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

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LL	Lifelong Learning
CNN	Converlutional Neural Network
BERT	Bidirectional Encoder Representations from Transformers
AE	Aspect Extraction
ASC	Aspect Sentiment Classification
RRC	Review Reading Comprehension
RCRC	Review Conversational Reading Comprehension

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 [1]. 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 [2–4] in computer vision, or pre-trained word embeddings or language models [5, 6] 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 $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$, where the new or $(n+1)$-th task is performed with the help of the knowledge accumulated over the past n tasks [7–9].

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 2, 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 3, 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 4, 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, 5, 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 6, 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 2) and review classification (using methods in 3). Then I focus on tasks in aspect-based sentiment analysis (ABSA). I first describe the usage of domain word embeddings (from 3) and contextualized word representation (from 4) 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 5).

CHAPTER 2

LIFELONG CLASSIFICATION

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). I call this type of classification under the *closed-world assumption* because the classes seen in testing must have appeared in training. However, this assumption is often violated in real-world applications. For example, on a social media site, new topics emerge constantly and in e-commerce, new categories of products appear daily. 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

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* (OWL) [9–11]. 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 [10–13], 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 [12] and DOC [13]). There are incremental learning techniques (e.g., iCaRL [14] and DEN [15]) that can incrementally learn to classify new classes. However, they

miss the capability of rejecting examples from unseen classes. This paper 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 $S = \{c_1, \ldots, c_m\}$ and has an OWL model/classifier for S but is unaware of a set of unseen classes $U = \{c_{m+1}, \ldots\}$ (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 $U = \{c_{m+2}, \ldots\}$) and added to S (now $S = \{c_1, \ldots, c_m, c_{m+1}\}$, 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) [9–11]. 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 [10, 12, 13]. 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 [12, 13] or partial re-training in an incremental manner [10, 11].

Our work is also related to class incremental learning [14–16], where new classes can be added dynamically to the classifier. For example, iCaRL [14] 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) [17], 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 [18–22]. 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 [23–25] (in that we do not require training but need to read training examples), k-nearest neighbors (kNN) (with additional rejection capability, metric learning [26] and learning to vote), and Siamese networks [27–29] (regarding processing a pair of examples). However, all those techniques work in closed-worlds with no rejection capability. Product classification has been studied in [30–35], 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 an unified approach, I proposes an entirely new OWL method based on meta-learning [17–21]. 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., kNN). 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, kNN cannot do rejection on unseen classes.

As an overview, Fig. Figure 1 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 Sec. 2.3. 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 $p(c = 1|x_t, x_{a_{1:k}|x_t,c})$ 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 $p(c = 1|x_t, x_{a_{1:k}|x_t,c})$ as $p(c|x_t, x_{a_{1:k}})$ 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.

As we can see, the proposed framework works like a supervised lazy learning model, such as the k-nearest neighbor (kNN) 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.

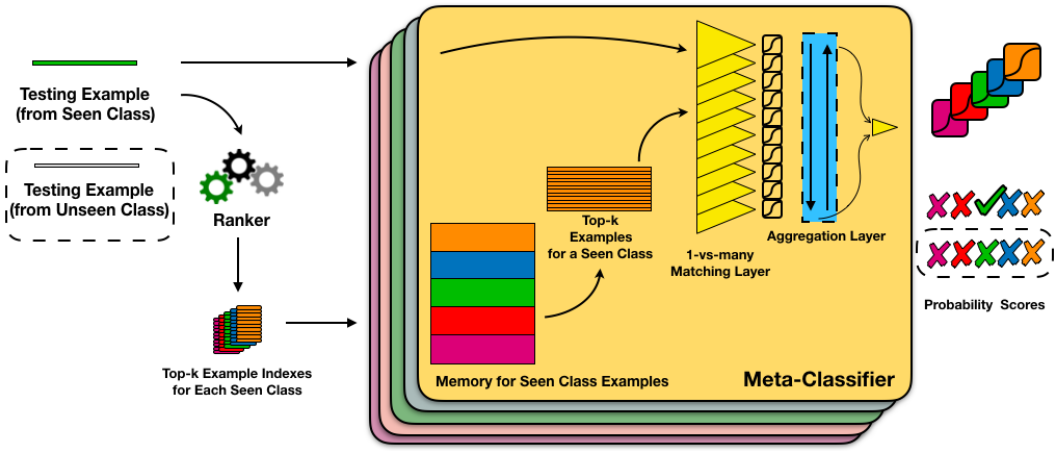

Figure 1. Overview of the L2AC framework

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 Sec. 2.4 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 section, 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 [3]) 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 Sec. 2.4.

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 Fig. Figure 1. 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:

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

Since there are k nearest examples, we have k similarity scores denoted as $r_{1:k}$. The hyper-parameters are detailed in Sec. 2.4.

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 Sec. 2.4.

We adopt a (many-to-one) BiLSTM [36, 37] 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:

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

Inspired by DOC [13], for each class $c \in S$, we evaluate Eq. Equation 2.2 as:

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

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, eq. Equation 2.3 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 [38]. We also enforce $(S \cup U) \cap M = \varnothing$ in Sec. 2.4, 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 $c' \in M \backslash c$ 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* $M' \cap M = \varnothing$ (also $M' \cap (S \cup U) = \varnothing$), to ensure generalization on the seen/unseen classes.

2.4 Results

We want to address the following Research Questions (RQs): **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 [39] 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 $cosine(h_*, h_{a_i}) = \frac{h_* \cdot h_{a_i}}{|h_*|_2 |h_{a_i}|_2}$, 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 Sec. 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 [13], 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 [13].

Besides using macro F1 as used in [13], we also use weighted F1 score overall classes (including seen and the rejection class) as the evaluation metric. Weighted F1 is computed as

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

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 [36, 37] on top of a GloVe [40] embedding (840b.300d) layer as the encoder (other choices are also possible). Similar to feature encoders trained from ImageNet [3], 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 $n = 9$ 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 [41] optimization.

Compared Methods

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-k**5**-n**9**	93.07(0.33)	86.48(0.54)	86.5(0.46)	85.99(0.33)	84.68(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

To the best of our knowledge, DOC [13] is the only state-of-the-art baseline for open-world learning (with rejection) for text classification. It has been shown in [13] that DOC significantly outperforms the methods CL-cbsSVM and cbsSVM in [11] and OpenMax in [12]. 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 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.

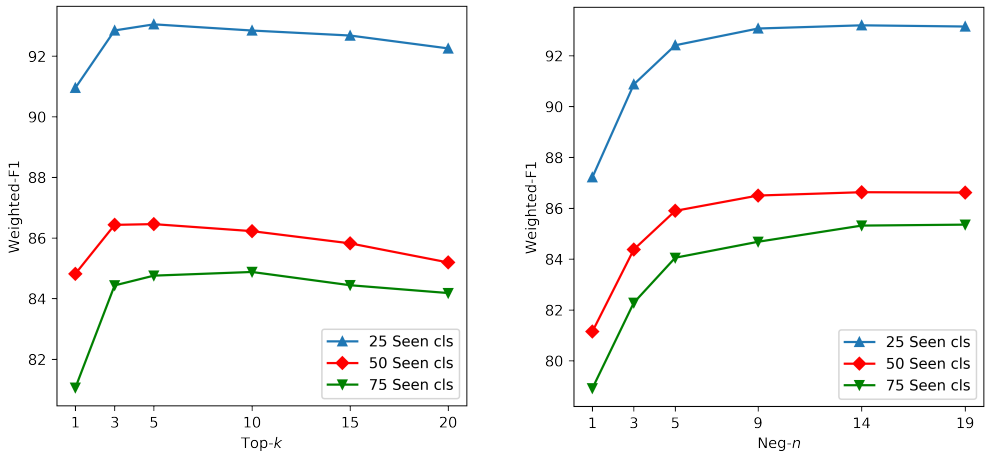

Figure 2. Weighted F1 scores of k and n for OWL

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 [13] 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 [13] 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 kNN) 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 Table I, we can see that L2AC outperforms DOC, especially when the number of seen classes is small. First, from Fig. Figure 2 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 Table Table I, 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

Learning word embeddings [40,42–44] has received a great deal of attention due to its success in numerous NLP applications, e.g., named entity recognition [45], sentiment analysis [46] and syntactic parsing [47]. The key to the success of word embeddings is that a large-scale corpus can be turned into a huge number (e.g., billions) of training examples.

Two implicit assumptions are often made about the effectiveness of embeddings to down-stream tasks: 1) the training corpus for embedding is available and much larger than the training data of the down-stream task; 2) the topic (domain) of the embedding corpus is closely aligned with the topic of the 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-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 [40] 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 [48–50] via transfer learning [51]. 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.

3.2 Lifelong Domain Word Embeddings

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

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

This problem definition is in the *lifelong learning* (LL) setting, where the new or $(n+1)$-th task is performed with the help of the knowledge accumulated over the past n tasks [52]. 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 Embedding via Meta-learning) to tackle the challenges.

To deal with the first challenge, for a word in the new domain, L-DEM learns to identify similar contexts of the word in the past domains. Here the context of a word means the surrounding words of that word in a domain corpus. We call such context *domain context* (of a word). For this, we introduce a multi-domain meta-learner that can identify similar (or relevant) domain contexts that can be later used in embedding learning in the new domain. To tackle the second challenge, L-DEM augments the new domain corpus with the relevant domain contexts (knowledge) produced by the meta-learner from the past domain corpora and uses the combined data to train the embeddings in the new domain. For example, 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 [42]. Many earlier methods used complex neural networks [53]. More recently, a simple and effective unsupervised model called skip-gram (or word2vec in general) [44, 53] 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 [40,54], they are often sub-optimal for domain-specific tasks [48,49,55,56]. However, a single domain corpus is often too small for training high-quality embeddings [56].

Our problem setting is related to *Lifelong Learning* (LL). Much of the work on LL focused on supervised learning [52,57,58]. In recent years, several LL works have also been done for unsupervised learning, e.g., topic modeling [59], information extraction [60] and graph labeling [61]. 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 [51]. Transfer learning has been used in cross-domain word embeddings [48,49]. However, LL is different from transfer learning or multi-task learning [52]. 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 [17] to learn such similarities. Recently, meta-learning has been applied to various aspects of machine learning, such as learning an optimizer [18], and learning initial weights for few-shot learning [20]. The way we use meta-learning is about domain-independent learning [62]. It learns similarities of domain contexts of the same word.

3.3 L-DEM Approach

The proposed L-DEM system is depicted in Figure Figure 6. Given a series of past domain corpora $D_{1:n} = \{D_1, D_2, \ldots, D_n\}$, and a new domain corpus D_{n+1}, the system learns to generate

the new domain embeddings by exploiting the relevant information or knowledge from the past n domains. Firstly, a base meta-learner M is trained from the first m past domains (not shown in the figure) (see Section 4), which is later used to predict the similarities of *domain contexts* of the same words from two different domains. Secondly, assuming the system has seen $n - m$ past domain corpora $D_{m+1:n}$, when a new domain D_{n+1} comes, the system produces the embeddings of the $(n + 1)$-th domain as follows (discussed in Section 5): (i) the base meta-learner first is adapted to the $(n + 1)$-th domain as M_{n+1} (not shown in the figure) using the $(n + 1)$-th domain corpus; (ii) for each word w_i in the new domain, the system uses the adapted meta-learner M_{n+1} to identify every past domain j that has the word w_i with domain context similar to w_i's domain context in the new domain (we simply call such domain context from a past domain *similar domain context*); (iii) all new domain words' similar domain contexts from all past domain corpora $D_{m+1:n}$ are aggregated. This combined set is called the *relevant past knowledge* and denoted by $\mathcal{A}$; (iv) a modified word2vec model that can take both domain corpus D_{n+1} and the relevant past knowledge of $\mathcal{A}$ is applied to produce the embeddings for the $(n + 1)$-th new domain. 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. We discuss the detailed KB content in Section 5.1.

