Lifelong Machine Learning

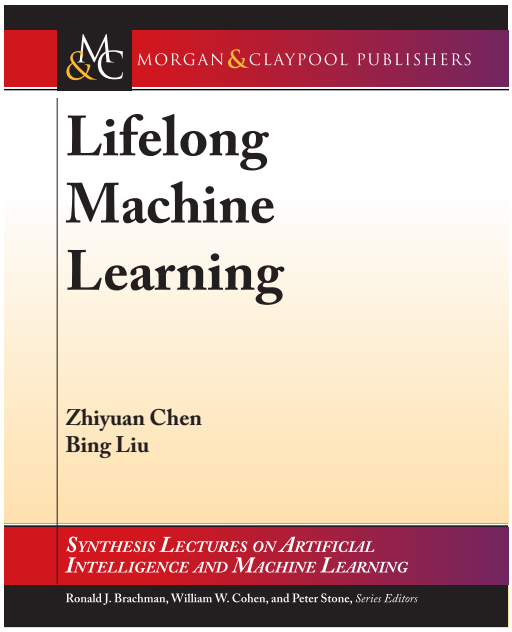
November, 2016

Zhiyuan Chen and Bing Liu

czyuanacm@gamil.com, liub@cs.uic.edu

Draft: This is mainly an early draft of the book.
We also updated a few places after the publication, highlighted in yellow.

Zhiyuan Chen and Bing Liu. Lifelong Machine Learning.
Morgan & Claypool Publishers, Nov 2016.



Lifelong Machine Learning

Lifelong Machine Learning

Zhiyuan Chen
Google, Inc.

Bing Liu
University of Illinois at Chicago

SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING #31

ABSTRACT

Lifelong machine learning (or *lifelong learning*) is an advanced machine learning paradigm that learns continuously, accumulates the knowledge learned in previous tasks, and uses it to help future learning. In the process, the learner becomes more and more knowledgeable and effective at learning. This learning ability is one of the hallmarks of human intelligence. However, the current dominant machine learning paradigm learns *in isolation*: given a training dataset, it runs a machine learning algorithm on the dataset to produce a model. It makes no attempt to retain the learned knowledge and use it in future learning. Although this *isolated learning paradigm* has been very successful, it requires a large number of training examples, and is only suitable for well-defined and narrow tasks. In comparison, we humans can learn effectively with a few examples because we have accumulated so much knowledge in the past which enables us to learn with little data or effort. Furthermore, we are able to discover new problems in the usage process of the learned knowledge or model. This enables us to learn more and more continually in a self-motivated manner. We can also adapt our previous konwledge to solve unfamilar problems and learn in the process. Lifelong learning aims to achieve these capabilities. As statistical machine learning matures, it is time to make a major effort to break the isolated learning tradition and to study lifelong learning to bring machine learning to a new height. Applications such as intelligent assistants, chatbots, and physical robots that interact with humans and systems in real-life environments are also calling for such lifelong learning capabilities. Without the ability to accumulate the learned knowledge and use it to learn more knowledge incrementally, a system will probably never be truly intelligent. This book serves as an introductory text and survey to lifelong learning.

KEYWORDS

lifelong machine learning; lifelong learning; learning with memory; cumulative learning; multi-task learning; transfer learning; continual learning

Zhiyuan dedicates this book to his wife Vena Li and his parents.
Bing dedicates this book to his wife Yue He, his children
Shelley and Kate, and his parents.

Contents

Preface

This book started with a tutorial on *lifelong machine learning* that we gave at *the 24th International Joint Conference on Artificial Intelligence* (IJCAI) in 2015. At that time, we had worked on the topic for a while and published several papers in ICML, KDD, and ACL. When the publisher from Morgan & Claypool Publishers contacted us about the possibility of developing a book on the topic, we were excited. We strongly believe that lifelong machine learning (or simply lifelong learning) is very important for the future of machine learning and artificial intelligence (AI). Our original research interest in the topic stemmed from extensive application experiences in sentiment analysis (SA) in a startup company several years ago. A typical SA project starts with a client who is interested in consumer opinions expressed in social media about their products or services and those of their competitors. There are two main analysis tasks that a SA system needs to do: (1) discovering the entities (e.g., *iPhone*) and entity attributes/features (e.g., *battery life*) that people talked about in an opinion document such as an online review and (2) determining whether an opinion about an entity or entity attribute is positive, negative, or neutral [Liu, 2012, 2015]. For example, from the sentence "*iPhone is really cool, but its battery life sucks,*" a SA system should discover that the author is (1) positive about *iPhone* but (2) negative about iPhone's *battery life*.

After working on many projects in many domains (which are types of products or services) for clients, we realized that there is a great deal of sharing of information across domains and projects. As we see more and more, new things get fewer and fewer. It is easy to notice that sentiment words and expressions (such as *good*, *bad*, *poor*, *terrible*, and *cost an arm and a leg*) are shared across domains. There is also a great deal of sharing of entities and attributes. For example, every product has the attribute of *price*, most electronic products have *battery*, and many of them also have *screen*. It is silly not to exploit such sharing to significantly improve SA to make it much more accurate than discovering such sharing but only working on each project and its data in isolation. The classic machine learning paradigm learns exactly in isolation. Given a dataset, a learning algorithm simply runs on the data to produce a model. The algorithm has no memory and thus is unable to use the past learned knowledge. In order to exploit knowledge sharing, a SA system needs to retain and accumulate the knowledge learned in the past and use it to help future learning and problem solving, which is exactly what *lifelong machine learning* aims to do.

It is not hard to imagine that this sharing of information or knowledge across domains and tasks is generally true in every field. It is particularly obvious in natural language processing because the meanings of words and phrases are basically the same across domains

and tasks and so is the sentence syntax. No matter what subject matters we talk about, we use the same language although one subject may use only a small subset of the words and phrases in a language. If that is not the case, it is doubtful that a natural language would have ever been developed by humans. Thus, lifelong machine learning is generally applicable, not just restricted to sentiment analysis.

The goal of this book is to introduce this emerging machine learning paradigm and to present a comprehensive survey and review of important research results and the latest ideas in the area. We also want to propose a unified framework for the research area. Currently, there are several research topics in machine learning that are closely related to lifelong machine learning, most notably, multi-task learning and transfer learning, because they also employ the idea of knowledge sharing and transfer. This book brings all these topics under one roof and discusses their similarities and differences. We see lifelong machine learning as an important extension to these related problems. Through this book, we would also like to motivate and encourage researchers to work on lifelong machine learning. We believe it represents a major future research direction for both machine learning and artificial intelligence. Without the capability of retaining and accumulating the knowledge learned in the past, making inference about it, and using the knowledge to help future learning and problem solving, achieving general intelligence is quite unlikely.

Two main principles have guided the writing of this book. First, it should contain strong motivations for conducting research in lifelong machine learning in order to encourage graduate students and researchers to work on lifelong machine learning problems. Second, the writing should be accessible to practitioners and upper-level undergraduate students who have basic knowledge of machine learning and data mining. Yet there should be sufficient in-depth materials for graduate students who plan to pursue Ph.D. degrees in the machine learning and/or data mining field.

This book is thus suitable for students, researchers, and practitioners who are interested in machine learning, data mining, natural language processing, or pattern recognition. Lecturers can readily use the book in class for courses in any of these related fields. Lecture slides are available online.

Zhiyuan Chen and Bing Liu
September 2016

Acknowledgments

We would like to thank our collaborators and students in our group: Geli Fei, Zhiqiang Gao, Annice Kim, Doo Soon Kim, Huayi Li, Qian Liu, Sahisnu Mazumder, Arjun Mukherjee, Nianzu Ma, Lei Shu, Shuai Wang, Yueshen Xu, and Yuanlin Zhang, for their contributions of numerous research ideas over the years. We are especially grateful to the two expert reviewers of the book, Eric Eaton and Matthew E. Taylor. Despite their busy schedules, they read the first draft of the book very carefully and gave us so many excellent comments and suggestions, which are not only insightful and comprehensive, but also detailed and very constructive. Their suggestions have helped us improve the book tremendously.

On the publication side, we thank the editors of Synthesis Lectures on Artificial Intelligence and Machine Learning, Ronald Brachman, William W. Cohen, and Peter Stone, for initiating this project. The President and CEO of Morgan & Claypool Publishers, Michael Morgan and his staff Samantha Draper have given us all kinds of help promptly whenever requested, for which we are very grateful.

Our greatest gratitude go to our own families. Zhiyuan would like to thank his wife Vena Li and his parents. Bing Liu would like to thank his wife Yue, his children Shelley and Kate, and his parents. They have helped in so many ways.

The writing of this book was partially supported by two National Science Foundation (NSF) grants IIS-1407927 and IIS-1650900, an NCI grant R01CA192240, and a gift from Bosch. The content of the book is solely the responsibility of the authors and does not necessarily represent the official views of the NSF, NCI, or Bosch. The Department of Computer Science at the University of Illinois at Chicago provided computing resources and a supportive environment for this project. Working in Google has given Zhiyuan a broader perspective on machine learning.

Zhiyuan Chen and Bing Liu
September 2016

CHAPTER 1

Introduction

Machine learning (ML) has been instrumental for the advances of both data analysis and artificial intelligence (AI). The recent success of deep learning brings it to a new height. ML algorithms have been used in almost all areas of computer science and many areas of natural science, engineering, and social sciences. Practical applications are even more widespread. It is safe to say that without effective ML algorithms, many industries would not have flourished, e.g., Internet commerce and Web search.

The current dominant paradigm for machine learning is to run an ML algorithm on a given dataset to generate a model. The model is then applied in real-life performance tasks. This is true for both supervised learning and unsupervised learning. We call this paradigm *isolated learning* because it does not consider any other related information or the previously learned knowledge. The fundamental problem with this isolated learning paradigm is that it does not retain and accumulate knowledge learned in the past and use it in future learning. This is in contrast to our human learning. We humans never learn in isolation. We always retain the knowledge learned in the past and use it to help future learning and problem solving. That is why whenever we encounter a new situation or problem, we may notice that many aspects of it are not really new because we have seen them in the past in some other contexts. Without the ability to accumulate knowledge, an ML algorithm typically needs a large number of training examples in order to learn effectively. For supervised learning, labeling of training data is often done manually, which is very labor-intensive and time-consuming. Since the world is too complex with too many possible tasks, it is almost impossible to label a large number of examples for every possible task or application for an ML algorithm to learn. To make matters worse, everything around us also changes constantly, and the labeling thus needs to be done continuously, which is a daunting task for humans. Even for unsupervised learning, collecting a large volume of data may not be possible in many cases.

To solve the problem, researchers have proposed one-shot learning and zero-shot learning, which use only a small number of examples or even no example to learn. However, for such techniques to work, the system must have a great deal of prior domain knowledge. The question is where the knowledge comes from. In most cases, the knowledge is from human users. This is clearly not an ideal solution for an intelligent system because it means that the learner of the system still has to rely on humans and cannot learn by itself. It is more desirable to learn and cumulate the required prior knowledge from the past tasks Chen et al. [2014].

In contrast, we human beings seem to learn quite differently. We accumulate and maintain the knowledge learned from previous tasks and use it seamlessly in learning new tasks and solving new problems. Over time we learn more and more and become more and more knowledgeable, and more and more effective at learning. *Lifelong Machine Learning* (LML) (or simply *lifelong learning*) aims to mimic this human learning process and capability. This type of learning is quite natural because things around us are closely related and interconnected. Knowledge learned about some subjects can help us understand and learn some other subjects. For example, we humans do not need 1,000 positive online reviews and 1,000 negative online reviews of movies as an ML algorithm would need in order to build an accurate classifier to classify positive and negative reviews about a movie. In fact, for this task, without a single training example, we can already perform the classification task. How can that be? The reason is simple. It is because we have accumulated so much knowledge in the past about the language expressions that people use to praise and criticize things, although none of those praises or criticisms may be in the form of online reviews. Interestingly, if we do not have such past knowledge, we humans are probably unable to manually build a good classifier even with 1,000 training positive reviews and 1,000 training negative reviews without spending an enormous amount of time. For example, if you have no knowledge of Arabic or Arabic sentiment expressions and someone gives you 2,000 labeled training reviews in Arabic and asks you to build a classifier manually, most probably you will not be able to do it without using a translator.

To make the case more general, we use natural language processing (NLP) as an example. It is easy to see the importance of LML to NLP because of several reasons. First, words and phrases have almost the same meaning in all domains and all tasks. Second, sentences in every domain follows the same syntax or grammar. Third, almost all natural language processing problems are closely related to each other, which means that they are inter-connected and affect each other in some ways. The first two reasons ensure that the knowledge learned can be used across domains and tasks due to the sharing of the same expressions and meanings and the same syntax. That is why we humans do not need to re-learn the language (or learn a new language) whenever we encounter a new application domain. For example, assume we have never studied psychology, and we want to study it now. We do not need to learn the language used in the psychology text except some new concepts in the psychology domain because everything about the language itself is the same as in any other domain or area. The third reason ensures that LML can be used across different types of tasks. For example, a named entity recognition (NER) system has learned that iPhone is a product or entity, and a data mining system has discovered that every product has a price and the adjective "expensive" describes the price attribute of an entity. Then, from the sentence *"The picture quality of iPhone is great, but it is quite expensive,"* we can safely extract "picture quality" as a feature or attribute of iPhone, and detect that "it" refers to iPhone not the picture quality with the help of those pieces

of prior knowledge. Traditionally, these problems are solved separately in isolation, but they are all related and can help each other because the results from one problem can be useful to others. This situation is common for all NLP tasks. Note that we regard anything from unknown to known as a piece of knowledge. Thus, a learned model is a piece of knowledge and the results gained from applying the model are also knowledge, although they are different kinds of knowledge. For example, iPhone being an entity and picture quality being an attribute of iPhone are two pieces of knowledge.

Realizing and being able to exploit the sharing of words and expressions across domains and the inter-connectedness of tasks is still insufficient. A large quantities of knowledge is often needed in order to effectively help the new task learning because the knowledge gained from one previous task may contain only a tiny bit or even no knowledge that is applicable to the new task (unless the two tasks are extremely similar). Thus, it is important to learn from a large number of diverse domains to accumulate a large amount of diverse knowledge over time. A future task can pick and choose the appropriate knowledge to use to help its learning. As the world also changes constantly, the learning should thus be continuous and lifelong, which is what we humans do.

Although we used NLP as an example, the general idea is true for any other area because again things in the world are related and inter-connected. There is probably nothing that is completely unrelated to anything else. Thus knowledge learned in the past in some domains can be applied in some other domains in similar contexts in the future. The classic isolated learning paradigm is unable to perform such lifelong learning. Isolated learning is only suitable for narrow and restricted tasks. It is probably not sufficient for building an intelligent system that can learn continuously to achieve close to the human level of intelligence. LML aims to make progress in this direction. With the popularity of interactive robots, intelligent personal assistants, and chatbots, LML is becoming increasingly important because these systems have to interact with humans and/or other systems, learn constantly in the process, and retain and accumulate the knowledge learned in the interactions in the ever changing environments to enable them to learn more and learn better over time and to function seamlessly.

1.1 A BRIEF HISTORY OF LIFELONG LEARNING

The concept of *lifelong machine learning* (LML) was proposed around 1995 in [Thrun and Mitchell, 1995]. Since then it has been researched in four main areas. Here we give a brief history of the LML research in each of these areas.

1. *Lifelong Supervised Learning.* Thrun [1996b] first studied lifelong concept learning, where each previous or new task aims to recognize a particular concept or class using binary classification. Several LML techniques were proposed in the contexts of memory-based learning and neutral networks. The neural network approach was improved in [Silver and Mercer, 1996, 2002, Silver et al., 2015]. Fei et al. [2016] extended

this form of LML to *cumulative learning*, which, on encountering a new class or concept, builds a new multi-class classifier that can classify all the previous and the new classes by efficiently updating the old classifier. It can also detect unseen classes, which enables a limited form of self-motivated learning. Ruvolo and Eaton [2013b] proposed an efficient lifelong learning algorithm to improve the multi-task learning method in [Kumar et al., 2012]. Here the learning tasks are independent of each other. Ruvolo and Eaton [2013a] considered ELLA in an active task selection setting. Chen et al. [2015] further proposed an LML technique in the context of Naïve Bayesian classification. A theoretical study of LML was done by Pentina and Lampert [2014] in the PAC-learning framework.

2. *Lifelong Unsupervised Learning*. Papers in this area are mainly about lifelong topic modeling and lifelong information extraction. Chen and Liu [2014a,b] and Wang et al. [2016] proposed several lifelong topic modeling techniques that mine knowledge from topics produced from many previous tasks and use it to help generate better topics in the new task. Liu et al. [2016] proposed an LML approach based on recommendation for information extraction in the context of opinion mining. Shu et al. [2016] proposed a lifelong relaxation labeling method to solve a unsupervised classification problem.

3. *Lifelong Semi-Supervised Learning*. The work in this area is represented by the NELL (*Never-Ending Language Learner*) system [Carlson et al., 2010a, Mitchell et al., 2015], which has been reading the Web continuously for information extraction since January 2010, and it has accumulated millions of entities and relations.

4. *Lifelong Reinforcement Learning*. Thrun and Mitchell [1995] first proposed some LML algorithms for robot learning which tried to capture the invariant knowledge about each individual task. Tanaka and Yamamura [1997] treated each environment as a task for LML. Ring [1998] proposed a continual-learning agent that aims to gradually solve complicated tasks by learning easy tasks first. Wilson et al. [2007] proposed a hierarchical Bayesian lifelong reinforcement learning method in the framework of Markov Decision Process (MDP). Fernández and Veloso [2013] worked on policy reuse in a multi-task setting. A nonlinear feedback policy that generalizes across multiple tasks is proposed in [Deisenroth et al., 2014]. Bou Ammar et al. [2014] proposed a policy gradient efficient lifelong learning algorithm following the idea in ELLA [Ruvolo and Eaton, 2013b]. This work was further enhanced with cross-domain lifelong reinforcement learning [Bou Ammar et al., 2015a] and with constraints for safe lifelong reinforcement learning [Bou Ammar et al., 2015c].

LML techniques working in other areas also exist. For example, Kapoor and Horvitz [2009] studied predictive user modeling under LML and Kapoor and Horvitz [2009] worked on managing and using user feedback with the help of LML. Silver et al. [2013] wrote a

survey of LML trying to encourage more researchers to work on LML. The survey was presented at the AAAI 2013 Spring Symposium on Lifelong Machine Learning.

As we can see, although LML has been proposed for more than 20 years, research in the area has not been extensive. There could be many reasons. Some of the reasons may be as follows: First, the machine learning research for the past 20 years has focused on statistical and algorithmic approaches. LML typically needs a systems approach that combines multiple components and learning algorithms. Systems approaches to learning were not in favor. This may partially explain that although the LML research has been limited, closely related paradigms of transfer learning and multi-task learning have been researched fairly extensively because they can be done in statistical and algorithmic fashion. Second, much of the past machine learning research and applications focused on supervised learning using structured data, which are not easy for LML because there is little to be shared across tasks and domains. For example, the knowledge learned from a supervised learning system on a loan application is hard to be used in a health or education application because they do not have much in common. Also, most supervised learning algorithms generate no additional knowledge other than the final model or classifier, which is difficult to use as prior knowledge for another classification task even in similar domains. Third, many effective machine learning methods such as SVM and deep learning cannot easily use prior knowledge even if such knowledge exists. These classifiers are black boxes and hard to decompose or interpret. They are generally more accurate with more training data. Fourth, related areas such as transfer learning and multi-task learning were popular partly because they typically need only two and just a few similar tasks and datasets and do not require retention of explicit knowledge. LML, on the other hand, needs significantly more previous tasks and data in order to learn and to accumulate a large amount of explicit knowledge so that the new learning task can pick and choose the suitable knowledge to be used to help the new learning. This is analogous to human learning. If one does not have much knowledge, it is very hard for him/her to learn more knowledge. The more knowledge that one has, the easier it is for him/her to learn even more. For example, it is close to impossible for an elementary school pupil to learn graphical models. Even for an adult, if he has not studied probability theory, it is also infeasible for him to learn graphical models either.

Considering these factors, we believe that one of the more promising areas for LML is text mining or natural language processing (NLP) due to its extensive sharing of knowledge across domains and tasks and inter-relatedness of NLP tasks as we discussed above. The text data is also abundant. Lifelong supervised, unsupervised, semi-supervised, and reinforcement learning can all be applied to text data.

1.2 DEFINITION OF LIFELONG LEARNING

The earlier definition of LML is as follows [Thrun, 1996b]: The system has performed N tasks. When faced with the $(N+1)$th task, it uses the knowledge gained from the N

tasks to help the $(N+1)$th task. Here we extend this definition by giving it more details, mainly by adding an explicit *knowledge base* (KB) to stress the importance of knowledge accumulation and meta-mining of additional higher-level knowledge from the knowledge retained from previous learning. We will improve this early definition shortly.

Definition 1.1 *Lifelong machine learning* (LML) is a continuous learning process. At any time point, the learner has performed a sequence of N learning tasks, $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_N$. These tasks, which are also called the *previous tasks*, have their corresponding datasets $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_N$. The tasks can be of different *types* and from different *domains*. When faced with the $(N+1)$th task $\mathcal{T}_{N+1}$ (which is called the *new* or *current task*) with its data $\mathcal{D}_{N+1}$, the learner can leverage the *past knowledge* in the *knowledge base* (KB) to help learn $\mathcal{T}_{N+1}$. The objective of LML is usually to optimize the performance on the new task $\mathcal{T}_{N+1}$, but it can optimize on any task by treating the rest of the tasks as the previous tasks. KB maintains the knowledge learned and accumulated from learning the previous tasks. After the completion of learning $\mathcal{T}_{N+1}$, KB is updated with the knowledge (e.g., intermediate as well as the final results) gained from learning $\mathcal{T}_{N+1}$. The updating can involve consistency checking, reasoning, and meta-mining of additional higher-level knowledge.

Since this definition is quite general, some remarks are in order:

1. The definition shows three key characteristics of LML: (1) continuous learning, (2) knowledge accumulation and maintenance in the knowledge base (KB), and (3) the ability to use the past knowledge to help future learning. That is, the lifelong learner learns a series of tasks, possibly never ending, and in the process, it becomes more and more knowledgeable, and better at learning. These characteristics make LML different from related learning paradigms such as transfer learning [Jiang, 2008, Pan and Yang, 2010, Taylor and Stone, 2009] and multi-task learning [Caruana, 1997, Chen et al., 2009, Lazaric and Ghavamzadeh, 2010], which do not have one or more of these characteristics. We will discuss these related paradigms and their differences from the LML paradigm in detail in Chapter 2.

2. The tasks do not have to be from the same domain. There is no unified definition of a *domain* in the literature that is applicable to all areas. In most cases, the term is used informally to mean a *subject area* where there are often multiple different tasks of the same type or of different types (e.g., information extraction, coreference resolution, and entity linking). Some researchers even use domain and task interchangeably because there is only one task from each domain in their study. We also use them interchangeably in many cases in this book due to the same reason but will distinguish them when needed.

3. The shift to the new task can happen abruptly or gradually. The tasks and their data do not have to be provided by some external systems or human users. Ideally,

a lifelong learner can find its own learning tasks and training data in its interaction with the environment by performing self-motivated learning. For example, a service robot in a hotel may be trained to recognize the faces of a group of guests initially in order to greet them, but in its interaction with the guests it may find a new guest whom it does not recognize. It can then take some pictures and learn to recognize him/her and associate him/her with a name obtained by asking the guest. In this way, the robot can greet the new guest next time in a personalized manner.

4. The definition does not give details about knowledge or its representation in the knowledge base (KB) because of our limited understanding. Current papers use only one or two specific types of knowledge suitable for their proposed techniques. The problem of knowledge representation is still an active research topic with few mature results for practical use. The definition also does not specify how to maintain and update the knowledge base. For a particular application, one can design a KB based on the application need. We will discuss some possible components of the KB below.

5. The definition indicates that LML may require a *systems approach* that combines multiple learning algorithms and different knowledge representation schemes. It is not likely that a single learning algorithm is able to achieve the objective of LML.

6. There is still no generic LML system that is able to perform LML in all possible domains for all possible types of tasks. In fact, we are far from that. That is, unlike many machine learning algorithms such as SVM and deep learning, which can be applied to any learning task as long as the data is represented in a specific format. Current LML algorithms are still quite specific to some types of tasks and data.

We now discuss the improved definition of LML, which is mainly due to the work in Fei et al. [2016], Shu et al. [2017a,b]

Definition 1.2 *Lifelong machine learning* (LML) is a continuous learning process. At any time point, the learner has performed a sequence of N learning tasks, $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N$. These tasks, which are also called the *previous tasks*, have their corresponding datasets $\mathcal{D}_1$, $\mathcal{D}_2$, $\ldots$, $\mathcal{D}_N$. The tasks can be of different *types* and from different *domains*. When faced with the $(N+1)$th task $\mathcal{T}_{N+1}$ (which is called the *new* or *current task*) with its data $\mathcal{D}_{N+1}$, the learner can leverage the *past knowledge* in the *knowledge base* (KB) to help learn $\mathcal{T}_{N+1}$. Note that the task can be given or discovered by the system itself (see below). The objective of LML is usually to optimize the performance on the new task $\mathcal{T}_{N+1}$, but it can optimize on any task by treating the rest of the tasks as the previous tasks. KB maintains the knowledge learned and accumulated from learning the previous tasks. After the completion of learning $\mathcal{T}_{N+1}$, KB is updated with the knowledge (e.g., intermediate as well as the final results) gained from learning $\mathcal{T}_{N+1}$. The updating can involve consistency checking, reasoning, and

meta-mining of additional higher-level knowledge. Furthermore, a LML learner should also be able to:

1. learn and function in the open environment, where it not only can apply the learned model or knowledge to solve problems but also discover new problems, which form new tasks to be learned.

2. learn to improve the model performance in the application or testing of the learned knowledge or model. This is like that after job training, we still learn on the job to become better and better at doing the job.

Without these capabilities, a LML system will not be able to learn new knowledge or function in a dynamic open environment by itself. An AI system will never be truly intelligent. By *open environment*, we mean that the application environment may contain novel objects that have not been learned before. For example, we want to build a greeting robot for hotels. At any point in time, the robot has learned to recognize all existing hotel guests. When it sees an existing guest, it can call his/her name and chat. At the same time, it must also detect any new guests that it has not seen before. On seeing a new guest, it can say hello, ask for his/her name, take many pictures, and learn to recognize the guest. Next time when it sees the person again, it can call his/her name and chat like an old friend. The real-world road environment for self-driving cars is another very typical dynamic and open environment. We believe that without the lifelong learning capability, self-diving cars are unlikely to work well in the real-life environment.

There are currently two main LML approaches at the high level based on the type of past knowledge that they exploit to learn the new task. We distinguish two kinds of past or shared knowledge (note that only shared knowledge can be used cross tasks):

1. *Global knowledge*: Many existing LML methods assume that there is a *global latent structure* among tasks that are shared by all tasks [Bou Ammar et al., 2014, Ruvolo and Eaton, 2013b, Tanaka and Yamamura, 1997, Thrun, 1996b, Wilson et al., 2007] (Sections 3.2, 3.5, 6.1, 6.2, and 6.3). This global structure can be learned and leveraged in the new task learning. These approaches mainly grew out of multi-task learning.

2. *Local knowledge*: Many other methods do not assume such a global latent structure among tasks [Chen and Liu, 2014a,b, Chen et al., 2015, Fei et al., 2016, Liu et al., 2016, Shu et al., 2016] (Sections 3.4, 3.6, 4.2, 4.3, 4.4, and 4.5). Instead, during the learning of a new task they pick and choose the pieces of knowledge learned from previous tasks to use based on the need of the specific task. This means that different tasks may use different pieces of knowledge learned from different previous tasks. We call such pieces of knowledge the *local knowledge* because they are local to their individual previous tasks.

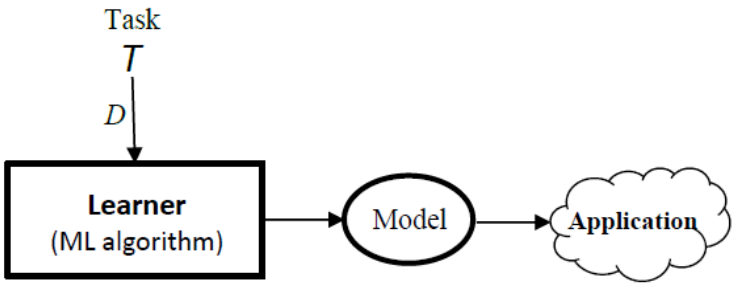


Figure 1.1: The classic machine learning paradigm.

We can also distinguish two types of tasks.

1. *Independent tasks*: Every task $\mathcal{T}_i$ is independent of the other tasks. This means that each task can be learned independently, although due to their similarities and sharing of some latent structures or knowledge, learning $\mathcal{T}_i$ can leverage the knowledge gained from previous tasks. Most existing LML methods except that in [Fei et al., 2016] (Section 3.4) use independent tasks.

2. *Dependent tasks*: Each task $\mathcal{T}_i$ has some dependence on some other tasks. For example, Fei et al. [2016] (Section 3.4) proposed a specific setting where each new supervised learning task adds a new class to the previous classification problem, and needs to build a new multi-class classifier which can classify data from all previous and current classes. The authors of the paper called this problem *cumulative learning*.

1.3 LIFELONG LEARNING SYSTEM ARCHITECTURE

Before presenting the LML system architecture, we first give the architecture of the classic machine learning paradigm, which is given in Figure 1.1, where there is only a single task T with its data D. The learned model is used in its intended application.

From Definition 1.1 and the associated marks, we can see a general process of LML and also an LML system architecture. Figure 1.2 illustrates the architecture and the process. Below, we first describe the key components of the system and then discuss the LML process. We note that this general architecture is for illustration purposes. Not all existing systems use all the components or sub-components. In fact, most current systems are much simpler.

1. **Knowledge Base (KB)**: It mainly stores the previously learned knowledge. It has a few sub-components:

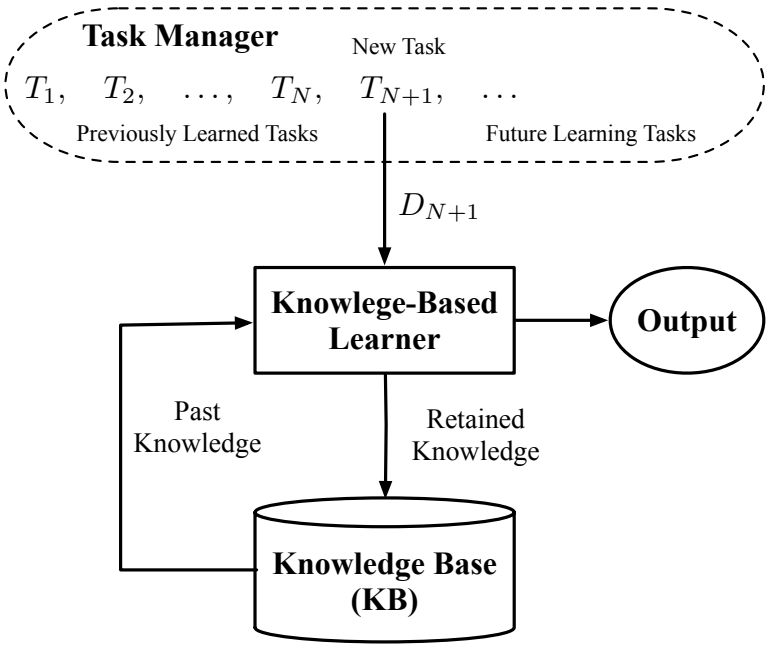


Figure 1.2: The lifelong machine learning system architecture.

(a) *Past Information Store* (PIS): It stores the information resulted from the past learning, including the resulting models, patterns, or other forms of outcome. PIS may also involve sub-stores for information such as (1) the original data used in each previous task, (2) intermediate results from each previous task, and (3) the final model or patterns learned from each previous task. As for what information or knowledge should be retained, it depends on the learning task and the learning algorithm. For a particular system, the user needs to decide what to retain in order to help future learning.

(b) *Meta-Knowledge Miner* (MKM). It performs meta-mining of the knowledge in the PIS and in the meta-knowledge store (see below). We call this *meta-mining* because it mines higher-level knowledge from the saved knowledge. The resulting

knowledge is stored in the Meta-Knowledge Store. Here multiple mining algorithms may be used to produce different types of results.

(c) *Meta-Knowledge Store* (MKS): It stores the knowledge mined or consolidated from PIS (Past Information Store) and also from MKS itself. Some suitable knowledge representation schemes are needed for each application.

(d) *Knowledge Reasoner* (KR): It makes inference based on the knowledge in MKB and PIS to generate more knowledge. Most current systems do not have this sub-component. However, with the advance of LML, this component will become increasingly important.

Since the current LML research is still in its infancy, as indicated above, none of the existing systems has all these sub-components.

2. **Knowledge-Based Learner (KBL)**: For LML, it is necessary for the learner to be able to use prior knowledge in learning. We call such a learner a *knowledge-based learner*, which can leverage the knowledge in the KB to learn the new task. This component may have two sub-components: (1) *Task knowledge miner* (TKM), which makes use of the raw knowledge or information in the KB to mine or identify knowledge that is appropriate for the current task. This is needed because in some cases, KBL cannot use the raw knowledge in the KB directly but needs some task specific and more general knowledge mined from the KB [Chen and Liu, 2014a,b]. (2) The *learner* that can make use of the mined knowledge in learning.

3. **Output**: This is the learning result for the user, which can be a prediction model or classifier in supervised learning, clusters or topics in unsupervised learning, etc. Note that the Output node can be replaced with two nodes as in Figure 1.3.

4. **Task Manager (TM)**: It receives and manages the tasks that arrive in the system, and handles the task shift and presents the new learning task to the KBL to start the LML process.

Enhanced LML Architecture: With the enhanced LML Definition 1.2, the LML architecture becomes that in Figure 1.4. It now can discover new problems from the application process which form new tasks to be learned (upper right corner) Fei et al. [2016], Shu et al. [2017b]. It also can improve the model performance during the application process (lower right corner) Shu et al. [2017a].

Lifelong Learning Process: A typical lifelong learning process starts with the Task Manager assigning a new task to the KBL. KBL then works with the help of the past knowledge stored in the KB to produce the output (e.g., a model) to the user and also send the information or knowledge that needs to be retained for future use to the KB.

LML is highly challenging because what knowledge to retain, how to use previous knowledge, and how to maintain the KB are all difficult problems. Below, we highlight two

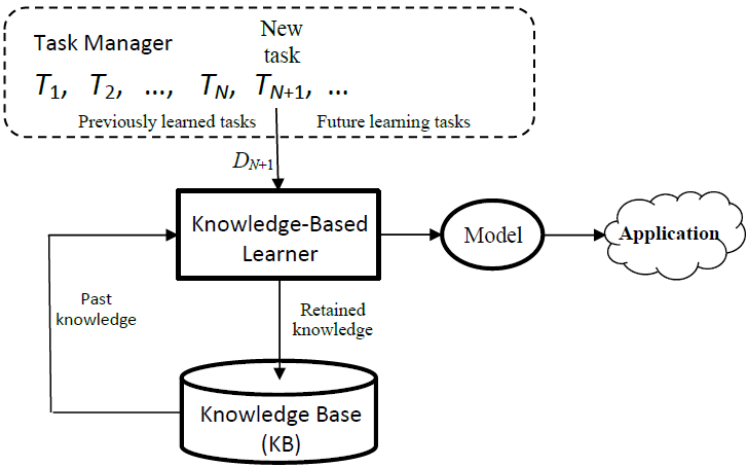


Figure 1.3: Replacing the Output node with the Model and Application nodes.

hidden but fundamental challenges of LML based on our experiences with several projects. We will describe how the existing research deals with these challenges throughout the book.

1. *Correctness of knowledge*: Incorrect knowledge is detrimental to the new learning. LML can be regarded as a continuous bootstrapping process. Errors can propagate from previous tasks to subsequent tasks to generate more and more errors. But we humans seem to have a good idea of what is correct or what is wrong.

2. *Applicability of knowledge*. Although a piece of knowledge may be correct in the context of some previous tasks, it may not be applicable to the current task. Application of inappropriate knowledge has the same negative consequence as the above case. Again, we humans are quite good at recognizing the right context for a piece of knowledge.

1.4 EVALUATION METHODOLOGY

Unlike the classic isolated learning where the evaluation of a learning algorithm is based on training and testing using data from the same task domain, LML needs a different evaluation methodology because it involves a sequence of tasks and we want to see improvements in the learning of later tasks. Experimental evaluation of an LML algorithm in the current research is commonly done using the following steps:

1. *Run on the data from the previous tasks*: We first run the algorithm on the data from a set of previous tasks, one at a time in a given sequence, and retain the knowledge

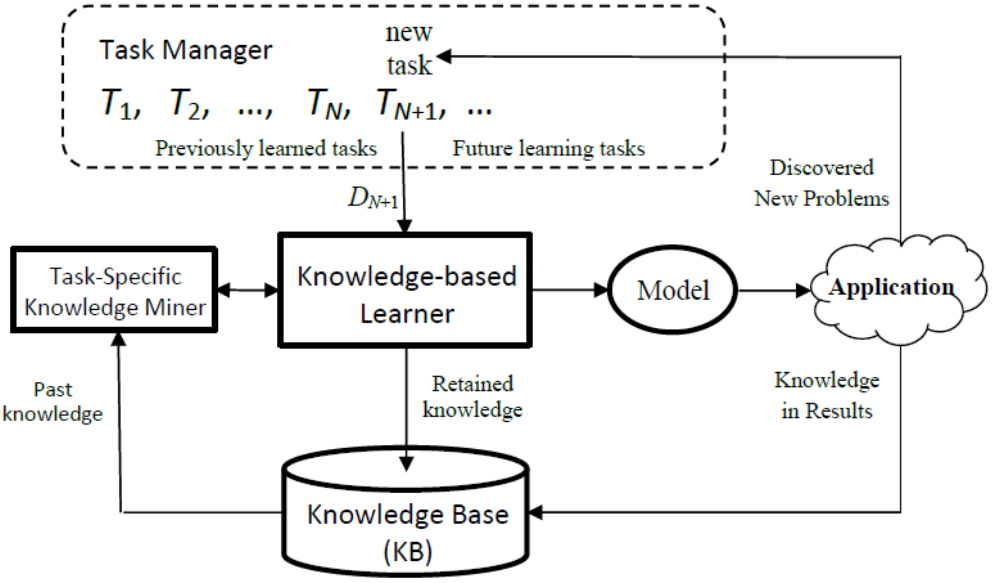


Figure 1.4: The enhanced LML system architecture.

gained in the knowledge base (KB). Clearly, there can be multiple variations or versions of the algorithm (e.g., with different types of knowledge used and more or less knowledge used) which can be experimented with.

2. *Run on the data from the new task*: We then run the algorithm on the new task data by leveraging the knowledge in the KB.

3. *Run baseline algorithms*: For comparison, we run some baseline algorithms. There are usually two kinds of baselines. The first kind of baselines are algorithms that perform isolated learning on the new data without using any past knowledge. The second kind of baselines are existing LML algorithms.

4. *Analyze the results*: This step compares the results from steps 2 and 3 and analyzes the results to make some observations, e.g., to show the results from the LML algorithm in step 2 is superior to those from the baselines in step 3.

