Learning Task-Oriented Dialog with Neural Network Methods

Submitted in partial fullfillment of the requirements for
the degree of

Doctor of Philsophy
in
Department of Electrical and Computer Engineering

Bing Liu

B.Eng Electrical and Electronic Engineering, Nanyang Technological University
M.S. Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

September 2018

Acknowledgement

First and foremost I want to thank my advisor Professor Ian Lane. Since the first day I joined the research group, Ian believed in me like nobody else and gave me endless support. He gave me superb mentorship both in academia and in life throughout my Ph.D. journey at Carnegie Mellon University. Ian inspired me by his hardworking and passionate attitude. He encouraged me to focus on solving big problems and guided me with invaluable advises. I appreciate his contribution of time, ideas, and support to make my Ph.D. experience stimulating and productive. I am extremely lucky to have him as my advisor.

I want to thank my co-advisor Professor John Shen, who has always been an encouraging, supportive, and caring mentor to me. With his rich experience in both academia and industry, he guided me to see things from a perspective that is beyond my personal experience. I am thankful for the excellent example he has provided as a professor and mentor.

I would like to thank Professor Alexander Rudnicky, Dr. Dilek Hakkani-Tür, and Dr. Antoine Raux for their great help on my research and serving on my thesis committee. I still remember the days when I attended Professor Alexander Rudnicky's research group discussion in their Sphinx lunch meetings. Learning from him and other senior group members on diverse research topics in the field of spoken language processing was an extremely enriching experience for me during the early stage of my research career. I want to thank Dr. Dilek Hakkani-Tür for mentoring me and giving me advice on research during and after my summer internship. She set an example of excellence as a researcher who can see the big picture without losing sight of the details. I also want to thank Dr. Antoine Raux for the enlightening discussions on research problems and research career in industry. He set an excellent example for me in bringing spoken dialog research from the lab to the real world.

The last five years at Carnegie Mellon has been an amazing learning experience. I want to thank professor Martin Griss for bringing me to CMU and being such an inspiring mentor. I cherished every moment working with him at the beginning stage of my Ph.D. research under his enormous guidance and support. I am also very grateful to Professor Ole Mengshoel, Professor Bob Iannucci, Professor Osman Yagan, Professor Bhiksha Raj, Professor Anupam Datta, Professor Pei Zhang, Professor Patrick Tague, Professor Jia Zhang, Professor Joy Ying Zhang, and Professor Stuart Evans for their advice and help. In addition, a special thank to the academic staff: Stephanie Caruso, Nathan Snizaski, Sari Smith, Lauren Schachar, Samantha Goldstein, and Wendy Fong, who have provided me enormous support at different stages of the Ph.D. program. Many thanks to all my lab mates and friends at Carnegie Mellon. William Chan thanks for organizing the reading group and the insightful discussions on deep neural networks. Chirs Lee thanks for the discussions on linear algebra and the help on computing systems. Tiancheng Zhao thanks for all the in-depth discussions on spoken dialog system both in research and in practice. Wonkyum Lee, Jungsuk Kim, Avneesh Saluja, Srikanth Kallakuri, David Cohen, Suyoun Kim, Guan-Lin Chao, Benjamin Elizalde, Rahul Rajan, Abhinav Jauhri, Ervin Teng, Tong Yu, Ming Zeng, Chih Hu, Yingrui Zhang, Yong Zhuang, Vivian Chen, Zhou Yu, Shanghang Zhang, Zhiding Yu, Zheng Sun, Lucas Sun, Heng-Tze Cheng, Senaka Buthpitiya, Dongzhen Piao, Faisal Luqman, Le Nguyen, David Huang, Xiao Wang, Yuseung Kim, Brian Ricks, Yuan Tian, Shijia Pan, Aniruddha Basak, Jun Han, Xinlei Chen, Carlos Ruiz, Chihiro Suga, and many many others, thank you for the great research discussions and the fun we had together. Moreover, I would like to acknowledge the financial support from various organizations and funding agencies including College of Engineering at Carnegie Mellon University and Samsung Research.

In addition to the learning experience at Carnegie Mellon, I have also gained valuable research and development experience in industry during summer internships. I had the fortune to learn from Dr. Gokhan Tür, who had been a great mentor in guiding me to conduct impactful research during my internship at Google. I want to thank the group leader Dr. Larry Heck, who has always been encouraging and supportive to me. I also like to express my appreciation to Pararth Shah, Abhinav Rastogi, Ankur Bapna, Jindong Chen, Shyam Upadhyay, Izzeddin Gür, Ondřej Klejch, and many other Deep Dialogue team members for the insightful research discussions. Moreover, I want to thank my colleagues at Capio AI. Thank you Jungsuk Kim, Jamie Cheng, Shekar Pasumarthi, Sahil Kumar, Yuri de Souza, and Yuki Nishida for being such amazing teammates. Thank you Tony Chen for being a great leader and mentor. I cherished the days and nights we worked together in building up the systems at the early stage of the start-up company.

Besides my colleagues and friends in the U.S., I would also like to thank my undergraduate research advisor Professor Ping Shum. It was him who introduced me to the research world. During my three years' undergraduate research experience with Professor Shum, he gave me tremendous guidance, confidence, and support. He never stopped encouraging me to strive for the best and to explore in further depth. I also want to thank Professor Meng Joo Er, Dr. Xueping Cheng, and Dr. Jing Zhang for mentoring me and guiding me through the fundamentals of research. I would not be here finishing this thesis without their support.

Finally and most importantly, I would like to express my deepest gratitude to my families. I want to thank my dad and mon, Enfu Liu and Chenhui Lou, for their continuous and unparalleled love, encouragement, and support. I want to thank my sister Xiaowen Xu for always listening to me and giving me inspiration. It was them who always stand behind me giving me endless love and supporting me with full of their hearts. Without them, I would never have the ability and courage to chase my dream. This thesis is dedicated to them.

Abstract

Dialog system is class of intelligent system that interacts with human via natural language interfaces with a coherent structure. Based on the nature of the conversation, dialog systems are generally divided into two sub-classes, task-oriented dialog systems that are created to solve specific problems, and chit-chat systems that are designed for casual chat and entertainment. This thesis focuses on task-oriented dialog systems.

Conventional systems for task-oriented dialog are highly handcrafted, usually built with complex logic and rules. These systems typically consist of a pipeline of separately developed components for spoken language understanding, dialog state tracking, dialog policy, and response generation. Despite the recent progress in spoken language processing and dialog learning, there are still a variety of major challenges with current systems. Firstly, the handcrafted modules designed with domain specific rules inherently make it hard to extend an existing system to new domains. Moreover, modules in current system are interdependent in the processing pipeline. Updating an upper-stream module may change its output distribution which can make other down-stream modules sub-optimal. Last but not least, current systems are mostly configured and trained offline. They lack the flexibility to learn continuously via interaction with users.

In this thesis, we address the limitations of the conventional systems and propose a data-driven dialog learning framework. We design a neural network based dialog system that can robustly track dialog state, interface with knowledge bases, and incorporate structured query results into system responses to successfully complete task-oriented dialogs. The system can be optimized end-to-end with error signals backpropagating from system output to raw natural language system input. In learning such system, we propose imitation and reinforcement learning based methods for hybrid offline training and online interactive learning with human-in-the-loop. The system is enabled to continuously improve itself through the interaction with users. In addition, we address several practical concerns with interactive dialog learning. In addressing the impact of inconsistent user ratings (i.e. the rewards) for dialog policy optimization, we propose an adversarial learning method which can be used to effectively estimate the reward for a dialog. In addressing the sample efficiency issue in online interactive learning with users, we propose a method by integrating the learning experience from real and imagined interactions to improve the dialog learning efficiency. We perform the system evaluation in both simulated environments and real user evaluation settings. Empirical results show that our proposed system can robustly track dialog state over multiple dialog turns and produce reasonable system responses. The proposed interactive learning methods also lead to promising improvement on task success rate and human user ratings.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Research Problem

Dialog systems, also known as conversational agents or chatbots, are playing an increasingly important role in today's business and social life. People communicate with a dialog system in natural language form, via either textual or auditory input, for entertainment and for completing daily tasks. Based on the nature of the conversation, dialog systems can be generally divided into two categories, chit-chat systems and task-oriented dialog systems. Chit-chat systems [60, 61, 71, 77, 98] are designed to engage users and provide mental support by conducting chit-chat type of conversation. Topic of such conversation can be in a wide range. Users usually have no specific goal to complete when talking to a chit-chat system. Task-oriented dialog systems [14, 21, 20, 101], on the other hand, are designed to assist user to complete tasks. They are typically designed with processes for processing requests from users, providing related information, and taking actions. Comparing to chit-chat systems, task-oriented dialog systems usually involve retrieving information from external resources and planning over multiple dialog turns. The focus of this thesis is on task-oriented dialog systems.

Conventional task-oriented dialog systems have a series of components connected in a pipeline. User's input in natural language format (text input or speech transcriptions) is firstly passed to a spoken language understanding (SLU) component, whose job is to identify user's intent and extract useful semantic information from the user's query, two tasks that are often referred to as intent identification and slot filling. Outputs of the SLU module are then passed to the dialog state tracker (DST), where the dialog state is maintained over the past dialog turns. Dialog state in conventional systems usually includes slots and values that the user has expressed so far and the user's most recent intent. The dialog state maintained in the state tracker are then used as input to the dialog policy module, which emits a system dialog act indicating what the system should say next. Dialog act can be seen as a semantic label of an agent's action (e.g. confirming a request, asking a question, and acknowledging an utterance). The dialog act is then passed to a natural language generator to form a final natural language system response to user.

Despite the progress in spoken language processing and dialog learning over the past decades, there are still a variety of major challenges with the conventional dialog systems. Firstly, current systems are still highly handcrafted, especially in modules for dialog state tracking and dialog policy. Such handcrafted modules in a processing pipeline inherently makes it hard for a system to extend to other task domains. The domain specific logic and rules usually have to be redesigned for a new task domain. Secondly, modules in current systems are process interdependent. As a result, each component in the pipeline is ideally retrained when preceding components are updated, so that we have input similar to the training examples at run-time. This domino effect causes several issues in practice. Last but not least, current systems lack the flexibility in learning continuously through user interactions. There are attempts in optimizing dialog policy online with user feedback. However, the other system modules (e.g. SLU and DST) still require significant effort from developers for offline model update and tuning.

1.2 Thesis Statement

In this thesis, we address the research question on whether we can learn *end-to-end* task-oriented dialog system effectively through *interaction* with users.

We propose a data-driven learning framework for task-oriented dialogs that are based on domain ontology. On system architecture side, we design a neural network based task-oriented dialog model that can be optimized end-to-end towards more efficient task completion and higher user satisfaction. On system learning side, we propose imitation and reinforcement learning based methods that enable the system to effectively learn from users in an interactive manner. We further address several practical concerns in interactive dialog learning and propose methods for improving dialog learning efficiency. The thesis presents a thorough study and evaluation of the proposed framework in both simulated environments and real user evaluation settings.

1.3 Thesis Outline

In this dissertation, we address two key aspects of task-oriented dialog system: end-to-end dialog modeling and interactive dialog learning. Firstly, we study how we can best model task-oriented dialog by representing dialog context over multiple turns and injecting knowledge from external resources. We present our neural network based dialog system [92, 107] that models spoken language understanding, dialog state tracking, and dialog policy in an end-to-end manner.

Secondly, we explore how we can learn such system in an interactive manner with human-in-the-loop. We discuss how the system can continuously improve itself through different modes of interactions with users by learning from user teaching and feedback [93, 107]. We present how we can optimize the dialog agent end-to-end during interactive learning with users and discuss the benefits of performing such end-to-end system optimization.

Lastly, we investigate how we can improve the system learning efficiency during its online interaction with users. We address the user rating sparsity and consistency issue by proposing an adversarial learning method [105] in estimating the dialog reward for policy optimization. We further design a learning method [106] by integrating the learning experience from real user interactions and from the "imagined" interactions to ameliorate the sample efficiency issue in online policy optimization.

We start off by introducing the background and related works of task-oriented dialog systems in Chapter 2. An overview of the proposed interactive dialog learning framework is presented in Chapter 3. In Chapter 4, we describe the proposed neural network based models for robust spoken language understanding. In Chapter 5, we present the proposed neural dialog system architecture and show how the SLU sequence model can be extended to model end-to-end task-oriented dialog. Chapter 6 presents the proposed interactive dialog learning methods with human-in-the-loop. Chapter 7 and 8 present how we can improve interactive dialog learning efficiency with adversarial training and learning from "imagined" interactions. Finally, we conclude the dissertation and discuss future work in Chapter 9.

Chapter 2

Background and Related Work

2.1 Task-Oriented Dialog Systems

In this section we review the conventional architecture for task-oriented dialog systems. We discuss the limitations of the current systems, and give motivations for the following chapters on end-to-end dialog learning.

Figure 2.1 shows the architecture of a typical modular-based of task-oriented spoken dialog system. User interacts with the system by dictating to a voice interface. The automatic speech recognition (ASR) module transcribes the user's speech input to natural language text format. The speech transcriptions are passed to the spoken language understanding (SLU) module, where the user's intention and other key named entities are extracted. This information is then formatted as the input to the dialog state tracker (DST), which maintains the current state of the dialog. Output of state tracker is passed to the dialog policy module, which produces a dialog act based on current dialog state and the facts retrieved from external resources (such as a database or a knowledge base). The dialog act emitted by the dialog policy module serves as the input to the natural language generator (NLG), where the dialog act is translated to a natural language format system response. Finally, the text

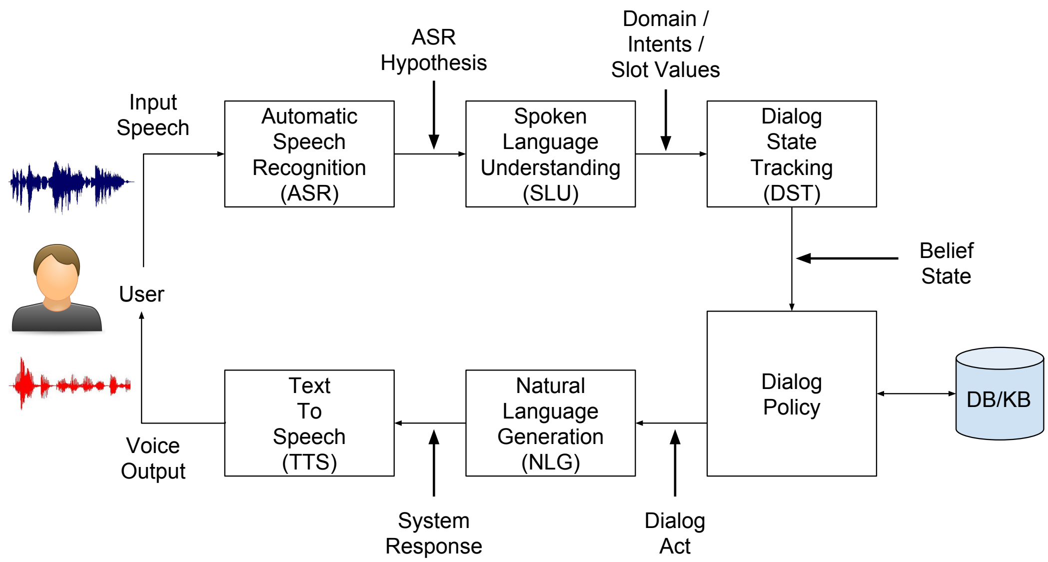

Figure 2.1: Pipeline architecture for task-oriented spoken dialog systems

to speech (TTS) module transforms the text format system response to speech and sends it back to user.

In this thesis, we propose an end-to-end dialog learning framework that models the functionality of three core components of the above described system, spoken language understanding, dialog state tracking, and dialog policy. We first review each of these system components in the following sections.

2.1.1 Spoken Language Understanding

Spoken language understanding (SLU) module is a critical component in spoken dialog systems. SLU typically involves identifying a user's intent and extracting semantic constituents from the user's natural language query, two tasks that are often referred to as intent detection and slot filling.

Intent Detection

Intent detection can be framed as a semantic utterance classification problem, and classifiers like support vector machines [12] and deep neural networks [25, 30, 28] can be applied. Given an utterance with a sequence of words $\mathbf{w} = (w_1, w_2, ..., w_T)$, the goal of intent detection is to assign an intent class c from a pre-defined set of intent classes, such that:

$$\hat{c} = \arg\max_{\mathrm{c}} P(c|\mathbf{w}) \tag{2.1}$$

Recent neural network based intent classification models involve using neural bag-of-words (NBoW), where words are mapped to high dimensional vector space and concatenated to serve as input to a classifier. More structured neural network approaches for utterance classification include using recursive neural network [41], convolutional neural network [45, 66], and recurrent neural network models [48, 81, 110, 104]. Comparing to basic NBoW methods, these models can better capture the structural patterns in the sequence of words in an utterance.

Slot Filling

Slot filling extracts semantic constituents by searching input text to fill in values for predefined slots in a semantic frame [56]. Slot filling task can also be viewed as a sequence labeling task that assigns an appropriate semantic label to each word in the given input utterance. In the below example from ATIS [1] corpus following the popular IOB (in/out/begin) annotation method, *Seattle* and *San Diego* are tagged as the *from* and *to* locations respectively according to the slot labels, and *tomorrow* is tagged as the *departure date*. Other words in the example utterance that carry no semantic meaning are assigned "O" label.

Utterance	show	flights	from	Seattle	to	San	Diego	tomorrow
Slots	O	O	O	B-fromloc	O	B-toloc	I-toloc	B-depart_date
Intent	Flight							

Figure 2.2: ATIS corpus sample with intent and slot annotation (IOB format).

Given an utterance consisting of a sequence of words $\mathbf{w} = (w_1, w_2, ..., w_T)$, the goal of slot filling is to find a sequence of semantic labels $\mathbf{s} = (s_1, s_2, ..., s_T)$, one for each word in the utterance, such that:

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}} P(\mathbf{s}|\mathbf{w}) \tag{2.2}$$

Popular sequence models for slot filling including using hidden Markov models [17], conditional random fields [18] and recurrent neural networks [37, 48, 50, 56, 55, 59, 70].

Joint Learning of SLU

For the SLU module in task-oriented dialog systems, the intent detection and slot filling tasks are usually handled separately by different models. Such design has several limitations. Firstly, one has to train two separated models for these tasks in each domain, which introduces additional burden to dialog system development and maintenance. In addition, there is no effective knowledge sharing between different SLU tasks. This may lead to sub-optimal system performance as the close relations between user's intents and semantic slots expressed in the natural language query is not effectively utilized. In later of this thesis (section 4.1), we will discuss how we can jointly model these two tasks to improve the overall system performance.

In spoken dialog systems, inputs to the SLU module are the speech recognition (ASR) hypotheses of the user's auditory input. The word error rate (WER) of ASR systems has a direct impact to the performance of the downstream SLU module. Language model (LM), which assigns a probability estimate of a sequence of words, plays an important role in state-of-the-art ASR systems [29]. Language modeling is also closely related to language understanding, as the semantic meanings that are being incrementally captured in user's speech directly indicate what a user is likely to say next. Previous work on speech and language processing usually treats ASR and SLU separately. The benefit of integrating SLU and LM for speech recognition and understanding is not well explored. We will later (section 4.2) discuss how we can use a single neural network model to jointly optimize SLU and LM to enable a tighter integration of spoken language processing and understanding.

2.1.2 Dialog State Tracking

Dialog state tracker (DST) is a core component in task-oriented spoken dialog systems [101, 79]. DST refers to the task of maintaining a distribution over possible dialog states. The dialog state in task-oriented dialog systems can be seen as a full representation of the constraints that the user has expressed so far. In slot-based dialog systems, the dialog state is usually expressed in terms of a list of goal slots and the probability distribution of candidate values for each slot. These output distributions are also referred to as belief state. In estimating the current dialog state, input to the state tracker may contain the ASR hypotheses, SLU hypotheses, and previous system dialog acts.

Early systems for dialog state tracking are designed with domain specific rules that are highly hand-crafted. Such systems might be easy to implement, but they are not capable of modeling uncertainties in ASR and SLU in a principled manner [54]. To address this limitation, learning based dialog state trackers have been proposed in literature. Generative approaches for state tracking include using dynamic Bayesian network [20], which models dialog state and user action as latent variables. The network can be optimized with Bayesian inference. Such generative model has limitations in flexibly introducing arbitrary features [54]. Discriminative models using conditional random fields [34] and recurrent neural networks [43, 44, 58] address this limitation and achieve advanced state tracking performance.

2.1.3 Dialog Policy

Dialog policy module generates a dialog act based on the current dialog state, which serves as a central part of dialog management [10, 21, 16, 38]. The dialog act can be seen as shallow representations of the natural language utterance semantics. A dialog act usually consists of a speech act and grounding information represented by slots and values. For example, the dialog act $affirm(food = italian)$ indicates that the system is to affirm the user that the restaurant serves *Italian* food, where $affirm$ is the speech act, and $food$ and $italian$ form the slot-value pair.

A dialog act is produced by the system at the end of each dialog turn based on the dialog state, which contains information of the entire sequence of interactions between the system and the user so far:

$$\hat{a_k} = \arg\max_{\text{a}} P(a_k|a_{<k}, u_{<k}) \tag{2.3}$$

where $a_{<k}$ and $u_{<k}$ represent the system acts and user input before the kth turn.

2.1.4 End-to-End Dialog Systems

Conventional task-oriented dialog systems use a pipeline design by connecting the above core system components together. Such pipeline system has a number of limitations. Firstly, such design makes it hard to extend a working system to new domains, as each system module has to be retrained separately with data and knowledge from the target domain. Secondly, credit assignment and error tracking can be challenging in modular-based systems. The compounding errors from upper-stream modules to down-stream modules make it hard to identify the true source of errors in the pipeline. Last but not least, the optimization targets of the individual modules might not fully align with the overall system evaluation metrics. For example, it is not clear whether improvement on SLU intent detection accuracy will lead to the same level of improvement on overall dialog task success.

In addressing these limitations of pipeline dialog systems, researchers have recently started exploring end-to-end solutions for task-oriented dialogs, inspired by the success of end-to-end trainable neural network models in non-task-oriented chit-chat dialog settings [60, 77]. Wen et al. [101] proposed an end-to-end trainable neural dialog model with modularly connected system components for SLU, DST, and dialog policy. Although these system components are end-to-end connected, they are pre-trained separately. The intent and state tracking networks are trained in supervised manner with specifically collected labels. The policy and generation networks are separately trained on the system utterances. It is not clear how the errors in state tracking during run-time will impact the performance of the policy and generation networks. Moreover, the system is trained with supervised learning on fixed dialog corpora, and thus may not generalize well to unseen dialog states when interacting with users.

Bordes and Weston [85] proposed a task-oriented dialog model from a machine reading and reasoning approach. They used an RNN to encode the dialog state and applied end-to-end memory networks to learn its representation. In the same line of research, people explored using query-reduction networks [97], gated memory networks [88], and copy-augmented networks [87] to learn the dialog state RNN. Similar to [101], these systems are trained on fixed sets of simulated and human-machine dialog corpora, and thus are not capable to learn interactively from users. The grounding information from knowledge base are pulled offline based on existing dialog corpus. It is unknown whether the same reasoning capability achieved in offline model training can be generalized to online user interaction. Moreover, these systems skip the slot-based state tracking stage, and directly generate or select a final system response based on dialog context. Such systems work well in the evaluation setting using machine-machine or human-machine dialog corpora, but may face difficulty in modeling human-human conversations as human agent's natural language response can in very diverse forms.

Williams et al. [102] proposed a hybrid code network for task-oriented dialog that can be trained with supervised and reinforcement learning (RL). They show that RL performed on top of supervised pre-training model with labeled dialogs improves learning speed dramatically. Li et al. [90] and Dhingra et al. [111] also proposed end-to-end task-oriented dialog models that can be trained with hybrid supervised learning and RL. These systems apply RL directly on supervised pre-training models, without discussing the potential issue with dialog state distribution mismatch between supervised training and interactive learning. Moreover, these end-to-end dialog models are trained and evaluated using rule-based user simulators. Ideally, RL based interactive model learning should be performed with human users by collecting real user feedback. In performing interactive learning with human user, online learning efficiency becomes a critical factor. This sample efficiency issue with RL policy learning is not addressed in these works.