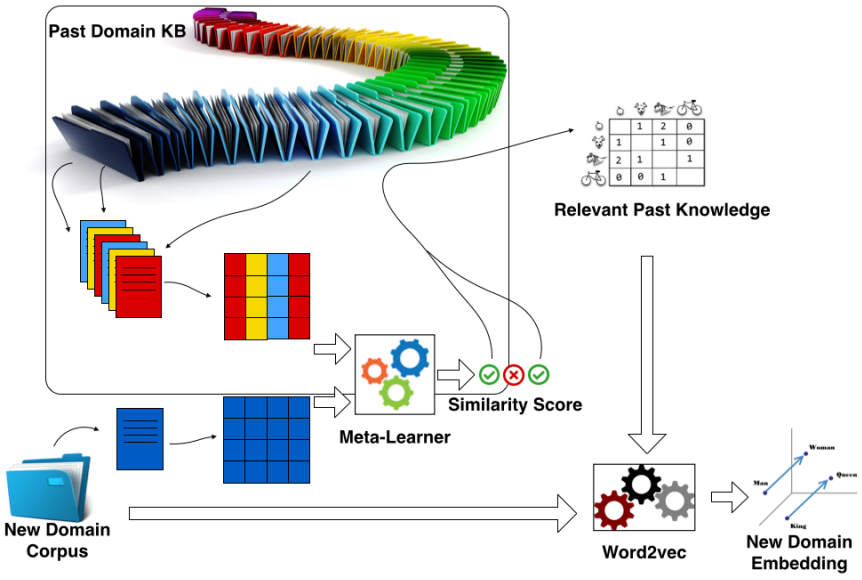

Figure 3. Overview of L-DEM

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

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

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:

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

where $|\cdot|_1$ is the l_1-norm, abs$(\cdot)$ computes the absolute value of element-wise subtraction $(-)$ as the matching function, $\boldsymbol{W}$s and b are weights and $\sigma(\cdot)$ is the sigmoid function. The majority of trainable weights resides in $\boldsymbol{W}_1$, which learns continuous features from the set of f context words. These weights can also be interpreted as a general embedding matrix over V_{wf}. These embeddings (not related to the final domain embeddings in Section 17) 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 1, which finds relevant past knowledge $\mathcal{A}$ (see below) based on the knowledge base (KB) $\mathcal{K}$ and the new domain corpus D_{n+1}.

The KB $\mathcal{K}$ has the following pieces of information: (1) the vocabulary of top-f frequent words $\mathcal{K}.V_{wf}$ (as discussed in Section 4.1), (2) the base meta-learner $\mathcal{K}.M$ (discussed in Section 4.2), and (3) domain knowledge $\mathcal{K}_{m+1:n}$. The domain knowledge has the following information: (i) the vocabularies $V_{m+1:n}$ of past $n - m$ domains, (ii) the sets of past word domain contexts $C_{m+1:n}$ from the past $n-m$ domains, where each C_j is a set of key-value pairs $(w_{i,j}, \mathcal{C}_{w_{i,j}})$ and $\mathcal{C}_{w_{i,j}}$ is a list of context words (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 1 retrieves the past domain knowledge in line 1. Lines 2-4 prepare the new domain knowledge. The BuildFeatureVector function produces a set of feature vectors as $E_{n+1} = \{\mathbf{x}_{w_{i,n+1,k}} | w_i \in V_j \text{ and } k = \{1, 2\}\}$ over two sub-corpora of the new domain corpus D_{n+1}. The ScanContextWord function builds a set of key-value pairs, where the key is a word from the new domain $w_{i,n+1}$ and the value $\mathcal{C}_{w_{i,n+1}}$ is a list of context words for the word $w_{i,n+1}$ from the new domain corpus. We use the same size of 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 $(\mathbf{x}_{w_{i,n+1,1}}, \mathbf{x}_{w_{i,n+1,2}})$ in the same way as described in Section

4.1. A negative example is exampled as $(\mathbf{x}_{w_{i,n+1,1}}, \mathbf{x}_{w_{i,j,2}})$, where $m+1 \leq j \leq n$. The initial weights of M_{n+1} are set as the trained weights of the base meta-learner M.

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.

$(V_{m+1:n}, C_{m+1:n}, E_{m+1:n}) \leftarrow \mathcal{K}_{m+1:n}$

$V_{n+1} \leftarrow$ BuildVocab(D_{n+1})

$C_{n+1} \leftarrow$ ScanContextWord(D_{n+1}, V_{n+1})

$E_{n+1} \leftarrow$ BuildFeatureVector$(D_{n+1}, \mathcal{K}.V_{wf})$

$M_{n+1} \leftarrow$ AdaptMeta-learner$(\mathcal{K}.M, E_{m+1:n}, E_{n+1})$

$\mathcal{A} \leftarrow \emptyset$

for $(V_j, C_j, E_j) \in (V_{m+1:n}, C_{m+1:n}, E_{m+1:n})$ **do**

  $O \leftarrow V_j \cap V_{n+1}$

  $F \leftarrow \{(\mathbf{x}_{o,j,1}, \mathbf{x}_{o,n+1,1}) |$ $o \in O$ and $\mathbf{x}_{o,j,1} \in E_j$ and $\mathbf{x}_{o,n+1,1} \in E_{n+1}\}$

  $S \leftarrow M_{n+1}$.inference(F)

  $O \leftarrow \{o | o \in O$ and $S[o] \geq \delta\}$

  for $o \in O$ **do**

    $\mathcal{A}[o]$.append$(C_j[o])$

  end

end

$\mathcal{K}_{n+1} \leftarrow (V_{n+1}, C_{n+1}, E_{n+1})$

return $\mathcal{A}$

Retriving Relevant Past Knowledge

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

Augmented Embedding Training

We now produce the new domain embeddings via a modified version of the skip-gram model [44] that can take both the new domain corpus D_{n+1} and the relevant past knowledge $\mathcal{A}$. Given a new domain corpus D_{n+1} with the vocabulary V_{n+1}, the goal of the skip-gram model is to learn a vector representation for each word $w_i \in V_{n+1}$ in that domain (we omit the subscript $_{n+1}$ in $w_{i,n+1}$ for simplicity). Assume the domain corpus is represented as a sequence of

words $D_{n+1} = (w_1, \ldots, w_T)$, the objective of the skip-gram model maximizes the following log-likelihood:

$$\begin{aligned}\mathcal{L}_{D_{n+1}} = \sum_{t=1}^{T} \Big(\sum_{w_c \in \mathcal{W}_{w_t}} \big(\log \sigma(\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_c}) \\ + \sum_{w_{c'} \in \mathcal{N}_{w_t}} \log \sigma(-\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_{c'}}) \big) \Big),\end{aligned} \tag{3.2}$$

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; $\boldsymbol{u}$ and $\boldsymbol{v}$ are word vectors (or embeddings) we are trying to learn. The objective of skip-gram on data of relevant past knowledge $\mathcal{A}$ is as follows:

$$\begin{aligned}\mathcal{L}_{\mathcal{A}} = \sum_{(w_t, \mathcal{C}_{w_t}) \in \mathcal{A}} \Big(\sum_{w_c \in \mathcal{C}_{w_t}} \big(\log \sigma(\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_c}) \\ + \sum_{w_{c'} \in \mathcal{N}_{w_t}} \log \sigma(-\boldsymbol{u}_{w_t}^T \cdot \boldsymbol{v}_{w_{c'}}) \big) \Big).\end{aligned} \tag{3.3}$$

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

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

We use the default hyperparameters of skip-gram model [44] to train the domain embeddings.

3.4 Results

Following [63], we use the performances of down-stream tasks to evaluate the proposed method. We do not evaluate the learned embeddings directly as in [40, 44] 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 [64], 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.

	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

So the validation and testing domain corpora have no overlap with the training domain corpora. We sample 2 sub-corpora for each domain and limit the size of each sub-corpus to 10 MB. We randomly select 2000, 500, 1000 words from each training domain, validation domain, and testing domain, respectively, and ignore words with all-zero feature vectors to obtain pairwise examples. The testing 1000 words are randomly drawn and they have 30 overlapping words with the training 2000 words, but not from the same domains. So in most cases, it's testing the unseen words in unseen domains. We set the size of a context window to be 5 when building feature vectors. This ends up with 80484 training examples, 6234 validation examples, and 20740 test examples. For comparison, we train 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 1, which roughly doubled the number of training examples from the new domain corpus. The size of the context window for building domain context is set to 5, which is the same as word2vec.

Baselines and Our System

Unless explicitly mentioned, the following embeddings have 300 dimensions, which are the same size as many pre-trained embeddings (GloVec.840B [40] or fastText English Wiki [54]).

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 [54]. 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 [36] 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 [41] 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

3.5.1 – Motivation

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. In 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.

3.5.2 – 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 $\mathbf{x} = (x_1, \ldots, x_n)$. 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). And the CNN filters will adjust to the new features accordingly. But the embeddings of unseen words from test data still have the old features that may be mistakenly extracted by CNN. Then 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.

3.5.3 – Result

We conducted experiments on two settings, one is in the same setting as for lifelong domain embeddings [65]; the other is for a sequence labeling task in sentiment analysis. We detail the architecture for aspect extraction later in 6.

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 3.5.3. 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)
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

	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

Beyond word embeddings that only carry independent word-level features, the meaning (thus feature) 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 [5], GPT/GPT2 [66, 67], BERT [6], XLNet [68], RoBERTa [69], ALBERT [70], ELECTRA [71]. The idea behind the progress is that even though the word embedding [40, 44] 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 [6] 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 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 [6].) 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: $\textbf{BERT}_{\textbf{BASE}}$: 12 layers, 768 hidden dimensions and 12 attention heads (in transformer) with the total number of parameters, 110M;
$\textbf{BERT}_{\textbf{LARGE}}$: 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 $(\texttt{[CLS]}, x_{1:j}, \texttt{[SEP]}, x_{j+1:n}, \texttt{[SEP]})$,

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 $\mathcal{L}_{\text{MLM}}$ and the loss function of next text piece prediction be $\mathcal{L}_{\text{NSP}}$, 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 $\mathcal{L}_{\text{MRC}}$, which is in a similar setting as the loss $\mathcal{L}_{\text{RRC}}$ 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 : $\mathcal{D}_{\text{DK}}$: one batch of DK data;

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

u: number of sub-batches.

