TRANSFER REINFORCEMENT LEARNING FOR TASK-ORIENTED DIALOGUE SYSTEMS

by

KAIXIANG MO

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in Computer Science and Engineering

March 2018, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

KAIXIANG MO

TRANSFER REINFORCEMENT LEARNING FOR TASK-ORIENTED DIALOGUE SYSTEMS

by

KAIXIANG MO

This is to certify that I have examined the above Ph.D. thesis

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by

the thesis examination committee have been made.

PROF. QIANG YANG, THESIS SUPERVISOR

PROF. DIT-YAN YEUNG, ACTING HEAD OF DEPARTMENT

Department of Computer Science and Engineering

26 March 2018

ACKNOWLEDGMENTS

Firstly, I want to thank my supervisor Prof. Qiang Yang for his sincere advice and support, it is my great honor to have such a visionary, intelligent and energetic supervisor. Prof. Yang encourages me to aim higher and dream bigger, and he also teaches me how to identify, formulate and address important and novel problems. During my Ph.D. study, Prof. Yang is extremely supportive and gives me great opportunities to get in touch with the industry and work with real industrial needs. Apart from the great achievements, Prof. Yang is always ambitious and ready to take on new academic and industrial challenges, which encourages me when I am faced with new problems.

I would also like to thank my other committee members, Prof. Mei Ling Helen Meng, Prof. Pascale Fung, Prof. Lei Chen and Prof. Xiaojuan Ma, for spending valuable time on my thesis defense and giving me invaluable suggestions. Their constructive comments are very helpful for improving this thesis.

I would like to thank Prof. Rong Pan, who opened the door to machine learning research for me. I also sincerely thank Dr. Yu Zhang and Prof. Yangqiu Song, for their great help and patience when discussing research ideas with me.

The other current and former members of my research group also help me in various ways, I would like to give my thanks to Dou Shen, Sinno Jialin Pan, Vincent Wenchen Zheng, Bin Cao, Nathan Nan Liu, Qian Xu, Si Shen, Evan Wei Xiang, Yin Zhu, Lili Zhao, Ben Tan, Zhongqi Lu, Bin Wu, Ying Wei, Lianghao Li, Liya Ji, Ruiming Xie, Bo Liu, Wenya Zhu, Yinghua Zhang, Yuxiang Wu, Weiyang Wang, Zheng Li, Guangneng Hu, Yanzhou Huang.

I would also like to thank the other friends I met in HKUST, I spent a good time with them and I could not have survived without them. Thanks to Wenliang Zhong, Naiyan Wang, Xiangming Dai, Peixian Chen, Siyi Li, Ming Wen, Zhourong Chen, Ruiliang Zhang, Hao Wang, Minghao Jiang, Xingjian Shi.

Finally, I want to give my deepest gratitude to my parents and my girlfriend for their unconditional love, support, and understanding. I dedicate this thesis to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TRANSFER REINFORCEMENT LEARNING FOR TASK-ORIENTED DIALOGUE SYSTEMS

by

KAIXIANG MO

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

ABSTRACT

Dialogue systems are attracting more and more attention recently. Dialogue systems can be categorized into open-domain dialogue systems and task-oriented dialogue systems. Task-oriented dialogue systems are designed to help user finish a specific task, and there are four modules, namely the spoken language understanding module, the dialogue state tracking module, the dialogue policy module and the natural language generation module. One of the most important modules is the dialogue policy module, which aims to choose the best reply according to the dialogue context. In this thesis, we focus on the dialogue policy of task-oriented dialogue systems.

Reinforcement learning is usually used in the dialogue policy. However, traditional reinforcement learning algorithms rely heavily on a large number of training data and accurate reward signals. Transfer learning can leverage knowledge from a source domain and improve the performance of a model in the target domain with little target domain data. However, traditional transfer learning methods focus on supervised learning setting, and they cannot handle knowledge transfer in reinforcement setting since they do not consider the states. Transfer reinforcement learning (TRL) aims to transfer dialogue policy knowledge across different domains. In the target domain, the state and action can be aligned to the source domain state and action, so the dialogue policy can be transferred from the source domain to the target domain.

The key to transfer reinforcement learning is learning to build the mapping between the source and the target domains, and transfer only domain independent common knowledge while minimiz-

ing the negative transfer caused by the domain-dependent knowledge. In this thesis, we propose a unified framework for transfer reinforcement learning problems in task-oriented dialogue systems, including 1) How to transfer dialogue policies across different users with different preferences in personalized task-oriented dialogue system? 2) How to transfer fine-granularity common knowledge when the common knowledge is mixed with the domain-dependent knowledge? 3) How to transfer dialogue policies across dialogue systems built with different sets of speech-acts and slots? We will use both large-scale simulations and large-scale real-world datasets to valid this research. The thesis will also discuss the latest progress in the field and point out some future directions for future investigation.

CHAPTER 1

INTRODUCTION

Dialogue systems can communicate with people via natural language speeches or text sentences. Reinforcement learning is widely used to model the dialogue policy in task-oriented dialogue systems However, there is not enough training data to train a dialogue policy in many real-world situations, due to the privacy issue and the high annotation cost. Traditional transfer learning methods can only work on supervised learning problems, and they could not deal with reinforcement learning problems that have different state spaces, action spaces, state transition probabilities and reward functions.

In this thesis, we focus on the transfer reinforcement learning (TRL) problems in task-oriented dialogue systems. Transfer reinforcement learning seeks to leverage the knowledge in a source domain to help to build a policy in a target domain. Compared with traditional transfer learning, the transfer reinforcement learning problems are more general, since they have to consider the cross-domain state mapping, action mapping, transition function mapping and reward function mapping.

1.1 Motivation

Dialogue systems can communicate with people via natural language speeches or text sentences. There are many applications of dialogue systems, including but not limited to daily chatting, customer services, answering knowledge related questions via internet searching or database searching and guiding a user to finish specific tasks. Dialogue systems can be classified into two classes: open-domain dialogue systems [114, 32, 119, 77, 98] and task-oriented dialogue systems [76, 184, 169, 166, 177]. Open-domain dialogue systems do not limit the dialogue topic to a specific domain, and typically they do not have a clear dialogue goal. Typical open-domain dialogue systems include the Xiaoice from Microsoft, an example is shown in Figure 1.1(a). Task-oriented dialogue systems aim to solve a specific task via dialogues, typical task-oriented dialogue systems include Amazon Alexa, Google Home, etc. An example task-oriented dialogue is shown in Figure 1.1(b). In this thesis, we focus on the dialogue systems which aim to assist users to finish a task such as ordering a cup of coffee.

	Open-domain Dialogue (X=customer, Y=system)
X_1	nothin much, and how's the book?!
Y_1	its good but i'm only like halfway through cuz i don't feel like reading. i'm so bored ...
X_2	that's good! i have the book but i'm bored too.

(a) Open-Domain Dialogues.

	Coffee Shopping Dialogue (X=customer, Y=system)
X_1	Can I have coffee please?
Y_1	**What coffee would you like?**
X_2	I would like a cup of Mocha.
Y_2	Hot Mocha or Iced Mocha?
X_3	Iced Mocha.
Y_3	What cup size do you want?
X_4	Tall.
Y_4	What is your address?
X_5	No.1199 Mingsheng Road.
Y_5	Please pay.
Y_6	Payment Completed.
R	**<Task-Completion-Feedback>**

(b) Task Oriented Dialogues.

Figure 1.1: Dialogue Systems.

Reinforcement learning [135] aims to train an agent to perform a series of actions under different states, with the goal of maximizing the total reward. Compared with supervised learning, reinforcement learning is different since there is no ground-truth action label for each state, and the agent can only learn from the total reward signal. For example, the latest AlphaGo Zero can learn to play Go by being told whether it wins or not, without a teacher telling it the correct action at each time. Figure 1.2 shows that AlphaGo beats the human champion.

Reinforcement learning is widely used to model dialogue policy in task-oriented dialogue systems [14, 76, 152, 125, 184]. Dialogue reinforcement learning is different from other reinforcement learning problems in the following ways. Firstly, unlike chess and Go, dialogues do not have a clear ground-truth state space. Secondly, unlike some physical control system, the transition model of a dialogue is unknown. Thirdly, unlike some UAV control systems that have a fixed number of control signal, the number of possible reply sentences of a dialogue system is not enumerable. Finally, a dialogue has the natural ambiguity of a language. The meaning of a word in a sentence is conditional on its context and the same word might have completely different meanings. And the same meaning can also be expressed via different sentences in many different ways. Due to these unique characteristics of dialogue systems, training a dialogue policy requires a lot of training data.

However, there is no enough training data to train a dialogue policy in many real-world situ-

Figure 1.2: AlphaGo beats the human champion with reinforcement learning.

ations, due to the privacy issue and the high annotation cost. For example, if we want to build a dialogue system for a new task such as the online coffee ordering, there is only little data in this new domain and it is very difficult to collect a large number of data before the dialogue system goes online. If we want to build a personalized dialogue system for each customer, it is typically impossible to collect a large number of dialogue data from a single customer.

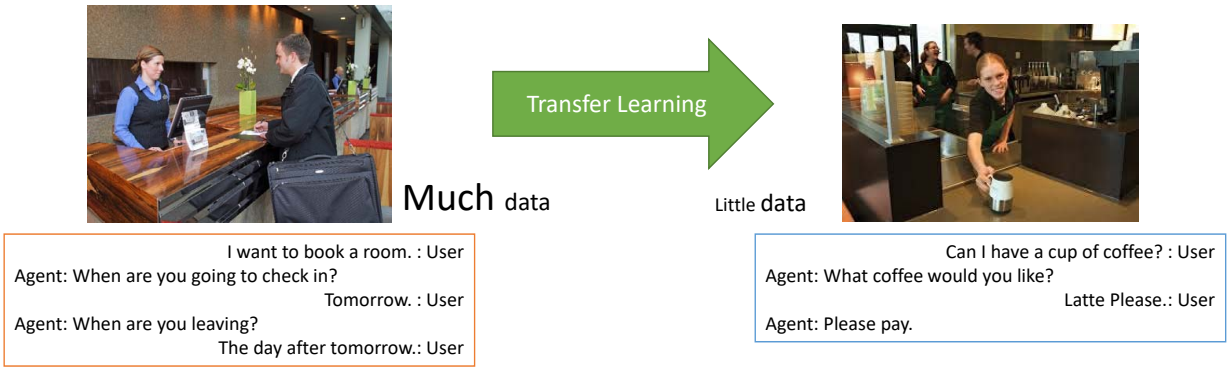


Figure 1.3: Transfer Reinforcement Learning for Task-oriented Dialogue Systems.

Transfer learning [102] can leverage knowledge from a source domain to effectively learn a model in a target domain with little training data. However, the traditional transfer learning can only work on supervised learning problems, and it could not deal with reinforcement learning problems that have different state spaces, action spaces, state transition probabilities and the reward functions.

Transfer reinforcement learning (TRL) seeks to leverage the knowledge in a source domain to

help to build a policy in the target domain. Compared with the traditional transfer learning, the transfer reinforcement learning problems [141] is more general, since it has to consider the cross-domain state mapping, action mapping, transition function mapping and reward function mapping. The differences of between transfer learning, reinforcement learning and transfer reinforcement learning are listed in Table 1.1.

Table 1.1: Summary of different learning problems.

	No States	Have States
No Transfer	Supervised Learning	Reinforcement Learning
Transfer	Traditional Transfer Learning	Transfer Reinforcement Learning

Transfer Reinforcement Learning problem for task-oriented dialogue systems is an important problem. As shown in Figure 1.3, it aims to build a dialogue system with only a few in-domain dialogue data and a lot of dialogue data in another domain. For example, if we want to build a personalized coffee ordering dialogue system for each user, but the data collected from each user is not sufficient for building a whole dialogue system. Due to the difference of customer preferences, naively sharing training dialogues across users might have a risk of personal information leakage, and it might have a negative impact on the system performance. How can we safely transfer dialogue policies across different users? Another example, when building a new hotel booking dialogue system, if we have only a small number of hotel booking dialogues, but we have a lot of restaurant booking dialogues from another dialogue system, how can we transfer knowledge across different domains and across different systems? In summary, it boils down to the question: how can we transfer dialogue policies across different users, domains, and systems?

1.2 Problem Description

In this section, we firstly define the concepts in task-oriented dialogue systems, then formulate the problem of transfer reinforcement learning in the task-oriented dialogue system, finally we will introduce the framework we use to address the research objectives above.

In this thesis, matrices are denoted in the bold capital case, column vectors are in bold lower case, scalars are in lower case.

A user utterance is denoted by X and an agent response is denoted by Y. A dialogue is denoted by $\{X_n, Y_n\}_{i=1}^{N}$, where n denotes the n-th turn and N is the number of dialogue turns in

Table 1.2: Notation Definition

Notation	Description	Notation	Description
Data			
$\mathcal{D}^s = \{\{X_n^s, Y_n^s, r_n^s\}_{n=1}\}$	Source Dialogues	$\mathcal{D}^t = \{\{X_n^t, Y_n^t, r_n^t\}_{n=1}\}$	Target Dialogues
X^s	Source User Utterance	X^t	Target User Utterance
Y^s	Source Agent Reply	Y^t	Target Agent Reply
$H_n^s = \{\{X_j^s, Y_j^s\}_{j=1}^{n-1}, X_n^s\}$	Dialogue Context/State for the n-th turn in Source	$H_n^t = \{\{X_j^t, Y_j^t\}_{j=1}^{n-1}, X_n^t\}$	Dialogue Context/State for the n-th turn in Target
A^s	Source Speech-act	A^t	Target Speech-act
S^s	Source Slot Set	S^t	Target Slot Set
V^s	Source Slot Value Set	V^t	Target Slot Value Set
$\pi^s : H_n^s \mapsto Y_n^s$	Source Dialogue Policy	$\pi^t : H_n^t \mapsto Y_n^t$	Target Dialogue Policy
Distribution			
$p(V^s)$	Probability distribution of Source Slot Value Set	$p(V^t)$	Probability distribution of Target Slot Value Set

the dialogue. The dialogue context for generating the n-th agent reply Y_n is denoted by $H_n = \{\{X_j, Y_j\}_{j=1}^{n-1}, X_n\}$. A dialogue dataset is denoted by $\mathcal{D} = \{\{X_n, Y_n, r_n\}_{n=1}^N\}$, where r_n is the immediate reward in the n-th turn.

A speech-act [1] is a symbol representing a specific intent of a sentence in the dialogue, and we denote the set of all speech-acts in a domain by $A = \{a_1, a_2, \cdots\}$, where a_i is a speech-act. A slot is a piece of information related to the dialogue domain, and we denote the set of all slots in a domain by $S = \{s_1, s_2, \cdots\}$, where s_i is a specific slot. A slot-value v_j is a specific value for a slot s_i, and we use V_{s_i} to denote the set of all possible slot values for slot s_i. For example, the user utterance "I want a cup of latte" can be represented by a speech-act $a =$ Inform, a slot $s =$ Coffee and a slot value $v =$ Latte.

The reinforcement learning problem in dialogue system can be modelled with Markov decision process. A state corresponds to a dialogue context, and it is also denoted by H_n. The state space can also be real value vector space, which can be either designed or learned. If the dialogue task is episodic, there is an initial dialogue state H_0 and a final dialogue state H_f. An action corresponds to an agent reply, and it is denoted by Y_n as well. After the user responds X_{n+1} to the last agent response Y_n, the dialogue transits to a new dialogue state H_{n+1}, and this is the transition function $H_n \times X_{n+1} \times Y_n \mapsto H_{n+1}$. The transition function is non-deterministic because the user's reply is somehow non-deterministic. The reward function $H_n \mapsto r_n$ maps a dialogue state H_n to a reward scalar r_n which is the immediate reward for reaching a dialogue state. A dialogue policy $\pi : H_n \mapsto Y_n$ defines how to generate an agent response in each dialogue state. The success of

a dialogue policy is determined by how well it can maximize the reward function in each episode under the dialogue policy π. The dialogue policy can be represented by a Q-function $Q(H_n, Y_n)$, which returns the expected cumulative reward if we take action Y_n at state H_n and then follow the dialogue policy π.

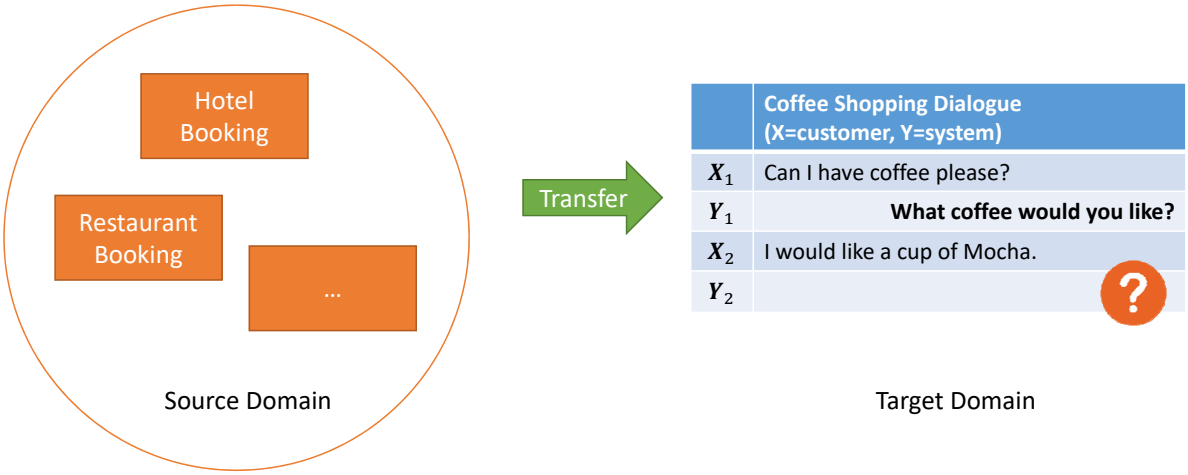


Figure 1.4: The problem of Transfer Reinforcement Learning in Task-oriented dialogue system.

The transfer reinforcement learning problem in task-oriented dialogue systems aims to transfer a dialogue policy from a source domain to a target domain, as illustrated in Figure 1.4. The inputs of the problems are

1. Abundant source domain dialogue data $\mathcal{D}^s = \{\{X_n^s, Y_n^s, r_n^s\}_{n=1}\}$.
2. Abundant target domain dialogue data $\mathcal{D}^t = \{\{X_n^t, Y_n^t, r_n^t\}_{n=1}\}$.

The output of the problem is

1. A dialogue policy π^t in the target domain.

1.3 Research Challenges

Transfer reinforcement learning problems for task-oriented dialogue systems have several unique challenges.

1. Firstly, different persons have different preferences. In personalized dialogue systems, the agent replies are specially personalized to make the dialogue more concise and convenient for the user. However, different users have different preferences, so different users might

respond differently to a suggestion. For example, for the suggestion "Would you like a cup of latte today?", Alice might say yes because she loves latte but Bob might refuse because he loves black coffee. The different user preferences lead to different data distributions, so the training data cannot be naively shared across different users. Improper transfer learning techniques might lead to a personal information leakage and a bad dialogue policy. For example, the system might suggest Alice's private address and phone number to Bob, causing a personal information leakage, and Bob might be irritated if the system keeps making wrong suggestions.

2. Secondly, useful common knowledge might be mixed with personal information. In an end-to-end personalized dialogue system, the appropriate dialogue replies for two different users might be very similar except for a small part, for example, "Would you like a cup of latte?" for Alice and "Would you like a cup of black coffee?" for Bob. Current existing transfer learning techniques can only transfer coarse granularity knowledge on the sentence level, i.e., it can either transfer a training dialogue sentence or transfer nothing at all. However, only a common part of the whole sentence should be transferred. How to transfer the common part of a sentence while keeping the personal part untouched remains a big challenge.

3. Thirdly, different domains have different slots. Slots are the information related to a specific domain, for example for the coffee ordering domain, possible slots might include the type of coffee, the temperature, and the cup size. Different domains might have a different set of slots, for example, the restaurant booking domain and the hotel booking domain have different slots. However, it is intuitively easy for a human to transfer restaurant booking experience to hotel booking domain since the high-level pattern of service is similar across different domains. How to transfer dialogue policies across different domains with different slots is a big challenge.

4. Finally, different dialogue systems might have different speech-acts. Many dialogue systems use speech-acts in task-oriented dialogue systems. However, different dialogue system mights be built with different sets of speech-acts, since speech-acts are only symbols representing dialogue intents. For example, a company might use the symbol "inform" to indicate the user is telling the system some useful information, but another company might use the symbol "tell" or even "act1" to indicate this meaning. Due to the differences of speech-act symbols and an unknown cross-domain speech-act mapping relationship, dialogue policies can not be transferred across dialogue systems with different speech-acts.

In this thesis, we study the problem of transfer reinforcement learning in task-oriented dialogue systems.

1.4 Contributions: The Unified TRL Model

To solve the challenges stated in the introduction, we propose to model the target policy with a Q-function consist of two parts. The first part of the Q-function corresponds to the transferable common dialogue policy, which can be shared across the source and target tasks. The second part of the Q-function models the domain-dependent factors of the dialogue policy. In order to allow a flexible switch between general model and personal model, the factor of the common Q-function and the personal Q-function should be modelled as a function of the dialogue state. In order to allow dialogue policy transfer across different domains with different slots and speech-acts, the common dialogue policy part would have a translation function that can translate the target domain states and target domain actions to the source domain states and source domain actions. The unified model can be formulated as

$$Q^t(H^t, Y^t) = (1 - O^p(H^t))Q_g(f(H^t), f(Y^t)) + O^p(H^t)Q_p(H^t, Y^t), \tag{1.1}$$

where $Q_g()$ is the domain-independent common Q-function, $Q_p()$ is the domain-dependent Q-function, $O^p()$ is the weighting factor of the domain-dependent Q-function $Q_p()$, and $f()$ is a cross-domain mapping function that translate the target domain state H^t and action Y^t to the corresponding source domain state and action. For the research problems mentioned above, we can have a specific model specified for each problem based on the unified model. And we summarize the key contributions as follows:

1. In order to avoid the possible negative transfer caused by transferring knowledge across different persons with different preferences, we propose the system "PETAL", to separate the common dialogue state model from the personal preferences model. In the traditional rule-based dialogue system, the dialogue states, system speech-acts and the user speech-acts have to be manually defined, and it is difficult to reuse the system when the dialogue states and speech-acts are hard to define. In personalized task-oriented dialogue systems, the data collected from each user is usually insufficient for training a traditional learning-based dialogue system. A personalized dialogue system trained on a small dataset is likely to fail on unseen but common dialogues due to the overfitting. Traditional transfer learning dialogue systems transfer dialogue data across users with different preferences, but they do not model

the difference between different users. As a result, the transferred policy might harm the performance of the personalized dialogue systems in the target domain, like making wrong recommendations. The proposed transfer learning dialogue system, PETAL system, does not require a manually-defined ground-truth state space and it can model the personalized future expected reward. It can transfer the common dialogue policy across users with different preferences and avoid the negative transfer. The transfer reinforcement learning personal model can be formulated as

$$Q^t(H^t, Y^t) = Q_g(H^t, Y^t) + w_p Q_p(H^t, Y^t), \tag{1.2}$$

where $Q_g()$ is the shared common Q-function for all users, $Q_p()$ is the personalized factor for each user and w_p is a learned personal weight factor. The general Q-function $Q_g()$ is shared across all users, while the personal Q-function $Q_p()$ learns the personal preference of each user.

2. In order to transfer more fine-granularity phrase-level dialogue knowledge and avoid the negative transfer caused by different preferences of different persons, we propose a fine-granularity end-to-end personalized dialogue policy transfer model. Although traditional multi-tasking can be used to transfer dialogue knowledge across different users by sharing training dialogues, current neural network transfer models work on the entire sentence. However, the phrases concerning personal information from different users should not be transferred across users at all, for it might cause a personal information leakage and negative transfer. For example, an address from one user might be mistakenly used to make suggestions to another user. In this thesis, we propose a personalized decoder that can transfer shared phrase-level knowledge between different users while keeping the personalized information of each user intact. A novel personal control gate is introduced, enabling the personalized decoder to switch between generating the personalized phrases and the shared phrases. The proposed personalized decoder model can be easily combined with various deep models and can be trained with reinforcement learning. The fine-granularity end-to-end personalized dialogue policy model is formulated as

$$Q^t(H^t, Y^t) = (1 - O^p(H^t))Q_g(H^t, Y^t) + O^p(H^t)Q_p(H^t, Y^t), \tag{1.3}$$

where $Q_g()$ is the general Q-function that can generate common phrases, $Q_p()$ is the personal Q-function that can generate phrases according to the personal preference, and $O^p(H^t)$ is the fine-granularity control gate function that decides which model to use based on the current dialogue context.

3. In order to transfer dialogue policies across domains with different slots and different speech-acts, we propose a dialogue policy transfer model "PROMISE" that can simultaneously transfer across different speech-acts and slots without additional database information. Existing cross-domain dialogue policy transfer models cannot transfer across domains with different speech-acts, and the slot-mapping matrix is either based on common slots or built with human heuristic and requires the access of the full database. However, in many situations there might not be any common slots and the full database access might not be available, so the existing methods can only be used in limited situations. The proposed PROMISE model can learn to align different speech-acts and slots automatically at the same time, and it does not require common slots or full database access, and it is a learning-based method which does not depend on human heuristic. The cross-domain transfer dialogue policy is formulated as

$$Q^t(H^t, Y^t) = Q^s(f(H^t), f(Y^t)), \tag{1.4}$$

where $Q^t()$ is the Q-function in the target domain, $Q^s()$ is the Q-function in the source domain and $f()$ is a learned cross-domain mapping function that can map the target domain state and action to the source domain. $Q^s()$ is trained on abundant data in the source domain so it is sufficiently good. With the learned cross-domain slot mapping and speech-act mapping matrix, the source policy can be transferred to the target domain. The core problem is to design a learning algorithm to learn the cross-domain mapping function $f()$.

1.5 Thesis Organization

The thesis is organized into eight chapters as shown in Figure 1.5. In Chapter 1, we introduce the motivation of the transfer reinforcement learning problem in task-oriented dialogue systems, we also give a problem overview, the key challenges and our contributions in this thesis. In Chapter 2, we briefly review the related works in reinforcement learning and transfer learning, and then introduce previous works in the transfer reinforcement learning problem. We summarize a few most related works of transfer reinforcement learning in task-oriented dialogue systems. In Chapter 3, we give a detailed survey of the task-oriented dialogue system. We will introduce the components in typical modular task-oriented dialogue systems and the end-to-end task-oriented dialogue systems, we will also review previous transfer learning works in task-oriented dialogue systems. In Chapter 4, we formulate the problem of transferring common dialogue structure across different users in a personalized task-oriented dialogue system, and we propose the PErsonalized Task-oriented diALogue (PETAL) model for this problem. In Chapter 5, we formulate the problem of transfer-

ring fine-granularity knowledge in a personalized task-oriented dialogue system, and we propose a personalized decoder model for this problem. In Chapter 6, we formulate the problem of transferring dialogue policy across different domains, and we propose the Policy tRansfer across dOMaIns and SpEech-acts (PROMISE) model to solve this problem. In Chapter 7, we summarize the above models and propose a unified transfer reinforcement learning model, and we show how to learn the unified model under different settings with different granularities. In Chapter 8, we conclude the thesis by summarizing the contributions of this thesis, and we also point out some promising future directions.

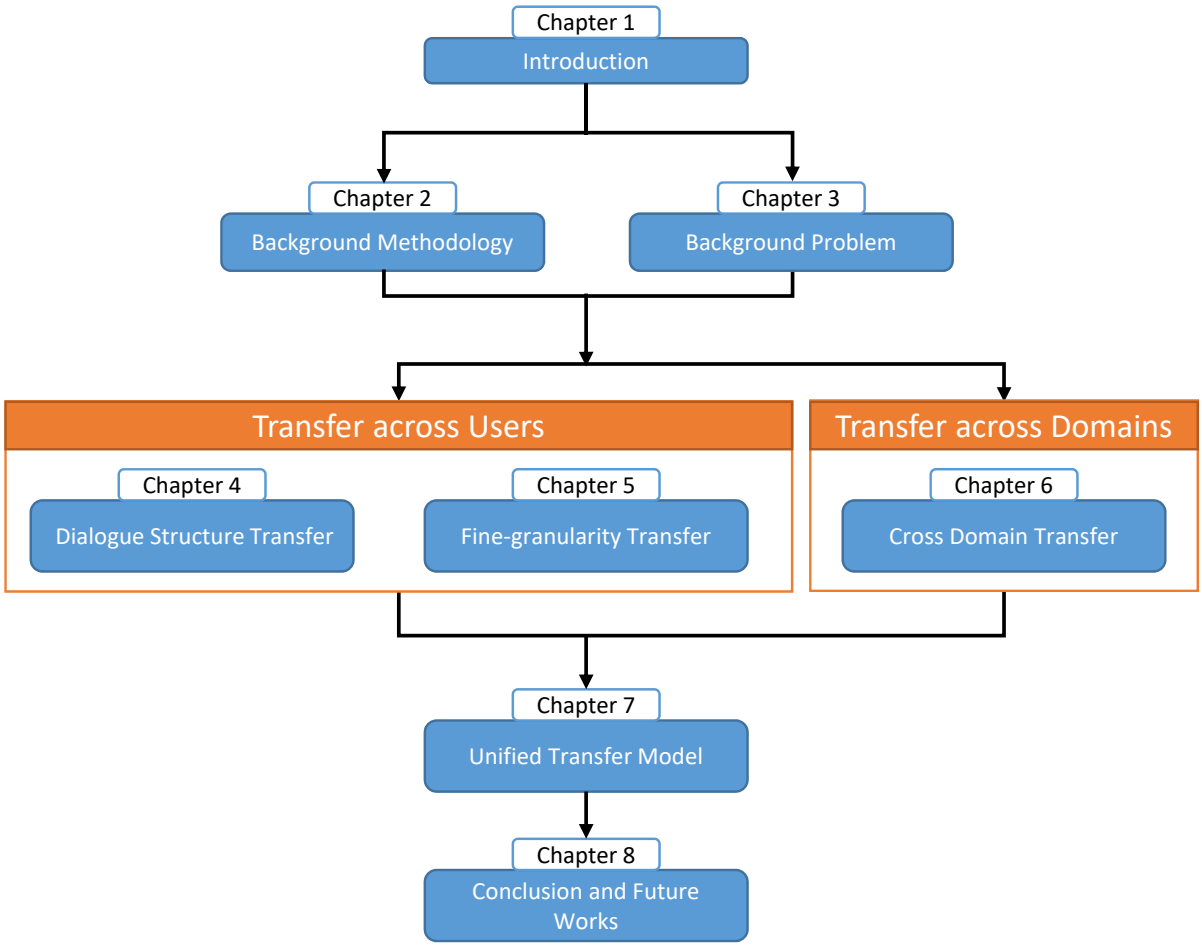


Figure 1.5: The organization of this thesis.

CHAPTER 2

RELATED WORK PART 1: TRANSFER REINFORCEMENT LEARNING

In this section, we briefly introduce the related works in reinforcement learning, transfer learning and reinforcement transfer learning. We also systematically review the related works of transfer reinforcement learning in task-oriented dialogue systems, and show **the relationship between our works and the existing works**.

2.1 Reinforcement Learning

Reinforcement learning [135] aims to train an agent to perform a series of actions under different states, with the goal of maximizing the total reward. Compared with supervised learning, reinforcement learning is different since there is no ground-truth action label for each state, so the agent can only learn from the total reward signal. After the agent performs a task, it will receive a reward signal related to the performance of this agent, and the agent is expected to learn from this feedback reward. The reward might include a success reward indicating whether the task is successfully completed and a negative step reward reflecting the cost of taking each action. By maximizing the reward, the agent can learn a policy to complete the task with the least cost. There are a lot of applications of reinforcement learning, such as robotics control, drone control, dialogue system, game playing, etc. Without the need for ground-truth labels on each state, reinforcement learning agents have the potential to perform better than any human. Recent progress shows that reinforcement learning agents can learn to play various games better than human beings. For example, the latest AlphaGo Zero can learn to play Go by being told whether it wins or not, without a teacher telling it the correct action at each step.

Reinforcement learning is widely used to model dialogue policies in task-oriented dialogue systems. A popular way to model dialogue management module is the Markov Decision Process (MDP) [14, 76, 152, 125]. The dialogue manager is modelled as a system that tries to achieve a goal through a series of interaction with the user. The information about the current dialogue situation is modelled with a state, and the system can choose the optimal system action, then the next dialogue state depends only on the current state and the action taken by the system, which

follows Markov assumption. In some situations, due to possible typo and speech recognition error, the dialogue action and the current state of the system can not be fully determined. Partially Observable Markov Decision Process (POMDP) [184] is used to model the dialogue policy. At each step, instead of tracking the ground-truth dialogue state, the POMDP system keeps track of a probability distribution on all possible dialogue states. The probability distribution of all possible dialogue states is called the belief state. The belief state is assumed to follow Markov assumption, which means the next belief state depends only on the current belief state and the action taken. The POMDP policy will decide the best system action based on the current belief state instead of the true state. Actually, POMDP can be modelled by an MDP in the belief state space.

Dialogue reinforcement learning is different from other reinforcement learning problems in the following ways. Firstly, unlike chess and Go, dialogues do not have a clear ground-truth state space. Secondly, unlike some physical control system, the transition model of a dialogue is unknown. Thirdly, unlike some UAV control systems that have a fixed number of control signals, the number of possible reply sentences of a dialogue system is not enumerable. Finally, dialogues have the natural ambiguity of a language. The meaning of a word in a sentence is conditional on its context and the same word might have completely different meanings. And the same meaning can also be expressed via different sentences in many different ways. Due to these unique characteristics of the dialogue systems, training a dialogue policy requires a lot of training data.

2.1.1 Q-learning

Q-learning [161] is a popular model-free reinforcement learning algorithm used in value-function based reinforcement learning model. Usually, the policy will be modelled with a action-value function call Q-function, which is defined as the expected return starting from state H, taking action Y, and then following policy π by

$$Q(H_n, Y_n) = \mathbb{E}(\sum_{k=0}^{\infty} \gamma^k r_{n+k} | H_n, Y_n). \tag{2.1}$$

In the batch training setting which does not require exploration, the policy can be specified by

$$Y = \arg\max_{\tilde{Y}'} Q(H, Y').$$

According to the bellman equation [11], an optimal Q-function should satisfy the following equation

$$Q(H_n, Y_n) = r_n + \max_{Y'} \lambda Q(H_{n+1}, Y') - Q(H_n, Y_n),$$

where H_n is a state, Y_n is an action, r_n is an immediate reward received after taking action Y_n, and H_{n+1} is the next system state. The bellman equation implies the training objective of Q-learning, and Q-learning updates the Q-function by

$$Q(H_n, Y_n) \leftarrow Q(H_n, Y_n) + \alpha(r_n + \max_{Y'} \lambda Q(H_{n+1}, Y') - Q(H_n, Y_n)),$$

where the $< H_n, Y_n, r_n, H_{n+1} >$ is a training trajectory.

2.1.2 Gaussian Process Reinforcement Learning

If Gaussian process is used to model the Q-function, then we will have to use the Gaussian process reinforcement learning. Gaussian process [41, 33] is non-parametric and can avoid the limitation when the solution is constraint by the human-chosen basis. The Q-function is formulated by

$$Q^{\pi}(H, Y) \sim \mathcal{GP}(m(H, Y), k((H, Y), (H, Y))) \tag{2.2}$$

where H is the state, Y is the action, $m(H, Y)$ is the prior mean function and $k((H, Y), (H', Y'))$ is the kernel function. The kernel function $k((H, Y), (H', Y'))$ can be factorized into separate kernels over state space and action space by

$$k((H, Y), (H', Y')) = k_s(H, H')k_a(Y, Y').$$

The prediction process is as follows. Given training state-action sequences

$$\mathbf{B} = [(H_0, Y_0), \cdots, (H_n, Y_n)]^T$$

and the corresponding immediate rewards

$$\mathbf{r} = [r_0, \cdots, r_n]^T,$$

the $Q^{\pi}(H, Y)$ for any state-action pair (H, Y) is given by

$$Q(H, Y)|\mathbf{B}, \mathbf{r} \sim \mathcal{N}(\bar{Q}(H, Y), cov((H, Y)(H, Y))),$$

where the posterior mean is given by

$$\bar{Q}(H, Y) = \mathbf{k}(H, Y)^T \mathbf{H}^T (\mathbf{HKH}^T + \sigma^2 \mathbf{HH}^T)^{-1} (\mathbf{r} - \mathbf{m}), \tag{2.3}$$

and the covariance is given by

$$cov((H, Y), (H, Y)) = k((H, Y), (H, Y)) - k(H, Y)^T \mathbf{H}^T (\mathbf{HKH}^T + \sigma^2 \mathbf{HH}^T)^{-1} \mathbf{Hk}(H, Y), \tag{2.4}$$

where $\mathbf{m} = [m(H_0, Y_0), \cdots, m(H_n, Y_n)]^T$, $\mathbf{k}(H, Y) = [k((H_0, Y_0), (H, Y)), \cdots, k((H_n, Y_n), (H, Y))]^T$, $\mathbf{K}$ is the Gram Matrix [109], $\mathbf{H}$ is the band matrix with diagonal $[1, -\gamma]$, and σ^2 is an additive noise factor controlling the variability in the Q-function.

Gaussian process Q-learning is powerful none-parametric model, and it can void the limitation of a human selected basis. It can be trained with less training samples. However, Gaussian process based models have very high computational complexity and can only process small datasets.

2.2 Transfer Learning

Transfer learning [102] can leverage the knowledge from a source domain to effectively learn a model in a target domain with little training data. With the transferred knowledge, the transfer target model can achieve better performance with the same amount of target training data, or achieve the same level of performance with less amount of target training data or less training time. From the perspective of features and labels, transfer learning can be used to transfer knowledge across domains with different feature distributions, different feature sets, and different labels. Homogeneous transfer learning aims to transfer across domains with the same feature space, but with different feature distributions, e.g., from book reviews to movie reviews, from sketches to real images. Heterogeneous transfer learning studies the problem of knowledge transfer across domains with different feature sets, e.g., from movie posters to movie reviews. From the perspective of the transferred knowledge, transfer learning can be categorized into instance-based, feature-based and model-based transfer learning. Instance-based transfer learning transfers data instances across tasks, feature-based transfer learning learns a feature mapping from a source domain to a target domain, and model-based transfer learning transfers a source model to the target domain. The applications of transfer learning ranges from image classification [81], semantic classification [43], WiFi localization [55], recommendation systems [103] and more.

However, traditional transfer learning can only work on supervised learning problems, and it could not deal with reinforcement learning problems that have different states. In the supervised transfer learning problems, the major domain differences are features and labels, while in the reinforcement learning problems, there could be possible domain differences in state spaces, action spaces, state transition probabilities, and the reward functions. Building a cross-domain mapping requires learning the state mapping, the action mapping, the transition mapping and the reward mapping simultaneously, which is much more difficult.

2.3 Transfer Reinforcement Learning

Transfer reinforcement learning seeks to leverage the knowledge in a source domain to help to build a policy in a target domain. Compared with the traditional transfer learning, the transfer reinforcement learning [141] is more general, since it has to consider the cross-domain state mapping, action mapping, transition function mapping and reward function mapping. For a detailed survey of transfer reinforcement learning, please refer to [141]. The differences between transfer learning, reinforcement learning, and transfer reinforcement learning are listed in the Table 1.1. Note that the term task and domain can be used interchangeably.

From the perspective of state space, action space, transition function, reward function and the availability of cross-domain mapping, transfer reinforcement learning techniques can be classified into the following categories.

1. Same state space and action space. In this setting, the source task and the target task have the same state space and action space, or the source task and the target task are formulated such that they have the same abstracted state space and action space.

2. Different state variable and actions with inter-task mapping. In this setting, the source task and target task do not have the same state space or action space, but a cross-domain mapping is provided as input.

3. Learning inter-task mapping. The source and target tasks have different state spaces and action spaces, and the inter-task mapping is not provided. In order to achieve knowledge transfer, an effective cross-domain mapping has to be learned first.

From the perspective of the transferred knowledge, transfer reinforcement learning techniques can be classified into the following categories.

1. In value function transfer, the value function is transferred from the source domains to the target domain, and this is the most straight-forward way. The assumption is that source and target domains have similar value functions.

2. In policy transfer, the policy is transferred from the source domains to the target domain. Different from the value function transfer which transfers Q-value, policy transfer directly transfers the selected action under a certain state and it is invariant to the scale of the value function. The assumption is that the source policy and target policy are similar in some way.

3. In option transfer, the options (policies for sub-tasks) [137] are transferred from the source domains to the target domain. The assumption is that there are some common sub-tasks between the source domains and the target domain.

4. In instance transfer, some data trajectories are transferred from the source domains to the target domain. The assumption is that the state transition functions and reward functions in the source and target domains have some local similarities.

2.3.1 Transferability: the Similarity between Domains

In this section, we review some related works on measuring the similarity between MDPs in reinforcement learning domains/tasks. Measuring domain similarity is a core problem in transfer learning, it helps the user to decide whether it is suitable to perform transfer learning from a source domain to a certain target domain, and it is the key for source domain selection in the multi-source transfer learning problem.

There are four major categories of similarity measures between reinforcement learning tasks.

1. Transfer-advantage-based domain similarity methods directly perform the transfer and measure the domain similarity by the performance improvement in the target domain.

2. Feature-based domain similarity methods find a common feature space between the source and target domains and measure the similarity of the transition functions, reward functions or policies in this common feature space.

3. Heuristic-based domain similarity methods compare two domains based on heuristics such as the difference of the transition functions and the reward functions in the source domain and the target domain.

4. Graph-based domain similarity methods project the source and target MDPs into graphs and compute the graph similarity.

Transfer-Advantage-based Domain Similarity

Transfer-advantage-based domain similarity methods directly perform the transfer and measure the domain similarity by the performance improvement in the target domain. Carrol *et al.* [19] discuss the desirable properties of a task similarity metric, and they conclude that there is no such a single best transfer learning similarity measure for all tasks. They propose four task similarity

measures. The first one is the usefulness of the source policy actually being transferred to the target domain, the second task similarity is the policy overlap in the source and target tasks, the third task similarity is the mean square error of the Q-function in the source and target domains, and the final task similarity measure is the mean square error the expected immediate reward in each corresponding state action pair. Fernández *et al.* [30] propose to use empirical transfer advantage to measure the similarity of domains. Before each run, a source policy is selected from the source library according to the softmax of the reuse gain score. The reuse gain score of a source policy is defined as the averaged reward gained while exploring the target domain with this source policy.

Transfer-advantage-based domain similarities can most accurately reflect the performance improvement of transfer learning methods. However, the transfer-advantage-based domain similarities are difficult to compute before actually learning the target domain tasks.

Feature-based Domain Similarity

Feature-based domain similarity methods aim to find a common feature space between the source and target domains and measure the similarity of the transition functions, reward functions or policies in this common feature space. Konidaris *et al.* [65, 67] assume tasks are related because all tasks are taken by the same agent. As a result, all tasks share a set of common features, that is the sensor input of this particular agent. They propose the agent-space which is the space expanded by the features of this agent, and then compare MDPs based on their similarities over these related features. In this work, transfer learning takes place through functions defined over these related features in the agent-space. They learn a set of options defined on this agent-space, and then reuse these learned options in future tasks to improve performance. The assumptions of this work hold when all tasks are taken by the same agent, but they are not applicable to tasks that are taken by different agents. And it is hard to define the agent-space automatically in advance. Feature-based domain similarities can be computed before actually conducting the transfer learning experiments.

Heuristic-based Domain Similarity

Heuristic-based domain similarity methods compare two domains based on the heuristics such as the difference of the transition functions and the reward functions in the source domain and the target domain.

Lazaric *et al.* [72] propose to selectively choose samples from source domains and transfer to the target domain. In order to select appropriate source tasks, they propose a task compliance score

based on the similarity of the reward functions and the state transition functions. In order to select samples from a source task, they propose the sample relevance scores. They propose to transfer the samples that are from a high compliance score source task but are far away from the target domain, since these samples might contain information about the unexplored region in the target domain.

Ammar *et al.* [2] propose to measure the distance between MDPs by measuring the difference between the transition functions of two MDP models. They use a Restricted Boltzmann Machine-based model to approximate the transition dynamics of the source MDP, and then they calculate the reconstruction error of this source dynamics model on the target domain data, and the final distance between the source and target domains is measured based on the reconstruction error.

Song *et al.* [126] propose two metrics for MDP similarities. They firstly define the concept of homogeneous MDPs and extend Fernández's metrics to the merged graph of two homogeneous MDPs. Then the Kantorovich metric and the Hausdorff metric are used to compute the distances between two MDPs. Kantorovich metric is computationally inefficient, while Hausdorff is more efficient but it might miss out some similar MDPs due to outliers.

For any state H, H' in the same MDP, their distance $d(H, H')$ is defined as

$$d(H, H') = \max_{\forall Y}\{|r_H^Y - r_{H'}^Y| + cT_K(d)(P_H^Y, P_{H'}^Y)\},$$

where $c \in (0, 1)$, P_H^Y, (resp. $P_{H'}^Y$) is the transition probability when action Y is taken at state H (resp. H'), and $T_K(d)(P_H^Y, P_{H'}^Y)$ is the Kantorovich distance between the two probabilistic transitions. MDPs are homogeneous if they satisfy the following three conditions. Firstly, the state representations are equivalent, i.e. they have the same state variables. Secondly, there is a one-to-one correspondence between their action spaces. Thirdly, the two MDPs have the same reward when reaching the same sub-goals or goals. For two homogeneous MDP $M_1 = \{\mathcal{H}_1, \mathcal{Y}_1, R_1, P_1\}$ and $M_2 = \{\mathcal{H}_2, \mathcal{Y}_2, R_2, P_2\}$, the distance between any state $H \in M_1$ and any state $H' \in M_2$ is defined as

$$d'(H, H') = \max_{\forall Y}\{|(r_1)_H^Y - (r_2)_{H'}^Y| + cT_K(d')((P_1)_H^Y, (P_2)_{H'}^Y)\},$$

where $(r_1)_H^Y$ (resp. $(r_2)_{H'}^Y$) and $(P_1)_H^Y$(resp. $(P_2)_{H'}^Y$)) are the immediate reward and the probability transition function in M_1 (resp. M_2) and $T_K(d')((P_1)_H^Y, (P_2)_{H'}^Y)$ is the Kantorovich distance between the two probabilistic transitions. For two state probabilistic transtion functions $(P_1)_H^Y$ and $(P_2)_{H'}^Y$ in different MDPs, their Kantorovich distance [27] is

$$T_K(d') = \min_{\lambda_{k,t}, k\in[1,|\mathcal{H}_1|], t\in[1,|\mathcal{H}_2|]} \sum_{k=1}^{|\mathcal{H}_1|}\sum_{t=1}^{|\mathcal{H}_2|} \lambda_{kt} d'(H_k, H'_t)$$

s.t. $\forall k, \sum_t \lambda_{k,t} = (P_1)^Y_{HH_k}$, $\forall t, \sum_k \lambda_{k,t} = (P_2)^Y_{H'H'_t}$, $\forall k, t, \lambda_{k,t} \geq 0$, where $H_k \in \mathcal{H}_1$, $H_t \in \mathcal{H}_2$, $|\mathcal{H}_1|$ represent the number of states in M_1 and $|\mathcal{H}_2|$ is the number of states in M_2.