There are several additional considerations in carrying out an LML experimental evaluation.

1. *A large number of tasks*: A large number of tasks and datasets are needed to evaluate an LML algorithm. This is because the knowledge gained from a few tasks may not be able to improve the learning of the new task much as each task may only provide

a very small amount of knowledge that is useful to the new task (unless all the tasks are very similar) and the data in the new task is often quite small.

2. *Task sequence*: The sequence of the tasks to be learned can be significant, meaning that different task sequences can generate different results. This is so because LML algorithms typically do not guarantee optimal solutions for all previous tasks. To take the sequence issue into consideration in the experiment, one can try several random sequences of tasks and generate results for the sequences. The results can then be averaged for comparison purposes. Existing papers mainly use only one random sequence in their experiments.

3. *Progressive experiments*: Since more previous tasks should generate more knowledge than fewer previous tasks, and more knowledge in turn should enable an LML algorithm to produce better results for the new task, it is thus desirable to show how the algorithm behaves on the new task as the number of previous tasks increases.

Finally, we note that it is not our intention to cover all possible kinds of evaluations in the current research on LML. Our purpose is simply to introduce the common evaluation methodology of an LML algorithm. In evaluating a specific algorithm, the researcher has to consider the special characteristics of the algorithm (such as its assumptions and usage settings) and the related research in order to design a set of good experiments.

1.5 ROLE OF BIG DATA IN LIFELONG LEARNING

In social sciences such as education, cognitive science, and philosophy, it is common knowledge that like the "Matthew effect" of *the rich get richer*, the more you know the more you can learn. The more you know, the easier it is for you to learn something new. If we do not know anything, it is very hard to learn anything new. These are intuitive as each one of us must have experienced this in our lives. For example, it is probably impossible to teach a person how to solve differential equations if the person has only primary school education. It is also important to learn from a wide range of domains, which gives us a wider vocabulary and a wide range of knowledge to be incorporated in learning and problem solving so that it is easier for us to learn in diverse domains. These should be the same for computer systems. To learn and accumulate a large quantity of knowledge, the system needs a large volume of diverse data. Fortunately, such big datasets are now readily available, which should allow a computer system to learn and acquire a broad range of knowledge continuously to become more and more knowledgeable, and more and more effective at learning and problem solving. This is the goal of lifelong learning.

1.6 OUTLINE OF THE BOOK

This book surveys and introduces this important and emerging field. Although the body of literature is not particularly large, related papers have been published in a large number of conferences and journals. There is also a large number of papers that do not exhibit all the characteristics of LML, but are related to some extent. It is thus hard, if not impossible, to cover all of the important work in the field. This book should not be taken to be an exhaustive account of everything in the field.

The book is organized as follows. In Chapter 2, we discuss some related machine learning paradigms to set the background. We will see that these paradigms are different from LML because they lack one or more of the key characteristics of LML. However, all these paradigms involve some forms of knowledge sharing or transfer across tasks or even can be made continuous learning in some cases. Thus, we regard LML as a more advanced paradigm that extends these related paradigms in the progression to make ML more intelligent and closer to human learning.

In Chapter 3, we focus on discussing existing research on supervised LML, where we will give fairly detailed descriptions of earlier and more recent supervised LML methods. In Chapter 4, we present unsupervised LML, where we focus on two lifelong topic models, a lifelong information extraction method, and a lifelong relaxation labeling method. Chapter 5 focuses on lifelong semi-supervised learning, where we give an overview of the NELL system (Never-Ending Language Learner). Lifelong reinforcement learning is covered in Chapter 6. In Chapter 7, we conclude the book and discuss some challenges and future directions of the lifelong learning research.

CHAPTER 2

Related Learning Paradigms

As described in the introduction chapter, lifelong machine learning (LML) (or simply lifelong learning) has three key characteristics: continuous learning process, explicit knowledge retention and accumulation, and the use of previously learned knowledge to help new learning tasks. There are several machine learning paradigms that have related characteristics. This chapter discusses the most related ones, i.e., transfer learning or domain adaption, multi-task learning, online learning, and reinforcement learning. The first two paradigms are more closely related to LML because they both involve some kind of knowledge transfer across domains or tasks, but they don't learn continuously and don't retain or accumulate learned knowledge explicitly. Online learning and reinforcement learning involves continuous learning processes but they focus on the same learning task with a time dimension. These differences will become clearer after we review some representative techniques for each of these related learning paradigms.

2.1 TRANSFER LEARNING

Transfer learning is a popular topic of research in machine learning and data mining. It is also commonly known as *domain adaptation* in natural language processing. It usually involves two domains: a *source domain* and a *target domain*. Although there can be more than one source domain, in almost all existing research only one source domain is used. The source domain normally has a large amount of labeled training data while the target domain has little or no labeled training data. The goal of transfer learning is to use the labeled data in the source domain to help learning in the target domain (see three excellent surveys of the area [Jiang, 2008, Pan and Yang, 2010, Taylor and Stone, 2009]). Note that in the literature, some researchers also use the terms *source task* and *target task* rather than *source domain* and *target domain*, but by far, the latter terminologies are more commonly used as the source and the target tasks are often from different domains or quite different distributions [Pan and Yang, 2010].

There are many types of knowledge that can be transferred from the source domain to the target domain to help learning in the target domain. For example, Bickel et al. [2007], Dai et al. [2007b,c], Jiang and Zhai [2007], Liao et al. [2005], and Sugiyama et al. [2008] directly treated certain parts of data instances in the source domain as the knowledge with instance reweighing and importance sampling and transfer it over to the target domain. Ando and Zhang [2005], Blitzer et al. [2006, 2007], Dai et al. [2007a], Daume III [2007], and

Wang and Mahadevan [2008] used features from the source domain to generate new feature representations for the target domain. Bonilla et al. [2008], Gao et al. [2008], Lawrence and Platt [2004], and Schwaighofer et al. [2004] transferred learning parameters from the source domain to the target domain. To give a flavor of transfer learning, we briefly discuss some existing transfer learning methods below.

2.1.1 STRUCTURAL CORRESPONDENCE LEARNING

One of the popular transfer learning techniques is the Structural Correspondence Learning (SCL) proposed in [Blitzer et al., 2006, 2007]. This method is mainly used in text classification. The algorithm works as follows: given labeled data from the source domain and unlabeled data from both the source and target domains, SCL tries to find a set of *pivot* features that have the same characteristics or behaviors in both domains. If a non-pivot feature is correlated with many of the same pivot features across different domains, this feature is likely to behave similarly across different domains. For example, if a word w co-occurs very frequently with the same set of *pivot* words in both domains, then w is likely to behave the same (e.g., holding the same semantic meaning) across domains.

To implement the above idea, SCL first chooses a set of m features which occur frequently in both domains and are also good predictors of the source label (in their paper these were the features with the highest mutual information with the source label). These pivot features represent the shared feature space of the two domains. SCL then computes the correlations of each pivot feature with other non-pivot features in both domains. This produces a correlation matrix $\mathbf{W}$ where row i is a vector of correlation values of non-pivot features with the ith pivot feature. Intuitively, positive values indicate that those non-pivot features are positively correlated with the ith pivot feature in the source domain or in the target domain. This establishes a feature correspondence between the two domains. After that, singular value decomposition (SVD) is employed to compute a low-dimensional linear approximation θ (the top k left singular vectors, transposed) of $\mathbf{W}$. The final set of features for training and for testing is the original set of features $\mathbf{x}$ combined with $\theta\mathbf{x}$ which produces k real-valued features. The classifier built using the combined features and the labeled data in the source domain should work in both the source and the target domains.

Pan et al. [2010] proposed a method similar to SCL at the high level. The algorithm works in the setting where there are only labeled examples in the source domain and unlabeled examples in the target domain. It bridges the gap between the domains by using a spectral feature alignment (SFA) algorithm to align domain-specific words from different domains into some unified clusters, with domain independent words as the bridge. Domain-independent words are like pivot words above and can be selected similarly.

2.1.2 NAÏVE BAYES TRANSFER CLASSIFIER

Many transfer learning methods have been proposed in the context of Naïve Bayesian classification [Chen et al., 2013a, Dai et al., 2007b, Do and Ng, 2005, Rigutini et al., 2005]. Here we briefly describe the work in [Dai et al., 2007b] to give a favor of such methods.

Dai et al. [2007b] proposed a method called *Naïve Bayes Transfer Classifier* (NBTC). Let the labeled data from the source domain be $\mathcal{D}_l$ with the distribution $\mathfrak{D}_l$, and the unlabeled data from the target domain be $\mathcal{D}_u$ with the distribution $\mathfrak{D}_u$. $\mathfrak{D}_l$ may not be the same as $\mathfrak{D}_u$. A two-step approach is employed in NBTC:

1. build an initial Naïve Bayesian classifier using the labeled data $\mathcal{D}_l$ under $\mathfrak{D}_l$ from the source domain.

2. run an EM (Expectation-Maximization) algorithm together with the target unlabeled data to find a local optimal model under the target domain distribution $\mathfrak{D}_u$.

The objective function of NBTC is as follows, which aims to find a local optimum of the *Maximum a Posteriori* (MAP) hypothesis under $\mathfrak{D}_u$:

$$h_{MAP} = \operatorname*{argmax}_h P_{\mathfrak{D}_u}(h) \times P_{\mathfrak{D}_u}(\mathcal{D}_l, \mathcal{D}_u|h) \ . \tag{2.1}$$

This equation considers the probability of the source domain labeled data and the target domain unlabeled data under the hypothesis h. The labeled data provides the supervised information while estimating the probability of the unlabeled data under $\mathfrak{D}_u$ ensures that the model fits for $\mathcal{D}_u$. Based on the Bayes' rule, NBTC maximizes the log-likelihood $l(h|\mathcal{D}_l, \mathcal{D}_u) = \log P_{\mathfrak{D}_u}(h|\mathcal{D}_l, \mathcal{D}_u)$,

$$\begin{aligned} l(h|\mathcal{D}_l, \mathcal{D}_u) \propto & \log P_{\mathfrak{D}_u}(h) \\ & + \sum_{d \in \mathcal{D}_l} \log \sum_{c \in C} P_{\mathfrak{D}_u}(d|c,h) \times P_{\mathfrak{D}_u}(c|h) \\ & + \sum_{d \in \mathcal{D}_u} \log \sum_{c \in C} P_{\mathfrak{D}_u}(d|c,h) \times P_{\mathfrak{D}_u}(c|h) \ , \end{aligned} \tag{2.2}$$

where C is the set of classes and $d \in \mathcal{D}_l$ is a document in $\mathcal{D}_l$. To optimize it, Dai et al. [2007b] applied the EM algorithm as follows:

- **E-Step**:
$$P_{\mathfrak{D}_u}(c|d) \propto P_{\mathfrak{D}_u}(c) \prod_{w \in d} P_{\mathfrak{D}_u}(w|c) \tag{2.3}$$

- **M-Step**:
$$P_{\mathfrak{D}_u}(c) \propto \sum_{i \in \{l,u\}} P_{\mathfrak{D}_u}(\mathcal{D}_i) \times P_{\mathfrak{D}_u}(c|\mathcal{D}_i) \tag{2.4}$$

$$P_{\mathfrak{D}_u}(w|c) \propto \sum_{i \in \{l,u\}} P_{\mathfrak{D}_u}(\mathcal{D}_i) \times P_{\mathfrak{D}_u}(c|\mathcal{D}_i) \times P_{\mathfrak{D}_u}(w|c, \mathcal{D}_i) \ , \tag{2.5}$$

where $w \in d$ represents a word in document d. $P_{\mathfrak{D}_u}(c|\mathcal{D}_i)$ and $P_{\mathfrak{D}_u}(w|c, \mathcal{D}_i)$ can be rewritten via the Naïve Bayesian classification formulation (see [Dai et al., 2007b] for more details). The above E-step and M-step are repeated to reach a local optimal solution.

Chen et al. [2013a] proposed two EM-type algorithms called FS-EM (Feature Selection EM) and Co-Class (Co-Classification). FS-EM uses feature selection as the mechanism to transfer knowledge from the source domain to the target domain in each EM iteration. Co-Class further adds the idea of co-training [Blum and Mitchell, 1998] to deal with the imbalance of the shared positive and negative features. It builds two Naïve Bayesian classifiers, one on labeled data, and the other on the unlabeled data with predicted labels. An earlier work for cross-language text classification also used a similar idea in the context of Naïve Bayesian classification [Rigutini et al., 2005], which transfers knowledge from the labeled data in English to the unlabeled data in Italian.

2.1.3 DEEP LEARNING IN TRANSFER LEARNING

In recent years, deep learning or deep neural network has emerged as a major learning method and has achieved very promising results [Bengio, 2009]. It has been used by several researchers for transfer learning.

For example, instead of using the traditional raw input as features which may not generalize well across domains, Glorot et al. [2011] proposed to use the low-dimensional features learned using deep learning to help prediction in the new domain. In particular, Stacked Denoising Auto-encoder of [Vincent et al., 2008] was employed in [Glorot et al., 2011]. In an auto-encoder, there are typically two functions: an encoder function $h()$ and a decoder function $g()$. The reconstruction of input x is given by $r(x) = g(h(x))$. To train an auto-encoder, the objective function is to minimize the reconstruction error $loss(x, r(x))$. Then, auto-encoders can be trained and stacked together as a hierarchical network. In this network, the auto-encoder at level i takes the output of the $(i-1)th$ auto-encoder as input. Level 0 takes the raw input. In denoising an auto-encoder, the input vector x is stochastically corrupted into another vector $\hat{x}$ and the objective function is to minimize a denoising reconstruction error loss $loss(x, r(\hat{x}))$. In [Glorot et al., 2011], the model is learned in a greedy layer-wise fashion using stochastic gradient descent. The first layer uses logistic sigmoid to transform the raw input. For the upper layers, the softplus activation function, $log(1 + exp(x))$, is used. After learning the auto-encoders, a linear SVM with squared hinge loss is trained on the labeled data from the source domain and tested on the target domain.

Yosinski et al. [2014] studied the transferability of features in each layer of a deep neural network. They argued that the lowest level or the raw input layer is very *general* as it is independent of the task and the network. In contrast, the features from the highest level depend on the task and cost function, and thus are *specific*. For example, in a supervised

learning task, each output unit corresponds to a particular class. From the lowest level to the highest level, there is a transfer from generality to specificity. To experiment the transferability of features in each layer in a deep neural network, they trained a neural network from the source domain and copy the first n layers to the neural network for the target domain. The remaining layers in the target neural network are randomly initialized. They showed that transferred features in the neural network from the source domain are indeed helpful to the target domain learning. Also in the transfer learning setting, Bengio [2012] focused on unsupervised pre-training of representations and discussed potential challenges of deep learning for transfer learning.

2.1.4 DIFFERENCE FROM LIFELONG LEARNING

Transfer learning is different from lifelong learning in the following aspects. We want to note that since the literature on transfer learning is extensive, the differences described here may not be applicable to every individual transfer learning paper.

1. Transfer learning is not concerned with continuous learning or knowledge accumulation. Its transfer of information or knowledge from the source domain to the target domain is only one-time. It does not retain the transferred knowledge or information for future use. LML, on the other hand, represents continuous learning. Knowledge retention and accumulation are essential for LML as they not only enable the system to become more and more knowledgeable, but also allow it to learn additional knowledge more accurately and easily in the future.

2. Transfer learning is unidirectional. It transfers knowledge only from the source domain to the target domain, but not the other way around because the target domain has little or no training data. However, in LML, the learning result from the new domain or task can be used to improve learning in previous domains or tasks if needed.

3. Transfer learning typically involves with only two domains, a source domain and a target domain (although in some cases there are more than one source domain). It assumes that the source domain is very similar to the target domain; otherwise the results can be detrimental. The two similar domains are usually selected by human users. LML, on the other hand, normally considers a large (possibly infinite) number of tasks/domains. In solving a new problem, the learner can pick and choose the appropriate past knowledge to be used in current learning. It does not have the assumption made by transfer learning. That is, in LML, if there is useful knowledge from the past, use it. If not, just learn using the current domain data. However, since LML typically involves a large number of past domains, the system has a large amount of past knowledge. The new learning task is very likely to find some pieces of the past knowledge useful.

2.2 MULTI-TASK LEARNING

Multi-task learning learns multiple related tasks simultaneously, aiming at achieving a better performance by using the relevant information shared by multiple tasks [Caruana, 1997, Chen et al., 2009, Li et al., 2009]. The rationale is to introduce inductive bias in the joint hypothesis space of all tasks by exploiting the task relatedness structure. It also prevents overfitting in the individual task and thus has a better generalization ability. Note that unlike that in transfer learning, we mostly use the term *multiple tasks* rather than *multiple domains* as much of the existing research in the area is based on multiple similar tasks from the same domain of application. We now define *multi-task learning*, which is also referred to as *batch multi-task learning*.

Definition 2.1 *Multi-task Learning* (MTL) is concerned with learning multiple tasks $\mathcal{T} = \{1, 2, \ldots, N\}$ simultaneously Each task $t \in \mathcal{T}$ has its training data $\mathcal{D}^t$. The goal is to maximize the performance across *all* tasks.

Since most existing works on multi-task learning focused on supervised learning, here we discuss only multi-task supervised learning. Let each task t have the training data $\mathcal{D}^t = \{(\mathbf{x}_i^t, y_i^t) : i = 1, \ldots, n_t\}$, where n_t is the number of training instances in $\mathcal{D}^t$. $\mathcal{D}^t$ is defined by a hidden (or latent) true mapping $\hat{f}^t(\mathbf{x})$ from an instance space $\mathcal{X}^t \subseteq \mathbb{R}^d$ to a set of labels $\mathcal{Y}^t$ ($y_i^t \in \mathcal{Y}^t$) (or $\mathcal{Y}^t = \mathbb{R}$ for regression). d is the feature/pattern dimension. We want to learn a mapping function $f^t(\mathbf{x})$ for each task t so that $f^t(\mathbf{x}) \approx \hat{f}^t(\mathbf{x})$. Formally, given a loss function $\mathcal{L}$, multi-task learning minimizes the following objective function:

$$\sum_{t=1}^{N} \sum_{i=1}^{n_t} \mathcal{L}\left(f(\mathbf{x}_i^t),\ y_i^t\right) \quad . \tag{2.6}$$

In contrast to this batch multi-task learning, *online multi-task learning* aims to learn the tasks sequentially and accumulate knowledge over time and leverage the knowledge to help subsequent learning (or to improve some previous learning task). Online multi-task learning is thus lifelong learning.

2.2.1 TASK RELATEDNESS IN MULTI-TASK LEARNING

Multi-task learning assumes that tasks are closely *related*. There are different assumptions in terms of *task relatedness*, which lead to different modeling approaches.

Evgeniou and Pontil [2004] assumed that all data for the tasks come from the same space and all the task models are close to a global model. Under this assumption, they modeled the relation between tasks using a task coupling parameter with regularization. Baxter [2000] and Ben-David and Schuller [2003] assumed that the tasks share a common underlying representation, e.g., using a common set of learned features. Some other works used

probabilistic approaches assuming that the parameters share a common prior [Daumé III, 2009, Lee et al., 2007, Yu et al., 2005].

Task parameters can also lie in a low dimensional subspace, which is shared across tasks [Argyriou et al., 2008]. Instead of assuming all tasks sharing the full space, Argyriou et al. [2008] assumed that they share a low rank of the original space. However, the low rank assumption does not distinguish tasks. When some unrelated tasks are considered, the performance can deteriorate. To address this issue, some papers assumed that there are disjoint groups of tasks and applied clustering to group tasks [Jacob et al., 2009, Xue et al., 2007]. The tasks within a cluster are considered close to each other. On the other hand, Yu et al. [2007] and Chen et al. [2011] assumed that there is a group of related tasks while the unrelated tasks are a small number of outliers. Gong et al. [2012] assumed that the related tasks share a common set of features while the outlier tasks do not. [Kang et al., 2011] incorporated grouping structures using regularization. However, each group's subspace does not overlap, meaning that the possible sharing structure between tasks from different groups is ignored.

Recently, Kumar et al. [2012] assumed that the parameter vector of each task is a linear combination of a finite number of underlying basis or latent components. Instead of using the assumption of disjoint task groups [Jacob et al., 2009, Xue et al., 2007], they assumed that the tasks in different groups can overlap with each other in one or more bases. Based on this idea, they proposed a multi-task learning model called GO-MTL. We detail it in the next subsection. Maurer et al. [2013] proposed to use sparse coding and dictionary learning in multi-task learning. Extending GO-MTL, Ruvolo and Eaton [2013b] proposed the Efficient Lifelong Learning Algorithm (ELLA) that dramatically improves the efficiency and makes it an online multi-task learning method, which is regarded as a lifelong learning method as it satisfies the LML definition. We will introduce ELLA in Section 3.5.

2.2.2 GO-MTL: MULTI-TASK LEARNING USING LATENT BASIS

GO-MTL (Grouping and Overlap in Multi-Task Learning) [Kumar et al., 2012] takes a parametric approach to model building in which the model or the prediction function $f^t(\mathbf{x}) = f^t(\mathbf{x}; \boldsymbol{\theta}^t)$ for each task t is governed by the task-specific parameter vector $\boldsymbol{\theta}^t \in \mathbb{R}^d$, where d is the dimension of the data. Given N tasks, GO-MTL assumes that there are k ($< N$) *latent basis model components* among the models of the multiple tasks. Each basis model component $\mathbf{l}$ is represented by a vector of size d. The k basis model components are represented by a $d \times k$ matrix $\mathbf{L} = (\mathbf{l}_1, \ldots, \mathbf{l}_k)$. The parameter vector $\boldsymbol{\theta}^t$ of the model for each task t is assumed to be a linear combination of a subset of k basis model components and the weight vector $\mathbf{s}^t$, i.e., $\boldsymbol{\theta}^t = \mathbf{L}\mathbf{s}^t$, and $\mathbf{s}^t$ is assumed to be sparse. Considering all the tasks, we have:

$$\underset{d\times N}{\boldsymbol{\Theta}} = \underset{d\times k}{\mathbf{L}} \times \underset{k\times N}{\mathbf{S}} \quad , \tag{2.7}$$

where $\mathbf{\Theta} = [\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots, \boldsymbol{\theta}^N]$ and $\mathbf{S} = [\boldsymbol{s}^1, \boldsymbol{s}^2, \ldots, \boldsymbol{s}^N]$.

The idea is that each task can be represented by *some* of the basis model components. This mechanism takes into consideration of both related and unrelated tasks. A pair of related tasks will lead to the overlapping of their linear weight vectors ($\mathbf{s}$) while two unrelated tasks can be distinguished via their little linear weight vector overlapping. Thus GO-MTL does not assume disjointed groups of tasks like [Jacob et al., 2009, Xue et al., 2007]. As discussed above, the disadvantage of disjoint groups is that the tasks from different groups will not have interactions with each other. However, it is possible that although the tasks are in different groups, they may be negatively correlated or they still share some information, both of which can be useful for multi-task learning. The partial overlap among tasks is thus allowed in GO-MTL, which is flexible in dealing with sophisticated task relatedness without strong assumptions.

Objective Function

Given the training data $\mathcal{D}^t$ for each task t, the objective function is to minimize the predictive loss over all tasks while encouraging the sharing of structures between the tasks, which is defined as follows:

$$\sum_{t=1}^{N} \sum_{i=1}^{n_t} \mathcal{L}\left(f(\mathbf{x}_i^t; \mathbf{L}\mathbf{s}^t), y_i^t\right) + \mu \left\|\mathbf{S}\right\|_1 + \lambda \left\|\mathbf{L}\right\|_F^2 \quad , \tag{2.8}$$

where $\mathcal{L}$ is the empirical loss function, $(\mathbf{x}_i^t, y_i^t)$ is the ith labeled instance in the training data for task t. The function f is $f(\mathbf{x}_i^t; \mathbf{L}\mathbf{s}^t) = \boldsymbol{\theta}^t \mathbf{x}_i^t = (\mathbf{L}\mathbf{s}^t)^{\mathrm{T}} \mathbf{x}_i^t$. $\|\cdot\|_1$ is the L_1 norm, which is controlled by μ as a convex approximation to the true vector sparsity. $\|\mathbf{L}\|_F^2$ is the Frobenius norm of matrix $\mathbf{L}$ and λ is the regularization coefficient for matrix $\mathbf{L}$.

Alternating Optimization

If the loss function $\mathcal{L}$ is convex, the objective function in Equation 2.8 is convex in $\mathbf{L}$ for a fixed $\mathbf{S}$, and convex in $\mathbf{S}$ for a fixed $\mathbf{L}$, but they are not jointly convex. Thus, the alternating optimization strategy is adopted to achieve a local minimum. For a fixed $\mathbf{L}$, the optimization function for $\mathbf{s}^t$ becomes:

$$\mathbf{s}^t = \underset{\mathbf{s}}{\operatorname{argmin}} \sum_{i=1}^{n_t} \mathcal{L}\left(f(\mathbf{x}_i^t; \mathbf{L}\mathbf{s}), y_i^t\right) + \mu \left\|\mathbf{s}\right\|_1 \quad . \tag{2.9}$$

On the other hand, for a fixed $\mathbf{S}$, the optimization function for $\mathbf{L}$ is:

$$\underset{\mathbf{L}}{\operatorname{argmin}} \sum_{t=1}^{N} \sum_{i=1}^{n_t} \mathcal{L}\left(f(\mathbf{x}_i^t; \mathbf{L}\mathbf{s}^t), y_i^t\right) + \lambda \left\|\mathbf{L}\right\|_F^2 \quad . \tag{2.10}$$

For optimization of Equation 2.9, the two-metric projection method in [Gafni and Bertsekas, 1984, Schmidt et al., 2007] was used in [Kumar et al., 2012]. Equation 2.10 has a closed form solution for squared loss function which is commonly used in regression problems. For classification problems, logistic loss and the Newton-Raphson method were used in optimization in [Kumar et al., 2012].

To initialize $\mathbf{L}$, each individual task's parameters are learned independently using their own data, which are stacked as columns in a weight matrix. The top-k left singular vectors of this weight matrix is used as the initial $\mathbf{L}$. The reason for this is that the singular vectors are the directions that capture the maximum variances of the task parameters.

2.2.3 DEEP LEARNING IN MULTI-TASK LEARNING

In recent years, deep neural network (DNN) has also been applied to multi-task learning. For example, Liu et al. [2015b] proposed a multi-task DNN for learning representations across multiple tasks. They considered two types of tasks, query classification and Web search ranking.

- For query classification, the model classifies whether a query belongs to a particular domain or not. In this work, the authors considered four domains ("Restaurant," "Hotel," "Flight," and "Nightlife"). A query can belong to multiple domains. These four domains are framed as four query classification tasks. The training data for query classification consists of pairs of query and label ($y_t = \{0, 1\}$ where t denotes a particular task or domain).

- For Web search ranking, given a query, the model ranks the documents by their relevance to the query. It is assumed that in the training data, there is at least one relevant document for each query.

In their proposed multi-task DNN model, the lower neural network layers are shared across multiple tasks while the top layers are task-dependent.

The Input Layer (l_1) is the word hash layer in which each word is hashed as a bag of n-grams of letters. For example, word *deep* is hashed as a bag of letter-trigrams {#-d-e, d-e-e, e-e-p, e-p-#} where # denotes the boundary. This method can hash the variations of the same word into the space close to each other, e.g., *politics* and *politician*.

Semantic-Representation Layer (l_2) maps l_1 into a 300 dimensional vector by:

$$l_2 = f(\mathbf{W}_1 \cdot l_1) \ , \tag{2.11}$$

where $\mathbf{W}_1$ is the weight matrix and f is defined as:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \ . \tag{2.12}$$

This layer is shared across multiple tasks.

Task-Specific Layer (l_3) for each task converts the 300 dimensional vector to a 128 dimensional vector that is task dependent for each task, using the following:

$$l_3 = f(\mathbf{W}_2^t \cdot l_2) \ , \tag{2.13}$$

where t denotes a particular task and $\mathbf{W}_2^t$ is another weight matrix. For a query classification task, the probability of a query belonging to a domain is obtained from l_3 using a sigmoid function $g(z) = \frac{1}{1+e^{-z}}$. For Web search ranking, the cosine similarity is used to compare layer l_3 of the query and each document. To learn the neural network, the mini-batch-based stochastic gradient descent (SGD) is used, which is an iterative algorithm. In each iteration, a task t is first randomly picked. Then a labeled instance from t is sampled and this labeled instance is used to update the neural network via SGD.

In [Collobert and Weston, 2008, Collobert et al., 2011], the authors proposed a unified neural network architecture for performing multiple natural language processing tasks, including part-of-speech tagging, chucking, name entity recognition, semantic role labeling, language modeling, and semantically related words (or synonyms) discovering. They built a deep neural network for all tasks jointly using weight-sharing. In the neural network, the first layer is for textual features of each word. The second layer extracts features from a sentence, treating the sentence as a sequence rather than a bag of words. The sequence is the input of the second layer. Long-distance dependencies between words in a sentence is captured by Time-Delay Neural Networks (TDNNs) [Waibel et al., 1989], which can model the effects between words outside a fixed window.

A classic TDNN layer converts a sequence $\mathbf{x}$ to another sequence $\mathbf{o}$ as follows:

$$\mathbf{o}^i = \sum_{j=1-i}^{n-i} \mathbf{L}_j \cdot \mathbf{x}_{i+j} \ , \tag{2.14}$$

where i denotes the time at which the ith word in the sentence is seen in TDNN (i.e., $\mathbf{x}_i$), n is the number of words in the sentence or the length of the sequence. $\mathbf{L}_j$ are the parameters of the layer. Similar to [Liu et al., 2015b], stochastic gradient descent is used to train the model, which repeatedly selects a task and one of its training examples to update the neural network.

Along a similar line, Huang et al. [2013a] applied deep neural network to multilingual data. They proposed a model called shared-hidden-layer multilingual DNN (SHL-MDNN), in which the hidden layers are shared across multiple languages. Furthermore, Zhang et al. [2014] applied deep multi-task learning to the problem of facial landmark detection by co-modeling the correlated tasks such as head pose estimation and facial attribute inference. There are also other applications of deep multi-task learning models to problems such as name error detection in speech recognition [Cheng et al., 2015], multi-label learning [Huang et al., 2013b], phoneme recognition [Seltzer and Droppo, 2013], and so on.

2.2.4 DIFFERENCE FROM LIFELONG LEARNING

The similarity of (batch) multi-task learning and lifelong learning is that they both aim to use some shared information across tasks to help learning. The difference is that multi-task learning is still working in the traditional paradigm. Instead of optimizing a single task, it optimizes several tasks simultaneously. If we regard the several tasks as one bigger task, it reduces to the traditional optimization which is actually the case in most optimization formulations of MTL. It does not accumulate any knowledge over time and it does not use the concept of continuous learning, which are the key characteristics of LML. Although one can argue that MTL can jointly optimize all tasks whenever a new task is added, it is quite difficult to optimize all tasks in the world simultaneously in a single process as they are too numerous and diverse. Some local and distributed optimizations are needed. Global optimization is also not efficient in terms of both the time and resources. Thus, it is important to retain knowledge to enable incremental learning of multiple tasks with the help of knowledge learned in the past from previous tasks. That is why we regard *online or incremental multi-task learning* as lifelong learning.

2.3 ONLINE LEARNING

Online learning (also known as incremental learning) is a learning paradigm where the training data points arrive in a sequential order. When a new data point arrives, the existing model is quickly updated to produce the best model so far. Its goal is thus the same as classic learning, i.e., to optimize the performance on the given learning task. It is normally used when it is computationally infeasible to train over the entire dataset or the practical applications cannot wait until a large amount of training data is collected. This is in contrast with the classic batch learning where all training data is available at the beginning.

In online learning, if whenever a new data point arrives re-training using all the available data is performed, it will be too expensive. Furthermore, during re-training, the model being used is already out of date. Thus, online learning methods are typically memory and run-time efficient due to the latency requirement in a real-world scenario.

There are a large number of existing online learning algorithms. For example, Kivinen et al. [2004] proposed some online learning algorithms for kernel-based learning like SVM. By extending the classic stochastic gradient descent, they developed computationally efficient online learning algorithms for classification, regression and novelty detection. Related online kernel classifiers were also studied in [Bordes et al., 2005].

Rather than using the traditional table data, Herbster et al. [2005] studied online learning on graphs. Their objective is to learn a function defined on a graph from a set of labeled vertices. One application of their problem is to predict users' preferences towards products in a social network. Ma et al. [2009] worked on the problem of detecting malicious Web sites using lexical and host-based features and URLs in an online setting. Mairal et al. [2009, 2010] proposed some online dictionary learning approaches for sparse coding, which

model data vectors as sparse linear combinations of some basic elements. Hoffman et al. [2010] also proposed an online variational Bayes algorithm for topic modeling.

Much of the online learning research focuses on one domain/task. Dredze and Crammer [2008] developed a multi-domain online learning method, which is based on parameter combination of multiple classifiers. In their setting, the model receives a new instance/example as well as its domain.

2.3.1 DIFFERENCE FROM LIFELONG LEARNING

Although online learning deals with future data in streaming or in a sequential order, its objective is very different from lifelong machine learning. Online learning still performs the same learning task over time. Its objective is to learn more efficiently with the data arriving incrementally. Lifelong learning, on the other hand, aims to learn from a sequence of different tasks, retain the knowledge learned so far, and use the knowledge to help future task learning. Online learning does not do any of these.

2.4 REINFORCEMENT LEARNING

Reinforcement Learning [Kaelbling et al., 1996, Sutton and Barto, 1998] is the problem where an agent learns actions through trial and error interactions with a dynamic environment. In each interaction step, the agent receives input that contains the current state of the environment. The agent chooses an action from a set of possible actions. The action changes the state of the environment. Then, the agent gets a value of this state transition, which can be reward or penalty. This process repeats as the agent learns a trajectory of actions to optimize its objective. The goal of reinforcement learning is to learn an optimal *policy* that maps states to actions that maximizes the long run sum of rewards. Details about various types of reinforcement learning tasks can be found in [Busoniu et al., 2010, Szepesvári, 2010, Wiering and Van Otterlo, 2012].

Transfer learning and multi-task learning have also been applied to reinforcement learning. For example, Banerjee and Stone [2007] demonstrated that feature-based value function transfer learning learns optimal policies faster than without knowledge transfer. Taylor et al. [2008] proposed a method to transfer data instances from the source to the target in a model-based reinforcement learning setting. A rule transfer method was also proposed for reinforcement learning [Taylor and Stone, 2007]. An excellent survey of transfer learning applied to reinforcement learning can be found in [Taylor and Stone, 2009].

Mehta et al. [2008] worked on multiple tasks sharing the same transition dynamics but different reward functions. Instead of fully observable experiments, Li et al. [2009] proposed a model-free multi-task reinforcement learning model for multiple partially observable stochastic environments. They proposed an off-policy batch algorithm to learn parameters in a regionalized policy representation. Lazaric and Ghavamzadeh [2010] assumed that in the multi-task reinforcement learning, only a small number of samples can be generated for

any given policy in each task. They grouped the tasks using similar structures and learn them jointly. They also assumed that tasks share structures via value functions which are sampled from a common prior.

Horde, an architecture for learning knowledge in reinforcement learning, was proposed in [Sutton et al., 2011]. Its knowledge is represented by a large number of approximate value functions. The reinforcement learning agent is decomposed into many sub-agents. The value function is approximated by the expected return for a trajectory of states and actions. The trajectory is obtained according to the policy of each sub-agent. The intuition is that each sub-agent is responsible for learning some partial information about interactions with the environment. The sub-agents can also use each other's results to achieve their own goals. The final decision of the agent is made by all sub-agents together. However, Sutton et al. [2011] focused on the same environment, which is related to but also different from lifelong learning. Along the lines of Horde, Modayil et al. [2014] modeled a generalization of the value function in reinforcement learning.

2.4.1 DIFFERENCE FROM LIFELONG LEARNING

A reinforcement learning agent learns by trial and error in its interactions with the environment which gives feedback or rewards to the agent. The learning is limited to one task and one environment. There is no concept of accumulating knowledge to help future learning tasks. Transfer and multi-task reinforcement learning paradigms have similar differences from lifelong learning as supervised transfer and multi-task learning discussed in Section 2.1.4 and Section 2.2.4.

2.5 SUMMARY

In this Chapter, we gave an overview of the main machine learning paradigms that are closely related to lifelong machine learning (LML) and described their differences from LML. In summary, we can regard LML as a generalization of or extension to these paradigms. The key characteristics of LML are the continuous learning process, knowledge accumulation in the knowledge base (KB), and the use of the past knowledge to help future learning. The related machine learning paradigms do not have one or more of these characteristics. In a nutshell, LML essentially tries to mimic the human learning process in order to overcome the limitations of the current isolated learning paradigm. Although we still do not understand the human learning process, that should not prevent us from making progresses in machine learning that exhibit some characteristics of human learning. From the next chapter, we review various existing LML research directions and representative algorithms and techniques.

CHAPTER 3

Lifelong Supervised Learning

This chapter presents existing techniques for lifelong supervised learning. We first use an example to show why the sharing of information across tasks is useful and how such sharing makes lifelong machine learning (LML) work. The example is about product review sentiment classification. The task is to build a classifier to classify a product review as expressing a positive or negative opinion. In the classic setting, we first label a large number of positive opinion reviews and negative opinion reviews and then run a classification algorithm such as SVM to build a classifier. In the LML setting, we assume that we have learned from many previous tasks (which may be from different domains). A task here has a set of reviews of a particular kind of product (a *domain*), e.g., camera, cellphone, or car. Let us use the naive Bayesian (NB) classification technique for classifier building. In NB classification, we mainly need the conditional probability of each word w given each class y (positive or negative), $P(w|y)$. When we have a task from a new domain D, the question is whether we need training data from D at all. It is well-known that the classifier built in one domain works poorly in another domain because words and language constructs used for expressing opinions in different domains can be quite different [Liu, 2012]. To make matters worse, the same word may express or indicate positive opinion in one domain but negative opinion in another. The answer to the question is *no* in some cases, but *yes* in some others.

The reason for the *no* answer is that we can simply append all training data from the past domains and build a classifier (probably the simplest LML method). This classifier can do wonders for some new domain tasks. It can classify dramatically better than the classifier trained using a modest number of training examples from the new domain D alone. This is because sentiment classification is mainly determined by words that express positive or negative opinions, called *sentiment words*. For example, *good*, *great*, and *beautiful* are positive sentiment words, and *bad*, *poor*, and *terrible* are negative sentiment words. These words are shared across domains and tasks, but in a particular domain only a small subset of them is used. After seeing the training data from a large number of domains, it is quite clear what words are likely to indicate positive or negative opinions. This means that the system already knows those positive and negative sentiment words and thus can do classification well without any in-domain training reviews from D. To some extent, this is similar to our human case. We don't need a single training positive or negative review to be able to classify reviews into positive and negative classes because we have accumulated so much knowledge in the past about how people praise and criticize things in natural language. Clearly, using one or two past domains for LML is not sufficient because sentiment words

used in these domains may be limited and may not even be useful to the new domain. Many non-sentiment words may be regarded as sentiment words incorrectly. Thus, big and diverse data holds a key for LML.