$\nabla_{\Theta}\mathcal{L} \leftarrow 0$

$\{\mathcal{D}_{\text{DK},1}, \ldots, \mathcal{D}_{\text{DK},u}\} \leftarrow \text{Split}(\mathcal{D}_{\text{DK}}, u)$

$\{\mathcal{D}_{\text{MRC},1}, \ldots, \mathcal{D}_{\text{MRC},u}\} \leftarrow \text{Split}(\mathcal{D}_{\text{MRC}}, u)$

for $i \in \{1, \ldots, u\}$ **do**

  $\mathcal{L}_{\text{partial}} \leftarrow \frac{\mathcal{L}_{\text{DK}}(\mathcal{D}_{\text{DK},i}) + \mathcal{L}_{\text{MRC}}(\mathcal{D}_{\text{MRC},i})}{u}$

  $\nabla_{\Theta}\mathcal{L} \leftarrow \nabla_{\Theta}\mathcal{L} + \text{BackProp}(\mathcal{L}_{\text{partial}})$

end

$\Theta \leftarrow \text{ParameterUpdates}(\nabla_{\Theta}\mathcal{L})$

Algorithm 1 describes one training step and takes one batch of data on domain knowledge (DK) $\mathcal{D}_{\text{DK}}$ and one batch of MRC training data $\mathcal{D}_{\text{MRC}}$ 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.

Post-training datasets

For domain knowledge post-training, we use Amazon laptop reviews [64] 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 [72]. 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 [73] that comes with 87,599 training examples from 442 Wikipedia articles.

To evaluate the performance of the post-trained model, I conducted 3 tasks on reviews with two on aspect-based sentiment analysis and one on review reading comprehension. The experiments are discussed in 6 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 [74], 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:

$$\begin{aligned}(&\texttt{[CLS]}\,\texttt{[Q]}\,q_1\,\texttt{[A]}\,a_1 \ldots \texttt{[Q]}\,q_{k-1}\,\texttt{[A]}\,a_{k-1}\\ &\texttt{[Q]}\,q_k\,\texttt{[SEP]}\,d_{1:n}\,\texttt{[SEP]}),\end{aligned}$$

where past QA pairs $q_1, a_1, \ldots, q_{k-1}, a_{k-1}$ 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 $(\text{RC})_2$ 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 $(\text{RC})_2$.

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 $(\text{RC})_2$ data, we harvest domain QA pairs and reviews (that are largely available online, see Section), 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 3.

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.

$\mathcal{T} \leftarrow \{\}$

for $(q', a') \in \mathcal{Q}$ **do**

  $x \leftarrow$ `[CLS]`

  $h \leftarrow \text{RandInteger}([0, h_{\text{max}}])$

  for $1 \rightarrow h$ **do**

    $q'', a'' \leftarrow \text{RandSelect}(\mathcal{Q} \backslash (q', a'))$

    $x \leftarrow x \oplus$ `[Q]` $\oplus\, q'' \oplus$ `[A]` $\oplus\, a''$

  end

  $x \leftarrow x \oplus$ `[Q]` $\oplus\, q'$ `[SEP]`

  $r_{1:m} \leftarrow \text{RandSelect}(\mathcal{R})$

  if *RandFloat*$([0.0, 1.0]) > 0.5$ **then**

    $(\_, a) \leftarrow \text{RandSelect}(\mathcal{Q} \backslash (q', a'))$

    $(u, v) \leftarrow (1, 1)$

  end

  else

    $a \leftarrow a'$

    $(u, v) \leftarrow (|x|, |x| + |a|)$

  end

The inputs to Algorithm 3 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 Section 26). 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 $\mathcal{Q}\backslash(q', a')$ 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 3 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.

Auxilary 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 [75], so we omit them for brevity. After pre-tuning, we fine-tune using the $(\text{RC})_2$ 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, the knowledge graph is also important data that aims to provide support for reasoning and structured interpretation. 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 [76,77] is a novel type of task that allows using an interactive dialogue 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 dialogue between the AI agent and a user. More

importantly, 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 dialogue policy.

5.2 Lifelong Knowledge Graph Reasoning

As an example of lifelong representation learning over a dynamic knowledge graph, we propose the following task:

Memory-grounded Conversational Recommendation[1]: 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 dialogue, 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 dialogue 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 dialogue (the formulation of a scenario in conversational recommendation can be found in **??**). There are many design choices of constructing a user memory graph and our goal is to

[1] **We omit details of graph construction here for brevity and describe details in Section 5.2.**

model the graph with easy extensibility, maintenance and interpretability for the generation of dialogue 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.

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 Table Table XIX
$\mathcal{V}$	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

We first define the entity sets and relations in Table Table V. To illustrate the construction of a user memory graph and its maintenance, we describe an example in Figure Figure 4. 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 Figure **??**, 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 $(e_{\text{Bob}} \rightarrow r_{\text{visited}} \rightarrow e_{\text{Seas}} \rightarrow r_{\text{has\_aspect}} \rightarrow e_{\text{affordable}} \rightarrow r_{\text{has\_aspect}} \rightarrow e_{\text{Basil}})$ 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 dialogue 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 dialogue policy generation (Section 5.3).

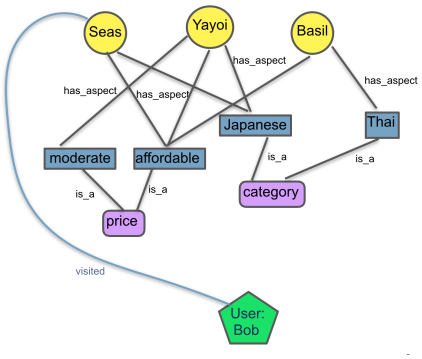

Figure 4. Construction of user memory graph

5.3 Graph Reasoner

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

Input: the input of UMGR is the past dialogue acts up to the current turn from the user $\boldsymbol{a}$, 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 dialogue and then associate all the new relations to that memory entity.).

Output: UMGR's output is the dialogue 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 dialogue 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 $\hat{y}^{\mathcal{A}} =$ *Recommendation*, the top-1 entity $\arg\max_{e_i \in \mathcal{C}}(\hat{y}^{\mathcal{C}})$ will be provided to the user. Similarly, $\hat{y}^{\mathcal{A}} =$ *Open Question* is related to the top-1 slot $\arg\max_{e_s \in \mathcal{S}}(\hat{y}^{\mathcal{S}})$ and $\hat{y}^{\mathcal{A}} =$ *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 dialogue 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) [78] inside UMGR. R-GCN is a GCN [79] with typed relations, where each relation is associated with their weights to enable reasoning over a heterogeneous graph. UMGR first encodes past dialogue acts $\boldsymbol{a}$ and entities $e \in \mathcal{E}$ into hidden dimensions.

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

where $W^{\mathcal{A}}$ and $W^{\mathcal{E}}$ are embedding layers and the past dialogue 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 `<ITEM>`) in $W^{\mathcal{E}}$. Then each entity in the user memory graph is encoded by multiple layers of R-GCN.

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

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.

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

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 dialogue acts, which is a classification problem. The loss for dialogue acts is defined as

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

where $\oplus$ is the concatenation operation and $y^{\mathcal{A}}$ is the annotated dialogue 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

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

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

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

where α, β and γ are hyper-parameters to align losses of different scales. Note that during training and prediction, all invalid dialogue acts (e.g., user dialogue 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 dialogue policy on-the-fly in each turn based on the updated user memory graph.

CHAPTER 6

NLP APPLICATIONS

In this chapter, I switch to NLP applications that leverage the concept of lifelong representation learning. I will first focus on tasks in aspect-based sentiment analysis: aspect extraction and aspect sentiment classification. Then I will discuss its application in question answering. I will propose some novel review-based QA tasks, with results indicating the importance of lifelong representation. Next, I will switch to the dialogue system. I will first talk about the conversational version of QA and then switch to a novel type of dialogue system: conversational recommendation, which leverages lifelong graph representation learning for reasoning dialogue policy.

6.1 Sentiment Analysis

Sentiment analysis aims to detect people's polarity from opinion text [80]. More specifically aspect-based sentiment analysis (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

One key task of fine-grained sentiment analysis of product reviews is to extract product aspects or features that users have expressed opinions on. This paper focuses on supervised aspect extraction using deep learning. Unlike other highly sophisticated supervised deep

learning models, this paper proposes a novel and yet simple CNN model employing two types of pre-trained embeddings for aspect extraction: general-purpose embeddings and domain-specific embeddings. Without using any additional supervision, this model achieves surprisingly good results, outperforming state-of-the-art sophisticated existing methods. To our knowledge, this paper is the first to report such a double embeddings based CNN model for aspect extraction and achieve very good results.

Aspect extraction is an important task in sentiment analysis [81] and has many applications [80]. 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 [82–84] and unsupervised approaches [81, 85–89]. Recently, supervised deep learning models achieved state-of-the-art performances [90]. Many of these models use handcrafted features, lexicons, and complicated neural network architectures [90–93]. 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 [94], 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 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 [40], or a general review embedding [91]. 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) [95] model for sequence labeling. Although most existing models use LSTM [36] as the core building block to model sequences [90,96], we noticed that CNN is also successful in many NLP tasks [97–99]. 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 [1]. 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 discuss the solutions in Section **??**.

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 [80,100,101]. This work focuses on the aspect level [81]. 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 [81,102], syntactic rules-based extraction [85,87,103], topic modeling [86,104–106], word alignment [107] and label propagation [61,108].

Traditionally, the supervised approach [82,84,109] uses Conditional Random Fields (CRF) [110]. Recently, deep neural networks are applied to learn better features for supervised aspect extraction, e.g., using LSTM [36,96,111] and attention mechanism [89,93] together with manual features [91,92]. Further, [90,92,93] 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 [96], where only the annotated data

[1] We notice that a GPU with more cores has no training time gain on a low-dimensional LSTM because extra cores are idle and waiting for the other cores to sequentially compute cells.

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 [50, 112] and domain-specific embeddings [56, 65]. However, we require the domain of the domain embeddings must exactly match the domain of the aspect extraction task. CNN [95, 97] is recently adopted for named entity recognition [113]. CNN classifiers are also used in sentiment analysis [91, 114]. 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 Figure Figure 6. 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.