The Hausdorff metric computes the distance between two subsets of a metric space. It is the largest distance of all the distances from a point in one set to the nearest point in another set. Given two MDPs M_1 and M_2, their Hausdorff-metric-based distance is defined as

$$\Phi(\mathcal{H}_1, \mathcal{H}_2) = \max\{\max_{H \in \mathcal{H}_1} \min_{H' \in \mathcal{H}_2} d'(H, H'), \max_{H' \in \mathcal{H}_2} \min_{H \in \mathcal{H}_1} d'(H, H')\},$$

where d' is the cross MDP state similarity metric defined above. The Kantorovich metric is used to find an optimal allocation of transportation resource, where each state in one MDP is regarded as a supply node and each state in the other MDP is regarded as a demand node, and the goal is to find a way to move the resource from the supply nodes to the demand nodes with minimal cost. The transportation unit cost between each pair of nodes is equal to the distance of the corresponding states in these two MDP. Given two MDPs M_1 and M_2, their Kantorovich-metric-based distance is defined as

$$\Psi(\mathcal{H}_1, \mathcal{H}_2) = \min_{l_{ij}} \sum_{i=1}^{|\mathcal{H}_1|} \sum_{j=1}^{|\mathcal{H}_2|} l_{i,j} d'(H_i, H'_j),$$

s.t. $\forall i, \sum_{j=1}^{|\mathcal{H}_2|} l_{i,j} = \frac{1}{|\mathcal{H}_1|}$, $\forall j, \sum_{i=1}^{|\mathcal{H}_1|} l_{i,j} = \frac{1}{|\mathcal{H}_2|}$, $\forall i, j, l_{ij} \geq 0$, where $d'(H, H')$ is the cross MDP state similarity metric defined above. Having defined the Hausdorff-metric-based distance and the Kantorovich-metric-based distance, the authors propose to transfer the value function between similar states with a one-to-one cross MDP state mapping. With the computed $l_{i,j}$ in the Kantorovich-metric-based distance, the authors propose to transfer value function with a many-to-many cross MDP state mapping.

Heuristic-based domain similarities are easy to compute and can be computed before actually conducting the transfer learning experiments. However, there is no guarantee that transferring from a heuristically closer domain leads to bigger transfer learning performance improvement.

Graph-based Domain Similarity

Another way to model the similarity between MDPs with different states and actions is to model MDPs with directed graphs. Each state can be viewed as a node, and each possible state transition (H_1, Y, H_2) can be viewed as a directly edge E_{H_1,Y,H_2} from node V_{H_1} to node V_{H_2} with edge weight p_{H_1,Y,H_2}, meaning if the action Y is taken in a state H_1 the probability of going to node H_2 is p_{H_1,Y,H_2}, note that there could be multiple edges between two nodes, because multiple actions could result in the transition from one state to another state.

In this way, the similarity between two unaligned MDPs becomes the similarity of two directed graph [68]. For the problem of graph similarity between two graphs that does not require an exact match, there are three main categories.

1. The first category is the graph isomorphism/edit distance methods, in which the similarity of graphs are measured by whether the two graphs are isomorphism [106], or one is isomorphic to a sub-graph of another graph, or whether they have common sub-graphs. The edit distance method is to find the minimum number of operations needed to modify one graph to another graph.

2. The second category is the feature-based methods. The key idea is that similar graphs might have similar statistical features, such as degree distributions, diameters, eigenvalues as in [162]. Based on these statistical features, two graphs can be compared to calculate graph similarity.

3. The third category is the iterative methods. The key idea is that two nodes are similar if their neighbours are similar. In each iteration, nodes update their similarity score according to their neighbours until convergence is reached. Famous algorithms include the similarity flooding algorithm [89], the SimRank algorithm [56], the recursive similarity algorithm proposed by Zager *et al.* [186], and the message-passing-based algorithm proposed by Bayati *et al.* [10].

Graph-based domain similarities are built upon solid theory, but some exact methods are computationally expensive.

2.3.2 Transfer from Multiple Sources

In this section, we discuss how to transfer from multiple sources. Many transfer reinforcement learning methods aim to transfer policy from a single source domain. However, it is possible that the useful knowledge is distributed in many source domains, transferring from only one single domain might have many limitations. One promising direction is to transfer from multiple source domains. The core problem is to identify the similarities between various source domains and the target domain, which is defined in Section 2.3.1. Based on the domain similarity score, knowledge from the related source domains can be transferred, while the unrelated source domains should be ignored.

Just like the single-source transfer reinforcement learning, the multi-source transfer reinforcement learning can use the same instance similarity measure, transfer the same type of knowledge and use the same learning objective functions.

1. Many multi-source transfer reinforcement learning methods propose to define or learn a similarity score between an instance in the target domain and an instance in the source domains, and proceed by transferring various instances with high/low similarity to the target domain [72].

2. Multi-source transfer reinforcement learning methods can also transfer the value function [126], the policy [30], the options [156] and the instances [72] from the source to the target domain, and they might share the same assumptions with single-source transfer reinforcement learning methods, i.e., the source domain and the target domain have a similar value function or policy, have some common sub-tasks, or have some local similarities in the transition functions and the reward functions.

3. Multi-source transfer reinforcement learning can also use all kinds of learning objectives [108] in the single-source transfer reinforcement learning methods.

Different from the single-source transfer reinforcement learning, the multi-source transfer reinforcement learning methods have an additional decision step of selecting one or multiple source domains, and they can transfer knowledge from multiple source domains simultaneously.

1. Unlike single-source transfer reinforcement learning where there is only one source task, multi-source reinforcement learning methods have to choose source domains before any instances or model parameters can be transferred to the target domain, and the domain similarities defined in Section 2.3.1 lie in the core of this problem. Fernández *et al.* [30] propose the reuse gain score to select source policies. The reuse gain score of a source policy is defined as the averaged reward gained while exploring the target domain with this source policy. Lazaric *et al.* [72] propose to selectively choose samples from source domains and transfer to the target domain. Before selecting the instances, firstly an appropriate source domain is identified by a task compliance score, which is defined as the probability of the target domain data being sampled from the corresponding source domain transition function and reward function. In this way, we can identify the source task which is most likely to generate the target domain data, and then the data in this source domain can be transferred to the target domain to improve performance.

2. Multi-source transfer reinforcement learning can transfer from more than two policies simultaneously. Song *et al.* [126] propose two metrics for MDPs similarities, and they propose a weighted transfer method based on the proposed Kantorovich-distance-based similarity measures. For each target state, the similarity weights between source states and the target state are calculated. The more similar a source state is, the more influence it has on the target state. In this way, one target domain state might be affected by multiple source domain data points. Lazaric *et al.* [72] propose to transfer source domain samples according to the task compliance score of each source domain, in this way, more than one source domains can contribute simultaneously to the target domain via the randomly selected data samples.

In conclusion, multi-source transfer reinforcement learning is a promising generalization to the single-source transfer reinforcement learning methods. It can leverage existing single-source transfer methodologies and it has better performance since it can rely on more similar source domains and it can transfer from multiple sources simultaneously.

2.3.3 Task Complexity

In this section, we discuss the complexity of a task-oriented dialogue system. But before proceeding to the complexity of the task-oriented dialogue system, we need to define the complexity of solving a general MDP problem.

Littman *et al.* [79] discuss the computational complexity of solving an MDP with three different methods. Assume the MDP is represented by $(\mathcal{H}, \mathcal{Y}, P, R)$, where $N = |\mathcal{H}|$ is the number of states and $M = |\mathcal{Y}|$ is the number of actions. The state transition function P is defined as follows. For all $H_i, H_{i+1} \in \mathcal{H}$, $Y \in \mathcal{Y}$, the probability of taking action Y in state H_i and transit to H_{i+1} is defined as

$$P^{Y}_{H_i, H_{i+1}} = Pr(H_{i+1}|H_i, Y).$$

Assume the transition function P and the reward function R are encoded with an $N * N * M$ probability matrix, and both $\mathcal{H}$ and $\mathcal{Y}$ are finite. The authors focus on the infinite-horizon case, where there are infinite number of training sequence. The three algorithms and their complexity are listed as follows.

1. Linear programming algorithm turns the MDP problem into a linear programming problem with NM constraints and N variables, so it could be solved in polynomial time in N, M. However, the existing polynomial time algorithms are extremely slow and are rarely used.

The most popular and practical methods are variations of Dantzig's simplex method [24]. Although simplex methods seem to work well, Klee and Minty [62] show that these algorithms take exponential running time in the worst case.

2. Policy iteration algorithm [54] iteratively alter between policy evaluation and policy improvement in each iteration and it has a complexity of $(O(MN^2)+O(N^3))*l$, where l is the number of iterations. Each policy improvement has $O(MN^2)$ complexity and each policy evaluation has $O(N^3)$ complexity, so the policy iteration algorithm runs in polynomial time if and only if the number of iterations l required is polynomial. However, Melekopoglou *et al.* [88] show a family of counterexamples that the policy iteration method runs in an exponential number of iterations. So the policy iteration is a pseudo-polynomial-time algorithm.

3. Value iteration algorithm [11] iteratively updates the value function, and its complexity is $O(N^3) * l$, where l is the number of iterations. The running time for each iteration is $O(N^3)$. It is shown that the total-cost functions are guarantee to converge to the optimal total-cost function, and the policy associated will converge to the optimal policy in a finite number of iterations. Again, value iteration is polynomial if and only if the total number of iterations required is polynomial. Tseng *et al.* [146] show that for a fixed discount factor γ, the value iteration takes polynomial time and its running time grows faster than $1/(1-\gamma)$.

Now we discuss the complexity related factors in a task-oriented dialogue system.

1. Number of Speech-acts $|A|$. Speech-acts are the set of all possible user intentions. The dialogue states and dialogue actions consist of speech-acts, so the more speech-acts a task has, the more complex the task is.

2. Number of Slots $|S|$. Slots are the set of information we need to collect for each task. Since a dialogue state has to record both the user's constraints on each slot and the questions about each slot, and most dialogue actions have slots, the more slots in a task, the more complex the task will be.

3. Number of Slot-values $|V|$. Slot-values are the set of all possible values for a slot, different slot-value represents different user preferences. In order to model user preference and make personalized suggestions, the dialogue states and the dialogue actions need to record information about the slot-value of each slot. The more slot-value, the more complex a task will be.

In a typical dialogue system where a dialogue state contains a user's speech-act, a set of user constraints on all slots and a set of binary user question indicator on all slots, the number of all possible dialogue states is $O(|A| * |V|^{|S|} * 2^{|S|})$. A typical user action in a dialogue system is a sentence with a user speech-act, a user slot and the corresponding slot-value, so the number of all possible dialogue actions is $O(|A| * |V| * |S|)$. If the user is allowed to specify the slot-value for an arbitrary number of slots in one sentence, then the number of all possible dialogue actions is $O(|A| * |V|^{|S|})$.

2.3.4 Transfer between Complex Domains

In this section, we discuss how to transfer policies between complex domains. Firstly we give a definition of a complex domain, then we define and analyze the problem and identify the core problem, finally, we conduct a brief survey of the existing related works.

A complex domain has the following characteristic:

1. Has a large number of states. For example in a task-oriented dialogue system, if the number of slots is big and if different user choices require different dialogue policies (the dialogue state must contain slot-value chosen by the user), then the number of all states will be exponentially larger.
2. The domain task might have sub-tasks.

The problem of transfer reinforcement learning between complex domains takes two inputs, a complex source domain with plenty of training data and a complex target domain with little training data. The problem output is a policy in the target domain, and the goal is to improve the target domain policy by transferring knowledge from the source domain.

The challenges of the transfer reinforcement learning between complex domains are listed as follows:

1. Learning a policy in one domain might need much data, since flat reinforcement learning may suffer from the curse of dimension [9]. For example in a task-oriented dialogue system, if the number of slots is big and the number of different slot-values is big, the state space will be very large, thus it will require a huge number of dialogue data to train.
2. Learning the cross-domain state correspondence matrix might either require more source and target training data to align, or require another auxiliary data source to constrain the

correspondence matrix. This is because the parameter space for the cross-domain state correspondence matrix is larger since the number of states is larger.

3. It is hard to reuse whole policies or discover sub-tasks. Since the state space is larger, the policy will be much more complex and it is less likely that a whole source policy can be directly reused, and thus it is natural to consider reusing options for sub-tasks. However, identifying sub-tasks can be also hard.

Hierarchical Reinforcement Learning (HRL) [137, 124, 9] is a principle framework for learning policies in complex tasks. Sutton *et al.* propose the concept of macro-action or option [137] which means a partial policy for a sub-task or a part of the state space, and options allow the agent to learn a task with less training data. HRL methods make use of hierarchical control architectures and learning algorithms. They decompose the big task into a hierarchy of sub-tasks and reuse small local policies across many sub-tasks, and it can greatly improve both the learning speed and the agent performance. Parr *et al.* propose the Hierarchies of Abstract Machines for HRL. As options are policies for a part of the state space with different complexity, they propose to generalize the MDP to the semi-Markov Decision Process (SMDP) [105] where actions can take a variable amount of time to complete. Dietterich *et al.* propose the MAXQ decomposition framework [28] which decompose the value function into an additive combination of value functions of smaller MDPs, later Kulkarni *et al.* propse the HDQN [69] which integrates hierarchical value functions to operate at different temporal scales, and their model shows superior performance on a complicated ATARI game "Montezumas Revenge" with a hierarchical structure.

A new core problem of transferring between complex domains is to identify sub-tasks and reuse sub-tasks, apart from the other core problems in transferring between simple domains. If we can identify sub-goals or sub-tasks in a complex domain, then the complexity of finding a policy will be much smaller since the big problem is decomposed into sub-tasks and the policy only needs to select from a limited number of options [137] for each sub-task. And these options can be transferred from the source to the target domain, based on the domain similarity discussed in Section 2.3.1.

Sub-goal discovery (also known as option discovery and automatic skill acquisition) aims to automatically discover important states and use them as sub-goals. McGovern *et al.* [87] propose to identify bottleneck states in trajectories and use them as sub-goals. Menache *et al.* [90] propose the Q-cut algorithm which uses an efficient Max-Flow/Min-Cut algorithm for identifying bottlenecks, and thus to discover sub-goals. Niekum *et al.* [100] propose to use clustering method to discover

sub-goals or skills. Konidaris *et al.* [66] attempt to discover skills for a robot given abstractions of the surroundings it has to operate with. Lakshminarayana *et al.* [71] propose to find metastable regions in the state space and associate them with abstract states, and the method works even without an exact model of the environment. Bacon *et al.* [6] propose to learn the option policies and their termination conditions simultaneously.

Option transfer methods aim to transfer options from source tasks to the target task. Perkins *et al.* [107] address an reinforcement learning problem in which the transition function can change from time to time, drawn randomly from a probability distribution. Some hand-coded options are provided to the agent. In each run, the agent will learn a policy over these provided options, in order to move quickly to the target state in an environment with a different transition function. Andre *et al.* [3] propose to learn sub-routines between tasks. They look for similarities between some local states of the source and the target domains, and if similar parts can be found in the target domain, the controller or local policies can be directly transferred to the target domain. Foster *et al.* [31] propose to identify sub-tasks in the source domains and reuse them in the target domain where the target state could change but most sub-tasks do not change. They propose to use an expectation maximization algorithm to identify different sub-tasks. When faced with a new task, the agent can learn to reuse the learned options for the sub-tasks, in order to improve the jumpstart and the total reward. Ravindran *et al.* [111] propose the concept of relativized options. They first learn a set of options from the source domains, and when learning in the target domain, the agent can use Bayesian parameter estimation to select source options as well as some possible transformations that can be applied to these source options, in order to maximize the transfer performance in the target domain. Asadi *et al.* [5] propose to identify states as "Locally forms a significantly stronger 'attractor' for state space trajectory" as sub-goals in the source tasks. After learning the options to reach these sub-goals, the agent learns a higher level policy to reuse these options. The model is called Hierarchical Bounded Parameter SMDP, which is proven to be within a bound of the performance of the optimal policy in the original model without options. Konidaris *et al.* [65] propose to view the reinforcement learning problem in the agent-space, instead of the problem space, since for the same agent, the agent sensors and actuators will not be changed, even if the environment is changed. They propose to learn the agent-space options in the source tasks and then reuse them in the target task. They suggested that the agent-space options will be more portable than those in problem-space since problem-space options can only be reused in domains that are very similar.

2.4 Scope of This Thesis

In this section, we formally define the scope of this thesis. We study the transfer learning problem between two dialogue domains, and the methods studied in this thesis are only tested for and thus limited to the task-oriented dialogue systems with the following constraints.

1. The tasks with a fixed set of speech-acts, slots and slot-values, and all speech-acts, slots and slot-values should have appeared in the training data, and no out-of-domain vocabulary will be accepted.

2. The tasks should consist of generally two stages, the entity constraint stage and the inform query stage. In the entity constraints stage, the agent will ask for information needed to find a particular entity, for example the agent might ask about the food preference in order to find a restaurant. In the inform query stage, a unique entity should have been already identified, and the agent will answer questions raised by the user with respect to some slot of this entity, for example, the user can ask about the phone number of a particular hotel.

3. The slots required to complete a task only depends on the domain itself and does not depend on the user constraint slot-values specified by the user, for example, the agent will always ask for the price range of the restaurant regardless of what food the user would like to have.

4. The tasks should not contain recursive sub-tasks. Each slot should have its own set of slot-values, and the slot should not consist of another sub-slot. For example, the food slot should only have slot-values like Chinese food or American Food, it should not have additional sub-slot like whether the food is spicy, whether the food is Halaal, etc.

2.4.1 Transfer Reinforcement Learning in Task-oriented Dialogue Systems

In this section, we systematically review the related works of transfer reinforcement learning in task-oriented dialogue systems. All related works are summarized in Table 2.1, and our works are marked with ✓.

Based on whether the source domain and the target domain use the same set of slots, all works in Table 2.1 can be divided into the left part and the right part.

The left part of the table contains the works where the source domain and the target domain use the same set of slots. Based on whether the source domain user and the target domain user have the same preference (have the same slot-value distribution), these works can be again divided into two

Table 2.1: The related works in transfer reinforcement learning in task-oriented dialogue system. Our works are marked with ✓.

<table>
<tr><td colspan="3">Same Domain $S^s = S^t$, $A^s = A^t$</td><td colspan="2">Cross Domain $S^s \neq S^t$</td></tr>
<tr><td>General
$p(V^s) = p(V^t)$</td><td colspan="2">None Transfer
[173, 174, 183, 75]
[78, 41, 33, 26]</td><td>Same
Speech-act
$A^s = A^t$</td><td>Gaussian Process
Transfer [37, 38, 34],
Bayesian Committee
Machine [36, 39]</td></tr>
<tr><td rowspan="3">Personal
$p(V^s) \neq p(V^t)$</td><td>Finetune</td><td>Linear Model
Transfer [42],
Gaussian Process
Transfer [20]</td><td rowspan="3">Different
Speech-act
$A^s \neq A^t$</td><td rowspan="3">✓in Chapter 6.</td></tr>
<tr><td>Personal
Q-fun</td><td>✓in Chapter 4.</td></tr>
<tr><td>Personal
Word
Gate</td><td>✓in Chapter 5.</td></tr>
</table>

parts. The upper left part contains the works of reinforcement learning dialogue systems without transfer learning, which is not the interest of this thesis. The lower left part contains the works that transfer personalized dialogue policy across different users. Existing methods for transferring dialogue policy across different users are mainly based on model fine-tuning [20, 42], where a model trained in the source domain user data are directly adapted on the target domain user data. However, these works do not model the differences between users, thus it might harm the dialogue performance of the transferred model in the target domain if wrong preferences are transferred. We propose a model with a personalized Q-function to explicitly model user differences for this problem in Chapter 4. In order to transfer fine-granularity common dialogue knowledge in the end-to-end dialogue systems, we propose a model with a novel personal word gating mechanism in Chapter 5.

The right part of the table contains the works where the source domain and the target domain use different sets of slots. Based on whether the source domain and the target domain use the same set of speech-acts, these works can be again divided into two parts. The upper right part contains the works in which the source and the target domains use the same set of speech-acts, and these work aim to transfer dialogue policies across different domains in the same dialogue systems. The works in [37, 38, 34] assume there are some common slots between the source and the target domains, and they do not work in situations when the source and target domains do not have common slots. The works in [36, 39] use the slot normalized entropy for slot matching, and they need to access all data rows in the dialogue database. However, these dialogue policy transfer methods cannot transfer across dialogue systems with different speech-acts, for example, they cannot trans-

fer between systems built by different companies. We propose a model in Chapter 6, which can simultaneously transfer across dialogue systems with different speech-acts and slots, and it does not require common slots or additional database information.

CHAPTER 3

RELATED WORK PART 2: A SURVEY OF TASK-ORIENTED DIALOGUE SYSTEMS

Task-oriented dialogue systems have been used to help people finish tasks in various domains. A typical task-oriented dialogue system consists of 4 components, which are the spoken language understanding module (SLU), the dialogue state tracking (DST) module, the dialogue policy learning (DPL) module and the natural language generation (NLG) module. Many different models are used in each component. What are the differences between these models, and how do we choose from them? In this chapter, we give an updated and organized review of task-oriented dialogue systems.

This chapter is organized as follows. We first give an overview of the structure of a typical task-oriented dialogue system in Section 3.1, then we introduce each of the four typical modules one by one in Section 3.2, Section 3.3, Section 3.4 and Section 3.5, and then we survey the end-to-end dialogue systems in Section 3.6, finally we give a conclusion in Section 3.7. For each dialogue module in Section 3.2, Section 3.3, Section 3.4 and Section 3.5, firstly we introduce the problem formulation, then we survey and compare the single domain dialogue systems without transfer learning, finally we survey and compare the transfer learning related works.

3.1 Overview: Task-oriented Dialogue System

Dialogue systems can be roughly categorized into open-domain dialogue systems and task-oriented dialogue systems. Open-domain dialogue systems [134, 122, 120, 121] do not limit their dialogue domains and can be used for chit-chat. Task-oriented dialogue systems [184] aim to guide a user to finish a certain task in a restricted domain. It can be deployed in a call center, in an on-line chatting platform or a customer service center. It can reduce the need of a human staff and thus reduce cost.

A task-oriented dialogue system takes text input from a user, and generates text responses to the user based on the context and the question. The design of the current task-oriented dialogue systems requires human knowledge on the target problem. Frame-based task-oriented dialogue systems are widely used. In the frame-based systems, a slot is the most basic concept. A slot is a piece of information with a set of possible values. For example, with respect to the choices of

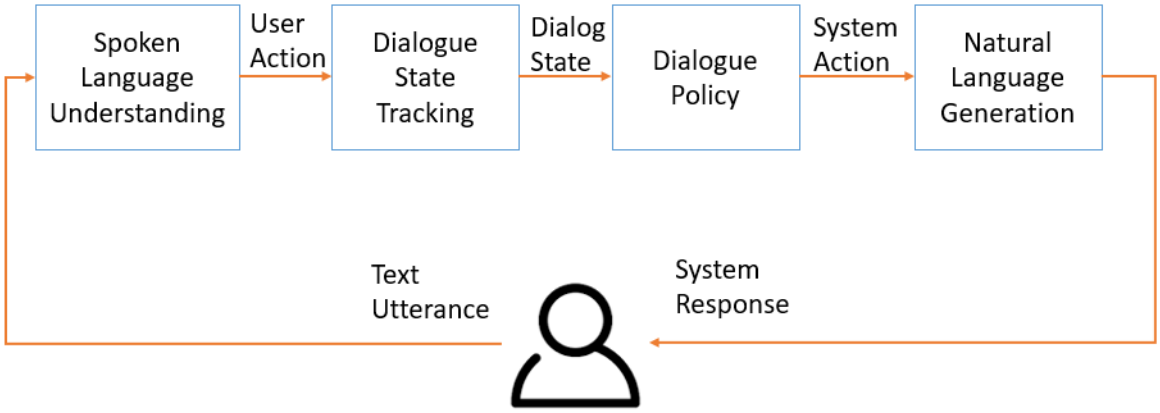


Figure 3.1: The Task-oriented Dialogue System Architecture

food, we can have Chinese food, Japanese food or Indian food, etc. Dialogues are treated as a set of computation on different slot values. Based on the functionality, a dialogue system can be divided into different components. Each component is responsible for a specific sub-task, and the communication between components are done through a set of slots and their values.

There are 4 basic components in a task-oriented dialogue system, and they are the spoken language understanding module (SLU) module, the dialogue state tracker (DST) module, the dialogue policy (DPL) module and the dialogue generation module (NLG). In this thesis, we assume the input is in raw text format instead of voice. The spoken language understanding module [180, 45, 83, 46, 181] is responsible for detecting speech-acts, slots and the corresponding slot-values. The SLU takes a user utterance as input and predicts the speech-acts and slot-values appeared in the user utterance. The dialogue state tracker [73, 133, 159, 51, 187] is responsible for inferring and maintaining dialogue states. The DST takes a parsed user utterance (including speech-acts, slots and slot-values) as input and keeps track of the current dialogue state based on the user utterance and the previous dialogue state. The dialogue policy module [173, 174, 183, 75] takes the current dialogue state as input and decides the next system action to take. The natural language generation module [169] converts the system action back to text response. Among these 4 modules, the dialogue policy module is the core component and it is responsible for deciding the next system action.

Based on the training frameworks, task-oriented dialogue systems can be categorized into modular dialogue systems and end-to-end dialogue systems. The components in modular dialogue systems can be trained or built separately with different objective functions, while end-to-end dialogue systems are trained jointly with a single objective function. Modular dialogue systems have

low inter-modular dependency, thus we can plug-in different components freely. Components can be trained or hand-crafted independently. The data needed to train each component can be obtained easily. However, without a consistent objective function, the whole system with different components might not work well. End-to-end dialogue systems have a single objective function. When trained jointly, the whole system can learn together and achieve a better performance, and it does not require intermediate annotation thus can reduce human efforts. But since all components are optimized jointly, we cannot switch components without retraining. And the training of an end-to-end dialogue system is more difficult.

3.1.1 System Components

Given the dialogue history $H_n = \{\{X_j, Y_j\}_{j=1}^{n-1}, X_n\}$, the task of a task-oriented dialogue system is to predict Y_n. There are 4 components in a task-oriented dialogue system.

1. The spoken language understanding (SLU) module. It takes raw a text utterance X_n as input and predict the abstract user action $\tilde{X}_n$ of the utterance at every time step n.

2. The dialogue state tracking (DST) module. Given a previous dialogue state $\tilde{H}_{n-1}$, an abstract system action $\tilde{Y}_{n-1}$ and an abstract user utterance $\tilde{X}_n$, the dialogue state tracking aims to keep track of the current dialogue state $\tilde{H}_n$.

3. The dialogue policy learning (DPL) module. The policy network takes a dialogue state $\tilde{H}_n$ as input and predict the abstract system action $\tilde{Y}_n$.

4. The natural language generation (NLG) module. It takes an abstract system action $\tilde{Y}_n$ as input and generate the final system response sentence Y_n.

3.2 Module 1: Spoken Language Understanding (SLU)

The spoken language understanding module is responsible for mapping a text utterance to a structured output such as speech-acts, slots and slot-values.

3.2.1 Problem Definition

The spoken language understanding module takes a raw text utterance X_n as input and predict the abstract user action $\tilde{X}_n$ of the utterance at every dialogue turn. The user action $\tilde{X}_n = \{a_n, \mathbf{s}_n = \{s_j = v_j\}\}$ consists of a user speech-act a_n (also known as intention) and a sequence of slot-value pair $\mathbf{s}_n = \{s_1 = v_1, s_2 = v_2, \cdots\}$.

There are two major problems in spoken language understanding, the first one is speech-act/intention classification and the other is slot-filling.

1. Speech-act/intention classification. The problem input is a user utterance X_n and the problem output is the speech-act a_n of the user utterance. Speech-act/intention classification can be viewed as a multi-label classification problem.

2. Slot-filling. The problem is to find all possible slot value pairs $\mathbf{s}_n = \{s_1 = v_1, s_2 = v_2, \cdots\}$ in the user utterance X_n. Slot-filling task can be viewed as a sequential classification problem on the word sequence of a sentence.

Since the speech-act classification is a simple multi-label classification problem, we will focus on the slot-filling problem. And we will focus on learning methods for slot-filling problem, other deterministic method such as parsing will not be considered.

Slot-filling maps a user utterance X_n to a sequence of abstract semantic label $\mathbf{s}_n = \{s_1 = v_1, s_2 = v_2, \cdots\}$. In our setting, $X_n = \{x_1, x_2, \cdots\}$ is the text input represented as a sequence of words, and $\mathbf{s}_n = \{s_1 = v_1, s_2 = v_2, \cdots\}$ is a sequence of labels, where the slot s_1 and the slot value v_1 are the labels of word x_t.

1. Input: a user utterance as a sequence of words, $X_n = \{x_1, x_2, \cdots\}$.

2. Output: a dialogue semantic label for each word, $\mathbf{s}_n = \{s_1 = v_1, s_2 = v_2, \cdots\}$.

3. Example: For an input sentence “who played zeus in the 2010 action movie Titans”, the expected output is a semantic tag for each word in the input sentence like “who:{} played:{}

zeus:{character=zeus} in:{} the:{} 2010:{year=2010} action:{genre=action} movie:{type=movie} Titans:{name=Titans}".

3.2.2 Single Domain Spoken Language Understanding

In the following section, we will introduce several slot-filling models including the Conditional Random Field (CRF), the Convolution Neural Network (CNN), the Recursive Neural Network (RNN) and the Long-Short Term Memory (LSTM).

CRF

Wang *et al.* [157] and Raymond *et al.* [112] use CRF [70] for slot-filling problem. The problem can be modelled by

$$\mathbf{s} = \arg\max_{\mathbf{s}'} P(\mathbf{s}'|X),$$

where $X = \{x_1, x_2, \cdots\}$ is an input word sequence and $\mathbf{s} = \{s_1, s_2, \cdots\}$ is the associated class label sequence. The probability of $\mathbf{s}$ is defined by

$$P(\mathbf{s}|X) = \prod_t P(s_t|x_t, s_{t-1}),$$

where $P(s_t|x_t, s_{t-1})$ is defined as

$$P(s_t|x_t, s_{t-1}) = \frac{1}{Z(x_t, s_{t-1})} \exp(\mathbf{w}_{s_t}^T f(s_{t-1}, s_t, x_t) + b_s),$$

$f(s_{t-1}, s_t, x_t)$ is the feature vector for time step t, and the feature vector includes n-gram lexical features in the sliding window and the state transition features, etc. $\mathbf{w}_{s_t}$ is the associated weight vector, and $Z(x_t, s_{t-1}) = \sum_{s'} \exp(\mathbf{w}_{s'}^T f(s_{t-1}, s', x_t) + b_{s'})$ is the softmax normalization term.

CNN-CRF

Xu *et al.* [179] propose a joint CNN-CRF model for the spoken language understanding problem. The problem can be modelled by

$$\mathbf{s} = \arg\max_{\mathbf{s}'} P(\mathbf{s}'|X).$$

The probability of $\mathbf{s}$ is defined by

$$P(\mathbf{s}|X) = \prod_t P(s_t|x_t, s_{t-1}),$$

where $P(s_t|x_t, s_{t-1})$ is defined as

$$P(s_t|x_t, s_{t-1}) = \frac{1}{Z(x_t, s_{t-1})} \exp(\text{trans}(s_{t-1}, s_t) + \mathbf{w}_{s_t}^T f(x_t)),$$

trans(s_{t-1}, s_t) is the transition probability from s_{t-1} to s_t, $f(x_t)$ is the feature vector extracted from the CNN window centered at x_t, and $Z(x_t, s_{t-1}) = \sum_{s'} \exp(\text{trans}(s_{t-1}, s') + \mathbf{w}_{s'}^T f(x_t))$ is the softmax normalization term.

Celikyilmaz *et al.* [21] propose to add additional word embeddings as additional features to $f(x_t)$ in slot-filling task, along with the CRF or the CNN-CRF model. The experimental results show that CRF-CNN model is better than CRF alone, and adding additional word embeddings can further improve the results.

RNN

Yao *et al.* [181], Mesnil *et al.* [92, 91] and Liu *et al.* [80] propose to use RNN [44] on the slot-filling problem. The problem is modelled by

$$\mathbf{s} = \arg\max_{\mathbf{s}'} P(\mathbf{s}'|X).$$

The probability of $\mathbf{s}$ is defined by

$$P(\mathbf{s}|X) = \prod_t P(s_t|x_t, x_{<t}),$$

where $P(s_t|x_t, x_{<t})$ is defined as

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}_h),$$

$$P(s_t|x_t, x_{<t}) = \frac{1}{Z(\mathbf{h}_t)} \sigma(\mathbf{w}_{s_t}^T \mathbf{h}_t + b_{s_t}),$$

$\mathbf{x}_t$ is feature vector calculated in the window centered at word x_t, which includes the word embedding vector $\mathbf{x}_t$ on x_t and the embedding of nearby words. $\Theta = \{\mathbf{W}_h, \mathbf{U}, \mathbf{b}_h, \mathbf{W}_s, b_s\}$ are parameters to learn. $Z(\mathbf{h}_t) = \sum_{s'} \exp(\mathbf{w}_{s'}^T \mathbf{h}_t + b_{s'})$ is the softmax normalization term. Unlike CRF which can model only a fix context length, the RNN can model the dialogue context of arbitrary length. As a result, the RNN requires more word level labelled training data.

LSTM

Yao *et al.* [180] propose to use LSTM [53] for slot-filling in the SLU. The LSTM can be formulated by

$$\mathbf{s} = \arg\max_{\mathbf{s}'} P(\mathbf{s}'|X).$$

The $P(\mathbf{s}|X)$ is defined as

$$P(\mathbf{s}|X) = \prod_t P(s_t|x_t, x_{<t}).$$

The four gates in the LSTM are calculated by

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i),$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f),$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c),$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_{t-1} + \mathbf{b}_f)$$

and the hidden state $\mathbf{h}_t$ is given by

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

where $\odot$ is the element-wise multiplication, and $\mathbf{x}_t$ is the feature vector calculated in the window centered at word x_t, which includes the word embedding vector on x_t and embedding vectors of the nearby words. Based on $\mathbf{h}_t$, we can calculate $\mathbf{p}_t$, the intermediate predicted output at current time step by

$$\mathbf{p}_t = \mathbf{W}_{hp}\mathbf{h}_t.$$

Then the final feature vector $\mathbf{q}_t$ is calculated based on M most recent intermediate output $\mathbf{p}_t$ by

$$\mathbf{q}_t = \sum_{i=0}^{M} \mathbf{W}_{pi}\mathbf{p}_{t-i} + \mathbf{b}_q.$$

The prediction probability of the output label s_t is predicted by

$$p(s_t|x_t, x_{<t}) = \frac{1}{Z(\mathbf{q}_t)}\sigma(\mathbf{w}_{s_t}^T\mathbf{q}_t + b_{s_t}),$$

where $Z(\mathbf{q}_t) = \sum_{s'} \exp(\mathbf{w}_{s'}^T\mathbf{q}_t + b_{s'})$ is the softmax normalization term. Using the LSTM can alleviate the gradient exploding and vanishing problem, and it can model long dependency better. However, there are more parameters in the LSTM than in the RNN, which requires more training data.

Evaluation for Single Domain SLU

To evaluate an SLU, the algorithm predicts each utterance in the test set and the predicted slot-values will be compared with ground-truth slot-values labelled by human. Popular evaluation metrics include accuracy for speech-act/intention classification, and F1 score for slot-value classification.

Public datasets include the Air Travel Information System (ATIS) [23], the Tourist Information (MEDIA) [16],the DARPA Communicator Travel Data [153], etc. Public challenges include the DARPA ATIS challenge and the DARPA COMMUNICATOR challenge. All results are compared on the public available dataset Air Travel Information System (ATIS).

We only compare models for the slot-filling task on the Air Travel Information System (ATIS), the results are shown in table 3.1.

Table 3.1: SlotValue Classification F-1 metrics on ATIS dataset

	CRF	CRF-CNN	RNN	LSTM
F1 ATIS [83]				
F1 ATIS [149]	0.8373			
F1 ATIS [179]	0.9100	0.9435	0.9411	
F1 ATIS [181]	0.9109		0.9411	
F1 ATIS [91]	0.9294		0.9498	
F1 ATIS [180]	0.9294	0.9435	0.9411	0.9492

CRF uses the label of previous word when predicting the label of the current word, this is better than doing simple classification on each word position because CRF has considered the label transition probability. However, we still need to define the word features by hand. And CRF considers only fixed window size, which is not flexible. CNN-CRF can learn local features for each word position by an end-to-end training. In this way we can learn better feature representations from data, and thus we can reduce the human efforts required. However, the framework is still CRF, which can only model a fixed number of previous words. RNN can model a very long dependency in a sentence. In RNN, all information about previous words is stored in the hidden variable, and theoretically it can consider arbitrary long dependency. However, the training of RNN suffers the exploding or vanishing gradient problem, which makes RNN hard to train. LSTM can choose to remember or forget the information stored in its memory based on the input, and it can solve the exploding or vanishing gradient problem appeared in RNN. As a result, it has better model capability than an RNN and it is more easy to train.

3.2.3 Transfer Learning for Spoken Language Understanding

In this section we summarize the transfer learning methods used for spoken language understanding. There are 3 methods to transfer knowledge for speech-act/intention classification problem in SLU, they are

1. Model Adaptation,
2. Instance-based Transfer,
3. Parameter Transfer.

Model Adaptation

Tur *et al.* [147] propose to use model adaptation and boosting for speech-act/intent classification transfer learning. The source domain and the target domain share the same set of speech-acts/intention labels, but have different distributions. The authors propose to regularize the target domain model with the KL divergence between the source model and the target model. The loss function in the target domain is defined as

$$L(\mathbf{w}) = \sum_n \sum_{\mathrm{a}'} (\ln(1+\exp(-A_n[\mathrm{a}']f(X_n,\mathrm{a}';\mathbf{w})))) + \eta \mathbf{KL}(P(A_n[\mathrm{a}']=1|X_n)||\sigma(f(X_n,\mathrm{a}';\mathbf{w}))),$$

$$\mathbf{KL}(p||q) = p\ln\frac{p}{q} + (1-p)\ln\frac{1-p}{1-q},$$

where the first term is the target domain model loss function and the second term is the KL divergence between the source model and the target model. n is the index of an instance and a' is an element in the label set. $P(A_n[\mathrm{a}']=1|\mathbf{X}_n)$ is the probability that $\mathbf{X}_n$ belongs to label a' in the source domain model, and $\sigma(f(\mathbf{X}_n),\mathrm{a}')$ is the probability that X_n belongs to label a' in the target domain model.

Instance-based Transfer

Tur *et al.* [148] propose to transfer instances on speech-act/intention classification problem. There are a number of similar intention classes in the source and target domains, but no mapping is available. The authors propose to use instances with similar label in the source domain to help the intention classification in the target domain. We denote the target domain dataset as $\{X, a\}$ and the source domain dataset as $\{X^s, a^s\}$. We denote the classifier in the source domain by

$$p(a^s|X^s) = f_A^s(X^s).$$

The classifier in the target domain is denoted by

$$p(a|X) = f_A(X).$$

The authors apply the target domain classifier on each instance X^s in the source domain. If the predicted probability of the source instance X^s belonging to a target class is above a threshold, i.e, $f_{a_i}(X^s) > \rho$, then X^s is transferred to the target domain as an instance of target domain class a_i. After some instances are transferred, the classifier in the target domain is retrained with the transferred source domain instances and the target domain instances.

Parameter Transfer

Yazdani *et al.* [182] propose to use parameter sharing between similar label classifiers, so the similar classifiers can have similar hyperplane. The classification model for each speech-act a is a linear classifier, and the parameter for the classifier of label $a_j(s_k = v_m)$ is a weight vector $\mathbf{w}_{a_j(s_k=v_m)}$. For each label $a_j(s_k = v_m)$, the multi-label classification is done by logistic regression as

$$y = \sigma(\mathbf{w}^T_{a_j(s_k=v_m)}\phi(x_i)).$$

The authors assume that the weight vector $\mathbf{w}_{a_j(s_k=v_m)}$ can be modelled by a two layer multi-layer perceptron classifier given the embedding vector of the label words as input, where the label names are treat as words. The weight parameters for label $a_j(s_k = v_m)$ can be generated by using the embeddings of word a_j, word s_k and word v_m, as

$$\mathbf{w}_{a_j(s_k=v_m)} = \sigma([\phi(a_j), \phi(s_k), \phi(v_m)]\mathbf{W}_{ih})\mathbf{W}_{ho}$$

where $\phi(x_i)$ is the word embedding for word x_i, σ is the activation function, $\mathbf{W}_{ih}$ is an $3d * h$ matrix and $\mathbf{W}_{ho}$ is a $d * d$ matrix. The parameter $\mathbf{W}_{ih}$ and $\mathbf{W}_{ho}$ are shared for all labels, so transfer learning is possible.

Jeong *et al.* [57] propose to divide the model parameters into domain-dependent parameters and domain-independent parameters, where the domain-independent parameters are shared across the source domain and the target domain to transfer knowledge. A CRF is used as the base model for slot-filling problem, and there are a number of shared labels cross domains. The probability of the slot set is given by

$$\mathbf{s} = \arg\max_{\mathbf{s}'} P(\mathbf{s}'|X)$$

where $X = \{x_1, x_2, \cdots\}$ is the input word sequence and $\mathbf{s} = \{s_1, s_2, \cdots\}$ is the associated class label sequence. The probability of the slot set can be factorized by

$$P(\mathbf{s}|X) = \prod_t P(s_t|x_t, s_{t-1}),$$

$$P(s_t|x_t, s_{t-1}) = \frac{1}{Z(x_t, s_{t-1})} \exp(\phi_d(s_{t-1}, s_t, x_t) + \phi_{\text{independent}}(s_{t-1}, s_t, x_t)),$$

where $\phi_d(s_{t-1}, s_t, x_t)$ is the domain-dependent model and $\phi_{\text{independent}}(s_{t-1}, s_t, x_t))$ is the domain-independent model. The model considered features such as n-gram lexical features in the sliding window, and state transition probability, etc. $Z(x_t, s_{t-1}) = \sum_{s'} \exp(\phi_d(s_{t-1}, s', x_t) + \phi_{\text{independent}}(s_{t-1}, s', x_t))$ is the softmax normalization term. $\phi_{\text{independent}}(s_{t-1}, s', x_t)$ is the domain-independent model shared across domains for knowledge transfer.

Model Comparison for Transfer Learning in SLU

Model adaptation can be used to adapt an existing model from a source domain to improve its performance in a target domain. Given the target domain data, the pre-trained model is fine-tuned on a few new instances in the target domain. However, the source domain and the target domain have to use the same kind of model. Instance transfer can work with any classifiers without modifying the classifier structure, and it is easy to train. However, the source domain and the target model need to be trained for multiple times before converging, which is time consuming. Parameter transfer can transfer a part of the common model parameters from a source domain to a target domain, thus it can reduce the negative transfer. However, the model parameters have to be partitioned into shared parameters and domain-independent parameters, as a result, many classifiers could not be used.

3.3 Module 2: Dialogue State Tracking (DST)

The dialogue state tracking module tracks the dialogue state according to the system action, the user utterance, and the previous dialogue state.

3.3.1 Problem Definition

Dialogue state tracking aims to keep track of the dialogue state $\tilde{H}_n$, given the previous dialogue state $\tilde{H}_{n-1}$, the abstract system action $\tilde{Y}_{n-1}$ and the abstract user utterance $\tilde{X}_n$.

1. Input: The crevious dialogue state $\tilde{H}_{n-1}$, the abstract system action $\tilde{Y}_{n-1}$ and the abstract user utterance $\tilde{X}_n$.
2. Output: The current dialogue state $\tilde{H}_n$.

3.3.2 State Representation

In a task oriented dialogue system, the dialogue state is predefined by human based on the domain slots and their slot-values. The number of possible states is exponential to the number of slots. Maintaining a probability distribution of all possible states requires a large amount of resource, here we introduce several methods to simplify the problem.

Typically, there are several kinds of information in a state, denoted by $\tilde{H}_n = \{g_n, \tilde{X}_n, H_n\}$, where $H_n = \{X_0, Y_0, \cdots, X_{n-1}, Y_{n-1}, X_n\}$ is the dialogue history.

1. User goal g_n. The user goal is a set of slot-values in each slot, representing the entity this user is looking for. For example, the user is looking for a restaurant in the north part of the city that can serve Chinese food. User goal tracking is the core tracking problem.
2. User last intent $\tilde{X}_n$. The user last intent is the direct intent of the last utterance of the user. For example, if the user is asking, then the system should provide information. If the user is confirming, then the system should indicate whether the confirmation is done.
3. Dialogue history H_n. The dialogue history contains the previous dialogues, so that the system can recognize what question has been asked and what information is already collected.

As we can see, the most challenging task is the tracking of the user goal g_n, because the goal space can be very large. Assume we have 5 slots, and there are 10 possible slot-values in each slot. The

total number of possible goals will be 10^5, and this is only for a small domain with 5 slots. Directly dealing with such a large state space will make the problem very complicated.

There are two popular ways to simplify the state representation and the state tracking.

1. Hidden information state model. The idea is to record the distribution of only top possible states.

2. Factorized model. The idea is to track each slot independently of other slots, so that we can record the distribution of all values in each slot. Simple dependencies between variables are represented as Bayesian networks. But this model cannot deal with complicated dependencies.

Hidden Information State Model

Young *et al.* [185] propose the hidden information state approach. To deal with the exponential large user goal space, the authors propose to use state grouping and state splitting to reduce the tracking complexity.

In order to track a dialogue state, at each round, based on the previous state, the system action, the user utterance, the state space can be split into 2 partitions, where each partition can be split further in the coming rounds. Dialogue states $\tilde{H}$ within each partition p_n are treated indiscriminately, meaning that all states $\tilde{H}$ within partition p_n have equal probability. Since the number of partitions grows exponentially with number of dialogue rounds, only the top N partitions are tracked.