2.2 Sequence Modeling

Task-oriented dialog modeling can be treated as a sequence learning problem, where the dialog model takes in user's input over a sequence of dialog turns and produce corresponding actions and responses. In the following sections, we give a brief review of sequence models that form the foundation of our proposed model architecture and learning methods.

2.2.1 Sequence Labeling

Sequence labeling is a class of machine learning problems where we have training examples of $\left\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) : i = 1, ..., N\right\}$ and we learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps any input sequence $\mathbf{x}$ to a corresponding label sequence $f(\mathbf{x})$. Here $\mathcal{X}$ represents the set of all possible input sequences, and $\mathcal{Y}$ represents the set of all possible output labels. $\mathbf{x}^{(i)}$ is the i-th training example, which is a sequence of observations $x_1^{(i)}, ..., x_{Ti}^{(i)}$. $\mathbf{y}^{(i)}$ is the corresponding label sequence $y_1^{(i)}, ..., y_{Ti}^{(i)}$. Generative and discriminative approaches have been applied on sequence labeling tasks. For generative models, the joint probability $P(\mathbf{x}, \mathbf{y})$ is modeled. During inference, the conditional probability $P(\mathbf{y}|\mathbf{x})$ can be derived with Bayes rule, and the predicted label sequence $\hat{\mathbf{y}}$ can be derived as:

$$\begin{aligned} \hat{\mathbf{y}} = \arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) &= \arg\max_{\mathbf{y}} \frac{P(\mathbf{y})P(\mathbf{x}|\mathbf{y})}{P(\mathbf{x})} \\ &= \arg\max_{\mathbf{y}} P(\mathbf{y})P(\mathbf{x}|\mathbf{y}) \end{aligned} \tag{2.4}$$

An important class of generative sequence labeling model is the Hidden Markov Model (HMM). In HMM sequence labeling model, the hidden states correspond to labels. Efficient training (forward and backward methods) and inference (Viterbi) algorithms have been developed for HMM following Markov assumption. A limitation with HMM is that to ensure tractability in modeling joint probability distribution, observation at a time step may only directly depend on the label at the same time step. This restrains the model from learning long range dependencies among observation elements in the sequence. To avoid making such unwarranted independent assumptions for tractable inference, an alternative approach is to model the conditional probability $P(\mathbf{y}|\mathbf{x})$ directly. Such discriminative approach ensures that interactive features from observation sequence can be used in the model, and no modeling effort is made specially on modeling the observation sequence itself [13].

Maximum Entropy Markov Model (MEMM) is one such discriminative method in sequence labeling that models the conditional distribution directly. The key advantage of MEMM compared to HMM is that much richer features from observation sequence can be used in the log linear model for classification. However, MEMM and other discriminative finite-state models suffer from the label bias problem [11], in which the state transition is likely to bias toward states with fewer outgoing connections.

Conditional Random Fields (CRFs) are introduced to keep the advantage of MEMM and avoid the label bias problem. Instead of normalizing per-state log linear conditional probabilities, a CRF defines a single log linear model over the label sequence for conditional probability given the observation sequence. The un-normalized transition scores at each step help to solve the label bias problem. Training can be performed with maximum likelihood criteria using iterative gradient methods. Viterbi algorithm is used during inference.

2.2.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a set of artificial neural network that store information and states with feedback connections. Similar to MEMM, input features of different types can be easily added to the network. In addition, the non-linear transformations of the input

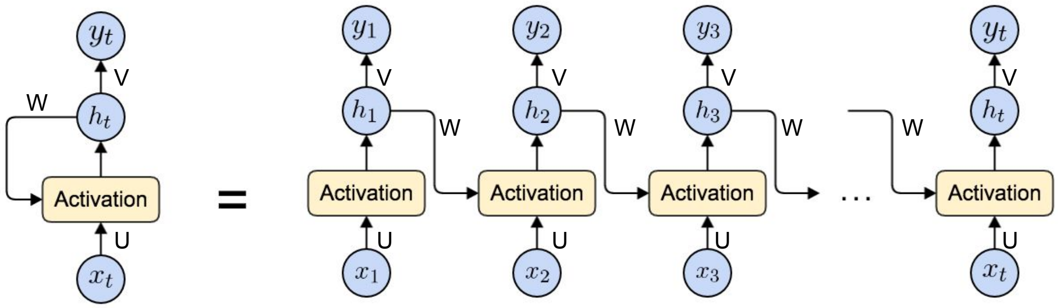

Figure 2.3: Unrolled recurrent neural network.

and state space equipped the RNN models with much larger modeling capacity comparing to classical log-linear models such as MEMM. A widely used RNN architecture is the Elman type RNN [7]. In contrast with feed-forward neural networks, the Elman RNN feeds the hidden layer output at time $t-1$ back to the same hidden layer at time t via recurrent connections. Thus, information stored in the hidden layer can be viewed as a summary of input sequence up till the current time. A non-linear and differentiable activation function is applied to the weighted sum of the input vector and previous hidden layer activation. Popular choices of such non-linear activation functions are sigmoid, hyperbolic tangent, and rectified linear unit. Hidden layer at time t can thus be expressed as:

$$h(t) = \theta(Ux(t) + Wh(t-1)) \tag{2.5}$$

Where θ is the non-linear activation function.

The output layer of the RNN contains neurons representing possible labels in the sequence labeling task. Input arriving at each unit of the output layer is calculated in the same way as in the hidden layer being the weighted sum over the units connecting to it. At each time step, sequence labeling task can be seen as a classification problem with K classes (typically with $K > 2$). Softmax function can be used to normalize the output activation and obtain the class probabilities:

$$\begin{aligned} y(t) &= \text{softmax}(Vh(t)) \\ \text{softmax}(z_m) &= \frac{e^{z_m}}{\Sigma_k e^{z_k}} \end{aligned} \tag{2.6}$$

In RNN training, parameters in the network can be optimized based on loss functions derived using maximum likelihood, with z_k being 1 if k is the true label, and 0 otherwise:

$$\mathcal{L}(x, z) = -\sum_{k=1}^{K} z_k log(y_k) \tag{2.7}$$

The loss function (negative log-likelihood) can be minimized using gradient descent. Network parameters are updated using back-propagation through time (BPTT) method considering influence of past states through recurrent connections. Error from the output layer is back-propagated to the hidden layers through the recurrent connections backwards in time. The weight matrices are updated after each training sample or mini-batch.

2.2.3 Long Short-Term Memory

Regular RNN by design is capable of storing long range context information. However, the range of context can be accessed in practice is limited due to vanishing and exploding gradient problems [4]. Numerous attempts were made to address such problems. One popular approach is using Long Short-Term Memory (LSTM) architecture [6], a special type of RNN that addresses the vanishing and exploding gradient problems.

LSTM model replaces the recurrent module that uses sigmoid or hyperbolic tangent activation function in basic RNN with memory block. Memory block may contain one or more memory cells. The cell state contains summarized information of previous observations, the propagation of which is regulated by cell gates. Such mechanism prevents the gradients from vanishing or exploding when passing across time steps.

$$
\begin{aligned}
\textit{Forget Gate}: \quad & f(t) = \text{sigmoid}(U_f x(t) + W_f h(t-1)) \\
\textit{Input Gate}: \quad & i(t) = \text{sigmoid}(U_i x(t) + W_i h(t-1)) \\
\textit{Cell State}: \quad & \tilde{C}(t) = \tanh(U_c x(t) + W_c h(t-1)) \\
& C(t) = f(t) * C(t-1) + i(t) * \tilde{C}(t) \\
\textit{Output Gate}: \quad & o(t) = \text{sigmoid}(U_o x(t) + W_o h(t-1)) \\
\textit{Cell Output}: \quad & h(t) = o(t) * \tanh(C(t))
\end{aligned} \tag{2.8}
$$

These controlling gates in memory block help the model to learn when to let the activation get in the cell state at each time step. Correspondingly, in the network training backward pass, the gates also learn when to let the error in and out at each step. Such mechanism helps to avoid gradient exploding and vanishing problems.

2.3 Policy Gradient Reinforcement Learning

In learning the dialog system via the interaction with users, we use reinforcement learning, especially policy-based reinforcement learning methods, to optimize the dialog policy with user feedback. Here we provide a brief review of the related learning algorithms.

A reinforcement learning algorithm learns by interacting with its environment. At each time step t, a reinforcement learning agent selects an action a_t to take based on current state s_t. The agent receives a numerical reward r_t from the environment indicating the success of taking that action. The expectation is that the agent learns to select actions that maximize the accumulated reward over time.

Policy gradient is a method that learns a parameterized policy that can select actions without calculating a value function for a state [26]. The policy parameters are learned based on the gradient of an objective function $J(\theta)$ (e.g. the accumulated reward) with respect to

Algorithm 1 REINFORCE

Initialize policy parameters θ
for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\}$ following $\pi_\theta(s,a)$ **do**
    for each step of the episode $t = 1, \ldots, T-1$ **do**
        Obtain return R from future step rewards
        $\theta \leftarrow \theta + \alpha\nabla_\theta log(\pi_\theta(s_t, a_t))R$
    end for
end for

the parameters. The policy can be parameterized in any way such that the policy $\pi_\theta(a|s)$ is differentiable with respect to the policy parameters θ. For problem with discrete action space, a popular choice of parameterization method is to form a transformation for each state-action pair $h_\theta(s,a)$. Each action in the action space is assigned a probability obtained by, for example, an exponential softmax distribution:

$$a \sim \pi_\theta(a|s) = \frac{\exp((h(s,a;\theta)))}{\sum_m \exp(h(s,m;\theta))} \tag{2.9}$$

In calculating the policy gradient, with likelihood ratio estimation, we have:

$$\begin{aligned} \nabla_\theta \pi_\theta(s,a) =& \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ & \pi_\theta(s,a)\nabla_\theta log(\pi_\theta(s,a)) \end{aligned} \tag{2.10}$$

REINFORCE [3] is popular policy gradient reinforcement learning method. It is a Monte Carlo algorithm that uses complete return R from time t which represents all future rewards till the end of an episode. This return R can be seen as an unbiased sample of the value of taking action a in state s.

REINFORCE as a Monte Carlo policy gradient method may suffer from high variance. One reason is that the current step return R depends on all future rewards in the trajectory. A baseline can be introduced during the gradient update which leaves the expected value of the update unchanged, but reduces the large variance with REINFORCE. A natural choice

Algorithm 2 REINFORCE with Baseline

Initialize policy parameters θ, and state value function parameters w
for each episode $\{s_1, a_1, r_2, ... , s_{T-1}, a_{T-1}, r_T\}$ following $\pi_\theta(s, a)$ **do**
  for each step of the episode $t = 1, ..., T - 1$ **do**
    Obtain return R from future step rewards
    Calculate advantage $\delta = R - v_w(s_t)$
    $w \leftarrow w + \alpha_w \nabla_w v_w(s_t)\delta$
    $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta log(\pi_\theta(s_t, a_t))\delta$
  end for
end for

for the baseline is the state value function $v_w(s)$, which estimate the value of given state s. The state-value function parameters w can also be learned with Monte Carlo method.

Chapter 3

End-to-End Dialog Learning Framework

In this chapter, we present an overview of the proposed dialog learning framework. We will first discuss the design considerations that motivate our study. Then, we describe the proposed learning framework in terms of system design, system learning, and system evaluation.

3.1 Design Considerations

Task-oriented dialog system helps users to complete a particular task by conducting multi-turn conversations. To successfully complete a task, a dialog system needs to understand the user's request in natural language format, reason and plan over multiple dialog turns, and provide responses that are based on the facts in the real world. Given the conventional system architecture and limitations described in section 2.1, we take the following design considerations in designing the proposed dialog learning framework.

3.1.1 Dialog Context Modeling

Corpus based chit-chat systems tend to do very little modeling of the dialog context. They usually focus on producing a system response that is natural and appropriate given the user's input utterance. Task-oriented dialog systems, on the other hand, are required to collect information from users and clarify user's request over multiple dialog turns in order to successfully complete a task. This makes dialog context modeling especially important [83, 84, 103]. Conventional frame-based systems maintain an explicit dialog state which is typically represented by a list of slot-value pairs plus the previous user dialog act. Such state representation can provide a reasonable indication of task completion progress, but it drops many other useful information such user style and user behavior patterns. These patterns can be valuable in better understanding the user and completing the task. We want to design a model with which a system can decide what information in the dialog context to maintain in order to best complete a task via a data-driven approach. Based on such more comprehensive state representation, the system learns to take actions which finally lead to maximized dialog rewards.

3.1.2 End-to-End System Optimization

Conventional systems for task-oriented dialog use a modular-based approach with independently trained system components connected in a pipeline. A limitation with this approach is that when one component is updated, ideally all other downstream components in the pipeline should be retrained so that they have input similar to the training examples at run-time. Moreover, there is a concern on how well the individual component optimization targets align with the overall system evaluation criteria. For example, it is not easy to examine how much of an improvement in user satisfaction is resulted from the improvement in slot filling F-1 score. We want to design a system that can be optimized directly towards the final system evaluation criteria, such as task success and user satisfaction. Not only the dialog policy can be adjusted with the feedback signal from the user as in [32, 78, 99], but the entire system can be optimized end-to-end. This helps to release the burden of manually upgrading and calibrating different dialog system components.

3.1.3 Continuous Interactive Learning

Unlike chit-chat systems that are usually modeled with single-turn context-response pairs [62, 60, 108], task-oriented dialog systems involves reasoning over multiple dialog turns and planning based on the fact in the real world information system. The state space in a multi-turn conversation grows exponentially with the growing number of dialog turns. Collecting and annotating a large enough dialog corpus that covers all possible dialog scenarios for a specific task is very challenging in practice. This makes it especially important for a system to be able to continuously learn through the interaction with users in addition to the initial corpus based model training. We want to design a dialog learning framework that allows an agent to effectively learn from different modes of user interactions. In the least demanding form, an agent can learn to adjust its policy by requesting users for simple forms of feedback (e.g. a binary or continuous reward). If a user is willing to provide more specific instructions or demonstrations, the agent can also learn effectively from such user teaching and further reduce the interactive learning cycles required.

3.2 System Design

Based on the above described design considerations, we proposed a deep neural network based interactive dialog learning framework. An overview of the framework is shown in Figure 3.1. We design a hierarchical recurrent neural network based dialog agent. With all functioning components modeled by neural networks and connected via differentiable operations, the entire system can be end-to-end optimized.

From an intelligent system learning point of view, we let the dialog agent learn to act by

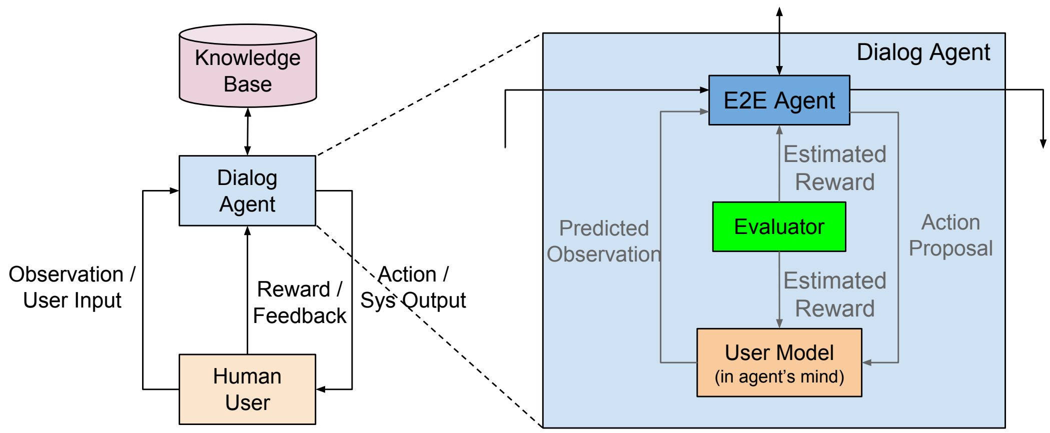

Figure 3.1: Proposed task-oriented dialog learning framework.

interacting with the world and maximizing the long-term success or expected reward. Ideally, the dialog agent should not only be able to passively receive signals from the environment (e.g. the user) and learn to act on it, but also to be able to understand the dynamics of the environment and predict the changes of environment state. This is also how we human beings learn from the world. We interact with the world and learn from the reward and penalty received. We try to understand how the world works and learn to predict the consequences of our actions. With this world knowledge in mind, we plan and act accordingly in order to achieve higher long-term rewards.

We design our dialog learning framework following the same philosophy. The dialog agent interacts with user in natural language format and continuously improves itself with the user feedback received. The dialog agent also learns to interface with external resources, such as a knowledge base or a database, so as to provide responses to user that are based on the facts in the real world. In addition, the dialog agent also learns to model the users and predict their behaviors. Such user model in the agent's mind can be used to simulate conversations that mimic the real conversations between the agent and users. By simulating such conversations and learning from it, the agent can learn more efficiently from the interactions with real users.

3.3 System Learning

In training the proposed neural dialog system, we design a hybrid learning method by combining offline supervised training on dialog corpus and interactive learning with human-in-the-loop. We pretrain the dialog agent in a supervised manner using task-oriented dialog corpus. The supervised training agent can continuously to improve itself via interacting with users and learning from user demonstration and feedback with imitation and reinforcement learning.

3.3.1 Learn from Dialog Corpus

Learning dialog model from scratch via interacting with users can be very challenging, as the agent has to learn robust natural language understanding in addition to a good policy. Instead, we propose to pretrain the dialog model by fitting annotated task-oriented dialog samples in a supervised manner. This enables a good initialization of the agent model parameters during the following interactive learning stage.

3.3.2 Learn from User Teaching

Once obtaining a basic supervised pretraining model, we deploy the agent to let it conduct task-oriented dialog with users and learn with human-in-the-loop. The agent may make mistakes and fail to complete a task, especially at the beginning of the interactive learning stage. We may ask a user (or an expert) to point out the mistakes that the agent makes in understanding the user's request and demonstrate the desired agent behavior or actions. We design an imitation dialog learning method that allows our agent to learn efficiently from such user teaching.

3.3.3 Learn from User Feedback

The above described interactive learning method with user teaching requires a user to provide instruction at each dialog turn. This is a demanding process and may only be applicable to a small group of expert users in practice. We want to limit the number of such imitation dialog learning cycles and continue to improve the agent using a form of supervision signal that is easier to obtain. We design an interactive dialog learning method with deep reinforcement learning that only uses simple forms of user feedback. Different from the turn-level corrections required during the imitation dialog learning from user teaching, the feedback is only collected from users at the end of a dialog.

Learning from user feedback has several practical concerns. For example, rating from a user to a system may not always be available, and it can vary from time to time. In addressing the impact of such inconsistency in user feedback or rating, we propose an adversarial learning method which can effectively estimate the reward (i.e. the user rating) for a dialog in guiding the dialog model optimization. Another practice concern with interactive learning with real user is that a user may easily get impatient with inferior system performance, especially at the early stage of the interactive learning sessions. How to learn efficiently from limited user interactions so that the system performance can be quickly improved to a sufficient satisfaction level is a big challenge. To address this sample efficiency challenge in online interactive dialog learning, we further propose a method by integrating the learning experience from real and imagined interactions to improve the interactive learning efficiency.

3.4 System Evaluation

We evaluate the propose models and learning methods in both simulated environments using user simulators and human evaluation settings with real users recruited via Amazon Mechanical Turk. The main evaluation metrics used are task success rate and user satisfaction scores

[8, 5].

3.4.1 Automatic Evaluation

For the supervised model training evaluation, we evaluate the model's capability in fitting the test set dialog samples. Specifically, we present the model performance on (1) dialog state tracking [42, 79] which estimates a user's goal at each turn and (2) accuracy in predicting the next system action or utterance [85, 97] based on the true label in the test set. For the interactive evaluation against user simulators, we present the learning curves for task success rate, average dialog length for successful dialogs, and average dialog rewards.

3.4.2 Human User Evaluation

We further evaluate the proposed system with human judges recruited via Amazon Mechanical Turk. Each judge is asked to rate each system at dialog turn level on a scale of 1 (frustrating) to 5 (optimal way to help the user). We collect multiple user ratings for each dialog turn and calculate an overall score for each system.

Chapter 4

Robust Spoken Language Understanding

In this chapter, we present our proposed models and learning methods for spoken language understanding (SLU), a critical component in spoken dialog systems. The main job of SLU is to identify a user's intent and extract semantic constituents from the user's natural language query, two tasks that are often referred to as intent detection and slot filling. Intent detection and slot filling are usually processed separately. We will describe how we can jointly model these two tasks with a single recurrent neural network based model, and show its advantages over training task-specific models.

We further discuss the need for incremental language understanding based on speech recognition hypothesis. Most of the recent models for SLU focus on offline training and evaluation based on the complete utterance from users. In speech recognition, instead of receiving the transcribed text at the end of the speech, users typically prefer to see the ongoing transcription while speaking. Similarly, in SLU, with real time intent identification and semantic constituents extraction, the downstream systems can be enabled to perform corresponding search or query while the user dictates. Motivated by this, we propose a model that performs online incremental SLU as new word from speech recognizer arrives. The incremental SLU outputs can further provide additional context for next word prediction in speech recognition decoding. This enables a tight connection between SLU and language modeling in speech recognition. This chapter is based on the publications in [73, 74].

4.1 Spoken Language Understanding with Multi-Task Learning

Intent detection and slot filling in SLU are usually processed separately. Intent detection can be treated as a semantic utterance classification problem, and slot filling can be treated as a sequence labeling task. Joint model for intent detection and slot filling has also been proposed in literature [40, 36]. Such joint model simplifies SLU systems, as only one model needs to be trained and fine-tuned for the two tasks.

Recently, encoder-decoder neural network models have shined in many sequence learning problems such as machine translation [47] and speech recognition [65]. Encoder-decoder model with attention can map sequences that are of different lengths when no alignment information is given. In slot filling, however, alignment is explicit, and thus alignment-based RNN models typically work well. We want to explore how the alignment information in slot filling can be best utilized in the encoder-decoder models, and on the other hand, whether the alignment-based RNN slot filling models can be further improved with the attention mechanism that introduced from the encoder-decoder architecture. Moreover, we want to investigate how slot filling and intent detection can be jointly modeled under such schemes.

4.1.1 Model

In this section, we first describe our approach on integrating alignment information to the encoder-decoder architecture for slot filling and intent detection. Following that, we describe the proposed method for introducing attention mechanism from the encoder-decoder architecture to the alignment-based RNN models.

Encoder-Decoder Model with Aligned Inputs

The encoder-decoder model for joint intent detection and slot filling is illustrated in Figure 4.1. On encoder side, we use a bidirectional RNN. The bidirectional RNN encoder reads the source word sequence forward and backward. The final encoder hidden state h_i at each time step i is a concatenation of the forward RNN state $\overrightarrow{h_i}$ and backward RNN state $\overleftarrow{h_i}$, i.e. $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$.

