CHAPTER 8

Continuous Knowledge Learning in Chatbots

This chapter discusses the emerging research topic of *lifelong interactive knowledge learning* for chatbots [Mazumder et al., 2018]. Continuous learning in an interactive environment is a key capability of human beings. One can only learn so much by being told or supervised because the world is simply too complex to be completely learned this way. In fact, we humans probably learn a great deal of our knowledge through interactions with other humans and the environment around us which constantly give us explicit and implicit feedback. This learning process is called *self-supervised* because it does not require human annotated/labeled training data. In the context of chatbots, lifelong interactive learning is critical because in order for a chatboot to be truly intelligent in human-machine conversation, it has to continually learn new knowledge in order to improve itself and to understand and to get to know each of its conversation partners. Note that we use the term chatbots to refer to all kinds of conversational agents, such as dialogue systems and question-answering systems.

Chatbots have a long history in AI and natural language processing (NLP). They became particularly and increasingly popular in the past few years due to the commercial success of some chatbots or virtual assistants such as Echo and Siri. Numerous chatbots have been developed or are under development, and many researchers are also actively working on techniques for chatbots.

Early chatbot systems were mostly built using markup languages such as AIML,[1] handcrafted conversation generation rules, and/or information retrieval techniques [Ameixa et al., 2014, Banchs and Li, 2012, Lowe et al., 2015, Serban et al., 2015]. Recent neural conversation models [Li et al., 2017b, Vinyals and Le, 2015, Xing et al., 2017] are able to perform some limited open-ended conversations. However, since they do not use explicit knowledge bases (KBs) and do not perform inference, they often suffer from generic and dull responses [Li et al., 2017a, Xing et al., 2017]. More recently, Ghazvininejad et al. [2017] and Le et al. [2016] proposed to use KBs to help generate responses for knowledge-grounded conversation. However, a major weakness of the existing chat systems is that they do learn new knowledge during conversation, i.e., their knowledge is fixed beforehand and cannot be expanded or updated during the conversation process. This seriously limits the scope of their applications. Even though some existing systems can use very large KBs, these KBs still miss a large number of facts (knowledge) [West

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

et al., 2014]. It is thus important for a chatbot to continuously learn new knowledge in the conversation process to expand its KB and to improve its conversation capability, i.e., learning on the job.

Since there is little work in this emerging area, this chapter presents only one work that aims to build a lifelong interactive knowledge learning engine for chatbots [Mazumder et al., 2018]. The goal of the engine is to learn a specific type of knowledge called *factual knowledge* during the interactive conversation process. A piece of such knowledge is called a *fact* and is represented as a triple: (s, r, t), which says that the source entity s and target entity t are linked by the relation r. For example, (*Obama*, *CitizenOf*, *USA*) means that Obama is a citizen of USA.

8.1 LILI: LIFELONG INTERACTIVE LEARNING AND INFERENCE

Mazumder et al. [2018] modeled the interactive knowledge learning as an *open-world knowledge base completion* problem, which is an extension to the traditional *knowledge base completion* (KBC) problem. KBC aims to infer new facts (knowledge) automatically from the existing facts in a given KB. It is defined as a binary classification problem. Given a query triple, (s, r, t), we predict whether the source entity s and target entity t can be linked by the relation r. Previous approaches [Bordes et al., 2011, 2013, Lao et al., 2011, 2015, Mazumder and Liu, 2017, Nickel et al., 2015] solve this problem under the *closed-world* assumption, i.e., s, r and t are all *known* to exist in the KB. This is a major weakness because it means that no new knowledge or facts may contain unknown entities or relations. Due to this limitation, KBC is not sufficient for knowledge learning in conversations because in a conversation, the user can say anything, which may contain entities and/or relations that are not already in the existing KB.

Mazumder et al. [2018] removed the closed-world assumption of KBC, and allowed all s, r, and t to be *unknown*. The new problem is called *open-world knowledge base completion* (OKBC). OKBC generalizes KBC and can naturally serves as a model for knowledge learning in conversation. In essence, OKBC is an *abstraction* of the knowledge learning and inference problem in conversation. The problem is solved in the open and interactive conversation process.

The paper claimed that in a conversation, two key types of factual information, true facts and queries, can be extracted from the user utterance. Queries are facts whose truth values need to be determined. The work does not deal with subjective information such as beliefs and opinions. The paper also does not study fact or relation extraction from natural language text (user utterances) as there has been an extensive work on these topics in natural language processing (NLP). It assumes that an extraction system is already in place.