Of course, this simple method does not always work. That is the reason for the *yes* answer above (i.e., requiring some in-domain training data). The reason is that for some domains the sentiment words identified from the past domains can be wrong. For example, the word "toy" usually indicates a negative opinion in a review as people often say that "*this camera is a toy*" and "*this laptop is a toy.*" However, when we classify reviews about children's toys, the word "toy" does not indicate any sentiment. We thus need some in-domain training data from D in order to detect such words to overwrite the past knowledge about the words. In fact, this is to solve the problem of *applicability of knowledge* in Section 1.3. With the correction, a lifelong learner can do much better. We will discuss the technique in detail in Section 3.6. In this case, the knowledge base (KB) of LML stores the conditional probability $P(w|y)$ for each previous task.

This chapter reviews those representative techniques of lifelong supervised learning. Most of the techniques can perform well with a small number of training examples.

3.1 DEFINITION AND OVERVIEW

We first present the definition of *lifelong supervised learning* based on the general definition of LML in Chapter 1. We then give a brief overview of the existing work.

Definition 3.1 *Lifelong supervised learning* is a continuous learning process where the learner has performed a sequence of N supervised learning tasks, $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_N$, and retained the learned knowledge in a knowledge base (KB). When a new task $\mathcal{T}_{N+1}$ arrives, the learner leverages the past knowledge in the KB to help learn a new model f_{N+1} from $\mathcal{T}_{N+1}$'s training data D_{N+1} After learning $\mathcal{T}_{N+1}$, the KB is also updated with the learned knowledge from $\mathcal{T}_{N+1}$.

Lifelong supervised learning started with the paper by Thrun [1996b], which proposed several earlier LML methods in the context of memory-based learning and neutral networks. We will review them in Sections 3.2 and 3.3. The neural network approach was improved in [Silver and Mercer, 1996, 2002, Silver et al., 2015]. In these papers, each new task focuses on learning one new concept or class. The goal of LML is to leverage the past data to help build a binary classifier to identify instances of this new class. In [Fei et al., 2016], a special form of LML called *cumulative learning* was proposed. Similar to the above works, each new task is represented with a new class of data that needs to be learned. However, unlike the above works, the system only maintains a single multi-class classification model at any time. When a new class arrives, the model is updated in order to classify all the past classes and the new class. It thus learns cumulatively. Fei et al. [2016] also proposed a similarity

space-based learning method to detect new classes that have not been seen in training (see Section 3.4). Ruvolo and Eaton [2013b] proposed the ELLA algorithm to improve the multi-task learning method GO-MTL [Kumar et al., 2012] to make it a lifelong learning method. Chen et al. [2015] further proposed a technique in the context of Naïve Bayesian classification. A theoretical study was also done by Pentina and Lampert [2014] in the PAC-learning framework. It provided a PAC-Bayesian generalization bound that quantifies the relation between the expected loss on a new task to the average loss on existing tasks for lifelong learning. In particular, they modeled the prior knowledge as a random variable and obtained its optimal value by minimizing the expected loss on a new task. Such loss can be transferred from the average loss on existing tasks via the bound. They showed two realizations of the bound on the transfer of parameters [Evgeniou and Pontil, 2004] and the transfer of low-dimensional representations [Ruvolo and Eaton, 2013b]. In this following sections, we present the main existing techniques of lifelong supervised learning. In presenting each technique, we also map the components of the technique to the components of the general LML architecture in Section 1.3.

3.2 LIFELONG MEMORY-BASED LEARNING

In Thrun [1996b], a lifelong supervised learning technique was proposed for two memory-based learning methods: k-nearest neighbors and Shepard's method. We discuss them below.

3.2.1 TWO MEMORY-BASED LEARNING METHODS

K-Nearest Neighbors (KNN) [Altman, 1992] is a widely used machine learning algorithm. Given a testing instance x, the algorithm finds K examples in the training data $\langle x_i, y_i\rangle \in \mathcal{D}$ whose feature vectors x_i are nearest to x according to some distance metric such as the Euclidean distance. The predicted output is the mean value $\frac{1}{K}\sum y_i$ of these nearest neighbors.

Shepard's method [Shepard, 1968] is another commonly used memory-based learning method. Instead of only using K examples as in KNN, this method uses all the training examples in $\mathcal{D}$ and weights each example according to the inverse distance to the test instance x, as shown below:

$$s(x) = \left(\sum_{\langle x_i, y_i\rangle \in \mathcal{D}} \frac{y_i}{\|x - x_i\| + \epsilon}\right) \times \left(\sum_{\langle x_i, y_i\rangle \in \mathcal{D}} \frac{1}{\|x - x_i\| + \epsilon}\right)^{-1}, \tag{3.1}$$

where $\epsilon > 0$ is a small constant to avoid the denominator being zero. Neither KNN nor Shepard's method can use the previous task data with different distributions or distinct class labels to help its classification.

3.2.2 LEARNING A NEW REPRESENTATION FOR LIFELONG LEARNING

Thrun [1996b] proposed to learn a new representation to bridge the gap among tasks for the above two memory-based methods to achieve lifelong learning, which was shown to improve the predictive performance especially when the number of labeled examples is small.

The interest of the paper is concept learning. Its goal is to learn a function $f : I \rightarrow \{0, 1\}$ where $f(x) = 1$ means that $x \in I$ belongs to a target concept (e.g., *cat* or *dog*); otherwise x does not belong to the concept. For example, $f_{dog}(x) = 1$ means that x is an instance of the concept dog. Let the data from the previous N tasks be $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_N\}$. Each past task data $\mathcal{D}_i \in \mathcal{D}^p$ is associated with an unknown classification function f_i. $\mathcal{D}^p$ is called the *support set* in [Thrun, 1996b]. The goal is to learn the function f_{N+1} for the current new task data $\mathcal{D}_{N+1}$ with the help of the support set.

To bridge the difference among different tasks and to be able to exploit the shared information in the past data (the support set), the paper proposed to learn a new representation of the data, i.e., to learn a *space transformation function* $g : I \rightarrow I'$ to map the original input feature vectors in I to a new space I'. The new space I' then serves as the input space for KNN or the Shepard's method. The intuition is that positive examples of a concept (with $y = 1$) should have similar new representations while a positive example and a negative example of a concept ($y = 1$ and $y = 0$) should have very different representations. This idea can be formulated as an energy function E for g:

$$E = \sum_{\mathcal{D}_i \in \mathcal{D}^p} \sum_{\langle x, y=1 \rangle \in \mathcal{D}_i} \left(\sum_{\langle x', y'=1 \rangle \in \mathcal{D}_i} \|g(x) - g(x')\| - \sum_{\langle x', y'=0 \rangle \in \mathcal{D}_i} \|g(x) - g(x')\| \right) . \quad (3.2)$$

The optimal function g^* is achieved by minimizing the energy function E, which forces the distance between pairs of positive examples of the concept ($\langle x, y = 1 \rangle$ and $\langle x', y' = 1 \rangle$) to be small, and the distance between a positive example and a negative example of a concept ($\langle x, y = 1 \rangle$ and $\langle x', y' = 0 \rangle$) to be large. In the implementation of [Thrun, 1996b], g was realized with a neural network and trained with the support set using Back-Propagation.

Given the mapping function g^*, rather than performing memory-based learning in the original space $\langle x_i, y_i \rangle \in \mathcal{D}_{N+1}$, x_i is first transformed to the new space using g^* to $\langle g^*(x_i), y_i \rangle$ before applying KNN or the Shepard's method.

To map aspects of this technique to the components of the general LML architecture of Section 1.3, we can see that the part of the technique that computes the mapping function g^* (the *shared knowledge* from the past) and transforms the data from the original space to the new space is basically the **knowledge-based learner** of Section 1.3. The **knowledge base** stores only the data from previous tasks. The technique does not deal with the correctness or applicability of the shared knowledge g^* (Section 1.3).

Since this approach does not retain any knowledge learned in the past but only accumulates the past data, it is thus inefficient if the number of previous tasks is large

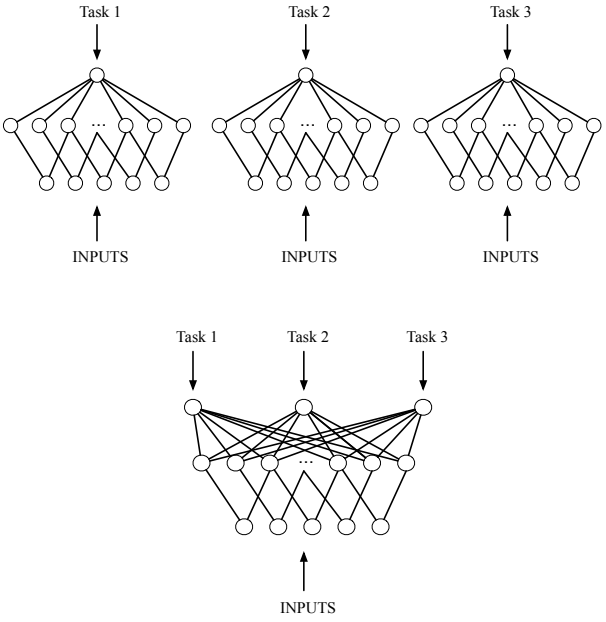


Figure 3.1: The top neural networks are trained independently for each task, and the bottom neural network is MTL net Caruana [1997].

because the whole optimization needs to be re-done using all the past data (the support set) whenever a new task arrives. In [Thrun, 1996b], an alternative method to the above energy-function-based approach was also proposed, which learns a distance function based on the support set. This distance function is then used in lifelong memory-based learning. This approach has similar weaknesses.

3.3 LIFELONG NEURAL NETWORKS

Here we introduce two early neural network approaches to lifelong supervised learning.

3.3.1 MTL NET

Although MTL net (Multi-task learning with neural network) Caruana [1997] is described as a lifelong learning method in [Thrun, 1996b], it is actually a batch multi-task learning

method. Based on our definition of lifelong learning, they are different learning paradigms. However, for historical reasons, we still give it a brief discussion here.

In MTL net, instead of building a neural network for each individual task, it constructs a universal neural network for all the tasks, see Figure 3.1. This universal neural network uses the same input layer for input from all tasks and uses one output unit for each task (or class in this case). There is also a shared hidden layer in MTL net that is trained in parallel using Back-Propagation [Rumelhart et al., 1985] on all the tasks to minimize the error on all the tasks. This shared layer allows features developed for one task to be used by other tasks. So some developed features can represent the common characteristics of the tasks. For a specific task, it will activate some hidden units that are related to it while making the weights of the other irrelevant hidden units small. Essentially, like a normal batch multi-task learning method, the system jointly optimizes the classification of all the past/previous and the current/new tasks. It is thus not regarded as a lifelong learning method based on the definition in this book (see Section 2.2.4). Several extensions of MTL net were made in [Silver and Mercer, 2002, Silver and Poirier, 2004, 2007], from generating and using virtual training examples to deal with the need for the training data of all previous tasks to adding contexts.

3.3.2 LIFELONG EBNN

This lifelong learning approach is in the context of EBNN (Explanation-Based Neural Network) [Thrun, 1996a], which again leverages the previous task data (or the support set) to improve learning. Same as Section 3.2.2, concept learning is the goal of this work, which learns a function $f : I \to \{0, 1\}$ to predict if an object represented by a feature vector $x \in I$ belongs to a concept ($y = 1$) or not ($y = 0$).

In this approach, the system first learns a *general distance function*, $d : I \times I \to [0, 1]$, considering all the past data (or the support set) and uses this distance function to share or transfer the knowledge of the past task data to the new task $\mathcal{T}_{N+1}$. Given two input vectors, say x and x', function d computes the probability of x and x' being members of the same concept (or class), regardless what the concept is. In [Thrun, 1996b], d is learned using an neural network trained with Back-Propagation. The training data for learning the distance function is generated as follows: For each past task data $\mathcal{D}_i \in \mathcal{D}^p$, each pair of examples of the concept generates an training example. For a pair, $\langle x, y = 1\rangle \in \mathcal{D}_i$ and $\langle x', y' = 1\rangle \in \mathcal{D}_i$, a positive training example is generated, $\langle (x, x'), 1\rangle$. For a pair $\langle x, y = 1\rangle \in \mathcal{D}_i$ and $\langle x', y' = 0\rangle \in \mathcal{D}_i$ or $\langle x, y = 0\rangle \in \mathcal{D}_i$ and $\langle x', y' = 1\rangle \in \mathcal{D}_i$, a negative training example is generated, $\langle (x, x'), 0\rangle$.

With the learned distance function in hand, EBNN works as follows: Unlike a traditional neural network, EBNN estimates the *slope* (tangent) of the target function at each data point x and adds it into the vector representation of the data point. In the new task $\mathcal{T}_{N+1}$, a training example is of the form, $\langle x, f_{N+1}(x), \nabla_x f_{N+1}(x)\rangle$, where $f_{N+1}(x)$ is

just the original class label of $x \in \mathcal{D}_{N+1}$ (the new task data). The system is trained using Tangent-Prop algorithm [Simard et al., 1992]. $\nabla_x f_{N+1}(x)$ is estimated using the gradient of the distance d obtained from the neural network, i.e., $\nabla_x f_{N+1}(x) \approx \frac{\partial d_{x'}(x)}{\partial x}$, where $\langle x', y' = 1 \rangle \in \mathcal{D}_{N+1}$ and $d_{x'}(x) = d(x, x')$. The rationale is that the distance between x and a positive training example x' is an estimate of the probability of x being a positive example, which approximates $f_{N+1}(x)$. As a result, the built EBNN fits both the current task data $\mathcal{D}_{N+1}$ and the support set through $\nabla_x f_{N+1}(x)$ and d.

Similar to lifelong KNN in Section 3.2, in this case, the part of the system that learns the distance function (the *shared knowledge*) and performs EBNN is the **knowledge-based learner** in Section 1.3. Again, the **knowledge base** stores only the past data. Similarly, this technique also does not deal with correctness or applicability of the shared knowledge d (see Section 1.3).

Like lifelong KNN, since lifelong EBNN does not retain any knowledge learned in the past but only accumulates the past data, it is also inefficient if the number of previous tasks is large because the training of the distance function d needs to be re-done using all the past data (the support set) whenever a new task arrives. Additionally, since every pair of data points in each past task dataset forms a training example for learning the distance function d, the training data for learning d can be huge.

3.4 CUMULATIVE LEARNING IN THE OPEN WORLD

Cumulative learning studied in this section was proposed in [Fei et al., 2016] for text classification. It is a special form of lifelong machine learning (LML). It is different from most other LML problems in that its tasks are of the second kind in Section 3.1. That is, each task introduces a new class and needs to build a classifier that classifies this class and all other classes in the past. It performs two functions: (1) detecting unseen classes in testing, and (2) incrementally adding new classes to the existing classification model without re-training the whole model from scratch. These two functions in fact enable the system to perform a limited form of self-motivated learning because by recognizing that something new is happening, the system has the opportunity to learn the new thing. Traditionally, self-motivated learning means that the learner has curiosity that motivates it to explore new territories and to learn new things. In the context of supervised learning, the key is for the system to recognize what it has not seen before. If the classification system cannot recognize anything new, there is no way for it to learn anything new except being told by a human user or an external system, which is not ideal for an intelligent system. Furthermore, it cannot function in a dynamic open environment, where new or novel objects appear all the time, e.g., in the self-driving car environment. Note that in this paper, the new data still needs to be labeled by the user before learning as it may consists of multiple unseen classes. This manual labeling is reasonable as this often happens in human learning, i.e.,

one may ask someone else what the data is about if one does not already know himself or herself.

Fei et al. [2016] used an example to motivate this type of learning. The 2016 presidential election in the USA was a hot topic on the social media and many social science researchers relied on the collected online user discussions to carry out their research. During the long campaign, every new proposal made by a candidate is followed by a huge amount of discussions on social media. A multi-class classifier is thus needed to track and organize the discussions. As the campaign goes on, the initially built classifier will inevitably encounter new topics (e.g. Donald Trump's plan for immigration reform or Hillary Clinton's new proposal for tax increase) that have not been covered in previous training. In this case, the classifier should first recognize these new topics when they occur rather than classifying them into some existing classes or topics. Second, after enough training examples of the new topics are collected, the existing classifier should incorporate the new classes or topics in a manner that does not require rebuilding the entire classification system from scratch.

Image recognition is another good example. Due to too many objects in the world, it is just impossible to build a classifier to recognize every one of them. The system has to learn cumulatively: recognizing each object that it has been trained on, detecting new objects that it has never seen before, and learning to recognize the new objects incrementally.

Cumulative learning is formally stated as follows: At a particular time point, the learner has built a multi-class classification model F_N based on all past N classes of data $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$ with their corresponding class labels $\mathcal{Y}^N = \{l_1, l_2, \ldots, l_N\}$. F_N is able to classify each test instance to either one of the known classes $l_i \in \mathcal{Y}^N$ or the unknown class C_0, which represents all new or unseen classes or topics in the test set. This is called *open world classification* (or simply *open classification*), which we will discuss below. When a new class of data $\mathcal{D}_{N+1}$ is added to the system with its class label l_{N+1}, the current model F_N is updated to produce F_{N+1} so that F_{N+1} can classify each test instance into one of the known classes $l_i \in \mathcal{Y}^{N+1} = \{l_1, l_2, \ldots, l_N, l_{N+1}\}$ or the unknown class C_0.

Cumulative learning is a form of LML because it conforms to the definition of LML in Chapter 1. Specifically, the new learning task $\mathcal{T}_{N+1}$ is to build a multi-class open classifier based on all the past and the current classes. The knowledge base contains the past model F_N and all the past training data.

Open (World) Classification or Learning: Classic supervised learning makes the *closed world assumption*, meaning that all the test classes have been seen in training. In cumulative learning, the test data may contain instances from classes that have not appeared in training. This is called *open world learning* (or simply *open learning* or *open classification*). Fei et al. [2016] proposed a technique to perform open classification based on a center-based similarity space learning method (called *CBS learning*), which we will discuss shortly.

3.4.1 TRAINING A CUMULATIVE LEARNING MODEL

This sub-section describes training in cumulative learning, which was inspired by human concept learning. Humans are exposed to new concepts all the time. One way we learn a new concept is perhaps by searching from the already known concepts, looking for the similar ones, and then trying to find the difference between these known concepts and the new one without using all the known concepts. For example, assume we have already learned the concepts like "movie," "furniture," and "soccer." Now we are presented with the concept of "basketball" and its set of documents. We find that "basketball" is similar to "soccer," but very different from "movie" and "furniture." Thus we just need to *accommodate* the new concept "basketball" into our old knowledge base by focusing on distinguishing the "basketball" and "soccer" concepts, and do not need to worry about the difference between "basketball" and "movie" or "furniture," because the concepts of "movie" and "furniture" can easily tell that documents from "basketball" do not belong to either of them.

Fei et al. [2016] adopted this idea and used the 1-vs-rest strategy of SVM for cumulative learning of multiple classes. Before the new class l_{N+1} arrives, the learning system has built a classification model F_N, which consists of a set of N 1-vs-rest binary classifiers $F_N = \{f_1, f_2, \ldots, f_N\}$ for the past N classes using their training sets $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$ and corresponding class labels $\mathcal{Y}^N = \{l_1, l_2, \ldots, l_N\}$. Each f_i is a binary classifier built using the CBS learning method (see Section 3.4.3) for identifying instances of class l_i. When a new dataset $\mathcal{D}_{N+1}$ of class l_{N+1} arrives, the system goes through the following two steps to update the classification model F_N to build a new model F_{N+1} in order to be able to classify test data or instances of all existing classes in $\mathcal{Y}^{N+1} = \{l_1, l_2, \ldots, l_N, l_{N+1}\}$ and recognize unseen class C_0 of documents.

1. Searching for a set of classes SC that are similar to the new class l_{N+1}.

2. Learning to separate the new class l_{N+1} and the previous classes in SC.

For step 1, the similarity between the new class l_{N+1} and the previous ones $\{l_1, l_2, ..., l_N\}$ is computed by running each of the 1-vs-rest past binary classifiers f_i in $F_N = \{f_1, f_2, ..., f_N\}$ to classify instances in $\mathcal{D}_{N+1}$. The classes of those past binary classifiers that accept (classify as positive) a certain number/percentage λ_{sim} of instances from $\mathcal{D}_{N+1}$ are regarded as similar classes and denoted by SC.

The step 2 of separating the new class l_{N+1} and classes in SC involves two sub-steps: (1) building a new binary classifier f_{N+1} for the new class l_{N+1} and (2) updating the existing classifiers for the classes in SC. It is intuitive to build f_{N+1} using $\mathcal{D}_{N+1}$ as the positive training data and the data of the classes in SC as the negative training data. The reason for updating classifiers in SC is that the joining of class l_{N+1} confuses those classifiers in SC. To re-build each classifier, the system needs to use the original negative data employed to build the existing classifier f_i and the new data $\mathcal{D}_{N+1}$ as the new negative

training data. The reason that the old negative training data is still used is because the new classifier still needs to separate class l_i from those old classes.

The detailed algorithm is given in Algorithm 1. Line 1 initializes SC to the empty set. Line 3 initializes the variable CT (count) to record the number of instances in $\mathcal{D}_{N+1}$ that will be classified as positive by classifier f_i. Lines 4-9 use f_i to classify each instance in $\mathcal{D}_{N+1}$ and record the number of instances that are classified (or accepted) as positive by f_i. Lines 10-12 check whether there are too many instances in $\mathcal{D}_{N+1}$ that have been classified as positive by f_i to render class l_i as similar to class l_{N+1}. λ_{sim} is a threshold controlling how many percents of instances in $\mathcal{D}_{N+1}$ should be classified to class l_i before considering l_i as similar/close to class l_{N+1}. Lines 14-17 build a new classifier f_{N+1} and update all the classifiers for classes in SC.

Algorithm 1 Cumulative Learning

Input: classification model $F_N = \{f_1, f_2, \ldots, f_N\}$, past datasets $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, new dataset $\mathcal{D}_{N+1}$, similarity threshold λ_{sim}.
Output: classification model $F_{N+1} = \{f_1, \ldots, f_N, f_{N+1}\}$

1: $SC = \varnothing$
2: **for** each classifier $f_i \in F_N$ **do**
3: $CT = 0$
4: **for** each test instance $x_j \in \mathcal{D}_{N+1}$ **do**
5: $class = f_i(x_j)$ // classify document x_j using f_i
6: **if** $class = l_i$ **then**
7: $CT = CT + 1$ // wrongly classified
8: **end if**
9: **end for**
10: **if** $CT > \lambda_{sim} \times |\mathcal{D}_{N+1}|$ **then**
11: SC = SC $\cup\{l_i\}$
12: **end if**
13: **end for**
14: Build f_{N+1} and add it to F_{N+1}
15: **for** each f_i of class $l_i \in SC$ **do**
16: Update f_i
17: **end for**
18: Return F_{N+1}

In summary, the learning process uses the set SC of similar classes to the new class l_{N+1} to control both the number of binary classifiers that need to be built/updated and also

the number of negative instances used in building the new classifier f_{N+1}. It thus greatly improves the efficiency compared to building a new multi-class classifier F_{N+1} from scratch.

Combining the above cumulative learning process and the underlying classifier *cbsSVM* discussed in Section 3.4.3, the new learner, called *CL-cbsSVM* (*CL* stands for *Cumulative Learning),* is able to tackle both challenges in cumulative learning.

3.4.2 TESTING A CUMULATIVE LEARNING MODEL

To test the new classification model $F_{N+1} = \{f_1, f_2, ..., f_N, f_{N+1}\}$, the standard technique of combining the set of 1-vs-rest binary classifiers to perform multi-class classification is followed with a rejection option for the unknown. As output scores from different SVM classifiers are not comparable, the SVM scores for each classifier are first converted to probabilities based on a variant of Platts algorithm [Platt and Others, 1999], which is supported in LIBSVM [Chang and Lin, 2011]. Let $P(y|x)$ be a probabilistic estimator, where $y \in Y^{N+1}$ $(= \{l_1, l_2, , l_N, l_{N+1}\})$ is a class label and x is the feature vector of a test instance. Let θ $(= 0.5)$ be the decision threshold, y^* be the final predicted class for x, and C_0 be the label for the unknown. Classification of the test instance x is done as follow:

$$y^* = \begin{cases} \text{argmax}_{y \in Y^{N+1}} P(y|x) & \text{if } P(y|x) \geq \theta \\ C_0 & \text{otherwise} \end{cases} . \tag{3.3}$$

The idea is that for the test instance x, each binary classifier $f_i \in F_{N+1}$ is used to produce a probability $P(l_i|x)$. If none of the probabilities is greater than θ $(= 0.5)$, the document represented by x is regarded as unseen/unknown and belonging to C_0; otherwise it is classified to the class with the highest probability.

3.4.3 OPEN WORLD LEARNING FOR UNSEEN CLASS DETECTION

This subsection describes CBS learning, which performs binary classification focusing on identifying positive class documents and has a superior ability to detect unseen classes or classifying them as not positive. It provides the base learning method for cumulative learning above [Fei et al., 2016]. It is based on the idea of reducing the *open space risk* while balancing the *empirical risk* in learning. Classic learners define and optimize over empirical risk, which is measured on the training data. For open learning, it is crucial to consider how to extend the classic model to capture the risk of the unknown by preventing over-generalization. To tackle this problem, Scheirer et al. [2013] introduced the concept of *open space risk*. Below, we first discuss the open space risk management strategy in Fei et al. [2016], and then apply an SVM-based CBS learning method as the solution towards managing the open space risk. The basic idea of CBS learning is to find a "ball" (decision boundary) to cover the positive class data area. Any document falling outside of the "ball" is considered not positive. Although CBS learning only performs binary classification, ap-

plying the 1-vs-rest method described in Section 3.4.2 gives a multi-class CBS classification model, which is called cbsSVM in [Fei et al., 2016].

Open Space Risk

Consider the risk formulation for open image recognition in [Scheirer et al., 2013], where apart from empirical risk, there is risk in labeling the open space (space away from positive training examples) as "positive" for any unknown class. Due to lack of information of a classification function on the open space, open space risk is approximated by a relative Lebesgue measure [Shackel, 2007]. Let S_o be a large ball of radius r_o that contains both the positively labeled open space O and all of the positive training examples; and let f be a measurable classification function, where $f_y(x) = 1$ means recognizing x as belonging to class y of interest and $f_y(x) = 0$ otherwise. In our case, y is simply any class of interest l_i.

In [Fei et al., 2016], O is defined as the positively labeled area that is sufficiently far from the center of the positive training examples. Let $B_{r_y}(cen_y)$ be a closed ball of radius r_y centered around the center cen_y of positive class y, which, ideally, should tightly covers all positive examples of class y only; S_o be a larger ball $B_{r_o}(cen_y)$ of radius r_o with the same center cen_y. Let classification function $f_y(x) = 1$ for $x \in B_{r_o}(cen_y)$, and $f_y(x) = 0$ otherwise. Also let q be the positive half space defined by a binary SVM decision hyperplane Ω obtained using positive and negative training examples. We also define the size of ball B_{r_o} to be bounded by Ω, $B_{r_o} \cap q = B_{r_o}$. Then the positive open space is defined as $O = S_o - Br_y(cen_y)$. S_o needs to be determined during learning for the positive class.

This open space formulation greatly reduces the open space risk compared to traditional SVM and 1-vs-Set Machine in [Scheirer et al., 2013]. For traditional SVM, classification function $f_y^{SVM}(x) = 1$ when $x \in q$, and its positive open space is approximately $q - B_{r_y}(cen_y)$, which is only bounded by the SVM decision hyperplane Ω. For 1-vs-Set Machine in [Scheirer et al., 2013], $f_y^{1-vs-set}(x) = 1$ when $x \in g$, where g is a slab area with thickness δ bounded by two parallel hyperplanes Ω and Ψ ($\Psi||\Omega$) in q. And its positive open space is approximately $g - g \cap B_{r_y}(cen_y)$. Given open space formulations of the traditional SVM and 1-vs-Set Machine, we can see that both methods label an unlimited area as the positively labeled space, while Fei et al. [2016] reduces it to a bounded area of a "ball."

Given the open space definition, the question is how to estimate S_o for the positive class. Fei et al. [2016] used the center-based similarity space learning (CBS learning), which transforms the original document space to a similarity space. The final classification is performed in the CBS space. Below, we introduce CBS learning and briefly discuss why it is suitable for the problem.

Center-based Similarity Space Learning

Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be the set of training examples, where x_k is the feature vector (e.g., with unigram features) representing a document and $y_k \in \{1, -1\}$ is its

class label. This feature vector is called a document space vector (or *ds-vector*). Traditional classification directly uses $\mathcal{D}$ to build a binary classifier. However, CBS learning transforms each ds-vector x_k (no change to its class label) to a center-based similarity space feature vector (CBS vector) *cbs-v*$_k$. Each feature in *cbs-v*$_k$ is a similarity between a center c_j of the positive class documents and x_k.

To make CBS learning more effective by generating more similarity features, multiple document space representations or feature vectors (e.g., one based on unigrams and one based on bigrams) can be used to represent each document, which results in multiple centers for the positive documents. There can also be multiple document similarity functions used to compute similarity values. The detailed learning technique is as follows.

For a document x_k, we have a set R_k of p ds-vectors, $R_k = \{d_1^k, d_2^k, \ldots, d_p^k\}$. Each ds-vector d_j^k denotes one document space representation of the document x_k, e.g., unigram representation or bigram representation. Then the centers of positive training documents can be computed, which are represented as a set of p centroids $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$. Each c_j corresponds to one document space representation in R_k. Rocchio method in information retrieval [Manning et al., 2008] is used to compute each center c_j (a vector), which uses the corresponding ds-vectors of all training positive and negative documents:

$$c_j = \frac{\alpha}{|\mathcal{D}_+|} \sum_{x_k \in \mathcal{D}_+} \frac{d_j^k}{\|d_j^k\|} - \frac{\beta}{|\mathcal{D} - \mathcal{D}_+|} \sum_{x_k \in \mathcal{D} - \mathcal{D}_+} \frac{d_j^k}{\|d_j^k\|} \ , \tag{3.4}$$

where $\mathcal{D}_+$ is the set of documents in the positive class and $|.|$ is the size function. α and β are parameters. It is reported that using the popular *tf-idf* (*term frequency and inverse document frequency*) representation, $\alpha = 16$ and $\beta = 4$ usually work well [Buckley et al., 1994]. The subtraction is used to reduce the influence of those terms that are not discriminative (i.e., terms appearing in both classes).

Based on R_k for a document x_k (in both training and testing) and the previously computed set $\mathcal{C}$ of centers using the training data, we can transform a document x_k from its document space representations R_k to one center-based similarity space vector *cbs-v*$_k$ by applying a similarity function Sim on each element d_j^k of R_k and its corresponding center c_j in $\mathcal{C}$:

$$\textit{cbs-v}_k = Sim(R_k, \mathcal{C}) \ . \tag{3.5}$$

Sim can contain a set of similarity measures. Each measure m is applied to p document representations d_j^k in R_k and their corresponding centers c_j in $\mathcal{C}$ to generate p similarity features (cbs-features) in *cbs-v*$_k$.

For ds-features, unigrams and bigrams with tf-idf weighting were used as two document representations. The five similarity measures in [Fei and Liu, 2015] were applied to measure the similarity of two vectors. Based on the CBS space representation, SVM is applied to produce a binary CBS classifier f_y.

Why Does CBS Learning Work?

We now briefly explain why CBS learning gives a good estimate to S_o. Due to using similarities as features, CBS learning generates a boundary to separate the positive and negative training data in the similarity space. Since similarity has no direction (or it covers all directions), the boundary in the similarity space is essentially a "ball" encompassing the positive class training data in the original document space. The "ball" is an estimate of S_o based on those similarity measures.

Mapping to Components in the LML Architecture in Section 1.3

We now map the components of CL-cbsSVM to the components of the general architecture of LML in Section 1.3. The **knowledge base** in Section 1.3 is the knowledge base in LML, which stores the previous model F_N and the training data from all previous tasks. Algorithm 1 is the **knowledge-based learner**. The issues of correctness and applicability of knowledge in Section 1.3 are not of concern here.

3.5 ELLA: AN EFFICIENT LIFELONG LEARNING ALGORITHM

This section focuses on the lifelong supervised learning system ELLA (Efficient Lifelong Learning Algorithm) proposed by Ruvolo and Eaton [2013a,b]. It maintains a sparsely shared basis (the past knowledge) for all task models, transfers knowledge from the basis to the new task, and refines the basis over time to maximize the performances across all tasks. Unlike cumulative learning, each task in ELLA is independent of other tasks. ELLA also follows the tradition of multi-task learning aimed at optimizing the performances of all tasks. Many other lifelong learning methods mainly optimize the performance of the new task, although they can help optimize any previous task if needed. In the presentation below, we try to use about the same notation as in the original paper for easy reference.

3.5.1 PROBLEM SETTING

As in a normal LML problem, ELLA receives a sequence of supervised learning tasks, $1, 2, \ldots, N$, in a lifelong manner. Each task t has its training data $\mathcal{D}^t = \{(\mathbf{x}_i^t, y_i^t) : i = 1, \ldots, n_t\}$, where n_t is the number of training instances in $\mathcal{D}^t$, and is defined by a hidden (or latent) true mapping $\hat{f}^t(\mathbf{x})$ from an instance space $\mathcal{X}^t \subseteq \mathbb{R}^d$ to a set of labels $\mathcal{Y}^t$ (or $\mathcal{Y}^t = \mathbb{R}$ for regression). Let d be the feature dimension.

ELLA extends the batch multi-task learning model GO-MTL [Kumar et al., 2012] (also in Section 2.2.2) to make it more efficient and become an incremental or online multi-task learning system, which is regarded as an LML system. Like Go-MTL, ELLA takes a parametric approach to model building in which the model or the prediction function

$f^t(\mathbf{x}) = f^t(\mathbf{x}; \boldsymbol{\theta}^t)$ for each task t is governed by a task-specific parameter vector $\boldsymbol{\theta}^t \in \mathbb{R}^d$. The goal of ELLA is to construct task models $f^1, \ldots, f^N$ so that:

1. for each task, $f^t \approx \hat{f}^t$.
2. a new model f^t can be added quickly when the training data for a new task t arrives.
3. each past model f^t can be updated efficiently after the addition of the new task.

ELLA assumes that the total number of tasks, the distribution of tasks and their order are all unknown [Ruvolo and Eaton, 2013b]. It also assumes that there can be a large number of tasks, while each task can have a large number of data points. Thus, an LML algorithm that is both effective and efficient is needed.

3.5.2 OBJECTIVE FUNCTION

In the same way as the GO-MTL model [Kumar et al., 2012] (see Section 2.2.2), ELLA maintains k sparsely shared basis model components for all task models. Let $\mathbf{L} \subseteq \mathbb{R}^{d \times k}$ be the k basis model components. Each task model's parameter vector $\boldsymbol{\theta}^t$ is assumed to be a linear combination of the weight vector $\mathbf{s}^t \in \mathbb{R}^k$ and the basis model components $\mathbf{L}$. We thus obtain the equation below (same as Equation 2.7):

$$\underset{d \times N}{\boldsymbol{\Theta}} = \underset{d \times k}{\mathbf{L}} \times \underset{k \times N}{\mathbf{S}} \quad , \tag{3.6}$$

where $\boldsymbol{\Theta} = [\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \ldots, \boldsymbol{\theta}^N]$ and $\mathbf{S} = [\boldsymbol{s}^1, \boldsymbol{s}^2, \ldots, \boldsymbol{s}^N]$. For each task t, $\boldsymbol{\theta}^t = \mathbf{L}\mathbf{s}^t$. The initial objective function of ELLA is the same as Equation 2.8 in GO-MTL except that it optimizes the average (rather than the sum) loss on the training data across all tasks, which is essential for the convergence guarantees:

$$\frac{1}{N} \sum_{t=1}^{N} \min_{\mathbf{s}^t} \left\{ \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}\left(f(\mathbf{x}_i^t; \mathbf{L}\mathbf{s}^t), y_i^t\right) + \mu \|\mathbf{s}^t\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2 \quad , \tag{3.7}$$

where $f(\mathbf{x}_i^t; \mathbf{L}\mathbf{s}^t) = \boldsymbol{\theta}^t \mathbf{x}_i^t = (\mathbf{L}\mathbf{s}^t)^{\mathrm{T}} \mathbf{x}_i^t$. Since the objective function is not jointly convex in $\mathbf{L}$ and the $\mathbf{s}^t$'s, to optimize it, one can use a common approach to computing a local optimum, i.e., alternately optimizing $\mathbf{s}^t$ while holding $\mathbf{L}$ fixed, and optimizing $\mathbf{L}$ while holding $\mathbf{s}^t$ fixed. However, as pointed out in Ruvolo and Eaton [2013b], there are two major inefficiency issues in the above objective function (which also exist in GO-MTL).

1. There is an explicit dependence on *all* of the previous training data (through the inner summation). That is, to compute the objective function, one needs to iterate all training instances in all tasks in order to compute their loss function values. If the number of tasks is large or the number of training instances in each task is large, this iteration can be very inefficient.

2. When evaluating a single candidate $\mathbf{L}$ in Equation 3.7, an optimization problem must be solved to recompute the value of each $\mathbf{s}^t$. This means each $\mathbf{s}^t$ will have to be updated when $\mathbf{L}$ is updated. This becomes increasingly expensive when there are more and more tasks.

Ruvolo and Eaton [2013b] proposed some approximation techniques to deal with the above two inefficiency issues, which we detail in the next subsection. The basic idea is to approximate the fit of a new task model using the single-task solution as a point estimate in the basis of model components learned over the past tasks, and then updates the basis to incorporate the new knowledge from the new task.

3.5.3 DEALING WITH THE FIRST INEFFICIENCY

To address the first issue, Ruvolo and Eaton [2013b] used the second-order Taylor expansion for approximation. Before giving the technical details, let us briefly review some mathematical foundations.

Taylor Expansion

In the single-variable case, i.e., when $g(x)$ is a one variable function, the second-order Taylor expansion near a constant value a is:

$$g(x) \approx g(a) + g'(a)(x-a) + \frac{1}{2}g''(a)(x-a)^2 \ , \tag{3.8}$$

where $g'()$ and $g''()$ are the derivative and the second-order derivative of function g.

In the multiple-variable case, i.e., when $g(\mathbf{x})$ is a multivariate function (assuming $\mathbf{x}$ has n values), the second-order Taylor expansion near vector $\mathbf{a}$ of a constant size n is:

$$g(\mathbf{x}) \approx g(\mathbf{a}) + \nabla g(\mathbf{a})(\mathbf{x}-\mathbf{a}) + \frac{1}{2}\left\|(\mathbf{x}-\mathbf{a})\right\|^2_{\boldsymbol{H}(\mathbf{a})} \ , \tag{3.9}$$

where $\|\mathbf{v}\|^2_{\mathbf{a}} = \mathbf{v}^{\mathrm{T}}\mathbf{a}\mathbf{v}$ and $\boldsymbol{H}(\mathbf{a})$ is called *Hessian Matrix* of function g.

Optimality Conditions in Unconstrained Optimization

Consider the problem of minimizing function $g : \mathbb{R}^n \to \mathbb{R}$, where g is twice continuously differentiable on $\mathbb{R}^n$:

$$\min_{\mathbf{x}\in\mathbb{R}^n} g(\mathbf{x}) \ . \tag{3.10}$$

Theorem 3.2 First-Order Necessary Conditions for Optimality. *Let function* $g : \mathbb{R}^n \to \mathbb{R}$ *be differentiable at a point* $\hat{\mathbf{x}} \in \mathbb{R}^n$. *If* $\hat{\mathbf{x}}$ *is a local minimizer, then* $\nabla g(\hat{\mathbf{x}}) = 0$.

