training is also applied to some other review-based tasks such as aspect extraction and aspect sentiment classification in aspect-based sentiment analysis.

For online commerce, question-answering (QA) serves either as a standalone application of customer service or as a crucial component of a dialog system that answers user questions. Many intelligent personal assistants (such as Amazon Alexa and Google Assistant) support online shopping by allowing the user to speak directly to the assistants. One major hindrance to this

mode of shopping is that such systems have limited capability to answer user questions about products (or services), which are vital for customer decision making. As such, an intelligent agent that can automatically answer customers' questions is very important for the success of online businesses.

Given the ever-changing environment of products and services, it is very hard, if not impossible, to pre-compile an up-to-date and reliable knowledge base to cover a wide assortment of questions that customers may ask, such as in factoid-based KB-QA [?, ?, ?, ?]. As a compromise, many online businesses leverage community question-answering (CQA) [?] to crowdsource answers from existing customers. However, the problem with this approach is that many questions are not answered, and if they are answered, the answers are delayed, which is not suitable for interactive QA. In this paper, we explore the potential of using product reviews as a large source of user experiences that can be exploited to obtain answers to user questions. Although there are existing studies that have used information retrieval (IR) techniques [?, ?] to find a whole review as the response to a user question, giving the whole review to the user is undesirable as it is quite time-consuming for the user to read it.

Inspired by the success of Machine Reading Comphrenesions (MRC) [?, ?], we propose a novel task called Review Reading Comprehension (RRC) as following.

Problem Definition: Given a question $q = (q_1, \dots, q_m)$ from a customer (or user) about a product and a review $d = (d_1, \dots, d_n)$ for that product containing the information to answer q , find a sequence of tokens (a text span) $a = (d_s, \dots, d_e)$ in d that answers q correctly, where $1 \leq s \leq n, 1 \leq e \leq n$, and $s \leq e$.

Questions
Q1: Does it have an internal hard drive ?
Q2: How large is the internal hard drive ?
Q3: is the capacity of the internal hard drive OK ?
Review <p>Excellent value and a must buy for someone looking for a Macbook . You ca n't get any better than this price and it come with_{A1} an internal disk drive . All the newer MacBooks do not . Plus you get 500GB_{A2} which is also a great_{A3} feature . Also , the resale value on this will keep . I highly recommend you get one before they are gone .</p>

TABLE XX

Review reading comprehension

A sample *laptop* review is shown in ???. We can see that customers may not only ask factoid questions such as the specs about some aspects of the laptop as in the first and second questions but also subjective or opinion questions about some aspects (capacity of the hard drive), as in the third question. RRC poses some *domain challenges* compared to the traditional MRC on Wikipedia, such as the need for rich product knowledge, informal text, and fine-grained opinions (there is almost no subjective content in Wikipedia articles). Research also shows that yes/no questions are very frequent for products with complicated specifications [?, ?].

To the best of our knowledge, no existing work has been done in RRC. This work first builds an RRC dataset called ReviewRC, using reviews from SemEval 2016 Task 5¹, which is a popular dataset for aspect-based sentiment analysis (ABSA) [?] in the domains of *laptop* and *restaurant*. We detail ReviewRC in Chapter ??. Given the wide spectrum of domains (types of products or services) in online businesses and the prohibitive cost of annotation, ReviewRC can only be considered to have a limited number of annotated examples for supervised training, which still leaves the domain challenges partially unresolved.

To simplify the writing, we refer MRC as a general-purpose RC task on formal text (non-review) and RRC as an end-task specifically focused on reviews.), where the former enhances domain-awareness and the latter strengthens MRC task-awareness. Although BERT gains great success on SQuAD, this success is based on the huge amount of training examples of SQuAD (100,000+). This amount is large enough to ameliorate the flaws of BERT that has almost no

¹<http://alt.qcri.org/semeval2016/task5/>. We choose these review datasets to align RRC with existing research on sentiment analysis.

questions on the left side and no textual span predictions based on both the question and the document on the right side. However, a small amount of fine-tuning examples is not sufficient to turn BERT to be more task-aware.

Related Works

Many datasets have been created for MRC from formally written and objective texts, e.g., Wikipedia (WikiReading [?], SQuAD [?,?], WikiHop [?], DRCD [?], QuAC [?], HotpotQA [?]) news and other articles (CNN/Daily Mail [?], NewsQA [?], RACE [?]), fictional stories (MCTest [?], CBT [?], NarrativeQA [?]), and general Web documents (MS MARCO [?], TriviaQA [?], SearchQA [?]). Also, CoQA [?] is built from multiple sources, such as Wikipedia, Reddit, News, Mid/High School Exams, Literature, etc. To the best of our knowledge, MRC has not been used on primarily subjective reviews. As such, we created a review-based MRC dataset called ReviewRC. Answers from ReviewRC are extractive (similar to SQuAD [?,?]) rather than abstractive (or generative) (such as in MS MARCO [?] and CoQA [?]). This is crucial because online businesses are typically cost-sensitive and extractive answers written by humans can avoid generating incorrect answers beyond the contents in reviews by an AI agent.

Community QA (CQA) is widely adopted by online businesses [?] to help users. However, since it solely relies on humans to give answers, it often takes a long time to get a question answered or even not answered at all as we discussed in the introduction. Although there exists researches that align reviews to questions as an information retrieval task [?,?], giving a whole review to the user to read is time-consuming and not suitable for customer service settings that require interactive responses.

Knowledge bases (KBs) (such as Freebase [?, ?, ?] or DBpedia [?, ?]) have been used for question answering [?]. However, the ever-changing nature of online businesses, where new products and services appear constantly, makes it prohibitive to build a high-quality KB to cover all new products and services.

Reviews also serve as a rich resource for sentiment analysis [?, ?, ?, ?]. Although document-level (review) sentiment classification may be considered as a solved problem (given ratings are largely available), aspect-based sentiment analysis (ABSA) is still an open challenge, where alleviating the cost of the human annotation is also a major issue. ABSA aims to turn unstructured reviews into structured fine-grained aspects (such as the “battery” of a laptop) and their associated opinions (e.g., “good battery” is *positive* about the aspect battery). Two important tasks in ABSA are aspect extraction (AE) and aspect sentiment classification (ASC) [?], where the former aims to extract aspects (e.g., “battery”) and the latter targets to identify the polarity for a given aspect (e.g., *positive* for *battery*). Recently, supervised deep learning models dominate both tasks [?, ?, ?, ?, ?] and many of these models use handcrafted features, lexicons, and complicated neural network architectures to remedy the insufficient training examples from both tasks. Although these approaches may achieve better performances by manually injecting human knowledge into the model, human baby-sat models may not be intelligent enough¹ and automated representation learning from review corpora is always preferred [?, ?]. We push forward this trend with the recent advance in pre-trained language models from deep

¹<http://www.incompleteideas.net/IncIdeas/BitterLesson.html>

learning [?, ?, ?, ?, ?]. Although it is practical to train domain word embeddings from scratch on large-scale review corpora [?], it is impractical to train language models from scratch with limited computational resources. As such, we show that it is practical to adapt language models pre-trained from formal texts to domain reviews.