Mazumder et al. [2018] dealt with the two types of information as follows. For a true fact, it is incorporated into the KB. Here the system needs to make sure that it is not already in the KB, which involves relation resolution and entity linking. The paper again assumes that an existing system can be used to do this. After a fact is added to the KB, it may predict that some related facts involving some existing relations in the KB may also be true. For example, if the user

says "Obama was born in USA," the system may guess that (*Obama*, *CitizenOf*, *USA*) (meaning that Obama is a citizen of USA) could also be true based on the current KB. To verify this fact, it needs to solve a KBC problem by treating (*Obama*, *CitizenOf*, *USA*) as a query. This is a KBC problem because the fact (*Obama*, *BornIn*, *USA*) extracted from the original sentence has been added to the KB. Then Obama and U.S. are in the KB. If the KBC problem is solved, it learns a new fact (Obama, CitizenOf, USA) in addition to the extracted fact (Obama, BornIn, USA). For a query fact, e.g., (Obama, BornIn, USA), extracted from a user question "Was Obama born in USA?" the system needs to solve an OKBC problem if any of "*Obama*," "*BornIn*," or "*USA*" is not already in the KB.

We can see that OKBC is the core problem of a knowledge learning engine for conversation. Thus, Mazumder et al. [2018] focused on solving the OKBC problem. It assumes that other tasks such as fact/relation extraction and resolution and inferring related facts of an extracted fact are solved by other sub-systems or existing algorithms.

The paper solves the OKBC problem by mimicking how humans acquire knowledge and perform reasoning in an interactive conversation. Whenever we humans encounter an unknown concept or relation while answering a query, we perform inference using our existing knowledge. If our knowledge does not allow us to draw a conclusion, we typically ask questions to others to acquire the related knowledge and use it in the inference. The process typically involves an *inference strategy* (a sequence of actions), which interleaves a sequence of *processing* and *interactive* actions. A processing action can be the selection of related facts, deriving inference chain, etc., that advances the inference process. An interactive action can be deciding what to ask, formulating a suitable question to ask, etc., that enable us to interact. The process helps grow the knowledge over time and the newly gained knowledge enables the system to communicate better in the future. This process is called *lifelong interactive learning and inference* (LiLi). Lifelong learning is reflected by the fact that the newly acquired facts are retained in the KB and used in inference for future queries, and that the accumulated knowledge in addition to the updated KB including past inference performances are leveraged to guide future interactions and learning. LiLi has the following capabilities:

1. ***formulating an inference strategy*** for a given query that embeds *processing* and *interactive* actions;
2. ***learning interaction behaviors*** (deciding what to ask and when to ask the user);
3. ***leveraging the acquired knowledge*** in the current and future inference process; and
4. ***performing 1, 2, and 3 in a lifelong manner*** for continuous knowledge learning.

LiLi starts with the closed-world KBC approach *path-ranking* (PR) [Gardner and Mitchell, 2015, Lao et al., 2011] and extends KBC to open-world knowledge base completion (OKBC). For a relation r, PR works by enumerating paths (except single-link path r) between entity-pairs linked by r in the KB and use them as features to train a binary classifier to predict

whether a query (s, r, t) should be in the KB. Here, a path between two entities is a sequence of relations linking them. In Mazumder et al. [2018], the latest PR method, C-PR [Mazumder and Liu, 2017], was adopted and extended to make it work in the open-world setting. C-PR enumerates paths by performing bidirectional random walks over the KB graph while leveraging the context of the source-target entity-pair. It also adopts and extends the compositional vector space model in Neelakantan et al. [2015] and Das et al. [2016] with continuous learning capability for prediction.

User: (Obama, CitizenOf, USA)? "Is Obama a citizen of USA?" [Query]
LiLi: I do not know what "**CitizenOf**" mean? Can you provide me an example? [*Ask for Clue*]
User: (David Cameron, CitizenOf, UK). "David Cameron is a citizen of UK." [SF1]
LiLi: Got it. Can you tell me how **"Obama"** and **"Honolulu"** are related? [*CLQ*]
User: (Obama, BornIn, Honolulu). "Obama was born in Honolulu." [SF2]
LiLi: Got it. Can you tell also me how **"Honolulu"** and **"Hawaii"** are related? [*MLQ*]
User: (Honolulu, CapitalOfState, Hawaii). "Honolulu is the state capital of Hawaii." [SF3]
LiLi: (Obama, CitizenOf, USA) is correct. [Answer]

Figure 8.1: An example of interactive inference and learning. Note that LiLi only works with triples. Each triple above is assumed to be extracted from the sentence after it.