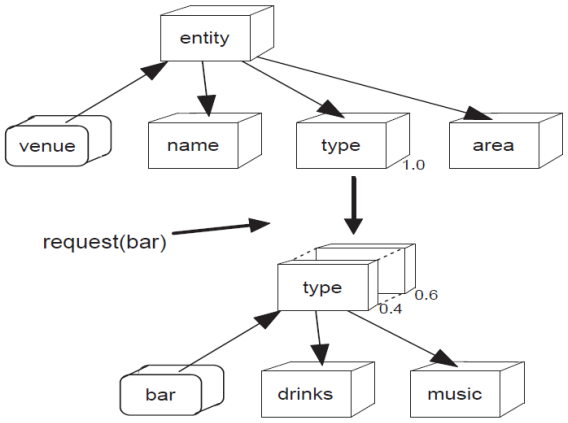


Figure 3.2: Partitioning of dialogue states in HIS [185]

At each round, the belief update formulation can be modelled by

$$\begin{aligned} b_{n+1}(p_{n+1}, \tilde{X}_{n+1}, H_{n+1}) = \quad & \eta P(X_{n+1}|\tilde{X}_{n+1}) P(\tilde{X}_{n+1}|p_{n+1}, \tilde{Y}_n) \\ & \textstyle\sum_{H_n} P(H_{n+1}|p_{n+1}, \tilde{X}_{n+1}, H_n, \tilde{Y}_n) \\ & P(p_{n+1}|p_n) b(H_n), \end{aligned}$$

where p_n is a partition and the term $P(p_{n+1}|p_n)$ represents the probability that a partition p_n being split into two sub partition, $p_n \rightarrow \{p_{n+1}, p_n - p_{n+1}\}$. An example is shown in Figure 3.2. $\tilde{X}_n$ is the user action, $\tilde{Y}_n$ is the system action, H_n is the dialogue history and X_n is the observed user utterance. $P(X_{n+1}|\tilde{X}_{n+1})$ is the observation model, $P(\tilde{X}_{n+1}|p_{n+1}, \tilde{Y}_n)$ is the user action model, $P(H_{n+1}|p_{n+1}, \tilde{X}_{n+1}, H_n, \tilde{Y}_n)$ is the dialogue state transition model and $P(p_{n+1}|p_n)b(H_n)$ represents belief refinement.

The dialogue state $\tilde{H}_n$ can be randomly sampled from the probability distribution defined as

$$\tilde{H}_n \sim b_n(p_n, \tilde{X}_n, H_n)$$

because all states within partition p_n has equal probability.

One problem is that the number of partitions are exponential to the length of the dialogue. In practice, pruning is required [175]. However, a slot can be prune if they are distributed in many partition. To solve this problem, Gavsic *et al.* [40] propose to track the user goal with respect to each slot.

HIS model can help us track the dialogue state efficiently, and arbitrary dependency relations between different states can be incorporated. However, only the top-N states can be tracked.

Bayesian Update of Dialogue State

Thomson *et al.* [143] propose the Bayesian Update of Dialogue State model (BUDS), in which they factorize the user goal in the dialogue state into a set of independent slots. The assumption is that the tracking probability of slot-values in different slots are independent (or have simple dependency) of each other. The dependencies of slot-values can be specified by a dynamics Bayesian network, as shown in Figure 3.3. When the Bayesian network is compiled, belief propagation algorithm can be used to update the belief. In case two concepts are conditionally independent, the algorithm can give exact updates on the marginal distribution for each concept. Only limited dependencies can be incorporated due to speed problem.

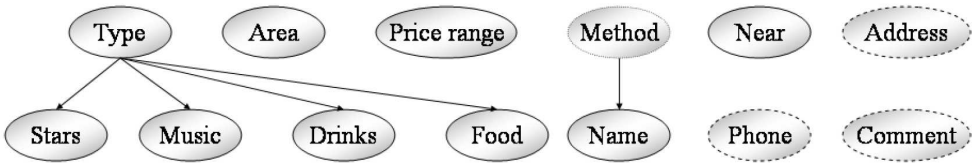


Figure 3.3: Simple dependencies in BUDS [143]

In BUDS model, the whole probability distribution of all slot-values in all slots are tracked. However, this model can only deal with very simple dependencies. In fact, the BUDS model is more widely used because it is easy to implement.

3.3.3 Single Domain Dialogue State Tracking

In this section we introduce the algorithms used to track dialogue states. The algorithms can be categorized into two categories.

1. Static Classifiers. Dialogue state tracking is modelled as a classification problem on the dialogue history. Firstly, some hand-designed features are extracted from the history, then various classifiers can be used to predict the dialogue states.

2. Sequential Classifiers. Dialogue state tracking is modelled as a sequential labelling problem. In this category, the label transition probability is taken into consideration as well as the features of user utterances.

Static Classifier

Dialogue state tracking is modelled as a classification problem on the dialogue history. At first some hand-designed features are extracted from the dialogue history, then various classifiers can be used to predict the dialogue states. This can be modelled by

$$p(\tilde{H}_n|\{\tilde{Y}_0, \cdots, \tilde{Y}_{n-1}\}\{\tilde{X}_0, \cdots, \tilde{X}_n\}),$$

where $\tilde{Y}_n$ is the abstract system action taken in round n, and $\tilde{X}_n$ is the abstract user action prediction from X_n in round n.

Here we list some features [93] for dialogue state classification:

1. The size of the SLU M-best list in the last turn;

2. The entropy of the SLU M-best list in the last turn;

3. The posterior score from the ASR in the last turn;

4. The probability of the top slot-value in each slot;

5. The number of times each slot has appeared in the history;

6. The number of possible negations and confirmations of each slot in the history;

7. The probability a slot-value being confused with another slot-value in each slot.

For classifiers, many machine learning discriminative classifiers can be used, including linear classifiers [15, 93], neural networks [132, 49, 113] and ranking models [176].

Static classifiers are easy to deploy because we can directly use off-the-shelf classification models. However, static classifiers do not consider the transition probability between dialogue states.

Sequential Classifier

In this section, we survey the sequential classifiers used in dialogue state tracking. Popular sequential classifiers include CRF, RNN and LSTM. The probability of the current state $\tilde{H}_n$ is defined by

$$p(\tilde{H}_n|\tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n)$$

where $\tilde{H}_n$ is the dialogue state at round n, $\tilde{Y}_n$ is the system action at round n, and $\tilde{X}_n$ is the user action predicted by SLU in round n.

Lee *et al.* [74] and Kim *et al.* [59] use CRF [70] to classify dialogue states by

$$p(\tilde{H}_n|\tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n) = \frac{1}{Z(\tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n)} \exp(\mathbf{w}_{\tilde{H}_n}^T f(\tilde{H}_n, \tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n) + b_{\tilde{H}_n})$$

$f(\tilde{H}_n, \tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n)$ is the feature vector for round n (such as n-gram lexical features in the sliding window, state transition features, and other features), $\mathbf{w}_{\tilde{H}_n}$ is the associated weight vector, $Z(\tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n) = \sum_{\tilde{H}'} \exp(\mathbf{w}_{\tilde{H}'}^T f(\tilde{H}', \tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n) + b_{\tilde{H}'})$ is the softmax normalization term.

Henderson *et al.* [50, 51] use RNN [44] to classify dialogue states by

$$\mathbf{h}_n = \sigma(\mathbf{W}_h \mathbf{h}_{n-1} + \mathbf{U} f(\tilde{Y}_{n-1}, \tilde{X}_n) + \mathbf{b}_h),$$

$$p(\tilde{H}_n) = \frac{1}{Z(\mathbf{h}_n)} \sigma(\mathbf{w}_{\tilde{H}_n}^T \mathbf{h}_n + b_{\tilde{H}_n}),$$

where $f(\tilde{Y}_{n-1}, \tilde{X}_n)$ is the feature vector calculated in round n. $\Theta = \{\mathbf{W}_h, \mathbf{U}, \mathbf{b}_h, \mathbf{W}_s, b_s\}$ are parameters to learn. $Z(\mathbf{h}_n) = \sum_{\tilde{H}'} \exp(\mathbf{w}_{\tilde{H}'}^T \mathbf{h}_n + b_{\tilde{H}'})$ is the softmax normalization term. CRF can model only a fix context length, while RNN can model context of arbitrary length. As a result, RNN requires more labelled training data to train.

Zilka *et al.* [187] use LSTM [53] to track dialogue states. The hidden state $\mathbf{h}_n$ is calculated by

$$\begin{bmatrix} \mathbf{i}_n \\ \mathbf{o}_n \\ \mathbf{f}_n \\ \hat{\mathbf{c}}_n \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \mathbf{h}_{n-1} \\ f(\tilde{Y}_{n-1}, \tilde{X}_n) \end{bmatrix},$$

$$\mathbf{c}_n = \mathbf{f}_n \odot \mathbf{c}_{n-1} + \mathbf{i}_n \odot \hat{\mathbf{c}}_n,$$

$$\mathbf{h}_n = \mathbf{o}_n \odot \tanh(\mathbf{c}_n),$$

where $f(\tilde{Y}_{n-1}, \tilde{X}_n)$ is the feature vector calculated in round n, $\mathbf{W} \in \mathbb{R}^{4d*2d}$, d is the dimension of the extracted feature vector. $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t$ are all vectors in $\mathbb{R}^d$. The probability of dialogue state $\tilde{H}_n$ is calculated by

$$p(\tilde{H}_n) = \frac{1}{Z(\mathbf{h}_n)} \sigma(\mathbf{w}_{\tilde{H}_n}^T \mathbf{h}_n + b_{\tilde{H}_n})$$

where $Z(\mathbf{h}_n) = \sum_{\tilde{H}'} \exp(\mathbf{w}_{\tilde{H}'}^T \mathbf{h}_n + b_{\tilde{H}'})$ is the softmax normalization term. Using LSTM can avoid the gradient vanishing problem and the gradient exploding problem in training, and it can model long term dependency better. However, the number of parameters in LSTM is larger, which requires more training data.

Evaluation for Single Domain DST

In this section we evaluate and compare the performance of the dialogue state tracking methods.

To evaluate dialogue state tracking algorithms, the predicted dialogue state will be compared with human labelled dialogue state. Two metrics widely used in evaluating dialogue state tracking are accuracy and the L2-norm. The accuracy measures the quality of the top hypothesis, that is whether the top predicted dialogue state is correct. The L2 measure the distance between predicted dialogue state probabilistic distribution with the ground-truth dialogue state probability distribution.

Existing datasets are from a series of challenges called Dialogue State Tracking Challenges. There are DSTC 1 (bus information), DSTC 2 (restaurant information) [47] and DSTC 3 (coffee shop, pub and restaurant) [48]. The DSTC 2 dataset is widely used, many results are reported in Table 3.2.

Table 3.2: Dialogue State Tracking Evaluation of Joint Goals

		Goal		Method		Request	
Method	Dataset	Accuracy	L2	Accuracy	L2	Accuracy	L2
Linear CRF	DSTC2	0.601	0.648	0.904	0.155	0.960	0.073
RNN	DSTC2	0.768	0.346	0.940	0.095	0.978	0.035
LSTM	DSTC2	0.72	0.64	0.93	0.14	0.97	0.06
Ranking	DSTC2	0.78	0.35	0.95	0.08	0.98	0.04

Static classifiers are easy to deploy because we can directly use off-the-shelf classification models. However, static classifiers do not consider the transition probability between dialogue states.

Sequential classifiers can model the state transition probability and the long-term dependency. However, sequential classifiers have more parameters and require more data.

3.3.4 Transfer Learning for Dialogue State Tracking

In this section, we review the multi-domain transfer learning papers for the dialogue state tracking problem. These algorithms can be categorized into feature-based [171, 113] and model-based [99].

1. Feature-based multi-domain dialogue state tracking aims to build general domain-independent features, so that the trained models can be reused in multi-domain setting.

2. Model-based multi-domain dialogue state tracking adapts an general domain-independent tracking model with the domain-dependent data to build a dialogue state tracker for multiple domains.

Feature-based Multi-domain DST

Feature-based multi-domain dialogue state tracking aims to build general domain-independent features, so that the trained models can be used in multi-domain setting.

Williams *et al.* [171] propose to use the shared synthetic features. Each instance in domain i has two set of features, one set of parameters is the domain-dependent parameter, the other set is the general domain-independent parameters. The general domain-independent parameters are shared across all domains, so the dialogue knowledge can be transferred across domains. For example, assuming we have 3 domains, the feature vector of an instance in the first domain is like $< \mathbf{f}_1, \mathbf{0}, \mathbf{0}, \mathbf{f}_1 >$, the feature vector in the second domain looks like $< \mathbf{0}, \mathbf{f}_2, \mathbf{0}, \mathbf{f}_2 >$, and the feature vector in the third domain looks like $< \mathbf{0}, \mathbf{0}, \mathbf{f}_3, \mathbf{f}_3 >$. $\mathbf{f}_i$ is the feature vector, and $\mathbf{0}$ is the zero vector

with the same size of the feature vector. We can see, knowledge is shared through the last part of the learned linear classifiers.

Ren *et al.* [113] propose to share the dialogue state tracking model across different domains by using a domain-dependent feature set. For each utterance, a joint feature matrix is extracted. In the joint feature matrix of the utterance, different features are extracted for each domain. The extracted feature for each target domain is specially designed, so that for different target domains, we can still use the same tracking model.

Model-based Multi-domain DST

Model-based multi-domain dialogue state tracking adapts an general domain-independent tracking model with the domain-dependent datasets to build a dialogue state tracker for multiple domains.

Mrkvsic *et al.* [99] propose to first train a generalize dialogue state tracker RNN for all domains, then initialize the domain-dependent tracker RNN with this general RNN. The key idea is to use the delexicalised features before processing, for example, "want available internet" would become "want tag-slot-value tag-slot-name". The delexicalised features not only allow knowledge transfer between domains, but also allow knowledge transfer between fine granularity slots. However, due to different data distributions in different slots, the model adaptation is still required to get the general model to work for each slot.

Model comparison for Transfer Learning in DST

Feature-based multi-domain DST methods achieve knowledge transfer by designing domain-independent features and sharing the same model, which requires careful feature design and lots of human efforts. Model-based multi-domain DST methods allow knowledge transfer between fine granularity slots, but it can only work on the delexicalised data.

3.4 Module 3: Dialogue Policy (DPL)

A popular model for the dialogue management module is Markov Decision Process (MDP) [14, 76, 152, 125]. The dialogue manager is modelled as a system that tries to achieve a goal through a series of interactions with the user. The information about the current dialogue situation is modelled with a state, and the dialogue manager can choose the optimal system action, then the next dialogue state depends only on the current state and the action taken by the system, which follows Markov assumption. In some situations, due to possible typos and speech recognition errors, the current state of the system can not be fully determined. The Partially Observable Markov Decision Process (POMDP) [184] is used to model the dialogue policy. At each step, in stead of tracking the ground-truth dialogue state, the POMDP system keep track of a probability distribution on all possible dialogue states. The probability distribution on all possible dialogue states is called a belief state. The belief state is assumed to follow the Markov assumption, which means the next belief state is dependent only on the current belief state and the action taken. The POMDP policy decides the best system action based on the current belief state instead of the true state. Actually, POMDP can be modelled by an MDP in the belief state space.

Based on whether multiple domains are considered, dialogue systems can be categorized into single-domain dialogue systems and multi-domain dialogue systems. Single domain means the training and testing data are in the same domain, and no domain difference is considered. The goal of single domain dialogue management is to learn the optimal dialogue policy in this domain. Multi-domain dialogue systems aim to use the knowledge in a source domain to help the policy learning in the target domain, with the goal of improving the dialogue performance in the target domain.

We organize all policy learning methods into 2 categories:

1. Single-domain Dialogue Policy Learning.
2. Transfer Learning for Multi-domain Dialogue Policy Learning.

3.4.1 Problem Definition

In a modular dialogue system, we formulate the dialogue management as a MDP without loss of generality, because a POMDP policy could be represented by a MDP on the belief state. The MDP is defined as $\{H, Y, P, \mathcal{R}, \gamma\}$. H are the dialogue states. Y are the replies of the agent, P is the state transition probability function, $\mathcal{R}$ is the reward function, $\gamma \in [0, 1]$ is the discounted factor.

At time step n, $\tilde{H}_n$ is the dialogue state, $\tilde{Y}_n$ is the agent reply and r_n is the reward. In time step n, we assume the SLU and the DST have provided the current state $\tilde{H}_n$, so we can observe $\tilde{H}_n, \tilde{Y}_n$ and r_n. The cumulative return is defined as

$$G_n = \sum_{k=0}^{\infty} \gamma^k r_{n+k}$$

and our goal is to find an optimal policy that can maximize the cumulative return.

The dialogue policy decides the best system action $\tilde{Y}_n$ under the current state $\tilde{H}_n$.

1. Input: a dialogue state $\tilde{H}_n$.
2. Output: a system action $\tilde{Y}_n$.
3. Objective function: maximize the expected future cumulative reward.
4. dataset: $< \tilde{H}_n, \tilde{Y}_n, r_n, \tilde{H}_{n+1} >$ for $n = 1, 2, \cdots, N$. For transfer learning problem, there are additional input source domain datasets $< \tilde{H}^s_n, \tilde{Y}^s_n, r^s_n, \tilde{H}^s_{n+1} >$ for $n = 1, 2, \cdots, N^s$.

3.4.2 Single-Domain Dialogue Policy Learning

In this section, we survey the policy learning in a single domain modular dialogue system. Single domain means the dialogue policy is trained and tested in one single domain, and we focus on the dialogue performance in this single domain only.

In dialogue system, a dialogue model is rarely assumed because of the complexity of human dialogue. Of course we can first estimate the transition probability and reward function and then find the optimal policy with model based methods, but in that case the error in the model will propagate to the policy function and might cause bigger error. In this survey, we primarily focus on model-free reinforcement learning methods.

Off-policy reinforcement learning aims to learn an optimal policy when the behaviour policy is different from the learning policy. The learning policy has to learn by "looking over the shoulder of another policy", and it can learn from a static dataset as well as learning in on-line setting. On-policy learning reinforcement learning learns by interacting with the environment. On-policy learning is more sample-efficient because it can control the next action, but it needs to balance between exploration and exploitation. On-policy learning reinforcement learning requires an interactive environment such as a user simulator, and it has to learn in on-line setting.

There are 3 popular RL-frameworks [135] in task-oriented dialogue systems. Based on the formulation of the policies and the value functions, RL-algorithms can be categorized into the following categories.

1. Value-based RL. Policy is represented by action-value function $Q(\tilde{H}, \tilde{Y})$ (Q-function). The best action is selected by $\tilde{Y} = \arg\max_{\tilde{Y}} Q(\tilde{H}, \tilde{Y})$. To learn the Q-function of the optimal policy, we iterative improve the policy defined by Q-function in a greedy manner at each time step.

2. Policy-based RL. Policy is directly represented by a policy function $\pi(\tilde{H}, \tilde{Y}) \sim p(\tilde{Y}|\tilde{H})$. The best action is defined by the policy function $\tilde{Y} = \arg\max_{\tilde{Y}'} \pi(\tilde{H}, \tilde{Y}')$. To learn the policy function, we can use policy iteration or policy gradient, and the ideas is to maximize the probability of the "good" episodes.

3. Actor critic RL. A Q-function $Q(\tilde{H}, \tilde{Y})$ and a policy function $\pi(\tilde{H}, \tilde{Y}) \sim p(\tilde{Y}|\tilde{H})$ are defined simultaneously. The best action can be selected by $\tilde{Y} = \arg\max_{\tilde{Y}} Q(\tilde{H}, \tilde{Y})$ or $\tilde{Y} = \arg\max_{\tilde{Y}'} \pi(\tilde{H}, \tilde{Y}')$. To learn the policy function, actor-critic RL uses the estimated action-value function $Q(\tilde{H}, \tilde{Y})$ rather than the directly observed noisy reward to update the policy function $\pi(\tilde{Y}|\tilde{H})$, in order to improve the stability.

Value-based Reinforcement Learning

The action value function is defined as the expected return starting from state $\tilde{H}$, taking action $\tilde{Y}$, and then following policy π by

$$Q(\tilde{H}_n, \tilde{Y}_n) = \mathbb{E}(\sum_{k=0}^{\infty} \gamma^k r_{n+k} | \tilde{H}_n, \tilde{Y}_n).$$

In the batch training setting which does not require exploration, policy can be specified by

$$\tilde{Y} = \arg\max_{\tilde{Y}'} Q(\tilde{H}, \tilde{Y}').$$

In the on-line on-policy training setting where exploration and exploitation are both needed, the policy can be specified by choosing a best action with probability $1 - \epsilon$ and choosing a random action with probability ϵ

$$\tilde{Y} = \begin{cases} \arg\max_{\tilde{Y}'} Q(\tilde{H}, \tilde{Y}') & \text{with probability } 1 - \epsilon \\ \tilde{Y}_{\text{rand}} & \text{with probability } \epsilon \end{cases}$$

There are many methods to learn the Q-function. In the on-line on-policy training scenario, when the environment model is stochastic, number of action is discrete, Sarsa [25] algorithm can be used to improve the policy by

$$\tilde{Y}_{n+1} = \max_{\tilde{Y}'} Q(\tilde{H}_{n+1}, \tilde{Y}')$$

$$Q(\tilde{H}_n, \tilde{Y}_n) = Q(\tilde{H}_n, \tilde{Y}_n) + \alpha(r_n + \lambda Q(\tilde{H}_{n+1}, \tilde{Y}_{n+1}) - Q(\tilde{H}_n, \tilde{Y}_n)).$$

In the off-line batch training setting, Q-learning [161] can be used to update the Q-function by

$$\tilde{Y}_{n+1} = \pi_{\text{action}}(\tilde{H}_{n+1}).$$

$$Q(\tilde{H}_n, \tilde{Y}_n) = Q(\tilde{H}_n, \tilde{Y}_n) + \alpha(r_n + \max_{\tilde{Y}'} \lambda Q(\tilde{H}_{n+1}, \tilde{Y}') - Q(\tilde{H}_n, \tilde{Y}_n)).$$

Least square value iteration [78] can also be used.

Here we introduce 4 kinds of model used to approximate $Q(\tilde{H}, \tilde{Y})$.

1. Grid-based models. All states $\tilde{H}_n$ are quantized to $N_{\tilde{H}}$ discrete states and all actions are quantized to $N_{\tilde{Y}}$ discrete actions. The Q-function is approximated by a look-up table.

2. Linear regression based models. A feature vector is extracted by $\phi(\tilde{H}, \tilde{Y})$, the Q-function is approximated by $\sigma(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w})$.

3. Gaussian process based models. The Q-function is approximated by a Gaussian process model $Q^\pi(\tilde{H}, \tilde{Y}) \sim \mathcal{GP}(0, k((\tilde{H}, \tilde{Y}), (\tilde{H}, \tilde{Y})))$.

4. Neural network based models. A feature vector is extracted by $\phi(\tilde{H}, \tilde{Y})$, and the Q-function is approximated by neural network $\text{NN}(\phi(\tilde{H}, \tilde{Y}), \mathbf{W})$.

Value-based RL 1: Grid-based model

The most simple idea is to use a limited number of discrete finite clusters to approximate the original state and action space. All states $\tilde{H}_n$ or all belief state vectors $\mathbf{b}_n$ are quantized to $N_{\tilde{H}}$ discrete states by

$$\bar{H}_n = f(\tilde{H}_n)$$

and all actions $\tilde{Y}_n$ are quantized to $N_{\tilde{Y}}$ discrete actions by

$$\bar{Y}_n = f(\tilde{Y}_n),$$

where $\bar{H}_n$ is the projected state id, and $\bar{Y}_n$ is the projected action id. The action value function $Q(\tilde{H}, \tilde{Y})$ is represented as a table of size $N_{\tilde{H}} * N_{\tilde{Y}}$. The Q-function $Q(\tilde{H}, \tilde{Y})$ representing the

expected cumulative future reward of taking action $\tilde{Y}$ under state $\tilde{H}$ can be calculated by looking up a table as

$$Q(\tilde{H}, \tilde{Y}) = \text{Table}(\bar{H}_n, \bar{Y}_n).$$

Williams *et al.* [173, 174], Young *et al.* [183] and Lefevre *et al.* [75] use hard or soft clusters to model the Q-function. The authors first use the parsed user utterance $\tilde{X}_n$ to update the belief state $\mathbf{b}_n$ and then map all possible system action $\tilde{Y}_n$ to $\mathbf{Y}_n$ with a hand-crafted feature extraction function. Then all belief states $\mathbf{b}_n$ are quantized to N_s discrete states by

$$\bar{b}_n = f(\mathbf{b}_n)$$

and all actions are quantized to N_a discrete actions by

$$\bar{Y}_n = f(\mathbf{Y}_n),$$

where $\bar{H}_n$ is the projected state id, and $\bar{Y}_n$ is the projected action id. The value function $Q(\mathbf{b}_n, \tilde{Y})$ is represented as a table of $N_s * N_a$. $Q(\mathbf{b}_n, \tilde{Y})$ represents the expected cumulative future reward of taking action $\tilde{Y}$ under belief state $\mathbf{b}_n$.

The policy is specified by

$$\tilde{Y} = \arg\max_{\tilde{Y}'} Q(\mathbf{b}_n, \tilde{Y}').$$

To learn the optimal Q-function on the quantized state cluster and action cluster, the authors first estimate the transition probability $p(\mathbf{b}'|\mathbf{b}, \tilde{Y})$ with another table of size $N_s * N_a * N_s$ via statistical calculation on the training data. Then the standard value iteration is used to update the approximated policy function on cluster level via

$$Q(\mathbf{b}_n, \tilde{Y}) = r(\mathbf{b}_n, \tilde{Y}) + \lambda \sum_{\mathbf{b}'} P(\mathbf{b}'|\mathbf{b}_n, \tilde{Y}) \max_{\tilde{Y}'} Q(\mathbf{b}', \tilde{Y}').$$

This grid-based approximation method is simple, and could work when there are a huge number of states or actions. It can even be used to approximate continuous state or action spaces with limited discrete finite clusters. The following optimization is also easy. However, it is generally hard to decide the number of clusters to used, and such quantization might introduce a lot of noise.

Value-based RL 2: Linear Model

Using discrete clusters to approximate a value function might introduce a lot of quantization noise. To deal with this problem, a linear model could be used to approximate the Q-function. With the linear model, the state space and the action space do not need to be quantized into discrete

clusters, thus it can reduce the noise introduced by quantization. First, a feature vector $\phi(\tilde{H}, \tilde{Y})$ is extracted from the state and action pair, then the action value function can be approximated by

$$Q(\tilde{H}, \tilde{Y}) = \sigma(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w})$$

where $\mathbf{w}$ is the weight vector of this linear function and σ is the activation function.

Li *et al.* [78] use a linear model to approximate the Q-function $Q(\tilde{H}, \tilde{Y})$, by

$$Q(\tilde{H}, \tilde{Y}) = \sigma(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w}).$$

The policy is specified by

$$\tilde{Y} = \arg\max_{\tilde{Y}'} Q(\tilde{H}, \tilde{Y}').$$

The authors use the least square method to train the linear model. The least square algorithm finds a parameter vector $\mathbf{w}$ by minimizing the sum-squared error between $Q(\tilde{H}_n, \tilde{Y}_n)$ and target G_n. The objective function is

$$LS(\mathbf{w}) = \sum_{n=1}^{N} (\sigma(\phi(\tilde{H}_n, \tilde{Y}_n)^T \mathbf{w}) - G_n)^2,$$

and the result is found by

$$\mathbf{w} = \arg\min_{\mathbf{w}} LS(\mathbf{w}).$$

Linear model Q-learning has less quantization error, and it is easy to train. However, the linear model assumes that the Q-function is on a plain, which might be too simple for a real world Q-function. And it requires a set of predefined features.

Value-based RL 3: Gaussian Process Model

Traditional models require a large number of dialogues to train, and they are parametric where the basis of policy representation/belief state are hand-designed by human. Gaussian process [41, 33] is non-parametric and it can avoid the limitation when the solution is constraint by the human-chosen basis. The formulation is

$$Q^{\pi}(\tilde{H}, \tilde{Y}) \sim \mathcal{GP}(m(\tilde{H}, \tilde{Y}), k((\tilde{H}, \tilde{Y}), (\tilde{H}, \tilde{Y})))$$

where $m(\tilde{H}, \tilde{Y})$ is the prior mean function and $k((\tilde{H}, \tilde{Y}), (\tilde{H}', \tilde{Y}'))$ is the kernel function. The kernel function $k((\tilde{H}, \tilde{Y}), (\tilde{H}', \tilde{Y}'))$ can be factorized into separate kernels over the state space and the action space by

$$k((\tilde{H}, \tilde{Y}), (\tilde{H}', \tilde{Y}')) = k_s(\tilde{H}, \tilde{H}') k_a(\tilde{Y}, \tilde{Y}').$$

Given training state-action sequences

$$\mathbf{B} = [(\tilde{H}_0, \tilde{Y}_0), \cdots, (\tilde{H}_n, \tilde{Y}_n)]^T$$

and the corresponding immediate rewards

$$\mathbf{r} = [r_0, \cdots, r_n]^T,$$

the $Q^\pi(\tilde{H}, \tilde{Y})$ for any state-action pair $(\tilde{H}, \tilde{Y})$ is given by

$$Q(\tilde{H}, \tilde{Y})|\mathbf{B}, \mathbf{r} \sim \mathcal{N}(\bar{Q}(\tilde{H}, \tilde{Y}), cov((\tilde{H}, \tilde{Y})(\tilde{H}, \tilde{Y}))),$$

where the posterior mean is given by

$$\bar{Q}(\tilde{H}, \tilde{Y}) = \mathbf{k}(\tilde{H}, \tilde{Y})^T \mathbf{H}^T (\mathbf{HKH}^T + \sigma^2 \mathbf{HH}^T)^{-1} (\mathbf{r} - \mathbf{m}),$$

and the covariance is given by

$$cov((\tilde{H}, \tilde{Y}), (\tilde{H}, \tilde{Y})) = k((\tilde{H}, \tilde{Y}), (\tilde{H}, \tilde{Y})) - k(\tilde{H}, \tilde{Y})^T \mathbf{H}^T (\mathbf{HKH}^T + \sigma^2 \mathbf{HH}^T)^{-1} \mathbf{Hk}(\tilde{H}, \tilde{Y}),$$

where $\mathbf{m} = [m(\tilde{H}_0, \tilde{Y}_0), \cdots, m(\tilde{H}_n, \tilde{Y}_n)]^T$, $\mathbf{k}(\tilde{H}, \tilde{Y}) = [k((\tilde{H}_0, \tilde{Y}_0), (\tilde{H}, \tilde{Y})), \cdots, k((\tilde{H}_n, \tilde{Y}_n), (\tilde{H}, \tilde{Y}))]^T$, $\mathbf{K}$ is the Gram Matrix [109], $\mathbf{H}$ is the band matrix with diagonal $[1, -\gamma]$, and σ^2 is an additive noise factor controlling the variability in the Q-function.

The policy can be constructed based on ϵ greedy by

$$\tilde{Y} = \begin{cases} \arg\max_{\tilde{Y}'} Q(\tilde{H}, \tilde{Y}') & \text{with probability } 1 - \epsilon \\ \tilde{Y}_{\text{rand}} & \text{with probability } \epsilon \end{cases}$$

However, such random exploration is inefficient. The authors propose another policy, that is to use active learning to explore the space where the model is less certain about, formulated by

$$\tilde{Y} = \begin{cases} \arg\max_{\tilde{Y}'} Q(\tilde{H}, \tilde{Y}') & \text{with probability } 1 - \epsilon \\ \arg\max_{\tilde{Y}'} \text{cov}((\tilde{H}, \tilde{Y}'), (\tilde{H}, \tilde{Y}')) & \text{with probability } \epsilon \end{cases}$$

To train a Gaussian process, the authors propose to use the on-line SARSA algorithm with the following update function

$$Q(\tilde{H}_n, \tilde{Y}_n) = Q(\tilde{H}_n, \tilde{Y}_n) + \alpha(r_{n+1} + \lambda Q(\tilde{H}_{n+1}, \tilde{Y}_{n+1}) - Q(\tilde{H}_n, \tilde{Y}_n))$$

Gaussian process Q-learning is a powerful none-parametric model, and it can void the limitation of a human selected basis. It can be trained with less training sample. However, Gaussian process based models have very high computational complexity and can only process small datasets.

Value-based RL 4: Neural Network model

To deal with non-linearity, some papers propose to use multi-layer neural network to approximate the Q-function as

$$Q(\tilde{H}, \tilde{Y}) = \text{NN}(\phi(\tilde{H}, \tilde{Y}), \mathbf{W}),$$

where $\phi(\tilde{H}, \tilde{Y})$ is the feature vector extracted. Daubigney *et al.* [26] use Multi-Layer Perceptron (MLP) to approximate the Q-function. To specify a policy, the ϵ greedy is typically used. Finally, the neural network Q-function can be optimized with any gradient-based method.

The neural network Q-learning has better representation ability and it can be trained to fit very complex functions. However, the number of parameters in a neural network is usually large, so it requires more training data to train. If the training data is insufficient, the model will easily over-fit.

In conclusion, Q-learning is easy to use and it does not require an policy function. However, it is not straight forward for it to support stochastic policy. Also, it can not deal with large action space or continuous action space, because finding an optimal action with Q-learning requires the evaluation of all possible actions in the action space.

Policy-based Reinforcement Learning

In Q-learning, selecting the best action requires the evaluation of $Q(\tilde{H}, \tilde{Y})$ for each action, which is inefficient for a large action space or a continuous action space. To deal with the continuous action space, one way is to directly specify a distribution on action space. Policy iteration is proposed to solve this problem. The core function of policy iteration is the policy function.

For a deterministic model, the policy model is a mapping function from a state $\tilde{H}$ to an action $\tilde{Y}$ defined as

$$\tilde{Y} = \pi(\tilde{H}).$$

For a probabilistic policy model, the probability of taking an action $\tilde{Y}$ under an state $\tilde{H}$ is $\pi(\tilde{H}, \tilde{Y})$, which is formulated by

$$\pi(\tilde{H}, \tilde{Y}) = p(\tilde{Y}|\tilde{H}).$$

To get a stochastic policy from a deterministic model, we can use the ϵ greedy algorithm as

$$\tilde{Y} = \begin{cases} \pi(\tilde{H}) & \text{with probability } 1-\epsilon \\ \tilde{Y}_{\text{rand}} & \text{with probability } \epsilon \end{cases}$$

To get a deterministic policy from a probabilistic policy model, the action can be decided by

$$\tilde{Y} = \pi(\tilde{H}) = \arg\max_{\tilde{Y}'} \pi(\tilde{H}, \tilde{Y}').$$

To get a stochastic policy from a probabilistic policy model, the best action can be randomly drawn from the probabilistic distribution:

$$\tilde{Y} \sim \pi(\tilde{H}, \cdot).$$

To train a policy function, we need to specify the loss function first. The averaged expected reward of a policy π on a training trajectory is

$$J_{avR}(\theta) = \sum_{\tilde{H}} d^{\pi_\theta}(\tilde{H}) \sum_{\tilde{Y}} \pi_\theta(\tilde{H}, \tilde{Y}) r_{\tilde{H}}^{\tilde{Y}}$$

where $d^{\pi_\theta}(\tilde{H})$ is the stationary distribution for Markov chain for policy π_θ. Then we improve the policy by increasing the log-probability of all actions from that episode in proportional to r, which is the goodness of the episode. We are actually maximizing the probability of the "good" episodes. We can iteratively update with the REINFORCE [178] algorithm as

$$\theta = \theta + \alpha \Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n) \tilde{G}_n,$$

where $\tilde{G}_n$ is an unbiased sample of action value function $Q(\tilde{H}_n, \tilde{Y}_n)$.

The probability $\pi(\tilde{H}, \tilde{Y})$ can be expressed by probability models or other models. Here we introduce 2 kinds of model for $p(\tilde{Y}|\tilde{H})$.

1. Softmax function. The policy function is approximated by a softmax function over a linear model, $\pi_\theta(\tilde{H}, \tilde{Y}) \propto \exp(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w})$.

2. Neural network. The policy function is approximated by a neural network with a softmax function. $\pi_\theta(\tilde{H}, \tilde{Y}) \propto \exp(\text{NN}(\tilde{H}, \tilde{Y}))$.

Policy-based RL 1: Softmax Policy-Iteration

Jurvcivcek *et al.* [58] use a softmax function to approximate the policy function. First, a feature extraction function is used to extract features $\phi(\tilde{H}, \tilde{Y})$ from a state and action pair, then the action value function can be approximated by

$$\pi_\theta(\tilde{H}, \tilde{Y}) = \frac{1}{Z(\tilde{H})} \exp(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w}),$$

where $\mathbf{w}$ is the weight vector of this linear function, $Z(\tilde{H}) = \sum_{\tilde{Y}} \exp(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w})$ is normalization term. To train a softmax policy, we can iteratively update the policy with

$$\theta = \theta + \alpha \Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n) \tilde{G}_n$$

where $\Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n)$ is defined as

$$\Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n) = (\phi(\tilde{H}_n, \tilde{Y}_n) - \mathbb{E}_{\pi_\theta}\phi(\tilde{H}_n, \cdot)).$$

Policy-based RL 2: Neural Network Policy-Iteration

Su *et al.* [130] and Wen *et al.* [166] use a neural network to approximate the policy function, and it is denoted as

$$\pi_\theta(\tilde{H}, \tilde{Y}) \propto \exp(\text{NN}(\tilde{H}, \tilde{Y})).$$

The policy network in [130] has one hidden layer and two output softmax layers. The output system action $\tilde{Y}$ contains speech-acts, slots and slot-values. The two output softmax layers are to predict the speech-acts and the slot-values in the predefined dialogue domain.

To train a neural network policy, gradient-based methods such as SGD can be used. It iteratively update the policy with

$$\theta = \theta + \alpha \Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n)\tilde{G}_n$$

where $\Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n)$ can be calculated by standard back propagation.

In conclusion, policy iteration reinforcement learning has several advantages. It can support stochastic policies and it can deal with a large and continuous action space. However, policy iteration might be unstable due to the direct use of the noisy reward signal. And typically there is no guarantee that a policy iteration algorithm will converge to the global optimal.

Actor-Critic Reinforcement Learning

The REINFORCE algorithm in policy iteration uses the noisy sample of action value function $\tilde{G}_n$ to update the policy, which might make the learning very slow. To improve the stability of the policy iteration, Su *et al.* [130], Jurvcivcek *et al.* [58] and Misu *et al.* [96, 97] propose to use the Actor-critic framework [136, 63] in task-oriented dialogue systems. An actor-critic method approximates the true expected reward with a learned Q-function, providing stable gradient for optimizing the policy function. Actor-critic methods require both the action value function model

$$Q(\tilde{H}_n, \tilde{Y}_n) = \mathbb{E}(\sum_{k=0}^{\infty} \gamma^k r_{n+k} | \tilde{H}_n, \tilde{Y}_n)$$

and the policy function model

$$\tilde{Y} = \pi(\tilde{H}).$$

The Q-function and the policy function can be approximated by models mentioned in the last two sections. Q-function can be approximated by linear models, Gaussian process models or neural network models. The policy function can be approximated by softmax models or neural network models.

To specify a policy, the best action can be directly drawn from the policy function

$$\tilde{Y} \sim \pi(\tilde{H}).$$

To specify an on-line algorithm which requires exploration, we can use the ϵ greedy defined by

$$\tilde{Y} = \begin{cases} \arg\max_{\tilde{Y}'} \pi(\tilde{H}, \tilde{Y}') & \text{with probability } 1-\epsilon \\ \tilde{Y}_{\text{rand}} & \text{with probability } \epsilon \end{cases}$$

To train a policy with actor critic, we have to iteratively predict the current policy, and then improve the current policy. To improve the stability of a policy-based method, instead of using the direct noisy reward with great variance, the estimated action-value function $Q(\tilde{H}, \tilde{Y})$ is used to update the policy function $\pi(\tilde{Y}|\tilde{H})$. To learn the policy function, we use an objective function

$$J_{avR}(\theta) = \sum_{\tilde{H}} d^{\pi_\theta}(\tilde{H}) \sum_{\tilde{Y}} \pi_\theta(\tilde{H}, \tilde{Y}) Q(\tilde{H}, \tilde{Y}),$$

where $d^{\pi_\theta}(s)$ is the stationary distribution for Markov chain for policy π_θ. Then we iteratively update the prediction function $Q(\tilde{H}, \tilde{Y})$ and policy function $\pi_\theta(\tilde{H}, \tilde{Y})$ by

$$Q(\tilde{H}_n, \tilde{Y}_n) = \alpha(r_{n+1} + \lambda Q(\tilde{H}_{n+1}, \tilde{Y}_{n+1}) - Q(\tilde{H}_n, \tilde{Y}_n))$$

and

$$\theta = \theta + \alpha \Delta_\theta \log \pi_\theta(\tilde{H}_n, \tilde{Y}_n) Q(\tilde{H}_n, \tilde{Y}_n).$$

The actor-critic framework improves the policy iteration framework by using an estimated Q-function to substitute the noisy reward signal, so it is typically more stable than the policy iteration. Like the policy iteration, it can deal with a large and continuous action space. However, actor-critic requires the careful choice of two compatible models, one for the Q-function and the other for the policy function.

Evaluation for Single Domain Dialogue Policy

To evaluate, the whole dialogue system is tested with the user simulator [117] on various domains such as the Town Information [144] and the Cambridge Restaurant Domain (TopTable) [41]. The

Table 3.3: Dialogue Policy Evaluation

Method	Dataset	Reward	Success Rate	# dialogue turns
Gaussian Process	TopTable	11.6 ± 0.4	0.912 ± 0.014	6.6 ± 0.2
Gaussian Process online	TopTable	13.4 ± 0.3	0.968 ± 0.009	6.0 ± 0.1
NAC [58]	TownInfo	3 ± 0.3		
NN based NAC [130]	TopTable		0.91	

BUDS [143] state representation is used. Evaluation metrics include the average reward, the success rate and the number of dialogue turns. The results are shown in Figure 3.3.

Q-learning framework is easy to use, and does not require a predefined policy function. However, it is not straight forward for Q-learning to support stochastic policies. And Q-learning can only be applied on situations with small and discrete action spaces, because choosing an optimal action requires an evaluation on every action.

Policy iteration framework does not require the definition of a Q-function, and we can easily define a stochastic policy based on the policy function. It can deal with large and continuous action spaces. However, the optimization of policy iteration can be unstable because the noisy reward in the dataset is directly used. And usually there is no guarantee that the policy iteration converges to the global optimal.

Actor-critic framework improves the policy iteration framework by using an estimated Q-function instead of the noisy reward signal, so it is typically more stable than the policy iteration. Like the policy iteration, it can deal with large and continuous action spaces. However, actor-critic requires the careful choice of two compatible models, one for the Q-function and the other for the policy function.

Then we compare different RL-frameworks qualitatively in Table 3.4. The comparison of different approximation models in Q-learning is summarized in Table 3.5.

Table 3.4: RL-framework Qualitative Comparison

Method	Q-Learning	Policy-Iteration	Actor-Critic
Core-Function	$Q(s, a)$	$\pi(s, a)$	$Q(s, a)$ and $\pi(s, a)$
State Space	Discrete and Continuous	Discrete and Continuous	Discrete and Continuous
Action Space	Discrete	Discrete and Continuous	Discrete and Continuous
Policy	Deterministic	Deterministic and Stochastic	Deterministic and Stochastic
Test Speed	Slow, require $Q(s, a) \forall a$	Fast, sample $a \sim \pi(s, a)$	Fast, sample $a \sim \pi(s, a)$
Convergence	Stable	Unstable	Stable

Table 3.5: Model comparison for Q-learning

Method	Grid	linear	Gaussian Process	Neural Network
Number of parameter	Small	Small	Linear to size of training data	Big
Representation ability	Low	Medium	High	Very High
Training method	Statistics	Gradient-based	Not required	Gradient-based
Testing speed	Fast	Fast	Slow, scan the whole training set	Medium

3.4.3 Transfer Learning for Dialogue Policy Learning

In this section, we survey the multi-domain transfer learning papers for the dialogue policy learning problem. Multi-domain transfer learning aims to use the knowledge in one domain to help the policy learning in the target domain, and the problem focuses on improving the dialogue performance in the target domain only. Here different domains [36, 35, 37, 38, 34, 158, 160] can be dialogues with different target tasks. Also, dialogues with different persons [20, 42, 8, 60] can be viewed as different domains since the data from different persons have different distributions. Here, we focus on the transfer learning-based reinforcement learning methods.

Most methods are based on the Q-learning framework. Based on the model used to approximate the Q-function, we categorize these works into linear model based, Gaussian process transfer and Bayesian committee machine transfer.

1. Linear model transfer for Q-learning. A simple and computational efficient model, but its use is limited to the case when the source domain data and the target domain data have the same feature space.

2. Gaussian process transfer for Q-learning. A powerful none-parametric model that can deal with data in different feature spaces, but in this case the source domain and the target domain have to share some common slots.

3. Bayesian committee machine transfer for Q-learning. A powerful ensemble model based on the Gaussian process that can transfer when the source domain and target domain have no common slots. An entropy-based cross-domain kernel function is used to transfer knowledge from a source domain to a target domain.

Most of the existing transfer learning methods are based on transferring state-action pairs from the source to the target domain. The data points in the source domain are weighted by their similarity to the target domain data points. The core problem is how to determine the similarities of cross-domain data points.

Linear Model transfer for Q-learning

Genevay *et al.* [42] propose to transfer a liner model. The task is to adapt an existing user model to a new user. A feature extraction function is used to extract feature vector $\phi(\tilde{H}, \tilde{Y})$ from the state and action pair, then the action value function can be approximated by

$$Q(\tilde{H}, \tilde{Y}) = \sigma(\phi(\tilde{H}, \tilde{Y})^T \mathbf{w}),$$

where $\phi(\tilde{H}, \tilde{Y})$ is the feature vector for the state-action pair $(\tilde{H}, \tilde{Y})$, $\mathbf{w}$ is the weight vector of this linear function, and σ is the activation function.

The authors propose to select only transitions that are far enough from the target domain. For each trajectory in the source domain $< \tilde{H}^s, \tilde{Y}^s, \tilde{H'}^s, r^s >$, if there exist a trajectory in the target domain $< \tilde{H}, \tilde{Y}, \tilde{H}', r >$ whose $\tilde{Y} = \tilde{Y}^s$ and $||\tilde{H} - \tilde{H}^s|| \leq \eta$, then this source trajectory $< \tilde{H}^s, \tilde{Y}^s, \tilde{H'}^s, r^s >$ will not be transferred to the target domain. The policy will be initially trained on the selected source data points $\mathcal{D}^s = \{< \tilde{H}^s, \tilde{Y}^s, \tilde{H'}^s, r^s >\}$

$$\mathbf{w}_{\text{init}} = \arg\min_{\mathbf{w}'} L^s(\mathbf{w}', \mathcal{D}^s = \{< \tilde{H}^s, \tilde{Y}^s, \tilde{H'}^s, r^s >\}).$$

Then the policy parameters are transferred to the target domain and continue to be updated on the target domain data $\mathcal{D} = \{< \tilde{H}, \tilde{Y}, \tilde{H}', r >\}$.