RRC may suffer from the prohibitive cost of annotating large-scale training data covering a wide range of domains. And BERT severely lacks two kinds of prior knowledge: (1) large-scale domain knowledge (e.g., about a specific product category), and (2) task-awareness knowledge (MRC/RRC in this case). We detail the technique of jointly incorporating these two types of knowledge as a post-training stage in Chapter ??.

Results

As there are no existing datasets for RRC and to be consistent with existing research on sentiment analysis, we adopt the *laptop* and *restaurant* reviews of SemEval 2016 Task 5 as the source to create datasets for RRC. We do not use SemEval 2014 Task 4 or SemEval 2015 Task 12 because these datasets do not come with the review(document)-level XML tags to recover whole reviews from review sentences. We keep the split of training and testing of the SemEval 2016 Task 5 datasets and annotate multiple QAs for each review following the way of constructing QAs for the SQuAD 1.1 datasets [?].

To make sure our questions are close to real-world questions, 2 annotators are first exposed to 400 QAs from CQA (under the laptop category in Amazon.com or popular restaurants in Yelp.com) to get familiar with real questions. Then they are asked to read reviews and independently label textual spans and ask corresponding questions when they feel the textual

spans contain valuable information that customers may care about. The textual spans are labeled to be as concise as possible but still human-readable. Note that the annotations for sentiment analysis tasks are not exposed to annotators to avoid biased annotation on RRC. Since it is unlikely that the two annotators can label the same QAs (the same questions with the same answer spans), they further mutually check each other’s annotations and disagreements are discussed until agreements are reached. Annotators are encouraged to label as many questions as possible from testing reviews to get more test examples. A training review is encouraged to have 2 questions (training examples) on average to have good coverage of reviews.

The annotated data is in the format of SQuAD 1.1 [?] to ensure compatibility with existing implementations of MRC models. The statistics of the RRC dataset (ReviewRC) are shown in ??.

Since SemEval datasets do not come with a validation set, we further split 20% of reviews from the training set for validation.

Dataset	Num. of Questions	Num. of Reviews
Laptop Training	1015	443
Laptop Testing	351	79
Restaurant Training	799	347
Restaurant Testing	431	90

TABLE XXI

Statistics of ReviewRC Dataset

Compared Methods

As BERT outperforms existing open-source MRC baselines by a large margin, we do not intend to exhaust existing implementations but focus on variants of BERT introduced in this paper.

DrQA is a baseline from the document reader¹ of DrQA [?]. We adopt this baseline because of its simple implementation for reproducibility. We run the document reader with random initialization and train it directly on ReviewRC. We use all default hyper-parameter settings for this baseline except the number of epochs, which is set as 60 for better convergence.

DrQA+MRC is derived from the above baseline with official pre-trained weights on SQuAD. We fine-tune document reader with ReviewRC. We expand the vocabulary of the embedding layer from the pre-trained model on ReviewRC since reviews may have words that are rare in Wikipedia and keep other hyper-parameters as their defaults.

For AE and ASC, we summarize the scores of the state-of-the-arts on SemEval (based the best of our knowledge) for brevity.

Lastly, we have the following BERT variants.

BERT leverages the vanilla BERT pre-trained weights and fine-tunes on all 3 end tasks. We use this baseline to show that BERT’s pre-trained weights alone have limited performance gains on review-based tasks.

¹<https://github.com/facebookresearch/DrQA>

BERT-DK post-trains BERT’s weights only on domain knowledge (reviews) and fine-tunes on the 3 end tasks. We use BERT-DK and the following BERT-MRC.

BERT-MRC post-trains BERT’s weights on SQuAD 1.1 and then fine-tunes on the 3 end tasks.

BERT-PT (proposed method) post-trains BERT’s weights using the joint post-training algorithm in Section ?? and then fine-tunes on the 3 end tasks.

Evaluation Metrics and Model Selection

To be consistent with existing research on MRC, we use the same evaluation script from SQuAD 1.1 [?] for RRC, which reports Exact Match (EM) and F1 scores. EM requires the answers to have an exact string match with human-annotated answer spans. F1 score is the averaged F1 scores of individual answers, which is typically higher than EM and is the major metric. Each F1 score is the harmonic mean of individual precision and recalls computed based on the number of overlapped words between the predicted answer and human-annotated answers.

We set the maximum number of epochs to 4 for BERT variants, though most runs converge just within 2 epochs. Results are reported as averages of **9** runs (9 different random seeds for random batch generation).¹

Result Analysis

¹We notice that adopting 5 runs used by existing researches still has a high variance for a fair comparison.

Domain	Laptop		Rest.	
Methods	EM	F1	EM	F1
DrQA [?]	38.26	50.99	49.52	63.73
DrQA+MRC [?]	40.43	58.16	52.39	67.77
BERT	39.54	54.72	44.39	58.76
BERT-DK	42.67	57.56	48.93	62.81
BERT-MRC	47.01	63.87	54.78	68.84
BERT-PT	48.05	64.51	59.22	73.08

TABLE XXII

RRC in EM (Exact Match) and F1.

The results of RRC are shown in ???. We observed that the proposed joint post-training (BERT-PT) has the best performance on all tasks in all domains, which show the benefits of having two types of knowledge. To our surprise, we found that the vanilla pre-trained weights of BERT do not work well for review-based tasks, although it achieves state-of-the-art results on many other NLP tasks [?]. This justifies the need to adapt BERT to review-based tasks. We noticed that the roles of domain knowledge and task knowledge vary for different tasks and domains. For RRC, we found that the performance gain of BERT-PT mostly comes from task-awareness (MRC) post-training (as indicated by BERT-MRC). The domain knowledge helps more for restaurant than for laptop. We suspect the reason is that certain types of knowledge (such as specifications) of laptop are already present in Wikipedia, whereas Wikipedia has little

knowledge about restaurant. We further investigated the examples improved by BERT-MRC and found that the boundaries of spans (especially short spans) were greatly improved.