Proof. From the definition of the first-order Taylor expansion, we have:

$$f(\mathbf{x}) = f(\hat{\mathbf{x}}) + \nabla f(\hat{\mathbf{x}})^{\mathrm{T}}(\mathbf{x} - \hat{\mathbf{x}}) + o(\|\mathbf{x} - \hat{\mathbf{x}}\|) \ , \tag{3.11}$$

that is

$$f(\mathbf{x}) - f(\hat{\mathbf{x}}) = \nabla f(\hat{\mathbf{x}})^{\mathrm{T}}(\mathbf{x} - \hat{\mathbf{x}}) + o(\|\mathbf{x} - \hat{\mathbf{x}}\|) \ , \tag{3.12}$$

where $\lim_{\mathbf{x}\to\hat{\mathbf{x}}} \frac{o(\|\mathbf{x}-\hat{\mathbf{x}}\|)}{\|\mathbf{x}-\hat{\mathbf{x}}\|} = 0$. Let $\mathbf{x} := \hat{\mathbf{x}} - \alpha\nabla f(\hat{\mathbf{x}})$, where α is a positive constant. Plugging it into Equation 3.12, then:

$$0 \leq \frac{f(\hat{\mathbf{x}} - \alpha\nabla f(\hat{\mathbf{x}})) - f(\hat{\mathbf{x}})}{\alpha} = -\|\nabla f(\hat{\mathbf{x}})\|^2 + \frac{o(\alpha\|\nabla f(\hat{\mathbf{x}})\|)}{\alpha} \ . \tag{3.13}$$

Taking the limit as $\alpha \downarrow 0$, we obtain:

$$0 \leq -\|\nabla f(\hat{\mathbf{x}})\|^2 \leq 0 \ . \tag{3.14}$$

Hence $\nabla f(\hat{\mathbf{x}}) = 0$. □

Removing Dependency

We now come back to ELLA. To remove the explicit dependence on *all* task training data, the second-order Taylor expansion is used to approximate the objective function in Equation 3.7. Let's first define a function $g(\boldsymbol{\theta}^t)$ as below:

$$g(\boldsymbol{\theta}^t) = \frac{1}{n_t}\sum_{i=1}^{n_t} \mathcal{L}\left(f(\boldsymbol{x}_i^t; \boldsymbol{\theta}^t), y_i^t\right) \ , \tag{3.15}$$

where $\boldsymbol{\theta}^t = \mathbf{L}\mathbf{s}^t$. Then the objective function in Equation 3.7 becomes:

$$\frac{1}{N}\sum_{t=1}^{N} \min_{\mathbf{s}^t} \left\{g(\boldsymbol{\theta}^t) + \mu\|\mathbf{s}^t\|_1\right\} + \lambda\|\mathbf{L}\|_F^2 \ . \tag{3.16}$$

Let's assume that the minimum solution of the function g is $\hat{\boldsymbol{\theta}}^t$, i.e., $\hat{\boldsymbol{\theta}}^t = \operatorname{argmin}_{\boldsymbol{\theta}^t} \frac{1}{n_t}\sum_{i=1}^{n_t} \mathcal{L}\left(f(\boldsymbol{x}_i^t; \boldsymbol{\theta}^t), y_i^t\right)$ (which is an optimal predictor learned on only the training data for task t). Then, the second-order Taylor expansion near $\hat{\boldsymbol{\theta}}^t$ is as follows:

$$g(\boldsymbol{\theta}^t) \approx g(\hat{\boldsymbol{\theta}}^t) + \nabla g(\hat{\boldsymbol{\theta}}^t)(\boldsymbol{\theta}^t - \hat{\boldsymbol{\theta}}^t) + \frac{1}{2}\|\boldsymbol{\theta}^t - \hat{\boldsymbol{\theta}}^t\|^2_{\boldsymbol{H}^t} \ , \tag{3.17}$$

where $\boldsymbol{H}^t = \boldsymbol{H}(\hat{\boldsymbol{\theta}}^t)$ is the Hessian Matrix of function g.

Considering that function g is used in the outer minimization in Equation 3.16, the first constant term in Equation 3.17 can be suppressed. According to the first-order necessary conditions (Theorem 3.2), $\nabla g(\hat{\boldsymbol{\theta}}^t) = 0$ since $\hat{\boldsymbol{\theta}}^t$ is the local minimum solution of function g, and thus the second term in Equation 3.17 can also be removed. Hence the new objective function after plugging in Equation 3.16 is:

$$\frac{1}{N}\sum_{t=1}^{N}\min_{\mathbf{s}^t}\left\{\|\boldsymbol{\theta}^t-\hat{\boldsymbol{\theta}}^t\|^2_{\boldsymbol{H}^t}+\mu\|\mathbf{s}^t\|_1\right\}+\lambda\|\mathbf{L}\|^2_F \quad . \tag{3.18}$$

As $\boldsymbol{\theta}^t = \mathbf{L}\mathbf{s}^t$, Equation 3.18 can be rewritten as:

$$\frac{1}{N}\sum_{t=1}^{N}\min_{\mathbf{s}^t}\left\{\|\hat{\boldsymbol{\theta}}^t-\mathbf{L}\mathbf{s}^t\|^2_{\boldsymbol{H}^t}+\mu\|\mathbf{s}^t\|_1\right\}+\lambda\|\mathbf{L}\|^2_F \quad . \tag{3.19}$$

$$\boldsymbol{H}^t=\frac{1}{2}\nabla^2_{\boldsymbol{\theta}^t,\boldsymbol{\theta}^t}\frac{1}{n_t}\sum_{i=1}^{n_t}\mathcal{L}\left(f(\boldsymbol{x}_i^t;\boldsymbol{\theta}^t),y_i^t\right)\Big|_{\boldsymbol{\theta}^t=\hat{\boldsymbol{\theta}}^t} \quad , \text{ and}$$

$$\hat{\boldsymbol{\theta}}^t=\underset{\boldsymbol{\theta}^t}{\operatorname{argmin}}\frac{1}{n_t}\sum_{i=1}^{n_t}\mathcal{L}\left(f(\boldsymbol{x}_i^t;\boldsymbol{\theta}^t),y_i^t\right) \quad .$$

Note that $\hat{\boldsymbol{\theta}}^t$ and $\boldsymbol{H}^t$ will remain the same if the training data for task t does not change. Thus, the new objective function in Equation 3.19 removes the dependence of the optimization on the training data of all previous tasks.

3.5.4 DEALING WITH THE SECOND INEFFICIENCY

The second efficiency issue is that when computing a single candidate $\mathbf{L}$, an optimization problem must be solved to recompute the value of each $\mathbf{s}^t$. To solve this problem, Ruvolo and Eaton [2013b] adopted this strategy: when the training data for task t is last encountered, only $\mathbf{s}^t$ is updated while $\mathbf{s}^{t'}$ for other tasks t' remain the same. That is, $\mathbf{s}^t$ is computed when the training data for task t is last encountered, and it is not updated later when training on other tasks. Although this seems to prevent the influence of earlier tasks from later tasks, they will benefit from the subsequent adjustment of the basis latent model components $\mathbf{L}$. Using the previously computed values of $\mathbf{s}^t$, the following optimization process is performed:

$$\mathbf{s}^t \leftarrow \underset{\mathbf{s}^t}{\operatorname{argmin}}\|\hat{\boldsymbol{\theta}}^t-\mathbf{L}_m\mathbf{s}^t\|^2_{\boldsymbol{H}^t}+\mu\|\mathbf{s}^t\|_1 \text{ , with fixed } \mathbf{L}_m, \text{ and}$$

$$\mathbf{L}_{m+1} \leftarrow \underset{\mathbf{L}}{\operatorname{argmin}}\frac{1}{N}\sum_{t=1}^{N}\left(\|\hat{\boldsymbol{\theta}}^t-\mathbf{L}\mathbf{s}^t\|^2_{\boldsymbol{H}^t}+\mu\|\mathbf{s}^t\|_1\right)+\lambda\|\mathbf{L}\|^2_F, \text{ with fixed } \mathbf{s}^t \quad .$$

where notation $\mathbf{L}_m$ refers to the value of the latent components at the mth iteration and t is assumed to be the particular task for which the training data just arrives. Note that if t is an existing task, the new training data is merged into the old training data of t.

For the specific steps in performing the updates in the preceding equations, please refer to the original paper. They depend on the type of the model and the loss function used. The paper presented two cases, linear regression and logistic regression.

Mapping to Components in the LML Architecture in Section 1.3

We now map the components of ELLA to the components of the general architecture of LML in Section 1.3. In ELLA, the **knowledge base** is the data structure that stores $\mathbf{L}$, the shared basis model components, the weight vector $\mathbf{s}^t$ for each task, and the training data for each task. **Knowledge-based learner** is the whole ELLA algorithm. The issues of correctness and applicability of previous knowledge are not explicitly handled, but are considered to some extent in the optimization process for the new task t.

3.5.5 ACTIVE TASK SELECTION

Lifelong learning in the above problem setting (Section 3.5.1) is a passive process, i.e., the system has no control over the order in which the learning tasks are presented. Ruvolo and Eaton [2013a] considered ELLA in an active task selection setting. Assuming that there is a pool of candidate tasks, rather than choosing a task randomly as in ELLA, Ruvolo and Eaton [2013a] chose tasks in a certain order with the purpose of maximizing future learning performance using as few tasks as possible. The problem is practical since each learning task may need a significant amount of time of manual labeling or each learning task may take a long time for the system to run. In such cases, learning in a task-efficient manner by choosing some tasks in certain order is more scalable to real-life LML problems.

Active Task Selection Setting

The active task selection setting in LML is defined as follows: instead of modeling training data of task t as in regular LML, the system has a pool of candidate unlearned tasks $\boldsymbol{\mathcal{T}}_{pool}$ to choose from. For each candidate task $t \in \boldsymbol{\mathcal{T}}_{pool}$, only a subset of training instances is labeled, which are denoted by $\mathcal{D}_c^t = (\mathbf{X}_c^t, \mathbf{y}_c^t)$. Based on these small subsets, one of the tasks, $t_{next} \in \boldsymbol{\mathcal{T}}_{pool}$ is chosen to learn next. After that, all the training data of t_{next} will be revealed, which is denoted by $\mathcal{D}^{(t_{next})} = (\mathbf{X}^{(t_{next})}, \mathbf{y}^{(t_{next})})$. Note that for each task t, $\mathcal{D}_c^t \subseteq \mathcal{D}^t$. The size of the candidate pool can be a fixed value or increase/decrease dynamically during learning.

Diversity Method

Here we introduce the *diversity* method for active task selection proposed in [Ruvolo and Eaton, 2013a] which was shown to perform the best compared to the other methods used in

the paper. In the context of ELLA, in order to maximize performance on future tasks, the model should have a flexible and robust set of latent components, i.e., $\mathbf{L}$. In other words, $\mathbf{L}$ should be adaptable to a wide variety of tasks. If $\mathbf{L}$ does not fit well for a new task t, it means that the information in t has not been represented well in the current $\mathbf{L}$. Thus, in order to solve the widest range of tasks, the next task should be the one that the current basis $\mathbf{L}$ performs the worst, i.e., the loss on the subset of the labeled data is maximal. This heuristic is described as follows:

$$t_{next} = \underset{t \in \boldsymbol{\mathcal{T}}_{pool}}{\operatorname{argmax}} \min_{\mathbf{s}^t} \|\hat{\boldsymbol{\theta}}^t - \mathbf{L}\mathbf{s}^t\|^2_{\boldsymbol{H}^t} + \mu\|\mathbf{s}^t\|_1 \quad , \tag{3.20}$$

where $\hat{\boldsymbol{\theta}}^t$ and $\boldsymbol{H}^t$ are obtained from the subset of the labeled data $\mathcal{D}^t_c$. Since Equation 3.20 tends to select tasks that are encoded poorly with the current basis $\mathbf{L}$, the selected tasks are likely to be very different from existing tasks, and it thus encourages diverse tasks.

Rather than simply choosing the task with the maximal loss value, another way (called *Diversity++*) is to estimate the probability of selecting task t as the square value of the minimal loss value for t, as below:

$$p(t_{next} = t) \propto \left(\min_{\mathbf{s}^t} \|\hat{\boldsymbol{\theta}}^t - \mathbf{L}\mathbf{s}^t\|^2_{\boldsymbol{H}^t} + \mu\|\mathbf{s}^t\|_1 \right)^2 \quad . \tag{3.21}$$

Then each time, a task is sampled based on the probability $p(t_{next})$. This is thus a stochastic variant of the diversity method above.

3.6 LSC: LIFELONG SENTIMENT CLASSIFICATION

This section introduces the *Lifelong Sentiment Classification* (LSC) system given in [Chen et al., 2015] in the context of Naïve Bayesian (NB) learning. It aims to classify whether a product review expresses a positive or negative opinion. Below we first briefly introduce NB classification and then review its lifelong extension for sentiment classification. Again for easy reference, we follow the notation in the original paper.

3.6.1 NAÏVE BAYESIAN TEXT CLASSIFICATION

Given a set of training documents $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$, a vocabulary of V (the set of distinct words/terms in $\mathcal{D}$) and a set of classes $C = \{c_1, c_2, \ldots, c_{|C|}\}$ associated with $\mathcal{D}$, Naïve Bayesian classification trains a classifier by computing the conditional probability of each word $w \in V$ given each class c_j, i.e., $P(w|c_j)$ and the prior probability of each class, $P(c_j)$ [McCallum and Nigam, 1998].

$P(w|c_j)$ is estimated based on the empirical word counts as follows:

$$P\left(w|c_j\right) = \frac{\lambda + N_{c_j,w}}{\lambda\left|V\right| + \sum_{v=1}^{|V|} N_{c_j,v}} \quad , \tag{3.22}$$

where N_{c_j} is the number of times that word w occurs in the documents of class c_j. λ $(0 \leq \lambda \leq 1)$ is used for smoothing. When $\lambda = 1$, it is known as *Laplace smoothing*. The prior probability of each class, $P(c_j)$, is estimated as follows:

$$P(c_j) = \frac{\sum_{i=1}^{|\mathcal{D}|} P(c_j|d_i)}{|\mathcal{D}|} \quad , \tag{3.23}$$

where $P(c_j|d_i) = 1$ if c_j is the label of the training document d_i and 0 otherwise.

For testing, given a test document d, NB computes the posterior probability $P(c_j|d)$ for each class c_j and picks the class with the highest $P(c_j|d)$ as the classification result:

$$P(c_j|d) = \frac{P(c_j)P(d|c_j)}{P(d)} \tag{3.24}$$

$$= \frac{P(c_j)\prod_{w \in d} P(w|c_j)^{n_{w,d}}}{\sum_{r=1}^{|C|} P(c_r)\prod_{w \in d} P(w|c_r)^{n_{w,d}}} \quad , \tag{3.25}$$

where $n_{w,d}$ is the number of times that word w appears in d.

NB is a natural fit for lifelong learning because past knowledge can serve as priors for the probabilities of the new task very easily. LSC exploits this idea. Let us answer two specific questions in the context of sentiment classification. The first question is why the past learning can contribute to the new/current task classification given that the current task already has labeled training data. The answer is that the training data may not be fully representative of the test data due to *sample selection bias* [Heckman, 1979, Shimodaira, 2000, Zadrozny, 2004] and/or small training data size, which is the case in [Chen et al., 2015]. For example, in a sentiment classification application, the test data may contain some sentiment words that are absent in the current training data, but they have appeared in review data in some previous tasks. So the past knowledge can provide the prior sentiment polarity information for the current new task. Note that for sentiment classification, sentiment words such as *good*, *nice*, *terrible*, and *poor* are instrumental. Note also that each task in [Chen et al., 2015] is actually from a different domain (or products). We thus use *task* and *domain* interchangeably from now on.

The second question is why the past knowledge can help even if the past domains are very diverse and not very similar to the current domain. The main reason is that in sentiment classification, sentiment words and expressions are largely domain independent. That is, their polarities (positive or negative) are often shared across domains. Hence having worked a large number of previous/past domains, the system has learned a lot of positive and negative sentiment words. It is important to note that only one or two past domains are not sufficient because of the low coverage of sentiment words in the limited domains.

3.6.2 BASIC IDEAS OF LSC

This subsection introduces the basic ideas of the LSC technique. We start by discussing what is stored in the knowledge base of LSC.

Knowledge Base

For each word $w \in V^p$ (where V^p is the vocabulary of all previous tasks), the knowledge base KB stores two types of information: *document-level knowledge* and *domain-level knowledge*.

1. *Document-level knowledge* $N^{KB}_{+,w}$ (and $N^{KB}_{-,w}$): number of occurrences of w in the documents of the positive (and negative) class in the previous tasks.

2. *Domain-level knowledge* $M^{KB}_{+,w}$ (and $M^{KB}_{-,w}$): number of domains in which $P(w|+) > P(w|-)$ (and $P(w|+) < P(w|-)$). Here, in each previous task, $P(w|+)$ and $P(w|-)$ are calculated using Equation 3.22. Here $+$ and $-$ stands for positive and negative opinion classes respectively.

The domain-level knowledge is complementary to the document-level knowledge as w may be extremely frequent in a domain but rare in other domains which leads to the superfluous effect of that domain on w at the document level.

A Naive Approach to Using Knowledge

From Section 3.6.1, we can see that the key parameters that affect Naïve Bayesian classification results are $P(w|c_j)$ which are computed using the empirical counts $N_{c_j,w}$ and the total number of words in the class of documents. In binary classification, $P(w|c_j)$ are computed using $N_{+,w}$ and $N_{-,w}$. This suggests that we can revise these counts appropriately to improve classification. Given the new task data $\mathcal{D}^t$, we denote the empirical word counts $N^t_{+,w}$ (and $N^t_{-,w}$) as the number of times that word w occurs in the positive (and negative) documents in $\mathcal{D}^t$. Here, we explicitly use superscript t to distinguish it from the previous tasks. The task becomes how to effectively use the knowledge in the knowledge base KB to update word counts to build a superior Naïve Bayesian classifier.

Given the knowledge base KB from the past learning tasks, one naive way to build a classifier is to sum up the counts in KB (served as priors) with the empirical counts $N^t_{+,w}$ and $N^t_{-,w}$ of $\mathcal{D}^t$, i.e., $X_{+,w} = N^t_{+,w} + N^{KB}_{+,w}$ and $X_{-,w} = N^t_{-,w} + N^{KB}_{-,w}$. Here, we call $X_{+,w}$ and $X_{-,w}$ *virtual counts* as they will be updated using optimization discussed in the next sub-section. In building the classifier, $N_{+,w}$ and $N_{-,w}$ (i.e., $N_{c_j,w}$) in Equation 3.22 are replaced by $X_{+,w}$ and $X_{-,w}$ respectively. This naive method turns out to be quite good in many cases, but it has two weaknesses:

1. The past domains usually contain much more data than the current domain which means $N^{KB}_{+,w}$ (and $N^{KB}_{-,w}$) may be much larger than $N^t_{+,w}$ (and $N^t_{-,w}$). As a result, the merged results may be dominated by the counts in the KB from the past domains.

2. It does not consider the domain dependent word polarity. A word may be positive in the current domain but negative in past domains. For example, past domains may indicate that the word "toy" is negative because there are a lot of past sentences like "this product is a toy." However, in the toy domain, the word expresses no sentiment.

The LSC system solves these two problems using an optimization method.

3.6.3 LSC TECHNIQUE

LSC uses *stochastic gradient descent* (SGD) to minimize the training error by adjusting $X_{+,w}$ and $X_{-,w}$ (virtual counts), which are the numbers of times that a word w appears in the positive and negative classes respectively.

For correct classification, ideally, we should have the posterior probability $P(+|d_i) = 1$ for each positive class $(+)$ document d_i, and $P(-|d_i) = 1$ for each negative class $(-)$ document d_i. In stochastic gradient descent, we optimize the classification of each $d_i \in \mathcal{D}^t$. Chen et al. [2015] used the following objective function for each positive document d_i (a similar objective function can also be formulated for each negative document):

$$F_{+,i} = P\left(+|d_i\right) - P(-|d_i) \ . \tag{3.26}$$

We omit the derivation steps and just give the final equations below. To simplify the equations, we define $g(\boldsymbol{X})$, a function of $\boldsymbol{X}$ where $\boldsymbol{X}$ is a vector consisting of $X_{+,w}$ and $X_{-,w}$ of each word w:

$$g\left(\boldsymbol{X}\right) = \beta^{|d_i|} = \left(\frac{\lambda\left|V\right| + \sum_{v=1}^{|V|} X_{+,v}}{\lambda\left|V\right| + \sum_{v=1}^{|V|} X_{-,v}}\right)^{|d_i|} \ , \tag{3.27}$$

$$\frac{\partial F_{+,i}}{\partial X_{+,u}} = \frac{\frac{n_{u,d_i}}{\lambda + X_{+,u}} + \frac{P(-)}{P(+)} \prod_{w \in d_i} \left(\frac{\lambda + X_{-,w}}{\lambda + X_{+,w}}\right)^{n_{w,d_i}} \times \frac{\partial g}{\partial X_{+,u}}}{1 + \frac{P(-)}{P(+)} \prod_{w \in d_i} \left(\frac{\lambda + X_{-,w}}{\lambda + X_{+,w}}\right)^{n_{w,d_i}} \times g(\boldsymbol{X})} - \frac{n_{u,d_i}}{\lambda + X_{+,u}} \ , \tag{3.28}$$

$$\frac{\partial F_{+,i}}{\partial X_{-,u}} = \frac{\frac{n_{u,d_i}}{\lambda + X_{-,u}} \times g(\boldsymbol{X}) + \frac{\partial g}{\partial X_{-,u}}}{\frac{P(+)}{P(-)} \prod_{w \in d_i} \left(\frac{\lambda + X_{+,w}}{\lambda + X_{-,w}}\right)^{n_{w,d_i}} + g(\boldsymbol{X})} \ . \tag{3.29}$$

In stochastic gradient descent, we update the variables $X_{+,u}$ and $X_{-,u}$ for the positive document d_i iteratively using:

$$X_{+,u}^{l} = X_{+,u}^{l-1} - \gamma \frac{\partial F_{+,i}}{\partial X_{+,u}} \ , \text{ and}$$
$$X_{-,u}^{l} = X_{-,u}^{l-1} - \gamma \frac{\partial F_{+,i}}{\partial X_{-,u}} \ ,$$

where u represents each word in d_i. γ is the learning rate and l represents each iteration. Similar update rules can be derived for each negative document d_i. $X^0_{+,u} = N^t_{+,u} + N^{KB}_{+,u}$ and $X^0_{-,u} = N^t_{-,u} + N^{KB}_{-,u}$ serve as the starting points. The iterative updating process stops when the counts converge.

Exploiting Knowledge via Penalty Terms

The above optimization can update the virtual counts for better classification in the current domain. However, it does not deal with the issue of domain dependent sentiment words, i.e., some words may change their polarities across different domains. Nor does it use the domain-level knowledge in the knowledge base KB (Section 3.6.2). We thus propose to add penalty terms into the optimization to accomplish these.

The idea is that if a word w can distinguish classes very well in the current domain training data, we should rely more on the current domain training data. So we define a set V_T of distinguishing words in the current domain. A word w belongs to V_T if $P(w|+)$ is much larger or much smaller than $P(w|-)$ in the current domain, i.e., $\frac{P(w|+)}{P(w|-)} \geq \sigma$ or $\frac{P(w|-)}{P(w|+)} \geq \sigma$, where σ is a parameter. Such words are already effective in classification for the current domain, so the virtual counts in optimization should follow the empirical counts ($N^t_{+,w}$ and $N^t_{-,w}$) in the current task/domain, which are reflected in the L2 regularization penalty term below (α is the regularization coefficient):

$$\frac{1}{2}\alpha \sum_{w \in V_T} \left(\left(X_{+,w} - N^t_{+,w}\right)^2 + \left(X_{-,w} - N^t_{-,w}\right)^2 \right) \quad . \tag{3.30}$$

To leverage domain-level knowledge (the second type of knowledge in the KB in Section 3.6.2, we want to use only those reliable parts of the knowledge. The rationale here is that if a word only appears in one or two past domains, the knowledge associated with it is probably not reliable or it is highly specific to those domains. Based on this idea, domain frequency is used to define the reliability of the domain-level knowledge. For w, if $M^{KB}_{+,w} \geq \tau$ or $M^{KB}_{-,w} \geq \tau$ (τ is a parameter), it is regarded as appearing in a reasonable number of domains, making its knowledge reliable. The set of such words is denoted by V_S. Then the second penalty term is:

$$\frac{1}{2}\alpha \sum_{w \in V_S} \left(X_{+,w} - R_w \times X^0_{+,w}\right)^2 + \frac{1}{2}\alpha \sum_{w \in V_S} \left(X_{-,w} - (1 - R_w) \times X^0_{-,w}\right)^2 \quad , \tag{3.31}$$

where the ratio R_w is defined as $M^{KB}_{+,w}/(M^{KB}_{+,w} + M^{KB}_{-,w})$. $X^0_{+,w}$ and $X^0_{-,w}$ are the starting points for SGD. Finally, the partial derivatives in Equations. 3.27, 3.28, and 3.29 are revised by adding the corresponding partial derivatives of Equations 3.30 and 3.31 to them.

Mapping to Components in the LML Architecture in Section 1.3

We now map the components of LSC to the components of the general architecture of LML in Section 1.3. In this case, the knowledge base of LSC is the **knowledge base** of LML in Section 1.3. It stores the domain level knowledge and document-level knowledge discussed in Section 3.6.2. The raw datasets of the previous tasks are not stored. **Knowledge-based learner** is the LSC algorithm. The issues of correctness and applicability of previous knowledge are dealt with through the penalty terms of Equations 3.30 and 3.31.

3.7 SUMMARY AND EVALUATION DATASETS

Although lifelong machine learning (LML) started with supervised learning more than 20 years ago, existing work is still limited in both the variety and depth. There is still no general mechanism or algorithm that can be applied in a variety of tasks like existing machine learning algorithms such as SVM, naive Bayes, and deep learning, which can be used for almost any supervised learning task. There are many reasons for this. Perhaps, the most difficult issue is that the research community still does not have a good understanding of what the knowledge is in general, how to represent knowledge, and how to use knowledge in learning effectively. A unified theory of knowledge and related issues is urgently needed. Another reason is that knowledge from supervised learning is difficult to use across domains because to some extent optimization is an obstacle for reuse or transfer. Each model is highly optimized for its specific task. It is difficult to pick and choose some pieces of the knowledge learned from previous tasks or domains and apply them to new tasks because a model is often not decomposable. For example, it is very difficult to reuse any knowledge in a SVM model or apply it in different but similar tasks. Simpler models are often much easier to reuse. For example, it is not hard to pick some rules from a rule-based classifier and use them to help learning a new task. This is probably why human learning is not optimized as we are not good at optimization.

Self-motivated learning is an area that has barely been touched in current research. Ideally, lifelong learning should also identify what to learn rather than always being guided or supervised by human users. Cumulative learning touched the issue. A self-motivated learning system must be able to detect new things using the open world classification paradigm. Without this capability, it is probably impossible to do self-motivated learning because the system would not know what to learn. All these problems and many others still need a great deal of future research.

Evaluation Datasets

To help researchers working in the field, we summarize the evaluation datasets used in the papers covered in the chapter. For those publicly available datasets, we provide their URLs.

Thrun [1996b] used a dataset of color camera images of different objects (such as bottle, hammer, and book) in evaluation. Caruana [1997] used the dataset 1D-ALVINN [Pomer-

leau, 2012] in the road-following domain. They also created the dataset 1D-DOORS [Caruana, 1997] in the object-recognition domain. In addition, a medical decision-making application was also tested in [Caruana, 1997]. Fei et al. [2016] evaluated their method using the 100-products Amazon review dataset created by Chen and Liu [2014b][1] and the popular text classification dataset 20-Newsgroup[2][3]. The 100-products Amazon review dataset contains Amazon reviews from 100 different types of products. Each type of product (or domain) has 1000 reviews. The 20-Newsgroup dataset contains news articles of 20 different topics. Each topic has about 1000 articles. Ruvolo and Eaton [2013b] used three (3) datasets in their evaluation. The first is the land mine dataset from [Xue et al., 2007], which detects whether or not a land mine appears in an area according to radar images. The second is the facial expression recognition challenge dataset in [Valstar et al., 2011][4]. The third is a London Schools dataset[5]. Chen et al. [2015] evaluated on Amazon reviews from 20 diverse product domains, which is a subset of the dataset in [Chen and Liu, 2014b][6].

[1] https://www.cs.uic.edu/~zchen/downloads/ICML2014-Chen-Dataset.zip
[2] http://qwone.com/~jason/20Newsgroups/
[3] https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups
[4] http://gemep-db.sspnet.eu
[5] https://github.com/tjanez/PyMTL/tree/master/data/school
[6] https://www.cs.uic.edu/~zchen/downloads/KDD2014-Chen-Dataset.zip

CHAPTER 4

Lifelong Unsupervised Learning

This chapter focuses on *lifelong unsupervised learning*. Much of work in this area is done in the contexts of topic modeling and information extraction. These two areas are well suited to lifelong machine learning (LML). In the case of topic modeling, topics learned in the past in related domains can obviously be used to guide the modeling in the new or current domain [Chen and Liu, 2014a,b, Wang et al., 2016]. The *knowledge base* (KB) (Section 1.3) stores the past topics. Note that in this chapter, we use the terms *domain* and *task* interchangeably as in the existing research, each task is from a different domain. In terms of information extraction (IE), LML is also natural because the goal of IE is to extract and accumulate as much useful information or knowledge as possible. The extraction process is thus by nature continuous and cumulative. The extracted information earlier can be used to help extract more information later with higher quality [Liu et al., 2016]. These all match the objectives of LML. In the IE case, the knowledge base stores the extracted information and some other forms of useful knowledge. Even with the current primitive LML techniques, these tasks can already produce significantly better results regardless whether the data is large or small. When the data size is small, these techniques are even more advantageous. For example, when the data is small, traditional topic models produce very poor results, but lifelong topic models can still generate very good topics. Ideally, as the knowledge base expands, more extractions will be made and few errors will incur. This is similar to our human learning. As we become more and more knowledgeable, it is easier for us to learn more and also make fewer mistakes. In the following sections, we discuss the current representative techniques of lifelong unsupervised learning which are geared towards achieving these properties.

4.1 LIFELONG TOPIC MODELING

Topic models, such as LDA [Blei et al., 2003] and pLSA [Hofmann, 1999], are unsupervised learning methods for discovering topics from a set of text documents. They have been applied to numerous applications, e.g., opinion mining [Chen et al., 2014, Liu, 2012, Mukherjee and Liu, 2012, Zhao et al., 2010], machine translation [Eidelman et al., 2012], word sense disambiguation [Boyd-Graber et al., 2007], phrase extraction [Fei et al., 2014] and information retrieval [Wei and Croft, 2006]. In general, topic models assume that each document

discusses a set of topics. probabilistically, a multinomial distribution over the set of topics, and each topic is indicated by a set of topical words, probabilistically, a multinomial distribution over the set of words. The two kinds of distributions are called *document-topic distribution* and *topic-word distribution* respectively. The intuition is that some words are more or less likely to be present given the topics of a document. For example, "sport" and "player" will appear more often in documents about sports, "rain" and "cloud" will appear more frequently in documents about weather.

However, fully unsupervised topic models tend to generate many inscrutable topics. The main reason is that the objective functions of topic models are not always consistent with human judgment [Chang et al., 2009]. To deal with this problem, we can use any of the following three approaches:

1. *Inventing better topic models*: This approach may work if a large number of documents is available. If the number of documents is small, regardless how good the model is, it will not generate good topics simply because topic models are unsupervised learning methods and insufficient data cannot provide reliable statistics for modeling. Some form of supervision or external information beyond the given documents is necessary.

2. *Asking users to provide prior domain knowledge*: This approach asks the user or a domain expert to provide some prior domain knowledge. One form of knowledge can be in the form of *must-links* and *cannot-links*. A must-link states that two terms (or words) should belong to the same topic, e.g., *price* and *cost*. A cannot-link indicates that two terms should not be in the same topic, e.g., *price* and *picture*. Some existing *knowledge-based topic models* (e.g., [Andrzejewski et al., 2009, 2011, Chen et al., 2013b,c, Hu et al., 2011, Jagarlamudi et al., 2012, Mukherjee and Liu, 2012, Petterson et al., 2010, Xie et al., 2015]) have used such prior domain knowledge to produce better topics. However, asking the user to provide prior knowledge is problematic in practice because the user may not know what knowledge to provide and wants the system to discover useful knowledge for him/her. It also makes the approach non-automatic.

3. *Using lifelong topic modeling*: This approach incorporates LML in topic modeling. Instead of asking the user to provide prior knowledge, prior knowledge is learned and accumulated automatically in the modeling of previous tasks. The approach works because of the observation that there are usually a great deal of sharing of concepts or topics across domains and tasks in natural language processing [Chen and Liu, 2014a,b], e.g., in sentiment analysis [Liu, 2012, 2015] as we discussed in the preface of this book. We will give some examples shortly too.

We focus on the third approach. Following the definition in Chapter 1, each task here means to perform topic modeling on a set of documents of a particular domain. The knowledge base (KB) stores all the topics obtained from each of the previous tasks, which are used in various ways as prior knowledge in different lifelong topic models.

At the beginning, the KB is either empty or filled with knowledge from an external source such as WordNet [Miller, 1995]. It grows with the results of incoming topic modeling tasks. Since all the tasks are about topic modeling, we use *domains* to distinguish the tasks. Two topic modeling tasks are different if their corpus domains are different. The scope of a domain is quite general. A domain can be a category (e.g., sports) or a product (e.g., camera) or an event (e.g., presidential election). We use $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_N$ to denote the sequence of previous tasks, $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_N\}$ to denote their corresponding data or corpora, and use $\mathcal{T}_{N+1}$ to denote the new or current task with its data $\mathcal{D}_{N+1}$.

Key questions in lifelong topic modeling

For lifelong topic modeling to work, several questions need to be answered. Different models have different strategies to answer these questions.

1. What past knowledge should be retained and accumulated in the KB? As indicated above, in existing models, only the output topics from each previous domain/task are retained.

2. What kinds of knowledge should be used in the new domain modeling and how to mine such knowledge from the KB? Note that the raw past topics in the KB may not be directly used in topic modeling. Current lifelong topic models use must-link and cannot-link types of knowledge mined from the raw past topics stored in the KB.

3. How to assess the quality of knowledge and how to deal with possibly wrong knowledge? Previous modeling can make mistakes, and wrong knowledge from the past is often detrimental to new modeling.

4. How to apply the knowledge in the modeling process to generate better topics in the new domain?

Why does lifelong topic modeling work?

The motivation for lifelong topic modeling is that topics from a large number of previous domains can provide high quality knowledge to guide the modeling in the new domain to produce better topics. Although every domain is different, there is often a fair amount of concept or topic overlapping across domains. Using product reviews of different types of products (or domains) as an example, we observe that every product review domain probably has the topic of *price*, reviews of most electronic products share the topic of *battery* and reviews of some products share the topic of *screen*. Topics produced from a single domain can be erroneous (i.e., a topic may contain some irrelevant words in its top ranked positions), but if a set of shared words among some topics generated from multiple domains can be found, these shared words are more likely to be correct or coherent for a particular topic. They can serve as a piece of prior knowledge to help topic modeling.

For example, we have product reviews from three domains. The classic topic model such as LDA [Blei et al., 2003] is used to generate a set of topics from each domain. Every domain has a topic about *price*, which is listed below with its top four words (words are ranked based on their probabilities under each topic):

- Domain 1: *price*, *color*, *cost*, *life*
- Domain 2: *cost*, *picture*, *price*, *expensive*
- Domain 3: *price*, *money*, *customer*, *expensive*

These topics are not perfect due to the incoherent words (words that do not indicate the main topic): *color*, *life*, *picture*, and *customer*. However, if we focus on those words that appear together in the same topic at least in two domains (the underlined words), we find the following two sets:

$$\{price, cost\} \text{ and } \{price, expensive\}.$$

The words in each of the sets are likely to belong to the same topic. As such, {*price*, *cost*} and {*price*, *expensive*} can serve as prior or past knowledge. That is, a piece of knowledge contains words that are semantically correlated. These two sets are called *must-links*.

With the help of the knowledge, a new model can be designed to adjust the probability and improve the output topics for each of the above three domains or a new domain. Given the above knowledge indicating *price* and *cost* are related, *price* and *expensive* are related, a new topic may be found in Domain 1: *price*, *cost*, *expensive*, *color*, which has three coherent words in the top four positions rather than only two words as in the original topic. This represents a good topic improvement.

In the next section, we review the LTM model [Chen and Liu, 2014a], which uses only must-links as prior knowledge. Its main idea is also applied in the LAST model for a sentiment analysis task [Wang et al., 2016]. In Section 4.3, we review the more advanced model AMC [Chen and Liu, 2014b], which can use both must-links and cannot-links as prior knowledge to model in a new domain with only a small set of documents. There is also another model called AKL [Chen et al., 2014] that clusters past topics before mining must-links. Since both LTM and AMC improve AKL, AKL will not be discussed further.

4.2 LTM: A LIFELONG TOPIC MODEL

LTM (Lifelong Topic Model) was proposed in [Chen and Liu, 2014a]. It works in the following lifelong setting: At a particular point in time, a set of N previous modeling tasks have been performed. From each past task/domain data (or document set) $\mathcal{D}_i \in \mathcal{D}^p$, a set of topics $\mathcal{S}_i$ has been generated. Such topics are called *prior topics* (or *p-topics* for short). Topics from all past tasks are stored in the *Knowledge Base* (KB) $\boldsymbol{\mathcal{S}}$ (known as the *topic base* in [Chen and Liu, 2014a]). At a new time point, a new task represented by a new domain

document set $\mathcal{D}_{N+1}$ arrives for topic modeling. This is also called the *current domain*. LTM does not directly use the p-topics in $\boldsymbol{\mathcal{S}}$ as knowledge to help its modeling. Instead, it mines *must-links* from $\boldsymbol{\mathcal{S}}$ and use the must-links as *prior knowledge* to help model inferencing for the $(N+1)$th task. The process is dynamic and iterative. Once modeling on $\mathcal{D}_{N+1}$ is done, its resulting topics are added to $\boldsymbol{\mathcal{S}}$ for future use. LTM has two key characteristics:

1. LTM's knowledge mining is targeted, meaning that it only mines useful knowledge from those relevant p-topics in $\boldsymbol{\mathcal{S}}$. To do this, LTM performs a topic modeling on $\mathcal{D}_{N+1}$ first to find some initial topics and then uses these topics to find similar p-topics in $\boldsymbol{\mathcal{S}}$. Those similar p-topics are used to mine must-links (knowledge) which are more likely to be applicable and correct. These must-links are then used in the next iteration of modeling to guide the inference to generate more accurate topics.