In [42], the task is to adapt an existing user model to a new user. The similarities between the source domain and the target domain $||\tilde{H} - \tilde{H}^s||$ are calculated based on a set of predefined features.

1. The constant feature 1.
2. The utility loss between the last proposed time-slot and the next one.
3. The square of the previous feature.
4. The number of time-slots which can still be proposed.
5. The length of the dialogue.
6. The speech recognition score.

Linear model transfer for Q-learning is a simple and efficient algorithm, but it can only transfer when the source and target data have the same feature space.

Gaussian Process transfer for Q-learning

In [36, 35, 37, 38, 34], the authors use a Gaussian process model for Q-function, which is defined as

$$Q^{\pi}(\tilde{H},\tilde{Y}) \sim \mathcal{GP}(m(\tilde{H},\tilde{Y}), k((\tilde{H},\tilde{Y}),(\tilde{H},\tilde{Y}))),$$

where $m(\tilde{H},\tilde{Y})$ is the prior mean function and $k((\tilde{H},\tilde{Y}),(\tilde{H}',\tilde{Y}'))$ is the kernel function. The kernel function $k((\tilde{H},\tilde{Y}),(\tilde{H}',\tilde{Y}'))$ can be factorized into separate kernels over the state space and the action space via

$$k((\tilde{H},\tilde{Y}),(\tilde{H}',\tilde{Y}')) = k_{\tilde{H}}(\tilde{H},\tilde{H}')k_{\tilde{Y}}(\tilde{Y},\tilde{Y}').$$

Given training state-action sequences $\mathbf{B} = [(\tilde{H}_0,\tilde{Y}_0),\cdots,(\tilde{H}_n,\tilde{Y}_n)]^T$ and the corresponding immediate rewards $\mathbf{r} = [r_0,\cdots,r_n]^T$, the Q-function $Q^{\pi}(\tilde{H},\tilde{Y})$ for any state-action pair $(\tilde{H},\tilde{Y})$ is given by

$$Q(\tilde{H},\tilde{Y})|\mathbf{B},\mathbf{r} \sim \mathcal{N}(\bar{Q}(\tilde{H},\tilde{Y}), cov((\tilde{H},\tilde{Y})(\tilde{H},\tilde{Y})))$$

where the posterior mean is given by

$$\bar{Q}(\tilde{H},\tilde{Y}) = \mathbf{k}(\tilde{H},\tilde{Y})^T\mathbf{H}^T(\mathbf{HKH}^T + \sigma^2\mathbf{HH}^T)^{-1}(\mathbf{r}-\mathbf{m}),$$

and the covariance is given by

$$cov((\tilde{H},\tilde{Y}),(\tilde{H},\tilde{Y})) = k((\tilde{H},\tilde{Y}),(\tilde{H},\tilde{Y})) - k(\tilde{H},\tilde{Y})^T\mathbf{H}^T(\mathbf{HKH}^T + \sigma^2\mathbf{HH}^T)^{-1}\mathbf{Hk}(\tilde{H},\tilde{Y}),$$

where $\mathbf{m} = [m(\tilde{H}_0,\tilde{Y}_0),\cdots,m(\tilde{H}_n,\tilde{Y}_n)]^T$, $\mathbf{k}(\tilde{H},\tilde{Y}) = [k((\tilde{H}_0,\tilde{Y}_0),(\tilde{H},\tilde{Y})),\cdots,k((\tilde{H}_n,\tilde{Y}_n),(\tilde{H},\tilde{Y}))]^T$, $\mathbf{K}$ is the Gram Matrix [109], $\mathbf{H}$ is the band matrix with diagonal $[1,-\gamma]$ and σ^2 is an additive noise factor controlling the variability in the Q-function.

To transfer a Gaussian process policy, there are basically 2 kinds of method.

1. $\bar{Q}(\tilde{H},\tilde{Y})$ Mean function transfer. Gavsic *et al.* [34, 36, 35] propose to directly transfer source data points to the target domain, to help build a better Q-function $\bar{Q}(\tilde{H},\tilde{Y})$. Gavsic *et al.* [37, 38] and Casanueva *et al.* [20] propose to use the mean function from the source domain as a prior on the target domain.

2. $cov((\tilde{H},\tilde{Y}),(\tilde{H},\tilde{Y}))$ Covariance Function transfer. Gavsic *et al.* [37, 38, 34, 36, 35] and Casanueva *et al.* [20] propose to define kernel functions on state-action pairs from different domains.

We can see that, both the mean function $\bar{Q}(\tilde{H}, \tilde{Y})$ and the covariance function $cov((\tilde{H}, \tilde{Y}), (\tilde{H}, \tilde{Y}))$ depend on the cross-domain kernel function $k((\tilde{H}, \tilde{Y}), (\tilde{H}', \tilde{Y}'))$, so the cross-domain kernel function is the core of the transfer learning. Based on different definitions of kernel functions, we have different kinds of transfer method.

Cross-domain kernel function

In [38], only the common slots in the basic domain $\mathcal{S}$ are used, slots not appeared in the source domain $\mathcal{S}$ are discarded. The belief state is defined according to BUDS [144], and the cross-domain kernel function is

$$k_{\tilde{H}}(\tilde{H}^s, \tilde{H}) = \sum_{s \in \mathcal{S}} < \tilde{H}^s_s, \tilde{H}_s >$$

where s are slots in the basic domain $\mathcal{S}$. The kernel function between a source speech-act a^s and a target speech-act a is defined as

$$k_A(a^s, a) = \begin{cases} \delta_{a^s}(a) & a \in \mathcal{A}^s \\ 0 & a \notin \mathcal{A}^s \end{cases}$$

where $\mathcal{A}^s$ and $\mathcal{A}^t$ are the collection of speech-acts in the source and the target domains respectively, and $\delta_{a^s}(a)$ is the kernel function defined in the source domain.

In [37], the kernel function is predefined. The authors define the cross-domain kernel function based on the common slots in the basic and the extended domains. For slots only appeared in the extended domain, the most similar slots are used to calculate the kernel function. The kernel is defined as

$$k_{\tilde{H}}(\tilde{H}^s, \tilde{H}) = \sum_{s^s \in \mathcal{S}} < \tilde{H}^s_{s^s}, \tilde{H}_{s^s} > + \sum_{s^t \notin \mathcal{S}} < \tilde{H}^s_{l(s^t)}, \tilde{H}_{s^t} >$$

where s^s are slots in the basic domain $\mathcal{S}$, s^t are slots in the extended domain $\mathcal{T}$, the function $l : \mathcal{T} \rightarrow \mathcal{S}$ finds a basic domain slot $l(s^t)$ that is most similar to the extended domain slot s^t. The kernel function for actions is defined as

$$k_a(a^s, a) = \begin{cases} \delta_{a^s}(a) & a \in \mathcal{A}^s \\ \delta_{a^s}(L(a)) & a \notin \mathcal{A}^s \end{cases}$$

where the function $L : \mathcal{A}^t \rightarrow \mathcal{A}^s$ maps an action that does not exist in the source domain to a replacement action in the source domain, and $\delta_{a^s}(a)$ is the kernel function defined in the source domain.

In [20], where the source domain and the target domain are from different persons with the same set of slots, the authors propose to use additional features to determine the kernel

$$k((\tilde{H}^s, \tilde{Y}^s), (\tilde{H}, \tilde{Y})) = k_{\tilde{H}}(\tilde{H}^s, \tilde{H}) k_{\tilde{Y}}(\tilde{Y}^s, \tilde{Y}) k_{\tilde{H}}(\mathbf{l}^s, \mathbf{l})$$

where $\mathbf{l}^s$ is the acoustic feature vector for the state-action pair in the source domain and $\mathbf{l}$ is the feature vector in target domain. Now the kernel depends on some external features, which could help better calculate the cross-domain data similarity.

Gavsic *et al.* [34] propose to build a distributed policy for each node in the knowledge graph. A dialogue policy is decomposed into a set of topic specific policies that are distributed across the class nodes in the graph. The root node in the knowledge graph is general for all its children nodes, so the policy in this root node can work for all sub-domains. And as more data from specific sub-domains (leaf nodes) are collected, the policies in the sub-domain can be improved without affecting the performance of the dialogue. The authors propose to match only the common slots first with

$$k_{\tilde{H}}(\tilde{H}^s, \tilde{H}) = \sum_{s \in \mathcal{S} \cup \mathcal{T}} < \tilde{H}^s_s, \tilde{H}_s >$$

and

$$k_A(a^s, a) = \begin{cases} \delta_{a^s}(a) & a \in \mathcal{A}^s \\ 0 & a \notin \mathcal{A}^s \end{cases}$$

where $\delta_{a^s}(a)$ is the kernel function defined in the source domain. If there is no common slots, the none-matching slots are renamed to be "slot-1", "slot-2" and treated as abstract slots. Abstract slots in the source domain and the target domain are matched one-by-one in order, if these abstract slots have different cardinalities, slots with a shorter vector is padded with zeros.

Gaussian process transfer for Q-learning does not assume a completely identical feature space in the source domain and the target domain, and it is a none-parametric method. However, Gaussian process transfer still assumes that there are common slots between the source and target domains. Also Gaussian process transfer is computationally expensive, thus it could not support large training datasets.

Bayesian Committee Machine Transfer for Q-learning

Previous methods assume the existence of common slots, however this assumption is not always true. When there is no common slots, we can use the Bayesian committee machine to transfer a dialogue policy. The Bayesian committee machine is an approach to combine policies trained on different datasets in different domains. Bayesian committee machine is particularly suitable for Gaussian process models [145]. Gavsic *et al.* [36, 35] use this model to combine different Q-functions learned from different datasets.

A Bayesian committee machine is a Gaussian process, of which the combined mean function

$\bar{Q}(\tilde{H}, \tilde{Y})$ is calculated as

$$\bar{Q}(\tilde{H}, \tilde{Y}) = \Sigma^{Q}(\tilde{H}, \tilde{Y}) \sum_{i=1}^{M} \Sigma_i^{Q}(\tilde{H}, \tilde{Y})^{-1} \bar{Q}_i(\tilde{H}, \tilde{Y}),$$

and the covariance function Σ^{Q} is calculated as

$$\Sigma^{Q}(\tilde{H}, \tilde{Y})^{-1} = -(M-1) * k((\tilde{H}, \tilde{Y}), (\tilde{H}, \tilde{Y}))^{-1} + \sum_{i=1}^{M} \Sigma_i^{Q}(\tilde{H}, \tilde{Y})^{-1},$$

where M is the number of policies in the Bayesian committee machine, $Q_i(\tilde{H}, \tilde{Y})$ is the Q-function of the i-th policies, $\bar{Q}_i$ is the mean of $Q_i(\tilde{H}, \tilde{Y})$ and Σ_i^{Q} is the covariance of $Q_i(\tilde{H}, \tilde{Y})$. Note that $Q_i(\tilde{H}, \tilde{Y})$ is trained on a set of state-action and reward pairs $\mathbf{r}_i$, $\mathbf{B}_i$ for $i \in \{1, \cdots, M\}$.

To evaluate a state-action pair $(\tilde{H}, \tilde{Y})$, the Bayesian committee machine requires this state-action pair to be predicted by all $Q_i(\tilde{H}, \tilde{Y})$, for $i \in 1, \cdots, M$. So a kernel function has to be defined between state-action pairs in different domains.

In [36], the policy Q_i is trained on the domain data $\mathbf{r}_i$ and $\mathbf{B}_i$, different policies are trained separately. In [35], the authors propose to train different Q_i in parallel via reinforcement learning. Training is done by distributing rewards of the whole Bayesian committee machine to each individual policy in different domains. They propose 3 different methods for distributing the rewards.

Like the Gaussian process, the cross-domain kernel function $k((\tilde{H}, \tilde{Y}), (\tilde{H}', \tilde{Y}'))$ is the core of the transfer learning method.

Entropy-based kernel function In [36, 35], no shared slots are assumed in the source domain and target domains. The source and target slots are matched one-by-one based on the normalized entropy ranking index.

For each slot s, the normalized entropy η is calculated by

$$\eta(s) = -\sum_{v \in V_s} \frac{p(s=v)\log(p(s=v))}{|V_s|}$$

where v is the slot-value from a slot-value set V_s, $|V_s|$ is the total number of slot values in slot s and where $p(s=v)$ is the empirical probability for an entity having slot value v in slot s. For example, if all restaurants have "area=centre", then the area slot has a normalized entropy equal to 0. The normalized entropy measures how useful a slot is in a dialogue.

For each domain $c \in \{\mathcal{S}, \mathcal{T}\}$, the slots are sorted based on their normalized entropy so that $\eta(s_i^c) \geq \eta(s_j^c)$ for $i \leq j$ in domain c. The kernel function between source domain $\mathcal{S}$ and target domains $\mathcal{T}$ is calculated in the following ways:

1. Iteratively, for s_i^c in domain $c \in \{\mathcal{S}, \mathcal{T}\}$ when the index i of the ordered list satisfies $i \leq min(|\mathcal{S}|, |\mathcal{T}|)$ ($|c|$ denotes the number of slots in domain c): Match the corresponding elements of belief state and actions, padding with zeros as necessary.

2. Otherwise if $i > min(|\mathcal{S}|, |\mathcal{T}|)$, disregard the elements of the belief state relating to unpaired slots j and if one of the actions is related to $slot_j$, consider the action kernel to be 0.

The Bayesian committee machine does not assume the existence of common slots in the source and target domains, instead an entropy-based cross-domain kernel function is defined, which can estimate the data similarity between different domains. However, each of the committee is a Gaussian process model, which is still computationally expensive, and thus it could not support large datasets.

Evaluation for Multi-domain dialogue policy

In this section we compare and discuss the advantages and disadvantages of different multi-domain policies. The algorithms are tested with user simulators.

The dataset used in the evaluation are

1. SFR: restaurant in San Francisco.

2. SFH: hotel in San Francisco.

3. L11: laptop with 11 properties.

The evaluation metrics used are

1. Reward: 20 reward at the end of each successful dialogue, -1 for each dialogue turn to encourage shorter conversation.

2. Success Rate: Ratio of conducting successful dialogues.

3. Number of Turns: The number of dialogue turns before completing a task.

We first report the qualitative performance comparison results, the results are in Table 3.6.

Then we compare different methods qualitatively in Table 3.7.

The linear model transfer for Q-learning method is simple and efficient, and it can easily support very large training datasets. However, this method can only be applied when the source domain and the target domain have the same feature space.

Table 3.6: Transfer Performance Comparison

Test Dataset	Train	Reward	Success	#Turn
SFR [34] Generic	SFR + SFH	8.58 ± 0.15	86.21 ± 0.68	8.52 ± 0.07
SFR [34] In-domain	SFR + SFH	8.68 ± 0.15	86.67 ± 0.67	8.54 ± 0.07
SFR [34] Adapted	SFR + SFH	9.62 ± 0.30	89.60 ± 1.90	8.24 ± 0.19
SFR [36, 35] Generic	SFR + SFH + L11	6.32 ± 0.21	72.04 ± 0.89	8.05 ± 0.08
SFR [36, 35] In-domain	SFR + SFH + L11	5.73 ± 0.21	68.17 ± 0.92	7.89 ± 0.08
SFR [36, 35] MBCM	SFR + SFH + L11	7.37 ± 0.20	76.60 ± 0.83	7.92 ± 0.08

Table 3.7: Transfer Setting Qualitative Comparison

Method	Common Features	Common Slot	No Common Slot
Linear-Feature [42]	Yes		
GP-kernel [34, 20]	Yes	Yes	
MBCM [36, 35]	Yes	Yes	Yes

The Gaussian process transfer does not require the source and target domains to share the same feature space, and it is none-parametric method, which is more flexible. However, the Gaussian process transfer assumes the existence of common slots in the source and target domains, which it is computationally expensive and it could not support very large training datasets.

The Bayesian committee machine does not assume the existence of common slots in the source and target domains, instead an entropy-based cross-domain kernel function is defined, which can estimate the data similarity between different domains. However, each of the committee is still a Gaussian process, which is computationally expensive and could not support large training dataset.

3.5 Module 4: Natural Language Generation (NLG)

In this section we introduce the Natural Language Generation (NLG).

3.5.1 Problem Definition

Natural language generation aims to convert a system action into an appropriate sentence. A system action $\tilde{Y}_n$ consists of a dialogue speech-act a_n and a set of attribute-value pairs $\mathbf{s}_n = \{s_i = v_i\}_{i=1}^{l}$.

1. Input: A dialogue speech-act and slot values $\tilde{Y}_n = \{a_n, \mathbf{s}_n = \{s_i = v_i\}_{i=1}^{l}\}$ where a_n is the dialogue speech-act and s_i and v_i are the name and value of the i-th attribute.
2. Output: The final system response sentence Y_n.
3. Objective: Generate text response with adequacy, fluency, readability and variation.

3.5.2 Single Domain Natural Language Generation

There are basically 3 kinds of method in single domain NLG.

1. Traditional Sentence Planning and Surface Realization. The language generation is separated into 2 independent parts. The sentence planning is to convert a structured system action into an intermediate representation such as a sentence planning tree, and the surface realization converts the intermediate representation to a final text sentence.
2. Corpus-based. Corpus-based methods learn a part of the generation decision from the data corpus, which can reduce the need of human knowledge and heuristic.
3. Neural Network based. Neural network based methods learn to directly generate the final sentence word by word, without explicit models of sentence planning and surface realization. No intermediate representation is generated by this kind of method.

Sentence Planning and Surface Realization based NLG

The language generation is separated into 2 independent parts. Firstly, the sentence planning is to convert a structured system action into an intermediate representation such as a sentence planning tree. Then the surface realization converts the intermediate representation to the final text sentence.

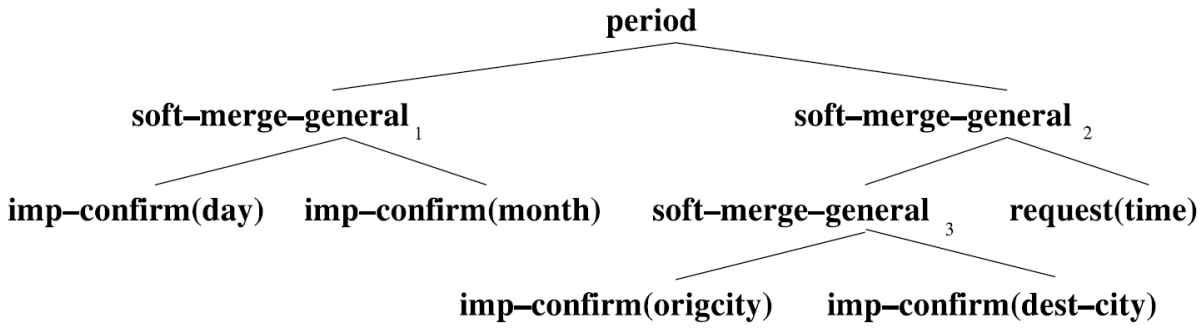


Figure 3.4: The Sentence Planning Tree [154] for the sentence "Leaving on September the 1st. What time would you like to travel from Newark to Dallas?"

Walker *et al.* [154] and Stent *et al.* [129] propose to use sentence planning and surface realization to generate natural languages, then they rank all candidates with statistical methods. The Figure 3.4 shows an example sentence planning tree for the sentence "Leaving on September the 1st. What time would you like to travel from Newark to Dallas?". The input system action is firstly converted into a sentence planning tree. Each leave node of the tree is a elementary system action, such as "request(depart-time)" in the airport domain. Two nodes are repeatedly merged into an interior node, and each merge uses one of the merge operation defined in the "Deep-Syntactic Structure". The list of all available merge operations are shown in Figure 3.5. The operations are Merge, Merge-general, Soft-Merge, Soft-Merge-General, Conjunction, Relative-clause, Adjective, Period and Random-Cueword. The root node of the tree represents a complete generated sentence. In this paper, the merging order can be arbitrary. The surface realization converts each node to human readable text recursively. The elementary system action is converted into a sentence by a human defined rule set. Two children sentences are merged into one parent sentence by following the rules in "Deep-Syntactic Structure".

Belz *et al.* [13] propose to use statistical models to learn the most likely context-free derivation for sentence planning given a corpus. Stent *et al.* [128] propose to automatically extract sentence plan construction rules from the public corpus. These extracted rules use only domain-independent features, making them able to generalize.

Conventional dialogue generation consists of sentence planning and surface realization, and most of the system are hand-crafted. The advantage is the intermediate representation has clear meaning and we can easily incorporate human knowledge into the system. However, such a system requires much corpus labelling and is expensive to build, and it has limited linguistic coverage. As a result, the hand-crafted components can hardly be reused in other tasks.

Rule	Sample first argument	Sample second argument	Result
MERGE	You are leaving from Newark.	You are leaving at 5	You are leaving at 5 from Newark
MERGE-GENERAL	What time would you like to leave?	You are leaving from Newark.	What time would you like to leave from Newark?
SOFT-MERGE	You are leaving from Newark	You are going to Dallas	You are traveling from Newark to Dallas
SOFT-MERGE-GENERAL	What time would you like to leave?	You are going to Dallas.	What time would you like to fly to Dallas?
CONJUNCTION	You are leaving from Newark.	You are going to Dallas.	You are leaving from Newark and you are going to Dallas.
RELATIVE-CLAUSE	Your flight leaves at 5.	Your flight arrives at 9.	Your flight, which leaves at 5, arrives at 9.
ADJECTIVE	Your flight leaves at 5.	Your flight is non-stop.	Your nonstop flight leaves at 5.
PERIOD	You are leaving from Newark.	You are going to Dallas.	You are leaving from Newark. You are going to Dallas

Figure 3.5: Example of operations in Deep-Syntactic Structure [154]

Corpus-based NLG

Corpus-based NLG aims to learn generation decisions from the data, which can reduce the system's dependency on the human heuristic or rules.

The class based N-gram model [101] is proposed for language modelling, and Ratnaparkhi *et al.* [110] propose some variants of this language model.

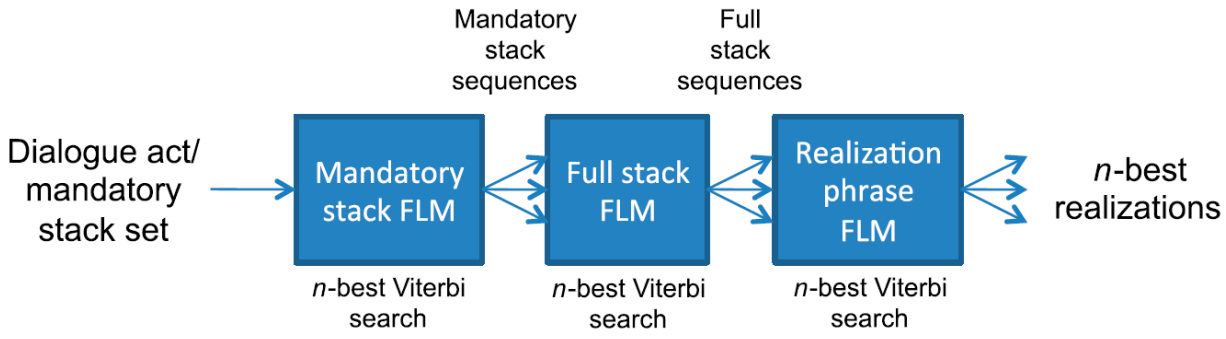


Figure 3.6: Generation Process in [86]

Mairesse *et al.* [86] propose a phase-based NLG system on top of a factored language model. It can learn from semantically aligned corpus. A system action is represented by a set of unordered mandatory semantic stacks, where each semantic stack is a piece of information that must be con-

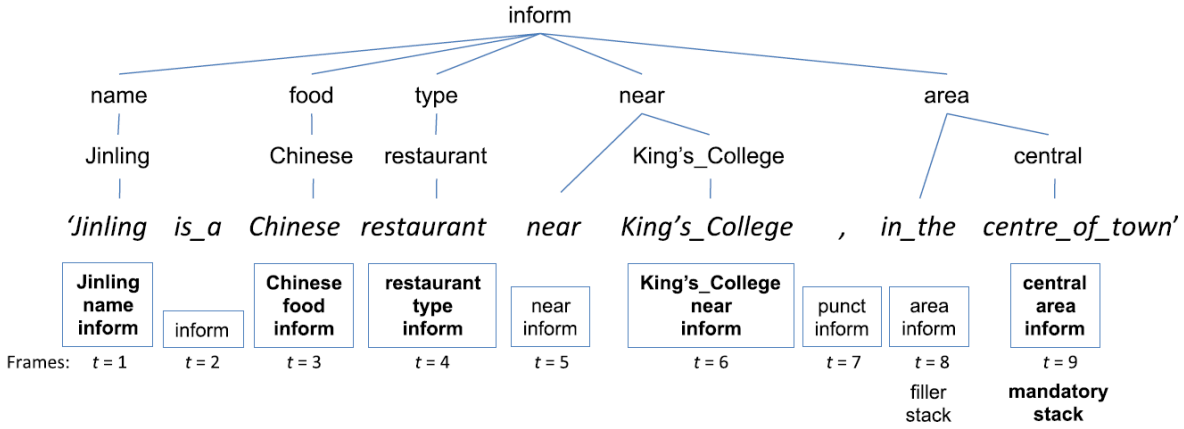


Figure 3.7: Ordered Mandatory Semantic Stack are in **bold** [86]

veyed in the generated sentence. The whole process is a sequence of searches conditioned on the system action, shown in Figure 3.6. In order to simplify the conditional probability, only the last 2 time steps are considered in the generation process. Given a set of **unordered mandatory** semantic stacks $q = \{s_i, v_i\}_{i=1}^{l}$, the model firstly samples the most likely **ordered mandatory** semantic stack sequence $\hat{\mathbf{q}}$ with an independent learned model via

$$\hat{\mathbf{q}} = \arg\max_{\hat{\mathbf{q}}} P(\hat{\mathbf{q}}|q)$$

and

$$P(\hat{\mathbf{q}}|q) = \prod_{i=1}^{N} P(\hat{\mathbf{q}}_i|q_{i-1}, q_{i-2}).$$

The **full sequence** of a semantic stack $\mathbf{q}$ is shown in Figure 3.7, and it contains the ordered mandatory semantic stack sequence $\hat{\mathbf{q}}$ as a sub-sequence. Then $\mathbf{q}$ is generated with another learned model as

$$\mathbf{q} = \arg\max_{\mathbf{q}} P(\mathbf{q}|\hat{\mathbf{q}}),$$

and

$$P(\mathbf{q}|\hat{\mathbf{q}}) = \prod_{i=1}^{N} P(\mathbf{q}_i|\hat{\mathbf{q}}_{i-1}, \hat{\mathbf{q}}_{i-2}).$$

Finally a sequence of words is generated conditioned on the full sequence of semantic stacks $\mathbf{q}$ by

$$Y = \arg\max_{Y} p(Y|\mathbf{q}),$$

and

$$P(Y|q) = \prod_{i=1}^{N} P(y_i|\mathbf{q}_{i-1}, \mathbf{q}_{i-2}).$$

Angeli *et al.* [4] propose to use log-linear models to rank and find the most suitable template for realization. The authors break the generation process into a set of 3 hierarchical decisions. The first one is the record selection, which is to select the record to talk about based on the dialogue state. The second one is the field decision, which is to select the field of the records to talk about. The final one is the template decision, which is to select a template to express the meaning. For each decision, the authors propose to train a log-linear classifier on top of a set of domain-independent features.

Kondadadi *et al.* [64] propose to use SVM [22] ranking models for all steps of the NLG process including sentence planning and surface realization. The authors propose 4 steps. The first step is to pre-process and extract templates, the second step is to build conceptual units such as company, time and date. The second step is semi-automatic and requires the help of a human expert. The third step is to extract various predefined features. The final step is to train a SVM ranking model based on the extracted features. It is shown that the SVM ranker can make the final result comparable to the performance of a human-authored text.

Corpus-based methods can learn part of the generation decision from data corpus, which can reduce the required human efforts and are less expensive. Statistical learning also makes the corpus-based methods prone to noise. However, the surveyed corpus-based models can only deal with a fixed length of context. As a result it might generate incoherent sentence if no enough context is considered.

Neural Network based NLG

Neural network based models use neural networks to directly generate natural language word by word, without explicit modelling the sentence planning and surface realization processes. The neural network can model context of arbitrary length. The whole model is almost automatic and requires no human heuristic or rules.

Mikolov *et al.* [94] propose to use a RNN [44] model for language modelling. It can model context of arbitrary length, and it is simple and can be trained on many domains. The hidden state vector the t-th word is

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{U}\mathbf{y}_t + \mathbf{b}_h),$$

and the probability of generating the current word y_t conditioned on all the history is defined as

$$p(\mathbf{y}_t|\mathbf{y}_{<t}) = \frac{1}{Z(\mathbf{h}_t)}\sigma(\mathbf{w}_{y_t}^T \mathbf{h}_t),$$

where $\mathbf{y}_t$ is the word embedding vector of y_t. $\Theta = \{\mathbf{W}_h, \mathbf{U}, \mathbf{b}_h, \mathbf{W}_y\}$ are parameters to learn, and $Z(\mathbf{h}_t) = \sum_{y'} \exp(\mathbf{w}_{y'}^T \mathbf{h}_t)$ is the softmax normalization term.

Mikolov *et al.* [95] propose several ways to reduce the number of parameters in the RNNLM. The idea is to factorize the conditional dependency by assuming each word y_t belongs to a specific class c_{y_t}. In the generation process, instead of directly generating a word, the model first decides the class and then generates the specific word based on the selected class via

$$p(y_t|y_{<t}) = p(y_t|c_{y_t})p(c_{y_t}|y_{<t})$$

where y_t is the t-th word in the generated sentences, and c_{y_t} is the class of the word y_t.

Wen *et al.* [164] propose to combine a semantically conditioned RNN sentence generation with a CNN and a RNN ranker. The conditional vector $\mathbf{s}_n$ is a system action one-hot vector, where each segment of $\mathbf{s}_n$ corresponds to a one-hot representation of an attribute in the system action $\tilde{Y}_n$, e.g., "<food>". $\mathbf{s}_n$ is used to control the generation process, in order to ensure all information is included in the generated sentence and to reduce information repetition. Firstly, an RNN is used to generate a set of delexicalised sentences, as in Figure 3.8.

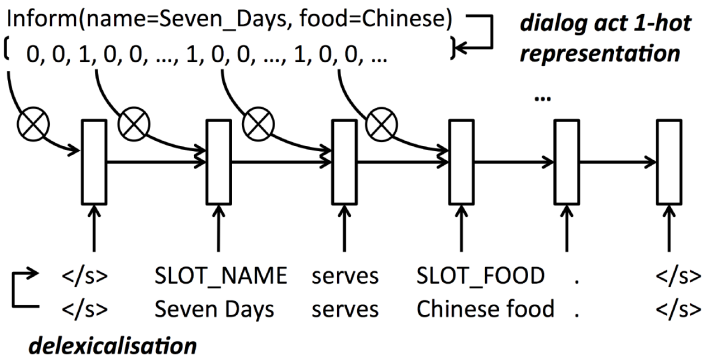


Figure 3.8: Conditional RNN architecture [164]

The hidden state is defined as

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{U}\mathbf{y}_t + \mathbf{W}_d \mathbf{s}_t)$$

and the probability of generating the next word y_t is

$$p(\mathbf{y}_t|\mathbf{y}_{<t}) = \frac{1}{Z(\mathbf{h}_t)} \sigma(\mathbf{w}_{y_t}^T \mathbf{h}_t),$$

where y_t is the t-th word in the generated sentence, $\mathbf{y}_t$ is the word embedding vector of y_t, σ is the activation function, and $Z(\mathbf{h}_t) = \sum_{y'} \exp(\mathbf{w}_{y'}^T \mathbf{h}_t)$ is the softmax normalization term. Note that all words has been delexicalised before the process. The conditioning vector $\mathbf{s}_t$ is a gated version of

the input conditional vector $\mathbf{s}_n$. The conditional vector $\mathbf{s}_n$ can be split into multiple segments where each segment $\mathbf{s}_s$ corresponds to a slot s in the dialogue domain. The content of each segment is updated by

$$\mathbf{s}_{s,t} = \mathbf{s}_s \odot \delta^{t-t_s}$$

where t_s is the time when slot s first appears in the generated sequence, δ is the decay factor, and $\odot$ is element-wise multiplication. The idea is to decay the probability of generating the slot s after it first appear, in order to avoid repetition.

Secondly, a CNN is trained to extract discriminative features with respect to the action type and the attribute value, then a ranking model is applied on the generated sentence candidates. Thirdly, a reverse direction RNN language model is used to score all generated sentences. Although the bidirectional RNN shows promising performance, but it is not straight forward to use in the sequential generation process. Finally, various ranking methods are considered simultaneously to decide the best generated sentence. However in the generation process, a heuristic function is still required to control the conditional vector $\mathbf{f}_t$. If some attribute values cannot be delexicalised effectively, the after ranker will easily make mistakes.

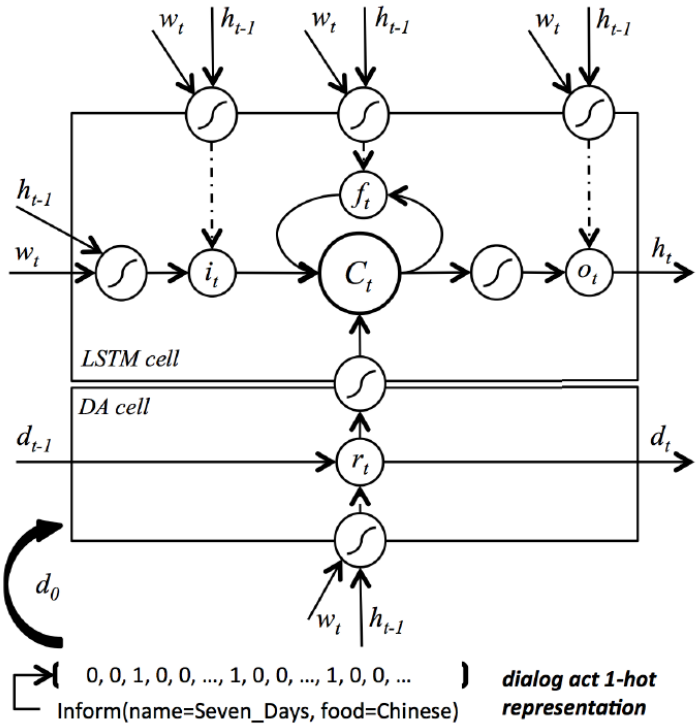


Figure 3.9: SC-LSTM architecture [169]

Wen *et al.* [169] propose to use a conditional LSTM [53] for natural language generation, the architecture of which is shown in Figure 3.9. The conditional vector $\mathbf{s}_n$ is a dialogue slot one-hot

vector, where each segment of $\mathbf{s}_n$ corresponds to a one-hot representation of an attribute slot in the system action $\tilde{Y}_n$, e.g., "⟨food⟩". An additional control gate $\mathbf{r}_t$ is introduced to control the masking vector of the conditional vector $\mathbf{s}_t$ at each time step t, instead of using a heuristic function. The model can be trained on unaligned datasets. The five control gates are calculated as

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{f}_t \\ \mathbf{r}_t \\ \hat{\mathbf{c}}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{y}_t \end{bmatrix},$$

where the speech-act conditional vector $\mathbf{s}_t$ for word t is defined as

$$\mathbf{s}_t = \mathbf{r}_t \odot \mathbf{s}_{t-1},$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t + \tanh(\mathbf{W}_{dc}\mathbf{s}_t),$$

and

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

where $\mathbf{W} \in \mathbb{R}^{5d*2d}$, d is the dimension of word embedding. $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t, \mathbf{r}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t$ are all vectors in $\mathbb{R}^d$. $\mathbf{y}_t$ is the embedding vector for word y_t.

To generate natural language responses based on the hidden state vector $\mathbf{h}_t$, the decoding process is defined as follows

$$p(y_t|\mathbf{h}_t) = \frac{\exp(f(\mathbf{h}_t, \mathbf{y}_t))}{\sum_{y'} \exp(f(\mathbf{h}_t, \mathbf{y}'))}$$

where

$$f(\mathbf{h}_t, \mathbf{y}_t) = \mathbf{h}_t^T \mathbf{y}_t,$$

here $f(\mathbf{h}_t, \mathbf{y}_t)$ denotes the association function between $\mathbf{h}_t$ and $\mathbf{y}$, the higher the association function, the more likely the word will be chosen under $\mathbf{h}_t$.

To train the whole generation network, the loss function is the cross entropy $\text{CE}(\hat{Y}, Y)$ between predicted word distribution $\hat{Y}$ and the actual response Y by

$$L(\Theta) = \text{CE}(\hat{Y}, Y) + ||\mathbf{s}_T|| + \sum_{t}^{T-1} \eta\xi^{||\mathbf{s}_{t+1} - \mathbf{s}_t||},$$

where $\mathbf{s}_T$ is the conditional vector at the last time step, and $\eta = 1e-4$ and $\xi = 100$ are constants. $||\mathbf{s}_T||$ penalises the loss function when the model failed to generate all the required slots, while $\sum_t^{T-1} \eta\xi^{||\mathbf{s}_{t+1} - \mathbf{s}_t||}$ discourages the model to turn off more than 1 gate in a single time step.

Neural network based methods such as RNNs can consider context of arbitrary length, and they can directly model the probability of generated words without the sentence planning and the surface realization. Neural network based methods can be trained in an end-to-end manner, without requiring the intermediate labelled data such as sentence planning tree label. However, a neural network has a large amount of parameters, and requires a large amount of data to train. And Neural network parameters can hardly be understood by human. As a result, human knowledge can hardly be incorporated into the neural network, and it is hard to debug.

Evaluation for Single Domain NLG

In this section we introduce some evaluation metrics and evaluation results of the surveyed natural language generation methods. Two popular datasets are the Tourist Information Dataset and the Restaurant in San Francisco Dataset. For evaluation metrics, the BLEU and the Slot Error Rate are the most frequently used metrics. The quantitative comparison results are shown in Table 3.8.

Table 3.8: Natural Language Generation Evaluation

Method	Dataset	BLEU	Slot Error Rate
Corpus-based ClassLM [101]	Tourist Info	0.06	
Corpus-based Bagel [86]	Tourist Info	0.37	
Corpus-based ClassLM [101]	Restaurant in SF	0.627	0.087
Neural Network Based RNNLM+CNN [164]	Restaurant in SF	0.710	0.015
Neural Network Based sc-LSTM [169]	Restaurant in SF	0.731	0.0046

Here we qualitatively compare different kinds of dialogue generation method.

The conventional dialogue generation consists of sentence planning and surface realization, and most of the system are hand-crafted. The intermediate representation has clear meanings and we can easily incorporate human knowledge into the system. However, it requires much corpus labelling and is expensive to build, and it has limited linguistic coverage. The hand-crafted components can hardly be reused in other tasks.

The corpus-based methods learn part of the generation decisions from the corpus, which can reduce the required human efforts and is less expensive. Statistical learning makes the corpus-based method prone to noise. However, the surveyed corpus-based model can only deal with a fixed length of context. As a result, it might generate incoherent sentences if no enough context is considered.

Neural network based methods such as the recursive neural network can consider context of arbitrary length, and it can directly model the probability of the generated words without the sentence

planning and surface realization. Neural network based methods can be trained in an end-to-end manner, without requiring intermediate labelled data such as the sentence planning tree labels. However, a neural network has a large amount of parameters, and requires a large amount of data to train. And the neural network parameters can hardly be understood by human. As a result, the human knowledge can hardly be incorporated into the neural network, and it is hard to debug.

3.5.3 Transfer Learning for Natural Language Generation

In this section, we review some papers on transfer learning for the natural language generation problem. Walker *et al.* [155] and Mairesse *et al.* [84, 85] propose to adapt general sentence planning models to different personal linguistic styles. Although these models can deal with many linguistic phenomenons, they are heavily dependent on the human knowledge and the hand-crafted rules. The RNN-based language models [123, 167, 168] are proposed to generate natural language. The RNN-based language models are more flexible and general, and they do not require human heuristics. We focus on transfer learning on the RNN-based language models.

There are mainly three types of transfer learning techniques in the RNN-based language models.

1. Model fine-tuning. A generate model is firstly trained on a general source domain dataset, and then the model is fine-tuned with instances from the target domain.

2. Curriculum learning. In each training epoch, the training instances are sorted such that the general instances are feed into the model first, and the specific target domain instances are trained in later of the epoch.

3. Instance synthesis. Synthetic target domain sentences are built from delexicalised source domain sentences by slot-value substituting.

Model fine-tuning Transfer for NLG

Wentoward *et al.* [167] and Wen *et al.* [170] propose to fine-tune an out-of-domain model with the in-domain data to achieve transfer learning. The base model is the sc-lstm [169] model. Firstly, the authors train an out-of-domain model with all source domain data. Secondly, the authors fine-tune the model parameters on various proportion of target domain data. Two baseline models are used, the first one is an encoder-decoder model and the other one is the deep sc-lstm model. The authors find that the sc-lstm model and deep sc-lstm model perform better than the encoder-decoder

model when the available target domain data is sufficient. However, when the target domain data is insufficient, the simpler encoder-decoder model performance better.

Curriculum learning Transfer for NLG

Shi *et al.* [123] propose to use curriculum learning [29] to adapt an RNNLM model. The authors propose 2 curriculum learning strategies. The first curriculum learning strategy is data-sorting, the idea of which is to use both data from the source domain and the target domain in each training epoch. The training data are sorted such that the model is firstly trained on plenty of source domain data and then trained on little target domain data. The second curriculum learning strategy is model fine-tuning. The idea is to train the whole model on the source domain first, and then fine-tune this model with the target domain data. The data-sorting strategy and the model fine-tuning strategy differ with respect to the point at which they move from the source domain data to the target domain data. In the data-sorting strategy, the model move from the source domain to the target domain at each training epoch. In the model fine-tuning strategy, the model is fully optimized with respect to the source domain, and then adapted to the target domain.

Instance Synthesis Transfer for NLG

Wen *et al.* [168] propose to combine the model fine-tuning and the synthetic data generation process for domain adaptation in natural language generation. The authors also propose to use the discriminative training. The paper is based on the sc-lstm model [169]. Firstly, the sc-lstm model is trained on the delexicalised source domain and fine-tuned in the delexicalised target domain. In this case, for sentences with new slot-values that appear only in the target domain, the model have to learn from scratch and no knowledge can be transferred. Secondly, some synthetic instances are

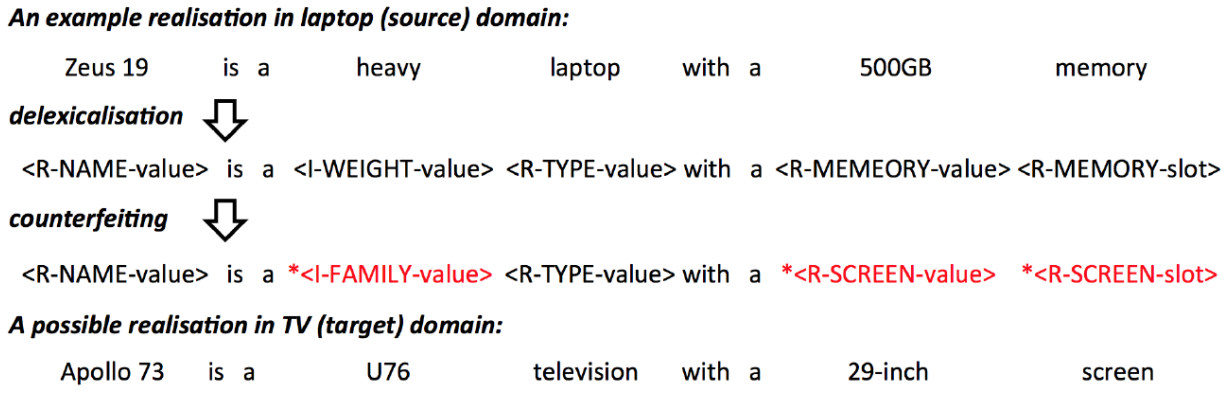


Figure 3.10: Data synthesis in [168]

built by adapting the source domain instances with new slot-values that appeared only in the target

domain. In detail, the slot-value tags in the source domain instances are substituted with similar new slot-value tags in the target domain, as shown in Figure 3.10. Then the synthetic data in the target domain can be used to train the model further, achieving knowledge transfer for new slot-values in the target domain. Note that in this data synthesis process, a similarity metric between the slot-value and target slot-value is required, and in this paper, the similarity is defined based on the slot type.

Model comparison for Transfer Learning in NLG

Transfer learning-based on model fine-tuning and curriculum learning differ with respect to the point at which they move from the source domain data to the target domain data, and both methods can be used to transfer low level language modelling knowledge. However these methods cannot benefit new slot-values that appear only in target domain. Instance synthesis transfer learning method can transfer language modelling knowledge to new slot-values in the target domain, assuming the expressions of different slot-values are similar.

3.6 End-to-end Dialogue System

In this section, we survey a special class of dialogue systems, called end-to-end task-oriented dialogue systems. Traditional modular dialogue systems require a large amount of hand-crafting, or a large amount of labelled data for each dialogue component.

Unlike the modular dialogue systems whose modules are trained or hand-crafted separately, the components in an end-to-end dialogue system can be trained together. An end-to-end dialogue systems have a single objective function. When trained jointly, the whole system can learn together and achieve a better performance, and it does not require intermediate annotations. End-to-end dialogue systems can reduce the amount of human labour required for building a dialogue system.

Unlike traditional dialogue management modules which first identify the current state and then decide the next action, in an end-to-end dialogue system, there is no unique definition of a ground-truth dialogue state. Instead in each time step, the whole dialogue system directly takes the current question as input and generates an output sentence, based on the the internal state. The internal state is updated at each time step, so the internal state represents the abstract dialogue state at each time step. The whole end-to-end dialogue system can be viewed as a deterministic policy function. The input of this policy function is the dialogue history and the current question, and the output of this policy function is the system answer. Unlike previous dialogue management modules, the state space and action space of end-to-end dialogue systems are the space of all possible sentences.

Based on the model used to build end-to-end task-oriented dialogue systems, we categorize all papers into 3 categories.

1. End-to-end training of separate modules. This method builds most of its dialogue components with trainable models, and all components are trained together. Some intermediate representations such as dialogue states and database outputs are still predefined by human.

2. Joint Dialogue State Tracking and Policy Learning. This method uses a LSTM module to jointly model the dialogue state tracking and policy learning process. In this case, the system action representation does not require hand-crafting.

3. Joint SLU, DST, DPL with Memory Network. A memory Network is used to build end-to-end dialogue system, which can track and record the dialogue state in the memory unit. All intermediate representations are based on embedding vectors, which are directly learned from the data.

3.6.1 Problem Definition

Given the dialogue historical including the current question, our goal is to predict the current response. Let us use X to denote the question, use Y to denote the response. A question is denoted by $X_n = \{x_1, x_2, \cdots, x_{N_n^x}\}$. A response is denoted by $Y_n = \{y_1, y_2, \cdots, y_{N_n^y}\}$. N is the number of dialogue rounds, N_n^x is the number of words in the n-th question, N_n^y is the number of words in the n-th response, where N_n^y is determined by the position of the "EOS". To make things consistent, the dialogue round is indexed by n and the word is indexed by t.