The last state of the forward and backward encoder RNN carries information of the entire source sequence. We use the last state of the backward encoder RNN to compute the initial decoder hidden state. The decoder is a unidirectional RNN. At each decoding step i, the decoder state s_i is calculated as a function of the previous decoder state s_{i-1}, the previous emitted label y_{i-1}, the aligned encoder hidden state h_i, and the context vector c_i:

$$s_i = f(s_{i-1}, y_{i-1}, h_i, c_i) \tag{4.1}$$

where the context vector c_i is computed as a weighted sum of the encoder states $\mathbf{h} = (h_1, ..., h_T)$ [52]:

$$c_i = \sum_{j=1}^{T} \alpha_{i,j} h_j \tag{4.2}$$

and

$$\begin{aligned} \alpha_{i,j} &= \frac{\exp(e_{i,j})}{\sum_{k=1}^{T} \exp(e_{i,k})} \\ e_{i,k} &= g(s_{i-1}, h_k) \end{aligned} \tag{4.3}$$

g a feed-forward neural network. At each decoding step, the explicit aligned input is the encoder state h_i. The context vector c_i provides additional information to the decoder and

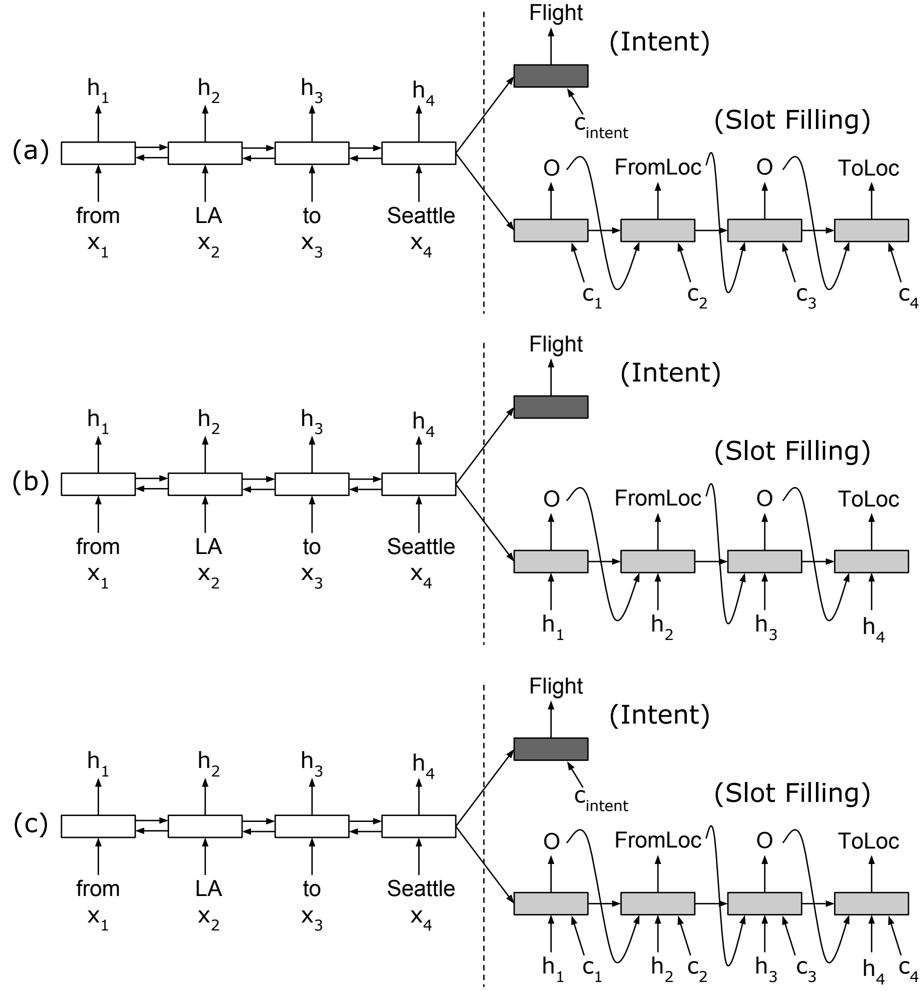

Figure 4.1: Encoder-decoder model for joint intent detection and slot filling. (a) with no aligned inputs. (b) with aligned inputs. (c) with aligned inputs and attention.

can be seen as a continuous bag of weighted features $(h_1, ..., h_T)$.

For joint modeling of intent detection and slot filling, we add an additional decoder for intent classification task that shares the same encoder with slot filling decoder. During model training, costs from both decoders are back-propagated to the encoder. The intent decoder generates only one single output which is the intent class distribution of the sentence, and thus alignment is not required. The intent decoder state is a function of the shared initial decoder state s_0, which encodes information of the entire source sequence, and the context vector c_{intent}, which indicates part of the source sequence that the intent decoder

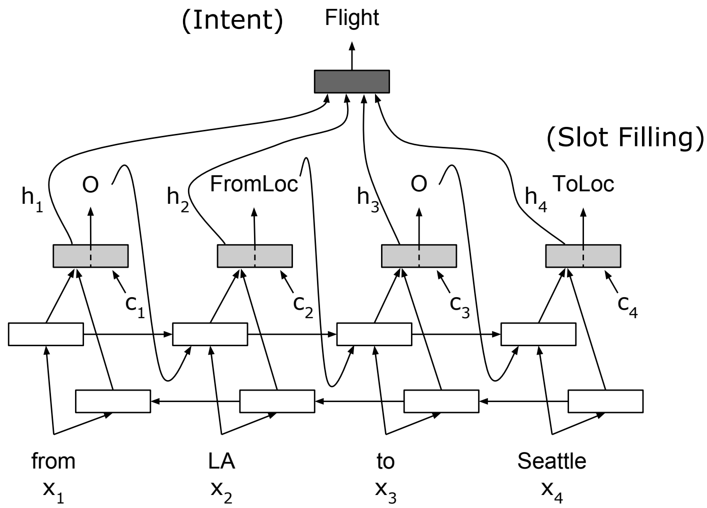

Figure 4.2: Attention-based LSTM model for joint intent detection and slot filling.

pays attention to.

Attention-Based RNN Model

The attention-based RNN model for joint intent detection and slot filling is illustrated in Figure 4.2[1]. The idea of introducing attention to the alignment-based RNN sequence labeling model is motivated by the use of attention mechanism in encoder-decoder models. In bidirectional RNN for sequence labeling, the hidden state at each time step carries information of the whole sequence, but information may gradually lose along the forward and backward propagation. Thus, when making slot label prediction, instead of only utilizing the aligned hidden state h_i at each step, we would like to see whether the use of context vector c_i gives us any additional supporting information, especially those require longer term dependencies that is not being fully captured by the hidden state.

In the proposed model, a bidirectional RNN (BiRNN) reads the source sequence in both forward and backward directions. We use LSTM cell for the basic RNN unit. Slot label dependencies are modeled in the forward RNN. Similar to the encoder module in the above described encoder-decoder architecture, the hidden state h_i at each step is a concatenation

[1]Code to reproduce our experiments: `https://github.com/HadoopIt/rnn-nlu`

of the forward state $\overrightarrow{h_i}$ and backward state $\overleftarrow{h_i}$, $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$. Each hidden state h_i contains information of the whole input word sequence, with strong focus on the parts surrounding the word at step i. This hidden state h_i is then combined with the context vector c_i to produce the label distribution, where the context vector c_i is calculated as a weighted average of the LSTM hidden states $\mathbf{h} = (h_1, ..., h_T)$.

For joint modeling of intent detection and slot filling, we reuse the pre-computed hidden states $\mathbf{h}$ of the bidirectional RNN to produce intent class distribution. If attention is not used, we apply mean-pooling [63] over time on the hidden states $\mathbf{h}$ followed by logistic regression to perform the intent classification. If attention is enabled, we instead take the weighted average of the hidden states $\mathbf{h}$ over time.

4.1.2 Experiments

We evaluate the proposed model on ATIS (Airline Travel Information Systems) data set [2] which widely used in SLU research. The data set contains audio recordings of people making flight reservations. We follow the ATIS corpus setup used in [57, 55, 36, 23]. The training set contains 4978 utterances from the ATIS-2 and ATIS-3 corpora, and the test set contains 893 utterances from the ATIS-3 NOV93 and DEC94 data sets. There are in total 127 distinct slot labels and 18 different intent types. We evaluate the system performance on slot filling using F1 score, and the performance on intent detection using classification error rate.

LSTM cell is used as the basic RNN unit in the experiments. Given the size the data set, we set the number of units in LSTM cell as 128. Word embeddings of size 128 are randomly initialized and fine-tuned during mini-batch training with batch size of 16. Dropout rate 0.5 is applied to the non-recurrent connections [51] during model training for regularization. Maximum norm for gradient clipping is set to 5.

Independent Training Model Results: Slot Filling

We first report the results on our independent task training models. Table 4.1 shows the slot filling F1 scores using our proposed architectures. Table 4.2 compares our proposed model performance on slot filling to previously reported results.

Table 4.1: Independent training model results on ATIS slot filling.

Model	**F1 Score**	**Average**
(a) Encoder-decoder NN with no aligned inputs	81.64	79.66 ± 1.59
(b) Encoder-decoder NN with aligned inputs	95.72	95.38 ± 0.18
(c) Encoder-decoder NN with aligned inputs & attention	**95.78**	95.47 ± 0.22
BiRNN no attention	95.71	95.37 ± 0.19
BiRNN with attention	**95.75**	95.42 ± 0.18

In Table 4.1, the first set of results are for variations of encoder-decoder models. Not to our surprise, the pure attention-based slot filling model that does not utilize explicit alignment information performs poorly. Letting the model to learn the alignment from training data does not seem to be appropriate for slot filling task. Line 2 and line 3 show the F1 scores of the non-attention and attention-based encode-decoder models that utilize the aligned inputs. The attention-based model gives slightly better F1 score than the non-attention-based one, on both the average and best scores. By investigating the attention learned by the model, we find that the attention weights are more likely to be evenly distributed across words in the source sequence. There are a few cases where we observe insightful attention (Figure 4.3) that the decoder pays to the input sequence, and that might partly explain the observed performance gain when attention is enabled.

The second set of results in Table 4.1 are for bidirectional RNN models. Similar to the previous set of results, we observe slightly improved F1 score on the model that uses attentions. The contribution from the context vector for slot filling is not very obvious. It seems that

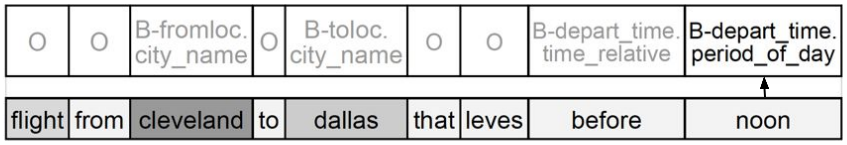

Figure 4.3: Illustration of the inferred attention. Darker shades indicate higher attention weights.

for sequence length at such level (average sentence length is 11 for this ATIS corpus), the hidden state h_i that produced by the bidirectional RNN is capable of encoding most of the information that is needed to make the slot label prediction. Table 4.2 compares our slot filling models to previous approaches. Results from both of our model architectures advance the best F1 scores reported previously.

Table 4.2: Comparison to previous approaches. Independent training model results on ATIS slot filling.

Model	**F1 Score**
CNN-CRF [36]	94.35
RNN with Label Sampling [55]	94.89
Hybrid RNN [57]	95.06
Deep LSTM [49]	95.08
RNN-EM [59]	95.25
Encoder-labeler Deep LSTM [70]	95.66
Attention Encoder-Decoder NN (with aligned inputs)	**95.78**
Attention BiRNN	**95.75**

Independent Training Model Results: Intent Detection

Table 4.3 compares intent classification error rate between our intent models and previous approaches. Intent error rate of our proposed models outperform the state-of-the-art results by a large margin. The attention-based encoder-decoder intent model advances the bidirectional RNN model. This might be attributed to the sequence level information passed from the encoder and additional layer of non-linearity in the decoder RNN.

Table 4.3: Comparison to previous approaches. Independent training model results on ATIS intent detection.

Model	Error (%)
Recursive NN [40]	4.60
Boosting [23]	4.38
Boosting + Simplified sentences [27]	3.02
Attention Encoder-Decoder NN	**2.02**
Attention BiRNN	**2.35**

Joint Model Results

Table 4.4 shows our joint training model performance on intent detection and slot filling comparing to previous reported results. As shown in this table, the joint training model using encoder-decoder architecture achieves 0.09% absolute gain on slot filling and 0.45% absolute gain (22.2% relative improvement) on intent detection over the independent training model. For the attention-based bidirectional RNN architecture, the join training model achieves 0.23% absolute gain on slot filling and 0.56% absolute gain (23.8% relative improvement) on intent detection over the independent training models. The attention-based RNN model seems to benefit more from the joint training. Results from both of our joint training approaches outperform the best reported joint modeling results.

Table 4.4: Comparison to previous approaches. Joint training model results on ATIS slot filling and intent detection.

Model	F1 Score	Intent Error (%)
RecNN [40]	93.22	4.60
RecNN+Viterbi [40]	93.96	4.60
Attention Encoder-Decoder NN (with aligned inputs)	**95.87**	**1.57**
Attention BiRNN	**95.98**	**1.79**

4.2 Contextual Language Modeling and Language Understanding

The above presented joint models are for offline SLU model training and evaluation. They are not suitable for *online* tasks where it is desired to produce outputs as the input sequence arrives. Joint SLU models proposed in previous work typically require intent and slot label predictions to be conditioned on the entire transcribed utterance. This limits the usage of these models in the online setting.

We propose an RNN-based online joint SLU model that performs intent detection and slot filling as the input word arrives. In addition, we discuss how the generated intent class and slot labels can be useful for next word prediction in online automatic speech recognition (ASR). We propose to perform intent detection, slot filling, and language modeling jointly in a conditional RNN model. The proposed joint model can be further extended for belief tracking in dialog systems when considering the dialog history beyond the current utterance. Moreover, it can be used as the RNN decoder in an end-to-end trainable sequence-to-sequence speech recognition model [69].

4.2.1 Model

In this section we describe the joint SLU-LM model in detail[2]. Figure 4.4 gives an overview of the proposed architecture.

Referring to the joint SLU-LM model shown in Figure 4.4, for the intent model, instead of predicting the intent only after seeing the entire utterance, in the joint model we output intent at each time step as input word sequence arrives. The intent generated at the last step is used as the final utterance intent prediction. The intent output from each time step is fed back to the RNN state, and thus the entire intent output history are modeled and can

[2]Code to reproduce our experiments: `https://github.com/HadoopIt/joint-slu-lm`

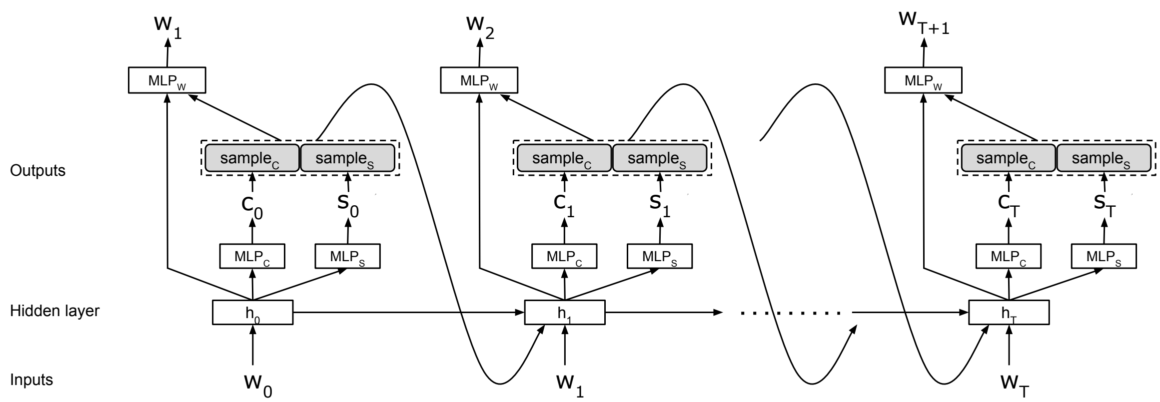

Figure 4.4: Proposed joint online RNN model for intent detection, slot filling, and next word prediction.

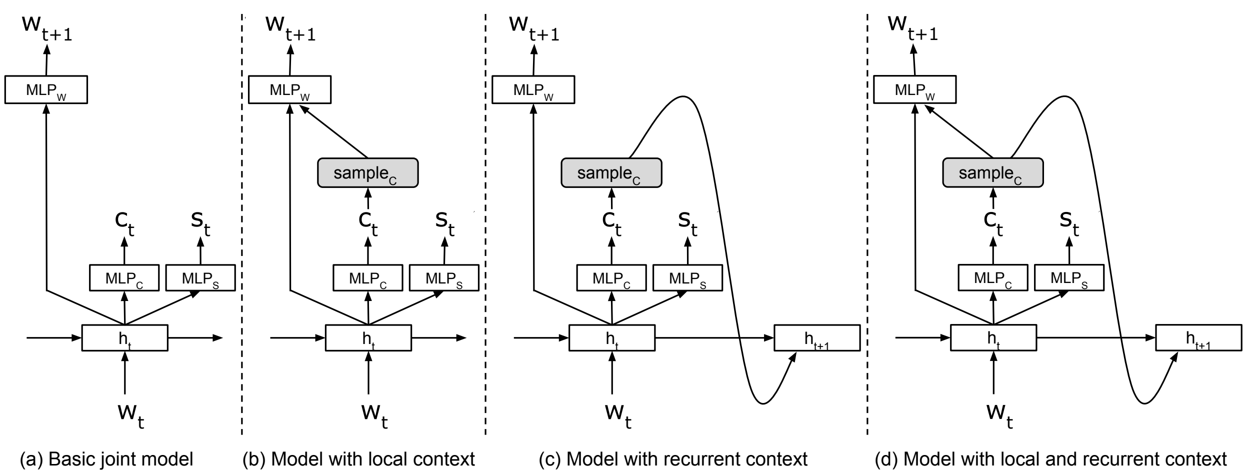

Figure 4.5: Joint online SLU-LM model variations. (a) Basic joint model. (b) Joint model with local intent context. (c) Joint model with recurrent intent context. (d) Joint model with both local and recurrent intent context.

be used as context to other tasks. It is not hard to see that during inference, intent classes that are predicted during the first few time steps are of lower confidence due to the limited information available. We describe the techniques that can be used to ameliorate this effect in the section below. For the intent model, with both intent and slot label connections to the RNN state, we have:

$$P(c_T|\mathbf{w}) = P(c_T|w_{\leq T}, c_{<T}, s_{<T}) \tag{4.4}$$

For the slot filling model, at each step t along the input word sequence, we want to model the slot label output s_t as a conditional distribution over the previous intents $c_{<t}$, previous

slot labels $s_{<t}$, and the input word sequence up to step t. Using the chain rule, we have:

$$P(\mathbf{s}|\mathbf{w}) = P(s_0|w_0)\prod_{t=1}^{T} P(s_t|w_{\leq t}, c_{<t}, s_{<t}) \tag{4.5}$$

For the language model, the next word is modeled as a conditional distribution over the word sequence together with intent and slot label sequence up to current time step. The intent and slot label outputs at current step, together with the intent and slot label history that is encoded in the RNN state, serve as context to the language model.

$$P(\mathbf{w}) = \prod_{t=0}^{T} P(w_{t+1}|w_{\leq t}, c_{\leq t}, s_{\leq t}) \tag{4.6}$$

Next Step Prediction

Following the model architecture in Figure 4.4, at time step t, input to the system is the word at index t of the utterance, and outputs are the intent class, the slot label, and the next word prediction. The RNN state h_t encodes the information of all the words, intents, and slot labels seen previously. The neural network model computes the outputs through the following sequence of steps:

$$h_t = \text{LSTM}(h_{t-1}, [w_t, c_{t-1}, s_{t-1}]) \tag{4.7}$$

$$P(c_t|w_{\leq t}, c_{<t}, s_{<t}) = \text{IntentDist}(h_t) \tag{4.8}$$

$$P(s_t|w_{\leq t}, c_{<t}, s_{<t}) = \text{SlotLabelDist}(h_t) \tag{4.9}$$

$$P(w_{t+1}|w_{\leq t}, c_{\leq t}, s_{\leq t}) = \text{WordDist}(h_t, c_t, s_t) \tag{4.10}$$

where LSTM is the recurrent neural network function that computes the hidden state h_t at a step using the previous hidden state h_{t-1}, the embeddings of the previous intent output c_{t-1} and slot label output s_{t-1}, and the embedding of current input word w_t. IntentDist, SlotLabelDist, and WordDist are multilayer perceptrons (MLPs) with softmax outputs over intents, slot labels, and words respectively. Each of these three MLPs has its own set of parameters. The intent and slot label distributions are generated by the MLPs with input being the RNN cell output. The next word distribution is produced by conditioning on current step RNN cell output together with the embeddings of the sampled intent and sampled slot label.

Training

The network is trained to find the parameters θ that minimize the cross-entropy of the predicted and true distributions for intent class, slot label, and next word jointly. The objective function also includes an $L2$ regularization term $R(\theta)$ over the weights and biases of the three MLPs. This equalizes to finding the parameters θ that maximize the below objective function:

$$\max_{\theta} \sum_{t=0}^{T} \Big[\alpha_c \log P(c^* | w_{\leq t}, c_{<t}, s_{<t}; \theta) + \alpha_s \log P(s_t^* | w_{\leq t}, c_{<t}, s_{<t}; \theta) \\ + \alpha_w \log P(w_{t+1} | w_{\leq t}, c_{\leq t}, s_{\leq t}; \theta) \Big] - \lambda R(\theta) \tag{4.11}$$

where c^* is the true intent class and and s_t^* is the true slot label at time step t. α_c, α_s, and α_w are the linear interpolation weights for the true intent, slot label, and next word probabilities. During model training, c_t can either be the true intent or mixture of true and predicted intent. During inference, however, only predicted intent can be used. Confidence of the predicted intent during the first few time steps is likely to be low due to the limited information available, and the confidence level is likely to increase with the newly arriving words. Conditioning on incorrect intent for next word prediction is not desirable. To mitigate this effect, we propose to use a *schedule* to increase the intent contribution to the context vector along the growing input word sequence. Specifically, during the first k time steps, we disable the intent context completely by setting the values in the intent vector to zeros. From step $k+1$ till the last step of the input word sequence, we gradually increase the intent context by applying a linearly growing scaling factor η from 0 to 1 to the intent vector. This scheduled approach is illustrated in Figure 4.6.

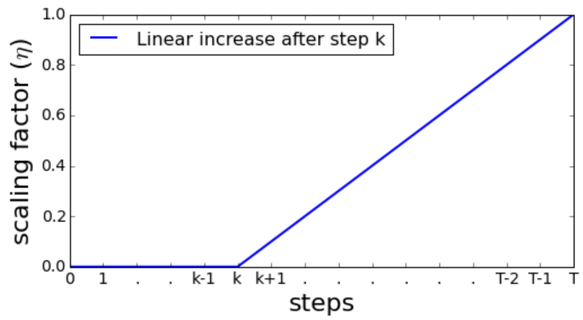

Figure 4.6: Schedule of increasing intent contribution to the context vector along with the growing input sequence.

Inference

For online inference, we simply take the greedy path of our conditional model without doing search. The model emits the best intent class and slot label at each time step conditioning on all previous emitted symbols:

$$\hat{c}_t = \arg\max_{c_t} P(c_t | w_{\leq t}, \hat{c}_{<t}, \hat{s}_{<t}) \tag{4.12}$$

$$\hat{s}_t = \arg\max_{s_t} P(s_t | w_{\leq t}, \hat{c}_{<t}, \hat{s}_{<t}) \tag{4.13}$$

4.2.2 Experiments

Results with True Text Input

Table 4.5 summarizes the experiment results of the joint SLU-LM model and its variations using ATIS text corpus as input. The basic joint model (row 6) uses a shared representation for all the three tasks. It gives slightly better performance on intent detection and next word prediction, with some degradation on slot filling F1 score. If the RNN output h_t is connected to each task output directly via linear projection without using MLP, performance

	Model	Intent Error	F1 Score	LM PPL
1	RecNN [41]	4.60	93.22	-
2	RecNN+Viterbi [41]	4.60	93.96	-
3	Independent training RNN intent model	2.13	-	-
4	Independent training RNN slot filling model	-	94.91	-
5	Independent training RNN language model	-	-	11.55
6	Basic joint training model	2.02	94.15	11.33
7	Joint model with *local* intent context	1.90	94.22	11.27
8	Joint model with *recurrent* intent context	1.90	94.16	10.21
9	Joint model with *local & recurrent* intent context	1.79	94.18	10.22
10	Joint model with *local* slot label context	1.79	94.14	11.14
11	Joint model with *recurrent* slot label context	1.79	**94.64**	11.19
12	Joint model with *local & recurrent* slot label context	1.68	94.52	11.17
13	Joint model with *local* intent + slot label context	1.90	94.13	11.22
14	Joint model with *recurrent* intent + slot label context	**1.57**	94.47	**10.19**
15	Joint model with *local & recurrent* intent + slot label context	1.68	94.45	10.28

Table 4.5: ATIS Test set results on intent detection error, slot filling F1 score, and language modeling perplexity.

drops for intent classification and slot filling. Thus, we believe the extra discriminative power introduced by the additional model parameters and non-linearity from MLP is useful for the joint model.