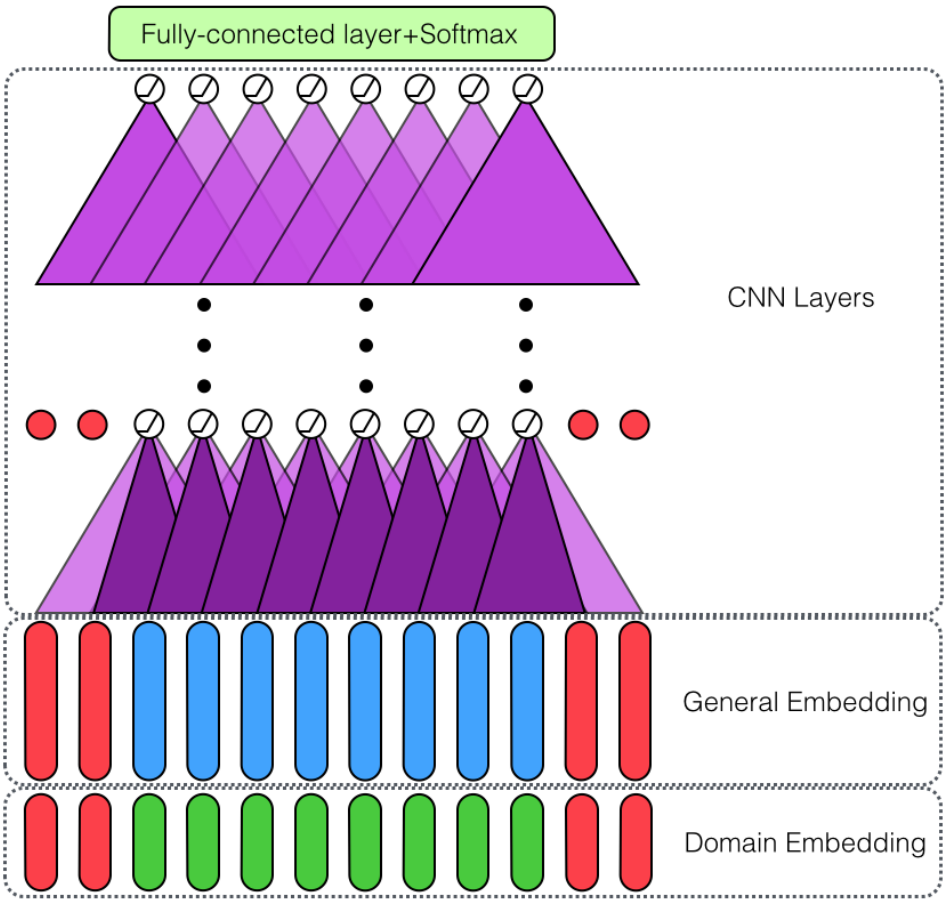

Figure 6. DE-CNN

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:

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

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 next section).

Aspect Extraction from Pre-trained Language Model

Further, based on the technique of post-training in **??**, we can also use the weights of pre-train or post-training for aspect extraction with an extra layer of token type classification.

We only extend BERT with one extra task-specific layer and fine-tune BERT on each end task. This can be illustrated in the second sub-figure in Figure 7.

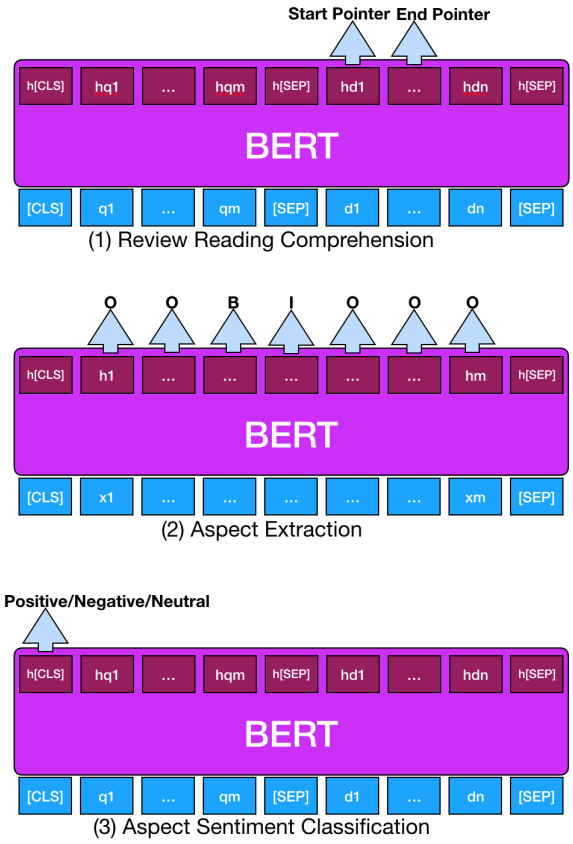

Figure 7. BERT for end tasks

The input sentence with m words is constructed as $x = (\texttt{[CLS]}, x_1, \ldots, x_m, \texttt{[SEP]})$. 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 $W_3 \in \mathbb{R}^{3*r_h}$ 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 $l_3 \in [0,1]^{3*|x|}$. 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.

AE is a task that requires intensive domain knowledge (e.g., knowing that "screen" is a part of a laptop). Previous study [72] has shown that incorporating domain word embeddings greatly improve the performance. Adapting BERT's general language models to domain reviews is crucial for AE, as shown in Sec.

Datasets

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

TABLE VI

Dataset for AE

Results

Following the experiments of a recent aspect extraction paper [90], we conduct experiments on two benchmark datasets from SemEval challenges [115,116] as shown in Table 6.1.1. 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 NLTK[1] to tokenize each sentence into a sequence of words.

For the general-purpose embeddings, we use the glove.840B.300d embeddings [40], 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 [54] to train domain embeddings. The laptop review corpus contains all laptop reviews from the Amazon Review Dataset [39]. The restaurant review corpus is from the Yelp Review Dataset Challenge [2]. We only use reviews from restaurant categories that the second dataset is selected from [3]. 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

---

[1] http://www.nltk.org/

[2] https://www.yelp.com/dataset/challenge

[3] http://www.cs.cmu.edu/~mehrbod/RR/Cuisines.wht

evaluation of the datasets[1] [2]. The results of the first two groups are copied from [90]. The first group uses single-task approaches.

CRF is conditional random fields with basic features[3] and GloVe word embedding [40].

IHS_RD [83] and **NLANGP** [117] are best systems in the original challenges [115, 116].

WDEmb [88] enhanced CRF with word embeddings, linear context embeddings and dependency path embeddings as input.

LSTM [90, 96] is a vanilla BiLSTM.

BiLSTM-CNN-CRF [118] is the state-of-the-art from the Named Entity Recogntion (NER) community. We use this baseline[4] 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 [92] 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.

---

[1] http://alt.qcri.org/semeval2014/task4

[2] http://alt.qcri.org/semeval2016/task5

[3] http://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html

[4] https://github.com/UKPLab/emnlp2017-bilstm-cnn-crf

CMLA [93] 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 [90] 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.

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 embeddings[1], 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 layer[2]. 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 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 [41] is 0.0001 because CNN training tends to be unstable.

---

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

[2] `https://github.com/allenai/allennlp`

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-OOD-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

F_1 score for AE

Results and Analysis

Table 6.1.1 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 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 [119], 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 $\textbf{BERT}_{\textbf{BASE}}$ (uncased) as the basis for all experiments[1]. Since post-training may take a large footprint on GPU memory (as BERT pre-training), we leverage FP16 computation[2] to reduce the size of both the model and hidden representations of data. We set a static loss scale of 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

---

[1]We expect $\textbf{BERT}_{\textbf{LARGE}}$ to have better performance but leave that to future work due to limited computational power.

[2]`https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html`

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 Section 4.2.1 and then fine-tunes on the 3 end tasks.

Discussion of Results

Domain	**Laptop**	**Rest.**
Methods	**F1**	**F1**
DE-CNN [72]	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 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 $x = (\texttt{[CLS]}, q_1, \ldots, q_m, \texttt{[SEP]}, d_1, \ldots, d_n, \texttt{[SEP]})$, where $q_1, \ldots, q_m$ now is an aspect (with m tokens) and $d_1, \ldots, d_n$ is a review sentence containing that aspect. After $h = \text{BERT}(x)$, we leverage the representations of `[CLS]` $h_{\text{[CLS]}}$, which is the aspect-aware representation of the whole input. The distribution of polarity is predicted as $l_4 = \text{softmax}(W_4 \cdot h_{\text{[CLS]}} + b_4)$, where $W_4 \in \mathbb{R}^{3*r_h}$ 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 Sec. 6.4.2 that BERT's weights only result in a limited gain or worse performance compared with existing baselines. In the next section, we introduce the post-training step to boost the performance of all these 3 tasks.

datasets

For ASC, we use SemEval 2014 Task 4 for both laptop and restaurant as existing research frequently uses this version. We use 150 examples from the training set of all these datasets for validation.

Domain	**Laptop**		**Rest.**	
Methods	**Acc.**	**MF1**	**Acc.**	**MF1**
MGAN [120]	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 IX

ASC in Accuracy and Macro-F1(MF1).

Compared Methods and Evaluation Metrics

MGAN [120] 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 [121], examples belonging to the *conflict* polarity are dropped due to a very small number of examples.

Results Discussion

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 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.

6.1.3 Hard Examples Learning for Aspect Sentiment Classification

Aspect-based sentiment classification (ASC) is an important task in fine-grained sentiment analysis. One challenge, however, is the hard examples in ASC datasets that are typically rare but very important for learning aspect-level sentiment (e.g., sentences with different polarities for different aspects). This paper focuses on learning hard examples for ASC and proposes a simple adaptive re-weighting (ARW) scheme to dramatically improve ASC for such complex sentences. Experimental results show that ARW is effective [1].

Introduction

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 [2]. 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

---

[1] The dataset and code will be released for future research.

[2] https://venturebeat.com/2019/01/24/amazon-rekognition-bias-mit/

problem [122, 123] 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) [124, 125]. 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 [126] found that such sentences are rare but hard to learn from 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 [122, 123], which aims to detect long-tailed and imbalanced classes of sub-regions in one image. In [123], a

loss-based weighting is proposed to adjust weights without explicitly re-balance the complex class distribution.

Aspect sentiment classification (ASC) [124] is an important task in sentiment analysis [125, 127]. It is different from document or sentence-level sentiment classification (SSC) [97, 127–129] 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 [90, 93, 130]. The problem has been widely dealt with using neural networks [131–133]. 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) [134, 135] and transferring from other tasks/domains [120, 136, 137]. Our re-weighting method is related to AdaBoost [138], 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 [139, 140] 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) [141]. 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 $i \in [1, n]$, 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

Algorithm 4: ARW Algorithm

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

e: maximum number of epochs.

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

$w_{1:n} \leftarrow \frac{1}{n}$

for *epoch* $\in \{1, \ldots, e\}$ **do**

  for $(a^b, x^b, y^b, w^b) \in$ *Batchify*$(\mathcal{D}_{tr}, w_{1:n})$ **do**

    $l^b \leftarrow \text{CrossEntropy}(p_\theta(\hat{y}^b|a^b, x^b), y^b)$

    $L^b \leftarrow \frac{\sum(w^b \cdot l^b)}{\sum w^b}$

    BackProp&ParamUpdate(L, M)

  end

  $\hat{y}_{1:n} \leftarrow \arg\max p_\theta(\hat{y}_{1:n}|a_{1:n}, x_{1:n})$

  $r \leftarrow \frac{\sum_{i=1}^{n}(w_i \mathbb{I}[y_i \neq \hat{y}_i])}{\sum_{i=1}^{n} w_i}$

  $\alpha \leftarrow \log(\frac{(1-r)+\epsilon}{r-\epsilon})$

  $w_{1:n} \leftarrow w_{1:n} \exp(\alpha \mathbb{I}[y_{1:n} \neq \hat{y}_{1:n}])$

end

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 4. 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 4 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 4 for brevity).

Experiment

Dataset

We adopt the SemEval 2014 Task 4[1] datasets, which contain two domains: *laptop* and *restaurant*. The statistics are shown in Table **??**. 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.