The errors on RRC mainly come from boundaries of spans that are not concise enough and incorrect location of spans that may have certain nearby words related to the question. We believe precisely understanding user's experience is challenging from only domain post-training given limited help from the RRC data and no help from the Wikipedia data.

6.4 Dialog System

Given the recent popularity of research in dialog system, I further discuss the usage of lifelong representation learning for conversational AI. I mainly focus on two tasks: one is the extension of RRC discussed in the previous section; the other is a novel task called conversational recommendation that aims to learn dynamic graph reasoning.

6.4.1 – Review Conversational Reading Comprehension (RCRC)

Seeking information to assess whether a product or service suits one's needs is an important activity in consumer decision making. One major hindrance for online businesses is that the consumers often have difficulty to get answers to their questions. With the ever-changing environment, it is very hard, if not impossible, for businesses to pre-compile an up-to-date knowledge base to answer user questions as in KB-QA [?, ?, ?, ?]. Although community question-answering (CQA) helps [?], one has to be lucky to get an existing customer to answer a question quickly. There is work on retrieving whole reviews relevant to a question [?, ?], but it is not ideal for the user to read the whole reviews to fish for answers.

TABLE XXIII

Review conversational reading comprehension (RCRC)

A Laptop Review:

I purchased my Macbook Pro Retina from my school since I had a student discount , but I would gladly purchase it from Amazon for full price again if I had too . The Retina is **great** , its **amazingly fast** when it boots up because of the **SSD storage** and the clarity of the screen is **amazing** as well...

Turns of Questions from a Customer:

*q*₁: how is retina display ?

*q*₂: speed of booting up ?

*q*₃: why ?

*q*₄: what 's the capacity of that ? (NO ANSWER)

*q*₅: is the screen clear ?

Inspired by conversational reading comprehension (CRC) [?, ?, ?], we explore the possibility of turning reviews into a valuable source of knowledge of real-world experiences and using it to answer customer or user multi-turn questions. We call this *Review Conversational Reading Comprehension* (RCRC). The conversational setting enables the user to go into details via more specific questions and to simplify their questions by either omitting or co-referencing information in the previous context. As shown in ??, the user first has an *opinion* question about “retina display” (an *aspect*) of a laptop. Then he/she carries (or omits) the question type *opinion* from the first question to the second question about another *aspect* “boot-up speed”. Later, he/she carries the *aspect* of the second question, but changes the question type to *opinion reason* and then co-references the *aspect* “SSD” from the third answer and asks for the capacity (a *sub-aspect*) of “SSD”. Unfortunately, there is no answer in this review. Finally, the customer asks another *aspect* as in the fifth question. RCRC is defined as follows.

RCRC Definition: Given a review that consists of a sequence of n tokens $d = (d_1, \dots, d_n)$, a history of past $k - 1$ questions and answers as the context $C = (q_1, a_1, q_2, a_2, \dots, q_{k-1}, a_{k-1})$ and the current question q_k , find a sequence of tokens (a textual span) $a = (d_s, \dots, d_e)$ in d that answers q_k based on C , where $1 \leq s \leq n$, $s \leq e \leq n$, and $s \leq e$, or return *NO ANSWER* ($s, e = 0$) if the review does not contain the answer for q_k .

Note that although RCRC focuses on one review, it can potentially be deployed on the setting of multiple reviews (e.g., all reviews for a product), where the context C may contain answers from different reviews. To the best of our knowledge, there are no existing review datasets

for RCRC. We first build a dataset called $(RC)_2$ based on laptop and restaurant reviews from SemEval 2016 Task 5.¹

Given the wide spectrum of domains in online businesses and the prohibitive cost of annotation, $(RC)_2$ has limited training data, as in many other tasks of sentiment analysis.

As a result, the challenge is how to effectively improve the performance of RCRC. We adopt BERT [?] as our base model since it can be either a feature encoder or a standalone model that achieves good performance on CRC [?]. BERT bears with task-agnostic features, which require task-specific architecture and many supervised training examples to train(fine-tune) on an end task. As $(RC)_2$ has limited training data, we propose a novel task-aware *pre-tuning* to further bridge the gap between BERT pre-training and RCRC task-awareness, as discussed in Chapter ?? . Pre-tuning requires no annotation of CRC (or RCRC) data but just QA pairs (from CQA) and reviews that are largely available online. The data are general and can potentially be used in other machine reading comprehension tasks. Experimental results show that the proposed approach achieves competitive performance even compared with the supervised approach using a large-scale annotated dataset.

Datasets

We adopt SemEval 2016 Task 5 as the review source for RCRC (to be consistent with research in sentiment analysis), which contains two domains *laptop* and *restaurant*. We kept the split of training and testing and annotated dialogs on each review. The annotation guideline can be

¹<http://alt.qcri.org/semeval2016/task5/> We choose this dataset to better align with existing research in sentiment analysis.

found in supplemental material¹. To ensure questions are real-world questions, annotators are first asked to read hundreds of community questions and answers (CQA) from real customers. The statistics of the annotated (RC)₂ dataset is shown in ?? . We use 20% of the training reviews as the validation set for each domain.

For the proposed pre-tuning, we collect QA pairs and reviews for these two domains. For *laptop*, we collect the reviews from [?] and QA pairs from [?] both under the laptop category of Amazon.com. We exclude products in the test data of (RC)₂. This gives us 113,728 laptop reviews and 19,104 QA pairs. For *restaurant*, we crawl reviews and all QA pairs from the top 60 restaurants in each U.S. city from Yelp.com. This ends with 197,333 restaurant reviews and 49,587 QA pairs. Based on the number of QAs, Algorithm 1 is run $k = 10$ times for laptop and $k = 5$ times for restaurant.

To compare with the performance of a fully-supervised approach, we leverage the CoQA dataset with 7,199 documents (covering domains in Children’s Story, Mid/High School Literature, News, Wikipedia, etc.) and 108,647 turns of question/answer span annotated via crowdsourcing.

Compared Methods

We compare the following methods:

DrQA is a CRC baseline coming with the CoQA dataset¹.