Row 7 to row 9 of Table 4.5 illustrate the performance of the joint models with local, recurrent, and local plus recurrent intent context, which correspond to model structures described in Figure 4.5(b) to 4.5(d). It is evident that the recurrent intent context helps the next word prediction, reducing the language model perplexity by 9.4% from 11.27 to 10.21. The contribution of local intent context to next word prediction is limited. We believe the advantageous performance of using recurrent context is a result of modeling predicted intent history and intent variations along with the growing word sequence. For intent classification and slot filling, performance of these models with intent context is similar to that of the basic joint model.

Row 10 to row 12 of Table 4.5 illustrate the performance of the joint model with local, recurrent, and local plus recurrent slot label context. Comparing to the basic joint model, the introduced slot label context (both local and recurrent) leads to a better language modeling performance, but the contribution is not as significant as that from the recurrent intent context. Moreover, the slot label context reduces the intent classification error from 2.02 to 1.68, a 16.8% relative error reduction. From the slot filling F1 scores in row 10 and row 11, it is clear that modeling the slot label dependencies by connecting slot label output to the recurrent state is very useful.

Row 13 to row 15 of Table 4.5 give the performance of the joint model with both intent and slot label context. Row 15 refers to the model described in Figure 4.4. As can be seen from the results, the joint model that utilizes two types of recurrent context maintains the benefits of both, namely, the benefit of applying recurrent intent context to language modeling, and the benefit of applying recurrent slot label context to slot filling. Another observation is that once recurrent context is applied, the benefit of adding local context for next word prediction is limited. It might hint that the most useful information for the next word prediction can be well captured in the RNN state, and thus adding explicit dependencies on local intent class and slot label is not very helpful.

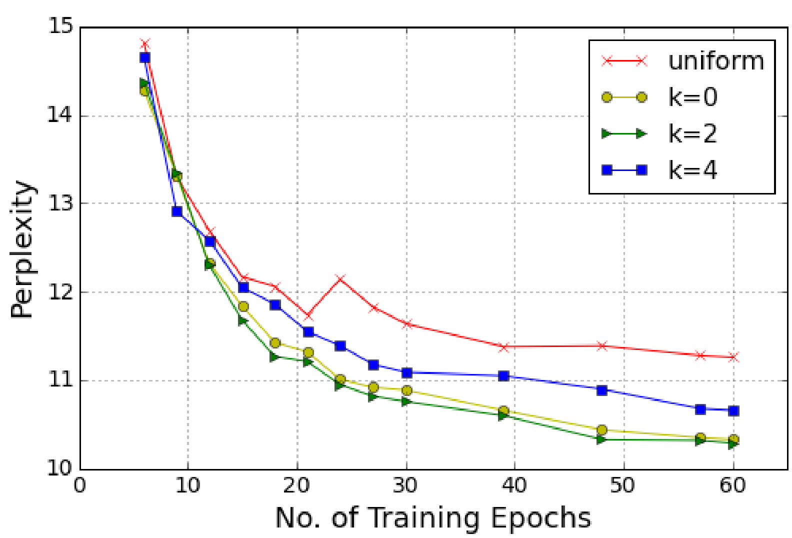

Figure 4.7: LM perplexity of the joint SLU-LM models with different schedules in adjusting the intent contribution to the context vector.

During the joint model training and inference, we used a schedule to adjust the intent contribution to the context vector by linearly scaling the intent vector with the growing input word sequence after step k. We found this technique to be critical in achieving advantageous language modeling performance. Figure 6 shows test set perplexities along the training epochs for models using different k values, comparing to the model with uniform ($\eta = 1$) intent contribution. With uniform intent contribution across time, the context vector does not bring benefit to the next word prediction, and the language modeling perplexity is similar to that of the basic joint model. By applying the adjusted intent scale ($k = 2$), the perplexity drops from 11.26 (with uniform intent contribution) to 10.29, an 8.6% relative reduction.

Results in ASR Settings

ASR Model (with LibriSpeech AM)	WER	Intent Error	F1 Score
2-gram LM decoding	14.51	4.63	84.46
2-gram LM decoding + 5-gram LM rescoring	13.66	5.02	85.08
2-gram LM decoding + Independent training RNN LM rescoring	12.95	4.63	85.43
2-gram LM decoding + Joint training RNN LM rescoring	**12.59**	**4.44**	**86.87**

Table 4.6: ATIS test set results on ASR word error rate, intent detection error, and slot filling F1 score with noisy speech input.

To further evaluate the robustness of the proposed joint SLU-LM model, we experimented with noisy speech input and performed SLU on the rescored ASR outputs. Model performance is evaluated in terms of ASR word error rate (WER), intent classification error, and slot filling F1 score. As shown in Table 4.6, the model with joint training RNN LM rescoring outperforms the models using 5-gram LM rescoring and independent training RNN LM rescoring on all the three evaluation metrics. Using the rescored ASR outputs (12.59% WER) as input to the joint training SLU model, the intent classification error increased by 2.87%, and slot filling F1 score dropped by 7.77% comparing to the setup using true text input. The performance degradation is expected as we used a more challenging and realistic setup with noisy speech input. These results in Table 4.6 show that our joint training model outperforms the independent training model consistently on ASR and SLU tasks.

4.3 Conclusion

In this chapter, we presented an attention-based bidirectional RNN model for joint intent detection and slot filling in SLU. Joint model training enables effective representation learning of the user's natural language input. Using a joint model for the two SLU tasks also simplifies the dialog system, as only one model needs to be trained and deployed. The proposed models achieved state-of-the-art performance for both intent detection and slot filling on the benchmarking ATIS task. We further described a conditional RNN model that can be used to jointly perform online spoken language understanding and language modeling. We showed that by incrementally modeling user's intent and slot label dependencies while new word in user's speech transcription arrives, the joint training model achieved superior performance in intent detection and language modeling with slight degradation on slot filling comparing to the independent training models. The joint model also showed consistent performance gain over the independent training models in the more challenging and realistic setup using noisy speech input.

Chapter 5

End-to-End Dialog Modeling

In the previous chapter, we presented the proposed neural network based models for robust spoken language understanding. In this chapter, we discuss how we can extend the recurrent models for SLU to end-to-end dialog modeling. We will first present the architecture of our neural network based task-oriented dialog model. Then, we discuss how we can train such model in an end-to-end manner. This chapter is based on the work presented in [92, 107].

Conventional systems for task-oriented dialog usually consist of modules connected in a pipeline for spoken language understanding, dialog management, and natural language generation (NLG) [9, 16, 14, 38]. Such pipeline system has a number of limitations. Firstly, modules in current systems are highly handcrafted and designed with domain-specific rules. This makes domain extension and user adaptation less flexible. Secondly, errors made in the upper stream modules of the pipeline propagate to downstream components and get amplified, making it hard to track the source of errors. Last but not least, current system modules are usually trained independently, and their individual optimization targets may not fully align with the overall system evaluation criteria (e.g. task success rate and user satisfaction).

To address these limitations, efforts have been made recently in designing end-to-end frame-works for task-oriented dialogs. Wen et al. proposed an end-to-end trainable neural network model [101] with modularly connected neural networks for each system component. Zhao and Eskenazi introduced an end-to-end reinforcement learning framework [82] that jointly performs dialog state tracking and policy learning. Li et al. proposed an end-to-end learning framework [90] that leverages both supervised and reinforcement learning signals and showed promising dialog modeling performance. Bordes and Weston proposed an end-to-end memory network method [85], and modeled task-oriented dialog with a reasoning approach without explicitly learning the dialog policy and tracking belief state.

In this chapter, we present the proposed end-to-end trainable neural network model for task-oriented dialog. The model uses a unified neural network with differentiable connections for belief tracking, knowledge base (KB) operation, and response creation. The model is able to track dialog state, interface with a KB, and incorporate structured KB query results into system responses to successfully complete task-oriented dialogs.

5.1 System Architecture

Figure 5.1 shows the overall architecture of the proposed end-to-end task-oriented dialog model. We use a hierarchical LSTM neural network to encode a dialog with a sequence of turns. User input to the system in natural language format is encoded to a continuous vector via a bidirectional LSTM utterance encoder, similar to encoding method described in Chapter 4.1. Instead of performing utterance level intent prediction and slot filling, we directly feed this continuous representation of user utterance to a dialog-level LSTM together with the encoding of the previous system action. State of this dialog-level LSTM maintains a continuous representation of the dialog state. Based on this state, the model generates a probability distribution over candidate values for each of the tracked goal slots. A query command can then be formulated with the state tracking outputs and issued to a knowledge base to retrieve requested information. Finally, the system produces a dialog action, which is

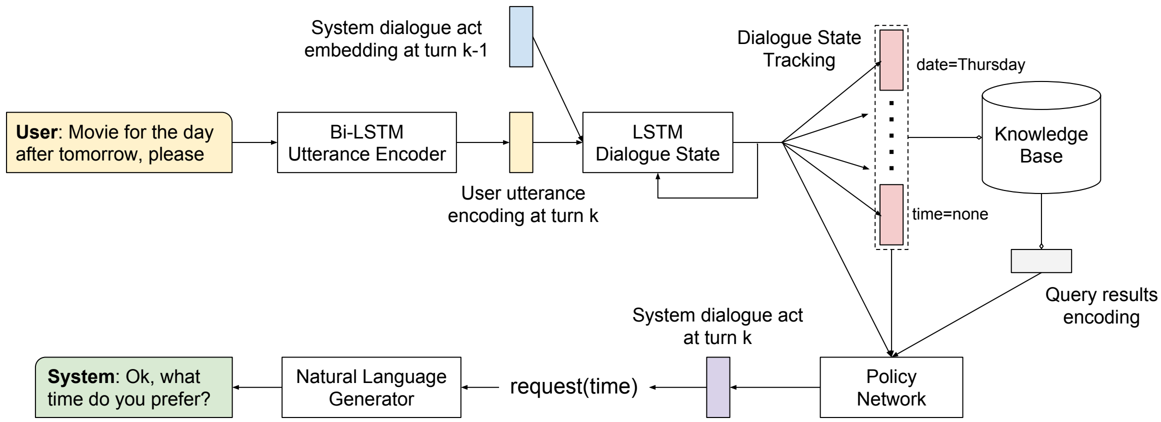

Figure 5.1: End-to-end task-oriented dialog system architecture.

conditioned on information from the dialog state, the estimated user's goal, and the encoding of the query results. This dialog action, together with the user goal tracking results and the query results, is used to generate the final natural language system response via a natural language generator. We describe each core model component in detail in the following sections.

5.1.1 Utterance Encoding

We use a bidirectional LSTM to encode the user utterance to a continuous representation. We refer to this LSTM as the utterance-level LSTM. The user utterance vector is generated by concatenating the last forward and backward LSTM states. Let $\mathbf{U}_k = (w_1, w_2, ..., w_{T_k})$ be the user utterance at turn k with T_k words. These words are firstly mapped to an embedding space, and further serve as the step inputs to the bidirectional LSTM. Let $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ represent the forward and backward LSTM state outputs at time step t. The user utterance vector U_k is produced by:

$$U_k = [\overrightarrow{h_{T_k}^{U_k}}, \overleftarrow{h_1^{U_k}}] \tag{5.1}$$

where $\overrightarrow{h_{T_k}}$ and $\overleftarrow{h_1}$ are the last states in the forward and backward LSTMs.

5.1.2 Dialog State Tracking

Dialog state tracking, or belief tracking, maintains the state of a conversation, such as user's goals, by accumulating evidence along the sequence of dialog turns. Our model maintains the dialog state in a continuous form in the dialog-level LSTM (LSTM_D) state s_k. s_k is updated after the model processes each dialog turn by taking in the encoding of user utterance U_k and the encoding of the previous turn system output A_{k-1}. This dialog state serves as the input to the dialog state tracker. The tracker updates its estimation of the user's goal represented by a list of slot-value pairs. A probability distribution $P(l_k^m)$ is maintained over candidate values for each goal slot type $m \in M$:

$$s_{A,k} = \text{LSTM}_\text{A}(s_{A,k-1},\ [U_k,\ A_{k-1}]) \tag{5.2}$$

$$P(l_{A,k}^m \mid \mathbf{U}_{\leq k}, \mathbf{A}_{<k}) = \text{SlotDist}_A^m(s_{A,k}) \tag{5.3}$$

where SlotDist_m is a single hidden layer MLP with softmax activation over slot type $m \in M$.

5.1.3 KB Operation

Conditioning on the state of the conversation, the model may issue an API call to query the KB based on belief tracking results. A simple API call command template is firstly generated by the model. The final API call command is produced by replacing the slot type tokens in the command template with the best hypothesis for each of the goal slot from the belief tracker. In movie search domain, an example API call template can be "*api_call* ⟨*movie*⟩ ⟨*date*⟩ ⟨*time*⟩", and the tokens are to be replaced with the dialog state tracker outputs to form the final API call command "*api_call star_wars sunday dontcare*".

In interfacing with KBs, instead of using a soft KB lookup as in [111], our model sends symbolic queries to the KB and leaves the ranking of the KB entities to an external recommender system. Entity ranking in real world systems can be made with much richer features (e.g.

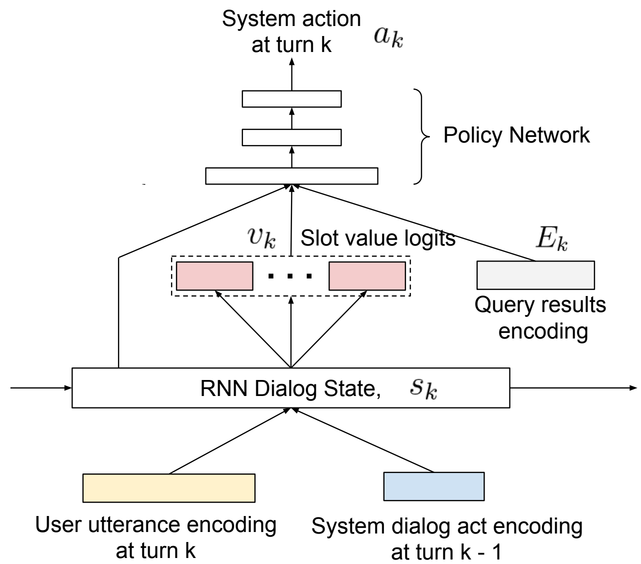

Figure 5.2: Proposed end-to-end task-oriented neural dialog model.

user profiles, local context, etc.) in the back-end system other than just following entity posterior probabilities conditioning on a user utterance. Hence ranking of the KB entities is not a part of our proposed neural dialog model. We assume that the model receives a ranked list of KB entities according to the issued query and other available sources, such as user models.

Once the KB query results are returned, we save the retrieved entities to a queue and encode the result summary to a vector. Rather than encoding the real KB entity values as in [85, 87], we only encode a summary of the query results (i.e. item availability and number of matched items). This encoding serves as a part of the input to the policy network.

5.1.4 Dialog Policy

Dialog policy selects the next system action in response to the user's input based on the current dialog state. We use a deep neural network to model the dialog policy. There are three inputs to the policy network, (1) the dialog-level LSTM state s_k, (2) the log probabilities of candidate values from the belief tracker v_k, and (3) the encoding of the query results summary E_k. The policy network emits a system action in the form of a dialog act conditioning on these inputs:

$$P(a_{A,k} \mid \mathbf{U}_{\leq k}, \mathbf{A}_{<k}, \mathbf{E}_{\leq k}) = \mathrm{PolicyNet}_{\mathrm{A}}(s_{A,k}, v_k, E_k) \tag{5.4}$$

where v_k represents the concatenated log probabilities of candidate values for each goal slot, E_k is the encoding of query results, and PolicyNet is a single hidden layer MLP with softmax activation function over all system actions.

5.1.5 System Response Generation

The emitted system action from the policy network is finally used to produce a system response in natural language format by combining the state tracker outputs and the retrieved KB entities. We use a template based natural language generator. The delexicalised tokens in the NLG template are replaced by the values from either the estimated user goal values or the KB entities, depending on the emitted system action.

5.2 Model Training

By connecting all the system components, we have an end-to-end model for task-oriented dialog. Each system component is a neural network that takes in underlying system component's outputs in a continuous form that is fully differentiable, and the entire system (utterance encoding, dialog state tracking, and policy network) can be trained end-to-end.

We train the system in a supervised manner by fitting task-oriented dialog samples. The model predicts the true user goal slot values and the next system action at each turn of a dialog. We optimize the model parameter set θ by minimizing a linear interpolation of cross-entropy losses for dialog state tracking and system action prediction:

$$\min_{\theta_A} \sum_{k=1}^{K} -\Big[\sum_{m=1}^{M} \lambda_{l^m} \log P(l_{A,k}^{m}{}^{*} | \mathbf{U}_{\le k}, \mathbf{A}_{<k}, \mathbf{E}_{<k}; \theta_A) \\ + \lambda_a \log P(a_{A,k}^{*} | \mathbf{U}_{\le k}, \mathbf{A}_{<k}, \mathbf{E}_{\le k}; \theta_A) \Big] \tag{5.5}$$

where λs are the linear interpolation weights for the cost of each system output. l_k^{m*} is the ground truth label for the tracked user goal slot type $m \in M$ at the kth turn, and a_k^* is the true system action in the corpus.

We perform mini-batch model training with batch size of 32 using Adam optimization method [46]. Regularization with dropout is applied to the non-recurrent connections [51] during model training with dropout rate of 0.5. We set the maximum norm for gradient clipping to 5 to prevent exploding gradients.

Hidden layer sizes of the dialog-level LSTM and the utterance-level LSTM are set as 200 and 150 respectively. Word embeddings of size 300 are randomly initialized. We also explore using pre-trained word vectors [35] that are trained on Google News dataset to initialize the word embeddings.

5.3 Experiments

5.3.1 Data Set

We evaluate the proposed model using DSTC2 [42] dataset in restaurant search domain. Bordes and Weston [85] transformed the original DSTC2 corpus by adding system commands and removing the dialog state annotations. This transformed corpus contains additional API calls that the system would make to the KB and the corresponding KB query results. We combine the original DSTC2 corpus and this transformed version by keeping the dialog state annotations and adding the system commands. We can thus perform more complete

User goal: {"goal": {"constraints": [["food", "french"], ["pricerange", "dontcare"]], "request-slots": ["addr", "phone"]}}	
Agent:	Hello, welcome to the Cambridge restaurant system. You can ask for restaurants by area, price range or food type. How may I help you?
User:	uh yes im looking for malaysian food
Agent:	I'm sorry but there is no restaurant serving malaysian food
User:	uh okay how about french food
Agent:	You are looking for a french restaurant right?
User:	yes and i dont care about the price range
Agent:	Sorry would you like something in the cheap price range or you dont care
User:	i dont care
Agent:	What part of town do you have in mind?
User:	uh it doesnt matter
Agent:	cote serves french food
User:	okay um
Agent:	Sure , cote is on Bridge Street City Centre
User:	whats the address
Agent:	Sure , cote is on Bridge Street City Centre
User:	okay and whats the phone number
Agent:	The phone number of cote is 01223 311053
User:	okay and um
Agent:	restaurant two two serves french food
User:	thank you bye

Table 5.1: DSTC2 dialog samples.

evaluation of our model's capability in tracking the dialog state, processing KB query results, and conducting complete dialog. Statistics of this augmented DSTC2 data set is summarized in the Table 5.2.

Table 5.2: Statistics of the augmented DSTC2 data set.

Number of train / dev / test dialogs	1618 / 500 / 1117
Number of turns per dialog in average (including API call commands)	9.21
Number of area / food / pricerange options	5 / 91/ 3
Number of delexicalised response candidates	78

Additionally, we evaluate the system on a large scale dialog corpus[1] in movie booking domain. The movie booking dialog corpus is generated [109] using a finite state machine based dialog agent and an agenda based user simulator [19] with natural language utterances rewritten

[1] The dataset can be accessed via `https://github.com/google-research-datasets/simulated-dialog`

by real users. The user simulator can be configured with different personalities, showing various levels of randomness and cooperativeness. This user simulator is also used to interact with our end-to-end training agent during imitation and reinforcement learning stages. We randomly select a user profile when conducting each dialog simulation. During model evaluation, we use an extended set of natural language surface forms over the ones used during training time to evaluate the generalization capability of the proposed end-to-end model in handling diverse natural language inputs.

5.3.2 Results and Analysis

We first experiment with different text encoding methods and recurrent model architectures to find best performing model. Table 5.3 shows the evaluation results of models using different user utterance encoding methods and different word embedding initialization on DSTC2 dataset. Bidirectional LSTM (Bi-LSTM) shows clear advantage in encoding user utterance comparing to bag-of-means on word embedding (BoW Emb) method, improving the joint goal prediction accuracy by 4.6% and the final system response accuracy by 1.4%. Using pre-trained word vectors (word2vec) boosts the model performance further. These results show that the semantic similarities of words captured in the pre-trained word vectors are helpful in generating a better representation of user input, especially when the utterance contains words or entities that are rarely observed during training.

Table 5.3: Prediction accuracy for entity pointer, joint user goal, delexicalised system response, and final system response of DSTC2 test set using different encoding methods and word vector initializations.

Model	**Entity Pointer**	**Joint Goal**	**De-lex Res**	**Final Res**
BoW Emb Encoder	93.5	72.6	55.4	51.2
+ word2vec	93.6	74.3	55.9	51.5
Bi-LSTM Encoder	93.8	**77.2**	55.8	52.6
+ word2vec	**94.4**	76.6	**56.6**	**52.8**

Table 5.4 and Table 5.5 show the supervised learning model performance on DSTC2 and the

movie booking corpus. Evaluation is made on DST accuracy. For the evaluation on DSTC2 corpus, we use the live ASR transcriptions as the user input utterances. Our proposed model achieves near state-of-the-art dialog state tracking results on DSTC2, on both individual slot tracking and joint slot tracking, comparing to the recent published results using RNN [43] and neural belief tracker (NBT) [76]. In the movie booking domain, our model also achieves promising performance on both individual slot tracking and joint slot tracking accuracy. Instead of using ASR hypothesis as model input as in DSTC2, here we use text based input which has much lower noise level in the evaluation of the movie booking tasks. This partially explains the higher DST accuracy in the movie booking domain comparing to DSTC2.

Table 5.4: Dialog state tracking results on the standard DSTC2 dataset

Model	**Area**	**Food**	**Price**	**Joint**
RNN	92	86	86	69
RNN+sem. dict	92	86	92	71
NBT	90	84	94	72
Our E2E model	90	84	92	72

Table 5.5: Dialog state tracking results on movie booking dataset

Model	**Num_ticket**	**Movie**	**Theater**	**Date**	**Time**	**Joint**
Our E2E model	98.22	91.86	97.33	99.31	97.71	84.57

Finally, we compare the performance of the proposed method to recently published works using per-response accuracy metric. Even though using the same evaluation measurement, our model is designed with slightly different settings comparing to other published models in Table 5.6. Instead of using additional matched type features [85, 97] (i.e. KB entity type feature for each word, e.g. whether a word is a food type or area type, etc.), we use user's goal slots at each turn that are mapped from the original DSTC2 dataset as additional supervised signals in our model. Moreover, instead of treating KB query results as unstructured text, we treat them as structured entities and let our model to pick the right entity by selecting the most appropriate entity pointer. Our proposed model successfully predicts 52.8% of the true system responses given the single best ASR transcription of the user utterances.