---

[1]http://alt.qcri.org/semeval2014/task4

Baselines

We evaluate all baselines on both accuracy (Acc.) and macro F1 (MF1) and adopt the following baselines: RAM [142][1], AOA [143], MGAN [120], TNET [133]. BERT-DK [134][2]. For the last model, we further challenge it by removing the aspects from the testing examples as there is 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 $(1 - p)^\gamma$ [123], 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.

---

[1]The first 4 baselines are adopted from `https://github.com/songyouwei/ABSA-PyTorch`.

[2]`https://github.com/howardhsu/BERT-for-RRC-ABSA`

+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

From Table **??**, 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.

Conclusion

This paper focuses on hard example learning for Aspect-based sentiment classification (ASC). We proposed a simple ARW algorithm to dramatically improve ASC for hard examples and using contrastive sentences to test the effectiveness of hard example learning.

6.2 Complementary Entity Recognition

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 dialogue 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 [144–147]. As a compromise, many online businesses leverage community question-answering (CQA) [148] 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 [148, 149] 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) [73,75], we propose a novel task called Review Reading Comprehension (RRC) as following.

Problem Definition: Given a question $q = (q_1, \ldots, q_m)$ from a customer (or user) about a product and a review $d = (d_1, \ldots, d_n)$ for that product containing the information to answer q, find a sequence of tokens (a text span) $a = (d_s, \ldots, d_e)$ 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
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 .

TABLE XII

Review reading comprehension

A sample *laptop* review is shown in Table Table XII. 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 [56, 148].

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 5[1], which is a popular dataset for aspect-based sentiment analysis (ABSA) [124] in the domains of *laptop* and *restaurant*. We detail ReviewRC in Sec. 6.4.2. 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 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, as shown in Sec.

---

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

Related Works

Many datasets have been created for MRC from formally written and objective texts, e.g., Wikipedia (WikiReading [150], SQuAD [73, 75], WikiHop [151], DRCD [152], QuAC [153], HotpotQA [154]) news and other articles (CNN/Daily Mail [155], NewsQA [156], RACE [157]), fictional stories (MCTest [158], CBT [159], NarrativeQA [160]), and general Web documents (MS MARCO [161], TriviaQA [162], SearchQA [163]). Also, CoQA [74] 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 [73, 75]) rather than abstractive (or generative) (such as in MS MARCO [161] and CoQA [74]). 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 [148] 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 [148, 149], 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 [144, 164, 165] or DBpedia [166, 167]) have been used for question answering [149]. 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 [124, 125, 127, 168]. 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) [124], 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 [72, 92, 93, 121, 169] 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 enough[1] and automated representation learning from review corpora is always preferred [72, 169]. We push forward this trend with the recent advance in pre-trained language models from deep learning [5, 6, 66, 170, 171]. Although it is practical to train domain word embeddings from scratch on large-scale review corpora [72], it is impractical to train language models from scratch

---

[1] `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`

with limited computational resources. As such, we show that it is practical to adapt language models pre-trained from formal texts to domain reviews.

6.3.2 – Review Reading Comprehension (RRC)

Following the success of SQuAD [73] and BERT's SQuAD implementation, we design review reading comprehension as follows. Given a question $q = (q_1, \ldots, q_m)$ asking for an answer from a review $d = (d_1, \ldots, d_n)$, we formulate the input as a sequence $x = (\texttt{[CLS]}, q_1, \ldots, q_m, \texttt{[SEP]}, d_1, \ldots, d_n, \texttt{[SEP]})$ where `[CLS]` is a dummy token not used for RRC and `[SEP]` is intended to separate q and d. Let 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 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: $s = \arg\max_{\text{Idx}_{\texttt{[SEP]}} < s < |x|}(l_1)$ and $e = \arg\max_{s \leq e < |x|}(l_2)$, where $\text{Idx}_{\texttt{[SEP]}}$ 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 $a = (d_s, \ldots, d_e)$.

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.

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 in

Results

Datasets

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 [73].

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 [73] to ensure compatibility with existing implementations of MRC models. The statistics of the RRC dataset (ReviewRC) are shown in Table Table XIII. 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 XIII

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 reader[1] of DrQA [172]. 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, to answer RQ1, RQ2, and RQ3, 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 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

---

[1]https://github.com/facebookresearch/DrQA

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 Section 4.2.1 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 [73] 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

---

[1]We 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 [172]	38.26	50.99	49.52	63.73
DrQA+MRC [172]	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 XIV

RRC in EM (Exact Match) and F1.

The results of RRC are shown in Tables Table XIV. 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 [6]. 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 Dialogue System

Given the recent popularity of research in a dialogue 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)

Inspired by conversational reading comprehension (CRC), this work studies a novel task of leveraging reviews as a source to build an agent that can answer multi-turn questions from potential consumers of online businesses. We first build a review CRC dataset and then propose a novel task-aware pre-tuning step running between language model (e.g., BERT) pre-training and domain-specific fine-tuning. The proposed pre-tuning requires no data annotation, but can greatly enhance the performance on our end task. Experimental results show that the proposed approach is highly effective and has competitive performance as the supervised approach.

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 [144–147]. Although community question-answering (CQA) helps [148], 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 [148, 149], but it is not ideal for the user to read the whole reviews to fish for answers.

TABLE XV

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) [74, 153, 173], 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 Table Table XV, 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 $d = (d_1, \ldots, d_n)$, a history of past $k-1$ questions and answers as the context $C = (q_1, a_1, q_2, a_2, \ldots, q_{k-1}, a_{k-1})$ and the current question q_k, find a sequence of tokens (a textual span) $a = (d_s, \ldots, d_e)$ 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 $(\text{RC})_2$ 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, $(\text{RC})_2$ 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 [6] as our base model since it can be either a feature encoder or a standalone model that achieves good performance on CRC [74]. 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 $(\text{RC})_2$ 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. 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 dialogues 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 material[1]. 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 Table Table XVI. We use 20% of the training reviews as the validation set for each domain.

TABLE XVI

Statistics of $(RC)_2$ Datasets.

Training	**Laptop**	**Restaurant**
# of reviews	445	350
# of dialogues	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%

[1]The annotated data is in the format of CoQA [74] to help future research. But we do not focus on generative annotation as in CoQA because businesses are sensitive to errors of generative models

For the proposed pre-tuning, we collect QA pairs and reviews for these two domains. For *laptop*, we collect the reviews from [39] and QA pairs from [56] both under the laptop category of Amazon.com. We exclude products in the test data of $(\text{RC})_2$. 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 dataset[1].

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

BERT[2] is the pre-trained BERT weights directly fine-tuned on $(\text{RC})_2$ for ablation study on the effectiveness of pre-tuning.

---

[1]https://github.com/stanfordnlp/coqa-baselines

[2]We choose $\text{BERT}_{\text{BASE}}$ as we cannot fit $\text{BERT}_{\text{LARGE}}$ into the memory.

BERT+review first tunes BERT on domain reviews using the same objectives as BERT pre-training and then fine-tunes on $(\text{RC})_2$. 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 $(\text{RC})_2$. 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 ($h_{\text{max}} = 9$ for Algorithm 3) 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 evaluation script as CoQA, which reports turn-level Exact Match (EM) and F1 scores for all turns in all dialogues.

TABLE XVII

RCRC on EM (Exact Match) and F1.

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

Result Analysis

As shown in Table Table XVII, BERT+Pre-tuning has significant performance gains over BERT fine-tuned directly on $(\text{RC})_2$ 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 [174] 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 dialogue, 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 dialogue is grounded onto this user memory graph for the reasoning of dialogue 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.

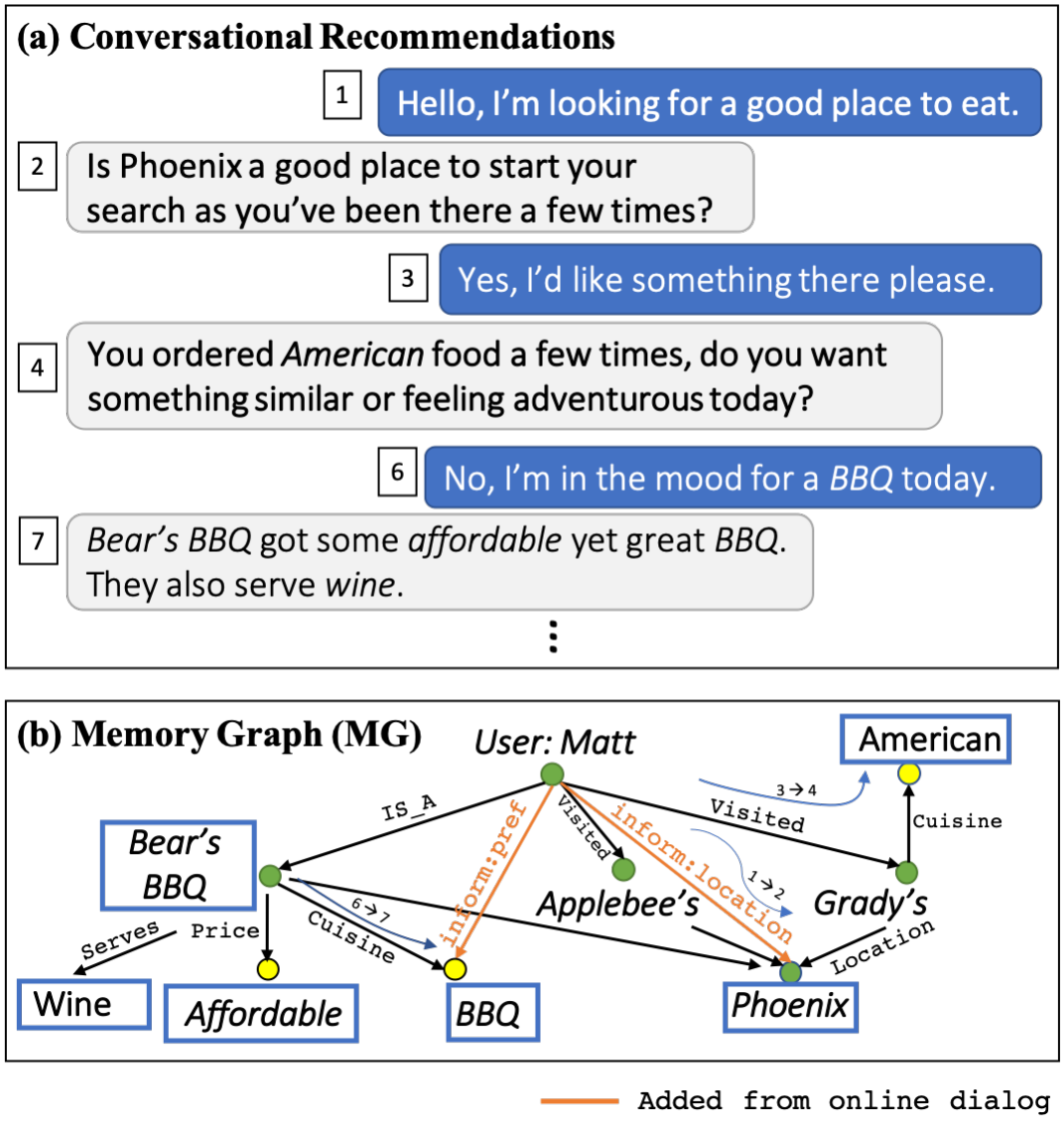

Figure 8. Conceptual illustration of Memory-grounded conversational recommendation

Conversational recommendation systems [76] are recently introduced to mitigate some of these challenges by tracking users' up-to-date preferences through dialogues. Most of the previous works focus on extending the conventional task-oriented dialogue 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 dialogue.