Given the dialogue history $H_n = \{\{X_j, Y_j\}_{j=1}^{n-1}, X_n\}$, the task of a task-oriented dialogue system is to predict Y_n.

1. Input: Dialogue history $H_n = \{\{X_j, Y_j\}_{j=1}^{n-1}, X_n\}$.
2. Output: Current system response Y_n.

3.6.2 End-to-end Dialogue System Structure

There are 5 components in the reviewed task-oriented dialogue systems.

1. Spoken Language Understanding (SLU). It takes raw text utterance X_n as an input and predicts the abstract user action $\tilde{X}_n$ of the utterance at every time step n.
2. Dialogue State Tracking (DST). It takes the abstract user action $\tilde{X}_n$ and the previous dialogue state $\tilde{H}_{n-1}$ as inputs and predicts the dialogue state $\tilde{H}_n$.
3. Database Operator (DB). It takes the dialogue state $\tilde{H}_n$ as an input and output the related database entries $\mathbf{J}_n$. All database operators reviewed in this section are hand-crafted and could not be trained.
4. Dialogue policy network (DPL). The policy network takes the abstract user action $\tilde{X}_n$, dialogue state $\tilde{H}_n$ and database entries $\mathbf{J}_n$ as the inputs, and predicts the system action $\tilde{Y}_n$.
5. Generation Network (NLG). It takes the system action $\tilde{Y}_n$ as an input, and generates the final system response sentence Y_n.

3.6.3 End-to-end training of Separate Modules

Wen *et al.* [166, 165] propose to train all components in the dialogue system in an end-to-end manner. All major components in the dialogue model can be trained via gradient-based methods. Here we introduce the components one by one.

Spoken Language Understanding

In [166], given a sequence of user input token $X_n = \{x_1, x_2, \cdots, x_{N_n^x}\}$, the SLU output is an abstract user action vector $\tilde{X}_n$ after parsing the entire X_n.

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{f}_t \\ \hat{\mathbf{c}}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where $\mathbf{W} \in \mathbb{R}^{4d*2d}$, d is the dimension of the word embedding. $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t$ are all vectors in $\mathbb{R}^d$. $\mathbf{x}_t$ is the embedding vector for word x_t. The abstract user action vector is calculated by

$$\tilde{X}_n = \mathbf{h}_{N_n^x}.$$

The above notation can be shorten as

$$\tilde{X}_n = \text{LSTM}(X_n).$$

Dialogue State Tracking

The Dialogue State Tracker takes the abstract user action $\tilde{X}_n$ and the dialogue history $\tilde{H}_{n-1}$ as inputs and predicts the dialogue state $\tilde{H}_n$, which is a set of slot-value pairs with their probabilities.

In [166], the DST maintains a multinomial distribution $p(v)$ over values $v \in V_s$ for each slot s and a binary distribution for each requestable slot.

To calculate the dialogue state $\tilde{H}_n$ for the n-th dialogue turn, we need to follow the steps.

1. Firstly, we need to calculate the CNN feature vector $\mathbf{f}_{s,v,\text{cnn}}^n$ for all s, v in the n-th dialogue turn.

2. Secondly, we update $p_{s,v}^n$, the probability of slot value v in slots s, for all slots and slot values, with $p_{s,v}^{n-1}$ and the CNN feature vector for the current turn.

3. Thirdly, we need to concatenate the $p_{s,v}^n$ and produce the final probability vector $\tilde{H}_n$.

The CNN feature vector $\mathbf{f}_{s,v,\text{cnn}}^n$ is a concatenation of the user side features at time n and the machine side features at time $n-1$, which can be denoted by

$$\mathbf{f}_{s,v,\text{cnn}}^n = [\text{CNN}_{s,v}^X(\tilde{X}_n), \text{CNN}_{s,v}^Y(\tilde{Y}_{n-1})],$$

where $[\mathbf{a_1}, \mathbf{a_2}]$ is the concatenation of two vectors.

To update $p_{s,v}^n$ with $p_{s,v}^{n-1}$ and the CNN feature vector $\mathbf{f}_{s,v,\text{cnn}}^n$, firstly we calculate the current feature vector $\mathbf{f}_{s,v}^n$ by

$$\mathbf{f}_{s,v}^n = [\mathbf{f}_{s,v,\text{cnn}}^n, p_{s,v}^{n-1}, p_{\emptyset}^{n-1}],$$

where $p_{s,v}^n$ is the probability of selecting slot value v and $p_{\emptyset}^n$ is the probability of not selecting any value in this slot s, up to time n.

Then the probability $p_{s,v}^n$ of each slot value v in slot s is updated by a belief tracker implemented by a Jordan-type RNN with a CNN feature extractor, denoted by

$$p_{s,v}^n = \frac{\exp(g_{s,v}^n)}{\exp(g_{\emptyset,s}) + \sum_{v' \in V_s} \exp(g_{s,v'}^n)},$$

where $g_{s,v}^n$ is defined by

$$g_{s,v}^n = \mathbf{w_s} \odot \text{sigmoid}(\mathbf{W}_s \mathbf{f}_{s,v}^n + \mathbf{b}_s) + b_s'.$$

The value v in the same slot s use the same RNN weight $\mathbf{W}_s$ but with a different slot-value-based feature vector $\mathbf{f}_{s,v}^n$.

The probability vector $\mathbf{p}_s^n$ for slot s until the n-th dialogue turn is the concatenation of probabilities of all values v in the slot s, denoted by

$$\mathbf{p}_s^n = \oplus_{v \in V_s} p_{s,v}^n.$$

The final dialogue state (the probability vector) $\tilde{H}_n$ is the concatenation of the probabilities vectors for all slot s, denoted by

$$\tilde{H}_n = \oplus_s \mathbf{p}_s^n.$$

Database Operator

The Database Operator takes the dialogue state $\tilde{H}_n$ as an input and outputs related database entries $\mathbf{J}_n$.

In [166], the query constraint-set q_n is calculated based on the output of DST by

$$q_n = \bigcup_{s' \in S_I} \arg\max_v p^n_{s',v},$$

where $p^n_{s,v}$ is the probability of the user choosing value v in slot s until the n-th dialogue turn, $\bigcup$ mean the set union and S_I is the set of slots that can be used to constraint the search (informable slots). Then the final database entries one-hot vector $\mathbf{J}_n$ is calculated by

$$\mathbf{J}_n = \text{SQL}(q_n).$$

The query constraint-set q_n is applied to the database to create a binary vector $\mathbf{J}_n$, where 1 indicate the entity matches all the constraints specified by the current user.

Dialogue Policy Network

The policy network takes the abstract user action $\tilde{X}_n$, the dialogue state $\tilde{H}_n$ and the database entry vector $\mathbf{J}_n$ as input, and predicts the system action $\tilde{Y}_n$.

In [166], the policy network is a deterministic policy function, which is modelled with a simple DNN on the summary belief state $\tilde{X}_n$, $\tilde{H}_n$, $\mathbf{J}_n$. The $\tilde{H}_n = \oplus_s \mathbf{p}^n_s$ is the concatenation of all belief state vectors. The mapping from the summary belief state to the abstract system action $\tilde{Y}_n$ is defined by

$$\tilde{Y}_n = \tanh(\mathbf{W}_{xo}\tilde{X}_n + \mathbf{W}_{ho}\tilde{H}_n + \mathbf{W}_{Jo}\mathbf{J}_n),$$

where $\mathbf{W}_{uo}$, $\mathbf{W}_{ho}$ and $\mathbf{W}_{Jo}$ are the parameters to learn.

Natural Language Generation

The input of the natural language generation network is the system action $\tilde{Y}_n$ produced by the policy network, and the output is the system response $Y_n = \{y_{n,1}, y_{n,2}, \cdots\}$.

For notation simplicity, in the following part of the Natural Language Generation we omit the index n and assume we are generating the n-th system reply, and we show how to generate $Y = \{y_1, y_2, \cdots\}$ with $\tilde{Y}$.

Language Model Type LSTM NLG In [166, 165], the NLG module is a LSTM [53] conditioned on the policy network output $\tilde{Y}_n$, we denote the generation process by

$$P(y_{t+1}|y_t, \mathbf{h}_{t-1}, \tilde{Y}) = \text{LSTM}(y_t, \mathbf{h}_{t-1}, \tilde{Y}),$$

where the input gate $\mathbf{i}_t$, the output gate $\mathbf{o}_t$, the forget gate $\mathbf{f}_t$ and the memory content $\hat{\mathbf{c}}_t$ are calculated by

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{f}_t \\ \hat{\mathbf{c}}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \tilde{Y} \\ \mathbf{h}_{t-1} \\ \mathbf{y}_t \end{bmatrix},$$

then the memory content $\mathbf{c}_t$ is updated as

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t,$$

and then the hidden state vector $\mathbf{h}_t$ is calculated by

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

where $\mathbf{W} \in \mathbb{R}^{4d*3d}$, d is the dimension of word embedding. $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t$ are all vectors in $\mathbb{R}^d$. $\mathbf{y}_t$ is the embedding vector for word y_t. Finally, y_{t+1} is generated like the traditional LSTM model.

Wen *et al.* [165] found that in an end-to-end dialogue system, the generation network faces competition between long-term dependency of expressing a correct logic and the short term dependency of generating the most fluent word. Such competition means, a model with higher BLEU score (has a better language model) will sacrifice the logical performance, thus it might lead to a lower dialogue success rate.

Memory Type LSTM NLG. Wen *et al.* [165] propose the Memory Type LSTM NLG. The main idea is that the speech-act conditioning vector should be isolated from the language model, so that the model has more flexibility to trade off between the speech-act conditioning vector and the language model. The authors propose to use another reading gate $\mathbf{r}_t$ to control the speech-act conditioning vector $\tilde{Y}$. The four gates of the Memory Type LSTM NLG are formulated as

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{f}_t \\ \mathbf{r}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \tilde{Y} \\ \mathbf{h}_{t-1} \\ \mathbf{y}_t \end{bmatrix}.$$

Then, the hidden state vector for the t-th word is generated by

$$\hat{\mathbf{c}} = \tanh(\mathbf{W}_c([\mathbf{y}_t, \mathbf{h}_{t-1}])),$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t + \mathbf{r}_t \odot \tilde{Y},$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

where $\mathbf{W} \in \mathbb{R}^{4d*3d}$, d is the dimension of word embedding. $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t$ are all vectors in $\mathbb{R}^d$. $\mathbf{y}_t$ is the embedding vector for word y_t. Then we can generate y_{t+1} like the conventional LSTM.

Hybrid Type LSTM NLG. Wen *et al.* [165] also propose the Hybrid Type LSTM NLG. The main idea is that the conditioning vector is isolated from the language model, so that the model has more flexibility to trade off between the conditioning vector and the language model. Another intuition is that the conditioning vector does not need long-term dependency, the decoupling of the conditioning vector and the language model could lead to better results. The four gates of the Hybrid Type LSTM NLG is formulated as

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{f}_t \\ \mathbf{r}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \tilde{Y} \\ \mathbf{h}_{t-1} \\ \mathbf{y}_t \end{bmatrix}.$$

Then the hidden state $\mathbf{h}_t$ is calculated by

$$\hat{\mathbf{c}} = \tanh(\mathbf{W}_c([\mathbf{y}_t, h_{t-1}])),$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t,$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) + \mathbf{r}_t \odot \tilde{Y},$$

where $\mathbf{W} \in \mathbb{R}^{4d*3d}$, d is the dimension of word embedding. $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t, \mathbf{h}_t$ are all vectors in $\mathbb{R}^d$. $\mathbf{y}_t$ is the embedding vector for word y_t.

LSTM Decoding Process To generate natural language response based on the hidden state vector $\mathbf{h}_t$, the decoding process is defined as

$$p(y_{t+1}|\mathbf{h}_t) = \frac{\exp(f(\mathbf{h}_t, \mathbf{y}_{t+1}))}{\sum_{y'} \exp(f(\mathbf{h}_t, \mathbf{y}'))},$$

where

$$f(\mathbf{h}_t, \mathbf{y}_t) = \mathbf{h}_t^T \mathbf{y}_t.$$

Here $f(\mathbf{h}_t, \mathbf{y})$ denotes the association function between $\mathbf{h}_t$ and the word embedding vector $\mathbf{y}$. The higher the association function, the more likely the word y will be chosen under $\mathbf{h}_t$.

The Language Model Type LSTM NLG is the most straight forward approach, but it suffers from the competition between the high level logic and the low level language model. For Memory

Type LSTM NLG and Hybrid Type LSTM NLG, the advantage is that the conditioning vector is isolated from the language model, alleviating the competition between the high level logic and the low level language model. The disadvantage is that such conditional vector has to be designed by human, which limits the generalization ability of the methods.

3.6.4 Joint DST and DPL

Williams *et al.* [177] propose to jointly model the dialogue state tracking module and the dialogue policy network with a LSTM, which can automatically infer a representation of the dialogue state. The input is a user utterance X_n, a dialogue state $\tilde{H}_n$ and a database entries $\mathbf{J}_n$, and the output is a selected answer Y_n from a set of candidate responses.

At each time step n, the LSTM policy network extracts a feature vector $\mathbf{f}_{X_n}$ from the summary belief state X_n, $\tilde{H}_n$, $\mathbf{J}_n$ by

$$\mathbf{f}_{X_n} = f(X_n, \tilde{H}_n, \mathbf{J}_n).$$

The feature vector $\mathbf{f}_{X_n}$ is used as an input at each time step n, then the LSTM updates its internal model and output a probability distribution on all possible system outputs by

$$\begin{bmatrix} \mathbf{i}_n \\ \mathbf{o}_n \\ \mathbf{f}_n \\ \hat{\mathbf{c}}_n \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{W} \begin{bmatrix} \mathbf{f}_{X_n} \\ \mathbf{h}_{n-1} \end{bmatrix}.$$

Then the hidden state $\mathbf{h}_n$ is calculated by

$$\mathbf{h}_n = \mathbf{o}_n \odot \tanh(\mathbf{c}_n),$$

where the $\mathbf{c}_n$ is updated as

$$\mathbf{c}_n = \mathbf{f}_n \odot \mathbf{c}_{n-1} + \mathbf{i}_n \odot \hat{\mathbf{c}}_n,$$

$\mathbf{W} \in \mathbb{R}^{4d*2d}$, d is the dimension of word embedding. $\mathbf{i}_n, \mathbf{o}_n, \mathbf{f}_n, \hat{\mathbf{c}}_n, \mathbf{c}_n, \mathbf{h}_n$ are all vectors in $\mathbb{R}^d$.

The system chooses a sentence from a candidate set based on the output of policy network via

$$p(Y_n|\mathbf{h}_n) = \frac{\exp(f(\mathbf{h}_{n-1}, \mathbf{y}_n))}{\sum_{\mathbf{y}'} \exp(f(\mathbf{h}_{n-1}, \mathbf{y}'))},$$

where the association function $f(\mathbf{h}_{n-1}, \mathbf{y}_n)$ is defined as

$$f(\mathbf{h}_{n-1}, \mathbf{y}_n) = \mathbf{h}_{n-1}^T \mathbf{y}_n,$$

and $\mathbf{y}$ is the embedding of the answer sentence Y. $f(\mathbf{h}_{n-1}, \mathbf{y})$ denotes the association function between $\mathbf{h}_{n-1}$ and Y, the larger the association function, the more likely the system output Y will be chosen under $\mathbf{h}_{t-1}$.

Using a LSTM to jointly model dialogue state tracking and the dialogue policy has some advantages. It can choose the optimal system action based on the learned dialogue state representation. However, training the LSTM requires many data.

3.6.5 Joint SLU, DST and DPL with Memory Network

Bordes *et al.* [17] propose to build a goal-oriented dialogue system with a Memory Network. The input is the raw text utterance X_n and dialogue history $H_n = \{\{X_j, Y_j\}_{j=1}^{n-1}, X_n\}$, and the output is the selected output Y_n.

Memory Network based end-to-end dialogue systems track and record the dialogue state in the memory unit of the memory network. The authors store the embedded conversation history (along with its time index, user/agent indicator) into the memory, then they use the last utterance X_{n-1} to be the input to the controller.

The memory is an array with the entries denoted by

$$\mathbf{M} = (\Phi(X_1), \Phi(Y_1), \cdots, \Phi(X_{n-1}), \Phi(Y_{n-1}))$$

where $\Phi()$ maps an utterance to a sentence embedding vector with length d.

They find salient memory slots by calculating the similarity scores p_i between $\mathbf{q}_0 = \Phi(X_n)$ and the memory content by

$$p_i = \text{softmax}(\mathbf{q}_j^T \mathbf{m}_i).$$

The selected memory $\mathbf{m}_i$ is transformed by a $d * d$ transform matrix $\mathbf{R}$ into an output denoted by

$$\mathbf{o}_j = \mathbf{R} \Sigma_i p_i \mathbf{m}_i.$$

The output $\mathbf{o}_h$ is added back to the controller state with an update function defined as

$$\mathbf{q}_{j+1} = \mathbf{o}_j + \mathbf{q}_j.$$

In the next iteration, $\mathbf{q}_{j+1}$ will be used instead of $\mathbf{q}_j$. They use $\mathbf{q}_j$ on iteration j, for a fixed number of iterations (3 or 4).

To rank all possible utterances, the final prediction score vector $\mathbf{y}$ is then defined as

$$\mathbf{y} = \text{softmax}(\mathbf{q}_h^T \mathbf{W} \Phi(Y_1), \cdots, \mathbf{q}_h^T \mathbf{W} \Phi(Y_C))$$

where C are number of candidate responses, and $\mathbf{W} \in \mathbb{R}^{d*d}$. The set of possible candidate responses includes all possible bot utterances and API calls. The whole model will then rank all possible utterances and then the top ranked candidate is chosen to be the answer via

$$Y_n = \arg\max_Y \mathbf{y}.$$

Using memory network to jointly model SLU, DST and DPL requires minimal human knowledge about the dialogue task, and it does not even require knowledge about possible slots. And it can be train in an end-to-end manner. However, it is very difficult to incorporate human knowledge into the model, and the internal results can hardly be explainable. The end-to-end training also requires more data and a careful optimization.

3.6.6 Evaluation for End-to-end dialogue system

To evaluate an end-to-end dialogue system, the system responses are compared with the human labelled ground truth, or tested with user simulator. The evaluation metrics include slot-match rate for spoken language understanding, BLEU for language generation, and task success rate for the dialogue policy. For systems that only select from a fixed set of response candidates, we can use the accuracy at turn level and the accuracy at dialogue level. Available datasets include the Restaurant in the Cambridge and the Phone dataset. We summarize the quantitative results reported in each of the paper in Table 3.9.

Table 3.9: End-to-end task-oriented dialogue system evaluation, based on Task Completion Rate (TCR), slot-match (Match), BLEU@top5 (BLEU-T5), BLEU@top1 (BLEU-T1), Accuarcy Turn level (Acc.T), Accuracy Dialogue level (Acc.D)

Method	Dataset	TCR	Slot-Match	BLEU-T5	BLEU-T1	Acc.T	Acc.D
Wen *et al.* [166]	Restaurant	0.8382	0.9088	0.2304	0.2369		
Wen *et al.* [165]	Restaurant	0.8180	0.6005	0.227	0.2400		
Williams *et al.* [177]	Phone	0.6900				0.9200	0.4800
Bordes *et al.* [17]	Restaurant					0.9340	0.1970

We summarize the reviewed end-to-end task-oriented dialogue systems in Table 3.10. The training methods are summarized in Table 3.11.

Modularized end-to-end trainable systems can easily incorporate human knowledge, and each module can be trained separately as initialization and then fine-tuned with reinforcement learning. However, if we do not have any knowledge about the intermediate results, it is very difficult to design a modularized end-to-end dialogue system.

Table 3.10: Model comparison with respect to Spoken Language Understanding (SLU), Dialogue State Tracking (DST), Dialogue Policy Learning (DPL), Natural Language Generation (NLG) and Database Operation (DB).

Method	SLU	DST	DPL	NLG	DB
Wen *et al.* [166]	LSTM/CNN	RNN+CNN	DNN	LSTM+Att.	Fixed
Wen *et al.* [165]	LSTM/CNN	RNN+CNN	DNN	LSTM+Att.+Mem.	Fixed
Williams *et al.* [177]	Fixed	LSTM		Template	Fixed
Bordes *et al.* [17]	MemNN			Template	Fixed

Table 3.11: Training method comparison in Spoken Language Understanding (SLU), Dialogue State Tracking (DST), Dialogue Policy (Policy), Natural Language Generation (NLG) and Database Operation (DB).

Method	SLU	DST	DPL	NLG	DB
Wen *et al.* [166]	SL-Joint	SL-Separate	SL-Joint	SL-Joint	Fixed
Wen *et al.* [165]	SL-Joint	SL-Separate	SL-Joint	SL(Snapshot)-Joint	Fixed
Williams *et al.* [177]	Fixed	SL+RL		Fixed	Fixed
Bordes *et al.* [17]	SL			Fixed	Fixed

Jointly modelling the dialogue system with a Memory Network [131] requires less human knowledge, and we do not even need to know about the possible slots and its value, and the system can directly learn important features from the training example. However, it is very hard to incorporate human knowledge in this framework. Also, the intermediate results can hardly be explained, and training the system requires much training data.

3.7 Conclusion and Future Directions

In conclusion, we give an organized review of the popular learning methods used in the task-oriented dialogue systems. In each section, we further categorize the algorithms and we give an qualitative and quantitative evaluation of the popular methods.

3.7.1 Task and Techniques Summary Table

The summary of tasks and techniques are listed in Table 3.12, and we also summarize the surveyed papers according to area and techniques in Figure 3.11, where the stars represent the areas we are interested in.

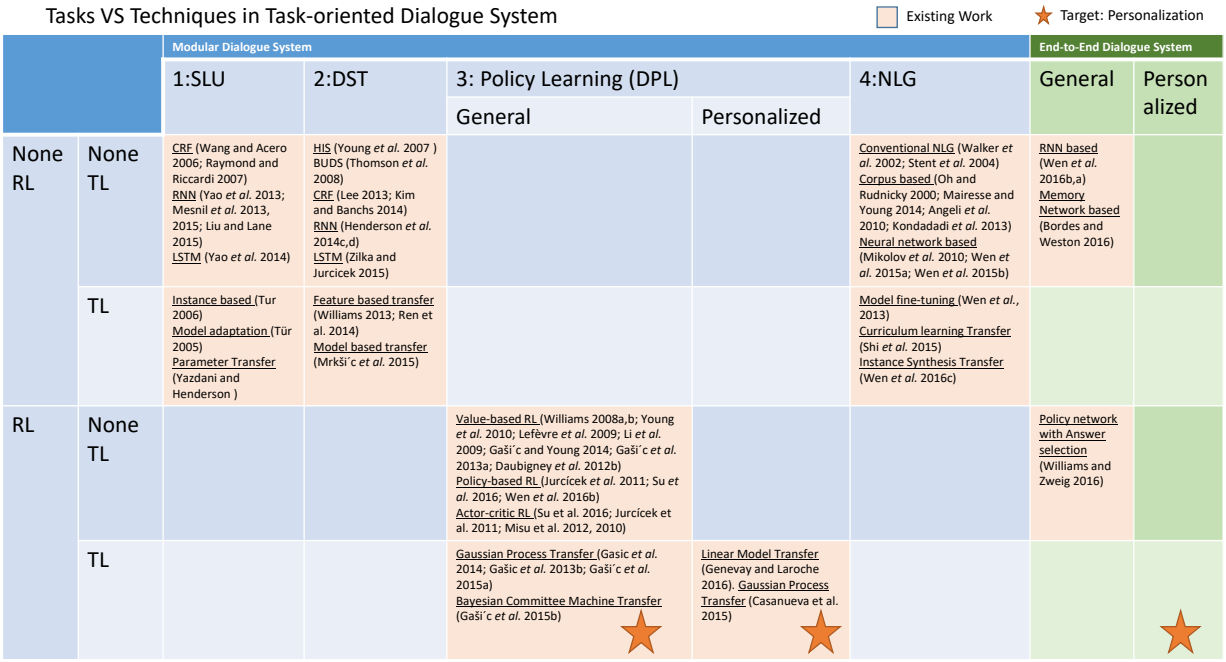


Figure 3.11: The Tasks VS Techniques Table of surveyed papers in each research area. Stars represent the areas we are interested in.

3.7.2 Dataset and Evaluation Metrics Table

The summary of datasets, tasks and evaluation metrics are listed in Table 3.13.

Table 3.12: The Model VS Task VS Formulation Table, summarized from the surveyed papers.

Models	Task	Formulation
Grid	SLU,DST,DPL	$Q(H,Y) = \text{Table}(f(H), f(Y))$ where $f()$ maps to a cluster index.
Linear	SLU,DST,DPL	$y = \mathbf{w}^T\mathbf{x}$.
Gaussian Process	DPL	$Q^\pi(H,Y) \sim \mathcal{GP}(m(H,Y), k((H,Y),(H,Y)))$.
MLP	SLU,DST,DPL	$y_0 = x$, $y_n = \sigma(\mathbf{W}y_{n-1})$.
CRF	SLU,DST	$P(s_t\|x_t, s_{t-1}) = \frac{1}{Z}exp(p(s_t\|s_{t-1}) + p(s_t\|x_t))$.
RNN	SLU,DST,NLG	$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}_h)$, $p(y_t\|x_t, x_{<t}) = \frac{1}{Z(\mathbf{h}_t)}\sigma(\mathbf{w}_{y_t}^T\mathbf{h}_t + b_{y_t})$.
LSTM	SLU,DST,NLG	$\begin{bmatrix}\mathbf{i}_t\\ \mathbf{o}_t\\ \mathbf{f}_t\\ \hat{\mathbf{c}}_t\end{bmatrix} = \begin{bmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{bmatrix}\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}$, $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t$, $\mathbf{h}_t = \mathbf{o}_t \odot tanh(\mathbf{c}_t)$, $p(y_t\|x_t, x_{<t}) = \frac{1}{Z(\mathbf{h}_t)}\sigma(\mathbf{w}_{y_t}^T\mathbf{h}_t + b_{y_t})$.
MemoryNN	SLU+DST+DPL	$\mathbf{M} = (\Phi(X_1), \Phi(Y_1), \cdots, \Phi(X_{n-1}), \Phi(Y_{n-1}))$, $\mathbf{q}_0 = \Phi(X_n)$, $p_i = \text{softmax}(\mathbf{q}_j^T\mathbf{m}_i)$, $\mathbf{o}_j = \mathbf{R}\Sigma_i p_i\mathbf{m}_i$, $\mathbf{q}_{j+1} = \mathbf{o}_j + \mathbf{q}_j$, $\mathbf{y} = \text{softmax}(\mathbf{q}_h^T\mathbf{W}\Phi(Y_1), \cdots, \mathbf{q}_h^T\mathbf{W}\Phi(Y_C))$, where $\Phi()$ outputs a sentence embedding vector, j is the index of hops in memory.

3.7.3 Future Directions

There are still many unsolved challenges in the task-oriented dialogue system. Current works still require many hand-crafting, for example we have to define the number of slots, the possible values in each slots, etc. How to design a system that requires less human knowledge is an important problem.

Another challenge is the knowledge transfer problem in task-oriented dialogue system. How to transfer the knowledge learned in one source domain to another domain is still an open question. Current works focus on the instance transfer and the cross-domain similarity calculation. However, these methods do not answer the question what knowledge is transferable and what knowledge is not suitable for transfer.

Dialogue personalization is another promising direction that needs exploration. In current task-oriented dialogue systems, the system do not model the unique preference and hobby of a customer.

Table 3.13: The Dataset VS Evaluation Metrics Table, summarized from the surveyed papers.

Dataset	Task	Evaluation Metrics
ATIS [23]	SLU	Accuracy, F1
TouristInfo [16]	SLU, NLG	Accuracy, F1, BLEU
DARPA Communicator [153]	SLU	Accuracy, F1
DSTC 1 [172]	DST	Accuracy, L2
DSTC 2 [47]	DST	Accuracy, L2
DSTC 3 [48]	DST	Accuracy, L2
TownInfo [144]	DST, DPL	Accuracy, L2, Reward, Success Rate, #Dialog Turns
Cambridge Restaurant [41]	DPL, NLG	Reward, Success Rate, #Dialog Turns, BLEU
SF Restaurant [36]	DPL, NLG	Reward, Success Rate, #Dialog Turns, BLEU
SF Hotel [36]	DPL, NLG	Reward, Success Rate, #Dialog Turns, BLEU
L11 [36]	DPL, NLG	Reward, Success Rate, #Dialog Turns, BLEU

Every customer has to specify every slots in every purchase, which is far from convenient. Talking to a dialogue system that remembers your preferences and knows you well might save the customer a lot of time and improve the user experience. How to use personalized information from one domain to help dialogues in another domain is also an interesting problem.

Current systems are mostly trained with user simulators, on predefined fix set of slots. How to design dialogue systems that can learn to extended their scopes and learn directly from user in the on-line environment is still an open problem.

CHAPTER 4

TRANSFER REINFORCEMENT LEARNING THROUGH PERSONALIZED Q-FUNCTION

It is difficult to train a personalized task-oriented dialogue system because the data collected from each individual is often insufficient. Personalized dialogue systems trained on a small dataset is likely to overfit and make it difficult to adapt to different user needs. One way to solve this problem is to consider a collection of multiple users as a source domain and an individual user as a target domain, and to perform transfer learning from the source domain to the target domain. By following this idea, we propose the PErsonalized Task-oriented diALogue (PETAL) system, a transfer reinforcement learning framework based on POMDP, to construct a personalized dialogue system. The PETAL system first learns common dialogue knowledge from the source domain and then adapts this knowledge to the target domain. The proposed PETAL system can avoid the negative transfer problem by considering differences between the source and target users in a personalized Q-function. Experimental results on a real-world coffee-shopping data and simulation data show that the proposed PETAL system can learn optimal policies for different users, and thus effectively improve the dialogue quality under the personalized setting.

4.1 Introduction

Personalized task-oriented dialogue systems aim to help a user complete a dialogue task better and faster than non-personalized dialogue systems. Personalized dialogue systems can learn about the preferences and habits of a user during interactions with the user, and then utilize this personalized information to speed up the conversation process. Personalized dialogue systems could be categorized into rule-based dialogue systems [142, 60, 8] and learning-based dialogue systems [20, 42]. In rule-based personalized dialogue systems, the dialogue state, system speech-act and user speech-act are predefined by developers, hence it is difficult to reuse this system when the dialogue state and the speech-act are hard to define manually. Learning-based personalized dialogue systems could learn states and actions from training data without requiring explicit rules designed by developers.

However, it is difficult to train a personalized task-oriented dialogue system because the data collected from each individual is often insufficient. A personalized dialogue system trained on a

small dataset is likely to fail on unseen but common dialogues due to the overfitting. One solution is to consider a collection of multiple users as a source domain and an individual user as a target domain and transfer common dialogue knowledge from the source domain to the target domain. When transferring dialogue knowledge, the challenge lies in the difference between the source and target domains. Some works [20, 42] have been proposed to transfer dialogue knowledge among similar users, but they did not model the difference among users, which might harm the performance in the target domain.

In this chapter, we propose a PErsonalized Task-oriented diALogue (PETAL) system, which is a transfer learning framework based on the POMDP for learning a personalized dialogue system. The PETAL system first learns common dialogue knowledge from the source domain and then adapts this knowledge to the target user. To achieve this goal, the PETAL system models personalized policies with a personalized Q-function defined as the expected cumulative general reward plus the expected cumulative personal reward. The personalized Q-function can model differences between the source and target users and thus can avoid the negative transfer problem brought by the differences. Experimental results on a real-world coffee-shopping dataset and simulation data show that the proposed PETAL system can choose optimal actions for different users and thus can effectively improve the dialogue quality under the personalized setting.

Our contributions are three-fold. Firstly, we tackle the problem of learning common dialogue knowledge from the source domain and adapting to the target user in a personalized dialogue system. In multi-turn dialogue systems, learning optimal responses in different situations is a non-trivial problem. One naive policy is to always choose previously seen sentences, but it is not necessarily optimal. For example, in the online coffee ordering task, such naive policy could incur many logical mistakes such as asking repeated questions and confirming the order before the user finishes ordering. Secondly, we propose a transfer learning framework based on the POMDP to model the preferences of different users. Unlike existing methods, the proposed PETAL system does not require a manually-defined ground-truth state space and it can model the personalized future expected reward. Finally, we demonstrate the effectiveness of the PETAL system on a real-world dialogue dataset as well as simulation data.

4.1.1 Challenges

In order to transfer common knowledge across different users in task-oriented dialogues systems, we have the following challenges.

1. There is no ground-truth state space for each task-oriented dialogue system, and there is no ground-truth dialogue state annotation for each dialogue sentence. Designing such dialogue state space and doing data annotation require a lot of efforts from the human experts. We have to build a model that can learn the underlying dialogue state from the dialogue data, despite great noise in the data.

2. Different users have different preferences, so the dialogue data for different user have different data distributions. Naively transferring all dialogue data across different users might lead to the leakage of personal information and have a negative impact on the performance of the dialogue policy. We have to design a model that can transfer common dialogue knowledge and avoid the transfer of personal information at the same time.

4.1.2 Related Works of Transfer Learning in Personalized Task-oriented Dialogue Systems

Personalized dialogue systems could be categorized into rule-based dialogue systems and learning-based dialogue systems. For rule-based personalized dialogue systems, Thompson *et al.* [142] propose an interactive system where users can choose a place via an interactive conversational process and the system could learn user preferences to improve future conversations. Personalization frameworks proposed in [60, 8] extract and utilize user-related facts (triples), and then generate responses by applying predefined templates to these facts. Different from rule-based personalized dialogue systems, learning-based personalized dialogue systems can learn states and actions from training data without requiring explicit rules. Casanueva *et al.* [20] propose to initialize personalized dialogue systems for a target user with data from similar users in the source domain to improve the performance for the target user. This work requires a predefined user similarity metric to select similar source users, and when the selected similar users are different from the target user, the performance for the target user will degrade. Genevay and Laroche [42] propose to select and transfer an optimized policy from source users to a target user by using a multi-armed stochastic bandit algorithm which does not require a predefined user similarity measure. However, this method has a high complexity since for each target user, it requires n^2 bandit selection operations where n is the number of source users. Moreover, similar to [20], the differences between selected source users and the target user will deteriorate the performance. Different from these works, the proposed method does not assume the predefined dialogue states and system speech-acts required by the rule-based systems, and it explicitly models the differences between users.

Transfer learning [141, 102, 139, 138, 163] has been applied to other tasks in dialogue systems.

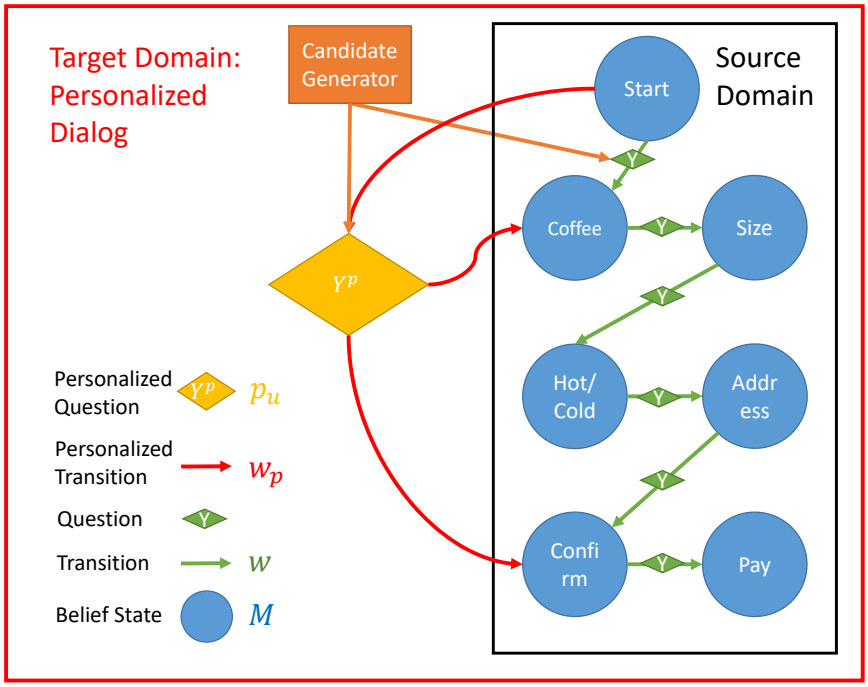


Figure 4.1: The flowchart of the proposed PETAL system on the coffee-ordering task.

Gasic *et al.* [37] use transfer learning to extend a dialogue system to include a previously unseen concept. Gasic *et al.* [38] propose an incremental scheme to adapt an existing dialogue management system to an extended domain. These two works transfer parameters in the policy of the source domain as a prior to the target domain. However, these two models do not deal with multiple source domains and they do not have explicit personalized mechanisms for different users. As a consequence, the negative transfer might occur when the differences between users are large. In contrast, the proposed method has an explicit personalization mechanism and can alleviate negative transfer.

In argumentation agents, there are some works [52, 116, 115] which study personalized dialogue systems. However, these works, which aim to influence users' goal, have different motivations from ours and their formulations are totally different from ours.

4.2 Model: Transfer Learning Framework for Personalized Dialogue Management (PETAL)

In this section, we introduce the proposed PETAL system. Figure 4.1 shows the flowchart of the PETAL system on a coffee-ordering task. Here we use PETAL to denote both the proposed framework and the proposed algorithm.

Matrices are denoted in bold capital case, row vectors are in the bold lower case, and scalars are in lower case. The text in the dialogues, denoted in curlicue, is represented by the the bag-of-words assumption. Each of the bag-of-words representations is a vector in which each entry has a binary value.

4.2.1 Problem Setting

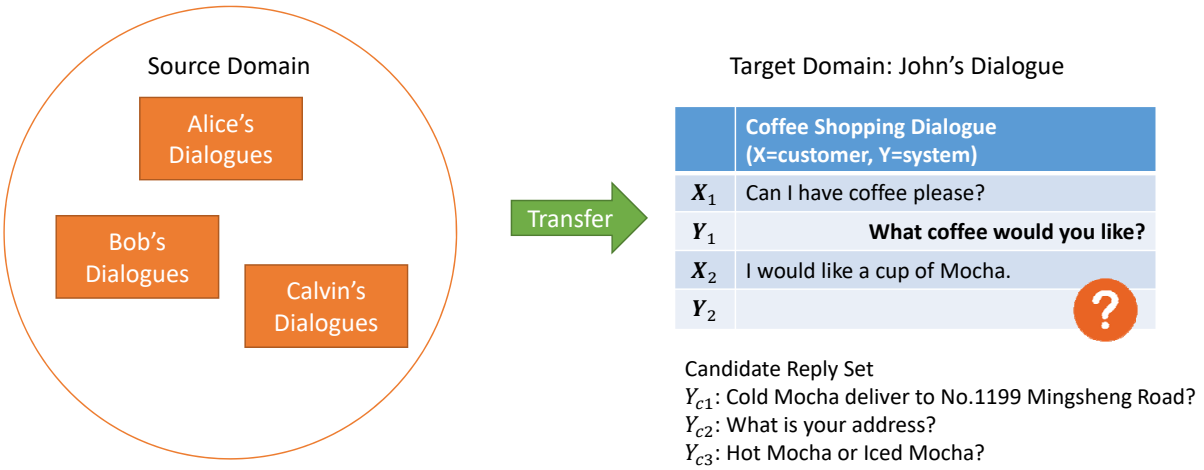


Figure 4.2: The problem illustration

The problem is illustrated in Figure 4.2. In a multi-turn dialogue system, the feedback is usually delayed, so it is more natural to formulate the problem with reinforcement learning. Since the current state of the dialogue is not observable and the ground-truth dialogue states are assumed to be unknown, we formulate the dialogue as a POMDP, $\tilde{H}$ denotes the ground-truth hidden unobservable states, Y denotes the replies of the agent, X denotes users' utterances, R is the reward function, and $\gamma \in [0, 1]$ is the discounted factor. In the n-th turn of a dialogue with a user u, $\tilde{H}_n^u$ is the hidden conversation state, X_n^u is the user utterance, Y_n^u is the reply of the agent, and r_n^u is the reward. In the n-th turn, we only observe X_n^u, Y_n^u and r_n^u. We define $\mathbf{b}_n^u$ as the belief state vector, which represents the probability distribution of unobserved $\tilde{H}_n^u$. Unlike previous works, we do not assume that the underlying ground-truth state space $\tilde{H}$ is provided. Instead we propose to learn a function to map the dialogue history $H_n^u = \{\{X_k^u, Y_k^u\}_{k=0}^{n-1}, X_n^u\}$ to a compact belief state vector $\mathbf{b}_n^u$.

The inputs for this problem include

1. Abundant dialogue data $\{\{X_n^{u_s}, Y_n^{u_s}\}_{n=0}^{T}\}$ of source customers $\{u_s\}$.

2. A few dialogue data $\{\{X_n^{u_t}, Y_n^{u_t}\}_{n=0}^{T}\}$ of the target customer u_t.

The expected output is

1. A policy π^{u_t} for target user.

4.2.2 The Framework

In order to solve the problem, we aim to find a policy π^{u_t} for the target user, which could choose an appropriate action $Y_n^{u_t}$ at the n-th turn based on current dialogue history $\mathcal{H}_n^{u_t}$, to maximize the cumulative reward defined as

$$\pi^{u_t} = \arg\max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}^{u_t}\right]. \tag{4.1}$$

To model belief states, we introduce a state projection matrix $\mathbf{M}$ to map the dialogue history H_n^u to the belief state $\mathbf{b}_n^u$, i.e., $\mathbf{b}_n^u = f(H_n^u; \mathbf{M})$.

The Q-function is defined as the expected cumulative reward according to policy π^u by starting from belief state $\mathbf{b}_n^u$ and taking action Y_n^u as

$$Q^{\pi^u}(H_n^u, Y_n^u) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}^u | H_n^u, Y_n^u\right]. \tag{4.2}$$

We choose value-based approaches because there is usually a small number of training data in the target domain, while policy-based approaches, which generate responses word by word, require a lot of training data.

In order to build a personalized dialogue system for the target user, we need to learn a personalized Q-function $Q^{\pi^{u_t}}$ for this user. However, since the training data $\{\{X_n^{u_t}, Y_n^{u_t}\}_n^T\}$ for the target user u_t is very limited, we can hardly estimate the personalized Q-function $Q^{\pi^{u_t}}$ accurately. In order to learn an accurate $Q^{\pi^{u_t}}$, we can transfer common dialogue knowledge from the source domain, which has a lot of data from many other users $\{\{X_n^{u_s}, Y_n^{u_s}\}_{n=0}^T\}$. However, different users may have different preferences, hence directly using the data from source users would bring negative effects. We propose to model the personalized Q-function as a general Q-function Q_g plus a personal Q-function Q_p:

$$Q^{\pi^u}(H_n^u, Y_n^u) = Q_g(H_n^u, Y_n^u; \mathbf{w}) + Q_p(H_n^u, Y_n^u; \mathbf{p}_u, w_p)$$

$$\approx \mathbb{E}_{\pi^u}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}^{u,g} | H_n^u, Y_n^u\right] + \mathbb{E}_{\pi_u}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}^{u,p} | H_n^u, Y_n^u\right], \tag{4.3}$$

where $r_t^{u,g}$ and $r_t^{u,p}$ denotes the general and personal rewards for user u at time t respectively, the general Q-function $Q_g(H_n^u, Y_n^u; \mathbf{w})$ captures the expected reward related to the general dialogue policy for all users, $\mathbf{w}$ is the set of parameters for the general Q-function and contains a large amount of parameters such that it requires a lot of training data, and the personal Q-function $Q_p(H_n^u, Y_n^u; \mathbf{p}_u, w_p)$ captures the expected reward related to the preference of each user.

The proposed framework is based on transfer learning. $\mathbf{M}$, $\mathbf{w}$ and w_p are shared across different users, which could be trained on source domains and then transferred to the target domain. These parameters contain the common dialogue knowledge, which is independent of users' preferences. Moreover, $\mathbf{p}_u$, which is user-specific, capture the preferences of different users.

4.2.3 Parametric Forms for Personalized Q-function

In this section, we introduce parametric forms for $f(H_n^u; \mathbf{M})$, $Q_g(H_n^u, Y_n^u; \mathbf{w})$ and $Q_p(H_n^u, Y_n^u; \mathbf{p}_u, w_p)$ in the personalized Q-function.

Dialogue states are defined as follows. All utterances and replies will be projected into state vectors with a state projection matrix $\mathbf{M}$, where $\mathbf{M}$ is initialized with the word2vec and will be updated in the learning process. The belief state vector $\mathbf{b}_n^u$ is denoted as

$$\mathbf{b}_n^u = f(H_n^u; \mathbf{M}), \tag{4.4}$$

which is mapped from the dialogue history

$$H_n^u = \{\{X_k^u, Y_k^u\}_{k=0}^{n-1}, X_n^u\}. \tag{4.5}$$

The belief state vector $\mathbf{b}_n^u$ consists of four parts, denoted by

$$\mathbf{b}_n^u = \left[\mathbf{x}_{n-1}^{h,u}, \mathbf{x}_n^u, \mathbf{y}_{n-2}^{h,u}, \mathbf{y}_{n-1}^u\right]. \tag{4.6}$$

In the four parts, all previous user utterances $\mathbf{x}_n^{h,u}$ is defined as

$$\mathbf{x}_n^{h,u} = \sum_{k=0}^{n} \xi^{n-k} \mathbf{x}_k^u, \tag{4.7}$$

the current user utterance $\mathbf{x}_n^u$ is defined as

$$\mathbf{x}_n^u = X_n^u \mathbf{M}, \tag{4.8}$$

all previous agent replies $\mathbf{y}_n^{h,u}$ is defined as

$$\mathbf{y}_n^{h,u} = \sum_{k=0}^{n} \xi^{n-k} \mathbf{y}_k^u, \tag{4.9}$$

and the last agent reply $\mathbf{y}^u_{n-1}$ is define as

$$\mathbf{y}^u_{n-1} = Y^u_{n-1}\mathbf{M} \tag{4.10}$$

where $\xi = 0.8$ is the memory factor to discount historical state vectors at each time step. In coffee ordering, the overlapping vocabulary of users and agents are limited to nouns which have consistent meaning, so it is more data efficient to learn one single $\mathbf{M}$ matrix instead of using different matrices for users and agents.