Table 5.6: Performance of the proposed model in terms of per-response accuracy comparing to previous approaches.

Model	Per-res Accuracy
Memory Networks [85]	41.1
Gated Memory Networks [95]	48.7
Sequence-to-Sequence [87]	48.0
Query-Reduction Networks [97]	51.1
Hierarchical LSTM E2E	**52.8**

To further understand the prediction errors made by our model, we conduct human evaluation by inviting 10 users to evaluate the appropriateness of the responses generated by our system. While some of the errors are made on generating proper API calls due to the errors in dialog state tracking results, we also find quite a number of responses that are considered appropriate by our judges but do not match to the reference responses in DSTC2 test set. For example, there are cases where our system directly issues the correct API call (e.g. "*api_call south italian expensive*") based on user's request, instead of asking user for confirmation of a goal type (e.g. "*Did you say you are looking for a restaurant in the south of town?*") as in the reference corpus. By taking such factors into consideration, our system is able to generate appropriate responses in 73.6% of the time based on feedback from the judges. These results show that the existing performance metrics do not well correlate with human judgments [75], and better dialogs evaluation measurements should to be further explored.

5.4 Conclusion

In this chapter, we described an end-to-end trainable neural network model for task-oriented dialog systems. The model is capable of interfacing with a knowledge base (KB) by issuing API calls and incorporating structured KB query results into system responses to successfully complete task-oriented dialogs. In the evaluation in a restaurant search domain using data from the second Dialog State Tracking Challenge (DSTC2) and a movie booking domain, we showed that the proposed model achieved robust performance in tracking dialog state over the sequence of dialog turns. We further demonstrated the model's superior performance in predicting the next system response comparing to prior end-to-end trainable neural network models.

Chapter 6

Dialog Learning with Human-in-the-Loop

In the previous chapter, we described our neural network based task-oriented dialog model and showed how we can train it offline in a supervised manner using annotated dialog samples. Comparing to chit-chat dialog models that are usually trained using single-turn context-response pairs, task-oriented dialog model involves reasoning and planning over multiple dialog turns. The dialog state space grows exponentially with the growing number of dialog turns. In practice, it can be very challenging to collect large enough dialog samples in every single task domain that cover all possible dialog scenarios for supervised model training. This makes it especially important for a system to be able to continuously learn from user interactions.

In this chapter, we present how the proposed neural dialog system can continuously improve itself through online interactions with humans. We let the supervised pre-trained dialog agent to interact with users and conduct task-oriented dialogs and show how we can continuously improve it with the feedback from users using deep reinforcement learning (RL) methods. Previous work on RL based interactive learning systems [32, 78, 99] can only improve the dialog policy module of the system using the user feedback. We will show how we can optimize the dialog agent end-to-end using the proposed neural network based model. We further illustrate the benefit of performing such end-to-end system optimization.

In addition to improving the system by only collecting simple forms of user feedback, we further investigate how we can enhance the learning efficiency by asking users for additional instructions. Imagine a scenario where a dialog agent does not know how to perform a certain task. Instead of letting the agent to explore the action space only by trial and error, we can ask users to demonstrate the desired agent behaviors and let the agent to quickly learn from such human demonstrations. We will show how we can let the agent to learn effectively from such human teaching and feedback with a hybrid imitation and reinforcement learning method. This chapter is based on the publication in [93, 107].

6.1 End-to-End Model Optimization with Deep RL

Many of the recently proposed end-to-end models are trained in supervised manner [101, 85, 87, 92] by learning from human-human or human-machine dialog corpora. Deep RL based systems [90, 91, 102, 111] that learns by interacting with human user or user simulator have also been studied in the literature. Comparing to supervised training models, systems trained with deep RL showed improved task success rate and model robustness towards diverse dialog scenarios.

We model task-oriented dialog as a sequential decision making problem, where an agent select an action to take at each dialog turn conditioning on the current dialog state. The goal is that such sequence of actions finally leads to a maximized expected reward. Our interactive learning system is based on the system architecture described in Chapter 5. Each system component takes in underlying component's outputs in a continuous from which is fully differentiable with respect to the system optimization target. This makes the entire system end-to-end trainable with the feedback received from the users.

6.1.1 Model

In this section, we define the state, action, and reward in our RL dialog model training setting and present the training details.

State The system maintains a continuous representation of the dialog state after processing the user inputs at each turn k. This continuous form dialog state is encoded in the dialog-level LSTM state $s_{A,k}$. This continuous form state, together with the agent's estimation of candidate value probability distribution for each slot and the encoding of the KB query results summary, forms the final state representation for the policy network, i.e. $state_k = [s_{A,k}, v_k, E_k]$. Such representation may encode information of the entire conversation history and related external knowledge up up till the current turn.

Action Actions of the dialog agent is the system action outputs $a_{A,k}$. An action is sampled by the agent based on a stochastic representation of the policy, which produces a probability distribution over actions given a dialog state. The action space is finite and discrete. The system action is defined with the act and slot types from a dialog act [33]. For example, the dialog act "$confirm(date = monday)$" is mapped to a system action "$confirm\_date$" and a candidate value "*monday*" for slot type "*date*". The slot types and values are from the dialog state tracking output.

Reward The reward r is received at the end of a dialog from the user. If a user simulator is used in the experiment, we apply task-completion as the metric in designing the dialog reward. One can extend it by introducing additional factors to the reward functions, such as naturalness of interactions or costs associated with KB queries. A positive reward is applied for successful tasks, and a zero reward is applied for failed tasks.

For each proceeding turns, we assign a small penalty as step reward to encourage the agent to complete the task in fewer steps. The final turn-level reward is obtained by summing the discounted step rewards in future turns with a discount factor $\gamma \in [0, 1)$. To reduce variance

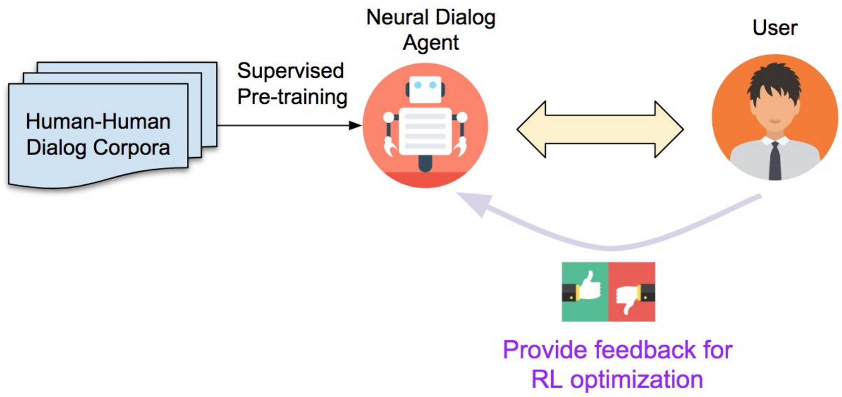

Figure 6.1: Interactive learning with human feedback.

in the calculated reward for policy update, we establish a baseline reward using the state value function $V(s_{A,k})$. This function is a feed-forward neural network with a single hidden layer and a regression function output. The final reward R_k used for RL training is thus the reward advantage over the baseline: $R_k = \sum_{t=k}^{K} \gamma^{t-k} r_t - V(s_{A,k})$.

6.1.2 Optimization

We optimize the dialog agent using REINFORCE [3] with the reward described above. The objective function is written as:

$$J_k(\theta_A) = \mathbb{E}_{\theta_A}[R_k] = \mathbb{E}_{\theta_A}\left[\sum_{t=k}^{K} \gamma^{t-k} r_t - V(s_{A,k})\right] \tag{6.1}$$

With likelihood ratio gradient estimator, the gradient of $J_k(\theta_A)$ can be derived with:

$$\begin{aligned} \nabla_{\theta_A} J_k(\theta_A) &= \nabla_{\theta_A} \mathbb{E}_{\theta_A}[R_k] \\ &= \sum_{a_{A,k} \in \mathcal{A}} \pi_{\theta_A}(a_{A,k}|\cdot) \nabla_{\theta_A} \log \pi_{\theta_A}(a_{A,k}|\cdot) R_k \\ &= \mathbb{E}_{\theta_A}[\nabla_{\theta_A} \log \pi_{\theta_A}(a_{A,k}|\cdot) R_k] \end{aligned} \tag{6.2}$$

During model training, we encourage the agent to explore the dialog action space by sampling agent actions from the softmax policy network output. During model evaluation, we let the agent to use the greedy policy by selecting action that has the highest probability score.

6.1.3 Experiments

We evaluate the proposed method on the dialog corpus in movie booking domain described in Chapter 5.3.1. The state size of the dialog-level and utterance-level LSTM is as 200 and 150. Hidden layer size of the policy network is set as 100. We used randomly initialized word embedding of size 300. Adam optimization method [46] with initial learning rate of 1e-3 is used for mini-batch training. Dropout rate of 0.5 is applied during training to prevent the model from over-fitting.

In the evaluation against a user simulator, we take a task-oriented dialog as successful if the goal slot values estimated by the state tracker fully match to the user's true goal values, and the system is able to offer an entity which is finally accepted by the user. Maximum allowed number of dialog turn is set as 15. A positive reward of +15.0 is given to the agent at the end of a success dialog, and a zero reward is given in a failure case. We apply a step penalty of -1.0 for each turn to encourage shorter dialog in completing the task.

Figure 6.2 shows the RL curves of the proposed model on dialog task success rate and average dialog turn size. Evaluation is based on dialog simulations between our proposed end-to-end dialog agent and the rule based user simulator. This is different from the evaluations based on fixed dialog corpora as in Table 5.4 and 5.5. The policy gradient based RL training is performed on top of the supervised training model. We compare models with two RL training settings, the end-to-end training and the policy-only training, to the baseline supervised learning (SL) model.

As shown in Figure 6.2(a), the SL model performs poorly during user interaction, indicating the limited generalization capability of the SL model to unseen dialog state. Any mistake made by the agent during user interaction may lead to deviation of the dialog from the training dialog trajectories and states. The SL agent does not know how to recover from an

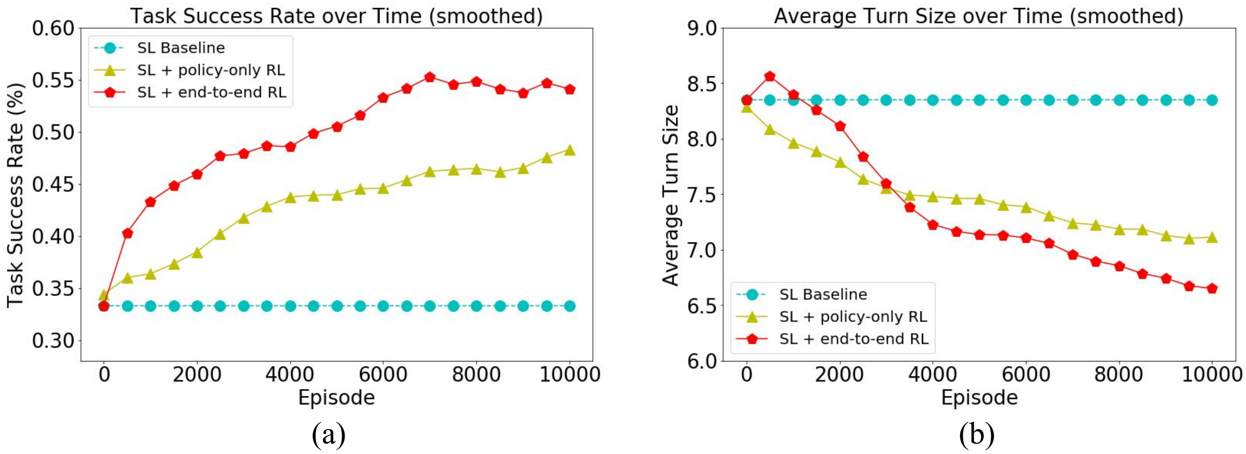

Figure 6.2: RL curves on (a) dialog task success rate and (b) average dialog turn size.

unknown state, which leads to final task failure. RL model training, under both end-to-end learning and policy-only learning settings, continuously improves the task success rate with the growing number of user interactions. We see clear advantage of performing end-to-end model update in achieving higher dialog task success rate comparing to only updating the policy network during interactive learning.

Figure 6.2(b) shows the learning curves for the average number of turns in successful dialogs. We observe decreasing number of dialog turns along the growing number of interactive learning episodes. This shows that the dialog agent learns better strategies to successfully complete the task in fewer numbers of turns. Similar to the results for task success rate, the end-to-end training model outperforms the model with policy-only optimization during RL training, achieving lower average number of dialog turns in successfully completing a task.

Human Evaluation

We further evaluate our proposed method with human judges recruited via Amazon Mechanical Turk. Each judge is asked to read a dialog between our model and the user simulator and rate each system turn on a scale of 1 (frustrating) to 5 (optimal way to help the user). Each turn is rated by 3 different judges. We rate the three models with 100 dialogs each: (i) the SL model, (ii) SL with policy-only RL model, and (iii) SL with end-to-end RL model.

Table 6.1 lists the mean and standard deviation of human evaluation scores over all system turns: end-to-end optimization with RL clearly improves the quality of the model according to human judges.

Model	**Score**
SL	3.987 ± 0.086
SL + policy-only RL	4.261 ± 0.089
SL + end-to-end RL	4.394 ± 0.087

Table 6.1: Human evaluation results. Mean and standard deviation of crowd worker scores (between 1 to 5).

6.2 Dialog Learning with Human Teaching and Feedback

In the previous section, we described how we can let the neural dialog agent to learn interactively from user feedback with deep reinforcement learning methods. The system optimization with RL is performed based on the supervised pretraining model using human-human or human-machine dialog corpus. Such hybrid supervised and reinforcement learning method is also used in several other recent works on dialog and policy learning, as training dialog policy online from scratch with RL typically requires a large number of interactive learning sessions before an agent can reach a satisfactory performance level.

A potential drawback with such pretraining approach is that the model may suffer from the mismatch of dialog state distributions between supervised training and interactive learning stages. While interacting with users, the agent's response at each turn has a direct influence on the distribution of dialog state that the agent will operate on in the upcoming dialog turns. If the agent makes a small mistake and reaches an unfamiliar state, it may not know how to recover from it and get back to a normal dialog trajectory. This is because such recovery situation may be rare for good human agents and thus are not well covered in the supervised training corpus. This will result in compounding errors in a dialog which may

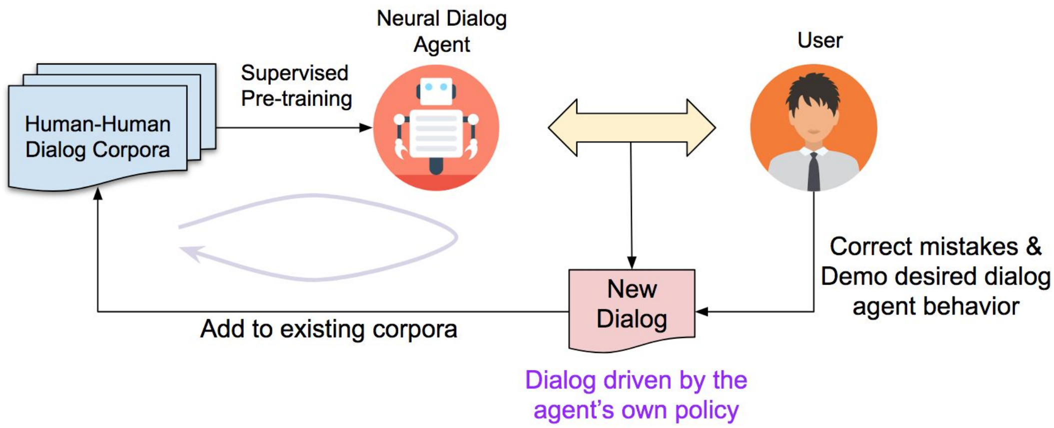

Figure 6.3: Interactive learning with human teaching.

lead to failure of a task. RL exploration might finally help to find corresponding actions to recover from a bad state, but the search process can be very inefficient.

To ameliorate the effect of dialog state distribution mismatch between offline training and RL interactive learning, we propose a hybrid imitation and reinforcement learning method. We first let the agent to interact with users using its own policy learned from supervised pretraining. When an agent makes a mistake, we ask users to correct the mistake by demonstrating the agent the right actions to take at each turn. This user corrected dialog sample, which is guided by the agent's own policy, is then added to the existing training corpus. We fine-tune the dialog policy with this dialog sample aggregation [24] and continue such user teaching process for a number of cycles. Since asking for user teaching at each dialog turn is costly, we want to reduce this user teaching cycles as much as possible and continue the learning process with RL by collecting simple forms of user feedback (e.g. a binary feedback, positive or negative) only at the end of a dialog.

6.2.1 Imitation Learning with Human Teaching

Once obtaining a supervised training dialog agent, we let the agent to learn interactively from users by conducting task-oriented dialogs. Supervised learning succeeds when training and test data distributions match. During the agent's interaction with users, any mistake

Algorithm 3 Dialog Learning with Human Teaching and Feedback

Train model end-to-end on dialog samples D with MLE and obtain policy $\pi_{\theta_A}(a|s)$
for learning iteration $k = 1 : K$ **do**
    Run $\pi_{\theta_A}(a|s)$ with user to collect new dialog samples D_π
    Ask user to correct the mistakes in the tracked user's goal for each dialog turn in D_π
    Add the newly labeled dialog samples to the existing corpora: $D \leftarrow D \cup D_\pi$
    Train model end-to-end on D and obtain an updated policy $\pi_{\theta_A}(a|s)$
end for
for learning iteration $k = 1 : N$ **do**
    Run $\pi_{\theta_A}(a|s)$ with user for a new dialog
    Collect user feedback as reward r
    Update model end-to-end and obtain an updated policy $\pi_{\theta_A}(a|s)$
end for

made by the agent or any deviation in the user's behavior may lead to a different dialog state distribution than the one that the supervised learning agent saw during offline training. A small mistake made by the agent due to this covariate shift [22, 24] may lead to compounding errors which finally lead to failure of a task. To address this issue, we propose a dialog imitation learning method which allows the dialog agent to learn from human teaching. We let the supervised training agent to interact with users using its learned dialog policy $\pi_{\theta_A}(a|s)$. With this, we collect additional dialog samples that are guided by the agent's own policy, rather than by the expert policy as those in the supervised training corpora. When the agent makes mistakes, we ask users to correct the mistakes and demonstrate the expected actions and predictions for the agent to make. Such user teaching precisely addresses the limitations of the currently learned dialog model, as these newly collected dialog samples are driven by the agent's own policy. Specifically, in this study we let an expert user to correct the mistake made by the agent in tracking the user's goal at the end of each dialog turn. This new batch of annotated dialogs are then added to the existing training corpus. We start the next round of supervised model training on this aggregated corpus to obtain an updated dialog policy and continue this dialog imitation learning cycles.

6.2.2 Experiments

Evaluations of interactive learning with imitation and reinforcement learning are made on metrics of (1) task success rate, (2) dialog turn size, and (3) dialog state tracking accuracy. Figures 6.4, 6.5, and 6.6 show the learning curves for the three evaluation metrics. In addition, we compare model performance on task success rate using two different RL training settings, the end-to-end training and the policy-only training, to show the advantages of performing end-to-end system optimization with RL.

Task Success Rate

As shown in the learning curves in Figure 6.4, the SL model performs poorly. This might largely due to the compounding errors caused by the mismatch of dialog state distribution between offline training and interactive learning. We use an extended set of user NLG templates during interactive evaluation. Many of the test NLG templates are not seen by the supervised training agent. Any mistake made by the agent in understanding the user's request may lead to compounding errors in the following dialog turns, which cause final task failure. The red curve (`SL + RL`) shows the performance of the model that has RL applied on the supervised pretraining model. We can see that interactive learning with RL using a weak form of supervision from user feedback continuously improves the task success rate with the growing number of user interactions. We further conduct experiments in learning dialog model from scratch using only RL (i.e. without supervised pretraining), and the task success rate remains at a very low level after 10K dialog simulations. We believe that it is because the dialog state space is too complex for the agent to learn from scratch, as it has to learn a good NLU model in combination with a good policy to complete the task. The yellow curve (`SL + IL 500 + RL`) shows the performance of the model that has 500 episodes of imitation learning over the SL model and continues with RL optimization. It is clear from the results that applying imitation learning on supervised training model efficiently improves

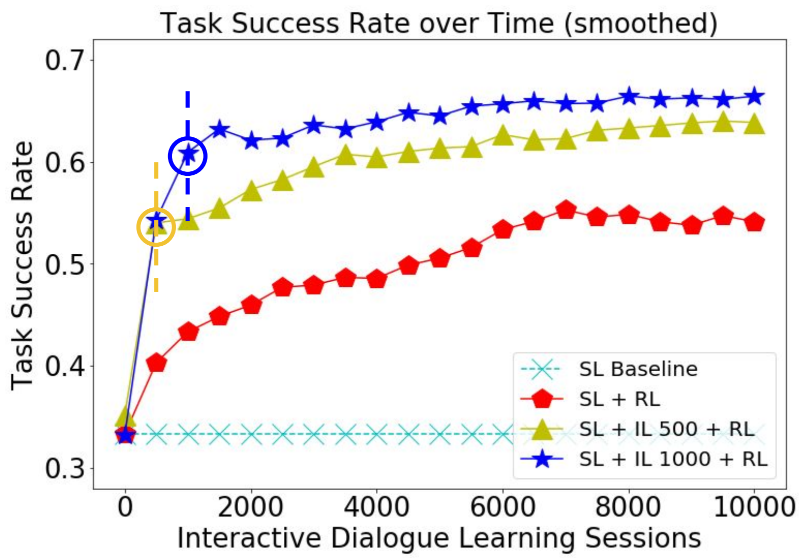

Figure 6.4: Interactive learning curves on task success rate.

task success rate. RL optimization after imitation learning increases the task success rate further. The blue curve (`SL + IL 1000 + RL`) shows the performance of the model that has 1000 episodes of imitation learning over the SL model and continues with RL. Similarly, it shows hints that imitation learning may effectively adapt the supervised training model to the dialog state distribution during user interactions.

Average Dialog Turn Size

Figure 6.5 shows the curves for the average turn size of successful dialogs. We observe decreasing number of dialog turns in completing a task along the growing number of interactive learning sessions. This shows that the dialog agent learns better strategies in successfully completing the task with fewer number of dialog turns. The red curve with RL applied directly after supervised pretraining model gives the lowest average number of turns at the end of the interactive learning cycles, comparing to models with imitation dialog learning. This seems to be contrary to our observation in Figure 6.4 that imitation learning with human teaching helps in achieving higher task success rate. By looking into the generated dialogs, we find that the `SL + RL` model can handle easy tasks well but fails to complete more chal-

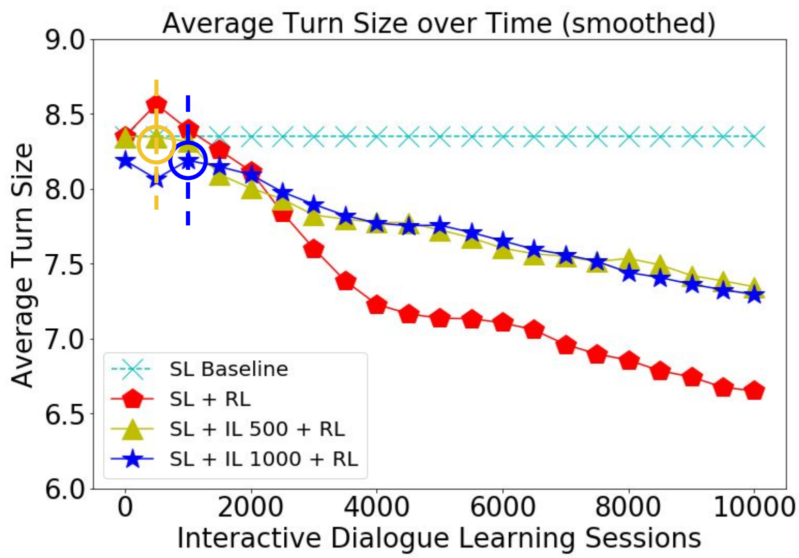

Figure 6.5: Interactive learning curves on average dialog turn size.

lenging tasks. Such easy tasks typically can be handled with fewer number of turns, which result in the low average turn size for the `SL + RL` model. On the other hand, the imitation plus RL models attempt to learn better strategies to handle those more challenging tasks, resulting in higher task success rates and also slightly increased dialog length comparing to `SL + RL` model.