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 dialogues. 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 dialogues, 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 dialogue, 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 dialogue 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 dialogue 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, [76] 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.

[77,175] propose a reinforcement learning (RL) setting for a conversational recommendation system, where the dialogue policy is learned with multiple policies and recommendation signals.

[176] 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 [177–181]. Most of the state-of-the-art approaches [182–184] 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 dialogue 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 [79, 185–187] is a type of neural networks proposed to operate on graph structures. Several extensions to the original graph neural network have been proposed [188, 189], most notably R-GCNs [78], which can be applied on large-scale and highly multi-relational data. Many applications of GNNs include [190], which introduces graph-based reasoning for an offline recommendation system. A few works have recently been proposed to allow graph reasoning in dialogue systems. [191, 192] propose new corpus to learn knowledge graph paths that connect dialogue turns. [193] introduces a knowledge-grounded dialogue generation task given a knowledge graph that is dynamically updated. However, these workes often focus on response generation and do not address the conversational recommendation task.

Preliminary on Semantic Space

As discussed in the introduction, one key step to enable a dialogue being grounded and maintained on a user memory graph is to first define the semantic space of dialogue acts, items, their slots and values (we borrow these terms from task-oriented dialogue 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

dialogue acts for recommendation and then introduce slots and values specifically defined for the recommendation in a restaurant domain.

Dialogue Acts

The goal of designing dialogue acts $\mathcal{A}$ is to formalize the intentions from both the user and agent sides. Table **??** demonstrates the dialogue acts for both the user and the agent. From the agent's perspective, note that although existing conversational recommendation [76, 175, 176] 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 dialogue 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 dialogue 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 Table Table XIX. 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).

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 XIX

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

Dataset

Based on the definition in Section 6.4.2, 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 dialogue system may involve building artificial scenarios (a pre-defined setting for collecting a dialogue) [194, 195] 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 of recommender system. As a result, we first introduce a simple way to create large-scale scenarios for dialogue transcription, as in Sec. 6.4.2. Then we set up a Wizard-of-Oz environment [177–180] to collect dialogues from crowd workers and further annotate transcribed dialogues based on scenarios, as in Sec. 6.4.2. Our *MGConvRex* can be used for research in almost all crucial components of a dialogue system such as natural language understanding, sentiment analysis, dialogue state tracking, dialogue policy generation, natural language generation, etc.

Scenario Generation

A scenario is a pre-defined user-agent setting to collect a dialogue 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* $\mathcal{C}$ [2] from all available items[3].
- 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 dialogue, and optionally the user's visited items H.

---

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

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

[3] To 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.

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 dialogue, we further ask the workers to rate the whole dialogue and each other's work, where dialogues with ratings lower than 4 are filtered out. Lastly, we annotate dialogue acts, items, slots, values and users' utterance-level and entity-level sentiment for each turn of dialogues. The guidelines, screenshots of the Wizard-of-Oz UI can be found in the Appendix.

Summary of MGConvRex: After annotation, we split the dialogues 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 Table **??**.

Results

Experimental Framework

While there exist many frameworks for task-oriented dialogue systems [194–196] 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 [1] for training, offline and online evaluation of supervised (imitation) learning and reinforcement learning agents. One key

[1] We will release the code along with baselines for future research.

component of our framework is the rule-based user simulator, which can be served for both evaluation and training of reinforcement learning agent[1].

Evaluation Metrics

We propose the following metrics to evaluate UMGR over the MGConvRex dataset both offline (against the collected dialogues) 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 dialogue-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 dialogues. Note that the types of predicted entities (items, slots or values) depend on the predicted dialogue acts $\hat{y}^{\mathcal{A}}$, so correctly predicted entities must have correctly predicted dialogue acts first.

Item Matching Rate (IMR) (Dialog-level): this measures all predicted items in a dialogue 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

---

[1]The the user simulator is detailed in Appendix.

metrics that all past turns (from the human-annotated dialogues) 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 dialogues used for the offline evaluation. The maximum number of turns is simulated as 11.

Compared Methods

Our framework implements the following methods:

RandomAgent: As a baseline, we implement an agent that randomly picks a dialogue 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 [197]. 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 dialogue 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 dialogue 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 dialogue acts encoder, demonstrating the importance of the dialogue acts in policy generation.

- Prev. User Act Only: this study only uses the most recent dialogue act from the user. We use this to show how many past dialogue 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 dialogue 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 Table **??**). We use these two baselines to demonstrate that prior knowledge of user memory history aids in predicting dialogue policy.

Results

The results are shown in Table Table XXI. 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 dialogue policy prediction relying just on the user memory graph (-*No Dialogue Acts*), adding previous dialogue 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* $\mathcal{G}$ 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 dialogue policy, leading to poor performance in the recommendation.

w/ *vs* w/o Hisory. 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.

Conclusion: We build a conversational recommendation system that can collect and maintain a user's up-to-date needs and preferences for the 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 dialogue acts, items, slots, and values prediction.

	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 X

	Laptop		**Rest.**	
	Acc.	MF1	Acc.	MF1
RAM [142]				
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 [143]				
on Full Test Set	74.5	-	81.2	-
on Contrastive Test Set	42.86	33.53	42.98	33.66
MGAN [120]				
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 [133]				
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 [134]				
on Full Test Set	76.9	73.65	84.21	76.2
on Full Test Set **w/o** aspect	76.0	73.05	80.03	72.95
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 [123]				
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

	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 XXI. Results of UMGR

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

CITED LITERATURE

1. Goodfellow, I., Bengio, Y., and Courville, A.: Deep learning. Book in preparation for MIT Press, 2016.

2. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* , pages 248–255. Ieee, 2009.

3. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* , 115(3):211–252, 2015.

4. Krizhevsky, A., Sutskever, I., and Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* , pages 1097–1105, 2012.

5. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L.: Deep contextualized word representations. In *Proceedings of NAACL-HLT* , pages 2227–2237, 2018.

6. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* , pages 4171–4186, 2019.

7. Thrun, S.: Lifelong learning algorithms. In *Learning to learn* , pages 181–209. Springer, 1998.

8. Silver, D. L., Yang, Q., and Li, L.: Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning* , pages 49–55. Citeseer, 2013.

9. Chen, Z. and Liu, B.: *Lifelong machine learning* . Morgan & Claypool Publishers, 2018.

10. Bendale, A. and Boult, T.: Towards open world recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* , pages 1893–1902, 2015.

11. Fei, G., Wang, S., and Liu, B.: Learning cumulatively to become more knowledgeable. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* , pages 1565–1574. ACM, 2016.

12. Bendale, A. and Boult, T. E.: Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* , pages 1563–1572, 2016.

13. Shu, L., Xu, H., and Liu, B.: Doc: Deep open classification of text documents. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* , pages 2911–2916, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

14. Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H.: icarl: Incremental classifier and representation learning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* , pages 5533–5542. IEEE, 2017.

15. Lee, J., Yun, J., Hwang, S., and Yang, E.: Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547* , 2017.

16. Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R.: Progressive neural networks. *arXiv preprint arXiv:1606.04671* , 2016.

17. Thrun, S. and Pratt, L.: *Learning to learn* . Springer, 2012.

18. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N.: Learning to learn by gradient descent by gradient descent. In *NIPS* , pages 3981–3989, 2016.

19. Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D.: Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734* , 2017.

20. Finn, C., Abbeel, P., and Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning* , pages 1126–1135, 2017.

21. Finn, C., Xu, K., and Levine, S.: Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817* , 2018.

22. Fan, Y., Tian, F., Qin, T., Li, X.-Y., and Liu, T.-Y.: Learning to teach. *arXiv preprint arXiv:1805.03643* , 2018.

23. Lampert, C. H., Nickisch, H., and Harmeling, S.: Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* , pages 951–958. IEEE, 2009.

24. Palatucci, M., Pomerleau, D., Hinton, G. E., and Mitchell, T. M.: Zero-shot learning with semantic output codes. In *NIPS* , pages 1410–1418, 2009.

25. Socher, R., Ganjoo, M., Manning, C. D., and Ng, A.: Zero-shot learning through cross-modal transfer. In *NIPS* , pages 935–943, 2013.

26. Xing, E. P., Jordan, M. I., Russell, S. J., and Ng, A. Y.: Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems* , pages 521–528, 2003.

27. Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R.: Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems* , pages 737–744, 1994.

28. Koch, G., Zemel, R., and Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop* , volume 2, 2015.

29. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. In *Advances in Neural Information Processing Systems* , pages 3630–3638, 2016.

30. Shen, D., Ruvini, J. D., Somaiya, M., and Sundaresan, N.: Item categorization in the e-commerce domain. In *Proceedings of the 20th ACM international conference on Information and knowledge management* , pages 1921–1924. ACM, 2011.

31. Shen, D., Ruvini, J.-D., and Sarwar, B.: Large-scale item categorization for e-commerce. In *Proceedings of the 21st ACM international conference on Information and knowledge management* , pages 595–604. ACM, 2012.

32. Chen, J. and Warren, D.: Cost-sensitive learning for large-scale hierarchical classification. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management* , pages 1351–1360. ACM, 2013.

33. Gupta, V., Karnick, H., Bansal, A., and Jhala, P.: Product classification in e-commerce using distributional semantics. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* , pages 536–546, 2016.

34. Cevahir, A. and Murakami, K.: Large-scale multi-class and hierarchical product categorization for an e-commerce giant. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* , pages 525–535, 2016.

35. Kozareva, Z.: Everyone likes shopping! multi-class product categorization for e-commerce. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* , pages 1329–1333, 2015.

36. Hochreiter, S. and Schmidhuber, J.: Long short-term memory. *Neural computation* , 9(8):1735–1780, 1997.

37. Schuster, M. and Paliwal, K. K.: Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* , 45(11):2673–2681, 1997.

38. Lake, B., Salakhutdinov, R., Gross, J., and Tenenbaum, J.: One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society* , volume 33, 2011.

39. He, R. and McAuley, J.: Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web* , pages 507–517. International World Wide Web Conferences Steering Committee, 2016.

40. Pennington, J., Socher, R., and Manning, C.: Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* , pages 1532–1543, 2014.

41. Kingma, D. P. and Ba, J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* , 2014.

42. Mnih, A. and Hinton, G.: Three new graphical models for statistical language modelling. In *ICML* , pages 641–648, 2007.

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

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

45. Sienčnik, S. K.: Adapting word2vec to named entity recognition. In *NCCL* , pages 239–243, 2015.

46. Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C.: Learning word vectors for sentiment analysis. In *ACL* , pages 142–150, 2011.

47. Durrett, G. and Klein, D.: Neural crf parsing. *arXiv* , 2015.

48. Bollegala, D., Maehara, T., and Kawarabayashi, K.-i.: Unsupervised cross-domain word representation learning. In *ACL* , pages 730–740, 2015.