In order to model the correlations between entries in $\mathbf{y}^u_n$ and $\mathbf{b}^u_n$, the general Q-function $Q_g(H^u_n, Y^u_n; \mathbf{w})$ is defined as

$$Q_g(H^u_n, Y^u_n; \mathbf{w}) = \mathbf{y}^u_n \mathbf{W} (\mathbf{b}^u_n)^T, \tag{4.11}$$

where superscript ^{T} denotes the transpose of a vector or matrix, $\mathbf{W} \in \mathbb{R}^{d \times 4d}$ is a parameter matrix to be learned. Based on the properties of the Kronecker product and operator $\mathrm{vec}(\cdot)$ which transforms a matrix to a vector in a column-wise manner, we can rewrite $Q_g(H^u_n, Y^u_n; \mathbf{w})$ as a linear function on $\mathbf{w} = \mathrm{vec}(\mathbf{W})^T \in \mathbb{R}^{4d^2}$:

$$Q_g(H^u_n, Y^u_n; \mathbf{w}) = (\mathbf{b}^u_n \otimes \mathbf{y}^u_n)\mathbf{w}^T, \tag{4.12}$$

where $\mathbf{b}^u_n \otimes \mathbf{y}^u_n$ is the Kronecker product of $\mathbf{b}^u_n$ and $\mathbf{y}^u_n$. In multi-turn dialogue systems, there should be different optimal actions in different belief states. The rationale to use the Kronecker product is that the general Q-function should depend on the combination of belief state $\mathbf{b}^u_n$ and action $\mathbf{y}^u_n$, but not independently on $\mathbf{b}^u_n$ and $\mathbf{y}^u_n$. The rationale of using linear function in the general Q-function is, since the belief vector consists of a set of sentence embedding vectors, which are calculated from a set of word embedding vectors, so it is equivalent to a linear function with belief embedding matrix as a parameter. The general Q-function can be viewed as a dot product to measure the semantic matching of two vectors, the belief state vector $\mathbf{b}^u_n$ and the translated response sentence embedding $\mathbf{y}^u_n\mathbf{W}$. This can be viewed as a linear function on the Kronecker product of $\mathbf{b}^u_n$ and $\mathbf{y}^u_n$.

The personal Q-function learns personalized preference for each user to avoid the negative effect brought by transferring biased dialogue knowledge across users with different preferences. We denote by V_j the set of all possible choices in the j-th choice set we want to collect and by $\{v^u_{nj}\}^m_{j=1}$ the choices proposed in the n-th agent response Y^u_n, where m is the total number of order choices, hence v^u_{nj} is an exact choice in V_j. For example, in the coffee-ordering task, $V_1 = \{$Latte, Cappuccino,$\ldots\}$ could be the type of coffees and v^u_{n1} could be any coffee in V_1. From the user side, v^u_{nj} is just the choice of user u for the j-th choice set in the n-th dialogue turn. For example, v^u_{n1} could be "latte" and v^u_{n2} could be "iced". Based on an assumption that different choice sets are independent of each other, for the j-th choice set V_j, the probability of a user u to choose v^u_{nj}

follows a categorical distribution denoted as

$$\mathcal{C}(v_{nj}^u; \mathbf{p}_j^u) = p_{j,v_{nj}^u}^u \tag{4.13}$$

where $|V_j|$ denotes the cardinality of a set, $\mathbf{p}_j^u \in \mathbb{R}^{|V_j|}$, and $p_{j,k}^u$ denotes the k-th entry in $\mathbf{p}_j^u$. Hence the personal Q-function for user u is formulated as

$$Q_p(H_n^u, Y_n^u; \mathbf{p}_u, w_p) = w_p \sum_{j=1}^{m} \mathcal{C}(v_{nj}^u; \mathbf{p}_{uj}) \delta(V_j, H_n^u), \tag{4.14}$$

where the personal preference $\mathbf{p}_u = \{\mathbf{p}_{uj}\}_{j=1}^m$ for user u is learned from the training data of that user, $\delta(V_j, H_n^u)$ equals 1 if the user has not yet made a choice about V_j in the dialogue history H_n^u and 0 otherwise. $\delta(V_j, H_n^u)$ implies whether the system will receive a personal reward in the rest of the dialogue, as the Q-function models the cumulative future reward. Here w_p controls the importance of the personalized reward and it is learned from data. When w_p is close to zero, the personalized Q-function will depend on the general dialogue policy. Note that $\sum_{j=1}^m \mathcal{C}(v_{nj}^u | \mathbf{p}_{uj}) \delta(V_j, H_n^u)$ is 0 if we know nothing about the user, or Y_n^u does not show any personal preference of user u. Because the vocabulary of choices is much smaller than the whole vocabulary, we can estimate the personal preference parameters $\mathbf{p}_u$ with a few dialogue data $\{\{X_n^{u_t}, Y_n^{u_t}\}_n^T\}$ from the target user.

By combining the general and personal Q-functions, the personalized Q-function can finally be defined as

$$Q^{\pi_u}(H_n^u, Y_n^u) = (\mathbf{b}_n^u \otimes \mathbf{y}_n^u)\mathbf{w}^T + w_p \sum_{j=1}^{m} \mathcal{C}(v_{nj}^u | \mathbf{p}_{uj}) \delta(V_j, H_n^u). \tag{4.15}$$

Here $\mathbf{M}, \mathbf{w}, w_p$ are shared across different users, which could be trained on the source domains and then transferred to the target domain.

4.2.4 Reward

The total reward is the sum of general reward and personal reward, which can be defined as follows:

1. A personal reward $r^{u,p}$ of 0.3 will be received when the user confirms the suggestion of the agent and a negative reward of -0.2 will be received if the user declines the suggestion by the agent. This is related to the personal information of the user. For example, the user could confirm the address suggested by the agent.

2. A general reward $r^{u,g}$ of 0.1 will be received when the user provides the information about each c_j.

3. A general reward $r^{u,g}$ of 1.0 will be received when the user proceeds with payment.

4. A general reward $r^{u,g}$ of -0.05 will be received by the agent for each dialogue turn to encourage shorter dialogue, -0.2 will be received by the agent if it generates non-logical responses such as asking repeated questions.

Note that the personal reward could not be distinguished from the general reward during the training process. The reward function is designed to encourage the agent to actively make personalized suggestion when the agent can make correct guess with probability (denoted as p) higher than 0.6. The expected reward is $0.3p-0.2(1-p)$ for making suggestion and 0.1 for asking general question. When $p > 0.6$, $0.3p - 0.2(1 - p)$ is larger than 0.1.

4.2.5 Loss Function and Parameter Learning

There are in total four sets of parameters to be learned. We denote all the parameters by $\Theta = \{\mathbf{M}, \mathbf{w}, w_p, \{\mathbf{p}_u\}\}$. When dealing with real-world data, the training set consists of (H_n^u, Y_n^u, r_n^u), which records optimal actions provided by human, and hence the loss function is defined as follows:

$$\mathcal{L}(\Theta) = \mathbb{E}\left[\left(r_n^u + \max_{Y'_{n+1}} \gamma Q(H_{n+1}^u, Y'_{n+1}|\Theta) - Q(H_n^u, Y_n^u|\Theta)\right)^2\right], \tag{4.16}$$

In the on-policy training with a user simulator, the loss function is defined as

$$\mathcal{L}(\Theta) = \mathbb{E}\left[\left(r_n^u + \gamma Q(H_{n+1}^u, Y_{n+1}^u|\Theta) - Q(H_n^u, Y_n^u|\Theta)\right)^2\right]. \tag{4.17}$$

where r_n^u is the reward obtained at time step n and H_{n+1}^u is the update dialogue history at time step $n + 1$.

We use the value iteration method [12] to learn both the general and personal Q-functions. We adopt an online stochastic gradient descent algorithm [18] with learning rate 0.0001 to optimize our model. Specifically, we use the State-Action-Reward-State-Action (SARSA) algorithm. In the on-policy training with the simulation, the model has decreasing probability $\eta = 0.2e^{-\frac{\beta}{1000}}$ of choosing a random reply in the candidate set so as to ensure the sufficient exploration, where β is the number of training dialogues seen by the algorithm.

4.2.6 Algorithm and Complexity

The detailed PETAL algorithm is shown in Algorithm 1. We train our model for each user in the source domain. $\mathbf{M}$, $\mathbf{w}$ and w_p are shared by all users and there is a separate $\mathbf{p}_u$ for each user in

Algorithm 1 The PETAL Algorithm

Input: $\mathcal{D}^s$,$\mathcal{D}^t$
Output: $\Theta = \{\mathbf{M}, \mathbf{w}, w_p \{\mathbf{p}_u\}\}$
procedure TRANSFER ALGORITHM($\mathcal{D}^s$,$\mathcal{D}^t$)
    $\{\mathbf{M}, \mathbf{w}, w_p\} \leftarrow$ TRAIN-SOURCE-MODEL($\mathcal{D}^s$)
    $\{\mathbf{M}, \mathbf{w}, w_p, \{\mathbf{p}_u\}\} \leftarrow$TRANSFER($\mathcal{D}^t$,$\mathbf{M}$,$\mathbf{w}$,$w_p$)
function TRAIN-SOURCE-MODEL($\mathcal{D}^s$)
    for $\{X_n^u, Y_n^u\}$ in $\mathcal{D}^s$ **do**
        $\mathbf{p}_u = \mathbf{0}$
        if $\mathbf{p}_u$ exist **then** load $\mathbf{p}_u$
        else $\mathbf{p}_u \leftarrow \mathbf{0}$
        for $(H_n^u, Y_n^u, r_n^u, H_{n+1}^u, Y_{n+1}^u)$ in $\{X_n^u, Y_n^u\}$ **do**
            $\Theta_{t+1} \leftarrow \Theta_t + \alpha \Delta_\Theta \mathcal{L}(\Theta_t)$
    return $\{\mathbf{M}, \mathbf{w}, w_p\}$
function TRANSFER($\mathcal{D}^t$, $\mathbf{M}, \mathbf{w}, w_p$)
    for $\{\{X_n^u, Y_n^u\}_n^T\}$ in $\mathcal{D}^t$ **do**
        $\mathbf{p}_u = \mathbf{0}$
        if $\mathbf{p}_u$ exist **then** load $\mathbf{p}_u$
        else $\mathbf{p}_u \leftarrow \mathbf{0}$
        for $(H_n^u, Y_n^u, r_n^u, H_{n+1}^u, Y_{n+1}^u)$ in $\{X_n^u, Y_n^u\}$ **do**
            $\Theta_{t+1} \leftarrow \Theta_t + \alpha \Delta_\Theta \mathcal{L}(\Theta_t)$
    return $\{\mathbf{M}, \mathbf{w}, w_p, \{\mathbf{p}_u\}\}$

the source domain. We transfer $\mathbf{M}$, $\mathbf{w}$ and w_p to the target domain by using them to initialize the corresponding variables in the target domain, and then we train them as well as $\mathbf{p}_u$ for each target user with limited training data. Since the source and target users might have different preferences, $\mathbf{p}_u$ learned in source domain is not very useful in the target domain. The personal preference of each target user will be learned separately in each $\mathbf{p}_u$. Without modeling $\mathbf{p}_u$ for each user, different preferences of the source and target users might interfere with each other and thus cause the negative transfer.

The number of parameters in our model is around $d^2 + dl$, where l is the total vocabulary size and d is the dimension of the state vector. In our experiment where $l = 1,500$ and $d = 50$, the number of parameters in the general Q-function is about $85k$ and that for the personal Q-function is about 100 for each user, hence the parameters in the personalized Q-function could be learned accurately with the limited data in the target domain.

While the joint training of source loss and target loss might help us to find a better solution, the two-step method has its advantage. It is good when one can keep the source data private if they only deliver the source model, and is more computational efficient since the source data only need to be trained once.

4.3 Experiments

In this section, we experimentally verify the effectiveness of the proposed PETAL model by conducting experiments on a real-world dataset and a simulation dataset.

4.3.1 Baselines

We compare the proposed PETAL model with six baseline algorithms which are listed as follows:

1. NoneTL: The dialogue system is trained only with the data from target users.
2. Sim [20]: The dialogue system is trained with the data from both target user and the most similar user in the source domain.
3. Bandit [42]: For each target user, the most useful source user is identified by a bandit algorithm.
4. PriorSim [37]: For each target user, the policy from the most similar user in the source domain is used as a prior.
5. PriorAll [37]: For each target user, the dialogue policy trained on all the users in the source domain is used as a prior.
6. All: The policy is trained on all source users' data.

In order to avoid possible performance difference caused by different base models, we implemented the about baseline methods on top of the same base model.

4.3.2 Experiments on Real-World Data

Table 4.1: Statistics of the datasets

	Source Domain		Target Domain	
Dataset	Users	Dialogues	Users	Dialogues
Real Data	52	1,859	20	329
Simulation	11	176,000	5	100

In this section, we evaluate our model on a real-world dataset. This real-world dataset, which is collected between July 2015 and April 2016 from an O2O coffee ordering service in a major

instant message platform in China, contains 2,185 coffee dialogues between 72 consumers and coffee makers. The users order coffee by providing the coffee type, the temperature, the cup size and the delivery address, hence there are 4 order choices. We select 52 users with more than 23 dialogues as the source domain. Each of the remaining 20 users is used separately as a target domain. In total, there are 1,859 coffee dialogues in the source domain and 329 coffee dialogues in the target domain. 221 earlier dialogues in the target domain are used as the training set and the remaining 108 dialogues form the test set. The statistics of this dataset is shown in Table 4.1. Note that the popular DSTC datasets do not have personalized preferences and thus could not be used in this chapter.

Settings

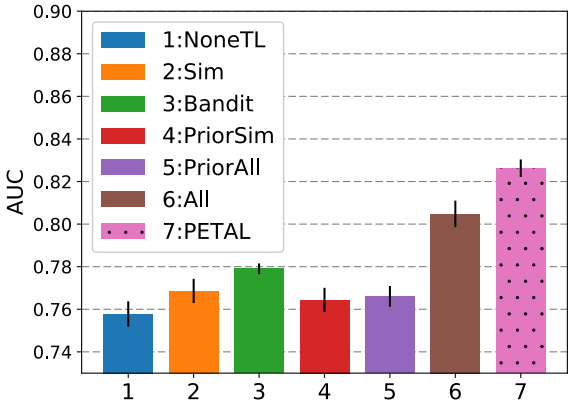
Each user in the target domain is regarded as a target user. We transfer knowledge from the whole source domain to each target user in the target domain. Firstly, we train the source model in the source domain. Then, we transfer the model to the target domain and train on the training set of the target domain. Finally, we test the model on the testing set of the target domain.

Evaluation Metrics

For each turn of the testing conversation, a model will rank the ground-truth reply Y_n^u among 10 randomly chosen agent replies. The label assigned to Y_n^u is 1 and those for randomly chosen agent replies are 0. By following [177], we calculate the AUC score for each turn in a conversation and the performance of an algorithm is measured by the average AUC score of each dialogue for every user in the test set. AUC is the area under the ROC curve which measures the probability that a ground-truth reply will be rank in front of random replies. If a model can rank ground-truth reply higher, then this model is better.

Results

In Figure 4.3(a) and Table 4.3, we report the mean and standard deviation of averaged AUC score with 5 different random seeds, which are used to randomly sample agent replies as candidates. The performance of "NoneTL", "PriorSim" and "PriorAll" are worse than "All" which directly transfers training data, because fitting only target domain data can cause the overfitting. Transferring data from similar users (i.e., "Sim") is not as good as transferring data from all source users (i.e., "All"), because common knowledge has to be learned from more data. The proposed "PETAL" method



(a) Real-world Average AUC (the higher the better).

Figure 4.3: Average AUC in the Real-world dataset.

performs the best because it learns common knowledge from all users and avoids the negative transfer caused by different preferences among source and target users, which indicates that the proposed personalized model fits dialogues better and demonstrates the effectiveness of PETAL on this real-world dataset.

Table 4.2: A case study on the real-world dataset. The last column shows candidate responses, where the ground-truth response is marked with *. The first and second columns show predicted rewards of "All" and "PETAL" on these candidates.

User utterance :		I want a cup of coffee.
All	PETAL	Response Candidates
0.86	1.36	* Same as before? Tall hot americano and deliver to Central Conservatory of Music?
0.99	0.92	All right, deliver to No.1199 Beiyuan Road, Chaoyang District, Beijing?
0.72	0.69	What's your address?

Case Study

A case study is shown in Table 4.2, where we show three candidates. From the results, we can see that the proposed "PETAL" method ranks the ground-truth response in the first place based on the predicted reward given by the learned personalized Q-function but the "All" method without personalization ranks a wrong address higher, which demonstrates the effectiveness of the proposed method.

4.3.3 Experiments on Simulation Data

In this section, we compare our model with baseline models on the simulated coffee-ordering dialogue data. The simulated users order coffee by providing their coffee type, temperature, size and delivery address, and the agent's reply by choosing from a set of predefined candidate responses without knowing the speech-act.

Settings

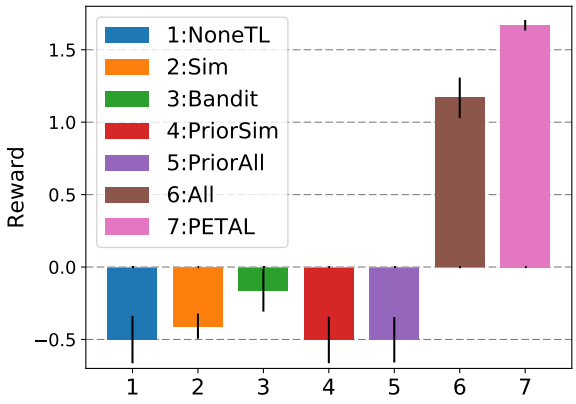
We have 11 simulated users in the source domain, in which 10 users have their own coffee preferences while the rest one has no preference. The target domain has 5 users, which have different preferences from users in the source domain. A simulator is designed based on the real-world dataset used in the previous section. The simulator will order according to his preference with probability 0.8 and otherwise, the simulator will order coffee randomly. The training set of each user in the target domain has 20 dialogues and the test set has 300 dialogues. The reward in the experiment is the same as the reward defined in the reward section. Specifically, if the user rejects to pay due to the non-logical response made by the system, the system receives a negative reward. Firstly, we train the source model by interacting with the source users for a fixed iteration. Then we transfer the source model to target domain and train on the target domain user for 20 dialogues. Finally, we test the model on the target user for 300 dialogues.

Evaluation Metrics

Each model will choose a reply from a set of candidates generated with templates and updated dialogue information at each turn, and the simulated users will react to the chosen reply accordingly. We use three evaluation metrics to measure the quality of the whole dialogue. For each model, we report the mean and standard deviation of averaged reward [42], averaged success rate [20] and averaged dialogue length over all possible target users, repeated for 5 times with different random seeds.

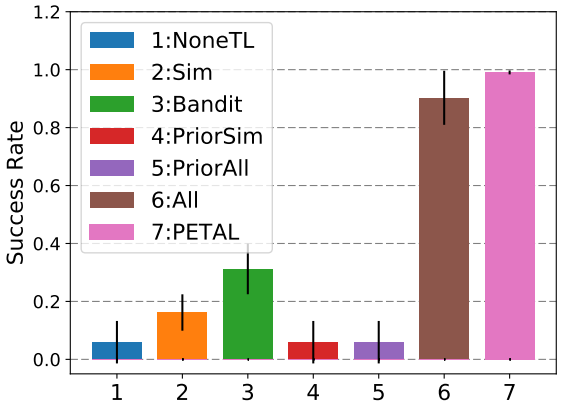
Results

The results are shown in Figure 4.4(a), Figure 4.5(a), Figure 4.5(b) and Table 4.3. PETAL outperforms all baselines and obtains the highest average reward, the highest success rate and the lowest dialogue length, which implies that PETAL has found a better dialogue policy which can adapt its

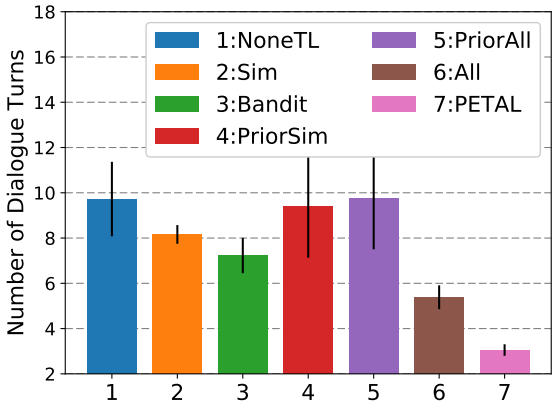


(a) Simulation Average Reward (the higher the better).

Figure 4.4: Averaged reward in the Simulation.



(a) Simulation Success Rate (the higher the better).



(b) Simulation Average Dialogue Length (the lower the better).

Figure 4.5: Simulation Success Rate and Dialogue Length.

behavior according to the preference of target users and again demonstrates the effectiveness of PETAL in a live environment.

Case Study

We show a typical case for the simulation data in Tables 4.4 and 4.6. The non-personalized dialogue system corresponding to the "All" model has to ask the users all the choices even for frequent users in Table 4.6, because there is no universal recommendation for all the frequent users with different preferences. However, PETAL has learned the target users' preferences in previous dialogues. As shown in Table 4.4, the response from the agent is specially tailored for the target user because

Table 4.3: Experimental results on both the real-world and simulation data

	Real-World	Simulation		
Methods	AUC	Reward	SucRate	Length
NoneTL	0.7577 ± 0.0059	-0.5040 ± 0.1531	0.0592 ± 0.0733	9.1882 ± 1.5200
Sim	0.7685 ± 0.0056	-0.0711 ± 0.0522	0.3679 ± 0.0446	7.1681 ± 0.6715
Bandit	0.7790 ± 0.0025	-0.1600 ± 0.0784	0.3166 ± 0.0511	8.1937 ± 0.3648
PriorSim	0.7643 ± 0.0057	-0.3982 ± 0.2395	0.1085 ± 0.1178	9.6772 ± 2.4038
PriorAll	0.7660 ± 0.0048	-0.5014 ± 0.1514	0.0592 ± 0.0733	9.4897 ± 1.9928
All	0.8048 ± 0.0062	1.2713 ± 0.1421	0.9306 ± 0.0361	5.0625 ± 0.6621
PETAL	**0.8262 ± 0.0041**	**1.6385 ± 0.0718**	**0.9841 ± 0.0138**	**3.0838 ± 0.2695**

personalized questions given by the PETAL method can guide the user to complete the coffee-ordering task faster than general questions, leading to shorter dialogue and higher averaged reward. If the user does not want everything as usual, which is shown in the second case of Table 4.5, PETAL can still react correctly due to the shared dialogue knowledge transferred from the source domain. These cases show that PETAL can choose different optimal actions for different users and effectively shorten the conversation.

Table 4.4: Personalized Dialogue Case, when the user wants everything as usual.

User:	I want a cup of coffee.
Agent:	Same as before? Tall hot macchiato and deliver to No.1199 Minsheng Road, Pudong District Shanghai?
User:	Yes.
Agent:	Please pay.
User:	Payment completed.

Table 4.5: Personalized Dialogue Case, when the user wants to try new options.

User:	I want a cup of coffee.
Agent:	Same as before? Tall hot macchiato and deliver to No.1199 Minsheng Road, Pudong District, Shanghai?
User:	I want iced mocha today.
Agent:	Sure, please pay.
User:	Payment completed.

Table 4.6: A Non-Personalized Dialogue Case

User:	I want a cup of coffee.
Agent:	Hot coffee?
User:	Yes.
Agent:	What would you like to drink?
User:	Hot macchiato, please.
Agent:	What's your address?
User:	No.1199 Minsheng Road, Pudong District, Shanghai.
Agent:	Alright. Tall, Grande, or Venti?
User:	Tall.
Agent:	Please pay.
User:	Payment completed.

4.4 Conclusion

In this chapter, we tackle the problem of learning the dialogue states and dialogue policy automatically from the source domain and adapt to the target user with little data. We propose the PETAL system, a transfer reinforcement learning framework based on the POMDP. The PETAL system first learns common dialogue knowledge from the source domain and then adapts this knowledge to the target user. We propose to model a personalized policy with a personalized Q-function, which can avoid the negative transfer problem brought by differences between the source users and the target user. The policy can learn to choose different actions appropriately for different users. Experimental results on the real-world data and the simulation data show that PETAL can learn different optimal policies for different users and thus effectively improves the dialogue quality under the personalized setting.

CHAPTER 5

TRANSFER REINFORCEMENT LEARNING THROUGH PERSONAL WORD GATING

Training a personalized dialogue system requires a lot of data, and the data collected for a single user is usually insufficient. One common practice for this problem is to share training dialogues between different users and train multiple sequence-to-sequence dialogue models together with transfer learning. However, current sequence-to-sequence transfer learning models operate on the entire sentence, which might cause negative transfer if different personal information from different users is mixed up. We propose a personalized decoder model to transfer finer granularity phrase-level knowledge between different users while keeping personal preferences of each user intact. A novel personal control gate is introduced, enabling the personalized decoder to switch between generating personalized phrases and shared phrases. The proposed personalized decoder model can be easily combined with various deep models and can be trained with reinforcement learning. Real-world experimental results demonstrate that the phrase-level personalized decoder improves the BLEU over multiple sentence-level transfer baseline models by as much as 7.5%.

5.1 Introduction

Task-oriented dialogue systems can be categorized into rule-based systems and learning-based systems. Learning-based dialogue systems do not require the predefined dialogue state, which are more general and suitable for the situation when exact dialogue states are hard to define. In particular, neural network based dialogue systems do not require the predefined templates and are more flexible. In this chapter, we focus on neural network based task-oriented dialogue systems.

Personalized dialogue systems can help the target user to complete a task faster and make the dialogue more efficient. In a personalized dialogue system, the personal information and preference of the target user are recorded and utilized, thus the personal dialogue policy can generate personalized sentences and speed up the dialogue process for the target user. In personalized dialogue systems, the dialogue policy and response generation for each user is different. Training a whole personalized dialogue system requires a lot of data and the data collected from a single user

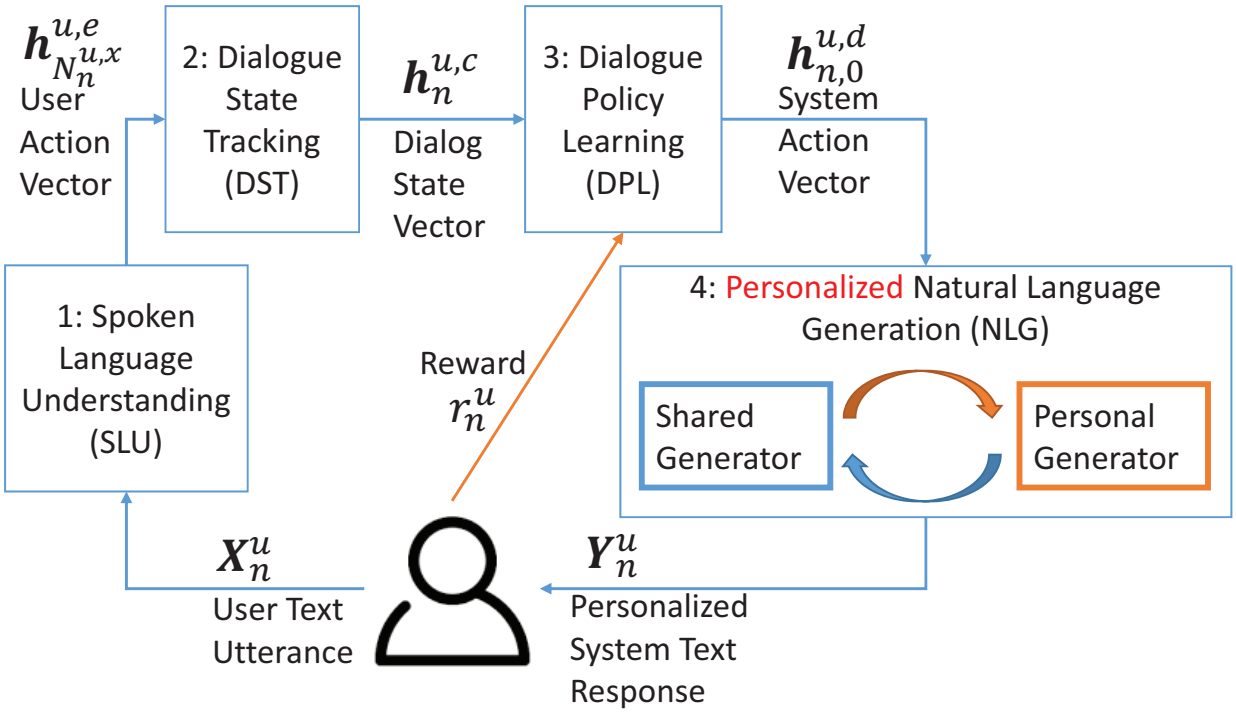


Figure 5.1: Framework for Personalized Task-oriented Dialogue System

is usually insufficient. Multi-tasking can be used to transfer dialogue knowledge across different users by sharing training dialogues.

However, current neural network transfer models work on the entire sentence, but the phrases concerning personal information from different users should not be transferred across users at all. For example, if the dialogue data from a sugar lover is used to train a personalized dialogue system for a diabetes patient, the system might recommend sweet drinks to the patient and cause a disaster. Hence, we think that knowledge transfer in personalized dialogue systems should be conducted in finer granularity.

In this chapter, we propose a personalized decoder that can transfer shared phrase-level knowledge between different users while keeping the personalized information of each user intact. A novel personal control gate is introduced in the proposed personalized decoder, enabling the decoder to switch between generating shared phrases and personal phrases. For example in Figure 5.2(b), "Hot Latte" is a personal phrase and "Still" and "?" are shared phrases. The personalized decoder can generate different personal phrases for different users in a sentence, while the knowledge for the shared phrases is shared among all users. The proposed personalized decoder can be easily used with various base models to achieve fine-grained phrase-level dialogue knowledge transfer between different users. Real-world experimental results show that the personalized decoder can improve the BLEU score over multiple sentence-level transfer models by as much as

7.5%.

5.1.1 Challenges

To address the transfer reinforcement learning problem and learn an end-to-end dialogue policy for the target user, we are facing the following challenges.

1. There is no ground-truth dialogue state space for each dialogue domain, and there is no state annotation for each dialogue sentence. We have to design an end-to-end dialogue policy, which can learn the dialogue state as a latent factor.

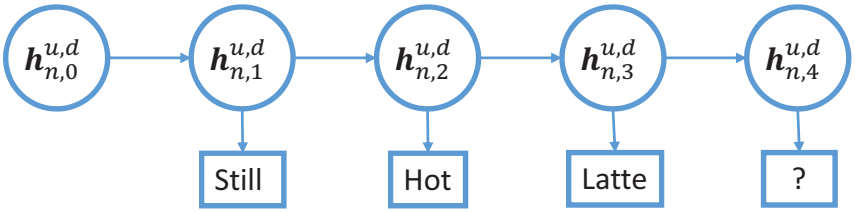
2. Different users have different preferences, resulting in a different data distribution for each user. Naively transferring the whole dialogue sentence across users might lead to personal information leakage, and the policy in the target domain might generate inappropriate suggestions. We have to design a model that could separate the common knowledge with the personal knowledge.

3. The common knowledge is mixed with the personal information that should not be transferred. Existing methods either transfer the whole dialogue sentence or transfer nothing at all, and either way could not fully exploit the knowledge in the source domain safely. We have to design a fine-granularity control mechanism, which could enable the transfer of only the common knowledge.

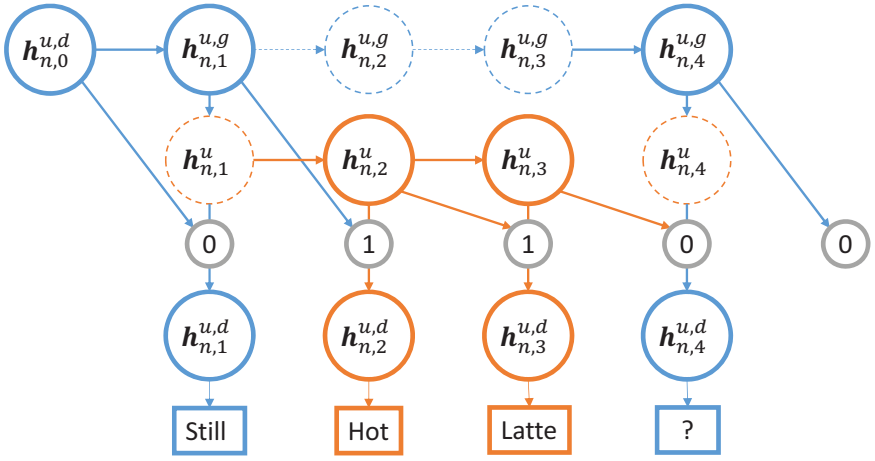
5.1.2 Related work of Transfer Learning in Neural Based Dialogue Response Generation Systems

learning-based task-oriented dialogue systems [20, 42, 37, 38, 177, 17] can select appropriate answers from a set of predefined answers. Gašic *et al.* [37, 38] used transfer learning in the dialogue management module to extend a dialogue system to handle previous unseen concepts. Williams *et al.* [177, 17] used learning methods in spoken language understanding and dialogue management. These methods require the predefined answer candidates or templates. Wen [169, 165, 166] proposed to use trainable modules for each part of the dialogue system which do not require predefined answer candidates or templates. However, this model still requires the predefined slots for dialogue policy learning, which limits its application.

Neural based dialogue response generation systems require neither the answer candidates/templates, nor the predefined slots for dialogue policy. Sequence to sequence (seq2seq) models and



(a) Response generation with a basic decoder. $\mathbf{h}^{u,c}_n$ is not shown since it is the same for all time step t.



(b) Personalized response generation with the proposed personalized decoder. The personal control gate at different time steps are denoted in gray circles.

Figure 5.2: Response generation comparison

their variants [7, 134, 127, 118, 77] are widely used to model dialogues. However, data collected from each user is insufficient for training a personalized dialogue system.

Multi-task learning is used to transfer knowledge in sequence to sequence model. Luong *et al.* [82] proposed to share encoder/decoder across different tasks in order to achieve knowledge transfer. However, these knowledge transfer works on the granularity of sentences. Since different persons have different preferences and require different dialogue policies and responses, directly transferring sentences across different users might lead to a negative transfer.

In this chapter, a personalized decoder is proposed for response generation, which is capable of transferring dialogue knowledge, and it can easily be combined with many models including seq2seq [134] and HRED [118].

5.2 Model: Personalized Decoder

In this section, we firstly introduce the proposed personalized decoder, then introduce the combination of the personalized decoder with two models, and finally present the training method.

5.2.1 Problem

In this section, we first define notations and then present the problem settings.

Notation In this thesis, matrices are denoted in a bold capital case, column vectors are in a bold lower case, and scalars are in a lower case. Question in the n-th turn is denoted by $X_n^u = \{x_{n,t}^u\}_{t=1}^{N_n^{u,x}}$, and $N_n^{u,x}$ is the number of words in X_n^u. Response in the n-th turn is denoted by $Y_n^u = \{y_{n,t}^u\}_{t=1}^{N_n^{u,y}}$, where $N_n^{u,y}$ is the number of words in Y_n^u. To be consistent, a dialogue turn is indexed by n and a word is indexed by t.

Problem Definition

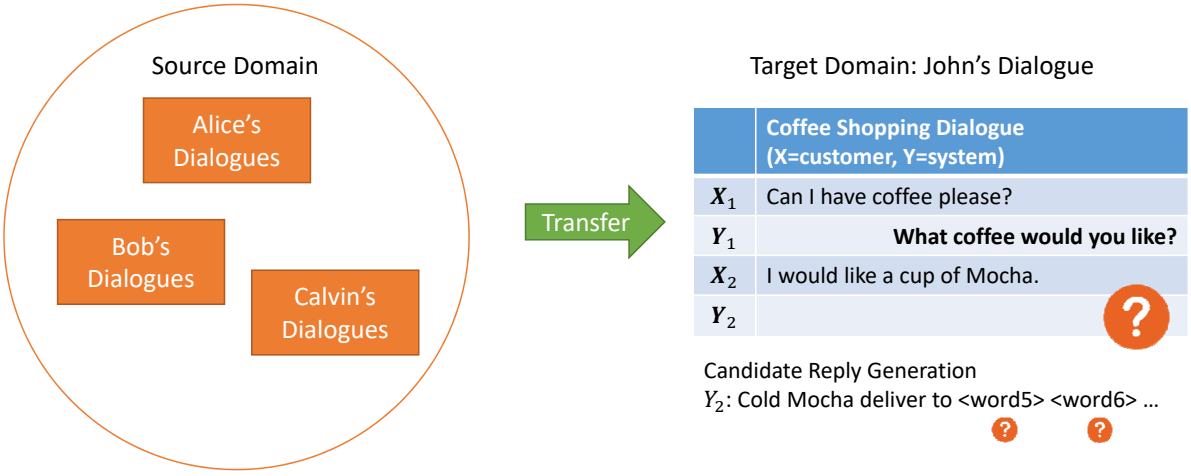


Figure 5.3: The problem illustration

The problem is illustrated in Figure 5.3. Given the conversation history of multiple users, we aim to learn an end-to-end personalized task-oriented dialogue system for each user. The input of the problem are:

1. Historical dialogue sessions $\mathcal{T}^u = \{X_n^u, Y_n^u, r_n^u\}_{n=1}$ of each user, where r_n^u is the reward obtained at n-th dialogue turn.

2. Personal word label $\mathcal{O}_n^u = \{o_{n,t}^u\}_{t=1}^N$ for each word in Y_n^u, where $o_{n,t}^u = 1$ means that $x_{n,t}^u$ is a personal word and $o_{n,t}^u = 0$ means the word $x_{n,t}^u$ is a general word.

The output of the problem are:

1. A dialogue policy π^u for each user u, which generates a response Y_n^u for each dialogue history $H_n^u = \{\{X_i^u, Y_i^u\}_{i=1}^{n-1}, X_n^u\}$.

Personal Word Label Annotation

In the setting of task-oriented dialogue systems, the personal words have a clear boundary. They are well defined as the words related to all possible choices (in our case, coffee-type, temperature, cup size, address) in the domain slots, so there is only one correct way to assign personal word label. We used 3 methods to annotate personal words.

1. Firstly, for coffee type, temperature and cup size slots in which all possible choices could be enumerated, we used keyword matching to identify the possible choices.

2. Secondly, for the address which cannot be enumerated, we use the semantic tree parser. Based on special keywords such as road, building, city name, the semantic parser can give a rough guess for the addresses.

3. Finally, in order to improve the label quality, human labelers are asked to correct mistakes in the previous steps and they are instructed to find the synonyms and abbreviations for all the slots considering coffee-type, temperature, cup size and addresses and mark these words as personal words.

In other task-oriented dialogue systems with a different domain or different dataset, we can follow a similar procedure to collect labels for personal words. We can either collect all possible choices in the slots or find the pattern for the slots. Moreover, we can use statistical techniques such as IDF and machine learning models to find the pattern of the slots automatically.

5.2.2 Decoder for Response Generation

In this section, we introduce the mathematical formulation of the proposed personalized decoder.

Basic Decoder The hidden state for the t-th word in the n-th turn is defined as

$$\mathbf{h}_{n,t}^{u,d} = \tanh(\mathbf{W}^d \mathbf{h}_{n,t-1}^{u,d} + \mathbf{U}^d \hat{\mathbf{y}}_{n,t-1}^{u} + \mathbf{V}^d \mathbf{h}_{n}^{u,c}),$$

where $\hat{\mathbf{y}}_{n,t-1}^{u}$ is the word embedding of the last word $\hat{y}_{n,t-1}^{u}$ in the same sentence, and $\tanh(\cdot)$ denotes the hyperbolic tangent function. The decoder RNN in Figure 5.2(a) takes $\mathbf{h}_{n,0}^{u,d}$ and $\mathbf{h}_{n}^{u,c}$ as inputs[1] and then generates the response word by word, where $\mathbf{h}_{n}^{u,c}$ is the embedding vector used to

[1] $\mathbf{h}_{n}^{u,c}$ is not shown in Figure 5.2(a) since it is the same for all time step t.

generate the current response. The probability of generating the next word $\hat{y}^u_{n,t} = \mathrm{y}$ given $\mathbf{h}^{u,d}_{n,t}$ and $\hat{y}^u_{n,t-1}$ is

$$\omega(\mathbf{h}^{u,d}_{n,t}, \hat{y}^u_{n,t-1}) = \mathbf{H}_0\mathbf{h}^{u,d}_{n,t} + \mathbf{E}_0\hat{\mathbf{y}}^u_{n,t-1} + \mathbf{b}_o \tag{5.1}$$

$$g(\mathbf{h}^{u,d}_{n,t}, \hat{y}^u_{n,t-1}, \mathrm{y}) = \mathbf{o}^T_{\mathrm{y}}\omega(\mathbf{h}^{u,d}_{n,t}, \hat{y}^u_{n,t-1}) \tag{5.2}$$

$$p(\hat{y}^u_{n,t} = \mathrm{y}) = \frac{\exp(g(\mathbf{h}^{u,d}_{n,t}, \hat{y}^u_{n,t-1}, \mathrm{y}))}{\sum_{\forall \mathrm{y}'} \exp(g(\mathbf{h}^{u,d}_{n,t}, \hat{y}^u_{n,t-1}, \mathrm{y}'))} \tag{5.3}$$

where $\mathbf{o}_{\mathrm{y}}$ is the output embedding for word y, and $\mathbf{H}_0$, $\mathbf{E}_0$ and $\mathbf{b}_o$ are parameters.

Personalized Dialogue Decoder for Phrase-level Transfer Learning In this section, we present the proposed personalized decoder as illustrated in Figure 5.2(b). While the sentence-level transfer is to transfer entire sentences, the proposed personalized decoder is on the phrase level and is to transfer a shared fraction of the sentences to the target domain, where a phrase is a short sequence of words containing a coherent meaning, for example, an address. In order to achieve maximum knowledge transfer and to avoid negative transfer caused by differences in user preferences, the proposed personalized decoder has a shared component and a personalized component. In order to learn to switch between the shared and personal components in the phrase level, we introduce a personal control gate $o^u_{n,t}$ for each word, which is learned from the training data.

Given the the embedding vector for the n-th response $\mathbf{h}^{u,c}_n$ and initial hidden state $\mathbf{h}^{u,d}_{n,0}$ for the predicted word $\hat{y}^u_{n,0}$, the initial states are computed as

$$\mathbf{h}^{u,g}_{n,0} = \mathbf{h}^{u,d}_{n,0}, \mathbf{h}^u_{n,0} = \mathbf{h}^{u,d}_{n,0}, \hat{o}^u_{n,0} = 0$$
$$\hat{\mathbf{y}}^u_{n,0} = \mathbf{0}, \hat{\mathbf{y}}^{u,g}_{n,0} = \mathbf{0}$$

where $\mathbf{h}^{u,g}_{n,t}$ is the hidden state for the shared component, and $\hat{y}^{u,g}_{n,t}$ records the last word generated by the shared component, $\mathbf{h}^u_{n,t}$ is the hidden state for the personal component, and $\mathbf{h}^{u,d}_{n,t}$ is the hidden state for generating the word $\hat{y}^u_{n,t}$.

The shared component adopts the GRU model to capture the long-term dependency and is shared by all users. Specifically, at each time step t, the shared component is defined as

$$\mathbf{z}^u_{n,t} = \sigma(\mathbf{W}^g_z\mathbf{h}^{u,g}_{n,t-1} + \mathbf{U}^g_z\hat{\mathbf{y}}^{u,g}_{n,t-1} + \mathbf{V}^g_z\mathbf{h}^{u,c}_n + \mathbf{b}_z) \tag{5.4}$$

$$\mathbf{r}^u_{n,t} = \sigma(\mathbf{W}^g_r\mathbf{h}^{u,g}_{n,t-1} + \mathbf{U}^g_r\hat{\mathbf{y}}^{u,g}_{n,t-1} + \mathbf{V}^g_r\mathbf{h}^{u,c}_n + \mathbf{b}_r) \tag{5.5}$$

$$\tilde{\mathbf{h}}^{u,g}_{n,t} = \sigma(\mathbf{W}^g_h(\mathbf{r}^u_{n,t} \odot \mathbf{h}^{u,g}_{n,t-1}) + \mathbf{U}^g_h\hat{\mathbf{y}}^{u,g}_{n,t-1} + \mathbf{V}^g_h\mathbf{h}^{u,c}_n + \mathbf{b}_h) \tag{5.6}$$

$$\hat{\mathbf{h}}^{u,g}_{n,t} = \mathbf{z}^u_{n,t} \odot \mathbf{h}^{u,g}_{n,t-1} + (1 - \mathbf{z}^u_{n,t}) \odot \tilde{\mathbf{h}}^{u,g}_{n,t}, \tag{5.7}$$

where $\odot$ denotes the element-wise product between vectors or matrices, $\sigma(\cdot)$ is the sigmoid function, $\mathbf{z}^u_{n,t}$ is the update gate, $\mathbf{r}^u_{n,t}$ is the forget gate, and $\hat{\mathbf{h}}^{u,g}_{n,t}$ is the tentative updated hidden state. If the t-th word is a shared word (i.e., $\hat{o}^u_{n,t} = 0$), then we update the shared hidden state and last general word as usual and otherwise $\mathbf{h}^{u,g}_{n,t}$ and $\hat{\mathbf{y}}^{u,g}_t$ remain unchanged. Thus $\mathbf{h}^{u,g}_{n,t}$ and $\hat{\mathbf{y}}^{u,g}_t$ can be updated as

$$\mathbf{h}^{u,g}_{n,t} = (1 - \hat{o}^u_{n,t}) \odot \hat{\mathbf{h}}^{u,g}_{n,t} + \hat{o}^u_{n,t} \odot \mathbf{h}^{u,g}_{n,t-1} \tag{5.8}$$

$$\hat{\mathbf{y}}^{u,g}_t = (1 - \hat{o}^u_{n,t}) \odot \hat{\mathbf{y}}^u_{t-1} + \hat{o}^u_{n,t} \odot \hat{\mathbf{y}}^{u,g}_{t-1}. \tag{5.9}$$

The personal component is a RNN model, which generates personalized sequence based on sentence context $\mathbf{h}^{u,g}_{n,t}$ from the shared component. There is an separate RNN model for each user. At each time step t, the personal component receives $\hat{\mathbf{y}}^u_{t-1}$, $\hat{o}^u_{n,t}$, $\mathbf{h}^u_{n,t-1}$ and $\mathbf{h}^{u,g}_{n,t-1}$ as inputs and outputs $\hat{\mathbf{h}}^u_{n,t}$, which is defined as

$$\hat{\mathbf{h}}^u_{n,t} = \sigma(\mathbf{W}^u \mathbf{h}^u_{n,t-1} + \mathbf{U}^u \hat{\mathbf{y}}^u_{n,t-1} + \mathbf{V}^u \mathbf{h}^{u,g}_{n,t-1}). \tag{5.10}$$

The personal hidden state will be update as

$$\mathbf{h}^u_{n,t} = (1 - \hat{o}^u_{n,t}) \odot \mathbf{h}^{u,g}_{n,t} + \hat{o}^u_{n,t} \odot \hat{\mathbf{h}}^u_{n,t}. \tag{5.11}$$

$\mathbf{h}^u_{n,t}$ equals $\hat{\mathbf{h}}^u_{n,t}$ if the control gate is on corresponding to $\hat{o}^u_{n,t} = 1$. If $\hat{o}^u_{n,t}$ equals 0, $\mathbf{h}^u_{n,t}$ will take the value of $\mathbf{h}^{u,g}_{n,t}$.