Dialog State Tracking Accuracy

Similar to the results on task success rate, we see that imitation learning with human teaching quickly improves dialog state tracking accuracy in just a few hundred interactive learning sessions. The joint slots tracking accuracy in the evaluation of SL model using fixed corpus is 84.57% as in Table 5.5. The accuracy drops to 50.51% in the interactive evaluation with the introduction of new NLG templates. Imitation learning with human teaching effectively adapts the neural dialog model to the new user input and dialog state distributions, improving the DST accuracy to 67.47% after only 500 imitation dialog learning sessions. Another encouraging observation is that RL on top of SL model and IL model not only improves task success rate by optimizing dialog policy, but also further improves dialog state

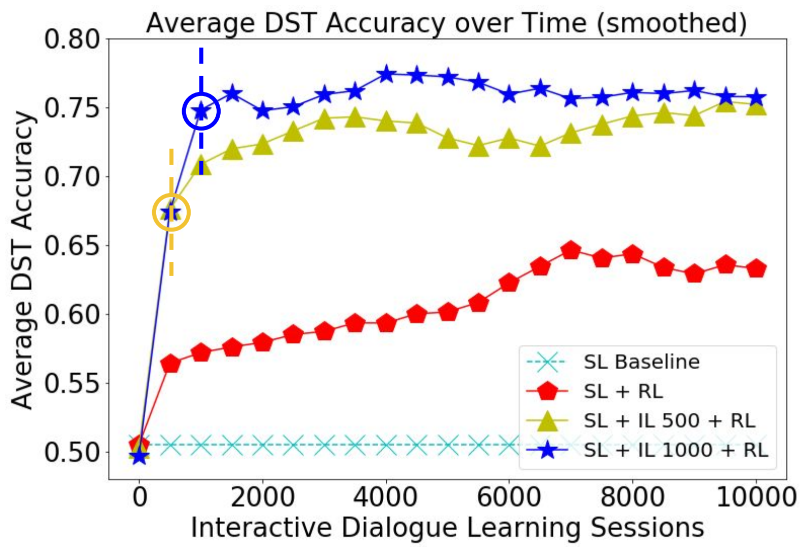

Figure 6.6: Interactive learning curves on dialog state tracking accuracy.

tracking performance. This shows the benefits of performing end-to-end optimization of the neural dialog model with RL during interactive learning.

Human Evaluation

We further evaluate the proposed method with human judges recruited via Amazon Mechanical Turk in the similar setting described in section 6.1. As shown in Table 6.2, performing interactive learning with imitation and reinforcement learning clearly improves the quality of the model according to human judges.

Model	**Score**
SL	3.987 ± 0.086
SL + IL 1000	4.378 ± 0.082
SL + IL 1000 + RL	4.603 ± 0.067

Table 6.2: Human evaluation results on dialog learning methods with human teaching and feedback.

6.3 Conclusion

In this chapter, we presented how we can train task-oriented dialog systems through user interactions, where the agent can improve itself via communicating with users and learning from the mistake it makes. We designed a hybrid imitation and reinforcement learning method, with which a supervised training agent can continuously improve itself by learning from user teaching and feedback with imitation and reinforcement learning. We evaluated the proposed learning method in both a simulated environment against user simulators and a real user evaluation setting. Experimental results showed that the proposed neural dialog agent can effectively learn from user teaching and improve task success rate with imitation learning. Applying reinforcement learning with user feedback after imitation learning with user teaching improved the model performance further, not only on the dialog policy but also on the dialog state tracking in the end-to-end learning framework.

Chapter 7

Dialog Learning with Adversarial Training

In the previous chapter, we discussed how we can let the dialog agent to learn interactively from users. Such training setting assumes the model has access to a reward signal at the end of a dialog, either in the form of a binary user feedback or a continuous user score. A challenge with such setting is that user feedback may be inconsistent [78] and may not always be available in practice. Furthermore, online dialog policy learning with RL usually suffers from sample efficiency issue [99], which requires an agent to make a large number of feedback queries to users.

To reduce the demand for high quality user feedback in online policy learning, solutions have been proposed to design or to learn a reward function that can be used to generate a reward in approximation to a user feedback. Designing a good reward function is not easy [8] as it typically requires strong domain knowledge. El Asri et al. [39] proposed a learning based reward function that is trained with task completion transfer learning. Su et al. [78] proposed an online active learning method for reward estimation using Gaussian process classification. These methods still require annotations of dialog ratings by users, and thus may also suffer from the rating consistency and learning efficiency issues.

To address the above discussed challenges, we investigate the effectiveness of learning dialog rewards directly from dialog samples. Inspired by the success of adversarial training in computer vision [53] and natural language generation [89], we propose an adversarial learning method for task-oriented dialog systems. We jointly train two models, a generator that interacts with the environment to produce task-oriented dialogs, and a discriminator that marks a dialog sample as being successful or not. The generator is a neural network based task-oriented dialog agent. The environment that the dialog agent interacts with is the user. Quality of a dialog produced by the agent and the user is measured by the likelihood that it fools the discriminator to believe that the dialog is a successful one conducted by a human agent. We treat dialog agent optimization as a reinforcement learning problem. The output from the discriminator serves as a reward to the dialog agent, pushing it towards completing a task in a way that is indistinguishable from how a human agent completes it.

In this chapter, we will discuss how the adversarial learning reward function compares to designed reward functions in learning a good dialog policy. We analyze the impact of the size of annotated dialog samples to the effectiveness of dialog adversarial learning. We further discuss the covariate shift issue in interactive adversarial learning and show how we can address that with partial access to user feedback. This chapter is based on the work presented in [105].

7.1 Model

7.1.1 Neural Dialog Agent

The generator is the neural network based task-oriented dialog agent that described in Chapter 5.1. The agent uses an LSTM recurrent neural network to model the sequence of turns in a dialog. At each turn, the agent takes a best system action conditioning on the current dialog state. A continuous form dialog state is maintained in the LSTM state $s_{A,k}$. At each dialog turn k, user input U_k and previous system output A_{k-1} are firstly encoded to continuous representations. The user input can either in the form of a dialog act or a natural language utterance. We use dialog act form user input in this experiment. The dialog act representation is obtained by concatenating the embeddings of the act and the slot-value pairs. If natural language form of input is used, we can encode the sequence of words using a bidirectional RNN and take the concatenation of the last forward and backward states as the utterance representation, similar to [80] and [92]. The belief tracker updates its estimation of the user's goal by maintaining a probability distribution $P(l_{A,k}^m$ over candidate values for each of the tracked goal slot type $m \in M$. Conditioning on the current dialog state $s_{A,k}$, the probability distribution of estimated user goal slot values v_k, and the encoding of the information retrieved from external sources E_k, the policy network produce a best action for the system to take.

7.1.2 Dialog Reward Estimator

The discriminator model is a binary classifier that takes in a dialog with a sequence of turns and outputs a label indicating whether the dialog is a successful one or not. The logistic function returns a probability of the input dialog being successful. The discriminator model design is as shown in Figure 7.1. We use a bidirectional LSTM to encode the sequence of turns. At each dialog turn k, input to the discriminator model is the concatenation of (1) encoding of the user input U_k, (2) encoding of the query result summary E_k, and (3) encoding of agent output A_k. The discriminator LSTM output at each step k, h_k, is a concatenation of the forward LSTM output $\overrightarrow{h_k}$ and the backward LSTM output $\overleftarrow{h_k}$: $h_k = [\overrightarrow{h_k}, \overleftarrow{h_k}]$.

Once obtaining the discriminator LSTM state outputs $\{h_1, \dots, h_K\}$, we experiment with four different methods in combining these state outputs to generated the final dialog representation d for the binary classifier:

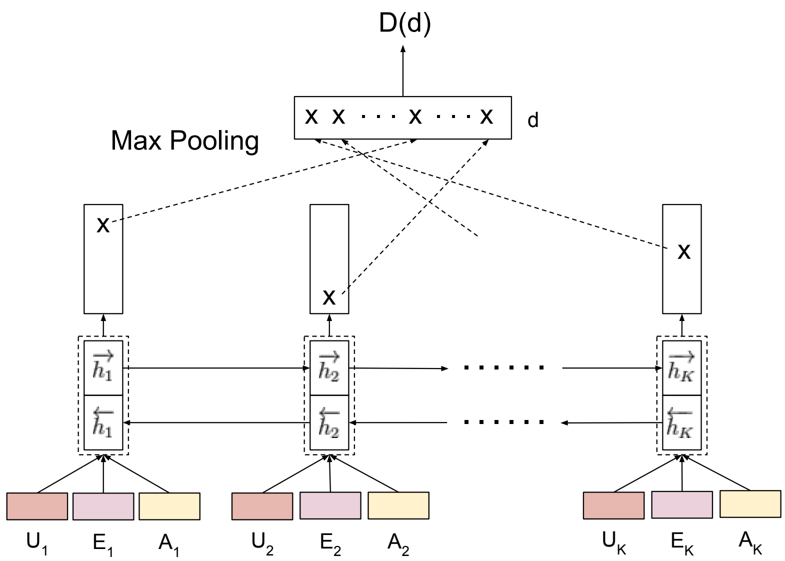

Figure 7.1: Design of the dialog reward estimator: Bidirectional LSTM with max pooling.

BiLSTM-last Produce the final dialog representation d by concatenating the last LSTM state outputs from the forward and backward directions: $d = [\overrightarrow{h_K}, \overleftarrow{h_1}]$

BiLSTM-max Max-pooling. Produce the final dialog representation d by selecting the maximum value over each dimension of the LSTM state outputs.

BiLSTM-avg Average-pooling. Produce the final dialog representation d by taking the average value over each dimension of the LSTM state outputs.

BiLSTM-attn Attention-pooling. Produce the final dialog representation d by taking the weighted sum of the LSTM state outputs. The weights are calculated with attention mechanism:

$$d = \sum_{k=1}^{K} \alpha_k h_k \tag{7.1}$$

and

$$\alpha_k = \frac{\exp(e_k)}{\sum_{t=1}^{K} \exp(e_t)}, \quad e_k = g(h_k) \tag{7.2}$$

g a feed-forward neural network with a single output node. Finally, the discriminator produces a value indicating the likelihood the input dialog being a successful one:

$$D(d) = \sigma(W_o d + b_o) \tag{7.3}$$

where W_o and b_o are the weights and bias in the discriminator output layer. σ is a logistic function.

7.1.3 Adversarial Model Training

Once we obtain a dialog sample initiated by the agent and a dialog reward from the reward function, we optimize the dialog agent using REINFORCE [3] with the given reward. The reward $D(d)$ is only received at the end of a dialog, i.e. $r_K = D(d)$. We discount this final reward with a discount factor $\gamma \in [0, 1)$ to assign a reward R_k to each dialog turn. The objective function can thus be written as $J_k(\theta_A) = \mathbb{E}_{\theta_A}[R_k] = \mathbb{E}_{\theta_A}\left[\sum_{t=k}^{K} \gamma^{t-k} r_t - V(s_k)\right]$, with $r_k = D(d)$ for $k = K$ and $r_k = 0$ for $k < K$. $V(s_{A,k})$ is the state value function which serves as a baseline value. The state value function is a feed-forward neural network with a single-node value output. Similar to the optimization method described in Chapter 6.1.2, we optimize the generator parameter θ_A to maximize $J_k(\theta_A)$. With likelihood ratio gradient estimator, the gradient of $J_k(\theta_A)$ can be derived with:

$$\begin{aligned}
\nabla_{\theta_A} J_k(\theta_A) &= \nabla_{\theta_A} \mathbb{E}_{\theta_A}[R_k] \\
&= \sum_{a_{A,k} \in \mathcal{A}} \pi_{\theta_A}(a_{A,k}|\cdot) \nabla_{\theta_A} \log \pi_{\theta_A}(a_{A,k}|\cdot) R_k \\
&= \mathbb{E}_{\theta_A}[\nabla_{\theta_A} \log \pi_{\theta_A}(a_{A,k}|\cdot) R_k]
\end{aligned} \tag{7.4}$$

where $\pi_{\theta_A}(a_{A,k}|\cdot) = P(a_{A,k}|s_{A,k}, v_k, E_k; \theta_A)$. The expression above gives us an unbiased gradient estimator. We sample agent action $a_{A,k}$ following a softmax policy at each dialog turn and compute the policy gradient. At the same time, we update the discriminator parameter θ_D to maximize the probability of assigning the correct labels to the successful

Algorithm 4 Adversarial Learning for Task-Oriented Dialog

Required: dialog corpus S_{demo}, user simulator U, dialog agent A, adversarial reward estimator D
Pretrain a dialog agent (i.e. the generator) A on dialog corpora S_{demo} with MLE
Simulate dialogs S_{simu} between U and A
Sample successful dialogs $S_{(+)}$ and random dialogs $S_{(-)}$ from $\{S_{demo}, S_{simu}\}$
Pretrain a reward function (i.e. the discriminator) D with $S_{(+)}$ and $S_{(-)}$
for number of training iterations **do**
  for G-steps **do**
    Simulate dialogs S_b between U and A
    Compute reward r for each dialog in S_b with D
    Update A with reward r
  end for
  for D-steps **do**
    Sample dialogs $S_{(b+)}$ from $S_{(+)}$
    Update D with $S_{(b+)}$ and S_b (with S_b as negative examples)
  end for
end for

dialog from human demonstration and the dialog conducted by the machine agent:

$$\nabla_{\theta_D}\Big[\mathbb{E}_{d\sim\theta_{demo}}\left[log(D(d))\right] + \mathbb{E}_{d\sim\theta_A}\left[log(1 - D(d))\right]\Big] \tag{7.5}$$

We continue to update both the dialog agent and the reward function via dialog simulation or real user interaction until convergence.

7.2 Experiments

7.2.1 Evaluation Setting

In this section, we present and analyze the empirical evaluation results. During dialog simulation, a dialog is marked as successful if the agent's belief tracking outputs fully match the informable [33] user goal slot values, and all user requested slots are fulfilled. This is the same evaluation criteria as used in [101] and [91]. It is important to note that such dialog success signal is usually not available during real user interactions, unless we explicitly ask users to provide this feedback.

We first compare dialog agent trained using the proposed adversarial reward to those using human designed reward and using oracle reward. We then discuss the impact of discriminator model design and model pretraining on the adversarial learning performance. Last but not least, we discuss the potential issue of covariate shift during interactive adversarial learning and show how we address that with partial access to user feedback.

7.2.2 Results and Analysis

Comparison to Other Reward Types

We first compare the performance of dialog agent using adversarial reward to those using designed reward and oracle reward on dialog success rate. Designed reward refers to reward function that is designed by humans with domain knowledge. A dialog is marked as successful if the agent's belief tracking outputs fully match the informable [33] user goal slot values, and all user requested slots are fulfilled. Based on this dialog success criteria, we can *design* a reward function for RL policy learning like below:

- +1 for each informable slot that is correctly estimated by the agent at the end of a dialog.
- If ALL informable slots are tracked correctly, +1 for each requestable slot successfully handled by the agent.

In addition to the comparison to human the reward, we further compare to the case of using oracle reward during agent policy optimization. Using oracle reward refers to having access to the final dialog success status. We apply a reward of +1 for a successful dialog, and a reward of 0 for a failed dialog. Performance of the agent using oracle reward serves as an upper-bound for those using other types of reward. For the learning with adversarial rewards, we use BiLSTM-max as the discriminator model. During RL training, we normalize

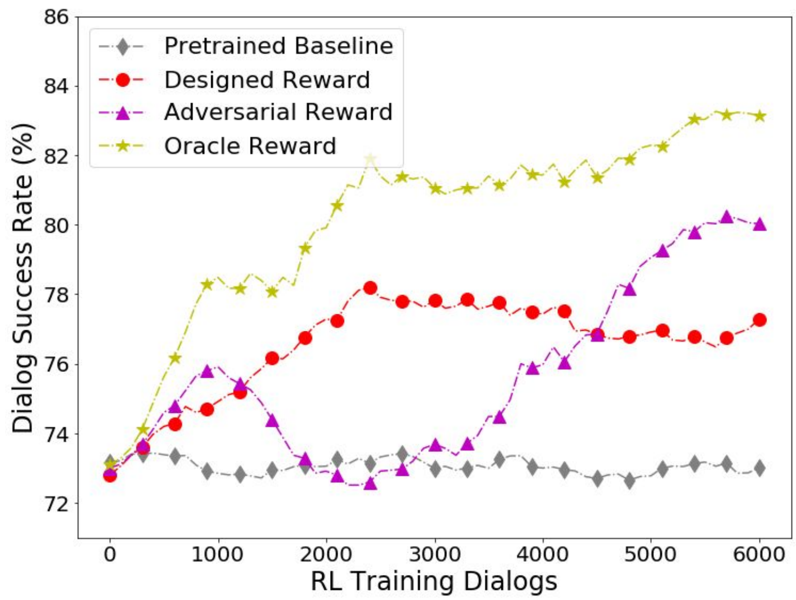

Figure 7.2: RL policy optimization performance comparing with adversarial reward, designed reward, and oracle reward.

the rewards produced by different reward functions.

Figure 7.2 show the RL learning curves for models trained using different reward functions. The dialog success rate at each evaluation point is calculated by averaging over the success status of 1000 dialog simulations at that point. The pretrain baseline in the figure refers to the supervised pretraining model. This model does not get updated during interactive learning, and thus the curve stays flat during the RL training cycle. As shown in these curves, all the three types of reward functions lead to improved dialog success rate along the interactive learning process. The agent trained with designed reward falls behind the agent trained with oracle reward by a large margin. This shows that the reward designed with domain knowledge may not fully align with the final evaluation metric. Designing a reward function that can provide an agent enough supervision signal and also well aligns the final system objective is not a trivial task [96]. In practice, it is often difficult to exactly specify what we expect an agent to do, and we usually end up with simple and imperfect measures. In our experiment, agent using adversarial reward achieves a 7.4% improvement on dialog success rate over the supervised pretraining baseline at the end of 6000 interactive dialog learning episodes, outperforming that using the designed reward (4.2%). This shows the advantage of performing adversarial training in learning directly from expert demonstrations and in addressing the challenge of designing a proper reward function. Another important point we observe in our experiments is that RL agents trained with adversarial reward, although enjoy higher performance in the end, suffer from larger variance and instability on model performance during the RL training process, comparing to agents using human designed reward. This is because during RL training the agent interfaces with a moving target, rather than a fixed objective measure as in the case of using the designed reward or oracle reward. The model performance gradually becomes stabilized when both the dialog agent and the reward model are close to convergence.

Impact of Discriminator Model Design

We study the impact of different discriminator model designs on the adversarial learning performance. We compare the four pooling methods described in section 7.1.2 in producing the final dialog representation. Table 7.1 shows the offline evaluation results on 1000 simulated test dialog samples. Among the four pooling methods, max-pooling on bidirectional LSTM outputs achieves the best classification accuracy in our experiment. Max-pooling also assigns the highest probability to successful dialogs in the test set comparing to other pooling methods. Attention-pooling based LSTM model achieves the lowest performance across all the three offline evaluation metrics in our study. This is probably due to the limited number of training samples we used in pretraining the discriminator. Learning good attentions usually requires more data samples and the model may thus overfit the small training set. We observe similar trends during interactive learning evaluation that the attention-based discriminator leads to divergence of policy optimization more often than the other three pooling methods. Max-pooling discriminator gives the most stable performance during our interactive RL training.

Model	Prediction Accuracy	Success Prob.	Fail Prob.
BiLSTM-last	0.674	0.580	0.275
BiLSTM-max	**0.706**	**0.588**	0.272
BiLSTM-avg	0.688	0.561	**0.268**
BiLSTM-attn	0.652	0.541	0.285

Table 7.1: Performance of different discriminator model design, on prediction accuracy and probabilities assigned to successful and failed dialogs.

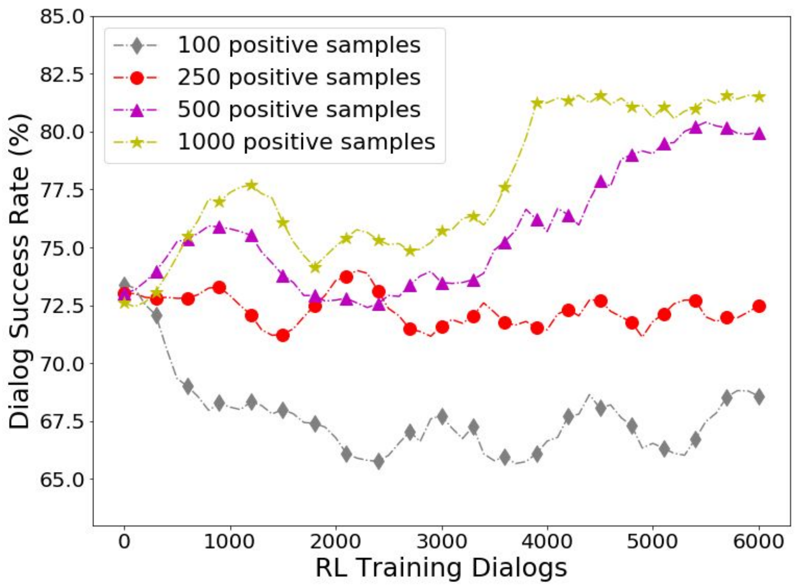

Figure 7.3: Impact of discriminator training sample size on RL dialog learning performance.

Impact of Annotated Dialogs for Discriminator Training

Annotating dialog samples for model training requires additional human efforts. We investigate the impact of the size of the annotated dialog samples on discriminator model training. The number of annotated dialog samples required for learning a good discriminator depends mainly on the complexity of a task. Given the rather simple nature of the slot filling based DSTC2 restaurant search task, we experiment with annotating 100 to 1000 discriminator training samples. We use BiLSTM-max discriminator model in these experiments. The adversarial RL training curves with different levels of discriminator training samples are shown in Figure 7.3. As these results illustrate, with 100 annotated dialogs as positive samples

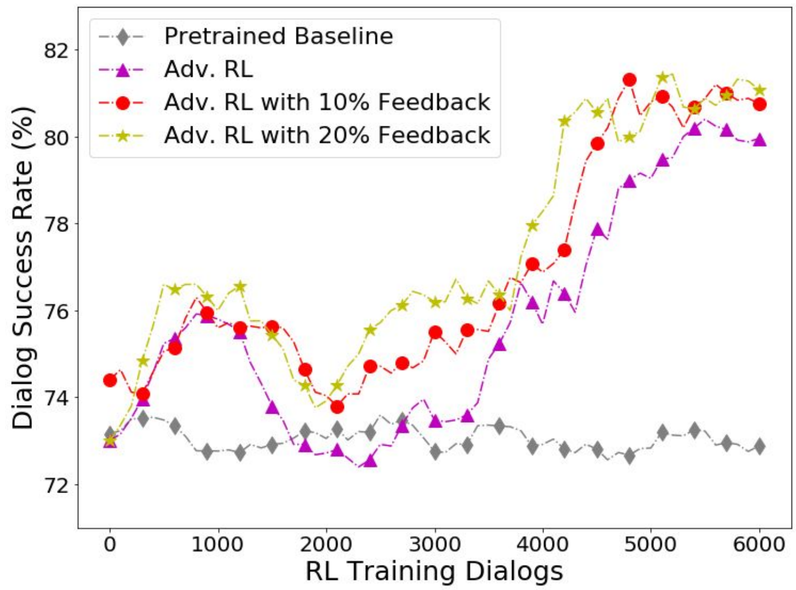

Figure 7.4: Addressing covariate shift in online adversarial dialog learning with partial access to user feedback.

for discriminator training, the discriminator is not able to produce dialog rewards that are useful in learning a good policy. Learning with 250 positive samples does not lead to concrete improvement on dialog success rate neither. With the growing number of annotated samples, the dialog agent becomes more likely to learn a better policy, resulting in higher dialog success rate at the end of the interactive learning sessions.