Given an OKBC query (s, r, t) (e.g., "(*Obama*, *CitizenOf*, *USA*)," which means whether Obama a citizen of USA), LiLi interacts with the user (*if needed*) by *dynamically* formulating questions (see the interaction example in Figure 8.1[2]) and leverages the interactively acquired knowledge (supporting facts (SFs) in the figure) for continued inference. To do so, LiLi formulates a query-specific inference strategy and executes it. LiLi is designed in a Reinforcement Learning (RL) setting that performs sub-tasks like formulating and executing a strategy, training a prediction model for inference, and retaining the knowledge for future use. The effectiveness of LiLi was empirically verified using two *standard* real-world KBs: *Freebase*[3] and *WordNet*.[3]

8.2 BASIC IDEAS OF LILI

As explained above, OKBC naturally serves as a model for knowledge learning in conversation. The question now is how to solve the OKBC problem. The key idea in Mazumder et al. [2018] is to map OKBC to KBC through interaction with the user by asking user questions. KBC already has existing solutions, e.g., C-PR.

[2]Note that the user query and system response are represented as triples as this paper does not build a conversation system. It only builds a core knowledge acquisition engine. Also, the query may be from a user or a system, e.g., a question-answer system or a conversation system that has extracted a candidate fact and wants to verify it and add it to the KB. This paper also does not study the case that the query fact is already in the KB, which is easy to verify. Moreover, as this work focuses on knowledge learning and inference *rather than conversation modeling* it simply uses template-based question generation to model LiLi's interaction with the user.

[3]https://everest.hds.utc.fr/doku.php?id=en:smemlj12

Mapping open-world to close-world. Clearly, the closed-world model KBC cannot solve the open-world OKBC problem. For example, the KBC method C-PR cannot extract path features and learn a prediction model when any of s, r, or t is unknown. LiLi solves this problem through *interactive* knowledge acquisition. If r is *unknown*, LiLi asks the user to provide a clue (an example of r). And if s or t is *unknown*, LiLi asks the user to provide a link (relation) to connect the unknown entity with an existing entity (automatically selected) in the KB. Such a query is referred to as a *connecting link query* (CLQ). The acquired knowledge, which basically makes s, r, and t known in the KB, reduces OKBC to KBC and makes the C-PR inference task feasible.

LiLi is designed as a combination of two interconnected models.

1. An RL model that learns to formulate a query-specific inference strategy for performing the OKBC task. LiLi's strategy formulation is modeled as a Markov Decision Process (MDP) with finite state ($\mathcal{S}$) and action ($\mathcal{A}$) spaces. A state $S \in \mathcal{S}$ consists of 10 binary state variables (Table 8.1), each of which keeps track of the results of an action $a \in \mathcal{A}$ (Table 8.2) taken by LiLi and thus, records the progress made in the inference process so far. We can see actions for user interactions in Table 8.2 that can turn the OKBC problem into a KBC problem. The RL algorithm Q-learning [Watkins and Dayan, 1992] with ϵ-greedy strategy is used to learn the optimal policy for training the RL model.

2. A lifelong prediction model for predicting whether a triple should be in the KB, which is invoked by an action while executing the inference strategy, is learned for each relation as in C-PR. LiLi uses deep learning to build the model. Since a model trained on a few examples (e.g., clues acquired for unknown r) with randomly initialized weights of the neural network model often perform poorly due to underfitting, it transfers the knowledge (weights) from the past most similar (with regard to r) task in an LL manner. LiLi uses a relation-entity matrix $\mathcal{M}$ to find the past most similar task for r (discussed below). See Mazumder et al. [2018] for more details.

The framework improves its performance over time through user interaction and knowledge retention. Compared to the existing KB inference methods, LiLi overcomes the following two challenges for OKBC.