2. LTM is a fault-tolerant model as it is able to deal with errors in automatically mined must-links. First, due to wrong topics (topics with many incoherent/wrong words or topics without a dominant semantic theme) in $\boldsymbol{\mathcal{S}}$ or mining errors, the words in a must-link may not belong to the same topic in general. Second, the words in a must-link may belong to the same topic in some domains, but not in others due to the domain diversity. Thus, to apply such knowledge in modeling, the model must deal with possible errors in must-links.

4.2.1 LTM MODEL

Like many topic models, LTM uses Gibbs sampling for inference [Griffiths and Steyvers, 2004]. Its graphical model is the same as LDA, but it has a very different sampler which can incorporate prior knowledge and also handle errors in the knowledge as indicated above. The LTM system is illustrated in Figure 4.1, in the general LML framework of Figure 1.2.

LTM works as follows (Algorithm 2): It first runs the **Gibbs sampler** of LTM for M iterations (or sweeps) to find a set of initial topics $\mathcal{A}_{N+1}$ from $\mathcal{D}_{N+1}$ with no knowledge (line 1). It then makes another M Gibbs sampling sweeps (lines 2-5). But before each of these new sweeps, it first mines a set of targeted *must-links* (knowledge) $\boldsymbol{\mathcal{K}}_{N+1}$ for every topic in $\mathcal{A}_{N+1}$ using the function **TopicKnowledgeMiner** (Algorithm 3, detailed in the next subsection) and then uses $\boldsymbol{\mathcal{K}}_{N+1}$ to generate a new set of topics $\mathcal{A}_{N+1}$ from $\mathcal{D}_{N+1}$. To distinguish topics in $\mathcal{A}_{N+1}$ from p-topics, these new topics are called the *current topics* (or *c-topics* for short). We say that the mined must-links are targeted because they are mined based on the c-topics in $\mathcal{A}_{N+1}$ and are targeted at improving the topics in $\mathcal{A}_{N+1}$. Note that to make the algorithm more efficient, it is not necessary to mine knowledge for every sweep. Section 4.2.2) focuses on the topic knowledge mining function of LTM. The Gibbs sampler will be given in Section 4.2.4. Line 6 simply updates the knowledge base, which is simple as each task is from a distinct domain in this paper. The set of topics is simply added to the **knowledge base** $\boldsymbol{\mathcal{S}}$ for future use.

Algorithm 2 Lifelong Topic Modeling (LTM)

Input: New domain data $\mathcal{D}_{N+1}$; Knowledge Base $\boldsymbol{\mathcal{S}}$
Output: Topics from new domain $\mathcal{A}_{N+1}$

1: $\mathcal{A}_{N+1} \leftarrow$ GibbsSampler($\mathcal{D}_{N+1}$, $\varnothing$, M) // Run M iterations with no knowledge
2: **for** $i = 1$ **to** M **do**
3: $\quad \boldsymbol{\mathcal{K}}_{N+1} \leftarrow$ TopicKnowledgeMiner($\mathcal{A}_{N+1}$, $\boldsymbol{\mathcal{S}}$)
4: $\quad \mathcal{A}_{N+1} \leftarrow$ GibbsSampler($\mathcal{D}_{N+1}$, $\boldsymbol{\mathcal{K}}_{N+1}$, 1) // Run with knowledge $\boldsymbol{\mathcal{K}}_{N+1}$
5: **end for**
6: $\boldsymbol{\mathcal{S}} \leftarrow$ UpdateKB($\mathcal{A}_{N+1}, \boldsymbol{\mathcal{S}}$)

Algorithm 3 TopicKnowledgeMiner

Input: topics from new domain $\mathcal{A}_{N+1}$; knowledge base $\boldsymbol{\mathcal{S}}$
Output: must-links (knowledge) $\boldsymbol{\mathcal{K}}_{N+1}$ for new domain

1: **for** each p-topic $s_k \in \boldsymbol{\mathcal{S}}$ **do**
2: $\quad j^* = \min_j$ KL-Divergence(a_j, s_k) for each c-topic $a_j \in \mathcal{A}_{N+1}$
3: $\quad$ **if** KL-Divergence(a_{j^*}, s_k) $\leq \pi$ **then**
4: $\quad\quad \mathcal{M}^{N+1}{}_{j^*} \leftarrow \mathcal{M}^{N+1}{}_{j^*} \cup \{s_k\}$
5: $\quad$ **end if**
6: **end for**
7: $\boldsymbol{\mathcal{K}}_{N+1} \leftarrow \cup_{j^*}$ FIM($\mathcal{M}^{N+1}{}_{j^*}$). // Frequent Itemset Mining

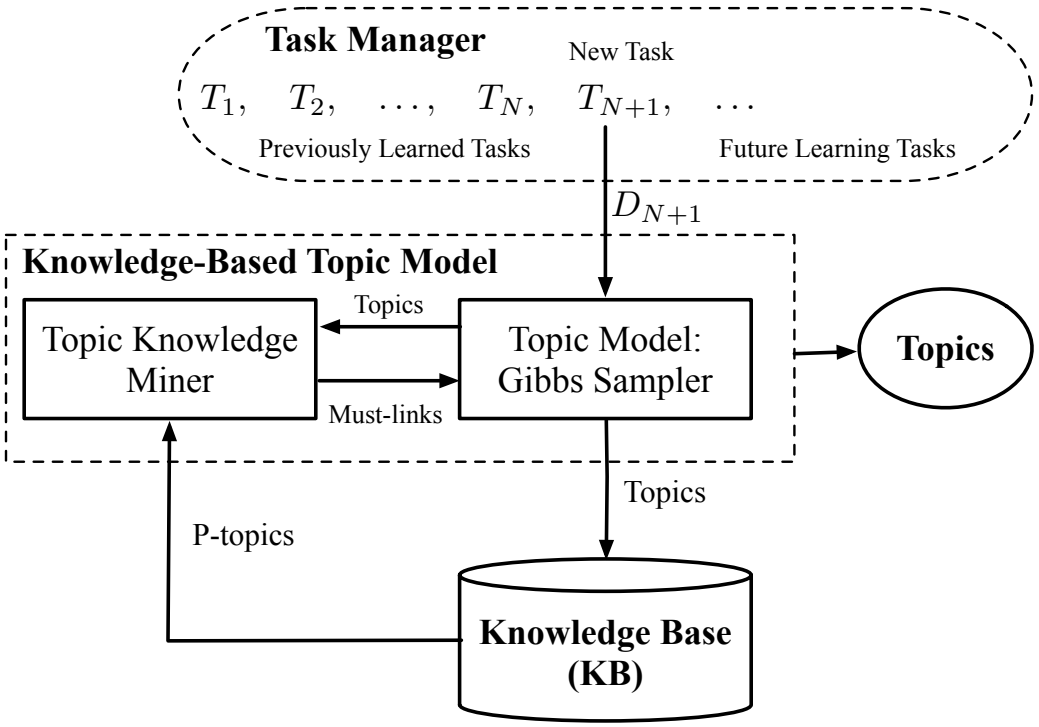


Figure 4.1: The Lifelong Topic Model (LTM) system architecture.

4.2.2 TOPIC KNOWLEDGE MINING

The **TopicKnowledgeMiner** function is given in Algorithm 3. For each p-topic s_k in $\boldsymbol{\mathcal{S}}$, it finds the best matching (or the most similar) c-topic a_{j^*} in the c-topic set $\mathcal{A}_{N+1}$ (line 2). The matching is done using KL-Divergence (line 2) since each topic is a distribution over words. ${\mathcal{M}^{N+1}}_{j^*}$ is used to store all matching p-topics for each c-topic a_{j^*} (line 4). Note that the matching p-topics are found for each individual c-topic a_{j^*} because a_{j^*} specific p-topics are preferable for more accurate knowledge (must-links) mining, which is done in line 7. In other words, these matching p-topics ${\mathcal{M}^{N+1}}_{j^*}$ are targeted towards each a_{j^*} and should provide high quality knowledge for a_{j^*}. ${\mathcal{M}^{N+1}}_{j^*}$ is mined to generate must-links

${\mathcal{K}^{N+1}}_{j^*}$ for each c-topic a_{j^*}. Must-links mined for all c-topics in $\mathcal{A}_{N+1}$ are stored in $\boldsymbol{\mathcal{K}}_{N+1}$. Below, we describe topic matching and knowledge mining in greater detail.

Topic matching (lines 2-5, Algorithm 3): To find the best match for s_k in $\boldsymbol{\mathcal{S}}$ with a c-topic a_{j^*} in $\mathcal{A}_{N+1}$, KL-Divergence is used, which computes the difference between two distributions (lines 2 and 3). Specifically, Symmetrised KL (SKL) Divergence is employed, i.e., given two distributions P and Q, the divergence is calculated as:

$$SKL(P,Q) = \frac{KL(P,Q) + KL(Q,P)}{2} \quad , \quad \text{and} \tag{4.1}$$

$$KL(P,Q) = \sum_i ln\left(\frac{P(i)}{Q(i)}\right) P(i) \quad . \tag{4.2}$$

The c-topic with the minimum SKL Divergence with s_k is denoted by a_{j^*}. Parameter π is used to ensure that the p-topics in ${\mathcal{M}^{N+1}}_{j^*}$ are reasonably correlated with a_{j^*}.

Mining must-link knowledge using frequent itemset mining (FIM): Given the p-topics in each matching set ${\mathcal{M}^{N+1}}_{j^*}$, this step finds sets of words that appear together multiple times in these p-topics. The shared words among matching p-topics across multiple domains are likely to belong to the same topic. To find such shared words in the matching set of p-topics ${\mathcal{M}^{N+1}}_{j^*}$, frequent itemset mining (FIM) is used [Agrawal and Srikant, 1994].

FIM is stated as follows: Given a set of transactions $\mathcal{X}$, where each transaction $x_i \in \mathcal{X}$ is a set of items. In our context, x_i is a set of top words of a p-topic (no probability attached). $\mathcal{X}$ is ${\mathcal{M}^{N+1}}_{j^*}$ without lowly ranked words in each p-topic as only the top words are usually representative of a topic. The goal of FIM is to find every itemset (a set of items) that satisfies some user-specified frequency threshold (also called *minimum support*), which is the minimum number of times that an itemset should appear in $\mathcal{X}$. Such itemsets are called *frequent itemsets*. In the context of LTM, a frequent itemset is a set of words that have appeared together multiple times in the p-topics of ${\mathcal{M}^{N+1}}_{j^*}$, which is a must-link.

Only frequent itemsets of length two, i.e., each must-link has only two words, are used in the LTM model, e.g., {battery, life}, {battery, power}, {battery, charge}. Larger sets tend to contain more errors.

4.2.3 INCORPORATING PAST KNOWLEDGE

As each must-link reflects a possible semantic similarity relation between a pair of words, the *generalized Pólya urn* (GPU) model [Mahmoud, 2008] is used to leverage this knowledge in the Gibbs sampler of LTM to encourage the pair of words to appear in the same topic. Below, we first introduce the Pólya urn model which serves as the basic framework to incorporate knowledge, and then presents the generalized Pólya urn model, which can deal with possible errors in must-links to make LTM fault-tolerant to some extent.

Simple Pólya Urn Model. The Pólya urn model works on colored balls and urns. In the topic model context, a term/word can be seen as a ball of a certain color and a topic

as an urn. The distribution of a topic is reflected by the color proportions of balls in the urn. LDA follows the simple Pólya urn (SPU) model in the sense that when a ball of a particular color is drawn from an urn, the ball is put back to the urn along with a new ball of the same color. The content of the urn changes over time, which gives a self-reinforcing property known as "the rich get richer". This process corresponds to assigning a topic to a term in Gibbs sampling.

Generalized Pólya urn Model. The generalized Pólya urn (GPU) model [Chen and Liu, 2014a, Mahmoud, 2008, Mimno et al., 2011] differs from SPU in that, when a ball of a certain color is drawn, two balls of that color are put back along with a certain number of balls of some other colors. These additional balls of some other colors added to the urn increase their proportions in the urn. This is the key technique for incorporating must-links as we will see below.

Applying the GPU model to topic modeling, when a word w is assigned to a topic t, each word w' that shares a must-link with w is also assigned to the topic t by a certain amount, which is decided by the matrix $\mathbf{A}'_{t,w',w}$. w' is thus promoted by w, meaning that the probability of w' under topic t is also increased. Here, a must-link of a topic t means this must-link is extracted from the p-topics matching with topic t.

The problem is how to set proper values for matrix $\mathbf{A}'_{t,w',w}$. To answer this question, let us also consider the problem of wrong knowledge. Since the must-links are mined from p-topics in multiple previous domains automatically, the semantic relationship of words in a must-link may not be correct for the current domain. It is a challenge to determine which must-link is not appropriate. One way to deal with the problem is to assess how the words in a must-link correlated with each other in the current domain. If they are more correlated, they are more likely to be correct for a topic in the domain and thus should be promoted more. If they are less correlated, they are more likely to be wrong and should be promoted less (or even not promoted).

To measure the correlation of two words in a must-link in the current domain, Pointwise Mutual Information (PMI) is used, which is a measure of words association in text [Church and Hanks, 1990]. In this case, it measures the extent to which two words tend to co-occur, which corresponds to the higher-order co-occurrence on which topic models are based [Heinrich, 2009]. The PMI of two words is defined as follows:

$$PMI(w_1, w_2) = log \frac{P(w_1, w_2)}{P(w_1)P(w_2)} \quad , \tag{4.3}$$

where $P(w)$ denotes the probability of seeing word w in a random document, and $P(w_1, w_2)$ denotes the probability of seeing both words co-occurring in a random document. These probabilities are empirically estimated using the current domain collection $\mathcal{D}_{N+1}$:

$$P(w) = \frac{\#\mathcal{D}_{N+1}(w)}{\#\mathcal{D}_{N+1}} \quad , \tag{4.4}$$

$$P(w_1, w_2) = \frac{\#\mathcal{D}_{N+1}(w_1, w_2)}{\#\mathcal{D}_{N+1}} \ , \tag{4.5}$$

where $\#\mathcal{D}_{N+1}(w)$ is the number of documents in $\mathcal{D}_{N+1}$ that contain word w and $\#\mathcal{D}_{N+1}(w_1, w_2)$ is the number of documents that contain both words w_1 and w_2. $\#\mathcal{D}_{N+1}$ is the total number of documents in $\mathcal{D}_{N+1}$. A positive PMI value implies a true semantic correlation of words, while a non-positive PMI value indicates little or no semantic correlation. Thus, only must-links with positive PMI values are considered. A parameter factor μ is added to control how much the GPU model should trust the word relationships indicated by PMI. The amount of promotion for word w' when seen w is defined as follows:

$$\mathbf{A}'_{t,w,w'} = \begin{cases} 1 & w = w' \\ \mu \times PMI(w, w') & (w, w') \text{ is a must-link of topic } t \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

4.2.4 CONDITIONAL DISTRIBUTION OF GIBBS SAMPLER

The GPU model is nonexchangeable, i.e., the joint probability of the words in any given topic is not invariant to the permutation of those words. The inference for the model can be computationally expensive due to the nonexchangeability of words, that is, the sampling distribution for the word of interest depends on each possible value for the subsequent words along with their topic assignments. LTM takes the approach of [Mimno et al., 2011] which approximates the true Gibbs sampling distribution by treating each word as if it were the last. The approximate Gibbs sampler has the following conditional distribution:

$$\begin{aligned} &P(z_i = t | \boldsymbol{z}^{-i}, \boldsymbol{w}, \alpha, \beta, \mathbf{A}') \propto \\ &\frac{n^{-i}_{d,t} + \alpha}{\sum_{t'=1}^{T}(n^{-i}_{d,t'} + \alpha)} \times \frac{\sum_{w'=1}^{V} \mathbf{A}'_{t,w',w_i} \times n^{-i}_{t,w'} + \beta}{\sum_{v=1}^{V}(\sum_{w'=1}^{V} \mathbf{A}'_{t,w',v} \times n^{-i}_{t,w'} + \beta)} \ , \end{aligned} \tag{4.7}$$

where n^{-i} is the count excluding the current assignment of z_i, i.e., $\boldsymbol{z}^{-i}$, $\boldsymbol{w}$ refers to all the words in all documents in the document collection $\mathcal{D}_{N+1}$ and w_i is the current word to be sampled with a topic denoted by z_i. $n_{d,t}$ denotes the number of times that topic t was assigned to words in document d, where d is the document index of word w_i. $n_{t,v}$ refers to the number of times that word v appears under topic t. α and β are predefined Dirichlet hyperparameters. T is the number of topics, and V is the vocabulary size. $\mathbf{A}'$ is the promotion matrix defined in Equation 4.6.

Mapping to Components in the LML Architecture in Section 1.3

The knowledge base of LTM is the **knowledge base** of Section 1.3. It stores only the past topics or p-topics from previous tasks. **Knowledge-based learner** here is the *knowledge-*

based topic model here in Figure 4.1. The issues of correctness and applicability of previous knowledge (must-links) are dealt with using frequent itemset mining (FIM) and Pointwise Mutual Information (PMI).

4.3 AMC: A LIFELONG TOPIC MODEL FOR SMALL DATA

The LTM model needs a fairly large set of documents in order to generate reasonable initial topics to be used in finding similar past topics in the knowledge base to mine appropriate must-link knowledge. However, when the document set (or data) is very small, this approach does not work because the initial modeling produces very poor topics, which cannot be used to find matching or similar past topics in the knowledge base to serve as prior knowledge. A new approach is thus needed. The AMC model (topic modeling with Automatically generated Must-links and Cannot-links) [Chen and Liu, 2014b] aims to solve the problem. AMC's must-link knowledge mining does not use any information from the new domain/task. Instead, it mines must-links from the past topics independent of the new domain. However, to make the resulting topics accurate, must-link knowledge is far from sufficient. Thus, AMC also uses cannot-links, which are hard to mine independent of the new domain data due to the high computational complexity. Cannot-links are mined dynamically. All these are detailed in this section.

Algorithm 4 AMC Model

Input: New domain data $\mathcal{D}_{N+1}$; Knowledge Base $\boldsymbol{\mathcal{S}}$; Must-links $\boldsymbol{\mathcal{M}}$
Output: Topics from new domain $\mathcal{A}_{N+1}$

1: $\boldsymbol{\mathcal{M}} \leftarrow$ MustLinkMiner($\boldsymbol{\mathcal{S}}$)
2: $\boldsymbol{\mathcal{C}} = \varnothing$ // $\boldsymbol{\mathcal{C}}$ stores cannot-links
3: $\mathcal{A}_{N+1} \leftarrow$ GibbsSampler($\mathcal{D}_{N+1}$, $\boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{C}}$, M); // Run M Gibbs iterations with must-links $\boldsymbol{\mathcal{M}}$ but no cannot-links
4: **for** $r = 1$ **to** R **do**
5: $\quad \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \cup$ CannotLinkMiner($\boldsymbol{\mathcal{S}}$, $\mathcal{A}_{N+1}$)
6: $\quad \mathcal{A}_{N+1} \leftarrow$ GibbsSampler($\mathcal{D}_{N+1}$, $\boldsymbol{\mathcal{M}}$, $\boldsymbol{\mathcal{C}}$, N)
7: **end for**
8: $\boldsymbol{\mathcal{S}} \leftarrow$ UpdateKB($\mathcal{A}_{N+1}$, $\boldsymbol{\mathcal{S}}$)

4.3.1 OVERALL ALGORITHM OF AMC

Algorithm 4 gives the overall algorithm of AMC, which is also illustrated in Figure 4.2. Line 1 mines a set of must-links $\boldsymbol{\mathcal{M}}$ using the function **MustLinkMiner** from previous topics (or p-topics) in the **knowledge base** (KB) $\boldsymbol{\mathcal{S}}$. Note here the must-links can be generated

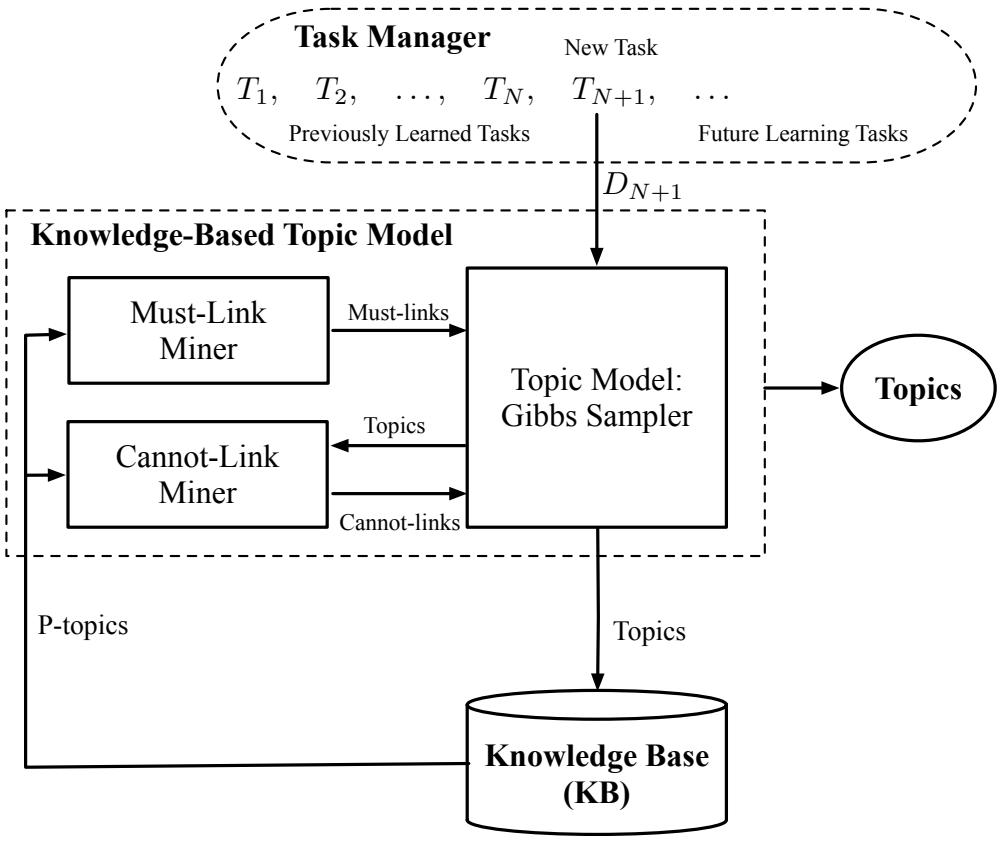


Figure 4.2: The AMC model system architecture.

offline independent of the current new task. Line 3 runs the proposed **Gibbs sampler** (introduced in Section 4.3.5) using only the must-links $\mathcal{M}$ to produce a set of topics $\mathcal{A}_{N+1}$, where M is the number of Gibbs sampling iterations. Line 5 mines cannot-links $\mathcal{C}$ using the function **CannotLinkMiner** based on the current topics $\mathcal{A}_{N+1}$ and p-topics in the knowledge base $\mathcal{S}$ (see Sections 4.3.3). Then line 6 uses both must-links $\mathcal{M}$ and cannot-links $\mathcal{C}$ to improve the resulting topics. This process can run iteratively (R times) to obtain a set of superior topics to be stored in the knowledge base and also output to the user. Function $UpdateKB(\mathcal{A}_{N+1}, \mathcal{S})$ (line 8) is very simple at the moment. If the domain of $\mathcal{A}_{N+1}$ exists in $\mathcal{S}$, replace those topics of the domain in $\mathcal{S}$ with $\mathcal{A}_{N+1}$; otherwise, $\mathcal{A}_{N+1}$ is added to $\mathcal{S}$.

4.3.2 MINING MUST-LINK KNOWLEDGE

Since AMC cannot use topics from the new domain to find similar topics in the knowledge base like LTM, it mines must-links directly from the knowledge base without considering the context of any new task using the function **MustLinkMiner**. Recall that each topic generated from a topic model, such as LDA, is a distribution over words, i.e., words with their associated probabilities. Words are commonly ranked based on their probabilities in a descending order. In practice, top words under a topic are expected to represent some similar semantic meaning. The lower ranked words usually have very low probabilities due to the smoothing effect of the Dirichlet hyperparameters rather than true correlations within the topic, leading to their unreliability. Thus, in [Chen and Liu, 2014b], only top 15 words are employed to represent a topic. This topic representation is used in mining both must-link and cannot-link knowledge.

Given knowledge base $\mathcal{S}$, similar to the LTM model in Section 4.2, must-links are sets of words that appear together in multiple topics and they are mined using the data mining technique *frequent itemset mining* (FIM). However, this technique is insufficient due to the problem with the single minimum support threshold used in classic FIM algorithms.

A single minimum support is not appropriate because generic topics, such as *price* with topic words like *price* and *cost*, are shared by many (even all) product review domains, but specific topics such as *screen*, occur only in product domains having such features. This means that different topics may have very different frequencies in the data. Thus, using a single minimum support threshold is unable to extract both generic and specific topics because if this threshold is set too low, the popular topics will result in numerous spurious frequent itemsets (which results in wrong must-links) and if it is set too high, must-links from less frequent topics will not be found. This is called the *rare item problem* in data mining and has been well documented in the data mining literature [Liu, 2007].

To address the above issue, the AMC model uses the *multiple minimum supports frequent itemset mining* (MS-FIM) algorithm in Liu et al. [1999]. MS-FIM is stated as follows: Given a set of transactions $\mathcal{R}$, where each transaction $r_i \in \mathcal{R}$ is a set of items from

a global item set $\mathcal{I}$, i.e., $r_i \subseteq \mathcal{I}$. In AMC, r_i is a topic comprising the top words of the topic (no probability attached). An item is a word. $\mathcal{R}$ is thus the collection of all p-topics in the knowledge base $\boldsymbol{\mathcal{S}}$ and $\mathcal{I}$ is the set of all words in $\boldsymbol{\mathcal{S}}$. In MS-FIM, each item/word is given a minimum itemset support (MIS). The minimum support that an itemset (a set of items) must satisfy is not fixed. It depends on the MIS values of all the items in the itemset. MS-FIM also has another constraint, called the *support difference constraint* (SDC), expressing the requirement that the supports of the items in an itemset must not be too different. MIS and SDC together can solve the above rare item problem.

The goal of MS-FIM is to find all itemsets that satisfy the user-specified MIS thresholds and SDC constraints. Such itemsets are called *frequent itemsets*. In AMC, a frequent itemset is a set of words which have appeared multiple times in the p-topics in the knowledge base $\boldsymbol{\mathcal{S}}$. The frequent itemsets of length two are used as the learned must-link knowledge, e.g., {battery, life}, {battery, power}, {battery, charge}, {price, expensive}, {price, pricy}, {cheap, expensive}.

Again each must-link used in AMC has only two words [Chen and Liu, 2014b]. As mentioned in Section 4.2.2, larger sets tend to contain more errors. Such errors are hard to deal with than those in pairs. The same rationale applies to cannot-links.

There are two key challenges in incorporating the must-link knowledge in modeling:

1. A word can have multiple meanings or senses. For example, *light* may mean "something that makes things visible" or "of little weight." Different senses may lead to distinct must-links. For example, with the first sense of *light*, the must-links can be {light, bright} and {light, luminance}. In contrast, {light, weight} and {light, heavy} indicate the second sense of light. Without dealing with this, it can cause the transitivity problem [Chen and Liu, 2014b]. That is, if words w_1 and w_2 form a must-link, and words w_2 and w_3 form a must-link, it implies a must-link between w_1 and w_3, i.e., w_1, w_2, and w_3 should be in the same topic. With transitivity, *light*, *bright*, and *weight* would be incorrectly assumed to be in the same topic.

2. Not every must-link is suitable for a domain. This is the same wrong knowledge problem discussed in Section 4.2.

To deal with the first issue, a must-link graph was proposed in [Chen and Liu, 2014b] to distinguish multiple senses in must-links to solve the transitivity problem. As the must-links are automatically mined from the set of p-topics (for past topics) in $\boldsymbol{\mathcal{S}}$, the p-topics may also provide some guidance on whether the mined must-links share the same word sense or not. Given two must-links m_1 and m_2, if they share the same word sense, the p-topics that cover m_1 should have some overlapping with p-topics that cover m_2. For example, must-links {light, bright} and {light, luminance} should be mostly coming from the same set of p-topics related to the semantic meaning "something that makes things visible" of *light*. On the other hand, little topic overlapping indicates likely different word senses. For

example, must-links {light, bright} and {light, weight} may come from two different sets of p-topics as they usually refer to different topics.

Following this idea, a must-link graph G is constructed where a must-link is a vertex. An edge is formed between two vertices if the two must-links m_1 and m_2 have a shared word. For each edge, the amount of their original p-topics overlapping is used to decide whether the two must-links share the same sense or not. Given two must-links m_1 and m_2, the p-topics in $\boldsymbol{\mathcal{S}}$ covering each of them are denoted by $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. m_1 and m_2 share the same sense if

$$\frac{\#(\mathcal{C}_1 \cap \mathcal{C}_2)}{Max(\#\mathcal{C}_1, \#\mathcal{C}_2)} > \pi_{overlap} \ , \tag{4.8}$$

where $\pi_{overlap}$ is the *overlap threshold* for distinguishing senses. This threshold is necessary due to errors in topics. The edges that do not satisfy the above inequality are discarded. The final must-link graph G can provide some guidance on selecting the right must-links sharing the same word sense in the Gibbs sampler (in Section 4.3.5) for dealing with the transitivity problem.

To tackle the second problem, Pointwise Mutual Information (PMI) was also used to approximate the semantic correlation using the current domain data. This is similar to that in the LTM model (Section 4.2.3) and thus will not be discussed again.

4.3.3 MINING CANNOT-LINK KNOWLEDGE

Although it is reasonable to find must-links from all past topics, it is problematic to find cannot-links from all past topics (p-topics) as it is prohibitive to do so. This is because for a word w, there are usually only a few words w_m that share must-links with w while there is a huge number of words w_c that can form cannot-links with w. In general, if there are V words in the vocabulary of all tasks or domains, then there are $O(V^2)$ potential cannot-links. However, for a new domain $\mathcal{D}_{N+1}$, most of these cannot-links are not useful because the vocabulary size of $\mathcal{D}_{N+1}$ is much smaller than V. Thus, AMC focuses only on those words that are relevant to $\mathcal{D}_{N+1}$.

Formally, given the knowledge base $\boldsymbol{\mathcal{S}}$ and the current c-topics $\mathcal{A}_{N+1}$ from the new task domain data $\mathcal{D}_{N+1}$, cannot-links from each pair of top words w_1 and w_2 in each c-topic $A_j \in \mathcal{A}_{N+1}$ are extracted. Based on this formulation, to mine cannot-links (using **CannotLinkMiner**), the mining algorithm enumerates every pair of top words w_1 and w_2 and checks whether they form a cannot-link or not. Thus, the cannot-link mining is targeted to each c-topic with the aim to improve the c-topic using the discovered cannot-links.

Given two words, CannotLinkMiner determines whether they form a cannot-link or not as follows: If the words seldom appear together in p-topics in $\boldsymbol{\mathcal{S}}$, they are likely to have distinct semantic meanings. Let the number of past domains that w_1 and w_2 appear in different p-topics be N_{diff} and the number of past domains that w_1 and w_2 share the same

topic be N_{share}. N_{diff} should be much larger than N_{share}. Two conditions or thresholds are necessary to control the formation of a cannot-link:

1. The ratio $N_{diff}/(N_{share} + N_{diff})$ (called the *support ratio*) is equal to or larger than a threshold π_c. This condition is intuitive.

2. N_{diff} is greater than a support threshold π_{diff}. This condition is needed because the above ratio can be 1, but N_{diff} can be very small and thus unreliable.

Some cannot-link examples are as follows: {battery, money}, {life, movie}, {battery, line} {price, digital}, {money, slow} and {expensive, simple}.

Similar to must-links, cannot-links can be wrong too. Wrong cannot-links are usually harder to detect and to verify than wrong must-links. Due to the power-law distribution of natural language words [Zipf, 1932], most words are rare and will not co-occur with most other words. The low co-occurrences of two words do not necessarily mean a negative correlation (cannot-link). Chen and Liu [2014b] proposed to detect and balance cannot-links inside the sampling process. More specifically, they extended Pólya urn model to incorporate the cannot-link knowledge, and also to deal with the issues above.

4.3.4 EXTENDED PÓLYA URN MODEL

Gibbs sampler for the AMC model differs from that of LTM as LTM does not consider cannot-links. A *multi-generalized Pólya Urn* (M-GPU) model was proposed in [Chen and Liu, 2014b] for AMC. We have introduced the simple Pólya urn (SPU) model, and the generalized Pólya urn (GPU) model in Section 4.2.3. We now extend the GPU model to the *multi-generalized Pólya urn model* (M-GPU).

Instead of involving only one urn at a time as in the SPU and GPU models, the M-GPU model considers a set of urns in the sampling process simultaneously [Chen and Liu, 2014b]. M-GPU allows a ball to be transferred from one urn to another, enabling multi-urn interactions. Thus, during sampling, the populations of several urns will evolve even if only one ball is drawn from one urn. This capability makes the M-GPU model more powerful and suitable for solving the complex problems discussed so far.

In M-GPU, when a ball is randomly drawn, a certain number of additional balls of each color are returned to the urn, rather than just two balls of the same color as in SPU. This is inherited from the GPU model. As a result, the proportions of these colored balls are increased, making them more likely to be drawn in this urn in the future. This is called the *promotion* of these colored balls in [Chen and Liu, 2014b]. Applying the idea, when a word w is assigned to a topic k, each word w' that shares a must-link with w is also assigned to topic k by a certain amount $\lambda_{w',w}$. The definition of $\lambda_{w',w}$ is similar to the promotion matrix in the LTM model (see Section 4.2.3). Thus, we will not discuss it further here.

To deal with multiple senses problem in M-GPU, Chen and Liu [2014b] exploited the fact that each word usually has only one correct sense or meaning under one topic. Since

the semantic concept of a topic is usually represented by some top words under it, the word sense that is the most related to the concept is treated as the correct sense. If a word w does not have multiple must-links, then there is no multiple sense problem. If w has multiple must-links, the rationale here is to sample a must-link (say m) that contains w to be used to represent the likely word sense from the must-link graph G. The sampling distribution will be given in the next sub-section. Then, the must-links that share the same word sense with m, including m, are used to promote the related words of w.

To deal with cannot-links, M-GPU defines two sets of urns to be used in sampling. The first set is the set of topic urns $U^K_{d\in\mathcal{D}_{N+1}}$, where each urn is for one document and contains balls of K colors (topics) and each ball inside has a color $k \in \{1 \dots K\}$. This corresponds to the document-topic distribution in AMC. The second set of urns is the set of word urns $U^V_{k\in\{1\dots K\}}$ corresponding to the topic-word distributions, with balls of colors (words) $w \in \{1 \dots V\}$ in each word urn.

Based on the definition of cannot-link, two words in a cannot-link cannot both have large probabilities under the same topic. As M-GPU allows multi-urn interactions, when sampling a ball representing word w from a word urn U^V_k, the balls representing the cannot-words of w, say w_c (sharing cannot-links with w) can be transferred to other urns (see Step 5 below), i.e., decreasing the probabilities of those cannot-words (words in a cannot-link) under this topic while increasing their corresponding probabilities under some other topic. The ball representing word w_c should be transferred to an urn which has a higher proportion of w_c. That is, an urn that has a higher proportion of w_c is randomly sampled for w_c to transfer to (Step 5b below). However, it is possible that there is no other urn that has a higher proportion of w_c. There are two ways to deal with this issue: 1) create a new urn to move w_c to, which was used in [Chen et al., 2013c]. This approach assumes that the cannot-link is correct. 2) keep w_c in the urn U^V_k as the cannot-link may not be correct, so it is possible that U^V_k is the right urn for w_c. As discussed in Section 4.3.3, a cannot-link can be wrong. For example, the model puts *battery* and *life* in the same topic k where *battery* and *life* have the highest probabilities (or proportions). However, a cannot-link {battery, life} wants to separate them after seeing them in the same topic. In this case, we should not trust the cannot-link as it wants to split the correlated words into different topics. Chen and Liu [2014b] took the second approach due to the noisy in cannot-links.

Based on all the above ideas, the M-GPU sampling scheme is presented as follows:

1. Sample a topic k from U^K_d and a word w from U^V_k sequentially, where d is the dth document in $\mathcal{D}_{N+1}$.

2. Record k and w, put back two balls of color k into urn U^K_d, and two balls of color w into urn U^V_k.

3. Sample a must-link m that contains w from the prior knowledge base. Get a set of must-links $\{m'\}$ where m' is either m or a neighbor of m in the must-link graph G.

4. For each must-link $\{w, w'\}$ in $\{m'\}$, we put back $\lambda_{w',w}$ number of balls of color w' into urn U_k^V based on matrix $\lambda_{w',w}$.

5. For each word w_c that shares a cannot-link with w:

   (a) Draw a ball q_c of color w_c (to be transferred) from U_k^V and remove it from U_k^V. The document of ball q_c is denoted by d_c. If no ball of color w_c can be drawn (i.e., there is no ball of color w_c in U_k^V), skip steps b) and c).

   (b) Produce an urn set $\{U_{k'}^V\}$ such that each urn in it satisfies the following conditions:

   i) $k' \neq k$

   ii) The proportion of balls of color w_c in $U_{k'}^V$ is higher than that of balls of color w_c in U_k^V.

   (c) If $\{U_{k'}^V\}$ is not empty, randomly select one urn $U_{k'}^V$ from it. Put the ball q_c drawn from Step a) into $U_{k'}^V$. Also, remove a ball of color k from urn $U_{d_c}^K$ and put back a ball of k' into urn $U_{d_c}^K$. If $\{U_{k'}^V\}$ is empty, put the ball q_c back to U_k^V.

4.3.5 SAMPLING DISTRIBUTIONS IN GIBBS SAMPLER

For each word w_i in each document d, sampling consists of two phases based on the M-GPU sampling process above:

Phase 1 (Steps 1-4 in M-GPU): calculate the conditional probability of sampling a topic for word w_i. The process enumerates each topic k and calculates its corresponding probability, which is decided by three sub-steps:

a) Sample a must-link m_i that contains w_i, which is likely to have the word sense consistent with topic k, based on the following conditional distribution:

$$P(m_i = m|k) \propto P(w_1|k) \times P(w_2|k) \ , \tag{4.9}$$

where w_1 and w_2 are the words in must-link m and one of them is the same as w_i. $P(w|k)$ is the probability of word w under topic k given the current status of the Markov chain in the Gibbs sampler, which is defined as:

$$P(w|k) \propto \frac{\sum_{w'=1}^{V} \lambda_{w',w} \times n_{k,w'} + \beta}{\sum_{v=1}^{V}(\sum_{w'=1}^{V} \lambda_{w',v} \times n_{k,w'} + \beta)} \ , \tag{4.10}$$

where $n_{k,w}$ refers to the number of times that word w appears under topic k. β is the predefined Dirichlet hyper-parameter.

b) After getting the sampled must-link m_i, a set of must-links $\{m'\}$ are created where m' is either m_i or a neighbor of m_i in the must-link graph G. The must-links in this set

$\{m'\}$ are likely to share the same word sense of word w_i according to the corresponding edges in the must-link graph G.

c) The conditional probability of assigning topic k to word w_i is defined as below:

$$\begin{aligned} & p(z_i = k|\boldsymbol{z}^{-i}, \boldsymbol{w}, \alpha, \beta, \lambda) \\ & \propto \frac{n_{d,k}^{-i} + \alpha}{\sum_{k'=1}^{K}(n_{d,k'}^{-i} + \alpha)} \\ & \times \frac{\sum_{\{w',w_i\}\in\{m'\}} \lambda_{w',w_i} \times n_{k,w'}^{-i} + \beta}{\sum_{v=1}^{V}(\sum_{\{w',v\}\in\{m'_v\}} \lambda_{w',v} \times n_{k,w'}^{-i} + \beta)} \ , \end{aligned} \tag{4.11}$$

where n^{-i} is the count excluding the current assignment of z_i, i.e., $\boldsymbol{z}^{-i}$. $\boldsymbol{w}$ refers to all the words in all documents in the new document collection $\mathcal{D}_{N+1}$ and w_i is the current word to be sampled with a topic denoted by z_i. $n_{d,k}$ denotes the number of times that topic k is assigned to words in document d. $n_{k,w}$ refers to the number of times that word w appears under topic k. α and β are predefined Dirichlet hyper-parameters. K is the number of topics, and V is the vocabulary size. $\{m'_v\}$ is the set of must-links sampled for each word v following Phase 1 a) and b), which is recorded during the iterations.