Partial Access to User Feedback

A potential issue with RL based interactive adversarial learning is the covariate shift [22, 68] problem. Part of the positive examples for discriminator training are generated based on the supervised pretraining dialog policy before the interactive learning stage. During interactive RL training, the agent's policy gets updated. The newly generated dialog samples based on the updated policy may be equally good comparing to the initial set of positive dialogs, but they may look very different. In this case, the discriminator is likely to give these dialogs low rewards as the pattern presented in these dialogs is different to what the discriminator is initially trained on. The agent will thus be discouraged to produce such type of successful dialogs in the future with these negative rewards. To address such covariate shift issue, we design a DAgger [24] style imitation learning method to the dialog adversarial learning. We assume that during interactive learning with users, occasionally we can receive feedback from users indicating the quality of the conversation they had with the agent. We then add those dialogs with good feedback as additional training samples to the pool of positive dialogs used in discriminator model training. With this, the discriminator can learn to assign high rewards to such good dialogs in the future. In our empirical evaluation, we experiment with the agent receiving positive feedback 10% and 20% of the time during its interaction with users. The experimental results are shown in Figure 7.4. As illustrated in these curves, the proposed DAgger style learning method can effectively improve the dialog adversarial learning with RL, leading to higher dialog success rate.

7.3 Conclusion

In this chapter, we discussed how we can apply adversarial learning to effectively estimate dialog rewards which are used to guide dialog model optimization during user interactions. The proposed method is an attempt towards addressing the rating consistency and learning efficiency issues in online dialog policy learning with user feedback. We showed that with limited number of annotated dialogs, the proposed adversarial learning method can effectively learn a reward function and use that to guide policy optimization with policy gradient based reinforcement learning. In the experiment in a restaurant search domain, we showed that the proposed adversarial learning method achieves advanced dialog success rate comparing to baseline methods using other forms of reward. We further discussed the covariate shift issue during interactive adversarial learning and showed how we can address it with partial access to user feedback.

Chapter 8

Dialog Learning with Real and Imagined Experiences

In previous chapters, we discussed how we can continuously improve the dialog agent during its interaction with users. Current model-free reinforcement learning methods for dialog policy optimization suffers from sample efficiency issue [99, 31]. This typically requires an agent to perform a large number of interactive learning sessions by querying users for feedback before it can reach a satisfactory performance level. Such learning efficiency issue poses several concerns in real world dialog system deployment. A user may easily get frustrated with the inferior system performance at the beginning of the learning stage. Dialog samples collected in such situation may severely deviate from the dialog distribution that the system is initially trained on and thus be of less value for guiding dialog policy optimization. We argue that to improve the online learning efficiency, an intelligent dialog agent should not only be able to passively receive signals from the environment (i.e. the user) and learn to act on it, but also to be able to understand the dynamics of the environment and plan accordingly. This is also how we human beings learn from the world.

In this chapter, we present a integrated learning method by combining learn-from-user and learn-from-imagination approaches to address the sample efficiency issue in online interactive learning with users. In addition to learning to act in different dialog scenarios or states towards task completion, an agent should also learn to model the user dynamics and predict their behaviors in conversations. Such user model in the agent's mind can help the agent to "imagine" future dialogs that mimic the conversations between the agent and a real user. Between consecutive learning sessions with real users, we perform a number of learn-from-imagination training cycles where an agent simulates the conversations with the user model in mind and learns from such dialog rollouts. The intuition behind this integrated learning method is that we want to enforce the dialog agent to fully digest the knowledge learned from real user interactions by simulating similar dialog scenarios internally. By imagining such conversations and learning from them, the agent may potentially learn more effectively and reduce the number of real user interactive learning cycles.

We will first describe how we can construct a user model that may effectively mimic user behavior in task-oriented dialogs. With such user model in mind, the dialog agent can imagine the conversation with a real user by simulating dialogs internally with the user model. We will further discuss how we can effectively combine the learn-from-user and learn-from-imagination experiences to improve online learning efficiency with users. This chapter is based on the work presented in [91, 106].

8.1 Statistical User Modeling

In RL based dialog policy learning, a user simulator [15, 64, 72] is usually used to train dialog agents in an interactive environment. Quality of such user simulator has a direct impact on the effectiveness of dialog policy learning. Designing a reliable user simulator or user model, however, is not trivial, often as difficult as building a good dialog agent. User simulators used in most of the recent RL based dialog models [82, 72, 90] are designed with expert knowledge and complex rules. Maintaining such handcrafted user model is a big challenge, especially when the dialog scenarios become more complex. Moreover, such user models lack the capability to learn from interactions. They can only be upgraded by developers who are equipped with deep domain knowledge.

We present a neural network based statistical user model. The model is data driven and can be configured with minimum knowledge of domain ontology (e.g. slot types in a task domain). In training such user model, we design a two-step learning method. We first bootstrap a basic neural user model by learning directly from dialog corpora with supervised training. We then improve the user model together with the neural dialog agent by simulating task-oriented dialogs between them and iteratively optimizing their dialog policies with deep RL. We present the user model design and training details, followed by empirical evaluations.

8.1.1 Model Design

Figure 8.1 shows the design of our user model. The user model is optimized towards mimicking the behavior of a real user in conducting task-oriented dialogs, where a user acts according to the dialog context and his goal in the task.

User State Representation

Our user model employs an LSTM recurrent neural network in maintaining the dialog state over multiple dialog turns. At the kth turn of a dialog, the user model takes in the goal encoding G_k, the previous user output encoding U_{k-1}, and the current turn agent input encoding A_k, to update its internal state conditioning on the previous dialog state $s_{U,k-1}$:

$$s_{U,k} = \text{LSTM}_\text{U}(s_{U,k-1},\ [G_k,\ U_{k-1},\ A_k]) \tag{8.1}$$

The user output is represented with dialog act representation, which consist of a user action and slot-value pairs.

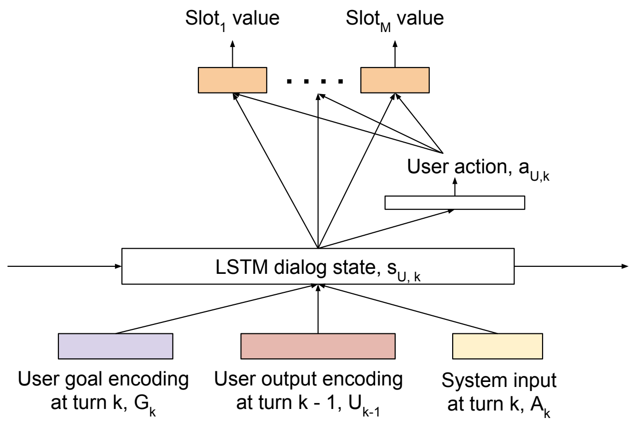

Figure 8.1: Neural user model architecture.

User Goal Encoding

Similar to how user's goal is defined in the Dialog State Tracking Challenges [33, 79], we define a user's goal G_k using a list of informable and requestable slots. Informable slots are attributes that users can provide a value for in describing their goal. Requestable slots are attributes that a user may ask the value of. We treat informable slots as inputs that can take multiple discrete values and treat requestable slots as inputs that take binary values (i.e. a slot is either requested or not). Each informable and requestable slot value is first mapped to an embedding space. The final user goal representation is produced by concatenating the embedding vectors for the values for each slots.

User Model Output

Once the user model updates its state $s_{U,k}$ at each turn, it predicts a corresponding user response in the form of dialog act. The user model firstly emits a user action (e.g. *inform*, *deny*, etc) based on the current state:

$$P(a_{U,k} \mid \mathbf{G}_{\leq k}, \mathbf{U}_{<k}, \mathbf{A}_{\leq k}) = \mathrm{PolicyNet}_{\mathrm{U}}(s_{U,k}) \tag{8.2}$$

Conditioning on user state $s_{U,k}$ and the emitted user action $a_{U,k}$, the user model further predicts the value for each informable slot that a user may likely to express:

$$P(l^m_{U,k} \,|\, \mathbf{G}_{\leq k}, \mathbf{U}_{<k}, \mathbf{A}_{\leq k}) = \mathrm{SlotDist_{U,m}}(s_{U,k}, a_{U,k}) \tag{8.3}$$

where $\mathrm{PolicyNet_U}$ and $\mathrm{SlotDist_{U,m}}$ are DNNs with single hidden layer and softmax activation over candidate values.

8.1.2 Model Training

Similar to the method described in Chapter 5.2 in training the neural dialog agent, we can train the user model in a supervised manner by fitting task-oriented dialog samples. The optimization target is to minimize the prediction error of the next user action and corresponding slot values based on context represented in the past dialog turns. With maximum likelihood estimation, we optimize the user model parameter set θ_U with:

$$\begin{aligned} \min_{\theta_U} \sum_{k=1}^{K} - \Big[& \lambda_a \log P(a^*_{U,k} | \mathbf{G}_{\leq k}, \mathbf{U}_{<k}, \mathbf{A}_{\leq k}; \theta_U) \\ & + \sum_{m=1}^{M} \lambda_{l^m} \log P({l^m_{U,k}}^* | \mathbf{G}_{\leq k}, \mathbf{U}_{<k}, a_{U,k}, \mathbf{A}_{\leq k}, ; \theta_U) \Big] \end{aligned} \tag{8.4}$$

where ${l^m_{U,k}}^*$ and $a^*_{U,k}$ are the ground truth labels for goal slots and user action the kth turn.

Supervised training user models via fitting fixed dialog corpora may perform well in the similar dialog scenarios that are covered by the training set but may not be robust enough to handle diverse dialog situations due to the limited varieties in the training corpora. The user model can be further improved in an interactive setting, similar to how we can improve the neural dialog agent interactively by simulating conversations and exploring large dialog action space.

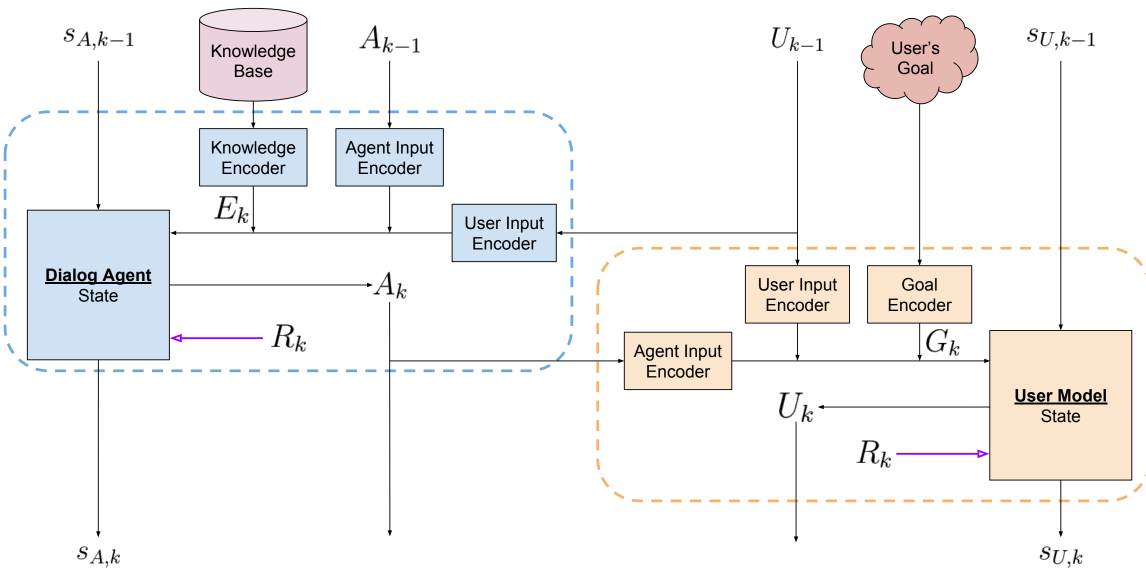

Figure 8.2: Joint dialog agent and user model training with deep reinforcement learning

We design a method in jointly optimizing both the dialog agent and user model by simulating task-oriented dialogs between the two agents and iteratively optimizing their dialog policies with deep reinforcement learning. The intuition is that we model task-oriented dialog as a goal fulfilling task, in which we let the dialog agent and the user model to positively collaborate to achieve the goal. The user model is given a goal to complete, and it is expected to demonstrate coherent but diverse user behavior. The dialog agent attempts to estimate the user's goal and fulfill his request by conducting meaningful conversations. Both the two agents aim to learn to collaborate with each other to complete the task but without exploiting the game.

Similar to the method described in Chapter 6.1.2 in optimizing the dialog agent model, we can also optimize the user model using the reward signal received with REINFORCE. The objective function is written as $J_k(\theta_U) = \mathbb{E}_{\theta_U}[R_k] = \mathbb{E}_{\theta_U}\left[\sum_{t=k}^{K} \gamma^{t-k} r_t - V(s_{U,k})\right]$, with $V(s_{U,k})$ being the state value function serving as a baseline reward. Using likelihood ratio gradient estimator, the gradient of $J_k(\theta_U)$ can be derived as:

$$\begin{aligned}
\nabla_{\theta_U} J_k(\theta_U) &= \nabla_{\theta_U} \mathbb{E}_{\theta_U}\left[R_k\right] \\
&= \sum_{a_{U,k} \in \mathcal{A}} \pi_{\theta_U}(a_{U,k}|\cdot) \nabla_{\theta_U} \log \pi_{\theta_U}(a_{A,k}|\cdot) R_k \qquad (8.5) \\
&= \mathbb{E}_{\theta_U}\left[\nabla_{\theta_U} \log \pi_{\theta_U}(a_{U,k}|\cdot) R_k\right]
\end{aligned}$$

8.2 Learning from Real and Imagined Experiences

With this user model, the neural dialog agent may "imagine" the potential conversations with a real user by simulating dialogs with the user model and learn quickly from these imagined experiences. To encourage the agent to explore broader dialog state space, we let both the agent and the user model to act by sampling actions from their corresponding softmax policy network output. We name such learning procedure as learn-from-imagination.

8.2.1 Method

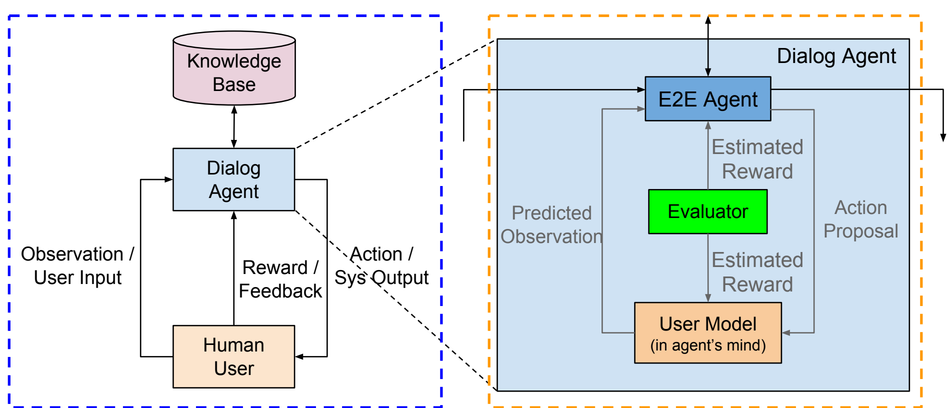

Figure 8.3: Integrated dialog learning with real (left) and imagined (right) experiences.

Figure 8.3 shows our proposed learning framework by combining learn-from-user and learn-from-imagination. The dialog agent interacts with user to complete task-oriented dialog and keeps improving itself with the feedback received from the user. The dialog agent also learns to interface with external resources, such as a knowledge base or a database, so as

Algorithm 5 Learning from Real and Imagined Experiences

Required: dialog corpus D, agent model A_{model}, user model U_{model}, real user U_{real}
Initialize and pretrain A_{model} on D with MLE
Initialize and pretrain U_{model} on D with MLE
for real user interaction $m \in \{1, \ldots, M\}$ **do**
    Run A_{model} with U_{real} to conduct dialog d_{real}
    Collect reward R_{real} from user for d_{real}
    Update A_{model} with R_{real}
    for imagined user interaction $n \in \{1, \ldots, N\}$ **do**
        Run A_{model} with U_{model} to roll out dialog d_{imag}
        Obtain reward R_{imag} for d_{imag}
        Update A_{model} and U_{model} with R_{imag}
    end for
end for

to be able provide responses to user that are based on the facts in the real world. Inside the dialog agent, the agent learns to model the user dynamics by predicting their behaviors in conversations. Such modeled user in the agent's mind can help the agent to simulate dialogs that mimic the conversations between the agent and a real user. By simulating such conversations and learning from it, the agent can learn more effectively from real users and reduce the number of interactive learning cycles.

We start with supervised pretraining of the dialog agent and user model with a set of seed annotated dialogs. This enables a reasonable initialization of the policy for the next stage optimization with RL. Once we obtain a supervised pretraining agent, we let the dialog agent to conduct task-oriented dialog with real users and continue to improve it with the user feedback at the end of the dialog. We apply policy gradient based RL in performing end-to-end model update.

Before letting the agent to start the next learning batch with users, we instruct the agent to practice the new knowledge learned from real user interactions by "imagining" the future conversation it might have with users. We let the agent to simulate task-oriented interactions with the user model using the updated policy. During these "imagined" interactions, the agent model and user model are further updated with task-completion based rewards.

8.3 Experiments

8.3.1 Training Environment and Setting

In our experiments, we utilize a publicly available dialog system platform PyDial [100] in training and evaluating our neural dialog agent. We target an application in a system providing restaurant information in Cambridge, UK. The task is to learn a dialog agent that is able to request and understand information from users, interface with backend systems or databases, and manage the dialog flow in successfully delivering requested information to users. This domain is defined with 6 slots, 3 of which are informable slots (food, are, and price range), with the remaining 3 as informable slots (address, phone, and postcode). There are in total 110 entities that have attributes in the above 6 slots.

We train our neural dialog agent against the handcrafted simulated user provided in PyDial, similar to the experimental setting in may prior work [99, 86]. In the experiment, this handcrafted user plays the role of the real user in our proposed learning framework. Although this handcrafted simulated user is sub-optimal in representing a real user, the simulation setting allows us to perform large scale experiments and in-depth analysis. We firstly generate 500 seed dialogs using the PyDial in the Cambridge Restaurant domain and use these dialogs to pretrain our neural dialog agent and neural user model. Agent and user actions in our system are defined by concatenating the act and slot types in the raw dialog act output (e.g. "$confirm(area = north)$" maps to "$confirm\_area$"). The corresponding slot values (e.g. $north$) are separately captured in the user goal estimation module for each slot. In total there are 29 agent actions and 61 user actions in our experimental setup.

For both the dialog agent and user model we use LSTM with state size of 150. Hidden layer size for the policy network and slot value estimation network is set as 100. Embedding size for actions and informable slot values are set as 20. In interactive model training, we set the maximum dialog turn size as 20. A task is marked as failed if the dialog goes beyond the maximum allow number of dialog turns. Dialog reward discount factor γ is set as 0.95. In learn-from-user stage, we update the agent model with every mini-batch of 25 interaction with real users. In learn-from-imagination stage, we experiment with updating the model with every mini-batch of 25, 50, and 100 roll-out sessions with the user model.

8.3.2 Results and Analysis

We first explain and analyze our empirical evaluation results. We first show the benefits of performing learn-from-imagination training and discuss the impact of different model training schedules. We then discuss the impact of jointly updating the user model during agent training process. Last but not least, we discuss how the different user model policy choices may affect the agent training effectiveness.

Learning From Imagined Experiences

We experimented with different learn-from-imagination sessions for each model update with real user interactions. Table 8.1 shows the results. We compare the performance of different model training procedures at different real user interactive learning stages (i.e. 1000, 3000, and 5000 dialogs). The baseline method is when we let the agent to only learn from real users and perform zero learn-from-imagination sessions (first row in Table 8.1). As can be seen from these results, all the 3 training procedures with imagined experiences helped the agent to achieve advanced task success rate comparing to the baseline method. More learn-from-imagination sessions per real user interaction batch led to higher task success rate at the early stage of the real user learning cycle (e.g. 1000 sessions), but not necessarily higher performance in the end. By looking into the dialogs conducted by the agent, we found that the average number of turns for dialogs with real users is 7.32. On the other hand, the average length of imagined dialogs with user model is only 4.26 turns. Conducting more imagination training sessions may help the agent to quickly explore different dialog scenarios, but at a risk of overfitting to the scenarios that can be covered by the user model. This might explain the decrease of agent performance in the end with more learn-from-imagination sessions between two learn-from-user batch.

Imagined Sess. Batch	**Real Sess. 1000**	**Real Sess. 3000**	**Real Sess. 5000**
0	87.2	89.6	90.4
25	88.5	**91.7**	**92.5**
50	89.3	91.1	92.2
100	**89.7**	90.3	90.6

Table 8.1: Performance of different learn-from-imagination configurations at different learn-from-user stages.

Impact of User Model Updates

We study the impact of updating the user model during learn-from-imagination training sessions. The user model is trained with the initial set of annotated dialog samples. The quality of this user model has a direct impact on the quality of the imagined dialogs and further the effectiveness of dialog learning with such imagined experiences. Continuous improvement of the user model may help the agent to learn better policy by exploring larger dialog space. We experimented with two different user model update settings, one with the user model being fixed during the agent training process, and the other with user model being jointly updated with the agent model during the learn-from-imagination stage. Results are shown in Fig. 8.4. As can be seen from these learning curves, both configurations for learn-from-imagination effectively improved dialog policy learning efficiency at the beginning stage of learn-from-user training cycles. With growing number of real user interactions, the performance of the agent with only learn-from-user experience (red curve) gradually caught up with the agent that learns from both real and imagined experiences. Agent trained with continuously updated user mode achieved superior task success rate comparing to the other baseline methods.

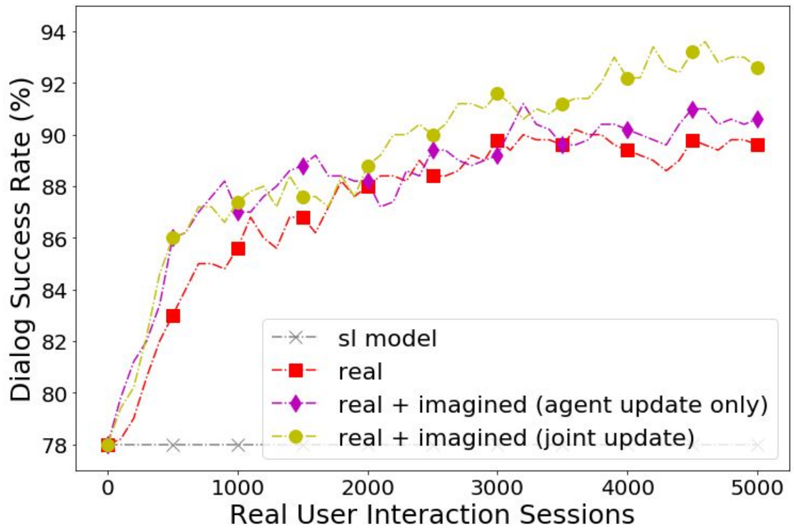

Figure 8.4: Comparing different configurations for user model update during learn-from-imagination stage.