¹The annotated data is in the format of CoQA [?] to help future research. But we do not focus on generative annotation as in CoQA because businesses are sensitive to errors of generative models

¹<https://github.com/stanfordnlp/coqa-baselines>

TABLE XXIV

Statistics of (RC) ₂ Datasets.		
Training	Laptop	Restaurant
# of reviews	445	350
# of dialogs	506	382
# of dialog /w 3+ turns	375	315
# of questions	1679	1486
% of no answers	24.3%	24.2%
Testing	Laptop	Restaurant
# of reviews	79	90
# of dialog	170	160
# of dialog /w 3+ turns	148	135
# of questions	804	803
% of no answers	26.6%	28.0%

DrQA+CoQA is the above baseline pre-tuned on the CoQA dataset and then fine-tuned on $(RC)_2$ to show that even DrQA pre-trained on CoQA is sub-optimal.

BERT¹ is the pre-trained BERT weights directly fine-tuned on $(RC)_2$ for ablation study on the effectiveness of pre-tuning.

BERT+review first tunes BERT on domain reviews using the same objectives as BERT pre-training and then fine-tunes on $(RC)_2$. We use this baseline to show that a simple domain-adaptation of BERT is not sufficient.

BERT+CoQA first fine-tunes BERT on the supervised CoQA data and then fine-tunes on $(RC)_2$. We use this baseline to show that even compared with using this large-scale supervised data, our pre-tuning is still very competitive.

BERT+Pre-tuning is the proposed approach.

Hyper-parameters and Evaluation

We set the maximum length of BERT to 256 with the maximum length of context+question to 96 ($h_{\max} = 9$ for Algorithm ??) and the batch size to 16. We perform pre-tuning for 10k steps. CoQA fine-tuning converges in 2 epochs. Fine-tune RCRC is performed for 4 epochs and most runs converged within 3 epochs. We search the maximum number of turns in context C for RCRC fine-tuning using the validation set, which ends with 6 turns for laptop and 5 turns for restaurant. Results are reported as averages of 3 runs. To be consistent, we leverage the same

¹We choose BERT_{BASE} as we cannot fit BERT_{LARGE} into the memory.

Domain	Laptop		Rest.	
Methods	EM	F1	EM	F1
DrQA	28.5	36.6	41.6	50.3
DrQA+CoQA(supervised)	40.4	51.4	47.7	58.5
BERT	38.57	48.67	46.87	55.07
BERT+review	34.53	43.83	47.23	53.7
BERT+CoQA(supervised)	47.1	58.9	56.57	67.97
BERT+Pre-tuning	46.0	57.23	54.57	64.43

TABLE XXV

RCRC on EM (Exact Match) and F1.

evaluation script as CoQA, which reports turn-level Exact Match (EM) and F1 scores for all turns in all dialogs.

Result Analysis

As shown in ??, BERT+Pre-tuning has significant performance gains over BERT fine-tuned directly on $(RC)_2$ by 9%. BERT is overall better than DrQA. But directly using review documents to adapt BERT does not yield better results as in BERT+review. We suspect the task of RCRC still requires a certain degree of general language understanding on the question side and BERT+review also has the effect of (catastrophic) forgetting [?] on such representation. Further, large-scale annotated CoQA data can boost the performance for both DrQA and BERT. However, our pre-tuning approach still has competitive performance and it requires no annotation at all.

We examine the errors of BERT+Pre-tuning and realize that both locations of span and span boundaries tend to have errors, indicating a significant room for improvement.

6.4.2 – Memory-grounded Conversational Recommendation

Conversational recommendation aims to collect users’ up-to-date preferences through dialog, instead of relying only on preferences learned offline. However, most existing systems make an unnatural assumption that users’ preferences can only be collected offline or online, and neglect the fact that the knowledge about a user is dynamic and cumulative. To this end, we propose a novel concept called *user memory graph*, which aims to maintain the knowledge about a user in a structured form for interpretability. Each turn of dialog is grounded onto this user memory graph for the reasoning of dialog policy, and more importantly, further accumulation of user knowledge.

Motivation

Traditional recommender systems (such as the collaborative filtering (CF) system) often aim to learn the static correlations between users’ preferences and associated items’ attributes. While it is a powerful approach that can leverage the vast offline user preferences data for effective recommendations, such a system is challenged when operating in the dynamic world, in which new users and items unseen during training frequently appear (so-called cold-start problems). More importantly, static systems fail to capture users’ preferences that may change from time to time.

Conversational recommendation systems [?] are recently introduced to mitigate some of these challenges by tracking users’ up-to-date preferences through dialogs. Most of the previous