49. Yang, W., Lu, W., and Zheng, V.: A simple regularization-based algorithm for learning cross-domain word embeddings. In *EMNLP* , pages 2898–2904, 2017.

50. Bollegala, D., Hayashi, K., and Kawarabayashi, K.-i.: Think globally, embed locally—locally linear meta-embedding of words. *arXiv:1709.06671* , 2017.

51. Pan, S. J. and Yang, Q.: A survey on transfer learning. *IEEE TKDE* , pages 1345–1359, 2010.

52. Chen, Z. and Liu, B.: *Lifelong Machine Learning* . Morgan & Claypool Publishers, 2016.

53. Mikolov, T., Yih, W.-t., and Zweig, G.: Linguistic regularities in continuous space word representations. In *hlt-Naacl* , pages 746–751, 2013.

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

55. Xu, H., Liu, B., Shu, L., and Yu, P. S.: Double embeddings and cnn-based sequence labeling for aspect extraction. In *ACL* , pages 148–154, 2018.

56. Xu, H., Xie, S., Shu, L., and Yu, P. S.: Dual attention network for product compatibility and function satisfiability analysis. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)* , 2018.

57. Thrun, S.: Is learning the n-th thing any easier than learning the first? In *NIPS* , pages 640–646, 1996.

58. Silver, D. L., Yang, Q., and Li, L.: Lifelong Machine Learning Systems: Beyond Learning Algorithms. In *AAAI Spring Symposium: LML* , page 05, 2013.

59. Chen, Z. and Liu, B.: Topic modeling using topics from many domains, lifelong learning and big data. In *ICML* , pages 703–711, 2014.

60. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al.: Never-ending learning. *Communications of the ACM* , 61(5):103–115, 2018.

61. Shu, L., Liu, B., Xu, H., and Kim, A.: Lifelong-rl: Lifelong relaxation labeling for separating entities and aspects in opinion targets. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* , pages 225–235, 2016.

62. Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V.: Domain-adversarial training of neural networks. *JMLR* , pages 2096–2030, 2016.

63. Nayak, N., Angeli, G., and Manning, C. D.: Evaluating word embeddings using a representative suite of practical tasks. *ACL 2016* , pages 19–23, 2016.

64. He, R. and McAuley, J.: Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *World Wide Web* , 2016.

65. Xu, H., Liu, B., Shu, L., and Yu, P. S.: Lifelong domain word embedding via meta-learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* . AAAI Press, 2018.

66. Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I.: Improving language understanding by generative pre-training. *URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language understanding paper.pdf* , 2018.

67. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I.: Language models are unsupervised multitask learners. *OpenAI Blog* , 1(8):9, 2019.

68. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V.: Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems* , pages 5754–5764, 2019.

69. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* , 2019.

70. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R.: Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations* , 2019.

71. Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D.: Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations* , 2019.

72. Xu, H., Liu, B., Shu, L., and Yu, P. S.: Double embeddings and cnn-based sequence labeling for aspect extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* . Association for Computational Linguistics, 2018.

73. Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P.: Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* , 2016.

74. Reddy, S., Chen, D., and Manning, C. D.: Coqa: A conversational question answering challenge. *arXiv preprint arXiv:1808.07042* , 2018.

75. Rajpurkar, P., Jia, R., and Liang, P.: Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822* , 2018.

76. Li, R., Kahou, S. E., Schulz, H., Michalski, V., Charlin, L., and Pal, C.: Towards deep conversational recommendations. In *Advances in Neural Information Processing Systems* , pages 9725–9735, 2018.

77. Kang, D., Balakrishnan, A., Shah, P., Crook, P. A., Boureau, Y.-L., and Weston, J.: Recommendation as a communication game: Self-supervised bot-play for goal-oriented dialogue. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* , pages 1951–1961, 2019.

78. Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M.: Modeling relational data with graph convolutional networks. In *European Semantic Web Conference* , pages 593–607. Springer, 2018.

79. Kipf, T. N. and Welling, M.: Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* , 2016.

80. Liu, B.: *Sentiment Analysis and Opinion Mining* . Morgan & Claypool Publishers, 2012.

81. Hu, M. and Liu, B.: Mining and summarizing customer reviews. In *KDD '04* , pages 168–177, 2004.

82. Jakob, N. and Gurevych, I.: Extracting opinion targets in a single- and cross-domain setting with conditional random fields. In *EMNLP '10* , pages 1035–1045, 2010.

83. Chernyshevich, M.: Ihs r&d belarus: Cross-domain extraction of product features using crf. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)* , pages 309–313, 2014.

84. Shu, L., Xu, H., and Liu, B.: Lifelong learning crf for supervised aspect extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* , pages 148–154, 2017.

85. Zhuang, L., Jing, F., and Zhu, X.-Y.: Movie review mining and summarization. In *CIKM '06* , pages 43–50, 2006.

86. Mei, Q., Ling, X., Wondra, M., Su, H., and Zhai, C.: Topic sentiment mixture: Modeling facets and opinions in weblogs. In *WWW '07* , pages 171–180, 2007.

87. Qiu, G., Liu, B., Bu, J., and Chen, C.: Opinion word expansion and target extraction through double propagation. *Computational Linguistics* , 37(1):9–27, 2011.

88. Yin, Y., Wei, F., Dong, L., Xu, K., Zhang, M., and Zhou, M.: Unsupervised word and dependency path embeddings for aspect term extraction. *arXiv preprint arXiv:1605.07843* , 2016.

89. He, R., Lee, W. S., Ng, H. T., and Dahlmeier, D.: An unsupervised neural attention model for aspect extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* , volume 1, pages 388–397, 2017.

90. Li, X. and Lam, W.: Deep multi-task learning for aspect term extraction with memory interaction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* , pages 2886–2892, 2017.

91. Poria, S., Cambria, E., and Gelbukh, A.: Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems* , 108:42–49, 2016.

92. Wang, W., Pan, S. J., Dahlmeier, D., and Xiao, X.: Recursive neural conditional random fields for aspect-based sentiment analysis. *arXiv preprint arXiv:1603.06679* , 2016.

93. Wang, W., Pan, S. J., Dahlmeier, D., and Xiao, X.: Coupled multi-layer attentions for co-extraction of aspect and opinion terms. In *Thirty-First AAAI Conference on Artificial Intelligence* , 2017.

94. Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K.: Occam's razor. *Information processing letters* , 24(6):377–380, 1987.

95. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* , 3361(10):1995, 1995.

96. Liu, P., Joty, S., and Meng, H.: Fine-grained opinion mining with recurrent neural networks and word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* , pages 1433–1443, 2015.

97. Kim, Y.: Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* , 2014.

98. Zhang, X., Zhao, J., and LeCun, Y.: Character-level convolutional networks for text classification. In *Advances in neural information processing systems* , pages 649–657, 2015.

99. Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N.: Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122* , 2017.

100. Pang, B. and Lee, L.: Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.* , 2:1–135, 2008.

101. Cambria, E. and Hussain, A.: *Sentic Computing Techniques, Tools, and Applications 2nd Edition* . Springer, 2012.

102. Popescu, A.-M. and Etzioni, O.: Extracting product features and opinions from reviews. In *HLT-EMNLP '05* , pages 339–346, 2005.

103. Wang, B. and Wang, H.: Bootstrapping both product features and opinion words from chinese customer reviews with cross-inducing. In *IJCNLP '08* , pages 289–295, 2008.

104. Titov, I. and McDonald, R.: A joint model of text and aspect ratings for sentiment summarization. In *ACL '08: HLT* , pages 308–316, 2008.

105. Lin, C. and He, Y.: Joint sentiment/topic model for sentiment analysis. In *CIKM '09* , pages 375–384, 2009.

106. Moghaddam, S. and Ester, M.: ILDA: interdependent lda model for learning latent aspects and their ratings from online product reviews. In *SIGIR '11* , pages 665–674, 2011.

107. Liu, K., Xu, L., Liu, Y., and Zhao, J.: Opinion target extraction using partially-supervised word alignment model. In *IJCAI '13* , pages 2134–2140, 2013.

108. Zhou, X., Wan, X., and Xiao, J.: Collective opinion target extraction in Chinese microblogs. In *EMNLP '13* , pages 1840–1850, 2013.

109. Mitchell, M., Aguilar, J., Wilson, T., and Van Durme, B.: Open domain targeted sentiment. In *ACL '13* , pages 1643–1654, 2013.

110. Lafferty, J., McCallum, A., and Pereira, F. C.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01* , pages 282–289, 2001.

111. Williams, R. J. and Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. *Neural computation* , 1(2):270–280, 1989.

112. Bollegala, D., Maehara, T., and Kawarabayashi, K.-i.: Unsupervised cross-domain word representation learning. *arXiv preprint arXiv:1505.07184* , 2015.

113. Strubell, E., Verga, P., Belanger, D., and McCallum, A.: Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* , pages 2670–2680, 2017.

114. Chen, T., Xu, R., He, Y., and Wang, X.: Improving sentiment analysis via sentence type classification using bilstm-crf and cnn. *Expert Systems with Applications* , 72:221–230, 2017.

115. Pontiki, M., Galanis, D., Pavlopoulos, J., Papageorgiou, H., Androutsopoulos, I., and Manandhar, S.: Semeval-2014 task 4: Aspect based sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)* , pages 27–

35, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

116. Pontiki, M., Galanis, D., Papageorgiou, H., Androutsopoulos, I., Manandhar, S., Mohammad, A.-S., Al-Ayyoub, M., Zhao, Y., Qin, B., De Clercq, O., et al.: Semeval-2016 task 5: Aspect based sentiment analysis. In *Proceedings of the 10th international workshop on semantic evaluation (SemEval-2016)* , pages 19–30, 2016.

117. Toh, Z. and Su, J.: Nlangp at semeval-2016 task 5: Improving aspect based sentiment analysis using neural network features. In *Proceedings of the 10th international workshop on semantic evaluation (SemEval-2016)* , pages 282–288, 2016.

118. Reimers, N. and Gurevych, I.: Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)* , pages 338–348, Copenhagen, Denmark, 09 2017.

119. Levy, O. and Goldberg, Y.: Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* , volume 2, pages 302–308, 2014.

120. Li, Z., Wei, Y., Zhang, Y., Zhang, X., Li, X., and Yang, Q.: Exploiting coarse-to-fine task transfer for aspect-level sentiment classification. *arXiv preprint arXiv:1811.10999* , 2018.

121. Tang, D., Qin, B., and Liu, T.: Aspect level sentiment classification with deep memory network. *arXiv preprint arXiv:1605.08900* , 2016.

122. Shrivastava, A., Gupta, A., and Girshick, R.: Training region-based object detectors with online hard example mining. In *CVPR* , pages 761–769, 2016.

123. Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P.: Focal loss for dense object detection. In *ICCV* , pages 2980–2988, 2017.

124. Hu, M. and Liu, B.: Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* , pages 168–177. ACM, 2004.

125. Liu, B.: *Sentiment analysis: Mining opinions, sentiments, and emotions* . Cambridge University Press, 2015.

126. Jiang, Q., Chen, L., Xu, R., Ao, X., and Yang, M.: A challenge dataset and effective models for aspect-based sentiment analysis. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* , pages 6279–6284, Hong Kong, China, November 2019. Association for Computational Linguistics.