Phase 2 (Step 5 in M-GPU): this sampling phase deals with cannot-links. There are two sub-steps:

a) For every cannot-word (say w_c) of w_i, one instance (say q_c) of w_c from topic z_i is sampled, where z_i denotes the topic assigned to word w_i in Phase 1, based on the following conditional distribution:

$$P(q = q_c|\boldsymbol{z}, \boldsymbol{w}, \alpha) \propto \frac{n_{d_c,k} + \alpha}{\sum_{k'=1}^{K}(n_{d_c,k'} + \alpha)} \ , \tag{4.12}$$

where d_c denotes the document of the instance q_c. If there is no instance of w_c in z_i, skip step b).

b) For each drawn instance q_c from Phase 2 a), resample a topic k (not equal to z_i) based on the conditional distribution below:

$$\begin{aligned} & P(z_{q_c} = k|\boldsymbol{z}^{-q_c}, \boldsymbol{w}, \alpha, \beta, \lambda, q = q_c) \\ & \propto \boldsymbol{I}_{[0,p(w_c|k)]}(P(w_c|z_c)) \\ & \times \frac{n_{d_c,k}^{-q_c} + \alpha}{\sum_{k'=1}^{K}(n_{d_c,k'}^{-q_c} + \alpha)} \\ & \times \frac{\sum_{\{w',w_c\}\in\{m'_c\}} \lambda_{w',w_c} \times n_{k,w'}^{-q_c} + \beta}{\sum_{v=1}^{V}(\sum_{\{w',v\}\in\{m'_v\}} \lambda_{w',v} \times n_{k,w'}^{-q_c} + \beta)} \ , \end{aligned} \tag{4.13}$$

where z_c (the same as z_i sampled from 4.11) is the original topic assignment. $\{m'_c\}$ is the set of must-links sampled for word w_c. Superscript $-q_c$ denotes the counts excluding the original assignments. $\boldsymbol{I}()$ is an indicator function, which restricts the ball to be transferred only to an urn that contains a higher proportion of word w_c. If there is no topic k that has a higher proportion of w_c than z_c, then keep the original topic assignment, i.e., assign z_c to w_c.

Mapping to Components in the LML Architecture in Section 1.3

Like LTM, The knowledge base of AMC is the **knowledge base** of LML in Section 1.3. It stores only the past topics or p-topics from previous tasks. **Knowledge-based learner** is the *knowledge-based topic model* in Figure 4.2. The issues of correctness and applicability of previous knowledge are dealt with explicitly using multiple minimum supports frequent itemset mining (MS-FIM), the cannot-link miner, the must-link graph, and Pointwise Mutual Information (PMI). They are further considered implicitly by the M-GPU model.

4.4 LIFELONG INFORMATION EXTRACTION

Information extraction (IE) is a rich ground for applying lifelong machine learning (LML) because the earlier extraction results are often very useful to later extraction and learning. NELL is a well-known representative system in this area Carlson et al. [2010a], Mitchell et al. [2015]. However, NELL is a semi-supervised learning system, it is thus not covered here. We study it in Chapter 5. This section introduces an application of LML to a specific unsupervised IE task based on the work in [Liu et al., 2016]. The IE task is aspect or opinion target extraction from opinion documents, which is a fundamental task of sentiment analysis (Liu 2012). It aims to extract opinion targets from opinion text. For example, from the sentence "*This phone has a great screen, but its battery life is short*," it should extract "screen" and "battery life." In product reviews, aspects are product attributes or features.

In [Liu et al., 2016], a syntactic dependency-based method called *double propagation* (DP) [Qiu et al., 2011] was adopted as the base extraction method, which was augmented with the LML capability. DP is based on the fact that opinions have targets and there are often syntactic relations between sentiment or opinion words (e.g., "great" in *"the picture quality is great"*) and target aspect (e.g., "picture quality"). Due to the syntactic relations, sentiment words can be recognized by identified aspects, and aspects can be identified by known sentiment words. The extracted sentiment words and aspects are used to identify new sentiment words and new aspects, which are used again to extract more sentiment words and aspects. This bootstrapping propagation process ends when no more sentiment words or aspects can be found. The extraction rules were designed based on dependency relations among sentiment words and aspects produced by dependency parsing.

Figure 4.3 shows the dependency relations between words in "*The phone has a good screen.*" If "good" is a known sentiment word (given or extracted), "screen," a noun modified

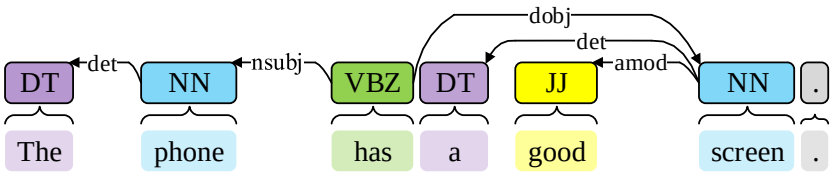


Figure 4.3: Dependency relations in the sentence "*The phone has a good screen.*"

by "good," is an aspect as they have a dependency relation *amod*. From a given seed set of sentiment words, one can extract a set of aspects from a syntactic rule like '*if a word A, whose part-of-speech (POS) is a singular noun (nn), has the dependency relation amod with (i.e., modified by) a sentiment word O, then A is an aspect.*' Similarly, one can use such rules to extract new aspects from the extracted aspects, and new sentiment words from the extracted aspects.

4.4.1 LIFELONG LEARNING THROUGH RECOMMENDATION

Although effective, syntactic rule-based methods such as DP still have room for major improvements. Liu et al. [2016] showed that incorporating lifelong learning can improve the extraction markedly.

To realize lifelong learning, Liu et al. [2016] used the idea of recommendation, in particular *collaborative filtering* [Adomavicius and Tuzhilin, 2005]. This type of recommendation uses the behavioral information of other users to recommend products/services to the current user. Following the idea, [Liu et al., 2016] used the information in reviews of a large number of other products (data of the previous tasks) to help extract aspects from reviews of the current product (data of the new task). The recommendation is based on the previous task data and extraction results. This method is called *lifelong IE through recommendations*. Two forms of recommendations were used: (1) *semantic similarity-based recommendation*, and (2) *aspect associations-based recommendation*.

1. Semantic similarity-based recommendation aims to solve the problem of missing synonymous aspects of DP using word vectors trained from a large corpus of reviews for similarity comparison. Word vectors are regarded as a form of prior or past knowledge learned from the past data. Let us see an example. Using the DP method, "picture" is extracted as an aspect from the sentence "*The picture is blurry*," but "photo" is not extracted from the sentence "*The phone is good, but not the photos.*" One reason for the inability to extract "photo" is that to ensure good extraction precision, many useful syntactic dependency rules with low precision are not used. The proposed semantic similarity-based recommendation can make use of the extracted aspect "picture" to

recommend "photo" ("photo" is a synonym of "picture") based on the semantic similarity of the word vectors of the two words.

2. The second form of recommendation is via aspect associations or correlations. This form is useful because in the first recommendation, "picture" cannot be used to recommend "battery" as an aspect because their semantic similarity value is very small. The idea of using the second form of recommendation is that many aspects are correlated or co-occur across domains. For example, those products with the aspect "picture" also have a high chance of using batteries as pictures are usually taken by digital devices that need batteries. If rules about such associations can be discovered, they can be used to recommend additional aspects. For this purpose, association rules from data mining [Liu, 2007] was employed. To mine associations, Liu et al. [2016] used the extraction results from the previous tasks stored in the knowledge base $\boldsymbol{\mathcal{S}}$.

The knowledge base contains two forms of information: the word vectors and the extraction results from the previous/past tasks.

4.4.2 AER ALGORITHM

The proposed extraction algorithm is called AER (Aspect Extraction based on Recommendations) [Liu et al., 2016] and is shown in Algorithm 5, which consists of three main steps: base extraction, aspect recommendation (which includes a knowledge learning sub-step discussed in Section 4.4.3), and knowledge base updating.

Step 1 (base extraction, lines $1-2$): Given the new document data $\mathcal{D}_{N+1}$ for the $(N+1)$th task for extraction and a set $\mathcal{O}$ of seed opinion or sentiment words, this step first uses the DP method (DPextract) to extract an initial (or base) set $\mathcal{A}^-$ of aspects using a set $\mathcal{R}^-$ of high precision rules (line 1). The set of high precision rules are selected from the set of rules in DP by evaluating their precisions individually using a development set. The set $\mathcal{A}^-$ of extracted aspects thus has very high precision but not high recall. Then, it extracts a set $\mathcal{A}^+$ of aspects from a larger set $\mathcal{R}^+$ of high recall rules ($\mathcal{R}^- \subseteq \mathcal{R}^+$) also using DPextract (line 2). The set $\mathcal{A}^+$ of extracted aspects thus has very high recall but low precision.

Step 2 (aspect recommendation, lines $3-7$): This step recommends more aspects using $\mathcal{A}^-$ as the base to improve the recall. To ensure recommendation quality, Liu et al. [2016] required that the recommended aspects must be from the set $\mathcal{A}^{diff} = \mathcal{A}^+ - \mathcal{A}^-$ (line 3). As indicated above, two forms of recommendation are performed, similarity-based using Sim-recom (line 4) and association rule-based using AR-recom (line 6). Their respective results $\mathcal{A}^s$ and $\mathcal{A}^a$ are combined with $\mathcal{A}^-$ to produce the final extraction result (line 7). Note that the word vectors $\mathcal{WV}$ required by Sim-recom is stored in the knowledge base $\boldsymbol{\mathcal{S}}$. The association rules $\mathcal{AR}$ used in AR-recom are mined from the extraction results of previous tasks also stored in the knowledge base $\boldsymbol{\mathcal{S}}$.

Step 3 (knowledge base update, line 8): This step updates the knowledge base $\mathcal{S}$, which is simple as each task is from a distinct domain in this paper. That is, the set of extracted aspects is simply added to the knowledge base $\mathcal{S}$ for future use.

We will not discuss Step 1 and step 3 further as they are fairly straightforward. Our focus is on the two recommendation methods, which will be introduced in Section 4.4.4. For the recommendations to work, we first need to learn the past knowledge in terms of word vectors $\mathcal{WV}$ and association rules $\mathcal{AR}$. We discuss knowledge learning next.

Algorithm 5 AER Algorithm

Input: New domain data $\mathcal{D}_{N+1}$, high precision aspect extraction rules $\mathcal{R}^-$, high recall aspect extraction rules $\mathcal{R}^+$, seed opinion words $\mathcal{O}$, and knowledge base $\mathcal{S}$
Output: Extracted aspect set $\mathcal{A}$

1: $\mathcal{A}^- \leftarrow$ DPextract$(\mathcal{D}_{N+1}, \mathcal{R}^-, \mathcal{O})$ // $\mathcal{A}^-$: aspect set with high precision
2: $\mathcal{A}^+ \leftarrow$ DPextract$(\mathcal{D}_{N+1}, \mathcal{R}^+, \mathcal{O})$ // $\mathcal{A}^+$: aspect set with high recall
3: $\mathcal{A}^{diff} \leftarrow \mathcal{A}^+ - \mathcal{A}^-$
4: $\mathcal{A}^s \leftarrow$ Sim-recom$(\mathcal{A}^-, \mathcal{A}^{diff}, \mathcal{WV})$ // $\mathcal{WV}$ is the set of word vectors stored in the knowledge base $\mathcal{S}$
5: $\mathcal{AR} \leftarrow$ MineAssociationRules$(\mathcal{S})$
6: $\mathcal{A}^a \leftarrow$ AR-recom$(\mathcal{A}^-, \mathcal{A}^{diff}, \mathcal{AR})$
7: $\mathcal{A} \leftarrow \mathcal{A}^- \cup \mathcal{A}^s \cup \mathcal{A}^a$
8: $\mathcal{A} \leftarrow$ UpdateKB$(\mathcal{A}, \mathcal{S})$

4.4.3 KNOWLEDGE LEARNING

Generating Word Vectors

In [Liu et al., 2016], word vectors were trained using neural networks in [Mikolov et al., 2013]. Researchers have shown that using word vectors trained this way is highly effective for the purpose of semantic similarity comparison [Mikolov et al., 2013, Turian et al., 2010]. There are several publicly available word vector resources trained from Wikipedia, Reuters news or Broadcast News for general NLP tasks such as POS tagging and Named Entity Recognition [Collobert and Weston, 2008, Huang et al., 2012, Pennington et al., 2014]. However, the initial experiments in [Liu et al., 2016] showed these word vectors were not accurate for their task. They thus trained the word vectors using a large corpus of 5.8 million reviews [Jindal and Liu, 2008]. Clearly, the word vectors can also be trained by just using the data in the past domains, but the paper did not try that. It will be interesting to see the difference in results produced with word vectors trained from the two data sources.

Mining Association Rules

Association rules are of the form, $X \longrightarrow Y$, where X and Y are disjoint sets of items, i.e., a set of aspects in our case. X and Y are called *antecedent* and *consequent* of the rule respectively. The *support* of the rule is the number of transactions that contains both X and Y divided by the total number of transactions, and the *confidence* of the rule is the number of transactions that contains both X and Y divided by the number of transactions that contains X. Given a transaction database DB, an association rule mining algorithm generates all rules that satisfies a user-specified *minimum support* and a *minimum confidence* constraints [Agrawal and Srikant, 1994]. DB contains a set of transactions. In our case, a transaction consists of all the aspects discovered from one previous domain or task, which is stored in the knowledge base $\boldsymbol{\mathcal{S}}$. Association rule mining has been well studied in data mining.

4.4.4 RECOMMENDATION USING PAST KNOWLEDGE

Recommending Aspects using Word Vectors

Algorithm 6 gives the details of Sim-recom($\mathcal{A}^-$, $\mathcal{A}^{diff}$, $\mathcal{WV}$), which recommends aspects based on aspect similarities measured using word vectors. For each term t in $\mathcal{A}^{diff}$, which can be a single word or a multi-word phrase, if the similarity between t and any term in $\mathcal{A}^-$ is at least ϵ (line 2), which means that t is very likely to be an aspect and should be recommended, then add t into $\mathcal{A}^s$ (line 3). The final recommended aspect set is $\mathcal{A}^s$.

Algorithm 6 Sim-recom Algorithm

Input: Aspect sets $\mathcal{A}^-$ and $\mathcal{A}^{diff}$, word vectors $\mathcal{WV}$
Output: Recommended aspect set $\mathcal{A}^s$

```
1: for each aspect term t ∈ A^diff do
2:    if Sim(t, A^-, WV) ≥ ε then
3:       A^s ← A^s ∪ {t}
4:    end if
5: end for
```

The function Sim(t, $\mathcal{A}^-$) in line 2 returns the maximum similarity between term t and the set of terms in $\mathcal{A}^-$, i.e.:

$$\text{Sim}(t, \mathcal{A}^-) = \max\{\text{VS}(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q}) : t_q \in \mathcal{A}^-\} \ . \tag{4.14}$$

where $\boldsymbol{\phi}_t$ is t's word vector, $\text{VS}(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q})$ is $\text{VS}^w(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q})$ if t and t_q are single words, otherwise, $\text{VS}(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q})$ is $\text{VS}^p(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q})$. $\text{VS}^w(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q})$ and $\text{VS}^p(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t_q})$ compute single words similarity and phrases or phrase-word similarity respectively. Given two terms t and t', their semantic similarity is calculated using their vectors $\boldsymbol{\phi}_t$ and $\boldsymbol{\phi}_{t'}$ in $\mathcal{WV}$ as below:

$$\text{VS}^w(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t'}) = \frac{\boldsymbol{\phi}_t^T \boldsymbol{\phi}_{t'}}{||\boldsymbol{\phi}_t|| \cdot ||\boldsymbol{\phi}_{t'}||} \quad . \tag{4.15}$$

Since there are no vectors for multi-word phrases in the pre-trained word vectors, the average cosine similarities of words in the phrases is used to evaluate phrase similarities:

$$\text{VS}^p(\boldsymbol{\phi}_t, \boldsymbol{\phi}_{t'}) = \frac{\sum_{i=1}^{L} \sum_{j=1}^{L'} \text{VS}^w(\boldsymbol{\phi}_{t_i}, \boldsymbol{\phi}_{t'_j})}{L \times L'} \quad , \tag{4.16}$$

where L is the number of single words in t, and L' is that of t'. The reason for using average similarity for multi-word phrases is that it considers the length of the phrases, and sets lower similarity value naturally if the lengths of two phrases are different.

Recommending Aspects Using Association Rules

Algorithm 7 gives the details of AR-recom, which recommends aspects based on aspect association rules. For each association rule r in $\mathcal{AR}$, if the antecedent of r is a subset of $\mathcal{A}^-$ (line 2), then recommend the terms in $\text{cons}(r) \cap \mathcal{A}^{diff}$ into the set $\mathcal{A}^a$ (line 3). The function $\text{ante}(r)$ returns the set of aspects in r's antecedent, and the function $\text{cons}(r)$ returns the set of (candidate) aspects in r's consequent.

Algorithm 7 AR-recom Algorithm

Input: Aspect sets $\mathcal{A}^-$ and $\mathcal{A}^{diff}$, association rules $\mathcal{AR}$
Output: Recommended aspect set $\mathcal{A}^a$

1: **for** each association rule $r \in \mathcal{AR}$ **do**
2: **if** $\text{ante}(r) \subseteq \mathcal{A}^-$ **then**
3: $\mathcal{A}^a \leftarrow \mathcal{A}^a \cup (\text{cons}(r) \cap \mathcal{A}^{diff})$
4: **end if**
5: **end for**

For example, one association rule in $\mathcal{AR}$ could be: $picture, display \rightarrow video, purchase$, whose antecedent contains "picture" and "display," and consequent contains "video" and "purchase." If both words "picture" and "display" are in $\mathcal{A}^-$, and only "video" is in $\mathcal{A}^{diff}$, then only "video" is added into $\mathcal{A}^a$.

Mapping to Components in the LML Architecture in Section 1.3

The **knowledge base** of Section 1.3 is the data structure in AER that stores the word vectors and the sets of extracted aspects from all previous tasks. **Knowledge-based learner** is the AER algorithm. The issues of correctness and applicability of previous knowledge

are not explicitly dealt with, but are implicitly handled to some extent by using word vectors trained from a relevant review corpus and by ensuring the recommended aspects are extracted by high recall rules from the current task domain data.

4.5 LIFELONG-RL: LIFELONG RELAXATION LABELING

Shu et al. [2016] proposed a *lifelong relaxation labeling* method called *Lifelong-RL*, which incorporates LML in the *relaxation labeling* method [Hummel and Zucker, 1983]. Lifelong-RL was applied to a sentiment analysis task. This section gives an overview of Lifelong-RL. We first introduce the relaxation labeling (RL) algorithm. We then briefly describe how to incorporate the lifelong learning capability in RL. For the application to the sentiment analysis task, please refer to the original paper.

4.5.1 RELAXATION LABELING

Relaxation Labeling (RL) is an unsupervised graph-based label propagation algorithm that works iteratively. The graph consists of nodes and edges. Each edge represents a binary relationship between two nodes. Each node n_i in the graph is associated with a multinomial distribution $P(L(n_i))$ ($L(n_i)$ being the label of n_i) on a label set Y. Each edge is associated with two conditional probability distributions $P(L(n_i)|L(n_j))$ and $P(L(n_j)|L(n_i))$, where $P(L(n_i)|L(n_j))$ represents how the label $L(n_j)$ influences the label $L(n_i)$ and vice versa. The neighbors $Ne(n_i)$ of a node n_i are associated with a weight distribution $w(n_j|n_i)$ with $\sum_{n_j \in Ne(n_i)} w(n_j|n_i) = 1$.

Given the initial values of these quantities as inputs, RL iteratively updates the label distribution of each node until convergence. Initially, we have $P^0(L(n_i))$. Let $\Delta P^{r+1}(L(n_i))$ be the change of $P(L(n_i))$ at iteration $r+1$. Given $P^r(L(n_i))$ at iteration r, $\Delta P^{r+1}(L(n_i))$ is computed by:

$$\Delta P^{r+1}(L(n_i)) = \sum_{n_j \in Ne(n_i)} \left(w(n_j|n_i) \times \sum_{y \in Y} P(L(n_i)|L(n_j) = y) \times P^r(L(n_j) = y) \right) \tag{4.17}$$

Then, the updated label distribution for iteration $r+1$, $P^{r+1}(L(n_i))$, is computed with:

$$P^{r+1}(L(n_i)) = \frac{P^r(L(n_i)) \times (1 + \Delta P^{r+1}(L(n_i)))}{\sum_{y \in Y} P^r(L(n_i) = y) \times (1 + \Delta P^{r+1}(L(n_i) = y))} \tag{4.18}$$

Once RL ends, the final label of node n_i is its highest probable label: $L(n_i) = \underset{y \in Y}{\text{argmax}}(P(L(n_i) = y))$.

Note that $P(L(n_i)|L(n_j))$ and $w(n_j|n_i)$ are not updated in each RL iteration but only $P(L(n_i))$ is. $P(L(n_i)|L(n_j))$, $w(n_j|n_i)$ and $P^0(L(n_i))$ are provided by the user or computed based on the application context. RL uses these values as input and iteratively

updates $P(L(n_i))$ based on Equations 4.17 and 4.18 until convergence. Next we discuss how to incorporate LML in RL.

4.5.2 LIFELONG RELAXATION LABELING

For LML, as usual, it is assumed that at any time step, the system has worked on N past domain data $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$. For each past domain data $\mathcal{D}_i \in \mathcal{D}^N$, the same Lifelong-RL algorithm has been applied and its result has been saved in the knowledge base (KB). Then the algorithm can borrow some useful prior/past knowledge in the KB to help RL in the new/current domain $\mathcal{D}_{N+1}$. Once the result of the current domain is produced, it is also added to the KB for future use.

We now discuss the specific types of information or knowledge that can be obtained from the previous tasks to help RL in the future, which are thus stored in the KB.

1. *Prior edges*: In many applications, the graph is not given. Instead, it has to be constructed based on the data from the new task/domain data $\mathcal{D}_{N+1}$. However, due to the limited data in $\mathcal{D}_{N+1}$, some edges between nodes that should be present are not extracted from the data. But such edges between the nodes may exist in some past domain data. Then, those edges and their associated probabilities can be borrowed.

2. *Prior labels*: Some nodes in the new task/domain may also exist in some previous tasks/domains. Their labels in the past domains are very likely to be the same as those in the current domain. Then, those prior labels can give us a better idea about the initial label probability distributions of the nodes in the current domain $\mathcal{D}_{N+1}$.

To leverage those edges and labels from the past domains, the system needs to ensure that they are likely to be correct and applicable to the current task. This is a challenge problem. Interested readers, please refer to the original paper [Shu et al., 2016].

4.6 SUMMARY AND EVALUATION DATASETS

Although lifelong supervised learning has been researched since the beginning of lifelong machine learning (LML) at around 1995. little research had been done on lifelong unsupervised learning until recently. Several papers were published in the past few years on lifelong topic modeling and lifelong information extraction. These methods all exploit the sharing of topics and concepts across tasks and domains in natural language. As discussed earlier in the chapter, NLP is quite suitable for LML precisely due to its extensive sharing of expressions, concepts, and syntactic structures across domains and tasks. We thus believe LML can have a major impact on NLP.

Here we would also like to highlight a question that people often ask about lifelong unsupervised learning. That is, when faced with a new task, can we combine all the past and the current data to form a big dataset to perform the task to achieve the same or even

better results for the new task? This combining data approach can be seen as a very simple form of LML. But clearly, the approach is not suitable for lifelong topic modeling because of three key reasons. First, with a large number of different domain datasets, there will be a huge number of topics, which makes it very difficult for the user to set the number of topics. Second, much poorer topics are likely to be the result due to the mix-up of the data from very different domains which cause wrong words to be grouped together to form incoherent topics. Thus, true topics specific to the new domain may be lost or mixed up with topics from other domains. Third, because of the fact that the new data is only a tiny portion of the big data, topics modeling will not focus on those small and domain specific topics but only on those big topics that cut cross many domains. Thus those important domain specific topics will be lost. In a similar way, other information extraction methods will have similar problems if all the data are combined into one. For example, this approach cannot solve the problems that the AER system tries to solve.

Evaluation Datasets

We now summarize the evaluation datasets used in the papers discussed in this chapter. For those publicly available datasets, we provide their URLs. Hopefully they will be useful to researchers in the field.

Chen and Liu [2014a] created a dataset containing online reviews from 50 domains or types of products, which are all electronic products. The reviews were crawled from Amazon.com. Each domain has 1,000 reviews. This dataset has been used in [Chen and Liu, 2014a, Wang et al., 2016]. This dataset also has four (4) larger review collections with 10,000 reviews in each collection. The dataset is publicly available [1]. Chen and Liu [2014b] expanded this dataset by adding another 50 domains of reviews, each of which contains reviews from a non-electronic product or domain. Some example product domains include Bike, Tent, Sandal, and Mattress. Again, each domain contains 1,000 reviews. This larger dataset is also available publicly [2]. This dataset has also been used for evaluation in [Liu et al., 2016, Shu et al., 2016]. Liu et al. [2016] and Shu et al. [2016] additionally employed two publicly available aspect-annotated review datasets. One has five review collections and the other has three review collections [Liu et al., 2015a] [3].

[1] https://www.cs.uic.edu/~zchen/downloads/ICML2014-Chen-Dataset.zip
[2] https://www.cs.uic.edu/~zchen/downloads/KDD2014-Chen-Dataset.zip
[3] http://www.cs.uic.edu/liub/FBS/sentiment-analysis.htm

CHAPTER 5

Lifelong Semi-Supervised Learning for Information Extraction

This chapter focuses on *lifelong semi-supervised learning*. It mainly covers one system, called NELL, which stands for *Never-Ending Language Learner* [Carlson et al., 2010a, Mitchell et al., 2015]. It is the only lifelong semi-supervised learning system that we are aware of. NELL is also a good example of the systems approach to lifelong machine learning (LML). It is perhaps the only live LML system that has been reading the Web to extract certain types of information (or knowledge) 24 hours a day and 7 days a week since January 2010. Although several efforts have been made by other researchers to read the Web to extract various types of knowledge to build large knowledge bases, e.g., WebKB [Craven et al., 1998], KnowItAll [Etzioni et al., 2004], and YAGO [Suchanek et al., 2007], they are not lifelong learning systems except ALICE [Banko and Etzioni, 2007]. ALICE works in an LML setting and is unsupervised. Its goal is to extract information to build a domain theory of concepts and their relationships. The extraction in ALICE is done using a set of hand-crafted lexico-syntactic patterns (e.g., "< ? grains > such as buckwheat" and "buckwheat is a < ? food >"). ALICE also has some ability to produce *general propositions* by *abstraction*, which deduces a general proposition from a set of extracted fact instances. ALICE's LML feature is realized by updating the current domain theory with new extractions and by using the output of each learning cycle to suggest the focus of subsequent learning tasks, i.e., the process is guided by earlier learned knowledge.

As a semi-supervised information extraction system, NELL has only a small number of labeled training examples for each of its learning tasks, which is far from sufficient to learn accurate extractors to extract reliable knowledge. Without reliable knowledge, lifelong learning is impossible because using wrong knowledge for future learning is highly detrimental. As we discussed several times earlier in the book, identifying the correct past knowledge is a major challenge for LML. NELL made an attempt to solve this problem by extracting different types of related knowledge using different types of data sources and by constraining the learning tasks so that the tasks can reinforce or help each other and constrain each other to ensure each of them extracts reasonably correct or robust knowledge.

5.1 NELL: A NEVER ENDING LANGUAGE LEARNER

A large part of human knowledge is learned by reading books and listening to lectures. Unfortunately, computers still cannot understand human language in order to read books to acquire knowledge systematically. The NELL system represents an effort to extract two types of knowledge from reading Web documents. Since January 2010, it has been reading the Web non-stop and has accumulated millions of facts with attached confidence weights (e.g., servedWith(tea, biscuits)), which are called *beliefs*, and are stored in a structured knowledge base. The *input* to NELL consists of the following:

1. an ontology defining a set of target categories and relations to be learned (in the form of a collection of *predicates*), a handful of seed training examples for each, and a set of constraints that couple various categories and relations (e.g., Person and Sport are mutually exclusive).

2. webpages crawled from the Web, which NELL uses to extract information.

3. occasional interactions with human trainers to correct some of the mistakes made by NELL.

With this input, the *goal* of NELL is two-fold:

1. extract facts from the webpages to populate the initial ontology. Specifically, NELL continuously extracts the following two types of information or knowledge.

   (a) *category* of a noun or noun phrase, e.g., Los Angeles is a *city*, Canada is a *country*, New York Yankees is a *baseball team*[1].

   (b) *relations* of a pair of noun phrases. For example, given the name of a university (say Stanford) and a major (say Computer Science), check whether the relation hasMajor(Stanford, Computer Science) is true.

2. learn to perform the above extraction tasks, also called the reading tasks, better than yesterday. Learning is done in a semi-supervised manner.

To achieve these objectives, NELL works iteratively in an infinite loop, i.e., hence *never-ending* or *lifelong*, like an EM algorithm. Each iteration performs two main tasks corresponding to the two objectives:

1. *Reading task*: read and extract the two types of information or knowledge from the Web to grow the knowledge base (KB) of structured facts (or beliefs). Specifically, NELL's category and relation extractors first propose extraction results as updates to the KB. The Knowledge Integrator (KI) module then records these individual

[1]Recently learned knowledge examples in NELL can be found at `http://rtw.ml.cmu.edu/rtw/`.

recommendations, makes the final decision about the confidence assigned to each potential belief after considering various consistency constraints, and then performs updates to the KB.

Because of a huge number of possible candidate beliefs and the large size of knowledge base, NELL considers only the beliefs which it has highest confidence, limiting each extractor or sub-system to propose only a limited number of new candidate beliefs for any given predicate on any given iteration. This enables NELL to operate tractably and also to be able to add millions of new beliefs over many iterations.

2. *Learning task*: learn to read better with the help of the accumulated knowledge in the updated knowledge base and coupling constraints. The evidence for improved reading is shown by the fact that the system can extract more information more accurately. Specifically, learning in NELL optimizes the accuracy of each learned function. The training examples consist of a combination of human-labeled instances (the dozen or so labeled seed examples provided for each category and relation in NELL's ontology), labeled examples contributed over time through NELLs crowdsourcing website, a set of NELL self-labeled training examples corresponding to NELL's current/updated knowledge base, and a large amount of unlabeled Web text. The last two sets of the training examples propel NELL's lifelong learning and self-improvement process over time.

   Since semi-supervised learning often gives low accuracy because of the limited number of labeled examples, NELL improves the accuracy and the quality of the extracted knowledge by coupling the simultaneous training of many extractors. These extractors extract from different data sources and are learned using different learning algorithms. The rationale is that the errors made by these extractors are uncorrelated. When multiple subsystems make uncorrelated errors, the probability that they all make the same error is much lower, which is the product of individual probabilities (considering them as independent events). These multiple extractors are linked by coupling constraints. That is, the under-constrained semi-supervised learning tasks can be made more robust by imposing constraints that arise from coupling the training of many extractors for different categories and relations. Their learning tasks are thus guided by one another's results, through the shared KB and coupling constraints.

Even with coupling constraints and sophisticated mechanisms to ensure extraction quality, errors can still be made, which may propagate, accumulate, and even multiply. NELL mitigates this problem further by interacting with some human trainer each day for about 10-15 minutes to fix some of the errors to prevent their propagation and producing poorer and poorer results subsequently.

5.2 NELL ARCHITECTURE

NELL's architecture is shown in Figure 5.1. There are four main components in NELL: data resources, knowledge base, subsystem components, and knowledge integrator.

Data Resources. Since the goal of NELL is to continuously read webpages crawled from the Web to extract knowledge, webpages are thus the data resources.

Knowledge Base. Knowledge base stores all the extracted knowledge that is expressed as beliefs. As mentioned above, two types of knowledge are stored in the knowledge base: instances of various categories and relations. A piece of knowledge can be a candidate fact or a belief. A candidate fact is extracted and proposed by the subsystem components, and may be promoted to a belief, which is decided by Knowledge Integrator.

Subsystem Components. It contains several subsystems, which are the extractors and learning components of NELL. As indicated earlier, in the reading phrase, these subsystems perform extraction and propose candidate facts to be included in the knowledge base. In the learning phrase, they learn based on their individual leaning methods with the goal of improving themselves using the current state of the knowledge base and the coupling constraints. Each subsystem is built based on different extraction methods taking input from different parts of the data resources. The four subsystems, CPL, CSEAL, CMC, and RL, are discussed in the next section.

Knowledge Integrator. Knowledge Integrator (KI) controls the condition of promoting candidate facts into beliefs. It consists of a set of hand-coded criteria. Specifically, KI decides what candidate facts are promoted to the status of beliefs. It is based on a hard-coded rule. The rule says that candidate facts with high confidence from a single source (those with posterior > 0.9) are promoted. Low-confidence candidates are promoted if they have been proposed by multiple sources. Mutual-exclusion and type-checking constraints are also used in KI. In particular, if a candidate fact does not satisfy a constraint (mutual-exclusion or type-checking) based on the existing beliefs in the KB, it is not promoted. Once a candidate fact becomes a belief, it never gets demoted.

5.3 EXTRACTORS AND LEARNING IN NELL

As we can see in Figure 5.1, there are four major subsystem components that perform extraction and learning [Carlson et al., 2010a]. We discuss them now.

- *Coupled Pattern Learner* (CPL): In the reading phrase, the extractors in the CPL subsystem extract both category and relation instances from unstructured free Web text using contextual patterns. At the beginning, they are the given seed patterns and later they are the learned and promoted patterns from previous iterations. Example category and relation extraction patterns are "mayor of X" and "X plays for Y," respectively.

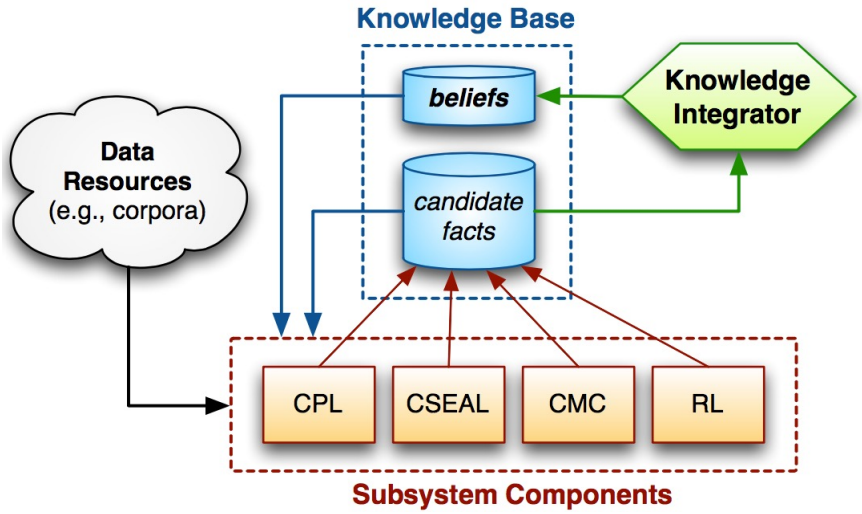


Figure 5.1: NELL system architecture [Carlson et al., 2010a].

In the learning phrase, such patterns are learned in CPL using some heuristics procedures based co-occurrence statistics between noun phrases and existing contextual patterns (both defined using part-of-speech tag sequences) for each predicate of interest. The learned patterns essentially serve as classification functions that classify noun phrases by semantic categories [Mitchell et al., 2015] (e.g., a boolean valued function that classifies whether any given noun phrase refers to a city). Relationships between predicates are used to filter out patterns that are too general.

Candidate instances and patterns extracted and learned are also filtered using mutual-exclusion and type-checking constraints to remove those possible invalid instances and patterns. Mutual-exclusion constraints enforce that mutually exclusive predicates cannot both be satisfied by the same input x. For example, x cannot be both a person and a car. Type-checking constraints are used to couple or to link relation extractors (or contextual patterns for relation extraction) with category extractors (or contextual patterns for category extraction). For example, given the relation universityHasMajor(x, y), x should be of type/category *university* and y should be of type/category *major*. Otherwise, the relation is likely to be wrong.

The remaining candidates are then ranked using simple co-occurrence statistics and estimated precisions. Only a small number of top ranked candidate instances and patterns are promoted and retained in the knowledge base for future used. Additional details about CPL can be found in [Carlson et al., 2010b].

- *Coupled SEAL* (CSEAL): CSEAL is an extraction and learning system that extracts facts from semi-structured webpages using wrapper induction. Its core system is an existing wrapper induction system called SEAL [Wang and Cohen, 2009]. SEAL is based on an semi-supervised machine learning model called *set-expansion*, also known as *learning from positive and unlabeled examples* (*PU learning*) [Liu et al., 2002]. Set expansion or PU learning is defined as follows. Given a set S of seeds of a certain target type (or positive examples), and a set of unlabeled examples U (which is obtained by querying the Web using the seeds), the goal of set-expansion is to identify examples in U that belong to S. SEAL uses *wrappers*. For a category, the wrapper is defined by character strings which specify the left context and right context of an entity to be extracted. The entities are mined from Web lists and tables of the category. For example, a wrapper <li class="player arg1"> <h4><a href="#"> indicates that *arg1* should be a player. An instance is extracted by a wrapper if it is found anywhere in the document matching with the left and right contexts of the wrapper. The relations are extracted in the same manner. However, wrappers for these predicates are learned independently in SEAL. SEAL does not have the mechanism for exploiting mutual exclusion or type-checking constraints. CSEAL added these constraints on top of SEAL so that the candidates extracted from the wrappers can be filtered out if they violate the mutual-exclusion and type-checking constraints.

  Again, the remaining candidates are ranked and only a small number of top-ranked candidate instances and patterns are promoted and retained in the knowledge base for future use. Additional details about CPL can be found in [Carlson et al., 2010b, Wang and Cohen, 2009].

- *Coupled Morphological Classifier* (CMC): CMC consists of a set of binary classifiers, one for each category, for classifying whether the extracted candidate facts/beliefs by other components/subsystems are indeed of their respective categories. To ensure high precision, the system classifies only up to 30 new beliefs per predicate in each iteration, with a minimum posterior probability of 0.75. All classifiers are built using logistic regression with L_2 regularization. The features are various morphological clues, such as words, capitalization, affixes, and parts-of-speech. The positive training examples are obtained from the beliefs in the current knowledge base, and negative examples are items inferred using mutual-exclusion constraints and the current beliefs in the knowledge base.