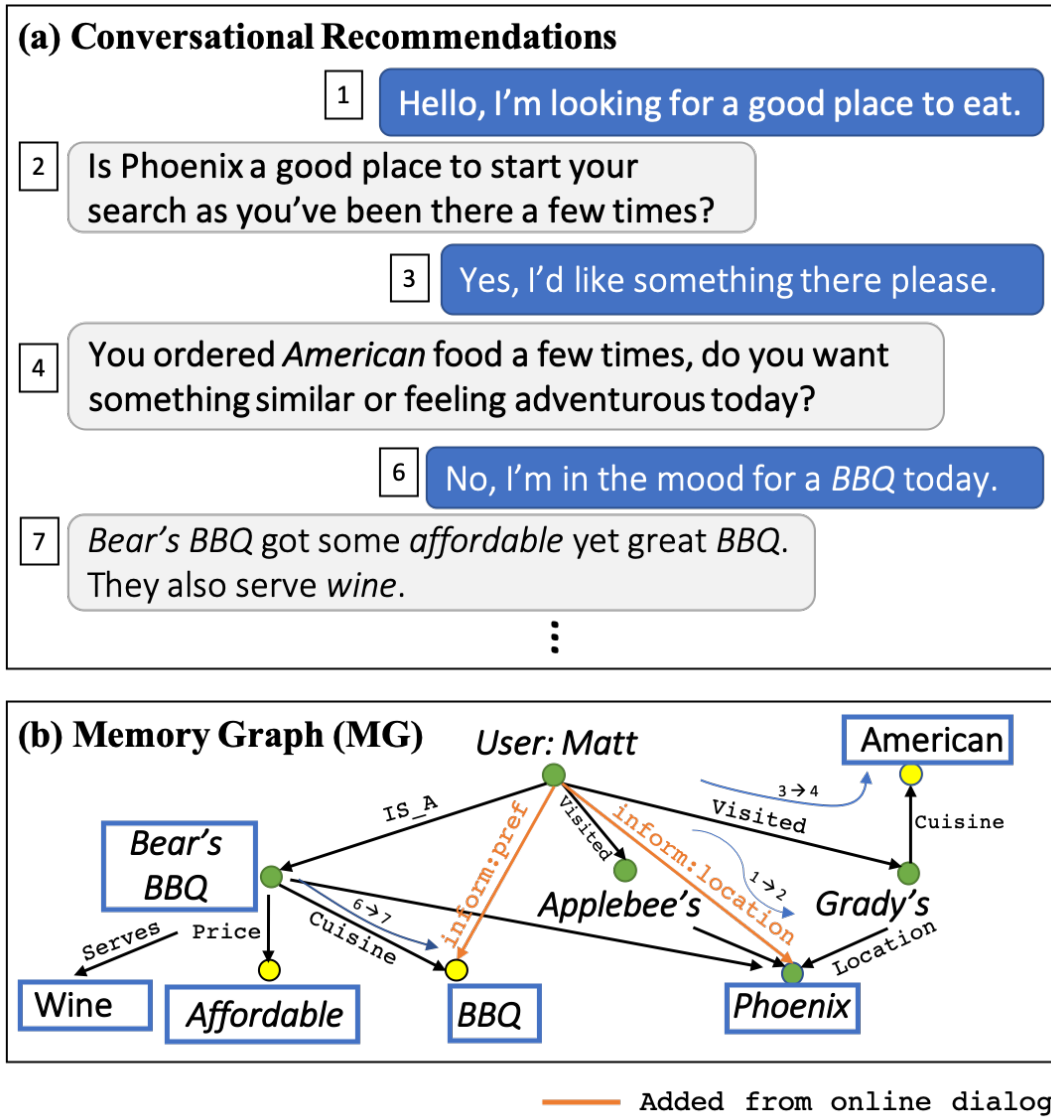


Figure 11. Conceptual illustration of Memory-grounded conversational recommendation

works focus on extending the conventional task-oriented dialog literature with a recommender system, which allows the conversational system to update user preferences online by asking relevant questions (called “System Ask User Respond (SAUR)” for the current dialog.

In summary, existing systems either favor a static offline recommendation over existing users or items or obtain short-term online updates on users’ preferences via dialogs. However, they unnaturally contrast offline with online preference learning and neglect the fact that the knowledge about a user is *cumulative* in nature. An intelligent system should be able to dynamically maintain and utilize knowledge about a user collected so far for recommendations.

To this end, we first introduce a novel concept called *user memory graph* to represent dynamic knowledge about users and associated items in a structured graph (e.g., previous offline history of items visited/recommended, user preferences newly obtained through dialogs, etc.), allowing for easy and holistic reasoning for recommendations. We then propose a new conversational recommendation system grounded onto this graph, conceptually defined more formally as follows:

Memory-grounded Conversational Recommendation: Given the history of previous items \mathcal{H} (interacted or visited, etc.), candidate items \mathcal{C} for recommendation, and their attributes (values), an agent first (1) constructs a user memory graph $\mathcal{G} = \{(e, r, e') | e, e' \in \mathcal{E}, r \in \mathcal{R}\}$ for user e_u ; then (2) for each turn $d \in D$ of a dialog, the agent updates \mathcal{G} with tuples of preference $\mathcal{G}' \leftarrow \mathcal{G} \cup \{(e_u, r_1, e_1), \dots\}$; (3) performs reasoning over \mathcal{G}' to yield a dialog policy π that either (i) performs more rounds of interaction by asking for more preference, or (ii) predicts optimal (or ground truth) items for recommendations $\mathcal{T} \subset \mathcal{C}$.

Related Work

Conversational Recommendation: Much existing research on conversational recommendation focus on combining a recommender system with a dialog state tracking system, through the “System Ask User Respond (SAUR)” paradigm. Once enough user preference is collected, such systems often make personalized recommendations to the user. For instance, [?] proposes to mitigate cold-start users by learning users’ preferences during conversations and by linking the learned preferences to existing similar users in a traditional recommender system.

[?, ?] propose a reinforcement learning (RL) setting for a conversational recommendation system, where the dialog policy is learned with multiple policies and recommendation signals.

[?] leverages reviews to mimic online conversations to update an existing user’s preference and re-rank items.

Task-oriented Dialogue Systems are widely studied with multiple popular benchmark datasets [?, ?, ?, ?, ?]. Most of the state-of-the-art approaches [?, ?, ?] focus on improving dialog state tracking with span-based pointer networks, which predicts information essential in completing a specified task (e.g., hotel booking, etc.)

Note that while conversational recommendation systems bears similarity to task-oriented dialog systems, the key difference is that conversational recommendation aims to collect user’s fine-grained soft preferences or sentiments, and utilize them collectively for ranking of items or asking better questions (policy selection), instead of collecting hard constraints (e.g., number of people, time and location) to filter a database and locate a record.

Graph Reasoning: Graph network [?, ?, ?, ?] is a type of neural networks proposed to operate on graph structures. Several extensions to the original graph neural network have been proposed [?, ?], most notably R-GCNs [?], which can be applied on large-scale and highly multi-relational data. Many applications of GNNs include [?], which introduces graph-based reasoning for an offline recommendation system. A few works have recently been proposed to allow graph reasoning in dialog systems. [?, ?] propose new corpus to learn knowledge graph paths that connect dialog turns. [?] introduces a knowledge-grounded dialog generation task given a knowledge graph that is dynamically updated. However, these works often focus on response generation and do not address the conversational recommendation task.

Definition of Dialog Acts, Slots and Values

As discussed in the introduction, one key step to enable a dialog being grounded and maintained on a user memory graph is to first define the semantic space of dialog acts, items, their slots and values (we borrow these terms from task-oriented dialog system, which refer to items' attributes) for utterances from both the user and agent. As a result, agents can turn unstructured utterances into structured data for user memory graph maintenance, integration and potentially future explainable reasoning for policy. In this section, we first introduce the dialog acts for recommendation and then introduce slots and values specifically defined for the recommendation in a restaurant domain.

Dialog Acts

The goal of designing dialog acts \mathcal{A} is to formalize the intentions from both the user and agent sides. ?? demonstrates the dialog acts for both the user and the agent. From the agent's

perspective, note that although existing conversational recommendation [?, ?, ?] assumes a passive user interacts with the system and propose a System Ask – User Respond (SAUR) paradigm, we further allow the user to actively participate in the recommendation by allowing User Ask - System Respond (UASR) paradigm. In our dialog act, *Open question*, *Yes/no question* and *Inform* can be used by a user to actively participate in the conversation. The dataset we created from crowd workers also indicates that human likes to use these active dialog acts in the context of conversational recommendation (see Appendix).