1. **Spareseness of KB.** A main issue of all PR methods like C-PR is the connectivity of the KB graph. If there is no path connecting s and t in the graph, path enumeration of C-PR gets stuck and inference becomes infeasible. In such cases, LiLi uses a template relation ("@-?-@") as the *missing link* marker to connect entity-pairs and continues feature extraction. A path containing "@-?-@" is called an *incomplete path*. Thus, the extracted feature set contains both complete (no missing link) and incomplete paths. Next, LiLi selects an incomplete path from the feature set and asks the user to provide a link for path completion. Such a query is referred to as *missing link query* (MLQ).

2. **Limitation in user knowledge.** If the user is unable to respond to MLQs or CLQs, LiLi uses a *guessing mechanism* to fill the gap. This enables LiLi to continue its inference even if the user cannot answer a system's question.

Table 8.1: State bits and their meanings.

State Bits	Name	Description
QERS	Query Entities and Relation Searched	Whether the query source (s) and target (t) entities and query relation (r) have been searched in KB or not
SEF	Source Entity Found	Whether the source entity (s) has been found in KB or not
TEF	Target Entity Found	Whether the target entity (t) has been found in KB or not
QRF	Query Relation Found	Whether the query relation (r) has been found in KB or not
CLUE	Clue Bit Set	Whether the query is a clue or not
ILO	Interaction Limit Over	Whether the interaction limit is over for the query or not
PFE	Path Feature Extracted	Whether path feature extraction has been done or not
NEFS	Non-empty Feature Set	Whether the extracted feature set is non-empty or empty
CPF	Complete Path Found	Whether the extracted path features are complete or not
INFI	Inference Invoked	Whether Inference instruction has been invoked or not

Table 8.2: Actions and their descriptions.

ID	Description
a_0	Search source (h), target (t) entities and query relation (r) in KB
a_1	Ask user to provide an example/clue for query relation r
a_2	Ask user to provide missing link for path feature completion
a_3	Ask user to provide the connecting link for augmenting a new entity with KB
a_4	Extract path features between source (s) and target (t) entities using C-PR
a_5	Store query data instance in data buffer and invoke prediction model for inference

8.3 COMPONENTS OF LILI

As LL needs to retain knowledge learned from past tasks and use it to help future learning, LiLi uses a *Knowledge Store (KS)* for knowledge retention. KS has four components:

1. **Knowledge Graph** (G): G (the KB) is initialized with base KB triples and gets expanded and updated over time with the acquired knowledge.

2. **Relation-Entity Matrix** ($\mathcal{M}$): $\mathcal{M}$ is a sparse matrix, with rows as relations and columns as entity-pairs and is used by the prediction model. Given a triple $(s, r, t) \in G$, we set $\mathcal{M}[r, (s, t)] = 1$ indicating r occurs for pair (s, t).

3. **Task Experience Store** ($\mathcal{T}$): $\mathcal{T}$ stores the predictive performance of LiLi on past learned tasks in terms of *Matthews correlation coefficient* (MCC)[4] that measures the quality of binary classification. So, for two tasks r and r' (each relation indicates a task), if $\mathcal{T}[r] > \mathcal{T}[r']$ [where $\mathcal{T}[r]$=MCC(r)], we say C-PR has learned r well compared to r'.

4. **Incomplete Feature DB** (Π_{DB}): Π_{DB} stores the frequency of an incomplete path π in the form of a tuple (r, π, e_{ij}^{π}) and is used in formulating MLQs. $\Pi_{DB}[(r, \pi, e_{ij}^{\pi})] = N$ implies LiLi has extracted incomplete path π N times involving entity-pair e_{ij}^{π} $[(e_i, e_j)]$ for query relation r.

The RL model learns even after training whenever it encounters an unseen state (in testing) and thus, gets updated over time. KS is updated continuously over time as a result of the execution of LiLi and takes part in future learning. The prediction model uses LL, where we transfer knowledge (parameter values) from the model for the past most similar task to help learn the current task. Similar tasks are identified by factorizing $\mathcal{M}$ and computing a task similarity matrix $\mathcal{M}_{sim}$. Besides LL, LiLi uses $\mathcal{T}$ to identify poorly learned past tasks and acquire more clues for them to improve its skillset over time.

LiLi also uses a stack, called *Inference Stack* ($\mathcal{IS}$) to hold query and its state information for RL. LiLi always processes stack top ($\mathcal{IS}$[top]). The clues from the user get stored in $\mathcal{IS}$ on top of the query during strategy execution and processed first. Thus, the prediction model for r is learned before performing inference on a query, transforming OKBC to a KBC problem.