Impact of User Model Policy

We further study the impact of different user model policy designs on the agent policy learning efficiency. We compare user model settings using greedy policy and softmax policy in learn-from-imagination stage. Greedy policy always selects the best estimated user action from the policy network, which enables a more consistent user model. Softmax policy, on the other hand, samples a user action from the policy network action probability distribution. This introduces additional randomness in user model behaviors. Fig. 8.5 shows the experiment results. Similar to previously observed trends, both policy configurations led to efficient agent policy learning at the beginning stage of the learn-from-user cycles. Agent trained with greedy policy user model did not show obvious advantage over agent that is trained only with real user experiences after 2500 learning sessions. This might because the agent has well exploited the greedy policy user model at this stage and stopped learning much new insights from the remaining imagined experiences. Agent trained with softmax policy user model achieved the highest success rate at the end of the interactive learning cycle.

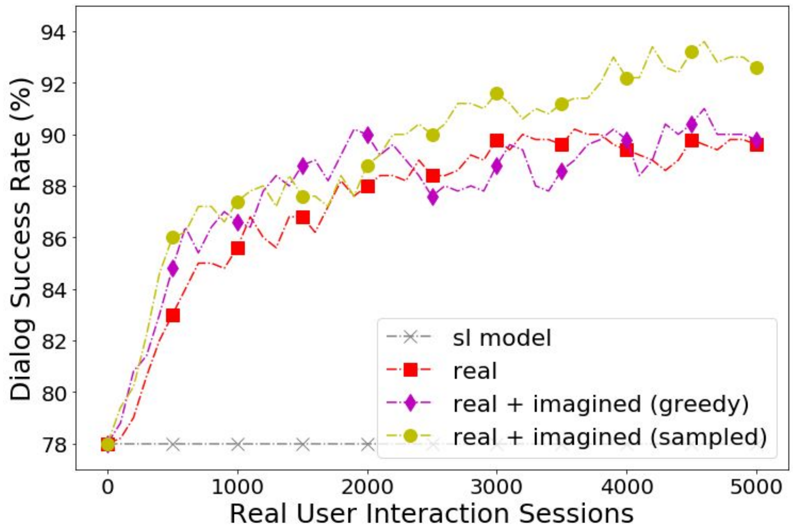

Figure 8.5: Comparing different choices of user model policy during learn-from-imagination stage.

8.4 Conclusion

In this chapter, we presented how we address the sample efficiency issue in online interactive dialog policy learning. Current model-free reinforcement learning based policy learning methods have limitations in sample efficiency. This requires a dialog agent to conduct a large number of interactive learning sessions with users in order to learn a good policy. We addressed this issue by proposing an integrated learning method that combines learn-from-user and learn-from-imagination experiences. The agent learns a user model which can imitate the behavior of a real user in conducting task-oriented dialogs. With such user model, the agent can simulate dialogs with the user model in mind and learn further from these imagined experiences in addition to learn from real user interactions. In the experiments in a restaurant search task domain, we showed that the proposed integrated learning method may effectively improve dialog policy learning efficiency within limited number of user interactions. We further discussed the how different user model designs and training procedures may impact the effectiveness of agent policy learning.

Chapter 9

Conclusion and Future Work

In this dissertation, we have presented a neural network based task-oriented dialog learning framework. We discussed the recent development of spoken dialog systems that powered by machine learning and especially deep learning methods. We identified challenges that arised in modeling longer context in conversation, optimizing dialog system end-to-end, and learning continuously through the interaction with users. We proposed solutions to address these challenges and evaluated them in simulated environments as well as with real users. In this chapter, we conclude the thesis and discuss future directions to continue this research on task-oriented dialog systems.

In Chapter 4, we described our multi-task learning models for robust spoken language understanding, a critical component in spoken dialog systems. We showed how we can learn better representations of user's natural language input by jointly optimizing intent identification and slot filling. In addition, we discussed how we can enable a tight integration of SLU and language modeling with a joint incremental model for contextual language modeling and understanding.

In Chapter 5, we discussed how we can extend the proposed SLU sequence model to end-to-end dialog modeling. We presented a hierarchical recurrent neural network based model that can robustly track dialog state, interface with external knowledge resources, and incorporate structured query results into system response to successfully complete a task. We showed that the proposed end-to-end dialog model achieved state-of-the-art performance in dialog state tracking and next utterance prediction in a widely studied bench-marking task from DSTC2.

In Chapter 6, we discussed how the proposed neural dialog model can continuously improve itself through the interaction with users, which makes the learning process truly *interactive*. We proposed a hybrid imitation and reinforcement dialog learning method, with which the end-to-end dialog agent can effectively learn from human teaching and feedback. We showed both in a simulated environment and with real user evaluation the effectiveness of conducting such interactive dialog learning with human-in-the-loop.

In Chapter 7, we presented an adversarial learning method in addressing the rating consistency and learning efficiency issues with interactive dialog learning. We showed that with limited number of annotated dialog samples, the proposed adversarial learning method can effectively learn a reward function and use that to guide policy optimization. We further discussed the covariate shift issue during interactive adversarial learning and showed how we can ameliorate it with partial access to user feedback.

In Chapter 8, we further address the sample efficiency issue with online interactive dialog learning with real users. We proposed an integrated learning method that combines learn-from-user and learn-from-imagination experiences. Instead of only letting the dialog agent to passively wait for a user's feedback and use it for policy learning, we also enabled the agent to learn a user model so that it can "imagine" the conversation in real world user interactions. We showed that such integrated learning method from real and imagined experience can effectively improve the policy learning efficiency during online user interactions.

With these contributions, we created a task-oriented dialog learning framework that enables *end-to-end* system optimization via efficient *interactive* learning with users. In this final section of the thesis, we highlight a few promising areas that can further advance the this research direction on task-oriented dialog systems:

Context and Knowledge Representation

In this thesis, we discussed methods for dialog context representation which includes information expressed in the conversation history in the current dialog between two parties. Our proposed system did not model and utilize environmental context, such as when and where a conversation is happening, or whether a conversation is closely related to some ongoing events. Such environmental context plays a significant role in human-human conversation. A big challenge in utilizing environmental context lies in that understanding a specific context and learning to prioritize it over the others usually requires a significant amount of common sense or world knowledge. How to abstract such knowledge and represent it in a way that can help an agent to make appropriate decisions is still an active research problem. Learning such context and knowledge representation purely from imitating the way human converse is not an optimal solution as it would require a massive amount of training data. Collecting such large annotated dialog corpus with related context signal is extremely challenging. In the proposed methods for modeling task-oriented dialog context, we attempted to use a combination of data-driven and ontology-based approach by introducing simple form of structures (e.g. slot types) in guiding the context representation learning. Such hybrid approach by injecting structure as a prior and learning continuously via interaction with the environment might be worthy to explore further for a more general form of context and knowledge representation learning.

User Modeling and Adaption

In this dissertation, we discussed the importance of user modeling in training a task-oriented dialog agent. A good user model enables a dialog agent to conduct large scale dialog simulations and learn a robust dialog policy. Comparing to many prior works that use rule-based user simulators in training the dialog system, we designed a statistical user model that is more flexible in introducing additional features and can continuously improve along with the dialog agent. Our user model is bootstrapped directly from dialog corpus, and thus can be seen as a single general user model that fits the statistics of the user behavior expressed in the entire corpus. A more desirable option is to have finer-grained control of a user model so that the model can be configured with diverse user behaviors (e.g. with different levels of randomness and cooperativeness). With such trainable and configurable user model, the dialog agent can be optimized to be more robust towards diverse user types. The agent can also learn to adapt to different user behaviours so as to complete a task in a more efficient and appropriate manner. The proposed user model is build with neural network which has the flexibility in introducing diverse types of features. We believe this can be a reasonable starting point towards building such advanced user models that are both trainable and configurable.

Cross Domain Knowledge Transfer

Domain extension has always been a challenge in real world spoken dialog systems. With conventional systems designed with complex modules and domain-specific rules, extending a system to new task domains is a painful process. In this thesis, we have discussed how we can model task-oriented dialog end-to-end from a data-driven approach with minimum structures introduced from domain ontology. This opens up possibilities in quickly bootstrapping a system in a new domain with significantly reduced efforts in designing domain-specific logics and annotating dialog samples. Our proposed system did not address how to effectively transfer the knowledge learned from one task domain to another. This can be knowledge for understanding common patterns in user requests, representing dialog state with context sharing, and learning common strategies in completing different tasks, etc. Effective knowledge transfer among task domains can help to further reduce the efforts required for data collection and annotation in the new domain. There are some recent efforts [94, 67] in cross-domain learning of spoken language understanding at single utterance level. It would be interesting to explore whether these techniques can be applied for effective cross-domain learning of task-oriented dialogs.

Research is a never-ending process. We hope the work we presented in this thesis can inspire researchers in the direction of spoken language understanding and dialog systems and pioneer a new class of end-to-end dialog learning systems.

References

[1] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington, "The atis spoken language systems pilot corpus," in *Proceedings of the DARPA speech and natural language workshop*, 1990, pp. 96–101.

[2] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington, "The atis spoken language systems pilot corpus," in *Proceedings, DARPA speech and natural language workshop*, 1990, pp. 96–101.

[3] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[4] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[5] C. Kamm, "User interfaces for voice applications," *Proceedings of the National Academy of Sciences*, vol. 92, no. 22, pp. 10 031–10 037, 1995.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on*, vol. 45, no. 11, pp. 2673–2681, 1997.

[8] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella, "Paradise: A framework for evaluating spoken dialogue agents," in *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, 1997, pp. 271–280.

[9] A. I. Rudnicky, E. H. Thayer, P. C. Constantinides, C. Tchou, R. Shern, K. A. Lenzo, W. Xu, and A. Oh, "Creating natural dialogs in the carnegie mellon communicator system.," in *Eurospeech*, 1999.

[10] A. Rudnicky and W. Xu, "An agenda-based dialog management architecture for spoken language systems," in *IEEE Automatic Speech Recognition and Understanding Workshop*, vol. 13, 1999.

[11] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[12] P. Haffner, G. Tur, and J. H. Wright, "Optimizing svms for complex call classification," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, IEEE, vol. 1, 2003, pp. I–632.

[13] H. M. Wallach, "Conditional random fields: An introduction," *Technical Reports (CIS)*, p. 22, 2004.

[14] A. Raux, B. Langner, D. Bohus, A. W. Black, and M. Eskenazi, "Let's go public! taking a spoken dialog system to the real world," in *Ninth European Conference on Speech Communication and Technology*, 2005.

[15] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young, "A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies," *The knowledge engineering review*, vol. 21, no. 2, pp. 97–126, 2006.

[16] S. Young, "Using pomdps for dialog management," in *Spoken Language Technology Workshop, 2006. IEEE*, IEEE, 2006, pp. 8–13.

[17] N. Nguyen and Y. Guo, "Comparisons of sequence labeling algorithms and extensions," in *Proceedings of the 24th international conference on Machine learning*, ACM, 2007, pp. 681–688.

[18] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding.," in *INTERSPEECH*, 2007, pp. 1605–1608.

[19] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, "Agenda-based user simulation for bootstrapping a pomdp dialogue system," in *NAACL-HLT*, 2007.

[20] J. D. Williams and S. Young, "Partially observable markov decision processes for spoken dialog systems," *Computer Speech & Language*, vol. 21, no. 2, pp. 393–422, 2007.

[21] D. Bohus and A. I. Rudnicky, "The ravenclaw dialog management framework: Architecture and systems," *Computer Speech & Language*, vol. 23, no. 3, pp. 332–361, 2009.

[22] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 661–668.

[23] G. Tur, D. Hakkani-Tur, and L. Heck, "What is left to be understood in atis?" In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, IEEE, 2010, pp. 19–24.

[24] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[25] R. Sarikaya, G. E. Hinton, and B. Ramabhadran, "Deep belief nets for natural language call-routing," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, IEEE, 2011, pp. 5680–5683.

[26] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[27] G. Tur, D. Hakkani-Tür, L. Heck, and S. Parthasarathy, "Sentence simplification for spoken language understanding," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, IEEE, 2011, pp. 5628–5631.

[28] L. Deng, G. Tur, X. He, and D. Hakkani-Tur, "Use of kernel deep convex networks and end-to-end learning for spoken language understanding," in *Spoken Language Technology Workshop (SLT), 2012 IEEE*, IEEE, 2012, pp. 210–215.

[29] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[30] G. Tur, L. Deng, D. Hakkani-Tür, and X. He, "Towards deeper understanding: Deep convex networks for semantic utterance classification," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, IEEE, 2012, pp. 5045–5048.

[31] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[32] M. Gašić, C. Breslin, M. Henderson, D. Kim, M. Szummer, B. Thomson, P. Tsiakoulis, and S. Young, "On-line policy optimisation of bayesian spoken dialogue systems via human interaction," in *ICASSP*, 2013.

[33] M. Henderson, B. Thomson, and J. Williams, *Dialog state tracking challenge 2 & 3*, 2013.

[34] S. Lee, "Structured discriminative model for dialog state tracking," in *Proceedings of the SIGDIAL 2013 Conference*, 2013, pp. 442–451.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[36] P. Xu and R. Sarikaya, "Convolutional neural network based triangular crf for joint intent detection and slot filling," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, IEEE, 2013, pp. 78–83.

[37] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu, "Recurrent neural networks for language understanding.," in *Interspeech*, 2013, pp. 2524–2528.

[38] S. Young, M. Gašić, B. Thomson, and J. D. Williams, "Pomdp-based statistical spoken dialog systems: A review," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1160–1179, 2013.

[39] L. El Asri, R. Laroche, and O. Pietquin, "Task completion transfer learning for reward inference," *Proc of MLIS*, 2014.

[40] D. Guo, G. Tur, W.-t. Yih, and G. Zweig, "Joint semantic utterance classification and slot filling with recursive neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*, IEEE, 2014, pp. 554–559.

[41] D. Guo, G. Tur, W.-t. Yih, and G. Zweig, "Joint semantic utterance classification and slot filling with recursive neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*, IEEE, 2014, pp. 554–559.

[42] M. Henderson, B. Thomson, and J. Williams, "The second dialog state tracking challenge," in *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, vol. 263, 2014.

[43] M. Henderson, B. Thomson, and S. Young, "Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised gate," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*, IEEE, 2014, pp. 360–365.

[44] M. Henderson, B. Thomson, and S. Young, "Word-based dialog state tracking with recurrent neural networks," in *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, 2014, pp. 292–299.

[45] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2014, pp. 1746–1751.

[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[48] P. Xu and R. Sarikaya, "Contextual domain classification in spoken language understanding systems using recurrent neural network," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, IEEE, 2014, pp. 136–140.

[49] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*, IEEE, 2014, pp. 189–194.

[50] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*, IEEE, 2014, pp. 189–194.

[51] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[52] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[53] E. L. Denton, S. Chintala, R. Fergus, *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494.

[54] M. Henderson, "Machine learning for dialog state tracking: A review," in *Machine Learning in Spoken Language Processing Workshop*, 2015.

[55] B. Liu and I. Lane, "Recurrent neural network structured output prediction for spoken language understanding," in *Proc. NIPS Workshop on Machine Learning for Spoken Language Understanding and Interactions*, 2015.

[56] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu, *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 23, no. 3, pp. 530–539, 2015.

[57] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu, *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 23, no. 3, pp. 530–539, 2015.

[58] N. Mrkšić, D. Ó Séaghdha, B. Thomson, M. Gasic, P.-H. Su, D. Vandyke, T.-H. Wen, and S. Young, "Multi-domain dialog state tracking using recurrent neural networks," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Association for Computational Linguistics, 2015, pp. 794–799.

[59] B. Peng, K. Yao, L. Jing, and K. Wong, "Recurrent neural networks with external memory for spoken language understanding," in *Natural Language Processing and Chinese Computing - 4th CCF Conference, NLPCC 2015, Nanchang, China, October 9-13, 2015, Proceedings*, 2015, pp. 25–35.

[60] L. Shang, Z. Lu, and H. Li, "Neural responding machine for short-text conversation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, 2015, pp. 1577–1586.

[61] A. Sordoni, M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan, "A neural network approach to context-sensitive generation of conversational responses," in *Proceedings of NAACL-HLT*, 2015, pp. 196–205.

[62] O. Vinyals and Q. Le, "A neural conversational model," in *ICML Deep Learning Workshop*, 2015.

[63] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, 2015, pp. 649–657.

[64] L. E. Asri, J. He, and K. Suleman, "A sequence-to-sequence model for user simulation in spoken dialogue systems," in *Interspeech*, 2016.

[65] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, IEEE, 2016, pp. 4960–4964.

[66] Y.-N. Chen, D. Hakkani-Tür, and X. He, “Zero-shot learning of intent embeddings for expansion by convolutional deep structured semantic models,” in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, IEEE, 2016, pp. 6045–6049.

[67] D. Hakkani-Tür, G. Tür, A. Celikyilmaz, Y.-N. Chen, J. Gao, L. Deng, and Y.-Y. Wang, “Multi-domain joint semantic frame parsing using bi-directional rnn-lstm.,” in *Interspeech*, 2016, pp. 715–719.

[68] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[69] N. Jaitly, Q. V. Le, O. Vinyals, I. Sutskever, D. Sussillo, and S. Bengio, “An online sequence-to-sequence model using partial conditioning,” in *Advances in Neural Information Processing Systems*, 2016, pp. 5067–5075.

[70] G. Kurata, B. Xiang, B. Zhou, and M. Yu, “Leveraging sentence-level information with encoder lstm for semantic slot filling,” in *EMNLP*, 2016.

[71] J. Li, W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao, “Deep reinforcement learning for dialogue generation,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 1192–1202.

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

[73] B. Liu and I. Lane, “Attention-based recurrent neural network models for joint intent detection and slot filling,” in *Interspeech*, 2016.

[74] B. Liu and I. Lane, “Joint online spoken language understanding and language modeling with recurrent neural networks,” in *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2016.

[75] C.-W. Liu, R. Lowe, I. Serban, M. Noseworthy, L. Charlin, and J. Pineau, “How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation,” in *EMNLP*, 2016.

[76] N. Mrkšić, D. O. Séaghdha, T.-H. Wen, B. Thomson, and S. Young, “Neural belief tracker: Data-driven dialogue state tracking,” *arXiv preprint arXiv:1606.03777*, 2016.

[77] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau, “Building end-to-end dialogue systems using generative hierarchical neural network models,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI’16, AAAI Press, 2016, pp. 3776–3783.

[78] P.-H. Su, M. Gašić, N. Mrkšić, L. Rojas-Barahona, S. Ultes, D. Vandyke, T.-H. Wen, and S. Young, “On-line active reward learning for policy optimisation in spoken dialogue systems,” in *Proceedings of ACL*, 2016.

[79] J. Williams, A. Raux, and M. Henderson, “The dialog state tracking challenge series: A review,” *Dialogue & Discourse*, vol. 7, no. 3, pp. 4–33, 2016.

[80] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480–1489.

[81] X. Zhang and H. Wang, "A joint model of intent determination and slot filling for spoken language understanding.," in *IJCAI*, 2016, pp. 2993–2999.

[82] T. Zhao and M. Eskenazi, "Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning," in *SIGDIAL*, 2016.

[83] B. Liu and I. Lane, "Dialog context language modeling with recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, IEEE, 2017, pp. 5715–5719.

[84] A. Bapna, G. Tur, D. Hakkani-Tur, and L. Heck, "Sequential dialogue context modeling for spoken language understanding," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, Association for Computational Linguistics, 2017, pp. 103–114.

[85] A. Bordes and J. Weston, "Learning end-to-end goal-oriented dialog," in *ICLR*, 2017.

[86] P. Budzianowski, S. Ultes, P.-H. Su, N. Mrkšić, T.-H. Wen, I. Casanueva, L. M. Rojas Barahona, and M. Gasic, "Sub-domain modelling for dialogue management with hierarchical reinforcement learning," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, 2017, pp. 86–92.

[87] M. Eric and C. D. Manning, "A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue," in *EACL*, 2017.

[88] M. A. Islam, M. Rochan, N. D. Bruce, and Y. Wang, "Gated feedback refinement network for dense image labeling," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 4877–4885.

[89] J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky, "Adversarial learning for neural dialogue generation," in *Proceedings of ACL*, 2017.

[90] X. Li, Y.-N. Chen, L. Li, and J. Gao, "End-to-end task-completion neural dialogue systems," in *IJCNLP*, 2017.

[91] B. Liu and I. Lane, "Iterative policy learning in end-to-end trainable task-oriented neural dialog models," in *Proceedings of 2017 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2017.

[92] B. Liu and I. Lane, "An end-to-end trainable neural network model with belief tracking for task-oriented dialog," in *Interspeech*, 2017.

[93] B. Liu, G. Tur, D. Hakkani-Tur, P. Shah, Heck, and Larry, "End-to-end optimization of task-oriented dialogue model with deep reinforcement learning," in *NIPS Workshop on Conversational AI*, 2017.

[94] B. Liu and I. Lane, "Multi-domain adversarial learning for slot filling in spoken language understanding," in *NIPS Workshop on Conversational AI*, 2017.

[95] F. Liu and J. Perez, "Gated end-to-end memory networks," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Association for Computational Linguistics, 2017, pp. 1–10.

[96] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv preprint arXiv:1704.03073*, 2017.

[97] M. Seo, S. Min, A. Farhadi, and H. Hajishirzi, "Query-reduction networks for question answering," in *International Conference on Learning Representations*, 2017.

[98] I. V. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. C. Courville, and Y. Bengio, "A hierarchical latent variable encoder-decoder model for generating dialogues.," in *AAAI*, 2017, pp. 3295–3301.

[99] P.-H. Su, P. Budzianowski, S. Ultes, M. Gasic, and S. Young, "Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, Saarbrücken, Germany: Association for Computational Linguistics, 2017, pp. 147–157.

[100] S. Ultes, L. M. R. Barahona, P.-H. Su, D. Vandyke, D. Kim, I. Casanueva, P. Budzianowski, N. Mrkšić, T.-H. Wen, M. Gasic, *et al.*, "Pydial: A multi-domain statistical dialogue system toolkit," *Proceedings of ACL 2017, System Demonstrations*, pp. 73–78, 2017.

[101] T.-H. Wen, D. Vandyke, N. Mrkšić, M. Gašić, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young, "A network-based end-to-end trainable task-oriented dialogue system," in *EACL*, 2017.

[102] J. D. Williams, K. Asadi, and G. Zweig, "Hybrid code networks: Practical and efficient end-to-end dialog control with supervised and reinforcement learning," in *ACL*, 2017.

[103] T. Zhao, R. Zhao, and M. Eskenazi, "Learning discourse-level diversity for neural dialog models using conditional variational autoencoders," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 2017, pp. 654–664.

[104] Y.-B. Kim, D. Kim, A. Kumar, and R. Sarikaya, "Efficient large-scale neural domain classification with personalized attention," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2018, pp. 2214–2224.

[105] B. Liu and I. Lane, "Adversarial learning of task-oriented neural dialog models," in *SIGDIAL*, 2018.

[106] B. Liu and I. Lane, "Dialog policy learning with real and imagined experiences," in *in preparation*, 2018.

[107] B. Liu, G. Tur, D. Hakkani-Tur, P. Shah, and L. Heck, "Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems," in *NAACL*, 2018.

[108] B. Liu, T. Yu, I. Lane, and O. Mengshoel, "Customized nonlinear bandits for online response selection in neural conversation models," in *AAAI*, 2018.

[109] P. Shah, D. Hakkani-Tür, B. Liu, and G. Tür, "Bootstrapping a neural conversational agent with dialogue self-play, crowdsourcing and on-line reinforcement learning," in *NAACL-HLT*, 2018.

[110] Y. Wang, Y. Shen, and H. Jin, "A bi-model based rnn semantic frame parsing model for intent detection and slot filling," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, vol. 2, 2018, pp. 309–314.

[111] B. Dhingra, L. Li, X. Li, J. Gao, Y.-N. Chen, F. Ahmed, and L. Deng, "Towards end-to-end reinforcement learning of dialogue agents for information access," in *Proceedings of ACL*.