- *Rule Learner* (RL): RL is a first-order relational learning system similar to FOIL [Quinlan and Cameron-Jones, 1993]. Its goal is to learn probabilistic Horn clauses and to use them to infer new relations from the existing relations in the knowledge base. This reasoning capability represents an important advance of NELL that does not exist in most current information extraction or LML systems.

In Mitchell et al. [2015], several new subsystem components were also proposed, e.g., NEIL (Never Ending Image Learner), which classifies a noun phrase using its associated visual images, and OpenEval (an online information validation technique), which uses real time Web search to gather the distribution of text contexts around a noun phrase to extract instances of predicates. More information about them and others can be found in the original paper.

5.4 COUPLING CONSTRAINTS IN NELL

We have already seen two types of coupling constraints, i.e., mutual-exclusion and type-checking constraints. NELL also uses several other coupling constraints to ensure the quality or precision of its extraction and learning results. We believe that coupling constraints are an important feature and novelty of NELL, which help solve a key problem in lifelong machine learning (LML), i.e., how to ensure that the learned or extracted knowledge is correct (see Section 1.3). Without a reasonable solution to this problem, LML is difficult because errors can propagate and even multiply as the iterative process progresses. Below are three other coupling constraints that NELL uses.

- *Multi-view co-training coupling constraint*: In many cases, the same category or relation can be learned from different data sources, or *views*. For example, a predicate instance can be learned from free text by CPL and also extracted from some semi-structured webpages by CSEAL using its wrapper. This constraint requires that the two results should agree with each other. In general, for extraction or learning categories, given a noun phrase X, multiple functions that use different sets of noun phrase features (or views) to predict if X belongs to a category Y_i should give the same result. The same idea also applies to extraction or learning of relations.

- *Subset/superset coupling constraint*: When a new category is added to NELL's ontology, its parents (supersets) are also specified. For example, "Snack" is declared to be a subset of "Food." If X belongs to "Snack", then X should satisfy the constraint of being "Food." This constraint couples or links the learning tasks that extract "Snack" to those that learn to extract "Food."

- *Horn clause coupling constraint*: The probabilistic Horn clauses learned from FOIL [Quinlan and Cameron-Jones, 1993] give another set of logic based constraints. For example, X living in Chicago and Chicago being a city in USA can lead to X lives in USA (with a probability p). In general, whenever NELL learns a Horn clause rule to infer new beliefs from existing beliefs in the knowledge base, this rule serves as a coupling constraint.

5.5 SUMMARY

NELL is a good example of a lifelong semi-supervised information extraction system. This chapter gave an introduction to its key ideas, architecture, and various sub-systems and algorithms. It is by no means exhaustive. Many specific aspects of the system were not given in-depth treatments. The system is also continuously evolving and is becoming more and more powerful. What is very valuable about NELL is that it is perhaps the only non-stop or continuous learning system that extracts information from both unstructured text and semi-structure documents on the Web. We believe that more such real lifelong learning systems should be constructed to truly realize continuous learning, knowledge accumulation, and problem solving. Such systems will allow researchers to gain true insights into lifelong learning on how lifelong learning may work in practice and what the technical challenges are. These insights will help us design better and more practically useful lifelong learning systems and techniques.

Finally, we map NELL's components to the components in the general architecture of LML in Section 1.3. NELL's knowledge base is the **knowledge base** of LML in Section 1.3. It stores all extracted facts and relations, known as *beliefs*. It also has some reasoning capabilities, which other current lifelong learning algorithms/systems do not have. **Knowledge-based learner** is the set of learners and extractors discussed in Section 5.3 and Section 5.4. The issues of correctness and applicability of knowledge are mainly handled by coupling constraints and interactions with human trainers.

CHAPTER 6

Lifelong Reinforcement Learning

This chapter discusses *lifelong reinforcement learning*. Reinforcement learning is the problem where an agent learns actions through trial and error interactions with a dynamic environment [Kaelbling et al., 1996, Sutton and Barto, 1998]. In each interaction step, the agent receives input the current state of the environment. It chooses an action from a set of possible actions. The action changes the state of the environment. Then, the agent gets the value of this state transition, which can be a reward or penalty. This process repeats as the agent learns a trajectory of actions to optimize its objective, e.g., to maximize the long run sum of rewards. The goal of reinforcement learning is to learn an *optimal policy* that maps states to actions (possibly stochastically). There is a recent surge in research in reinforcement learning due to its successful use in the computer program called *AlphaGo* [Silver et al., 2016], which won 4-1 against one of the legendary professional Go players Lee Sedol in March 2016[1].

Let us see an example of a reinforcement learning setting [Tanaka and Yamamura, 1997]. This example involves an agent trying to find gold in an $N \times N$ gridworld maze. The agent can choose one action from a set of possible actions, moving left/right/up/down and picking up an item. The maze, which is the environment, may have obstacles, monsters, and gold. When the agent picks up the gold, it gets a positive reward (say $+1000$). If the agent is killed by a monster, it gets a negative reward (say -1000). When the agent steps into an obstacle, it will retreat to the previous location. The agent keeps interacting with the environment through actions and reward feedback to learn the best sequence of actions. The goal is to maximize the total reward (final reward - cost of all actions taken).

Reinforcement learning is different from supervised learning in that there is no input/output pair in reinforcement learning. In supervised learning, the manual label indicates the best output label for an input. However, in reinforcement learning, after an action is taken, the agent is *not* told which action would have been in its best long term interests. So the agent needs to gain useful experience and learn an optimal sequence of actions through interactions with the environment via feedback.

However, in order to achieve high-quality performance, the agent usually needs a large amount of quality experience. This is particularly true in high-dimensional control problems.

[1] https://deepmind.com/alpha-go

The high cost of gaining such experience is a challenging issue. In order to overcome it, *lifelong reinforcement learning* has been proposed and studied by several researchers. The motivation is to use the experience accumulated from other tasks to improve the agent's decision making in the current new task.

Lifelong reinforcement learning was first proposed by Thrun and Mitchell [1995] who worked on a lifelong robot learning problem. They showed that with knowledge memorization, the robot can learn faster while relying less on real-world experimentations. Ring [1998] proposed a continual-learning agent that aims to gradually solve complicated tasks by learning easy tasks first. Tanaka and Yamamura [1997] treated each environment as a task, and constructed an artificial neural network for each task/environment. They then used the weights of the nodes in the neural networks for existing tasks to initialize the neural network for the new task. Konidaris and Barto [2006] proposed to use approximations of prior optimal value functions for initialization in a new task. The intuition is that an agent can be trained on a sequence of relatively easy tasks to gain experience and develop a more informative measure of reward, which can then be leveraged when performing harder tasks. Wilson et al. [2007] proposed a hierarchical Bayesian lifelong reinforcement learning technique in the framework of Markov Decision Process (MDP). In particular, they added a random variable to indicate MDP classes, and assumed that the MDP tasks being assigned to the same class are similar to each other. An nonparametric infinite mixture model was proposed to take into account the unknown number of MDP classes. Fernández and Veloso [2013] proposed a policy reuse method in lifelong reinforcement learning where policies learned from prior tasks are probabilistically reused to help a new task. A nonlinear feedback policy that generalizes across multiple tasks is also used as knowledge in [Deisenroth et al., 2014]. Knowledge policy is defined as a function of both state and task, which can account for unknown states in an existing task and states in a new task. Brunskill and Li [2014] studied lifelong reinforcement learning via PAC-inspired option discovery. They showed that the learned options from previous experience can potentially accelerate learning in the new task. Bou Ammar et al. [2014] proposed a Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA) that extends ELLA [Ruvolo and Eaton, 2013b] for lifelong reinforcement learning. Along the same line, Bou Ammar et al. [2015a] proposed a cross-domain lifelong reinforcement learner based on policy gradient methods. Later, Bou Ammar et al. [2015c] added constraints to PG-ELLA for safe lifelong learning. This chapter reviews the representative techniques proposed for lifelong reinforcement learning.

6.1 LIFELONG REINFORCEMENT LEARNING THROUGH MULTIPLE ENVIRONMENTS

Tanaka and Yamamura [1997] proposed a lifelong reinforcement learning technique that treats an environment as a task. In their problem setting, there is a set of tasks, i.e., a set of environments. The tasks are independent of one anther. For example, there is a set of

mazes and each maze setting is an environment. In each of the mazes, the places of start and gold are fixed while other environment factors such as the places of obstacles and monsters, or the maze size are different. Clearly, the environments and the tasks are assumed to share some common properties.

A two-step algorithm was proposed for learning [Tanaka and Yamamura, 1997]: a) acquiring bias from previous N tasks, and b) incorporating bias into the new $(N+1)$th task. The bias here is the knowledge in the LML context to be exploited. The bias consists of two parts, initial bias and learning bias. The initial bias is used to initialize the model starting stage. The learning bias is used to influence the modeling or learning process. A neural network was used as an example model in this work. To incorporate bias, the authors applied a stochastic gradient method [Kimura, 1995] with a new update equation. The details are discussed in the subsection below.

6.1.1 ACQUIRING AND INCORPORATING BIAS

For each task/environment, a neural network is constructed. To simplify the model, the authors used a 2-layer neural network. In each task t, each neural network node (i,j) has a weight $w_{i,j}^t$. The intuition is that if the weight of a node does not change much throughout the learning process of the tasks, it can be used as an invariant node. On the other hand, if the weight of a node varies a lot, it is likely to be a task dependent node.

Based on this idea, two types of biases are acquired from the previous N tasks and they are then applied in the learning phrase of the new $(N+1)$th task.

1. *Initial bias*: In reinforcement learning, the initial random walk stage is usually very expensive. It is thus important to have a good initialization in order to improve the speed of convergence and the final performance. Initial bias is used to provide a good initial stage in order to reduce this cost. The initial weight of a node (i,j) for the $(N+1)$th task is the average weight of the same node across all the previous N tasks, i.e., $\frac{1}{N}\sum_{t=1}^{N} w_{i,j}^t$.

2. *Learning bias*: Since the stochastic gradient method [Kimura, 1995] is used, the weight of each node can have different learning rate based on their variance in the previous tasks. Following this intuition, those nodes that have varying weights in the previous tasks are more likely to be task dependent, and thus require slightly larger learning rates than those nodes with little weight changes. So for a node (i,j) in the $(N+1)$th task, its weight update is performed as follows:

$$w_{i,j}^{N+1} \leftarrow w_{i,j}^{N+1} + \alpha\beta_{i,j}(1-\gamma)\Delta w_{i,j}^{N+1}, \text{ and} \tag{6.1}$$

$$\beta_{i,j} = \epsilon(1 + \max_{t=1,\ldots,N} w_{i,j}^{N+1} - \min_{t=1,\ldots,N} w_{i,j}^{N+1}) \ . \tag{6.2}$$

Here α is the universal learning rate for all nodes. $\beta_{i,j}$ is the learning bias for each node and it controls the learning rate. ϵ is the bias parameter.

In a nutshell, the neural network for the $(N+1)$th task is initialized with *initial bias* and then updated via *learning bias* with gradient updating equations of Equations 6.1 and 6.2.

Mapping to Components in the LML Architecture in Section 1.3

The **knowledge base** of Section 1.3 is the data structure that stores all the neural networks of the previous tasks. **Knowledge-based learner** is the whole training algorithm here, which first computes the initial bias and the learning bias from previous neural networks stored in the knowledge base and then uses them to initialize the neural network for the new task and to train the network. The issues of correctness and applicability of previous knowledge are not dealt with explicitly, but are handled implicitly to some extent in the new task optimization.

6.2 HIERARCHICAL BAYESIAN LIFELONG REINFORCEMENT LEARNING

Wilson et al. [2007] worked on reinforcement learning in the framework of Markov Decision Process (MDP). The way to solve a MDP problem is to find an optimal policy that minimizes the total expected costs/penalties. Instead of working on only one MDP task in isolation, the authors considered a sequence of MDP tasks, and proposed a model called MTRL (Multi-Task Reinforcement Learning). Although the term *multi-task* is used in the name, MTRL is in fact an online multi-task learning method, which is considered as a lifelong learning method. The key idea of MTRL is the use of the hierarchical Bayesian approach to model classes of MDPs. Each class (or cluster) has some shared structure, which is regarded as the shared knowledge and is transferred to a new MDP of the class. This strong prior makes the learning of the new MDP much more efficient.

6.2.1 MOTIVATION

This work assumes that the MDP tasks are chosen randomly from a fixed but unknown distribution [Wilson et al., 2007]. As a result, the MDP tasks share some aspects that enable the knowledge extraction and transfer. To understand why the shared aspects may help more quickly learn the optimal policy for a new MDP task, let's follow the gold-finding example at the beginning of the chapter.

Each MDP task is to find gold in a maze. The maze may contain obstacles, monsters, and gold. Depending on the type of environment, certain types of rocks might be good indicators of the presence of gold while some other types of rocks may be correlated with the absence of gold. Also, some signals such as noise or smell may come from monsters nearby. If an agent learns everything from scratch, it may take a long time to learn all these rules and adjust its behaviors. However, with the observations from previous MDP tasks, the agent may learn some useful knowledge, e.g., some monsters carry a strong smell.

Using such knowledge, the agent can quickly adjust itself to avoid this type of monsters when it detects the smell. The idea is that given the knowledge from the previous MDP tasks and a small amount of experience in the new MDP task, the agent can exploit the knowledge to explore the new MDP environment much more efficiently.

6.2.2 HIERARCHICAL BAYESIAN APPROACH

Bayesian modeling was applied to tackle the problem in the paper. In the single-task scenario, a Bayesian model-based reinforcement learning computes the posterior distribution $P(M|\Theta, \mathcal{O})$ where M denotes a random variable over MDPs. $\mathcal{O}$ is the observation set and Θ is the set of model parameters. This distribution is used to help the agent choose actions. It will evolve with more actions and observations. One naïve way to extend this single-task approach to lifelong learning is to assume that all the MDP tasks are the same and treat the observations as coming from a single MDP task. Obviously, if the MDP tasks are not the same, this naïve method does not perform well.

To consider the differences between MDP tasks, Wilson et al. [2007] proposed a hierarchical Bayesian model that adds a random variable C to indicate MDP classes (or groups of similar MDPs). The assumption is that the MDP tasks within the same class assignment are similar to each other while the MDP tasks with different class assignments are very different from each other. Here, $\mathcal{M}$ denotes an MDP task and M denotes a random variable over MDPs. The sequence of MDP tasks are represented by $\mathcal{M}_1, \mathcal{M}_2, \ldots$. Instead of having the posterior distribution as $P(M|\Theta, \mathcal{O})$ in the single-task case, the posterior distribution for ith task in the hierarchical case is modeled as $P(M|\Psi, \mathcal{O}_i)$ where $\Psi = \{\Theta, \mathcal{C}\}$. Θ denotes the parameters under each class and $\mathcal{C}$ means all class assignments. $\mathcal{O}_i$ is the observation set for task $\mathcal{M}_i$. Using this posterior distribution, an approximate MDP $\hat{M}_i$ is learned by leveraging previous tasks to approximate $\mathcal{M}_i$. This addition of the class layer makes the model hierarchical. The intuition is that the knowledge in a class can be transferred to a MDP task within the same class, but not to a MDP task outside the class.

To take into account of the unknown number of MDP tasks in lifelong reinforcement learning, a nonparametric infinite mixture model was used in the class layer. In the nonparametric infinite mixture model, it is assumed that there is an infinite number of classes (or mixture components), which account for the case of seeing a new MDP task that is dissimilar to all previous ones. Specifically, Dirichlet process is applied. Dirichlet process is a stochastic process involving a base distribution G_0 and a positive scaling parameter α. The parameter α governs the probability with which the Dirichlet Process assumes a new class should be assigned. This new class is also called an *auxiliary class*. Using the above process, a Gibbs sampling process can be designed to repeatedly sample class assignments until convergence.

6.2.3 MTRL ALGORITHM

We now present the MTRL algorithm (see Algorithm 8). At the beginning, without having any MDP task, the hierarchical model parameters Ψ are initialized to uninformed values (line 1). When each new MDP task $\mathcal{M}_i$ arrives (line 2), the algorithm goes through two steps: (1) it applies the knowledge Ψ learned from the previous MDP tasks to learn an approximate MDP $\hat{M}_i$ for $\mathcal{M}_i$ (lines $4-10$) and (2) it updates the old knowledge to generate the new knowledge from $\hat{M}_1, \ldots, \hat{M}_i$ (line 12) after considering the new task.

Algorithm 8 Hierarchical Bayesian MTRL Algorithm

Input: A sequence of MDP tasks $\mathcal{M}_1, \mathcal{M}_2, \ldots$
Output: Hierarchical model parameters Ψ

1: Initialize the hierarchical model parameters Ψ
2: **for** each MDP task $\mathcal{M}_i$ from $i = 1, 2, \ldots$ **do**
3: // Step 1: apply the past knowledge for fast learning of the new MDP task $\mathcal{M}_i$
4: $\mathcal{O}_i = \varnothing$; // $\mathcal{O}_i$ is the observation set for the environment in $\mathcal{M}_i$
5: **while** policy π_i has not converged **do**
6: $\hat{M}_i \leftarrow$ SampleAnMDP($P(M|\Psi, \mathcal{O}_i)$) // $P(M|\Psi, \mathcal{O}_i)$ is the posterior distribution
7: $\pi_i =$ Solve($\hat{M}_i$) // e.g., by value iteration
8: Run π_i in $\mathcal{M}_i$ for k steps
9: $\mathcal{O}_i = \mathcal{O}_i \cup$ {observations from k steps}
10: **end while**
11: // Step 2: learn the new parameters (knowledge) from $\hat{M}_1, \ldots, \hat{M}_i$
12: $\Psi \leftarrow$ UpdateModelParameters($\Psi|\hat{M}_1, \ldots, \hat{M}_i$)
13: **end for**

For step 1, the function SampleAnMDP samples a set of MDPs based on the posterior distribution $P(M|\Psi, \mathcal{O}_i)$, where M denotes a random variable over all MDPs, and returns an MDP with the highest probability (say $\hat{M}_i$). This is how the past knowledge is used. We will explain this function in Section 6.2.5). An optimal policy π_i is then learned for $\hat{M}_i$ (line 7) using a method like value iteration [Sutton and Barto, 1998]. After π_i is obtained, it is applied for k steps in the $\mathcal{M}_i$ environment (line 8). This part is similar to Thompson sampling [Strens, 2000, Thompson, 1933, Wang et al., 2005] except that a set of MDPs are sampled first and the one with the highest probability is selected. The observations gathered from the k steps are added into the observation set $\mathcal{O}_i$, which changes the posterior distribution $P(M|\Psi, \mathcal{O}_i)$. The system then goes to the next iteration to sample a new $\hat{M}_i$. This process is repeated until the policy π_i converges.

For step 2, line 12 learns a new set of hierarchical model parameters Ψ from $\hat{M}_1, \ldots, \hat{M}_i$, which contains the class assignment for each MDP task and the model pa-

rameters associated with each class. Note that the function UpdateModelParameters (see Section 6.2.4) can automatically decide the number of classes, as well as the inherent class structure in the hierarchical model.

6.2.4 UPDATING HIERARCHICAL MODEL PARAMETERS

We first describe how to update the hierarchical model parameters Ψ (line 12 in Algorithm 8). Details about sampling of a MDP (line 6) will be discussed in the next subsection. Gibbs sampling is used to find the proper set of model parameters (see Algorithm 9). The techniques in Algorithm 9 can handle the situation where the base distribution G_0 is not conjugate to the component distribution. In Gibbs sampling, the Markov chain state includes Θ and $\mathcal{C}$, where $\Theta = \{\theta_1, \ldots, \theta_K\}$ (K is the number of existing classes) is the set of class parameters and $\mathcal{C} = \{C_1, \ldots, C_i\}$ is the set of class assignments. The use of auxiliary classes allows for the assignments of novel or new classes. m is the number of such auxiliary classes and is empirically set to a small value.

In Algorithm 9, the Markov chain state is initialized with the current parameters (line 3). Lines $7-10$ draw the parameters for each auxiliary class. Line $12-14$ call Algorithm 10 to sample a class assignment for each $\hat{M}_j$. Given the class assignments, a new set of class parameters are sampled (line 16). The sampling depends on the specific form of MDP distribution, and was not specified in the paper. After the burn-in period, Gibbs sampler keeps running until it converges. The final Markov state is returned to update the hierarchical model parameters (line 20).

6.2.5 SAMPLING AN MDP

Finally, we describe the function SampleAnMDP (line 6 in Algorithm 8), which samples an MDP. For accurate sampling, the agent or system needs to have an accurate hierarchical model. Then it should update its model parameters Ψ (knowledge) whenever a new observation is available. However, this is computationally expensive for lifelong learning considering that the number of observations and the number of MDP tasks can both be large. Instead, Wilson et al. [2007] proposed to keep the parameters Ψ fixed when learning a new MDP. That's why line 12 in Algorithm 8 is outside of the while loop (lines $5-10$). Note that Ψ includes the class assignments $\mathcal{C}$ and class parameters Θ, which together is called an informed prior, and they remain fixed during the exploration of a new MDP.

The process of generating an MDP is such that a class c is sampled first and the MDP $\hat{M}_i$ is sampled afterward based on the class. The class is sampled with the help of Algorithm 10. Here is how the past knowledge is used to help future learning. That is, if c belongs to a known class $c \in \{1, ..., K\}$, then the information in θ_c is used as the prior knowledge for exploration (see below). Otherwise, the agent uses a new class and sample the class parameters θ_c from the prior G_0 (no past knowledge is used).

Algorithm 9 Update Hierarchical Model Parameters.

Input: Model estimates $\{\hat{M}_1, \ldots, \hat{M}_i\}$ for MDP tasks $\{\mathcal{M}_1, \ldots, \mathcal{M}_i\}$, MDP distribution F given a class, Dirichlet Process DP(G_0, α)
Output: Updated hierarchical model parameters $\hat{\Psi}$

Let i be the total number of MDPs seen so far.
Let m be the number of auxiliary classes
Initialize the Markov chain state $(\Theta_0, \mathcal{C}_0)$
$k \leftarrow 0$
while Gibbs sampling is not converged **do**
  $K = |\Theta_k|$
  for $c = K + 1$ **to** $K + m$ **do**
    Draw θ_c from G_0
    $\Theta_k = \Theta_k \cup \{\theta_c\}$
  end for
  $\hat{\Psi} = \{\Theta_k, \mathcal{C}_k\}$
  for $j = 1$ **to** i **do**
    c_j = SamplingClassAssignment$(\hat{\Psi}, \hat{M}_j, F, K, m, G_0, \alpha)$
  end for
  Remove all classes with zero MDPs
  Θ_{k+1} = Sample$(P(\Theta_k | c_1, \ldots, c_i))$
  $\mathcal{C}_k = \{c_1, \ldots, c_i\}$
  $k \leftarrow k + 1$
end while
return $\hat{\Psi} = \{\Theta_k, \mathcal{C}_k\}$

SampleAnMDP works as follow: at the beginning, $\hat{M}_i$ is initialized by sampling from the informed prior, and C_i is initialized similarly. In subsequent iterations, after each set of observations (line 8 in Algorithm 8), the agent samples a sequence of class assignments C_i by running Algorithm 10 multiple times and picks the most probable one as the class assignment for $\hat{M}_i$. Recall that α controls how likely the returned class c is an auxiliary class (unseen class), i.e., $K+1 \leq c \leq K+m$ (line 4 of Algorithm 10). Once the class c is sampled, the agent then samples an MDP from class c using the posterior distribution $P(M_i|\theta_c, \mathcal{O}_i)$. The algorithm is generic and applicable to different forms of MDP distribution F which lead to distinct specific sampling procedures. See the original paper [Wilson et al., 2007] for additional details.

Algorithm 10 Sampling Class Assignment

Input: Hierarchical model parameters Ψ, MDP parameter estimate $\hat{M}_j$, MDP distribution F given a class, the number of existing classes K, the number of auxiliary classes m, Dirichlet Process DP(G_0, α)
Output: Class assignment C_j for $\hat{M}_j$

1: Let i be the total number of MDPs seen so far
2: Let $n_{-j,c}$ be the number of MDPs assigned to class c without considering class assignment of $\hat{M}_j$
3: Let $F_{c,j}$ denotes $F(\theta_c, \hat{M}_j)$, the probability of $\hat{M}_j$ in class c (the exact form may differ in different problems)
4: Sample and return C_j according to:

$$P(C_j = c) \propto \begin{cases} \frac{n_{-j,c}}{i-1+\alpha} F_{c,j}, & 1 \leq c \leq K \\ \frac{\alpha/m}{i-1+\alpha} F_{c,j}, & K+1 \leq c \leq K+m \end{cases}$$

Mapping to Components in the LML Architecture in Section 1.3

The **knowledge base** of Section 1.3 is data structure that stores the hierarchical model parameters Ψ learned from previous tasks and all past approximate MDPs. **Knowledge-based learner** is the MTRL algorithm. The issues of correctness and applicability of previous knowledge is not explicitly handled, but are considered implicitly when it assigns a new class to a new task if the previous classes are not appropriate.

6.3 PG-ELLA: LIFELONG POLICY GRADIENT REINFORCEMENT LEARNING

Instead of augmenting the stochastic gradient method with lifelong learning capability in Section 6.1, Bou Ammar et al. [2014] employed a policy gradient method [Sutton et al., 2000]. Specifically, Bou Ammar et al. [2014] extended a single-task policy gradient algorithm to a lifelong learning algorithm called *Policy Gradient Efficient Lifelong Learning Algorithm* (PG-ELLA). The lifelong idea in PG-ELLA is similar to that in ELLA [Ruvolo and Eaton, 2013b] (Section 3.5). In this section, we first introduce policy gradient reinforcement learning and then present the PG-ELLA algorithm. Throughout this section, we adopt the notations in [Bou Ammar et al., 2014].

6.3.1 POLICY GRADIENT REINFORCEMENT LEARNING

In reinforcement learning, an agent sequentially chooses actions to perform to maximize its expected reward or return. As mentioned earlier, such problems are typically formalized as a Markov Decision Process (MDP) $\langle \mathcal{X}, \mathcal{A}, P, R, \lambda \rangle$. $\mathcal{X} \subseteq \mathbb{R}^d$ is the set of states that is potentially infinite with d being the dimension of the environment. $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ is the set of all possible actions and d_a is the number of possible actions. $P : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ is the state transition probability function, i.e., given a state and an action, it gives the probability of the next state. $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the reward function that provides the agent feedback. $\lambda \in [0, 1)$ is the degree to which rewards are discounted over time.

At each time step h, being in the state $x_h \in \mathcal{X}$, the agent must choose an action $a_h \in \mathcal{A}$. After the action is taken, a new state $x_{h+1} \sim p(x_{h+1}|x_h, a_h)$ is transited to as given by P. At the same time, a reward $r_{h+1} = R(x_h, a_h)$ is sent to the agent as feedback. A *policy* is defined as a probability distribution over pairs of state and action, $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$. $\pi(a|x)$ indicates the probability of choosing action a given state x. The goal of reinforcement learning is to find an optimal policy π^* that maximizes the expected return for the agent. The actual sequence of state-action pairs forms a *trajectory* $\tau = [x_{0:H}, a_{0:H}]$ over a possibly infinite horizon H.

Policy gradient methods have been widely applied in solving high-dimensional reinforcement learning problems [Bou Ammar et al., 2014, Peters and Bagnell, 2011, Peters and Schaal, 2006, Sutton et al., 2000]. In a policy gradient method, the policy is represented by a parametric probability distribution $\pi_{\boldsymbol{\theta}}(a|x) = p(a|x; \boldsymbol{\theta})$ that stochastically chooses action a given state x based on a vector $\boldsymbol{\theta}$ of control parameters. The objective is to find the optimal parameters $\boldsymbol{\theta}^*$ that maximize the expected average return:

$$\mathcal{J}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathfrak{R}(\tau)d\tau \ , \tag{6.3}$$

where $\mathbb{T}$ denotes the set of all possible trajectories. The distribution over the trajectory τ is defined as:

$$p_{\boldsymbol{\theta}}(\tau) = P_0(x_0) \prod_{h=0}^{H-1} p(x_{h+1}|x_h, a_h)\pi_{\boldsymbol{\theta}}(a_h|x_h) \ . \tag{6.4}$$

Here $P_0(x_0)$ represents the probability of the initial state. The average return $\mathfrak{R}(\tau)$ is defined as:

$$\mathfrak{R}(\tau) = \frac{1}{H} \sum_{h=0}^{H-1} r_{h+1} \ . \tag{6.5}$$

Most policy gradient algorithms learn the parameters $\boldsymbol{\theta}$ by maximizing a lower bound on the expected return of $\mathcal{J}(\boldsymbol{\theta})$ (Equation 6.3). It compares the result of the current policy $\pi_{\boldsymbol{\theta}}$ and that of a new policy $\pi_{\tilde{\boldsymbol{\theta}}}$. As in [Kober and Peters, 2011], this lower bound can be obtained using Jensen's inequality and the concavity of the logarithm:

$$\begin{aligned}
\log \mathcal{J}\left(\tilde{\boldsymbol{\theta}}\right) &= \log \int_{\mathbb{T}} p_{\tilde{\boldsymbol{\theta}}}(\tau)\mathfrak{R}(\tau)d\tau \\
&= \log \int_{\mathbb{T}} \frac{p_{\boldsymbol{\theta}}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} p_{\tilde{\boldsymbol{\theta}}}(\tau)\mathfrak{R}(\tau)d\tau \\
&\geq \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathfrak{R}(\tau) \log \frac{p_{\tilde{\boldsymbol{\theta}}}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} d\tau \text{ (using Jensen's inequality)} \\
&= -\int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathfrak{R}(\tau) \log \frac{p_{\boldsymbol{\theta}}(\tau)}{p_{\tilde{\boldsymbol{\theta}}}(\tau)} d\tau \\
&\propto -\mathfrak{D}_{KL}\left(p_{\boldsymbol{\theta}}(\tau)\mathfrak{R}(\tau)||p_{\tilde{\boldsymbol{\theta}}}(\tau)\right) = \mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}(\tilde{\boldsymbol{\theta}}) \ ,
\end{aligned} \tag{6.6}$$

where $\mathfrak{D}_{KL}$ denotes the KL-Divergence. From the above, one can minimize the KL-Divergence between the trajectory distribution $p_{\boldsymbol{\theta}}$ of the current policy $\pi_{\boldsymbol{\theta}}$ times its reward function $\mathfrak{R}$ and the trajectory distribution $p_{\tilde{\boldsymbol{\theta}}}$ of the new policy $\pi_{\tilde{\boldsymbol{\theta}}}$.

6.3.2 POLICY GRADIENT LIFELONG LEARNING SETTING

The problem setting of policy gradient lifelong learning is similar to the problem setting of ELLA (Efficient Lifelong Learning Algorithm) (Section 3.5.1). That is, the reinforcement learning tasks arrive sequentially in a lifelong manner. Each task t is a MDP $\langle \mathcal{X}^t, \mathcal{A}^t, P^t, R^t, \lambda^t \rangle$ with the initial state distribution P_0^t. Different from the supervised learning, each reinforcement learning task does not contain labeled training data. In each task, the agent learns multiple trajectories before moving to the next task. Let N be the number of tasks encountered so far. N may be unknown to the agent. The goal is to learn a set of *optimal* policies $\{\pi^*_{\boldsymbol{\theta}^1}, \pi^*_{\boldsymbol{\theta}^2}, \ldots, \pi^*_{\boldsymbol{\theta}^N}\}$ with corresponding parameters $\{\boldsymbol{\theta}^{1*}, \boldsymbol{\theta}^{2*}, \ldots, \boldsymbol{\theta}^{N*}\}$.

6.3.3 OBJECTIVE FUNCTION AND OPTIMIZATION

Similar to ELLA, PG-ELLA also assumes that each task model's parameters $\boldsymbol{\theta}^t$ can be represented by a linear combination of a set of shared latent components $\mathbf{L}$ (shared knowledge) and a task-specific coefficient vector $\mathbf{s}^t$, i.e., $\boldsymbol{\theta}^t = \mathbf{L}\mathbf{s}^t$ [Bou Ammar et al., 2014]. In other words, PG-ELLA maintains k sparsely shared basis model components for all task models. The k basis model components are represented by $\mathbf{L} \subseteq \mathbb{R}^{d \times k}$. The task-specific vector $\mathbf{s}^t$ should be sparse in order to accommodate the differences among tasks. The objective function of PG-ELLA is as follows:

$$\frac{1}{N}\sum_{t=1}^{N} \min_{\mathbf{s}^t} \left\{ -\mathcal{J}\left(\boldsymbol{\theta}^t\right) + \mu \|\mathbf{s}^t\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2 \quad , \tag{6.7}$$

where $\|\cdot\|_1$ is the L_1 norm, which is controlled by μ as a convex approximation to the true vector sparsity. $\|\mathbf{L}\|_F^2$ is the Frobenius norm of matrix $\mathbf{L}$ and λ is the regularization coefficient for matrix $\mathbf{L}$. This objective function is closely related to Equation (3.7) in ELLA. This objective function is not jointly convex in $\mathbf{L}$ and $\mathbf{s}^t$. Thus, the alternating optimization strategy was adopted to find a local minimum, i.e., optimizing $\mathbf{L}$ while fixing $\mathbf{s}^t$ and optimizing $\mathbf{s}^t$ while fixing $\mathbf{L}$.

Combining Equations 6.6 and 6.7, we obtain the objective function below:

$$\frac{1}{N}\sum_{t=1}^{N} \min_{\mathbf{s}^t} \left\{ -\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^t\right) + \mu \|\mathbf{s}^t\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2 \quad . \tag{6.8}$$

Note the following for $\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^t\right)$:

$$\mathcal{J}_{\mathcal{L},\boldsymbol{\theta}}\left(\tilde{\boldsymbol{\theta}}^t\right) \propto -\int_{\tau \in \mathbb{T}^t} p_{\boldsymbol{\theta}^t}(\tau) \mathfrak{R}^t(\tau) \log\left(\frac{p_{\boldsymbol{\theta}^t}(\tau)\mathfrak{R}^t(\tau)}{p_{\tilde{\boldsymbol{\theta}}^t}(\tau)}\right) d\tau \quad . \tag{6.9}$$

So the objective function can be rewritten as:

$$\frac{1}{N}\sum_{t=1}^{N} \min_{\mathbf{s}^t} \left\{ \left[\int_{\tau \in \mathbb{T}^t} p_{\boldsymbol{\theta}^t}(\tau) \mathfrak{R}^t(\tau) \log\left(\frac{p_{\boldsymbol{\theta}^t}(\tau)\mathfrak{R}^t(\tau)}{p_{\tilde{\boldsymbol{\theta}}^t}(\tau)}\right) d\tau \right] + \mu \|\mathbf{s}^t\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2 \quad . \tag{6.10}$$

Again similar to ELLA, there are two major inefficiencies when solving the objective function: a) the explicit dependence of *all* available trajectories of all tasks; b) the evaluation of a single candidate $\mathbf{L}$ depends on the optimization of $\mathbf{s}^t$ for each task t. To address the first issue, the second-order Taylor approximation is used to approximate the objective function. Following the steps in Section 3.5.3, one can yield the approximate objective function below:

$$\frac{1}{N}\sum_{t=1}^{N} \min_{\mathbf{s}^t} \left\{ \|\hat{\boldsymbol{\theta}}^t - \mathbf{L}\mathbf{s}^t\|_{\boldsymbol{H}^t}^2 + \mu \|\mathbf{s}^t\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2 \quad , \tag{6.11}$$

$$\boldsymbol{H}^t = \frac{1}{2}\nabla^2_{\tilde{\boldsymbol{\theta}}^t,\tilde{\boldsymbol{\theta}}^t}\left\{\int_{\tau\in\mathbb{T}^t} p_{\boldsymbol{\theta}^t}(\tau)\mathfrak{R}^t(\tau)\log\left(\frac{p_{\boldsymbol{\theta}^t}(\tau)\mathfrak{R}^t(\tau)}{p_{\tilde{\boldsymbol{\theta}}^t}(\tau)}\right)d\tau\right\}\Bigg|_{\tilde{\boldsymbol{\theta}}^t=\hat{\boldsymbol{\theta}}^t} \text{ and}$$

$$\hat{\boldsymbol{\theta}}^t = \underset{\tilde{\boldsymbol{\theta}}^t}{\operatorname{argmin}}\left\{\int_{\tau\in\mathbb{T}^t} p_{\boldsymbol{\theta}^t}(\tau)\mathfrak{R}^t(\tau)\log\left(\frac{p_{\boldsymbol{\theta}^t}(\tau)\mathfrak{R}^t(\tau)}{p_{\tilde{\boldsymbol{\theta}}^t}(\tau)}\right)d\tau\right\} \ .$$

The second issue arises when computing the objective function for a single $\mathbf{L}$. For each single candidate $\mathbf{L}$, an optimization problem must be solved to recompute each of the $\mathbf{s}^t$'s. When the number of tasks become large, this procedure becomes very expensive. The approach to remedying this issue follows that in Section 3.5.4. When task t is encountered, only $\mathbf{s}^t$ is updated while $\mathbf{s}^{t'}$ for other task t' remains the same. Consequently, any changes to $\boldsymbol{\theta}^t$ will be transferred to other tasks only through the shared base $\mathbf{L}$. Ruvolo and Eaton [2013b] showed that this strategy does not significantly affect the quality of model fit when there are a large number of tasks. Using the previously computed values of $\mathbf{s}^t$, the following optimizing process is performed:

$$\mathbf{s}^t \leftarrow \underset{\mathbf{s}^t}{\operatorname{argmin}} \|\hat{\boldsymbol{\theta}}^t - \mathbf{L}_m\mathbf{s}^t\|^2_{\boldsymbol{H}^t} + \mu\|\mathbf{s}^t\|_1 \ , \text{ with fixed } \mathbf{L}_m, \text{and}$$

$$\mathbf{L}_{m+1} \leftarrow \underset{\mathbf{L}}{\operatorname{argmin}} \frac{1}{N}\sum_{t=1}^{N}\left(\|\hat{\boldsymbol{\theta}}^t - \mathbf{L}\mathbf{s}^t\|^2_{\boldsymbol{H}^t} + \mu\|\mathbf{s}^t\|_1\right) + \lambda\|\mathbf{L}\|^2_F, \text{ with fixed } \mathbf{s}^t \ ,$$

where $\mathbf{L}_m$ refers to the value of latent components at the mth iteration and t is assumed to be the particular task the agent is working on. Additional details can be found in [Bou Ammar et al., 2014].

Mapping to Components in the LML Architecture in Section 1.3

The **knowledge base** of Section 1.3 is the data structure of PG-ELLA that stores $\mathbf{L}$, the shared basis model components, the task specific coefficient vector $\mathbf{s}^t$ of each task, and the MDP of each task. **Knowledge-based learner** is the whole PG-ELLA algorithm. Like ELLA, the issues of correctness and applicability of previous knowledge are not explicitly handled, but are considered to some extent in the optimization process for the new task t.