127. Pang, B., Lee, L., and Vaithyanathan, S.: Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10* , pages 79–86. Association for Computational Linguistics, 2002.

128. He, Y. and Zhou, D.: Self-training from labeled features for sentiment analysis. *Information Processing & Management* , 47(4):606–616, 2011.

129. He, Y., Lin, C., and Alani, H.: Automatically extracting polarity-bearing topics for cross-domain sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1* , pages 123–131. Association for Computational Linguistics, 2011.

130. Li, X., Bing, L., Li, P., and Lam, W.: A unified model for opinion target extraction and target sentiment prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence* , volume 33, pages 6714–6721, 2019.

131. Dong, L., Wei, F., Tan, C., Tang, D., Zhou, M., and Xu, K.: Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)* , volume 2, pages 49–54, 2014.

132. Nguyen, T. H. and Shirai, K.: PhraseRNN: Phrase recursive neural network for aspect-based sentiment analysis. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* , pages 2509–2514, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

133. Li, X., Bing, L., Lam, W., and Shi, B.: Transformation networks for target-oriented sentiment classification. *arXiv preprint arXiv:1805.01086* , 2018.

134. Xu, H., Liu, B., Shu, L., and Yu, P. S.: Bert post-training for review reading comprehension and aspect-based sentiment analysis. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics* , jun 2019.

135. He, R., Lee, W. S., Ng, H. T., and Dahlmeier, D.: Exploiting document knowledge for aspect-level sentiment classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* . Association for Computational Linguistics.

136. Wang, S., Lv, G., Mazumder, S., Fei, G., and Liu, B.: Lifelong learning memory networks for aspect sentiment classification. In *2018 IEEE International Conference on Big Data (Big Data)* , pages 861–870. IEEE, 2018.

137. Wang, S., Mazumder, S., Liu, B., Zhou, M., and Chang, Y.: Target-sensitive memory networks for aspect sentiment classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* , pages 957–967, 2018.

138. Freund, Y. and Schapire, R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* , 55(1):119–139, 1997.

139. Schwenk, H. and Bengio, Y.: Boosting neural networks. *Neural computation* , 12(8):1869–1887, 2000.

140. Mosca, A. and Magoulas, G. D.: Deep incremental boosting. *arXiv preprint arXiv:1708.03704* , 2017.

141. Gao, T. and Jojic, V.: Sample importance in training deep neural networks. 2016.

142. Chen, P., Sun, Z., Bing, L., and Yang, W.: Recurrent attention network on memory for aspect sentiment analysis. In *Proceedings of the 2017 conference on empirical methods in natural language processing* , pages 452–461, 2017.

143. Huang, B., Ou, Y., and Carley, K. M.: Aspect level sentiment classification with attention-over-attention neural networks. In *International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation* , pages 197–206. Springer, 2018.

144. Xu, K., Reddy, S., Feng, Y., Huang, S., and Zhao, D.: Question answering on freebase via relation extraction and textual evidence. *arXiv preprint arXiv:1603.00957* , 2016.

145. Fader, A., Zettlemoyer, L., and Etzioni, O.: Open question answering over curated and extracted knowledge bases. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* , pages 1156–1165. ACM, 2014.

146. Kwok, C., Etzioni, O., and Weld, D. S.: Scaling question answering to the web. *ACM Transactions on Information Systems (TOIS)* , 19(3):242–262, 2001.

147. Yin, J., Jiang, X., Lu, Z., Shang, L., Li, H., and Li, X.: Neural generative question answering. *arXiv preprint arXiv:1512.01337* , 2015.

148. McAuley, J. and Yang, A.: Addressing complex and subjective product-related queries with customer reviews. In *Proceedings of the 25th International Conference on World Wide Web* , pages 625–635. International World Wide Web Conferences Steering Committee, 2016.

149. Yu, Q. and Lam, W.: Aware answer prediction for product-related questions incorporating aspects. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* , pages 691–699. ACM, 2018.

150. Hewlett, D., Lacoste, A., Jones, L., Polosukhin, I., Fandrianto, A., Han, J., Kelcey, M., and Berthelot, D.: Wikireading: A novel large-scale language understanding task over wikipedia. *arXiv preprint arXiv:1608.03542* , 2016.

151. Welbl, J., Stenetorp, P., and Riedel, S.: Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association of Computational Linguistics* , 6:287–302, 2018.

152. Shao, C. C., Liu, T., Lai, Y., Tseng, Y., and Tsai, S.: Drcd: a chinese machine reading comprehension dataset. *arXiv preprint arXiv:1806.00920* , 2018.

153. Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W.-t., Choi, Y., Liang, P., and Zettlemoyer, L.: Quac: Question answering in context. *arXiv preprint arXiv:1808.07036* , 2018.

154. Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D.: Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600* , 2018.

155. Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P.: Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems* , pages 1693–1701, 2015.

156. Trischler, A., Wang, T., Yuan, X., Harris, J., Sordoni, A., Bachman, P., and Suleman, K.: Newsqa: A machine comprehension dataset. *arXiv preprint arXiv:1611.09830* , 2016.

157. Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E.: Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683* , 2017.

158. Richardson, M., Burges, C. J., and Renshaw, E.: Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* , pages 193–203, 2013.

159. Hill, F., Bordes, A., Chopra, S., and Weston, J.: The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301* , 2015.

160. Kočiskỳ, T., Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E.: The narrativeqa reading comprehension challenge. *Transactions of the Association of Computational Linguistics* , 6:317–328, 2018.

161. Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L.: Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268* , 2016.

162. Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L.: Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551* , 2017.

163. Dunn, M., Sagun, L., Higgins, M., Guney, V. U., Cirik, V., and Cho, K.: Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179* , 2017.

164. Dong, L., Wei, F., Zhou, M., and Xu, K.: Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* , volume 1, pages 260–269, 2015.

165. Yao, X. and Van Durme, B.: Information extraction over structured data: Question answering with freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* , volume 1, pages 956–966, 2014.

166. Lopez, V., Nikolov, A., Sabou, M., Uren, V., Motta, E., and d'Aquin, M.: Scaling up question-answering to linked data. In *International Conference on Knowledge Engineering and Knowledge Management* , pages 193–210. Springer, 2010.

167. Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., Gerber, D., and Cimiano, P.: Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web* , pages 639–648. ACM, 2012.

168. Liu, B.: Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies* , 5(1):1–167, 2012.

169. He, R., Lee, W. S., Ng, H. T., and Dahlmeier, D.: Exploiting document knowledge for aspect-level sentiment classification. *arXiv preprint arXiv:1806.04346* , 2018.

170. Howard, J. and Ruder, S.: Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* , volume 1, pages 328–339, 2018.

171. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I.: Language models are unsupervised multitask learners. *URL https://openai.com/blog/better-language-models/* , 2018.

172. Chen, D., Fisch, A., Weston, J., and Bordes, A.: Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051* , 2017.

173. Xu, H., Liu, B., Shu, L., and Yu, P.: BERT post-training for review reading comprehension and aspect-based sentiment analysis. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* , pages 2324–2335, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

174. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* , page 201611835, 2017.

175. Sun, Y. and Zhang, Y.: Conversational recommender system. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* , pages 235–244. ACM, 2018.

176. Zhang, Y., Chen, X., Ai, Q., Yang, L., and Croft, W. B.: Towards conversational search and recommendation: System ask, user respond. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* , pages 177–186. ACM, 2018.

177. Henderson, M., Thomson, B., and Williams, J. D.: The second dialog state tracking challenge. In *Special Interest Group on Discourse and Dialogue (SIGDIAL)* , 2014.

178. Wen, T.-H., Vandyke, D., Mrksic, N., Gasic, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., and Young, S.: A network-based end-to-end trainable task-oriented dialogue system. In *European Chapter of the Association for Computational Linguistics (EACL)* , 2016.

179. Budzianowski, P., Wen, T.-H., Tseng, B.-H., Casanueva, I., Ultes, S., Ramadan, O., and Gašić, M.: MultiWOZ - a large-scale multi-domain wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* , 2018.

180. Eric, M., Goel, R., Paul, S., Kumar, A., Sethi, A., Ku, P., Goyal, A. K., Agarwal, S., Gao, S., and Hakkani-Tur, D.: Multiwoz 2.1: Multi-domain dialogue state corrections and state tracking baselines. *arXiv preprint arXiv:1907.01669* , 2019.

181. Rastogi, A., Zang, X., Sunkara, S., Gupta, R., and Khaitan, P.: Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Association for the Advancement of Artificial Intelligence (AAAI)* , 2019.

182. Wu, C.-S., Madotto, A., Hosseini-Asl, E., Xiong, C., Socher, R., and Fung, P.: Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics* , 2019.

183. Gao, S., Abhishek Seth and, S. A., Chun, T., and Hakkani-Ture, D.: Dialog state tracking: A neural reading comprehension approach. In *Special Interest Group on Discourse and Dialogue (SIGDIAL)* , 2019.

184. Chao, G.-L. and Lane, I.: Bert-dst: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)* , 2019.

185. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G.: The graph neural network model. *IEEE Transactions on Neural Networks* , 20(1):61–80, 2008.

186. Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P.: Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems* , pages 2224–2232, 2015.

187. Defferrard, M., Bresson, X., and Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems* , pages 3844–3852, 2016.

188. Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R.: Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* , 2015.

189. Pham, T., Tran, T., Phung, D., and Venkatesh, S.: Column networks for collective classification. In *Thirty-First AAAI Conference on Artificial Intelligence* , 2017.

190. Xian, Y., Fu, Z., Muthukrishnan, S., de Melo, G., and Zhang, Y.: Reinforcement knowledge graph reasoning for explainable recommendation. In *SIGIR* , 2019.

191. Moon, S., Shah, P., Kumar, A., and Subba, R.: Opendialkg: Explainable conversational reasoning with attention-based walks over knowledge graphs. *ACL* , 2019.

192. Moon, S., Shah, P., Kumar, A., and Subba, R.: Memory grounded conversational reasoning. *EMNLP* , 2019.

193. Tuan, Y.-L., Chen, Y.-N., and Lee, H.-y.: DyKgChat: Benchmarking dialogue generation grounding on dynamic knowledge graphs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* , pages 1855–1865, Hong Kong, China, November 2019. Association for Computational Linguistics.

194. Li, X., Lipton, Z. C., Dhingra, B., Li, L., Gao, J., and Chen, Y.-N.: A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688* , 2016.

195. Li, X., Panda, S., Liu, J., and Gao, J.: Microsoft dialogue challenge: Building end-to-end task-completion dialogue systems. *arXiv preprint arXiv:1807.11125* , 2018.

196. Lee, S., Zhu, Q., Takanobu, R., Li, X., Zhang, Y., Zhang, Z., Li, J., Peng, B., Li, X., Huang, M., et al.: Convlab: Multi-domain end-to-end dialog system platform. *arXiv preprint arXiv:1904.08637* , 2019.

197. Nickel, M., Rosasco, L., and Poggio, T.: Holographic embeddings of knowledge graphs. *AAAI* , 2016.

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 EXPERIENCE: Research Assistant, Big Data and Social Computing Lab, Department of Computer Science, University of Illinois at Chicago, 2015 - 2020.

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

Teaching Assistant, Department of Computer Science, University of Illinois at Chicago:

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