Slots and Values

This paper focuses on the recommendation in the restaurant domain. We utilize the customer review dataset, which is widely used in existing research in recommender systems. By leveraging the metadata of restaurants, we define slots \mathcal{S} and their values \mathcal{V} as shown in ???. We select $|\mathcal{S}| = 10$ popular slots with rich values that can be encountered in the restaurant domain. We omit the full set of values for brevity and only list a few examples. (Please refer to our dataset for the exhaustive list).

Dataset

Based on the definition in Chapter ??, we create a large-scale dataset called *MGConvRex*. To the best of our knowledge, this is the first dataset for conversational recommendation that is grounded onto structured data of users' profile and items. Although curating a dataset for a task-oriented dialog system may involve building artificial scenarios (a pre-defined setting for collecting a dialog) [?, ?] due to limited access of real-world data for a particular task, conversational recommendation can leverage rich user behaviors that persist in the wild datasets

Dialog Act <i>a</i>	Description	Examples
User-side		
Greeting	Greeting to the agent	I'd like to find a place to eat.
Inform	Actively inform the agent your preference	I'd like to find a <i>thai</i> restaurant .
Answer	Answer to a question from the agent	I prefer <i>thai</i> food.
Reply	Reply to a recommendation	I'll give it a try.
Open question	Actively ask an open question about a recommended item.	What kind of food do they serve ?
Yes/no question	Actively ask an yes/no question about a recommended item.	Do they serve <i>thai</i> food ?
Thanks	Thanks the agent	Thanks for your help.
Agent-side		
Greeting	Greeting to the user.	How may I help you today ?
Open question	Ask an open question about a slot to the user	What kind of food do you prefer ?
Yes/no question	Ask a yes/no question about a value of a slot	I saw you've been to <i>thai</i> restaurant, do you still like that ?
Recommendation	Recommend items to the user.	How about <i>burger king</i> , which serves <i>fast food</i> ?
Answer	Answers user's questions on an item.	They serve <i>thai</i> food.
Thanks	Thanks the user	Enjoy your meal.

TABLE XXVI

Dialog acts for agent and user \mathcal{A}

Slot e_s	Example Value e_v
location	Las Vegas, NV; Toronto, ON
category	fast food; burger; thai
price	cheap; expensive
parking	garage; valet; lot
noise	average; quiet
ambience	classy; intimate
alcohol	full bar; beer and wine
good for meal	brunch; lunch; dinner
wifi	paid; free
attire	casual; formal

TABLE XXVII

Slots \mathcal{S} and values \mathcal{V} .

of recommender system. As a result, we first introduce a simple way to create large-scale scenarios for dialog transcription, as in Chapter ?? . Then we set up a Wizard-of-Oz environment [?, ?, ?, ?] to collect dialogs from crowd workers and further annotate transcribed dialogs based on scenarios, as in Chapter ?? . Our *MGConvRex* can be used for research in almost all crucial components of a dialog system such as natural language understanding, sentiment analysis, dialog state tracking, dialog policy generation, natural language generation, etc.

Scenario Generation

A scenario is a pre-defined user-agent setting to collect a dialog between two crowd workers, where one plays the user and the other plays the agent. Let $\mathbb{B} = \{0, 1\}$ be a binary number.

We define a scenario consisting of the following parts: $(e_u, C, H, V, P, \mathcal{T})$, where e_u is a user, $C \in \mathbb{B}^{|\mathcal{C}| \times |\mathcal{V}|}$ means the candidate items \mathcal{C} and their associated values \mathcal{V} , $H \in \mathbb{B}^{|\mathcal{H}| \times |\mathcal{V}|}$ is about visited items \mathcal{H} and their values user e_u has been to and known to the agent, $V \in \mathbb{B}^{|\mathcal{V}| \times |\mathcal{S}|}$ indicates values with their associated slots, $P \in \mathbb{B}^{|\mathcal{S}| \times |\mathcal{V}|}$ is the user preference (which value the user prefer for a slot) and $\mathcal{T} \subset \mathcal{C}$ is the ground-truth items. Each scenario is constructed in the following way:

- Preprocess reviews to keep users and items (restaurants) with at least 10 reviews (10-core users/items). We further filter out users with more than 100 reviews as they are suspected to be spam reviewers (not real-world users).
- Sort items (of reviews) by time and use a pre-defined timestamp (e.g., 01/01/2014) to separate items into two groups: visited items and future items for all users.
- For each user, random select $|\mathcal{T}| = 1$ ¹ items (with 4 or 5 ratings) as the *ground-truth items* \mathcal{T} . Use the slots / values of the ground-truth items as *user preference* P .
- For each user, negatively sample $|\mathcal{C}| - |\mathcal{T}|$ items and combine them with the ground-truth items \mathcal{T} as *candidate items* \mathcal{C} ¹ from all available items¹.

¹We use 1 ground-truth item to reduce the load of the transcribers and increase the difficulty of reasoning.

¹We choose $|\mathcal{C}| \in [10, 20]$ candidate items.

¹To allow real-world recommendation setting, we ensure certain similarity over candidate items such as all locations are from the same state as the ground-truth items.

Dataset	All Dialogs			Dialogs w/ History		Dialogs w/o History	
	# of Dial.	# of Turns	Avg. # of Turns	# of Dial.	Avg. # of Turns	# of Dial.	Avg. # of Turns
Train	3225	30858	9.57	1570	9.52	1655	9.62
Dev	266	2488	9.35	137	9.18	129	9.53
Test	2078	19818	9.54	982	9.45	1096	9.61

TABLE XXVIII. Statistics of MGConvRex Dataset

- For each user e_u , create two scenarios: one with *visited items* \mathcal{H} and one without. We keep $|\mathcal{H}| \in [5, 20]$ visited items to ensure enough statistical information for a user’s past history.

Wizard-of-Oz Collection

We build a wizard-of-oz system to randomly pair two crowd workers to engage in a chat session, where each scenario is split into two parts: (P, \mathcal{T}) for user and (e_u, C, H, V) for the agent. So in each session, the worker playing the user can see a user’s preference P and ground-truth items \mathcal{T} . The worker playing the agent can only see candidate items C and the user’s visited items H (if a scenario contains that). The user can tell the agent information from preference P via utterance or check whether recommended items $e_i \in \mathcal{T}$ and reply to agent accordingly (they are not allowed to tell the ground-truth directly). The job of a worker playing the agent is trying to guess the ground-truth item $e_t \in \mathcal{T}$, based on the values of the available candidate items C , the current preference collected from the user via dialog, and optionally the user’s visited items H . As a result, the goal of a conversation is like a game between the user and the agent, where the agent needs to guess the user’s current preference and find the ground-truth item.