6.3.4 SAFE POLICY SEARCH FOR LIFELONG LEARNING

PG-ELLA employs unconstrained optimization in learning. However, such unconstrained optimization could be fragile since the agent may learn to perform dangerous actions and cause physical damage to the agent or environment. Based on PG-ELLA, Bou Ammar et al. [2015c] proposed a safe lifelong learner for policy gradient reinforcement learning using an adversarial framework. It considered the safety constraints on each task when optimizing the overall performance. The objective function in [Bou Ammar et al., 2015c] is:

$$\min_{\mathbf{L},\mathbf{s}^t} \left[\eta^t \times l^t(\mathbf{L}\mathbf{s}^t)\right] + \mu\|\mathbf{s}^t\|_1 + \lambda\|\mathbf{L}\|_F^2 \tag{6.12}$$

$$s.t.\ \mathbf{A}^t\mathbf{L}\mathbf{s}^t \leq \mathbf{b}^t \quad \forall t \in \{1, 2, \ldots, N\}$$
$$\boldsymbol{\lambda}_{min}(\mathbf{L}\mathbf{L}^{\mathrm{T}}) \geq p \text{ and } \boldsymbol{\lambda}_{max}(\mathbf{L}\mathbf{L}^{\mathrm{T}}) \leq q \ ,$$

where constraints $\mathbf{A}^t \in \mathbb{R}^{d\times d}$ and $\mathbf{b}^t \in \mathbb{R}^d$ represent the allowed policy combinations. $\boldsymbol{\lambda}_{min}$ and $\boldsymbol{\lambda}_{max}$ are the minimum and maximum eigenvalues. p and q are bounding constraints on Frobenius norm to ensure the shared knowledge is effective and safe to use. η^t is the design weight for each task. The above objective function aims to make sure that the knowledge is safely transferred across tasks and avoid causing the agent to learn and perform irrational actions. For the method used in solving the optimization problem, please refer to [Bou Ammar et al., 2015c].

6.3.5 CROSS-DOMAIN LIFELONG REINFORCEMENT LEARNING

The works in [Bou Ammar et al., 2014, 2015c] above assume that the tasks come from a single task domain, i.e., they share a common state and action space. When the tasks have different state and/or action spaces, an *inter-task mapping* [Taylor et al., 2007] is usually needed to serve as a bridge between tasks. Taylor et al. [2007] studied transfer learning for reinforcement learning in this setting, i.e., transferring from one source domain to one target domain where the two domains have different state and action spaces. Given that an inter-task mapping is provided to an agent as input, Taylor et al. [2007] showed that the agent can learn one task and then significantly reduce the time it takes to learn another task.

Also in the transfer learning setting, Bou Ammar et al. [2015b] proposed an algorithm to automatically discover the inter-task mapping between two tasks. They focused on constructing an inter-state mapping and demonstrated the effectiveness of applying it from one task to another. The proposed method contains two steps: 1) It learns an inter-state mapping using the Unsupervised Manifold Alignment (UMA) method in [Wang and Mahadevan, 2009]. In particular, two sets of trajectories of states are collected from the source task and target task respectively. Then, each set is transformed to a state feature vector on which UMA is applied. 2) Given the learned inter-state mapping, a set of initial states in the target task is mapped into the states in the source task. Then, based on the mapped source task states, the optimal source task policy is used to produce a set of optimal state trajectories. Such optimal state trajectories are then mapped back to the target task to generate target task-specific trajectories. However, the work in [Wang and Mahadevan, 2009] does not generalize well to lifelong learning scenario as it only learns the mapping between a pair of tasks. It is computationally expensive to learn the mapping between each pair of tasks from a large pool of tasks for lifelong learning. Isele et al. [2016]

also proposed a zero-shot lifelong machine learning that models the inter-task relationship via task descriptors.

Bou Ammar et al. [2015a] proposed a more efficient method to maintain and transfer knowledge in a sequence of tasks in the lifelong setting. It is closely related to PG-ELLA [Bou Ammar et al., 2014]. The difference is that [Bou Ammar et al., 2015a] allows the tasks to come from different domains, i.e., from different state and/or action spaces. Bou Ammar et al. [2015a] assumes that all tasks can be grouped into different task groups where tasks within a task group is assumed to share a common state and action space. Formally, instead of formulating the task parameters as $\boldsymbol{\theta}^t = \mathbf{L}\mathbf{s}^t$ as in PG-ELLA (Section 6.3.3), Bou Ammar et al. [2015a] formulated them as $\boldsymbol{\theta}^t = B^{(g)}\mathbf{s}^t$ where g is the task group of $B^{(g)}$, which is the latent model components shared within g. Similar to PG-ELLA, $\mathbf{s}^t$ is assumed to be sparse to accommodate distinct tasks. Furthermore, $B^{(g)}$ is assumed to be $\Psi^{(g)}\mathbf{L}$ where $\mathbf{L}$ is the global latent model components (same as that in PG-ELLA), and $\Psi^{(g)}$ maps the shared latent components $\mathbf{L}$ into the basis for each group g of tasks. Basically, Bou Ammar et al. [2015a] added another layer, i.e., task group, to model the tasks from different domains. Theoretical guarantees were provided on the stability of the approach as the number of tasks and groups increases. Please refer to the original paper in [Bou Ammar et al., 2015a] for additional details.

6.4 SUMMARY AND EVALUATION DATASETS

This chapter introduced the existing lifelong learning work in the context of reinforcement learning. Again, the current work is not extensive. This is perhaps partly due to the fact that reinforcement learning was not as popular as traditional supervised learning in the past because of fewer real-life applications. However, reenforcement learning has come to the main stream due to the AlphaGo's success in beating the best human player in the board game of Go [Silver et al., 2016]. Although games have been the traditional application area of reinforcement learning, it has significantly more applications than just games. With the increased popularity of physical as well as software robots (such as chatbots and intelligent personal assistants) that need to interact with human beings and with other robots in real-life environments, reinforcement learning will become more and more important. Lifelong reinforcement learning will be important too because it is very hard to collect a large number of training examples in such real-life interactive environments with each individual human person or robot. The system has to learn and accumulate knowledge from all possible environments that it has experiences in to adapt itself to a new environment quick and to perform its task well.

Evaluation Datasets

Finally, to help researchers in the field, we summarize the evaluation datasets used in the papers discussed in this chapter. Tanaka and Yamamura [1997] used 9×9 mazes data in

their evaluation. Wilson et al. [2007] tested their hypotheses using a synthetic colored maze data where the task is to go from one location to another following the least cost path. PG-ELLA [Bou Ammar et al., 2014] was evaluated on three benchmark dynamic systems: Simple Mass Spring Damper [Bou Ammar et al., 2014], Cart-Pole [Bocsi et al., 2013], and Three-Link Inverted Pendulum [Bou Ammar et al., 2014]. Simple Mass Spring Damper and Cart-Pole were also used in [Bou Ammar et al., 2015c]. Other than the three dynamic systems, Quadrotor [Bouabdallah, 2007] was also used for evaluation in [Bou Ammar et al., 2015b]. Bou Ammar et al. [2015a] additionally considered Bicycle and Helicopter systems.

CHAPTER 7

Conclusion and Future Directions

This book surveyed the existing ideas and techniques of lifelong machine learning (LML). It also briefly covered closely related learning paradigms such as transfer learning and multi-task learning, and discussed their differences from LML. There have been some confusions among researchers about the differences of these learning paradigms, which are not surprising as they are indeed similar. Hopefully, our new definition of LML and subsequent discussions can clarify the differences and resolve the confusions.

Although LML was proposed in 1995, as we mentioned in the introduction chapter, the research in the field has not been extensive due to many factors, e.g., its own difficulty, lack of big data in the past, and the emphasis of statistical and algorithmic learning in the machine learning (ML) community in the past two decades. However, with the resurgence of AI and the progress and maturity of statistical machine learning algorithms, LML is becoming increasingly important because the ultimate goal of machine learning is to learn continuously and automatically in diverse domains to become more and more knowledgeable. A system is not intelligent in the general sense without the ability to learn many different types of knowledge, to accumulate it over time, and to use the knowledge to learn more and to learn better. Even if a system is extremely good at doing one difficult task, e.g., playing Go like AlphaGo or playing chess like Deep Blue, it is not an intelligent system in the general sense. Because of the physical limitations of human beings, our thinking, reasoning, and problem solving are probably not or cannot be optimized for complex tasks. A machine does not have these limitations and is bound to outperform human beings on well-defined and narrow tasks in restricted environments.

We believe that now it is time to put a significant amount of effort in the research of LML due to many reasons: First, there is a huge amount of data available now which enables a system to learn a large quantity of diverse knowledge. Without a large volume of existing knowledge, it is very difficult to learn more knowledge by leveraging the past knowledge. This is analogous to human learning. The more we know, the more and better we are able to learn. Second, statistical machine learning is becoming mature. Further improvements are becoming more and more difficult, while using the past learned knowledge to help learning is a natural way going forward, which mimics the human learning process. Existing research has shown that LML is highly effective. Third, with the increased use of intelligent personal

assistants, chatbots, and physical robots that interact with humans and other systems in real-life environments, continuous LML capabilities are becoming increasingly necessary. We expect a large amount of research will appear in the near future, which may result in major breakthroughs.

Below, we would like to highlight some challenging problems and future directions to encourage more research in LML. Their solutions can have fundamental impact on LML in specific and machine learning and AI in general.

1. **Correctness of knowledge**: How to know whether a piece of past knowledge is correct is crucial for LML. Because LML leverages the past knowledge to help future learning, incorrect past knowledge can be very harmful. In a nutshell, LML is a continuously bootstrapping learning process. Errors can propagate from previous tasks to subsequent tasks and result in more and more errors. This problem must be solved or mitigated to a great extent to ensure that LML is effective. Human beings solve this problem quite effectively. Even if mistakes are made initially, they can correct them later if new evidences are present. They can also backtrack and fix the errors along with the wrong inferences made based on the errors. An LML system should be able to do the same. Some existing LML systems have already tried to address this problem. For example, Chen and Liu [2014a] used frequent pattern mining to find those must-links (past knowledge) that appear in multiple domains and assumed those frequent must-links are more likely to be correct. They also explicitly checked the validity of the past knowledge in the modeling process. However, the existing methods are still quite rudimentary.

2. **Applicability of knowledge**: How to know whether a piece of knowledge is applicable to a new learning task is also critical for LML. Although a piece of knowledge may be correct and applicable in the context of some previous tasks, it may not be applicable to the current task due to the wrong context. Without solving this problem, LML will not be effective either. Again, the systems in [Chen and Liu, 2014a, Chen et al., 2015] proposed some preliminary mechanisms to deal with the problem in the context of topic modeling. However, the problem is far from being solved. Much research is needed. Clearly, this and the above problem are closely related.

3. **Knowledge representation and reasoning**: In the early days of AI, a significant amount of research was done on logic-based knowledge representation and reasoning. But in the past 20 years, AI research has shifted focus to statistical machine learning based on optimization. Since LML has a knowledge base (KB), knowledge representation and reasoning are naturally relevant and important. Reasoning allows the system to infer new knowledge from existing knowledge, which can be used in the new task learning. Important questions to be answered include what forms of knowledge are important, how to represent them, and what kinds of reasoning capabilities are use-

ful to LML. So far, little research has been done to address these questions in the context of LML. Knowledge in existing LML systems is mainly represented based on the direct needs of specific learning algorithms or applications. They still do not have the reasoning ability except NELL Mitchell et al. [2015], which has some reasoning capability.

4. **Learning with tasks of multiple types and/or from different domains**: About all current research of LML focuses on multiple tasks of the same type. In this case, it is easier to make use of the past knowledge. If different types of tasks are involved (e.g., entity recognition and attribute extraction), in order to transfer past knowledge from one type of tasks to another type, we need to make connections between these types of tasks. Otherwise, knowledge is hard to use across tasks. Again, the NELL system [Mitchell et al., 2015] made some attempts to do this. Ideally, this can be done automatically, but with the current technology, it is hard because the connection needs to be made via some higher-level knowledge, which needs to be learned separately.

   When the tasks are from different domains, LML is also more challenging as it is likely to need higher-level knowledge too to bridge the gap and to find the relatedness or similarity among the tasks [Bou Ammar et al., 2015a] in order to ensure knowledge applicability. In some cases, one may even need to learn from a large number of domains because each domain only contributes a tiny amount of knowledge (some domains may contribute none) that is useful to the new task [Chen and Liu, 2014a,b, Wang et al., 2016]. When multiple types of tasks from very different domains are all involved, the challenge will be even greater.

5. **Self-motivated learning**: Current ML techniques typically require human users to provide a large volume of training data (except in a few cases where the agent can learn by interacting with a simulator). If a robot is to interact with its environment and learn continuously, it needs to collect its own training data. For example, a robot sees a person that it has never seen before, it should take a video or many pictures of the person to collect positive training data. Actually, in this case, recognizing a stranger itself is already a challenge. It needs open classification [Fei et al., 2016], which most current supervised learning algorithms cannot do because they make the closed world assumption that only those classes appeared in training can appear in testing. In practice, this assumption is often violated. Open classification aims at solving this problem. Human beings do this all the time. In a more general term, self-motivated learning normally means that the robot has a sense of curiosity and is interested in exploring the unknown and learning new things by itself in the exploration process. Clearly, this is closely related to unsupervised learning and reinforcement learning. These forms of learning and, for that matter, integrated learning of all learning forms should also be made self-motivated. Note that self-motivated learning described here is

different from self-taught learning or unsupervised feature learning reported in [Raina et al., 2007]. In self-taught learning, a large amount of unlabeled data is used to learn a good feature representation of the input. The learned feature representation and a small amount of labeled data are then employed to build a classifier by applying a supervised learning method.

6. **LML for natural language learning**: Here we reiterate that NLP is perhaps one of the most suitable application areas for LML. First of all, most concepts are applicable across domains and tasks because the same words or phrases are used in different domains with the same or very similar meanings. It is unclear whether a human brain has a complex algorithm like HMM or CRF for extraction, but human beings clearly can do so well in entity recognition. We believe that one of the reasons is that when we come to extraction or recognition, we already know most of the answers because we have accumulated a great deal of entities in the past and know how to spot an entity in text from our past experiences. Second, all NLP tasks are closely related to each other as we discussed in the introduction chapter, which is obvious because they together make the meaning of a sentence. Thus, knowledge learned from one task can help learning of other tasks.

7. **Compositional learning**: Learning compositionally is likely to be very important for LML because we believe that it helps 'understanding' and enables us to manage the complexity of the real world. Complex world situations are different compositions and/or permutations of the automic situations. Classic machine learning is not compositional. For example, as humans, we learn a language by learning individual words and phrases first, and then sentences, paragraphs, and articles. This kind of learning is highly reusable. The current learning by labeling each entire sentence or even entire document with a single label is quite unnatural and hard to reuse. There are simply too many, almost infinite number of possible sentences, which makes it very difficult to learn things that do not occur frequently. For statistical machine learning to work, the data must occur sufficiently frequently in order to compute reliable statistics. However, if it is possible to learn in a bottom up fashion, from words, phrases, to sentences and whole documents, it is possible to understand those infrequent sentences because each of their component words or phrases may have appeared frequently. The syntactic structures of the sentences may have appeared frequently too. We believe that people learn compositionally. Compositional learning is especially useful for image recognition and natural language processing. For example, we not only can recognize a person as a whole, but also his/her face, head, arms, legs, torso, etc. For the head, you can recognize, eyes, mouth, nose, eyebrows, etc. Current learning algorithms do not learn compositionally. Compositional learning is likely to be very important for LML because it allows the system to share knowledge at any level of granularity.

8. **Learning in interaction with humans and systems**: It is very inefficient and perhaps even unlikely for an intelligent system to learn completely by exploring the world itself to become very intelligent. It should also learn from humans and other systems. This is the case for human learning. Much of our knowledge was taught by our teachers, parents, and other people who we have interacted with in our daily lives. Although in supervised learning, human beings can label data for an LML system, it is far from enough and quite unnatural too. It also seems that we humans seldom learn by using labeled data. Depending on the developers of the system or the user to provide knowledge to the system is also not sufficient. As different types of software and hardware robots are getting increasingly popular, these systems must be able to learn from humans and other robots who have the knowledge in a lifelong fashion. In this way, they will learn much quicker and become more knowledgeable and intelligent.

This list of directions or challenging problems is by no means exhaustive. There are many other challenges too. As an emerging field, current LML methods and systems are still primitive. The research area is a wide open field. A significant amount of research is still needed in order to make breakthroughs. Yet practical applications and intelligent systems call for this type of advanced machine learning in order to fundamentally advance the artificial intelligence research and applications. In the near future, we envisage that a number of large and complex learning systems will be built with the LML capability. Such systems with large knowledge bases will enable major progresses to be made. Without a great deal of prior knowledge already, it is difficult to learn more.

Bibliography

Gediminas Adomavicius and Alexander Tuzhilin. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *VLDB*, pages 487–499, 1994.

Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

Rie Kubota Ando and Tong Zhang. A High-performance Semi-supervised Learning Method for Text Chunking. In *ACL*, pages 1–9, 2005.

David Andrzejewski, Xiaojin Zhu, and Mark Craven. Incorporating Domain Knowledge into Topic Modeling via Dirichlet Forest Priors. In *ICML*, pages 25–32, 2009.

David Andrzejewski, Xiaojin Zhu, Mark Craven, and Benjamin Recht. A framework for incorporating general domain knowledge into latent Dirichlet allocation using first-order logic. In *IJCAI*, pages 1171–1177, 2011.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex Multi-task Feature Learning. *Machine Learning*, 73(3):243–272, 2008.

Bikramjit Banerjee and Peter Stone. General Game Learning Using Knowledge Transfer. In *IJCAI*, pages 672–677, 2007.

Michele Banko and Oren Etzioni. Strategies for Lifelong Knowledge Extraction from the Web. In *K-CAP*, pages 95–102, 2007.

Jonathan Baxter. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

Shai Ben-David and Reba Schuller. Exploiting Task Relatedness for Multiple Task Learning. In *COLT*, 2003.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends{®} in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio. Deep Learning of Representations for Unsupervised and Transfer Learning. *Unsupervised and Transfer Learning Challenges in Machine Learning*, 7, 2012.

Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative Learning for Differing Training and Test Distributions. In *ICML*, pages 81–88, 2007.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain Adaptation with Structural Correspondence Learning. In *EMNLP*, pages 120–128, 2006.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. In *ACL*, pages 440–447, 2007.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, pages 92–100, 1998.

Botond Bocsi, Lehel Csató, and Jan Peters. Alignment-based transfer learning for robot models. In *IJCNN*, pages 1–7, 2013.

Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task Gaussian Process Prediction. In *NIPS*, pages 153–160. 2008.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast Kernel Classifiers with Online and Active Learning. *The Journal of Machine Learning Research*, 6:1579–1619, 2005.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E Taylor. Online Multi-Task Learning for Policy Gradient Methods. In *ICML*, pages 1206–1214, 2014.

Haitham Bou Ammar, Eric Eaton, Jose Marcio Luna, and Paul Ruvolo. Autonomous Cross-Domain Knowledge Transfer in Lifelong Policy Gradient Reinforcement Learning. In *AAAI*, 2015a.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *AAAI*, 2015b.

Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. In *ICML*, 2015c.

Samir Bouabdallah. *Design and control of quadrotors with application to autonomous flying*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2007.

Jordan L Boyd-Graber, David M. Blei, and Xiaojin Zhu. A Topic Model for Word Sense Disambiguation. In *EMNLP-CoNLL*, pages 1024–1033, 2007.

Emma Brunskill and Lihong Li. PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *ICML*, pages 316–324, 2014.

Chris Buckley, Gerard Salton, and James Allan. The Effect of Adding Relevance Information in a Relevance Feedback Environment. In *SIGIR*, pages 292–300, 1994.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

Andrew Carlson, Justin Betteridge, and Bryan Kisiel. Toward an Architecture for Never-Ending Language Learning. In *AAAI*, pages 1306–1313, 2010a.

Andrew Carlson, Justin Betteridge, Richard C Wang, Estevam R Hruschka Jr., and Tom M Mitchell. Coupled Semi-supervised Learning for Information Extraction. In *WSDM*, pages 101–110, 2010b.

Rich Caruana. Multitask Learning. *Machine learning*, 28(1):41–75, 1997.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

Jonathan Chang, Jordan Boyd-Graber, Wang Chong, Sean Gerrish, and David M. Blei. Reading Tea Leaves: How Humans Interpret Topic Models. In *NIPS*, pages 288–296, 2009.

Jianhui Chen, Lei Tang, Jun Liu, and Jieping Ye. A convex formulation for learning shared structures from multiple tasks. In *ICML*, pages 137–144, 2009.

Jianhui Chen, Jiayu Zhou, and Jieping Ye. Integrating low-rank and group-sparse structures for robust multi-task learning. In *KDD*, pages 42–50, 2011.

Zhiyuan Chen and Bing Liu. Topic Modeling using Topics from Many Domains, Lifelong Learning and Big Data. In *ICML*, pages 703–711, 2014a.

Zhiyuan Chen and Bing Liu. Mining Topics in Documents : Standing on the Shoulders of Big Data. In *KDD*, pages 1116–1125, 2014b.

Zhiyuan Chen, Bing Liu, and M Hsu. Identifying Intention Posts in Discussion Forums. In *NAACL-HLT*, number June, pages 1041–1050, 2013a.

Zhiyuan Chen, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Discovering Coherent Topics Using General Knowledge. In *CIKM*, pages 209–218, 2013b.

Zhiyuan Chen, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Exploiting Domain Knowledge in Aspect Extraction. In *EMNLP*, pages 1655–1667, 2013c.

Zhiyuan Chen, Arjun Mukherjee, and Bing Liu. Aspect Extraction with Automated Prior Knowledge Learning. In *ACL*, pages 347–358, 2014.

Zhiyuan Chen, Nianzu Ma, and Bing Liu. Lifelong Learning for Sentiment Classification. In *ACL*, pages 750–756, 2015.

Hao Cheng, Hao Fang, and Mari Ostendorf. Open-Domain Name Error Detection using a Multi-Task RNN. In *EMNLP*, pages 737–746, 2015.

Kenneth Ward Church and Patrick Hanks. Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1):22–29, mar 1990.

Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *ICML*, pages 160–167, 2008.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Seán Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *AAAI*, pages 509–516, 1998.

Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Co-clustering Based Classification for Out-of-domain Documents. In *KDD*, pages 210–219, 2007a.

Wenyuan Dai, Gui-rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *AAAI*, 2007b.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for Transfer Learning. In *ICML*, pages 193–200, 2007c.

Hal Daume III. Frustratingly Easy Domain Adaptation. In *ACL*, pages 256–263, 2007.

Hal Daumé III. Bayesian Multitask Learning with Latent Hierarchies. In *UAI*, pages 135–142, 2009.

Marc Peter Deisenroth, Peter Englert, Jochen Peters, and Dieter Fox. Multi-task policy search for robotics. In *ICRA*, pages 3876–3881, 2014.

Chuong Do and Andrew Y Ng. Transfer learning for text classification. In *NIPS*, pages 299–306, 2005.

Mark Dredze and Koby Crammer. Online Methods for Multi-domain Learning and Adaptation. In *EMNLP*, pages 689–697, 2008.

Vladimir Eidelman, Jordan Boyd-Graber, and Philip Resnik. Topic Models for Dynamic Translation Model Adaptation. In *ACL*, pages 115–119, 2012.

Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Web-scale Information Extraction in Knowitall: (Preliminary Results). In *WWW*, pages 100–110, 2004.

Theodoros Evgeniou and Massimiliano Pontil. Regularized Multi–task Learning. In *KDD*, pages 109–117, 2004.

Geli Fei and Bing Liu. Social Media Text Classification under Negative Covariate Shift. In *EMNLP*, pages 2347–2356, 2015.

Geli Fei, Zhiyuan Chen, and Bing Liu. Review Topic Discovery with Phrases using the Pólya Urn Model. In *COLING*, pages 667–676, 2014.

Geli Fei, Shuai Wang, and Bing Liu. Learning Cumulatively to Become More Knowledgeable. In *KDD*, 2016.

Fernando Fernández and Manuela Veloso. Learning domain structure through probabilistic policy reuse in reinforcement learning. *Progress in Artificial Intelligence*, 2(1):13–27, 2013.

Eli M Gafni and Dimitri P Bertsekas. Two-metric projection methods for constrained optimization. *SIAM Journal on Control and Optimization*, 22(6):936–964, 1984.

Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge Transfer via Multiple Model Local Structure Mapping. In *KDD*, pages 283–291, 2008.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, pages 513–520, 2011.

Pinghua Gong, Jieping Ye, and Changshui Zhang. Robust Multi-task Feature Learning. In *KDD*, pages 895–903, 2012.

Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *PNAS*, 101 Suppl:5228–5235, 2004.

James J Heckman. Sample selection bias as a specification error. *Econometrica: Journal of the econometric society*, pages 153–161, 1979.

Gregor Heinrich. A Generic Approach to Topic Models. In *ECML PKDD*, pages 517 – 532, 2009.

Mark Herbster, Massimiliano Pontil, and Lisa Wainer. Online Learning over Graphs. In *ICML*, pages 305–312, 2005.

Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *NIPS*, pages 856–864, 2010.

Thomas Hofmann. Probabilistic Latent Semantic Analysis. In *UAI*, pages 289–296, 1999.

Yuening Hu, Jordan Boyd-Graber, and Brianna Satinoff. Interactive Topic Modeling. In *ACL*, pages 248–257, 2011.

Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *ACL*, pages 873–882, 2012.

Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7304–7308, 2013a.

Yan Huang, Wei Wang, Liang Wang, and Tieniu Tan. Multi-task deep neural network for multi-label learning. In *2013 IEEE International Conference on Image Processing*, pages 2897–2900, 2013b.

Robert A Hummel and Steven W Zucker. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (3):267–287, 1983.

David Isele, Mohammad Rostami, and Eric Eaton. Using Task Features for Zero-Shot Knowledge Transfer in Lifelong Learning. In *IJCAI*, 2016.

Laurent Jacob, Jean-philippe Vert, and Francis R Bach. Clustered Multi-Task Learning: A Convex Formulation. In *NIPS*, pages 745–752. 2009.

Jagadeesh Jagarlamudi, Hal Daumé III, and Raghavendra Udupa. Incorporating Lexical Priors into Topic Models. In *EACL*, pages 204–213, 2012.

Jing Jiang. A literature survey on domain adaptation of statistical classifiers. Technical report, 2008.

Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in NLP. In *ACL*, pages 264–271, 2007.

Nitin Jindal and Bing Liu. Opinion Spam and Analysis. In *WSDM*, pages 219–230, 2008.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, pages 237–285, 1996.

Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with Whom to Share in Multi-task Feature Learning. In *ICML*, pages 521–528, 2011.

Ashish Kapoor and Eric Horvitz. Principles of lifelong learning for predictive user modeling. In *User Modeling*, pages 37–46. 2009.

Hajime Kimura. Reinforcement learning by stochastic hill climbing on discounted reward. In *ICML*, pages 295–303, 1995.

Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2004.

Jens Kober and Jan Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, 84(1):171–203, 2011.

George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *ICML*, pages 489–496, 2006.

Abhishek Kumar, Hal Daum, and Hal Daume Iii. Learning Task Grouping and Overlap in Multi-task Learning. In *ICML*, pages 1383–1390, 2012.

Neil D Lawrence and John C Platt. Learning to Learn with the Informative Vector Machine. In *ICML*, 2004.

Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML*, pages 599–606, 2010.

Su-In Lee, Vassil Chatalbashev, David Vickrey, and Daphne Koller. Learning a Meta-level Prior for Feature Relevance from Multiple Related Tasks. In *ICML*, pages 489–496, 2007.

Hui Li, Xuejun Liao, and Lawrence Carin. Multi-task reinforcement learning in partially observable stochastic environments. *The Journal of Machine Learning Research*, 10: 1131–1186, 2009.

Xuejun Liao, Ya Xue, and Lawrence Carin. Logistic Regression with an Auxiliary Data Source. In *ICML*, pages 505–512, 2005.

Bing Liu. *Web Data Mining*. Springer, 2007.

Bing Liu. Sentiment Analysis and Opinion Mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.

Bing Liu. *Sentiment Analysis Mining Opinions, Sentiments, and Emotions*. Cambridge University Press, 2015.

Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. In *KDD*, pages 337–341. ACM, 1999.

Bing Liu, Wee Sun Lee, Philip S Yu, and Xiaoli Li. Partially Supervised Classification of Text Documents. In *ICML*, pages 387–394, 2002.

Qian Liu, Zhiqiang Gao, Bing Liu, and Yuanlin Zhang. Automated rule selection for aspect extraction in opinion mining. In *IJCAI*, pages 1291–1297, 2015a.

Qian Liu, Bing Liu, Yuanlin Zhang, Doo Soon Kim, and Zhiqiang Gao. Improving Opinion Aspect Extraction using Semantic Similarity and Aspect Associations. In *AAAI*, 2016.

Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *NAACL*, 2015b.

Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying Suspicious URLs: An Application of Large-scale Online Learning. In *ICML*, pages 681–688, 2009.

Hosam Mahmoud. *Polya Urn Models*. Chapman & Hall/CRC Texts in Statistical Science, 2008.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online Dictionary Learning for Sparse Coding. In *ICML*, pages 689–696, 2009.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online Learning for Matrix Factorization and Sparse Coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.

Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, and Others. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. Sparse coding for multitask and transfer learning. In *ICML*, pages 343–351, 2013.

Andrew McCallum and Kamal Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI Workshop Learning for Text Categorization*, 1998.

Neville Mehta, Sriraam Natarajan, Prasad Tadepalli, and Alan Fern. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, 73(3):289–312, 2008.

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

George A Miller. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11):39–41, 1995.

David Mimno, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing semantic coherence in topic models. In *EMNLP*, pages 262–272, 2011.

T Mitchell, W Cohen, E Hruschka, P Talukdar, J Betteridge, A Carlson, B Dalvi, M Gardner, B Kisiel, J Krishnamurthy, N Lao, K Mazaitis, T Mohamed, N Nakashole, E Platanios, A Ritter, M Samadi, B Settles, R Wang, D Wijaya, A Gupta, X Chen, A Saparov, M Greaves, and J Welling. Never-Ending Learning. In *AAAI*, 2015.

Joseph Modayil, Adam White, and Richard S Sutton. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 22(2):146–160, 2014.

Arjun Mukherjee and Bing Liu. Aspect Extraction through Semi-Supervised Modeling. In *ACL*, pages 339–348, 2012.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359, 2010.

Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *WWW*, pages 751–760, 2010.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global Vectors for Word Representation. In *EMNLP*, pages 1532–1543, 2014.

Anastasia Pentina and Christoph H Lampert. A PAC-Bayesian Bound for Lifelong Learning. In *ICML*, pages 991–999, 2014.

Jan Peters and J Andrew Bagnell. Policy gradient methods. In *Encyclopedia of Machine Learning*, pages 774–776. Springer, 2011.

Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IROS*, pages 2219–2225, 2006.

James Petterson, Alex Smola, Tibério Caetano, Wray Buntine, and Shravan Narayanamurthy. Word Features for Latent Dirichlet Allocation. In *NIPS*, pages 1921–1929, 2010.

John Platt and Others. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 10(3):61–74, 1999.

Dean A Pomerleau. *Neural network perception for mobile robot guidance*, volume 239. Springer Science & Business Media, 2012.

Guang Qiu, Bing Liu, Jiajun Bu, and Chun Chen. Opinion Word Expansion and Target Extraction through Double Propagation. *Computational Linguistics*, 37(1):9–27, 2011.

J Ross Quinlan and R Mike Cameron-Jones. FOIL: A Midterm Report. In *ECML*, pages 3–20, 1993.

Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught Learning : Transfer Learning from Unlabeled Data. In *ICML*, pages 759–766, 2007.

Leonardo Rigutini, Marco Maggini, and Bing Liu. An EM based training algorithm for cross-language text categorization. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 529–535, 2005.

Mark B Ring. CHILD: A first step towards continual learning. In *Learning to learn*, pages 261–292. 1998.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

Paul Ruvolo and Eric Eaton. Active Task Selection for Lifelong Machine Learning. In *AAAI*, pages 862–868, 2013a.

Paul Ruvolo and Eric Eaton. ELLA: An efficient lifelong learning algorithm. In *ICML*, pages 507–515, 2013b.

Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. Toward open set recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(7):1757–1772, 2013.

Mark Schmidt, Glenn Fung, and Rómer Rosales. Fast Optimization Methods for L1 Regularization: A Comparative Study and Two New Approaches. In *ECML*, pages 286–297, 2007.

Anton Schwaighofer, Volker Tresp, and Kai Yu. Learning Gaussian process kernels via hierarchical Bayes. In *NIPS*, pages 1209–1216, 2004.

Michael L. Seltzer and Jasha Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6965–6969, 2013.

Nicholas Shackel. Bertrand's Paradox and the Principle of Indifference*. *Philosophy of Science*, 74(2):150–175, 2007.

Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, 1968.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

Lei Shu, Bing Liu, Hu Xu, and Annice Kim. Separating Entities and Aspects in Opinion Targets using Lifelong Graph Labeling. In *EMNLP*, 2016.

Lei Shu, Hu Xu, and Bing Liu. Lifelong Learning CRF for Supervised Aspect Extraction. In *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL-2017, short paper)*, 2017a.

Lei Shu, Hu Xu, and Bing Liu. DOC: Deep Open Classification of Text Documents. In *Proceedings of 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP-2017, short papre))*, 2017b.

Daniel L Silver and Robert Mercer. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connection Science*, 8(2):277–294, 1996.

Daniel L Silver and Robert E Mercer. The Task Rehearsal Method of Life-Long Learning: Overcoming Impoverished Data. In *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 90–101, 2002.

Daniel L Silver and Ryan Poirier. Sequential consolidation of learned task knowledge. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 217–232, 2004.

Daniel L Silver and Ryan Poirier. Context-Sensitive MTL Networks for Machine Lifelong Learning. In *FLAIRS Conference*, pages 628–633, 2007.

Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong Machine Learning Systems: Beyond Learning Algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, pages 49–55, 2013.

Daniel L. Silver, Geoffrey Mason, and Lubna Eljabu. Consolidation Using Sweep Task Rehearsal: Overcoming the Stability-Plasticity Problem. In *Advances in Artificial Intelligence*, volume 9091, pages 307–322. 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever,

Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network. In *NIPS*, pages 895–903, 1992.

Malcolm Strens. A Bayesian framework for reinforcement learning. In *ICML*, pages 943–950, 2000.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, pages 697–706, 2007.

Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *NIPS*, pages 1433–1440, 2008.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, and Others. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NIPS*, pages 1057–1063, 2000.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

Fumihide Tanaka and Masayuki Yamamura. An approach to lifelong reinforcement learning through multiple environments. In *6th European Workshop on Learning Robots*, pages 93–99, 1997.

Matthew E Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *ICML*, pages 879–886, 2007.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Matthew E Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *The Journal of Machine Learning Research*, 8:2125–2167, 2007.

Matthew E Taylor, Nicholas K Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *ECML PKDD*, pages 488–505, 2008.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Sebastian Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers, 1996a.

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

Sebastian Thrun and Tom M Mitchell. *Lifelong robot learning*. Springer, 1995.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *ACL*, pages 384–394, 2010.

Michel F Valstar, Bihan Jiang, Marc Mehu, Maja Pantic, and Klaus Scherer. The first facial expression recognition and analysis challenge. In *IEEE International Conference on Automatic Face & Gesture Recognition*, pages 921–926. IEEE, 2011.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *ICML*, pages 1096–1103, 2008.

Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, pages 328–339, 1989.

Chang Wang and Sridhar Mahadevan. Manifold Alignment Using Procrustes Analysis. In *ICML*, pages 1120–1127, 2008.

Chang Wang and Sridhar Mahadevan. Manifold Alignment without Correspondence. In *IJCAI*, pages 1273–1278, 2009.

Richard C Wang and William W Cohen. Character-level analysis of semi-structured documents for set expansion. In *EMNLP*, pages 1503–1512, 2009.

Shuai Wang, Zhiyuan Chen, and Bing Liu. Mining Aspect-Specific Opinion using a Holistic Lifelong Topic Model. In *WWW*, 2016.

Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *ICML*, pages 956–963, 2005.

Xing Wei and W Bruce Croft. LDA-based document models for ad-hoc retrieval. In *SIGIR*, pages 178–185, 2006.

Marco Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, Learning, and Optimization*, 12, 2012.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical Bayesian approach. In *ICML*, pages 1015–1022, 2007.

Pengtao Xie, Diyi Yang, and Eric P Xing. Incorporating Word Correlation Knowledge into Topic Modeling. In *NAACL-HLT*, pages 725–734, 2015.

Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-Task Learning for Classification with Dirichlet Process Priors. *The Journal of Machine Learning Research*, 8:35–63, 2007.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, pages 3320–3328. 2014.

Kai Yu, Volker Tresp, and Anton Schwaighofer. Learning Gaussian Processes from Multiple Tasks. In *ICML*, pages 1012–1019, 2005.

Shipeng Yu, Volker Tresp, and Kai Yu. Robust Multi-task Learning with T-processes. In *ICML*, pages 1103–1110, 2007.

Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *ICML*, page 114. ACM, 2004.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial Landmark Detection by Deep Multi-task Learning. In *ECCV*, pages 94–108. 2014.

Wayne Xin Zhao, Jing Jiang, Hongfei Yan, and Xiaoming Li. Jointly Modeling Aspects and Opinions with a MaxEnt-LDA Hybrid. In *EMNLP*, pages 56–65, 2010.

George Kingsley Zipf. *Selected Papers of the Principle of Relative Frequency in Language*. Harvard University Press, 1932.

Authors' Biographies

ZHIYUAN CHEN

Zhiyuan Chen completed his Ph.D. at the University of Illinois at Chicago (UIC) under the direction of Professor Bing Liu. He joined Google in 2016. His Ph.D. thesis title was "Lifelong Machine Learning for Topic Modeling and Classification." His research interests include Machine Learning, Natural Language Processing, Text Mining, and Data Mining. He has proposed several lifelong machine learning algorithms to automatically mine information from text documents, and published more than 15 full research papers in premier conferences such as KDD, ICML, ACL, WWW, IJCAI, and AAAI. He has given threes tutorials about lifelong machine learning at IJCAI-2015, KDD-2016 and EMNLP-2016. He has served as a PC member for many prestigious natural language processing, data mining, AI, and Web research conferences. In recognizing his academic contributions, he was awarded Fifty For The Future from Illinois Technology Foundation in 2015. His homepage is: `https://www.cs.uic.edu/~zchen/`.

BING LIU

Bing Liu is a professor of Computer Science at the University of Illinois at Chicago. He received his PhD in Artificial Intelligence from the University of Edinburgh. His research interests include lifelong machine learning, sentiment analysis and opinion mining, data mining, machine learning, and natural language processing. He has published extensively in top conferences and journals in these areas, including a number of papers on lifelong machine learning. Two of his papers have received 10-year Test-of-Time awards from KDD, the premier conference of data mining and data science. He also authored three books: one on Web data mining and two on sentiment analysis. Some of his work has been widely reported in the press, including a front-page article in The New York Times. On professional services, he serves as the current Chair of ACM SIGKDD. He has served as program chairs of many leading data mining conferences including KDD, ICDM, CIKM, WSDM, SDM and PAKDD, as associate editors of leading journals such as TKDE, TWEB, and DMKD, and as area chairs of numerous natural language processing, AI, Web research, and data mining conferences. He is an ACM Fellow, an AAAI Fellow, and an IEEE Fellow.