The personal control gate $o^u_{n,t}$ is binary, i.e., $o^u_{n,t} \in \{0, 1\}$. The predicted control gate $\hat{o}^u_{n,t}$ at time t is a function of $\hat{o}^u_{n,t-1}$, $\mathbf{h}^{u,g}_{n,t-1}$, $\mathbf{h}^u_{n,t-1}$, and $\hat{\mathbf{y}}^u_{n,t-1}$ as[2]

$$p(\hat{o}^u_{n,t} = 1) = \begin{cases} \sigma(\mathbf{W}^g_o \mathbf{h}^{u,g}_{n,t-1} + \mathbf{U}^g_o \hat{\mathbf{y}}^u_{n,t-1} + \mathbf{b}_o) & \text{if } \hat{o}^u_{n,t-1} = 0 \\ \sigma(\mathbf{W}^u_o \mathbf{h}^u_{n,t-1} + \mathbf{U}^u_o \hat{\mathbf{y}}^u_{n,t-1} + \mathbf{b}^u_o) & \text{if } \hat{o}^u_{n,t-1} = 1 \end{cases}. \tag{5.12}$$

$\hat{o}^u_{n,t}$ decides whether to use the personal component to generate the next word. $\mathbf{h}^{u,d}_{n,t}$ is defined as

$$\mathbf{h}^{u,d}_{n,t} = (1 - \hat{o}^u_{n,t}) \odot \mathbf{h}^{u,g}_{n,t} + \hat{o}^u_{n,t} \odot \mathbf{h}^u_{n,t}, \tag{5.13}$$

where $\mathbf{h}^{u,d}_{n,t}$ is the hidden vector that directly generates the next word $\hat{y}^u_{n,t}$ and the probability of generating the next word $y^u_{n,t}$ is defined by the generation process in Eqs. (5.1-5.3).

The decoding procedure is as follows:

1. Initialize $\mathbf{h}^{u,g}_{n,0}$, $\mathbf{h}^u_{n,0}$, $\hat{o}^u_{n,0}$, $\hat{\mathbf{y}}^u_{n,0}$, $\hat{\mathbf{y}}^{u,g}_{n,0}$ based on $\mathbf{h}^{u,d}_{n,0}$ and $\mathbf{h}^{u,c}_n$. $\hat{o}^u_{n,0}$ is initialized to be 0 and $\hat{\mathbf{y}}^u_{n,0}$ is initialized to be a zero vector $\mathbf{0}$.

[2]In training process, the ground-truth $o^u_{n,t}$ is used as label to train the prediction function for $\hat{o}^u_{n,t}$.

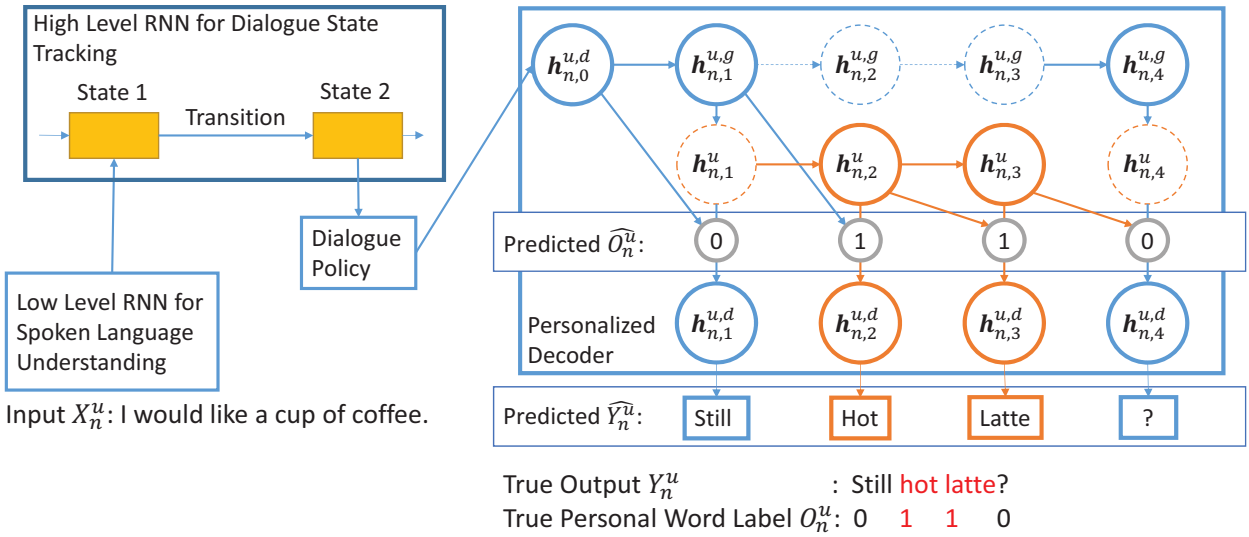


Figure 5.4: Model framework for personalized HRED, obtained by combining HRED with Personalized Decoder

2. Compute personal control gate $\hat{o}_{n,t}^u$ based on $\mathbf{h}_n^{u,c}$, $\hat{o}_{n,t-1}^u$, $\mathbf{h}_{n,t-1}^{u,g}$, $\mathbf{h}_{n,t-1}^u$ and $\hat{\mathbf{y}}_{t-1}^u$ with Eqs. (5.12).

3. Compute $\mathbf{h}_{n,t}^{u,g}$, $\mathbf{h}_{n,t}^u$ and the output hidden state $\mathbf{h}_{n,t}^{u,d}$ based on the personal control gate $\hat{o}_{n,t}^u$.

4. Generate $\hat{y}_{n,t}^u$ based on the output hidden state $\mathbf{h}_{n,t}^{u,d}$ with Eqs. (5.13).

5. Repeat step 2 to step 4 until the ending symbol.

The shared and personal components can be trained together with reinforcement learning as illustrated in the parameter learning section.

Compared with the basic decoder, the personalized decoder is novel in the following aspects:

1. Knowledge is transferred in the phrase level rather than the sentence level.

2. There is a shared component for knowledge transfer and a personalized component for each user in order to avoid negative transfer caused by different data distributions.

3. The personal control gate can learn when to use the shared component and when to use the personal component to generate a fluent and personalized response.

5.2.3 Combining HRED with Personalized Decoder

In this section, we show how the proposed personalized decoder can be combined with hierarchical recurrent encoder-decoder (HRED) [118] to make a personalized HRED, whose model is shown in

Fig. 5.4.

In the personalized HRED, there are a low-level encoding RNN, a high-level RNN, a dialogue policy and a personalized decoder. The low-level encoding RNN for spoken language understanding encodes user utterances into a user action vector, the high-level RNN is responsible for dialogue state tracking, the dialogue policy maps the dialogue state to the action vector, and the personalized decoder is responsible for generating words as the response.

Spoken Language Understanding with Word Encoder The low-level encoder RNN for spoken language understanding maps each $X_n^u = \{x_{n,1}^u, x_{n,2}^u, \cdots, x_{n,N_n^{u,x}}^u\}$ to a fixed dimension vector $\mathbf{h}_{N_n^{u,x}}^{u,e}$ as

$$\mathbf{h}_{n,t}^{u,e} = \tanh(\mathbf{W}^e \mathbf{h}_{n,t-1}^{u,e} + \mathbf{U}^e \mathbf{x}_{n,t}^u), \tag{5.14}$$

where $\mathbf{x}_{n,t}^u$ is the word embedding vector of $x_{n,t}^u$, $\mathbf{U}^e$ is the input embedding matrix and $\mathbf{W}^e$ is the weight matrix corresponding to hidden state (word-level) transition function. Similarly we can define a mapping from the response $Y_n^u = \{y_{n,1}^u, y_{n,2}^u, \cdots, y_{n,N_n^{u,y}}^u\}$ to the response vector $\bar{\mathbf{h}}_{N_n^{u,x}}^{u,e}$ as

$$\bar{\mathbf{h}}_{n,t}^{u,e} = \tanh(\mathbf{W}^e \bar{\mathbf{h}}_{n,t-1}^{u,e} + \mathbf{U}^e \mathbf{y}_{n,t}^u). \tag{5.15}$$

Dialogue State Tracking with Sentence RNN The high-level sentence RNN tracks dialogue states based on all previous sentences in the dialogue. This RNN takes $\bar{\mathbf{h}}_{n-1,N_{n-1}^{u,y}}^{u,e}$, $\mathbf{h}_{n,N_n^{u,x}}^{u,e}$ and $\mathbf{h}_{n-1}^{u,c}$ as the input to generate next dialogue state $\mathbf{h}_n^{u,c}$ as:

$$\bar{\mathbf{h}}_{n-1}^{u,c} = \tanh(\mathbf{W}^c \mathbf{h}_{n-1}^{u,c} + \mathbf{U}^c \bar{\mathbf{h}}_{n-1,N_{n-1}^{u,y}}^{u,e}) \tag{5.16}$$

$$\mathbf{h}_n^{u,c} = \tanh(\mathbf{W}^c \bar{\mathbf{h}}_n^{u,c} + \mathbf{U}^c \mathbf{h}_{n,N_n^{u,x}}^{u,e}), \tag{5.17}$$

where $\bar{\mathbf{h}}_{n-1,N_{n-1}^{u,y}}^{u,e}$ and $\mathbf{h}_{n,N_n^{u,x}}^{u,e}$ are encoding vectors of sentences Y_{n-1}^u and X_n^u, $\mathbf{U}^c$ is the embedding matrix for the input, and $\mathbf{W}^c$ is the weight matrix corresponding to the hidden state (sentence-level) transition function.

Dialogue Policy with Linear Transformation Given the current dialogue state $\mathbf{h}_n^{u,c}$, the action vector $\mathbf{h}_{n,0}^{u,d}$ is calculated as

$$\mathbf{h}_{n,0}^{u,d} = \tanh(\mathbf{D}_0 \mathbf{h}_n^{u,c} + \mathbf{b}_0) \tag{5.18}$$

where $\mathbf{D}_0$ and $\mathbf{b}_0$ are the policy parameters.

Response Generation with Personalized Decoder Then the personalized decoder takes $\mathbf{h}_n^{u,c}$ and $\mathbf{h}_{n,0}^{u,d}$ as inputs to generate the response Y_n^u word by word.

The joint probability of $(H_n^u, \mathcal{O}_n^u, Y_n^u)$ is given by:

$$p(H_n^u, \mathcal{O}_n^u, Y_n^u) = p(H_n^u)p(\mathcal{O}_n^u, Y_n^u|H_n^u) \tag{5.19}$$

$$p(H_n^u) = \prod_k^{n-1} p(\mathcal{O}_k^u, Y_k^u|H_k^u), \tag{5.20}$$

where $\mathcal{O}_n^u = \{o_{n,t}^u\}_{t=0}^{N_n^{u,y}}$ is the collection of control gate variables for sentence Y_n^u.

5.2.4 Combining Seq2Seq Model with Personalized Decoder

The proposed personalized decoder can be combined with the popular seq2seq model proposed in [134]. We can build a personalized seq2seq model by replacing the original decoder in the seq2seq model with the proposed personalized decoder. The whole encoder and the common component of the personalized decoder are shared across all users, while each user has his own personal component in the personalized decoder.

Moreover, the proposed personalized decoder can also be easily combined with many other models to achieve knowledge transfer.

5.2.5 Parameter Learning

The whole model is trained in an end-to-end manner with reinforcement learning [135] such that the model will generate personalized system response according to current dialogue state to maximize the future cumulative reward.

Reward The agent will receive general rewards and personal rewards, and the total reward is the sum of general reward and personal reward. The general and personal rewards will be received under the following conditions:

1. Personal rewards of 0.3 will be received when the user confirms the suggestion of the agent. This is related to the personal information of the user. For example, the user could confirm the address suggested by the agent.

2. General rewards of 0.1 will be received when the user provides the information about the target task.

3. General rewards of 1.0 will be received when the system helps the user finish the target task successfully.

4. General reward of -0.2 will be received by the agent when the user rejects to proceed if the system is generating non-logical responses. A reward of -0.05 will be received for each the dialogue turn.

The reward score is designed with the following strategy. The agent is encouraged to actively make personalized suggestions instead of asking general questions if it has more than 0.6 probability of being correct. By using p to denote the probability of correct guess, the expected reward of making suggestions is $0.3p0.2(1.0-p)$ and the expected reward of asking a general question is 0.1. Solving $0.3p0.2(1.0-p) > 0.1$ implies $p > 0.6$. If $p < 0.6$, then the agent is discouraged to make a random guess. The task completion reward is set to a larger value 1.0.

Loss function We use the policy gradient method to learn the parameter of the model. Specifically, we use the REINFORCE [178] algorithm. The training set $\mathcal{D}^u$ consists of a set of trajectories $\mathcal{T}^u = \{H_n^u, \mathcal{O}_n^u, Y_n^u, r_n^u\}_{n=1}$, which records actions provided by the user. We denote all these parameters by Θ, and the model policy is denoted by π.

We define the return of the trajectory $J^{\mathcal{T}^u} = \sum_{n=1}^{N} \gamma^{n-1} r_n^u$ where r_n^u is the reward obtained at n-th dialogue turn. Then the loss function is defined as the expected reward of the policy under all trajectory $\mathcal{T}^u \in \mathcal{D}^u$:

$$J(\Theta) = \int_{\mathcal{T}^u} p(\mathcal{T}^u|\Theta) J^{\mathcal{T}^u} d\mathcal{T}^u, \tag{5.21}$$

whose gradient can be computed as

$$\Delta_\Theta J(\Theta) = \mathbb{E}\{\sum_{n=0}^{N} \Delta_\Theta \log \pi_\Theta(H_n^u, \mathcal{O}_n^u, Y_n^u) J_n^{\mathcal{T}^u}\} \tag{5.22}$$

where $J_n^{\mathcal{T}^u} = \sum_{k=n}^{N} \gamma^{k-n} r_k^u$ is the future cumulative reward and N is the total number of dialogue turns.

Optimization We adopt the Adam algorithm [61] to optimize our model. We train the model on data collected from different users, where the shared parameters are updated in each iteration while the personalized parameters are updated based on data collected from the corresponding user only.

5.3 Experiments

In this section, we experimentally verify the effectiveness of the personalized decoder by conducting experiments on a real-world dataset and a simulation dataset.

We compare the proposed two personalized phrase-level transfer models with their sentence-level counterparts and none-transfer versions. Note that we do not assume the predefined dialogue states or templates, thus rule-based systems do not apply here. All methods are listed as follows:

1. None-transfer seq2seq [134] (denoted by "S2S"): The sequence to sequence model is trained only with data from each individual user without transfer learning.

2. Sentence-level transfer seq2seq [82] (denoted by "ST-S2S"): The sequence to sequence model trained in a multi-task setting with both the encoder and the decoder shared across all users. Sentence-level knowledge is transferred.

3. Sentence-level encoder transfer seq2seq [82] (denoted by "ST-E-S2S"): The sequence to sequence model is trained in a multi-task setting with the encoder shared across all users but a different decoder for each target user. Sentence-level knowledge is transferred.

4. Personalized phrase-level transfer seq2seq (denoted by "PT-S2S"): The sequence to sequence model is equipped with the proposed personalized decoder and trained in the multi-task setting. Phrase-level knowledge is transferred.

5. None-transfer HRED [118] (denoted by "HRED"): The HRED model is trained only with data from each individual user, without transfer learning.

6. Sentence-level transfer HRED (denoted by "ST-HRED"): The HRED model is trained in a multi-task setting, while sentence-level knowledge is transferred.

7. Personalized phrase-level transfer HRED (denoted by "PT-HRED"): The personalized HRED model with the proposed personalized decoder is trained in a multi-task setting. Phrase-level knowledge is transferred.

Table 5.1: Statistics of the dataset

	Train Set		Test Set	
Dataset	Users	Dialogues	Users	Dialogues
Simulation	10	50	10	2,000
Real	52	464	52	831

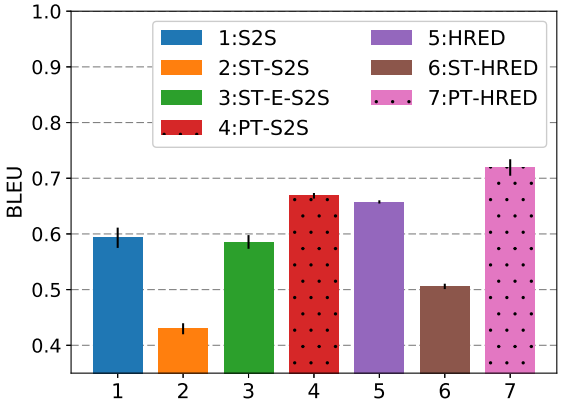
5.3.1 Simulation Experiments

In this section, we conduct experiments on a simulation dataset.

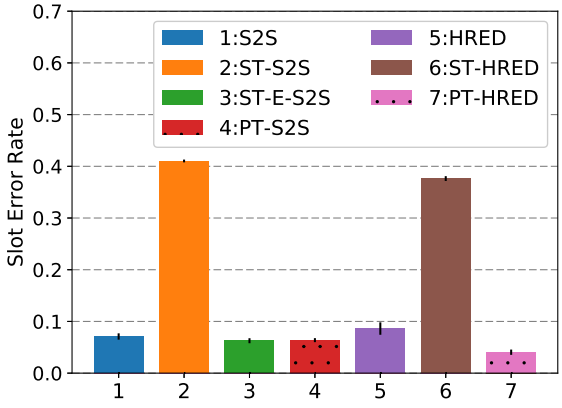
We build a rule-based user simulator for multi-turn coffee ordering services. The user simulator will order his favorite coffee with probability 0.8 and try new coffee with probability 0.2. The simulated user will answer questions about the coffee type, the temperature, the cup size and the delivery address. The user will give rewards according to the reward function defined in the reward section. All models will select an answer with the highest generative probability from a set of answer candidate templates, which is filled with the confirmed information obtained from the simulation user. In order to obtain ground-truth dialogues, we design a rule-based ground-truth agent which will choose the best reply with probability 0.8 and choose a random reply with probability 0.2. We have 10 simulated users in total. For each user, 5 dialogues are collected for training and 200 dialogues are collected for testing. In the training and testing processes, all interior ground-truth information and slot information will not be used.

We use the BLEU [104] and slot-error-rate as the off-line evaluation metric, and use the averaged reward and averaged success rate as the online evaluation metrics in order to fully evaluate our models. The original slot-error-rate is the ratio of the number of wrong-slots and missing-slots over the total number of slots in the given ground-truth structured system action. However, in our setting, the ground-truth structured system action is not given before response generation, thus the missing-slots do not make sense. We modify the definition of slot-error-rate so that it can adapt to our setting. The wrong-slots are defined as the slots not in the simulated user's preference. Hence we define the slot-error-rate as the ratio of the number of wrong-slots over the total number of slots in the generated sentence. Note that a low slot-error-rate doesn't necessarily mean the responses are fluent or appropriate, it only means fewer wrong-slots in the responses. For online testing, each simulated user will generate 1000 coffee orders and the mean and standard deviation of rewards obtained in the dialogues are reported. We run the experiments with 5 different random seeds and we report the mean and standard deviation of the BLEU, slot error rate, averaged rewards and success rate.

All results are shown in Figures 5.5(a)-5.6(b) with details listed in Table 5.2. The none-transfer models, i.e., "S2S" and "HRED", work well because the simulated dialogues are relatively simple. The sentence-level transfer methods "ST-S2S", "ST-E-S2S" and "ST-HRED" perform worst possibly due to the negative transfer. By analyzing the online evaluation records, we found that the sentence-level transfer methods suffer from, for example, using the wrong personal information as

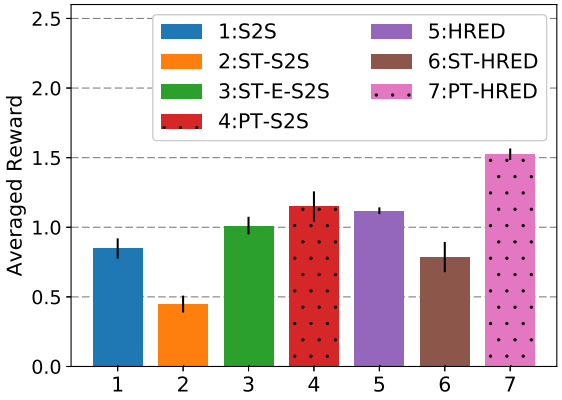


(a) BLEU for the simulation dataset

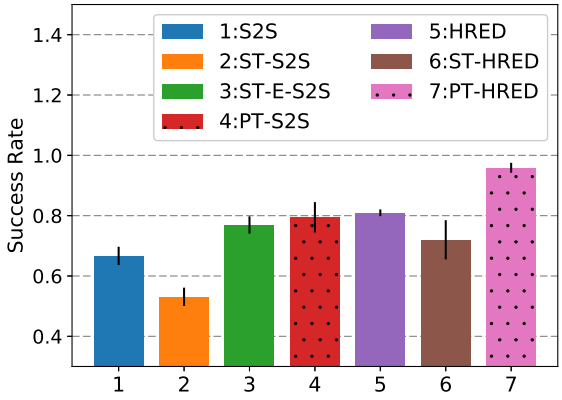


(b) Slot-error-rate for the simulation dataset

Figure 5.5: BLEU and Slot-error-rate for the simulation dataset



(a) Averaged reward for the simulation dataset



(b) Averaged success rate for the simulation dataset

Figure 5.6: Averaged reward and success rate for the simulation dataset

shown in the case study section. Personalized phrase-level transfer methods obtain most of the personal information correctly and hence achieve the highest averaged reward, success rate and BLEU scores. These results show that our personalized decoder can improve baseline models and avoid negative transfer caused by domain differences.

5.3.2 Real-World Experiments

In this section, we conduct experiments on a real-world dataset.

The dataset is a coffee ordering dataset collected from an O2O service in China. The dialogues are conducted between real customers and real waiters via instant messages. The customers are

Table 5.2: Simulation Experiment Results

	Simulation			
	BLEU	Reward	SuccessRate	SlotError
S2S	0.5931 ± 0.0182	0.8478 ± 0.0724	0.6667 ± 0.0302	0.0708 ± 0.0061
ST-S2S	0.4297 ± 0.0099	0.4485 ± 0.0606	0.5310 ± 0.0302	0.4103 ± 0.0026
ST-E-S2S	0.5856 ± 0.0123	1.0121 ± 0.0633	0.7691 ± 0.0285	**0.0627 ± 0.0048**
PT-S2S	**0.6685 ± 0.0050**	**1.1493 ± 0.1086**	**0.7945 ± 0.0503**	0.0638 ± 0.0038
HRED	0.6575 ± 0.0029	1.1187 ± 0.0241	0.8095 ± 0.0112	0.0862 ± 0.0120
ST-HRED	0.5058 ± 0.0048	0.7860 ± 0.1085	0.7199 ± 0.0650	0.3763 ± 0.0046
PT-HRED	**0.7193 ± 0.0148**	**1.5247 ± 0.0413**	**0.9587 ± 0.0164**	**0.0405 ± 0.0051**

frequent shoppers, and the waiters are specially trained. On average there are four turns of interactions in each dialogue. In this dataset, there are 52 users with 8.9 dialogues on average for each user. The statistics of this dataset are listed in Table 5.1.

We also use the BLEU as the evaluation metric. For the BLEU score, 1-gram and 2-gram are used because many sentences are relatively short. Reward and success rate cannot be calculated because policies cannot be run live on static data. For each dialogue turn in each testing dialogue, we randomly generate five system responses, and we calculate the averaged BLEU score over the five responses. We train each model with five different random seeds and report the mean and standard deviation of the scores in the test set.

All results are listed in Figure 5.7 with details listed in Table 5.3. We can see that "S2S" and "HRED" have lowest BLEU score, because the training data from each individual user is insufficient for training a competitive model. The sentence-level transfer methods "ST-S2S", "ST-E-S2S" and "ST-HRED" have higher BLEU score than none-transfer methods, which demonstrates that transfer learning can indeed help improve the performance. The personalized phrase-level transfer methods "PT-HRED" and "PT-S2S" outperform their corresponding sentence-level counterparts, i.e., "ST-S2S", "ST-E-S2S" and "ST-HRED", in terms of BLEU, which demonstrates the effectiveness of the personalized decoder. Specifically, "PT-HRED" improves "ST-HRED" by 7.5% in terms of BLEU. These experimental results again demonstrate that our personalized decoder improves several baseline models and alleviates the negative transfer effect.

5.3.3 Case study

In this section, we show a case to compare a sentence-level transfer model and a phrase-level transfer model to see how personalized decoder avoids negative transfer. As shown in Table 5.4, we can

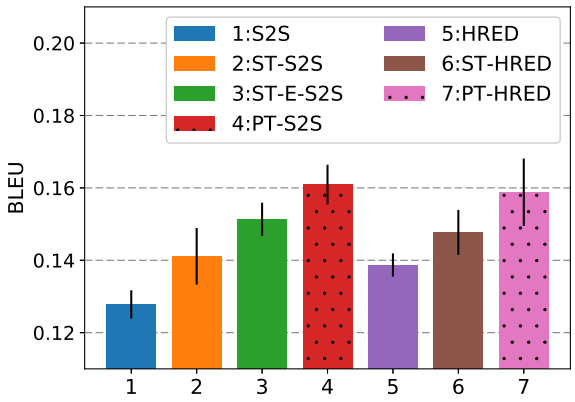


Figure 5.7: BLEU for the real-world dataset

Table 5.3: Real-data Experiment Results

	Real data
	BLEU
S2S	0.1278 ± 0.0039
ST-S2S	0.1411 ± 0.0078
ST-E-S2S	0.1513 ± 0.0046
PT-S2S	$\mathbf{0.1609 \pm 0.0055}$
HRED	0.1387 ± 0.0032
ST-HRED	0.1477 ± 0.0062
PT-HRED	$\mathbf{0.1588 \pm 0.0093}$

see that the sentence-level transfer method "ST-HRED" transfers the wrong personal information from other domains and thereby leads to the failure. On the contrary, "PT-HRED", the phrase-level transfer method, does transfer the correct personal information. As a result, it successfully avoids negative transfer caused by domain differences.

5.4 Conclusion

Training a personalized dialogue system requires a large amount of data, which is usually unavailable from an individual user. The existing seq2seq transfer learning models operate on the entire sentence, while personalized dialogues require finer granularity. We propose a personalized decoder, which can transfer phrase-level information between different users while keeping personalized information intact. The proposed personalized decoder can be easily combined with many models including seq2seq and HRED to achieve knowledge transfer. Experimental results demon-

Table 5.4: A case study by comparing the sentence-level transfer model "ST-HRED" and the phrase-level transfer model "PT-HRED", where the personal words are in bold.

Response Generation Comparison	
User:	I want a cup of coffee.
Truth Agent:	Same as before, **tall hot macchiato** and deliver to **Building No.1, Zhengda Wudaokou Plaza, No.1199 Minsheng Road, Pudong District, Shanghai**?
ST-HRED:	Same as before, tall hot latte and deliver to the Fiyta Building, Science and Technology Part, Shen Zhen?
PT-HRED:	Same as before, **tall hot macchiato** and deliver to **Building No.1, Zhengda Wudaokou Plaza, No.1199 Minsheng Road, Pudong District, Shanghai**?

strate that the phrase-level transfer personalized models improve the BLEU over multiple sentence-level transfer baseline models. We will study dialogue transfer learning under finer-granularity like prefix and stem in the future.

CHAPTER 6

TRANSFER REINFORCEMENT LEARNING THROUGH SIMULTANEOUS SPEECH-ACT AND SLOT ALIGNMENT

Dialogue policy transfer enables us to build dialogue policies in target domains with little data, by leveraging knowledge from a source domain with plenty of data. Dialogue sentences are usually represented by speech-acts and domain slots, and the dialogue policy transfer is usually achieved by assigning a slot mapping matrix based on human heuristics. However, existing dialogue policy transfer methods cannot transfer across dialogue systems with different speech-acts, for example, between systems built by different companies. Also, they depend on either common slots or slot entropy, which are not available in situations when the source and target slots are totally disjoint and full database access is not available to calculate slot-entropy. We propose the Policy tRansfer across dOMaIns and SpEech-acts (PROMISE) model, which is able to transfer dialogue policies across domains with different speech-acts and disjoint slots. The PROMISE model can learn to align different speech-acts and slots simultaneously, and it does not require common slots or slot entropy. Both real-world dialogue data experiments and simulation experiments demonstrate that PROMISE model can effectively transfer dialogue policies across different speech-acts and different domains.

6.1 Introduction

Based on the method, task-oriented dialogue systems can be divided into rule-based and learning-based systems. Learning-based dialogue systems can learn robust dialogue policies from dialogue training data without hand-crafted dialogue decisions made by the human, but a lot of training dialogues are needed to train the dialogue policy. In this chapter, we focus on the dialogue policy learning in multi-turn task-oriented dialogue systems.

Transfer learning can be used to build a dialogue policy on a target domain with limited data, by leveraging the dialogue policy and dialogue data from a different source domain. The sentences in task-oriented dialogue systems are usually represented by speech-acts and domain slots. For example, “I want to book a four-star hotel” can be represented by “inform(type=hotel, stars=4star)”, where “inform” is the speech-act, “type” and “stars” are the slots related to hotel booking domain,

and "hotel" and "4star" are specific slot values for the two domain slots. Many dialogue policy transfer methods have been proposed. Gavsic *et al.* [37, 38] propose to adapt a dialogue policy to an extended domain with additional slots by leveraging common slots and by manually defining a slot similarity function. Gavsic *et al.* [36] propose to calculate a normalized entropy for each source and target slot, and then build a cross-domain slot similarity matrix by aligning the source and target slots with similar normalized entropy.

However, existing dialogue policy transfer methods cannot transfer across dialogue systems built with different speech-acts, e.g., between systems built by different companies. The speech-act "inform" in a dialogue system might correspond to the speech-act "tell" or even "Action1" in another dialogue system. Moreover, existing dialogue transfer methods do not work in situations when the source and target domains do not have common slots and when full database access is not available to calculate the normalized entropy for each slot.

We propose the Policy tRansfer across dOMaIns and SpEech-acts (PROMISE) model, which is able to transfer dialogue policies across different domains and speech-acts. The proposed PROMISE model jointly learns a cross-system speech-act similarity matrix and a cross-domain slot similarity matrix, with plenty of source training dialogues and a few target dialogue data. In addition, the proposed algorithm does not require common slots or full database access to calculate the normalized entropy for each slot. Extensive simulations and real-world experiments show that the PROMISE model can effectively transfer dialogue policies across domains with different speech-acts and disjoint slots.

The contributions of this chapter are three folds:

1. We define and formulate the simultaneously cross speech-act and cross-domain dialogue policy transfer problem.

2. We propose a novel transfer learning model, the PROMISE, for the problem. The PROMISE model can transfer across different domain slots and different speech-acts, and it does not require common slots or a full database access to calculate the normalized entropy for each slot.

3. We conduct simulation experiments and collect a real-world cross-domain dialogue dataset to evaluate the proposed algorithm, the experimental results show that the proposed algorithm can effectively transfer dialogue policies across domains and speech-acts.

6.1.1 Challenges

To transfer dialogue policy across domains and systems, we have to overcome several challenges.

1. The source domain and the target domain have different sets of slots. Traditional cross-domain policy transfer methods require a human predefined cross-domain slot mapping matrix, which might be inaccurate and require a lot of human effort. Some methods propose to calculate a normalized entropy for each source/target slot in the whole entity database, but a full access to the whole entity database might not be accessible in many applications. We need to learn the cross-domain similarity matrix with only a small number of dialogue records in the target domain.

2. The source domain and the target domain have different speech-acts. Existing methods only consider the policy transfer across domains in the same dialogue system with the same set of speech-acts. However, if the source domain dialogue policy is built by a different group with different speech-acts, existing methods fail to transfer. The semantic meaning of dialogue states and dialogue actions are determined by the combination of speech-acts and domain slots, which makes the learning more complicated. In order to transfer dialogue policy in this scenario, we need to learn the cross-domain speech-act mapping matrix simultaneously with the slot mapping matrix, with only a small number of target domain dialogue records.

6.1.2 Related Work of Cross-Domain Dialogue Policy Transfer

In this section we introduce the related works.

Transfer learning [102] has been used in dialogue systems to solve the data sparsity problem in spoken language understanding, dialogue state tracking, dialogue policy learning and natural language generation. In this chapter, we focus on dialogue policy transfer learning problems on the multi-turn task-oriented dialogue systems. Gavsic *et al.* [37, 38] propose to adapt a dialogue policy to an extended domain with additional slots with policy fine-tuning and using source domain dialogue policy as prior. In this case, the source domain and the target domain should have a considerable portion of common slots, or the similarity value between new slots and old slots have to be manually assigned. Gavsic *et al.* [36] propose to build a cross-domain slot similarity matrix by using a normalized entropy for each slot, and matching the source slots and target slots according to the normalized entropy of each slot. However, the normalized entropy for each slot cannot be directly calculated from the training dialogue, because an additional full access to the

whole databases in the source domain and target domain is needed. Moreover, existing dialogue policy transfer learning methods cannot transfer across dialogue systems built with a different set of speech-acts.

Transfer learning techniques for reinforcement learning [141] has been used in various domains, but most methods require a predefined state mapping or action mapping assigned by human experts. Taylor *et al.* [140] propose to test all possible state and action mapping off-line one by one and choose the mapping that can minimize prediction error of target model on the source data. However, the number of all possible state and action mapping grows exponentially in the number of actions and slots, so this algorithm is computationally inefficient and impractical for real-world task-oriented dialogue systems.

In contrast to the above mentioned related works, the proposed PROMISE model is able to transfer dialogue policies across domains with different slots and across systems with different speech-acts, and it does not require the source and target tasks to share common slots or the full access to the whole database to calculate entropy. And our algorithm is optimized via convex optimization, which is computationally efficient.

6.2 Model: PROMISE

In this section, we introduce the dialogue model. Firstly we briefly introduce the single domain dialogue system, then we introduce the proposed PROMISE policy transfer model.

6.2.1 Notation

In this thesis, matrices are denoted in a bold capital case, column vectors are in a bold lower case, and scalars are in a lower case. The utterance of the user is denoted as X, and the utterance of the agent is denoted as Y. A multi-turn dialogue record is denoted as $\{X_n, Y_n, r_n\}_{n=0}$, where r_n is the immediate reward for the n-th dialogue turn.

6.2.2 Problem Definition

The problem is illustrated in Figure 6.1. Given a target domain with a few training dialogues and a source domain with plenty of training dialogues, the problem is to learn a dialogue policy in the target domain by leveraging knowledge in the source domain. Note that in this problem, the

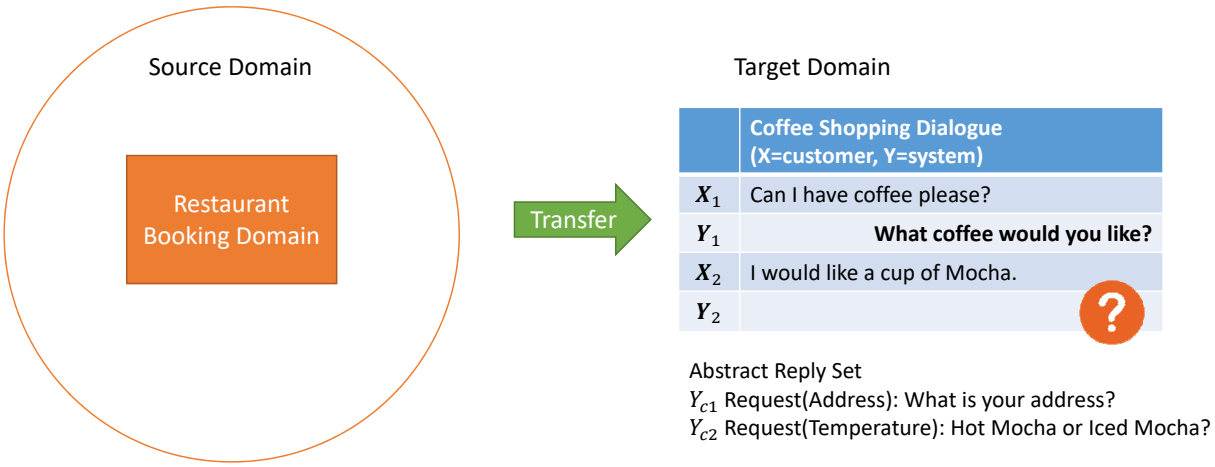


Figure 6.1: The problem illustration

speech-acts and domain slots in the source domain and target domain can be completely different, and no external database is available besides the training dialogues. The input of the problems are:

1. Plenty of source domain training dialogues $\mathcal{D}^s = \{X_n^s, Y_n^s, r_n\}_{n=0}$.
2. A few target domain training dialogues $\mathcal{D}^t = \{X_n^t, Y_n^t, r_n\}_{n=0}$.

The output of the problems is:

1. A dialogue policy π^t for the target domain.

6.2.3 Different Abstractions of Sentence and Dialogue State

In order to let the model have better generalization ability, there is three levels of abstraction for the sentences and the dialogue states in the PyDial [150] package. The details are summarized in Table 6.1.

1. **Original sentences and dialogue states.** This is the original input of the dialogue system, where each sentence is represented by a sequence of words. The user utterance in the n-th dialogue turn is notated by X_n and the agent reply is notated by Y_n. The dialogue state for the n-th agent reply Y_n is notated by

$$H_n = \{\{X_k, Y_k\}_{k=1}^{n-1}, X_n\}, \tag{6.1}$$

which is the collection of all previous user utterances and agent replies in the current dialogue session. For example, X_1="I want to find an expensive hotel".

Table 6.1: Different Levels of abstraction for sentences and dialogue states.

Abstraction Level	Notation	Description	Example
Original	X_n, Y_n, H_n	$H_n = \{\{X_k, Y_k\}_{k=1}^{n-1}, X_n\}$	X_1="I want to find an expensive hotel"
Abstracted	$\tilde{X}_n, \tilde{Y}_n, \tilde{H}_n$	$\tilde{X}_n = \{a_n, \{s_k = v_k\}_{k=1}\}$, $\tilde{Y}_n = \{a'_n, \{s'_k = v'_k\}_{k=1}\}$, $\tilde{H}_n = \{l_{db,n}, a_n, \{\tilde{s}_k, \tilde{v}_k\}, \{\hat{s}_k, \hat{v}_k\}\}$	$\tilde{X}_1$="inform(type=hotel, price=expensive)"; $\tilde{H}_1 = \{l_{db,n} = 10, a_1 = \text{inform}, \{\tilde{s}_1 = \text{price}, \tilde{v}_1 = \text{expensive}\}, \{\}\}$
Summary	$\mathbf{x}_n, \mathbf{y}_n, \mathbf{h}_n$	$\mathbf{x}_n = [\mathbf{a}_n, \mathbf{s}_n]$, $\mathbf{y}_n = [\mathbf{a'}_n, \mathbf{s'}_n]$, $\mathbf{h}_n = [\mathbf{l}_{db,n}, \mathbf{a}_n, \tilde{\mathbf{s}}_n, \hat{\mathbf{s}}_n]$	$\mathbf{x}_1 = [[0,0,0,1],[0,1,0,0]]$; $\mathbf{h}_1 = [[10],$ $[0,0,0,1],[0,1,0,0],[0,0,0,0]]$

2. **Abstracted sentences and states.** This is the first abstraction of the dialogue system, which is used in the spoken language understanding module, the dialogue state tracking module and the natural language generation module. In this abstraction level, each sentence is represented by a speech-act a, a collection of slots s and their slot-values v. An abstracted user utterance is denoted by

$$\tilde{X}_n = \{a_n, \{s_k = v_k\}_{k=1}\} \tag{6.2}$$

and an abstracted agent reply is denoted by

$$\tilde{Y}_n = \{a'_n, \{s'_k = v'_k\}_{k=1}\}. \tag{6.3}$$

The abstracted dialogue state for the i-th agent reply is denoted by

$$\tilde{H}_n = \{l_{db,n}, a_n, \{\tilde{s}_k, \tilde{v}_k\}, \{\hat{s}_k, \hat{v}_k\}\}, \tag{6.4}$$

where $l_{db,n}$ is the number of entities that match all requirements, a_n is the latest user speech-act, $\{\tilde{s}_k, \tilde{v}_k\}$ is the collection of constraints obtained from the user and $\{\hat{s}_k, \hat{v}_k\}$ is the collection of questions asked by users about slots $\hat{s}$. For example, X_1="I want to find an expensive hotel" can be represented by $\tilde{X}_1$="inform(type=hotel, price=expensive)", and $\tilde{H}_1 = \{l_{db} = 10, a_1 = \text{inform}, \{\tilde{s}_1 = \text{price}, \tilde{v}_1 = \text{expensive}\}, \{\}\}$.

3. **Summary sentences and states.** This is the highest abstraction level [144, 183] of the dialogue system used inside the dialogue policy module. In this abstraction level, each sentence is represented by a speech-act one-hot vector $\mathbf{a}_n$ and a slot one-hot vector $\mathbf{s}_n$ without the slot value. Compared with the abstracted sentence and abstracted dialogue state, there is no slot-value in the summary sentence and summary dialogue state, since the dialogue policy can select an appropriate response by knowing which slot is constrained without knowing the

exact slot value. A summary user utterance is notated by

$$\mathbf{x}_n = [\mathbf{a}_n, \mathbf{s}_n] \tag{6.5}$$

and a summary agent reply is denoted by

$$\mathbf{y}_n = [\mathbf{a'}_n, \mathbf{s'}_n], \tag{6.6}$$

where $\mathbf{a}_n$ is the one-hot vector for the speech-act and $\mathbf{s}_n$ is the one-hot vector for the corresponding slot. The summary dialogue state for the i-th agent reply is denoted by

$$\mathbf{h}_n = [\mathbf{l}_{db,n}, \mathbf{a}_n, \tilde{\mathbf{s}}_n, \hat{\mathbf{s}}_n], \tag{6.7}$$

where $\mathbf{l}_{db,n}$ is a one-hot database state vector indicating the number of entities matching the current user's requirements, $\mathbf{a}_n$ is the one-hot vector of the latest user speech-act, $\tilde{\mathbf{s}}_n$ is a vector indicating whether the user has provided information about each slot so far, and $\hat{\mathbf{s}}_n$ is a vector indicating whether the user has request information about each slot. For example, X_1="I want to find an expensive hotel" can be represented by $\mathbf{x}_1 = [[0,0,0,1],[0,1,0,0]]$, and $\mathbf{h}_1 = [[10],[0,0,0,1],[0,1,0,0],[0,0,0,0]]$.

6.2.4 Single Domain Dialogue System

We briefly introduce the single domain dialogue system we used to generate the agent reply Y_n on each dialogue round based on the dialogue state H_n, and how to update the dialogue state with the previous user utterance X_n. We build our model-based on the PyDial [150] package.

The spoken language understanding (SLU) module transforms a user utterance X into the abstracted user utterance $\tilde{X}_n = \{a_n, \{s_k = v_k\}_{k=1}\}$, and it is based on the regular expression for speed and easy of use. It can be written as

$$\tilde{X}_n = \text{SLU}(X_n). \tag{6.8}$$

The dialogue state tracking (DST) module tracks the current dialogue state $\tilde{H}_n$, based on the previous dialogue state $\tilde{H}_{n-1}$, the previous abstracted system utterance $\tilde{Y}_{n-1}$ and abstracted user utterance $\tilde{X}_n$. The current dialogue state tracking module are built with hand-crafted rules and it can be written as

$$\tilde{H}_n = \text{DST}(\tilde{H}_{n-1}, \tilde{Y}_{n-1}, \tilde{X}_n). \tag{6.9}$$

The dialogue policy model (DPL) can choose the agent reply $\tilde{Y}_n$ to maximize future total expected reward, $\tilde{Y}_n = \pi(\tilde{H}_n)$. The learned Q-function estimates the expected future reward for each possible agent reply candidate $\tilde{Y}_n$ under the current state, and the agent reply are sampled from these

candidates according to their Q-value. In this chapter, we use the Gaussian process based dialogue policy. But the proposed knowledge transfer algorithm can work for any dialogue policy based on Q-function. The policy can be written as

$$\tilde{Y}_n = \arg\max_{\tilde{Y}_j} Q(\tilde{H}_n, \tilde{Y}_j). \tag{6.10}$$

The natural language generation (NLG) module converts an abstracted agent reply in speech-act slot representation $\tilde{Y}_n$ into a fluent sentence Y_n. The current template-based natural language generation can be written as

$$Y_n = \text{NLG}(\tilde{Y}_n). \tag{6.11}$$

6.2.5 The PROMISE policy transfer model

In this section, we introduce the PROMISE policy transfer model. The PROMISE model can learn to map dialogue states and candidate agent replies from the target domain to the source domain, and then leverage the Q-function in the source domain.

The PROMISE policy transfer framework can be written as

$$Q^t(\mathbf{h}^t, \mathbf{y}^t) = Q^s(f_h^{t2s}(\mathbf{h}^t), f_y^{t2s}(\mathbf{y}^t)), \tag{6.12}$$

where $Q^s(\mathbf{h}^s, \mathbf{y}^s)$ is the source domain dialogue policy, $f_h^{t2s}(\mathbf{h}^t)$ is the cross-domain state translation function, $f_y^{t2s}(\mathbf{y}^t)$ is the cross-domain sentence translation function, which will be detailed in the following section 6.2.5. The source domain dialogue policy can be any dialogue policy, in this chapter we utilize the popular Gaussian process dialogue policy [41].

Cross-domain state and action mapping

Before defining the cross-domain state translation function $f_h^{t2s}(\mathbf{h}^t)$ and the cross-domain sentence translation function $f_y^{t2s}(\mathbf{y}^t)$, we need to firstly define the basic cross-domain speech-act mapping function $f_a^{t2s}(\mathbf{a}^t)$ and the cross-domain slot mapping function $f_s^{t2s}(\mathbf{s}^t)$.