The collected behavior from the agent side reflects human-level intelligence of reasoning over candidate items for recommendation. After transcribing a dialog, we further ask the workers to rate the whole dialog and each other’s work, where dialogs with ratings lower than 4 are filtered out. Lastly, we annotate dialog acts, items, slots, values and users’ utterance-level and entity-level sentiment for each turn of dialogs. The guidelines, screenshots of the Wizard-of-Oz UI can be found in the Appendix.

Summary of MGConvRex: After annotation, we split the dialogs by their associated scenarios into training, development and test sets. Note that we enforce all sets to have no overlapping on users so that the training cannot carry the knowledge from any particular user into testing. The statistics of MGConvRex can be seen in ??.

Experimental Framework

While there exist many frameworks for task-oriented dialog systems [?, ?, ?] due to its popularity, to the best of our knowledge, there’s no existing framework for conversational recommendation. Hence we first develop a new framework for training, offline and online evaluation of supervised (imitation) learning and reinforcement learning agents. One key component of our framework is the rule-based user simulator, which can be served for both evaluation and training of reinforcement learning agent.

Results

Evaluation Metrics

We propose the following metrics to evaluate UMGR over the MGConvRex dataset both offline (against the collected dialogs) and online (against user simulator).

Offline metrics

We report the following metrics to evaluate the model’s performance on dialog acts prediction, turn-level prediction over entities (items, slots, and values), and dialog-level item prediction.

Act Accuracy & F1 are reported for all dialog acts against turns in the testing set.

Entity Matching Rate (EMR, k@1, 3, 5) (Turn-level): these metrics measure the predicted top- k entities against the annotated test dialogs. Note that the types of predicted entities (items, slots or values) depend on the predicted dialog acts \hat{y}^A , so correctly predicted entities must have correctly predicted dialog acts first.

Item Matching Rate (IMR) (Dialog-level): this measures all predicted items in a dialog against the ground-truth item e_t .

Online metrics

In addition to offline evaluation, we report the following online metric against the user simulator to dynamically test the performance of recommendation. This mitigates an assumption in offline metrics that all past turns (from the human-annotated dialogs) are correct, which limits the interactive evaluation of conversations.

Success Rate: tracks whether the interaction with user simulators yields the ground-truth item e_t . We use the scenarios from the same test-set dialogs used for the offline evaluation. The maximum number of turns is simulated as 11.

Compared Methods

	Offline Evaluation						Online Evaluation
Methods	Act Acc.	Act F1	EMR			IMR	Success Rate
			@1	@3	@5		
RandomAgent	0.1769	0.182	0.0229	0.0229	0.0229	0.052	0.0659
RecAgent	0.2568	0.0681	0.0262	0.0262	0.0262	0.3826	0.3855
Pretrained Emb.	0.2859	0.0741	0.1264	0.2484	0.316	0.0	0.0
UMGR (Proposed)	0.643	0.5534	0.2329	0.4416	0.487	0.5226	0.4315
- No Dialogue Acts	0.3914	0.2137	0.2503	0.4383	0.4777	0.6165	0.4293
- Prev. User Act Only	0.6187	0.5375	0.2255	0.4175	0.4561	0.5693	0.4032
- Static \mathcal{G}	0.6355	0.5452	0.0957	0.2769	0.3494	0.0914	0.11
UMGR w/ History	0.5778	0.4761	0.0769	0.2111	0.2987	0.2872	0.2592
UMGR w/o History	0.6146	0.4575	0.0597	0.1546	0.2498	0.1122	0.1032

TABLE XXIX. Results of UMGR

Our framework implements the following methods:

RandomAgent: As a baseline, we implement an agent that randomly picks a dialog act and randomly pick a candidate item/slot/value to fill the current response to the user.

RecAgent: The agent always chooses *Recommendation* as the dialog act to enact and select a random item that has not been tried from candidate items. This leads to sub-optimal performance as it does not use or collect user preferences.

Pretrained Embeddings: We pre-train the graph embeddings for all entities and relations from the MG across all scenarios in the training set using the TransE-based graph prediction

approaches [?]. We utilize these for prediction of the future item/slot/value without having the R-GCN layers. While this approach is widely used in the related literature and carries cross-scenario knowledge, we show that using pre-trained graph embedding alone is sub-optimal for a particular user and that the dialog policy needs to perform dynamic reasoning over the user memory graph.

UMGR (Proposed): This is the proposed R-GCN based model. We choose the batch size to be 32, all hidden states to be size 64. The number of maximum dialog acts is set to 10. We use 5 layers of R-GCN based on validation on the development set. α, β, γ are set as 10, 10, 100 based on the scales of losses of different types, respectively. We further conduct the following ablation studies.

- **No Dialog Acts:** this study removes the dialog acts encoder, demonstrating the importance of the dialog acts in policy generation.
- **Prev. User Act Only:** this study only uses the most recent dialog act from the user. We use this to show how many past dialog acts are needed for good policy generation.
- **Static \mathcal{G} :** uses the initial user memory graph without making any updates during the conversation. We use this study to demonstrate that dynamic update of the user memory graph is crucial for reasoning better dialog policy.
- **w/ History v.s. - w/o History:** analyzes the effect of the history of visited items \mathcal{H} (the last two dataset folds in ??). We use these two baselines to demonstrate that prior knowledge of user memory history aids in predicting dialog policy.

Result Analysis

The results are shown in ???. From the results, we can see that UMGR achieves good performance for most of the metrics.

UMGR is effective in leveraging knowledge in the user memory graph. While the UMGR model already achieves reasonable accuracy in dialog policy prediction relying just on the user memory graph (-No Dialogue Acts), adding previous dialog act from the user (-Prev. User Act Only) significantly improves the performance. Lastly, we show that keeping user memory graph updated is crucial, as seen in *static G* not providing good rankings for entities.

UMGR vs. Pre-trained Graph Embeddings. We confirm that the static pre-trained graph embeddings provide limited capacity for reasoning over a large-graph across multiple scenarios to learn user-specific dialog policy, leading to poor performance in the recommendation.

w/ vs w/o History. Lastly, the contrasting results for with and without visited items \mathcal{H} in a user memory graph indicate that having more knowledge about a user’s experience is important in conversational recommendation.