8.4 A RUNNING EXAMPLE

The working of LiLi is involved. For details, please refer to Mazumder et al. [2018]. Here we provide a running example by working on the example shown in Figure 8.1. LiLi works on the example as follows: First, LiLi executes a_0 and detects that the source entity "*Obama*" and query relation "*CitizenOf*" are *unknown*. Thus, LiLi executes a_1 to acquire clue (SF1) for "*CitizenOf*" and pushes the clue (+ve example) and two generated -ve examples into $\mathcal{IS}$. Once the clues are processed and a prediction model is trained for "*CitizenOf*" by formulating separate strategies for them, LiLi becomes aware of "*CitizenOf*." Now, as the clues have already been popped from $\mathcal{IS}$, the query becomes $\mathcal{IS}[top]$ and the strategy formulation process for the query resumes. Next, LiLi asks user to provide a connecting link for "*Obama*" by performing a_3. Now, the query entities and relation being known, LiLi enumerates paths between "*Obama*" and "*USA*" by performing a_4. Let an extracted path be "$Obama - BornIn \rightarrow Honolulu - @-? - @ \rightarrow Hawaii - StateOf \rightarrow USA$" with missing link between $(Honolulu, Hawaii)$. LiLi asks

[4] https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

the user to fill the link by performing a_2 and then, extracts the complete feature "$BornIn \rightarrow CapitalOfState \rightarrow StateOf$." The feature set is then fed to the prediction model and inference is made as a result of a_5. Thus, the formulated inference strategy is: "$\langle a_0, a_1, a_3, a_4, a_2, a_5 \rangle$."

8.5 SUMMARY AND EVALUATION DATASETS

In this chapter, we discussed an initial attempt to build an engine for continuous knowledge learning in human-machine conversation. We first showed that the problem underlying the engine can be formulated as an open-world knowledge base completion (OKBC) problem. We then briefly described the lifelong interactive learning and inference (LiLi) approach to solving the OKBC problem. OKBC is a generalization of KBC (knowledge base completion). LiLi solves the OKBC problem by mapping OKBC to KBC through interacting with the user. The process is formulated as a query-specific inference strategy and modeled as and learned through RL. The resulting strategy is then executed to solve the problem which involves interacting with the user in a lifelong manner.

This work, however, is still preliminary, and has several weaknesses. First, it is not integrated with a chatbot system, and it assumes that the tasks of relation extraction, resolution, entity linking, etc., can be done by existing techniques. However, although there are many existing techniques for them, these tasks are still very challenging. Second, it is only designed for learning factual knowledge that can be expressed as triples. Many other forms of knowledge are not considered.

About evaluation datasets, three well-known KBs (1) FB15k,[5] (2) WordNet,[8] and (3) ConceptNet[6] are used in Mazumder et al. [2018]. For candidate fact extraction and conversation generation, one can learn using (1) a traditional relation extraction dataset such as the one used in TAC KBP Slot Filling challenge [Angeli et al., 2015], and (2) publicly available benchmark conversation datasets such as the Ubuntu dialogue corpus [Lowe et al., 2015], Cornell Movie—Dialogs Corpus[7] Danescu-Niculescu-Mizil and Lee [2011] and Wikipedia Talk Page Conversations Corpus[8] Danescu-Niculescu-Mizil et al. [2012], respectively.

[5] https://everest.hds.utc.fr/doku.php?id=en:smemlj12
[6] https://github.com/commonsense/conceptnet5/wiki/Downloads
[7] http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html
[8] http://www.cs.cornell.edu/~cristian/Echoes_of_power.html

CHAPTER 9

Lifelong Reinforcement Learning

This chapter discusses *lifelong reinforcement learning*. Reinforcement learning (RL) 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 on 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 RL is to learn an *optimal policy* that maps states to actions (possibly stochastically). There is a recent surge in research in RL 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] More recently, AlphaGo Zero [Silver et al., 2017][2] was designed to learn to master the game of Go from scratch without human knowledge, and it has achieved superhuman performance.

Let us see an example of a RL 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 +1,000). If the agent is killed by a monster, it gets a negative reward (say −1,000). 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).

RL is different from supervised learning in that there is no input/output pair in RL. In supervised learning, the manual label indicates the best output label for an input. However, in RL, 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.

[1] https://deepmind.com/alpha-go
[2] https://deepmind.com/blog/alphago-zero-learning-scratch/