We release a novel dataset with *user memory graph* grounding based on scenarios generated from the behaviors of real-world users. The user memory graph has the benefits of both accumulating pieces of knowledge about a user and interpretability. Experimental results on our R-GCN based reasoning model (UMGR) show promising results for dialog acts, items, slots, and values prediction.

CHAPTER 7

CONCLUSION

The paradigm of lifelong learning is essential for learning beyond the classic deep learning approach. Looking forward, the world keeps evolving and yields new data for new tasks, which probably are long-tailed or heavily-tailed. The existing approach may represent the majority general features well and assume they are generally good for any new knowledge. It lacks enough capability to represent the vast kinds of specific features that are required each (new) task. To make the learning effective in the long-term, an AI agent must be able to adapt to the changes in the world. This dissertation explores different forms of lifelong learning tasks, including classification, word embedding, contextualized word embedding, graph reasoning, and NLP applications. However, the research of lifelong learning does not stop just at these formulations. I expect future extended research of lifelong representation learning in the following areas: (1) similarity spaces of neural network that supports lifelong learning; (2) meta-learning over the formulation of lifelong learning tasks; (3) error-robust accumulation of knowledge.

APPENDICES



- [FAQ](#)
- [Corrections](#)
- [Submissions](#)

What is the copyright for materials in the ACL Anthology?

The ACL materials that are hosted in the Anthology are licensed to the general public under a liberal usage policy that allows unlimited reproduction, distribution and hosting of materials on any other website or medium, for non-commercial purposes. Prior to 2016, all ACL materials are licensed using the [Creative Commons 3.0 BY-NC-SA](#) (Attribution, Non-Commercial, Share-Alike) license. As of 2016, this policy has been relaxed further, and all subsequent materials are available to the general public on the terms of the [Creative Commons 4.0 BY](#) (Attribution) license; this means both commercial and non-commercial use is explicitly licensed to all.

Note that these policies only cover ACL materials. As with [the DOIs](#), this policy does not cover third-party materials. For reproduction privileges for such materials, please approach the respective organizations.



ACL materials are Copyright © 1963–2019 ACL; other materials are copyrighted by their respective copyright holders. Materials prior to 2016 here are licensed under the [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 International License](#). Permission is granted to make copies for the purposes of teaching and research. Materials published in or after 2016 are licensed on a [Creative Commons Attribution 4.0 International License](#).

The ACL Anthology is managed and built by the [ACL Anthology team](#) of volunteers.

Site last built on 02 February 2020 at 23:50 UTC with [commit ca34aeb0](#).

ACM Copyright¹

Reuse

Authors can reuse any portion of their own work in a new work of their own (and no fee is expected) as long as a citation and DOI pointer to the Version of Record in the ACM Digital Library are included.

- Contributing complete papers to any edited collection of reprints for which the author is not the editor, requires permission and usually a republication fee.
- Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. Authors can use any portion of their own work in presentations and in the classroom (and no fee is expected).
- Commercially produced course-packs that are sold to students require permission and possibly a fee.

¹<https://authors.acm.org/author-services/author-rights>



RightsLink®



Home



Help



Email Support



Sign in



Create Account



CER: Complementary entity recognition via knowledge expansion on large unlabeled product reviews

Conference Proceedings: 2016 IEEE International Conference on Big Data (Big Data)

Author: Hu Xu

Publisher: IEEE

Date: Dec. 2016

Copyright © 2016, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE

arXiv.org - Non-exclusive license to distribute

The URI <http://arxiv.org/licenses/nonexclusive-distrib/1.0/> is used to record the fact that the submitter granted the following license to arXiv.org on submission of an article:

- I grant arXiv.org a perpetual, non-exclusive license to distribute this article.
- I certify that I have the right to grant this license.
- I understand that submissions cannot be completely removed once accepted.
- I understand that arXiv.org reserves the right to reclassify or reject any submission.

Revision history

2004-01-16 - License above introduced as part of arXiv submission process

2007-06-21 - This HTML page created

[Contact](#)

IJCAI Copyright Letter

Returned Rights

In return for these rights, IJCAI hereby grants to the above authors, and the employers for whom the work was performed, royalty-free permission to:

1. retain all proprietary rights (such as patent rights) other than copyright and the publication rights transferred to IJCAI;
2. personally reuse all or portions of the paper in other works of their own authorship;
3. make oral presentation of the material in any forum;
4. reproduce, or have reproduced, the above paper for the author's personal use, or for company use provided that IJCAI copyright and the source are indicated, and that the copies are not used in a way that implies IJCAI endorsement of a product or service of an employer, and that the copies per se are not offered for sale. The foregoing right shall not permit the posting of the paper in electronic or digital form on any computer network, except by the author or the author's employer, and then only on the author's or the employer's own World Wide Web page or ftp site. Such Web page or ftp site, in addition to the aforementioned requirements of this Paragraph, must provide an electronic reference or link back to the IJCAI electronic server (<http://www.ijcai.org>), and shall not post other IJCAI copyrighted materials not of the author's or the employer's creation (including tables of contents with links to other papers) without IJCAI's written permission;
5. make limited distribution of all or portions of the above paper prior to publication.
6. In the case of work performed under U.S. Government contract, IJCAI grants the U.S. Govern-

ment royalty-free permission to reproduce all or portions of the above paper, and to authorize others to do so, for U.S. Government purposes. In the event the above paper is not accepted and published by IJCAI, or is withdrawn by the author(s) before acceptance by IJCAI, this agreement becomes null and void.

CITED LITERATURE

VITA

NAME: Hu Xu

EDUCATION: Ph.D., Computer Science, University of Illinois at Chicago,
Chicago, Illinois, 2020.

M.Eng., Electronics and Communication Engineering, Peking
University, Beijing, China, 2009.

ACADEMIC EX- Research Assistant, Big Data and Social Computing Lab, De-
PERIENCE: partment of Computer Science, University of Illinois at Chicago,
2015 - 2020.

Research Assistant, Social Media and Data Mining Lab, Depart-
ment of Computer Science, University of Illinois at Chicago,
2017 - 2020.

Teaching Assistant, Department of Computer Science, Univer-
sity of Illinois at Chicago:

- Language and Automata, Fall 2015, Spring/Summer/Fall
2016 and Fall 2017.
- Compiler Design, Spring 2017