The cross-domain speech-act mapping function is defined as follows. The source domain speech-act vector is denoted by $\mathbf{a}^s$ and the target domain speech-act vector is denoted by $\mathbf{a}^t$. We use A^s to denote the set of all source speech-acts and A^s to denote the set of all target domain speech-acts. We assume a d dimension embedding vector for each speech-act in the source and the target domains, so the source domain speech-act embedding matrix is $\mathbf{E}^{A^s} \in \mathbb{R}^{|A^s| * d}$ and the target

domain speech-act embedding matrix is $\mathbf{E}^{A^t} \in \mathbb{R}^{|A^t|*d}$. So the speech-act similarity matrix from target domain to source domain is defined as

$$\mathbf{M}_a^{t2s} = \text{softmax}(\mathbf{E}^{A^t}(\mathbf{E}^{A^s})^T), \tag{6.13}$$

where the element in the i-th row j-th column is the speech-act similarity between the i-th target domain speech-act and the j-th source speech-act, the superscript ^{T} denotes the transpose of a vector or matrix, softmax() denotes the column-wise softmax function making each column sums to 1. Given a speech-act vector in the target domain $\mathbf{a}^t$, the translated source speech-act vector is

$$\mathbf{a}^s = f_a^{t2s}(\mathbf{a}^t) = \mathbf{a}^t\mathbf{M}_a^{t2s}. \tag{6.14}$$

Similarity, the speech-act similarity matrix from the source domain to the target domain is defined as

$$\mathbf{M}_a^{s2t} = \text{softmax}(\mathbf{E}^{A^s}(\mathbf{E}^{A^t})^T), \tag{6.15}$$

and the cross-domain speech-act mapping function from the source domain to the target domain is defined as

$$\mathbf{a}^t = f_a^{s2t}(\mathbf{a}^s) = \mathbf{a}^s\mathbf{M}_a^{s2t}. \tag{6.16}$$

The cross-domain slot mapping function is defined as follows. The source domain slot vector is denoted by $\mathbf{s}^s$ and the target domain slot vector is denoted by $\mathbf{s}^t$. We use S^s to denote the set of all source domain slots and S^t to denote the set of all target domain slots. We have a d dimension embedding vector for each slot in the source and the target domains, so the source domain slot embedding matrix is $\mathbf{E}^{S^s} \in \mathbb{R}^{|S^s|*d}$ and the target domain slot embedding matrix is $\mathbf{E}^{S^t} \in \mathbb{R}^{|S^t|*d}$. So the slot similarity matrix from the target domain to the source domain is defined as

$$\mathbf{M}_s^{t2s} = \text{softmax}(\mathbf{E}^{S^t}(\mathbf{E}^{S^s})^T), \tag{6.17}$$

where the element in the i-th row j-th column is the slot similarity between the i-th target domain slot and the j-th source slot, the superscript ^{T} denotes the transpose of a vector or matrix, softmax() denotes the column-wise softmax function making each column sums to 1. Given a slot vector in the target domain $\mathbf{s}^t$, the translated source slot vector is

$$\mathbf{s}^s = f_s^{t2s}(\mathbf{s}^t) = \mathbf{s}^t\mathbf{M}_s^{t2s}. \tag{6.18}$$

Similarly, the slot similarity matrix from the source domain to the target domain $\mathbf{M}^{s2t}$ is defined as

$$\mathbf{M}_s^{s2t} = \text{softmax}(\mathbf{E}^{S^s}(\mathbf{E}^{S^t})^T), \tag{6.19}$$

and the cross-domain slot mapping function from the source domain to the target domain is defined as

$$\mathbf{s}^t = f_s^{s2t}(\mathbf{s}^s) = \mathbf{s}^s \mathbf{M}_s^{s2t}. \tag{6.20}$$

Now we are ready to define the cross-domain summary state translation function. The source domain summary state vector is denoted by $\mathbf{h}^s$ and the target domain summary state vector is denoted by $\mathbf{h}^t$. As the source domain summary state $\mathbf{h}^s$ can be expressed via speech-act one-hot vector and domain one-hot slots as

$$\mathbf{h}^s = [\mathbf{l}_{db}, \mathbf{a}^s, \tilde{\mathbf{s}^s}, \hat{\mathbf{s}^s}], \tag{6.21}$$

and the corresponding target summary state $\mathbf{h}^t$ can be expressed as

$$\mathbf{h}^t = [\mathbf{l}_{db}, \mathbf{a}^t, \tilde{\mathbf{s}^t}, \hat{\mathbf{s}^t}], \tag{6.22}$$

where the $\mathbf{l}_{db}$ is the vector representing the number of entities matching the current requirements in the database. The cross-domain state mapping function from the target domain to the source domain $f_h^{t2s}(\mathbf{h}^t)$ can be defined as

$$\mathbf{h}^s = f_h^{t2s}(\mathbf{h}^t), \tag{6.23}$$

$$f_h^{t2s}(\mathbf{h}^t) = [\mathbf{l}_{db}, f_a^{t2s}(\mathbf{a}^t), f_s^{t2s}(\tilde{\mathbf{s}^t}), f_s^{t2s}(\hat{\mathbf{s}^t})], \tag{6.24}$$

where the $\mathbf{l}_{db}$ in both the source and the target domains have the same format since it is the number of entities, $f_a^{t2s}()$ is the cross-domain speech-act translation function and $f_s^{t2s}()$ is the cross-domain slot translation function defined above.

The cross-domain sentence translation function can be also defined similarity. The source domain summary sentence vector $\mathbf{y}^s$ can be expressed via a speech-act one-hot vector and a slot one-hot vector as

$$\mathbf{y}^s = [\mathbf{a}^s, \tilde{\mathbf{s}}^s], \tag{6.25}$$

and the corresponding target domain summary sentence vector $\mathbf{y}^t$ can be expressed as

$$\mathbf{y}^t = [\mathbf{a}^t, \tilde{\mathbf{s}}^t]. \tag{6.26}$$

The cross-domain summary sentence mapping function can be defined as

$$\mathbf{y}^s = f_y^{t2s}(\mathbf{y}^t), \tag{6.27}$$

$$f_y^{t2s}(\mathbf{y}^t) = [f_a^{t2s}(\mathbf{a}^t), f_s^{t2s}(\tilde{\mathbf{s}}^t)]. \tag{6.28}$$

6.2.6 Parameter Learning

In this section, we introduce the learning objective and how the model is trained.

Reward Function

The reward function is defined in the PyDial [150] package. For each dialogue turn, the agent receives an immediate reward of -1 to punish longer dialogues. If the system informs the user an entity that matches his requirements or there is no such entity, then the dialogue is considered successful, and the agent will get a final reward of 20. The successful reward will only be observable after the whole dialogue is finished.

Loss Function

In this chapter, we combine the Q-learning objective with a series of regularization terms as the final loss function. The loss function is defined as

$$L(\Theta) = \mathbb{E}[r_n + \gamma \max_{\mathbf{y}'} Q^t(\mathbf{h}_{n+1}, \mathbf{y}') - Q^t(\mathbf{h}_n, \mathbf{y}_n)]^2 + \mathcal{R}(\Theta), \tag{6.29}$$

where $\mathcal{R}(\Theta)$ is the regularization term. We optimize the model with the Adam [61] algorithm.

Regularization

Regularization terms of the cross-domain speech-act mapping and the cross-domain slot mapping functions include

1. Since different speech-acts will co-occur with different slots, the first term regularizes that similar speech-acts should only appear with similar slots. Firstly, we need to model the probability of the slot vector $\mathbf{s}$ conditioned on the speech-act vector $\mathbf{a}$. We build a logistic regression prediction function on both the source and the target domains, and they are defined as

$$\mathbf{s}^s = c^s(\mathbf{a}^s), \tag{6.30}$$

$$\mathbf{s}^t = c^t(\mathbf{a}^t), \tag{6.31}$$

where $c^s()$ is a logistic regression prediction function trained in target domain to predict the slot vector $\mathbf{s}^s$ based on the speech-act vector $\mathbf{a}^s$ and $c^t()$ is a prediction function trained in target domain. The logistic regression prediction function $c^s()$ and c^t are trained by minimizing

the objective function

$$\mathcal{R}_{c^s}(\Theta) = \frac{1}{|\mathcal{D}^s|} \sum_{\{\mathbf{a}^s, \mathbf{s}^s\} \in \mathcal{D}^s} L_{ce}(c^s(\mathbf{a}^s), \mathbf{s}^s), \tag{6.32}$$

and

$$\mathcal{R}_{c^t}(\Theta) = \frac{1}{|\mathcal{D}^t|} \sum_{\{\mathbf{a}^t, \mathbf{s}^t\} \in \mathcal{D}^t} L_{ce}(c^t(\mathbf{a}^t), \mathbf{s}^t). \tag{6.33}$$

Then we are ready to define the cross-domain slot vector preservation loss. For every sentence $[\mathbf{a}^s, \mathbf{s}^s]$ in the source domain, $f_a^{s2t}(\mathbf{a}^s)$ are the corresponding similar speech-act vector in the target domain, and $(c^t \circ f_a^{s2t})(\mathbf{a}^s)$ is the predicted compatible slot vector in the target domain. The actual slot vector $\mathbf{s}^s$ should be similar with the slot vectors projected from the target domain, and the cross-domain slot vector preservation loss can be defined as

$$\mathcal{R}_{1s}(\Theta) = \frac{1}{|\mathcal{D}^s|} \sum_{\{\mathbf{a}^s, \mathbf{s}^s\} \in \mathcal{D}^s} L_{ce}((f_s^{t2s} \circ c^t \circ f_a^{s2t})(\mathbf{a}^s), \mathbf{s}^s) \tag{6.34}$$

where $L_{ce}()$ is the cross-entropy loss and $\circ$ is the functional composition. Similarly, for every sentence $[\mathbf{a}^t, \mathbf{s}^t]$ in the target domain, the cross-domain slot vector preservation loss can be defined as

$$\mathcal{R}_{1t}(\Theta) = \frac{1}{|\mathcal{D}^t|} \sum_{\{\mathbf{a}^t, \mathbf{s}^t\} \in \mathcal{D}^t} L_{ce}((f_s^{s2t} \circ c^s \circ f_a^{t2s})(\mathbf{a}^t), \mathbf{s}^t) \tag{6.35}$$

where $L_{ce}()$ is the cross-entropy loss and $\circ$ is the functional composition. The cross-domain slot vector preservation loss in both domains can be defined as

$$\mathcal{R}_1(\Theta) = \mathcal{R}_{1s}(\Theta) + \mathcal{R}_{1t}(\Theta). \tag{6.36}$$

2. Since there are a lot of data in the source domain, we assume the user's conditional reaction speech-act follow a specific distribution, and we can predict the user reaction to each dialogue action. With a good cross-domain mapping, the target user should react similarly to the target system action. The source domain user speech-act prediction function can be trained by minimizing the following loss function

$$L_{c_u^s}(\theta) = \frac{1}{|\mathcal{D}^s|} \sum_{\mathbf{x}_n = \{\mathbf{a}_n, \mathbf{s}_n\}, \mathbf{y}_{n-1} \in \mathcal{D}^s} L_{ce}(c_u^s(\mathbf{y}_{n-1}; \theta), \mathbf{a}_n), \tag{6.37}$$

$$\theta = \arg\min_{\theta'} L_{c_u^s}(\theta'), \tag{6.38}$$

where $c_u^s()$ is a logistic regression prediction function trained in the source domain to predict the user's next speech-act $\mathbf{a}_n$ based on the system reply $\mathbf{y}_{n-1}$ and θ is the parameter of $c_u^s()$. The cross-domain user prediction regularization can be expressed as

$$\mathcal{R}_2(\Theta) = \frac{1}{|\mathcal{D}^t|} \sum_{\{\mathbf{x}_n=\{\mathbf{a}_n,\mathbf{s}_n\},\mathbf{y}_{n-1}\}\in\mathcal{D}^t} L_{ce}((f_a^{s2t} \circ c_u^s \circ f_y^{t2s})(\mathbf{y}_{n-1}), \mathbf{a}_n), \tag{6.39}$$

where $\circ$ is the functional composition.

3. Different speech-acts have different occurrence probability and similar speech-acts should have similar occurrence probabilities. We define a cross-domain frequency loss function of both user speech-act and the agent speech-act as

$$\mathcal{R}_3(\Theta) = L_{KL}(f_a^{t2s}(\mathbf{p}_a^t), \mathbf{p}_a^s), \tag{6.40}$$

where $\mathbf{p}_a^t$ is the occurrence probability of target domain speech-acts, $\mathbf{p}_a^s$ is the occurrence probability of source domain speech-acts and L_{KL} is the KL-divergence.

4. One-to-one slot matching is good when source domain and target domain have a similar number of slots, but when two domains have a different number of slots, some slots in the source/target domain do not have a corresponding cross-domain slot. In this case, multiple source/target domain dialogue states might correspond to the same transferred dialogue state, which leads to bad performance. To encourage the slot mapping matrix to have more uniform slot mapping, so that the changes of a state in target domain would affect the state vector in the source domain, we encourage uniform slot mapping, and the regularization can be defined as

$$\mathcal{R}_4(\Theta) = \sum_j (\sum_i p_{s_i^s, s_j^t} - \frac{1}{|S^t|})^2, \tag{6.41}$$

where $p_{s_i^s, s_j^t}$ is the probability of mapping the source slot s_i^s to the target slot s_j^t.

6.3 Experiments

In this section, we conduct experiments on a simulation dataset and a real-world dataset to verify the effectiveness of the proposed algorithm.

6.3.1 Baselines

We compare the proposed PROMISE model with the following baseline methods.

1. None-Transfer method (denoted by "NoneTL"), which utilize only the target domain dataset to train a dialogue policy. The dialogue policy is trained with the Gaussian process dialogue policy.

2. Random Speech-act Mapping and Entropy slot-matching (denoted by "RAFS"), which has a random speech-act mapping and an entropy-based cross-domain slot matching [36].

3. Learned Speech-act Mapping and Entropy slot-matching (denoted by "LAFS"), which has a learned speech-act mapping and an entropy-based slot-matching matrix.

4. Perfect speech-act mapping and entropy-based slot-matching (denoted by "FAFS"), which has a ground-truth speech-act mapping and the entropy-based slot-matching. This method is used to demonstrate the performance upper bound of the transfer learning with a fixed slot-matching.

5. Perfect speech-act mapping and learned slot-matching (denoted by "FALS"), which has a ground-truth speech-act mapping matrix, and the slot-matching is learned with the proposed algorithm. This method is used to demonstrate the upper bound of transfer learning when the speech-act mapping is known and the slot-matching is learned from the data.

6.3.2 The Source and Target Domains

In the experiments, a dialogue policy in the Cambridge restaurants booking domain (denoted as "CamRestaurants") is transferred to the target Cambridge hotel booking domain (denoted as "CamHotels"). The CamRestaurants domain is a restaurant booking dialogue system in Cambridge, and the CamHotels domain is a hotel booking dialogue system in Cambridge. The CamRestaurants domain has 9 slots including 4 informable slots which can be used to constrain the scope of search and 5 other slots which can only be asked after an entity is identified, for example, price is a informable slot since customer can ask for a cheap restaurant, but phone is only requestable, because few customers will say "I want to find a restaurant with this phone number". The CamHotels domain has 6 informable slots and 5 other slots. The detailed speech-acts and slots are listed in Table 6.2.

Note that the ground-truth speech-act and slot mapping are not available, for example, the algorithms do not know "ack" in the source domain corresponds to "ack" in the target domain, because the source and target domains systems could have used different sets of speech-acts. Transferring

Table 6.2: The speech-acts and slots in the source and target domains dataset. The ground truth speech-act and slot mapping are not available, for example, the algorithms do not know "ack" in the source domain corresponds to "ack" in the target domain, because the source and target domain systems could have used different sets of speech-acts. Transferring dialogue policies across domains with different speech-acts requires to learn the cross-domain speech-act mapping and slot mapping from the training data.

	Speech-acts		Slots	
Domains	Belief	Other	Informable	Other
Source: CamRestaurants	ack, bye, hello, none, repeat, silence, thankyou	affirm, confirm, confreq, deny, inform, inform_alternatives, inform_byname, negate, null, reqalts, reqmore, request, restart, select	food, pricerange, name, area	addr, phone, postcode, signature, description
Target: CamHotels	ack, bye, hello, none, repeat, silence, thankyou	affirm, confirm, confreq, deny, inform, inform_alternatives, inform_byname, negate, null, reqalts, reqmore, request, restart, select	kind, name, area, hasparking, pricerange, stars	phone, addr, postcode, hasparking, hasinternet

dialogue policies across domains with different speech-acts requires learning the cross-domain speech-act mapping and slot mapping from the training data.

6.3.3 Simulation

In this section, we conduct a simulation experiment with the user simulator in PyDial [150] package.

Setting

In this simulation experiment, we use the user simulator in the Pydial package to train and test the dialogue policies. In each dialogue, the simulator built within the package randomly generates a set of requirements and the simulator tries to ask for an entity that matches all requirements. If an entity matching all requirements is suggested by the dialogue system within 20 dialogue turns, then the task is considered successful, and the agent will get a successful reward of 20 after the dialogue ends. For each dialogue turn, a slight punishment of -1 will be given to the agent to encourage shorter dialogues.

Table 6.3: The number of dialogues used in each experiment is shown in the table. The simulation is repeated with 10 different random seeds, and the real-world experiment is repeated for 5 times with 5 sets of target domain training data. Each set of real-world target domain training data have 20 dialogues, so the total number of target domain training dialogues is 100.

	Training		Testing
Domain	Source: CamRestaurants	Target: CamHotels	Target: CamHotels
Simulation (Only Total Reward)	1000	1-100	Live: 300
Real (Immediate Reward)	40	1-20	Static: 60 Live: 300

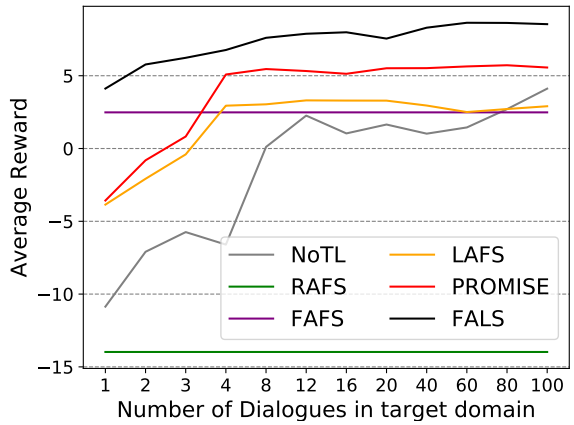
In the simulation, only the final total reward will be used to train the dialogue policy and no immediate rewards will be used. This is the setting where less labelling effort is needed for each dialogue(only the final reward), but in this case more training dialogues are needed to train a good dialogue policy. There are 1000 training dialogues in the source domain, and we vary the number of training dialogues in the target domain from 1 to 100. The detailed statistics of the simulation dataset is listed in Figure 6.3.

Evaluation

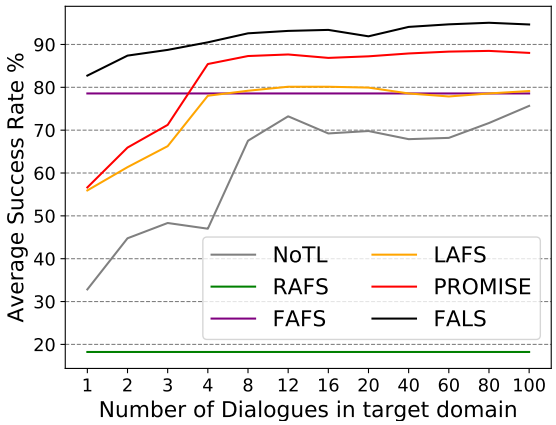
We use the averaged reward, the success rate and the number of dialogue turns to evaluate the performance of the dialogue system. We run the experiments with 10 different random seeds and we report the averaged performance. The averaged reward is the mean of all rewards obtained by the simulated user while interacting with the agent being tested, the success rate measures the probability that the simulated user successfully orders a cup of coffee within 20 turns, and the number of dialogue turns is the average number of dialogue turns before the simulated user can successfully complete an order.

Results

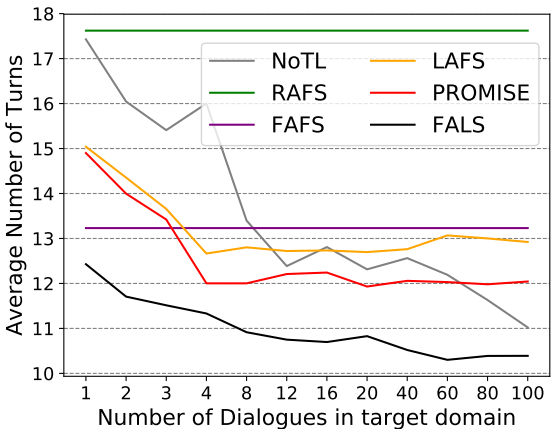
The averaged reward and averaged success rate are shown in Figure 6.2. The performance of "NoneTL" increases slowly, since it requires a large number of data to train a dialogue policy. The "RAFS" fails, meaning it is very necessary to find a reasonable speech-act mapping. As we can see, the proposed PROMISE model out-performs other baselines "LAFS","RAFS" significantly, which demonstrates learning speech-act and slot-mapping simultaneously is highly necessary and



(a) Averaged Reward for the simulation data, higher is better.



(b) Averaged Success Rate for the simulation data, higher is better.



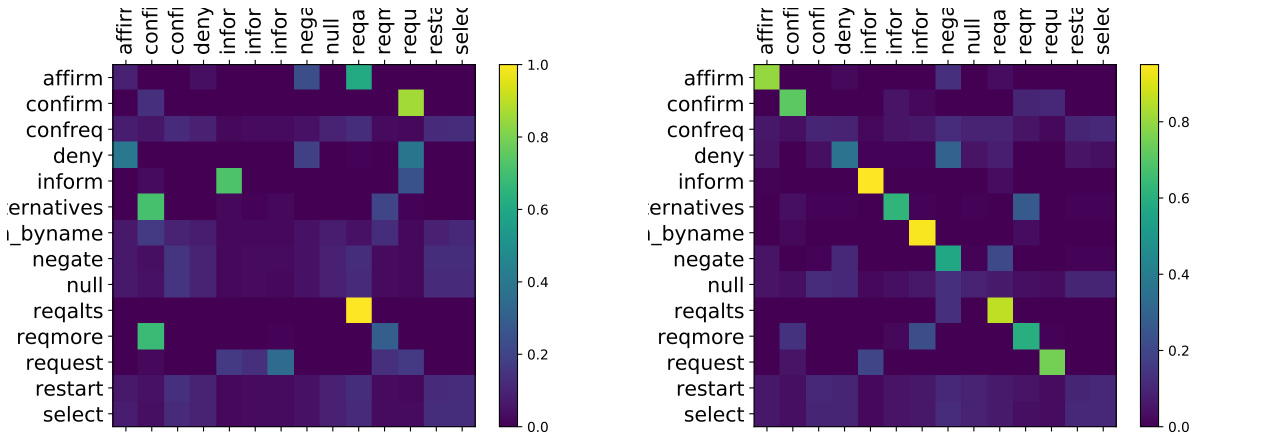
(c) Averaged Length for the simulation data, lower is better.

Figure 6.2: Experiment results for the simulation dataset. The proposed method is "PROMISE", the methods "FALS" and "FAFS" are the performance upper-bound since they use the ground truth speech-act mapping.

can boost the performance when there is no enough data for the target domain. The methods with ground-truth speech-act shows the upper-bound of the dialogue transfer policy, "FALS" gradually out-performs "FAFS", which shows that it is necessary to learn slot-mapping matrix. The propose PROMISE performs very similarly with the ground-truth "FALS", which demonstrate the effectiveness of the proposed PROMISE algorithm. The PROMISE is better than "LAFS" with a heuristic slot-mapping and an additional database, which shows that learning slot mapping is better than using human heuristics.

Result Visualization

In this section, we visualize the learned speech-acts mapping in the target domain, shown in Figure 6.3. The ground-truth speech-act mapping matrix is a diagonal matrix. As we can see, as more and more data is collected in the target domain, the learned speech-act mapping matrix is getting better.



(a) Speech-act mapping with 1 target domain dialogue. (b) Speech-act mapping with 100 target domain dialogue.

Figure 6.3: Visualization for the speech-act mapping, Figure 6.3(a) is the mapping when there is 1 dialogue in the target domain, and Figure 6.3(b) is the mapping when there are 100 dialogues in the target domain. The ground truth speech-act mapping is the diagonal matrix.

6.3.4 Real-world Experiment

In this section, we will conduct experiments with real-dataset.

Setting

The training dataset is collected when real human users are interacting with an online-learning dialogue agent via Wechat interface shown in Figure 6.4. After each dialogue, the human user gives a subjective feedback reward as a supervision signal, as shown in Figure 6.4(b). The dialogue agent is constantly improving its dialogue policy with a Gaussian process reinforcement learning. We have collected 40 training dialogues in the source domain and 100 dialogues in the target domain. The test dataset is collected when two human users are interacting with each other, where one is acting the customer and the other is acting the dialogue agent. While one of the dialogue agent in the training dataset is a reinforcement learning algorithm, both the customer and the coffee servant are real persons in the test dataset. 60 test dialogues are collected in the target domain as we

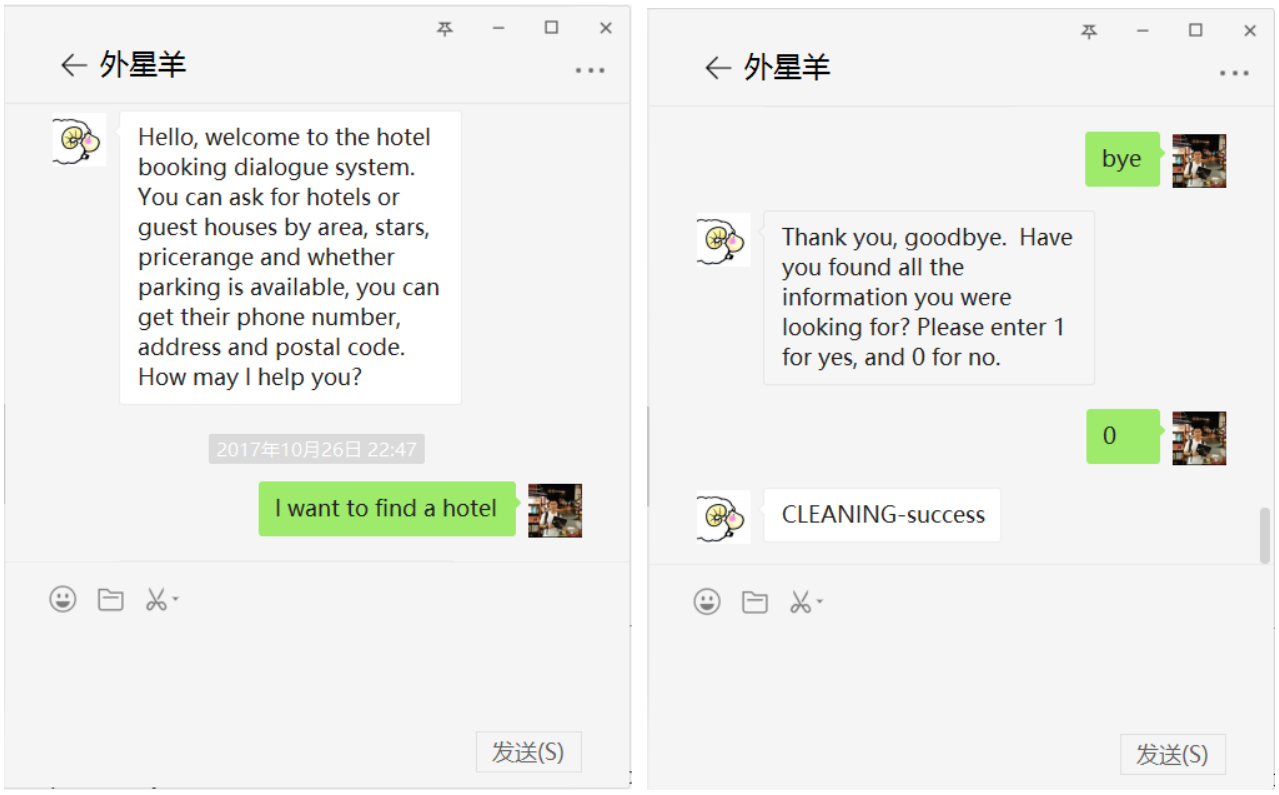


Figure 6.4: Real-data collection and testing interface on Wechat.

only care about the performance in the target domain. Five users participate in the data collection process, they also give immediate rewards on all dialogue turns and those immediate rewards will be used to train the dialogue policy. The reward setting is the same with the simulation experiment, where a final reward of 20 will be received by the dialogue agent for a successful dialogue, and a slight punishment of -1 will be received by the dialogue agent in each dialogue turn to encourage shorter dialogues.

In the real-world dataset, the immediate rewards generated in all dialogue turns will be used to train the dialogue policy. This is the setting where the annotators have to label the immediate rewards of all dialogue turns and it is more costly to label each dialogue, but fewer training dialogues are required to train a satisfactory dialogue policy. The detailed statistics of the real-world dataset is listed in Figure 6.3.

Evaluation

We use two kinds of evaluation metrics to evaluate the dialogue policies trained on the real-world dataset.

A static evaluation is conducted with the testing dataset, because a good dialogue policy should

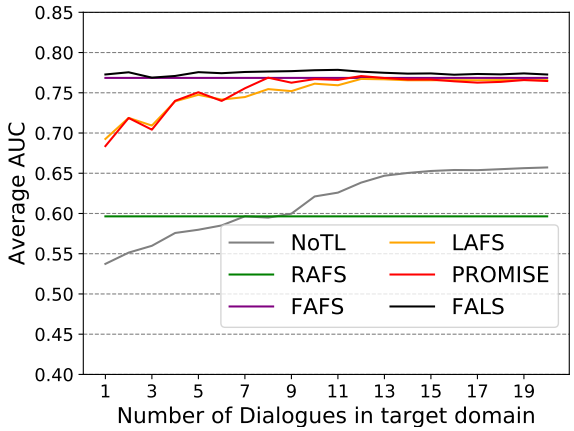
has similar behaviour with the human agent. In each testing dialogue turn, the dialogue agent produces a Q-score for each candidate reply and produces a ranking list of all candidate replies. The ground-truth reply actually performed by the human annotator is labelled as 1 and the other candidate replies are labelled as 0, then an AUC score is calculated for each dialogue turn. If the dialogue policy can rank the ground-truth reply higher, it will have higher AUC and it is considered better, since its behaviour is more similar with the ground-truth human agent. In each dialogue turn, we evaluate the Q-score for 10 times since the Q-score is sampled from a Gaussian distribution. We report the average AUC for all turns and for all samples.

A live evaluation is conducted with the user simulators, since a good dialogue policy should be able to serve simulator users well. The user simulator in the Pydial package interacts lively with the dialogue agent for 300 times where the averaged reward, the averaged success rate, and the averaged dialogue length are used as evaluation metrics.

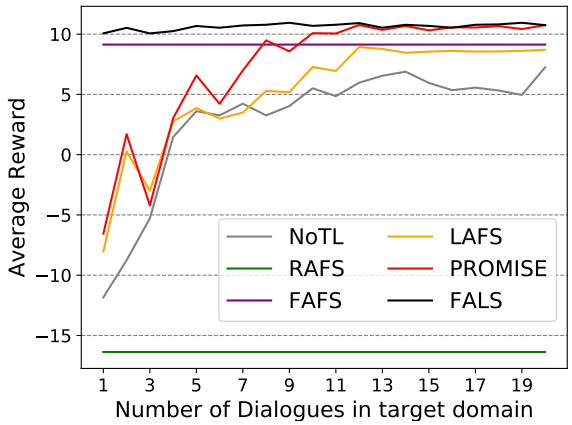
Results

The averaged AUC and averaged reward are shown in Figure 6.5 and Table 6.4. In the static evaluation, we can see that the performance of "NoTL" increases slowly. By looking at the details, we found that the "NoTL" has very large prediction variance, this is because the "NoTL" does not have enough target domain training dialogues to estimate the mean and variance of the Q-function accurately. The transfer learning methods PROMISE and "LAFS" performance much better and their performances are close to the performance upper-bound, since in the source domain they have already learned very good Q-function with accurate mean and small variance. This result shows the proposed method can effective transfer dialogue policy across domains with unknown speech-acts mappings and different slots. The PROMISE is slightly worse but comparable with the performance upper-bound and the "FALS" which use ground truth speech-act mapping.

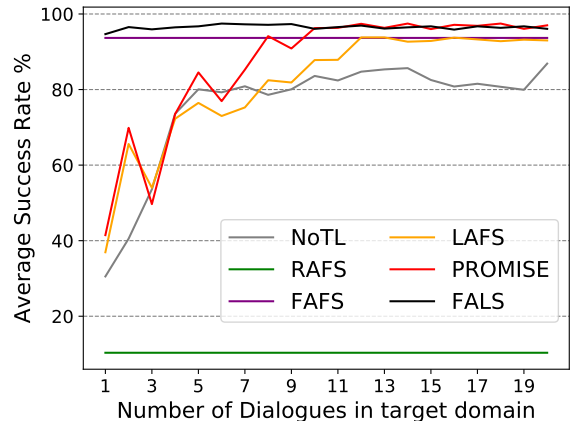
In the live evaluation, we can see that the PROMISE significantly outperforms the "NoTL" method when there are more than 7 training dialogues in the target domain, and its performance is close to the performance upper-bound when there are more than 12 training dialogues in the target domain. This result demonstrates the effectiveness of the proposed dialogue policy transfer method. The PROMISE outperforms the "LAFS" method which uses a heuristic slot-mapping method, which shows that it is better to learn a slot-mapping instead of using a heuristic slot-mapping.



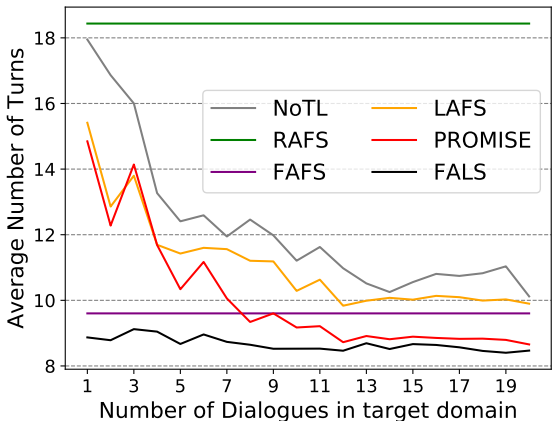
(a) Averaged AUC for the real dataset, higher is better.



(b) Averagd Reward for the real dataset, higher is better.



(c) Averaged Success Rate for the real dataset, higher is better.



(d) Averaged Dialogue Length for the real dataset, lower is better.

Figure 6.5: Experiment results for the Real-world dataset. The proposed method is PROMISE, the methods "FALS" and "FAFS" are the performance upper-bound since they use the ground truth speech-act mapping.

6.4 Conclusion

In this chapter, we tackle the problem of transferring dialogue policies across domains with different speech-acts and disjoint slots. We propose the Policy tRansfer across dOMaIns and SpEech-acts (PROMISE) model, which is able to transfer dialogue policies across different speech-acts and different domains. The PROMISE model can learn to align different speech-acts and slots simultaneously, and it does not require common slots or additional database to calculate slot entropy. Both real-world dialogue data experiments and simulation experiments demonstrate that PROMISE can effectively transfer dialogue policies across different speech-acts and different domains.

Table 6.4: Averaged AUC for static evaluation on the real-word dataset.

# target dialogue / Method	1	5	10	15	20
NoTL	0.5373	0.5798	0.6211	0.6528	0.6572
RAFS	0.5964	0.5964	0.5964	0.5964	0.5964
LAFS	0.6926	0.7474	0.7612	0.7655	0.7657
PROMISE	0.6837	0.7506	0.7670	0.7664	0.7646
FAFS (Upper bound)	0.7684	0.7684	0.7684	0.7684	0.7684
FALS (Upper bound)	0.7727	0.7756	0.7780	0.7740	0.7727

CHAPTER 7

THE UNIFIED TRANSFER REINFORCEMENT LEARNING MODEL

In this section, we summarize the above works with a unified transfer reinforcement learning model, and then we show how to use the this unified transfer reinforcement learning model and learn different personalized weighting function on dialogue policy transfer problems with different granularities.

7.1 The Unified Transfer Reinforcement Learning Model

To solve the research challenges stated in the introduction, we propose three models. All models can be unified as a model, where the target policy Q-function consists of two parts. The first part of the Q-function corresponds to the transferable common dialogue policy, which can be shared across the source and target tasks. The second part of the Q-function models the domain-dependent factors of the dialogue policy. In order to allow a flexible switch between the general model and the personal model, the factor of the common Q-function and the personal Q-function should be modelled as a function of the dialogue state. In order to allow dialogue policy transfer across different domains with different slots and speech-acts, the common dialogue policy part would have a translation function that can translate the target domain states and target domain actions to source domain states and source domain actions. The unified model can be formulated as:

$$Q^t(H^t, Y^t) = (1 - O^p(H^t))Q_g(f(H^t), f(Y^t)) + O^p(H^t)Q_p(H^t, Y^t), \tag{7.1}$$

where $Q_g()$ is the domain-independent common Q-function, $Q_p()$ is the domain-dependent Q-function, $O^p()$ is the weighting factor of the domain-dependent Q-function $Q_p()$, and $f()$ is a cross-domain mapping function that translates the target domain state H^t and action Y^t to the corresponding source domain state and action.

7.2 Learning the Unified Model

The unified model can be trained with typical reinforcement learning algorithms. We denote the set of all model parameters by $\theta = \{\theta_g, \theta_c, \theta_p, \theta_o\}$, where θ_g is the general parameter set shared

across all users in $Q_g()$, θ_c is the cross-domain mapping parameter set in $f(H^t)$ and $f(Y^t)$, $\theta_p = \{\theta_p^u; \forall u\}$ is the personal parameter set consists of a parameter θ_p^u for all users u in $Q_p()$, and θ_o is the parameter set for the personalized weighting function $O^p(H^t)$. The training dataset is denoted by $\mathcal{D} = \{\mathcal{T}\}$ and $\mathcal{T} = \{H_n, Y_n, r_n, {H'}_n\}_{n=1}^N$ is a multi-turn dialogue trajectory of length N. For Q-learning, the learning objective can be formulated as

$$L(\mathcal{D}; \theta) = \sum_{<H_n, Y_n, r_n, {H'}_n> \in \mathcal{D}} (Q(H_n, Y_n) - r_n - \gamma \max_{Y_i} Q({H'}_n, Y_i))^2. \tag{7.2}$$

For policy-gradient method, the learning objective is the expected reward of the policy under a multi-turn trajectory $\mathcal{T} = \{H_n, Y_n, r_n, {H'}_n\}_{n=1}^N$ in $\mathcal{D}$, defined as

$$J(\mathcal{T} \in \mathcal{D}; \theta) = -\mathbb{E}[P(\mathcal{T}; \theta) J^{\mathcal{T}}], \tag{7.3}$$

where $J^{\mathcal{T}}$ is the total reward obtained in a trajectory $\mathcal{T}$. The general training process is as follows:

1. **Source domain training.** Firstly, the model is trained in the source domain dataset $\mathcal{D}^s$, in order to get the source domain general parameter θ_g that could be shared with the target domain. For a personalized dialogue system where multiple users are treated as the source domain, we could also obtain a part of θ_p and θ_o, depending on the form of the actual model being used.

2. **Adaptation on target domain.** Then, the model is adapted in the target domain dataset $\mathcal{D}^t$, in order to learn the other adaptation parameters. For a personalized dialogue system, the target user parameters θ_p and the target user personalized weighting parameter θ_o will be also learned in the target domain. For cross-domain dialogue policy transfer, the cross-domain mapping parameter θ_c will be learned based on the source domain policy and the target domain dataset, in order to maximize the performance of the transferred policy in the target domain.

7.3 Learning Personalized Weighting Function in Different Granularities

In this section, we show how to learn the personal weighting function in settings with different granularities. We will show how to learn the global personal weight factor, the user-level personal weight factor and the word-level personal weight function. The personal weighting functions on different situations can be summarized as

$$O^p(H^t, u) \tag{7.4}$$

where u is the user and H^t can be state variables in different granularities, for example, H^t can be a dialogue state for each dialogue turn or H^t can be the RNN hidden state for each word.

In Chapter 4, we introduce a model with a global personal weight factor. We can further improve the model by learning a different personal weight factor for each target user. In Chapter 5, we introduce a personal word gating function, which works on the level of each user and each generated word. In summary, we organize the personal weighting function in three different granularities as follows:

1. **Global personal weight factor.** A global personal weight factor is a scalar variable indicating how much the dialogue policy should be affected by the user preference. In this setting, the personal Q-function part has a constant affect on the final Q-function for all dialogue situations. The assumption is that for all user in all situation, the personal user preference should have a certain proportion of affect on the final dialogue policy. The global personal weight factor is simple and straightforward, it can be learned from a limited number of training instances, and it is also robust. However, due to the simple nature, the model has a low model capacity and can not deal with situations when the dialogue policies should have different proportions of personalization for different users. The Global personal weight factor can be learned as a variable in the end-to-end supervised learning or reinforcement learning.

2. **User-level personal weight factor.** For every target user in the dialogue system, there is a separate scalar weight variable indicating how much the dialogue policy should be affected by the user preference. The user-level personal weight factors of different users could be different. The assumption is that different users might have different levels of personalization, some users might always act according to their preferences, and some users might want to try new options more. The dialogue policies for the users with a regular pattern should be more personalized. The user-level personal weight factors are more flexible than the global personal weight factor, since there are more parameters to learn. However, more training data for each target user is required to train the model. The user-level personal weight factors can be learned directly as a group of variables with end-to-end supervised learning or reinforcement learning.

3. **Word-level personal weight function.** The word-level personal weight function predicts the personal gating indicator variable for each generated word in each sentence, for each user. The personal gating variable indicates whether each generated word should be affected by the user preference, for each dialogue state and for each different user. This is the personalization

mechanism with the finest granularity. The assumptions is that some phrases in the dialogue sentences are related to the user's personal preference while some phrases are commonly seen in all dialogue sentences. The common phrase model could be shared across all users while the phrases related to personal preferences should not be shared at all. The word-level personal weight function is more flexible than the user-level personal weight factor, since it is dependent on the hidden state for each generated words, for each user. It can enable the dialogue policy transfer learning to operate on the fine-granularity phrase level, sharing the common phrase generator model. However, training a word-level personal weight function requires the most labelling information. The parameters of the word-level personal weight function can be trained directly as a target of supervised learning, and it can also be trained via end-to-end supervised learning or reinforcement learning.

7.3.1 Learning Global Personal Weight Factor

In this part, we introduce how to learn a single personal weight factor for all user. A global personal weight factor is a scalar variable used to model how much the dialogue policy should be affected by the personal preference. A large personal weight factor means that the dialogue policy is affected more by the personal preference while a small personal weight factor means the dialogue policy is more general.

If a global personal weight factor is used, it means the personal weight factor o^p does not depend on the user or the dialogue state. Here the dialogue state of user u is a state vector $\mathbf{h}_n^u$ which is extracted from the dialogue history $H_n^u = \{\{X_i^u, Y_i^u\}_{i=1}^{n-1}, X_n^u\}$, a dialogue action is a sentence embedding vector $\mathbf{y}_n^u$ extracted from the n-th candidate reply sentence Y_n^u, and the state vector contains all the information required to rank the reply candidate sentences. In a dialogue manager, the task of the dialogue policy $Q(\mathbf{h}_n^u, \mathbf{y}_n^u)$ is to rank a list of candidate reply sentences under a specific dialogue state.

The training loss function is formulated as

$$L(\mathcal{D};\theta) = \sum_{<\mathbf{h}_n^u, \mathbf{y}_n^u, r_n^u, \mathbf{h'}_n^u> \in \mathcal{D}} (Q(\mathbf{h}_n^u, \mathbf{y}_n^u) - r_n^u - \gamma \max_{\mathbf{y}_i^u} Q(\mathbf{h'}_n^u, \mathbf{y}_i^u))^2, \tag{7.5}$$

where $\mathcal{D} = \{\mathcal{D}^u\}$ is the training data for all users and $\mathcal{D}^u$ is the training data for user u, $\theta = \{\theta_g, \{\theta_p^u\}, o^p\}$ is the set of parameters we need to learn, θ_g, o^p are the parameters shared across all users and θ_p^u are the personal preference parameters for user u. $Q(\mathbf{h}_n^u, \mathbf{y}_n^u)$ is defined as

$$Q(\mathbf{h}_n^u, \mathbf{y}_n^u; \theta) = Q_g(\mathbf{h}_n^u, \mathbf{y}_n^u; \theta_g) + o^p Q_p(\mathbf{h}_n^u, \mathbf{y}_n^u; \theta_p^u). \tag{7.6}$$

The global personal weight factor o^p can be learned for all user via gradient based optimization methods such as gradient descent. For all data $< \mathbf{h}_n^u, \mathbf{y}_n^u, r_n^u, \mathbf{h}'^u_n > \in \mathcal{D}$, the update function for o^p is formulated as

$$o^p \leftarrow o^p + \alpha \Delta_{o^p} L(\mathcal{D}; \theta). \tag{7.7}$$

The global personal weight factor is simple and straightforward, it can be learned from a limited number of training instances, and it is also robust. However, due to the simple nature, the model has a low model capacity and can not deal with situations when the dialogue policy should have different proportion of personalization for different users.

7.3.2 Learning User-level Personal Weight Factor

In this part, we introduce how to learn a personal weight factor for each user. In order to model different personal weight factors for different users, the model has a separate personal weight variable for each user. For the users who have regular speech pattern, for example, those who always order the same kind of coffee to the same address, a personalized dialogue system should behaviour more according to the users' preferences and make personalized recommendations. For the users that have diversified interest who do not have regular option pattern, the dialogue system should be more general to deal with various new situations.

We use $\mathbf{o}$ to denote the user-level personal weight vector which consists of the personal weight of each user, and o^u is the scalar personal weight for user u. The dialogue state vector of user u is $\mathbf{h}_n^u$, which is extracted from the dialogue history $H_n^u = \{\{X_i^u, Y_i^u\}_{i=1}^{n-1}, X_n^u\}$. The dialogue action vector of user u is $\mathbf{y}_n^u$, and it is the sentence embedding vector extracted from the reply sentence Y_n^u. The task is to choose an appropriate reply sentence under each dialogue state. In a dialogue manager, the dialogue policy ranks each candidate reply with the Q-function $Q(\mathbf{h}_n^u, \mathbf{y}_n)$.

The training loss function is formulated as

$$L(\mathcal{D}; \theta) = \sum_{<\mathbf{h}_n^u, \mathbf{y}_n^u, r_n^u, \mathbf{h}'^u_n> \in \mathcal{D}^u} (Q(\mathbf{h}_n^u, \mathbf{y}_n^u) - r_n^u - \gamma \max_{\mathbf{y}_i^u} Q(\mathbf{h}'^u_n, \mathbf{y}_i^u))^2, \tag{7.8}$$

where $\mathcal{D} = \{\mathcal{D}^u\}$ is the training data for all users and $\mathcal{D}^u$ is the training data for user u, $\theta = \{\theta_g, \{\theta_p^u\}, \mathbf{o}\}$ is the set of all the parameters we need to train, θ_g is the general parameter shared across all users and there is a separate personal parameter θ_p^u and a o^u for each user. $Q(\mathbf{h}_n^u, \mathbf{y}_n^u)$ is defined as

$$Q(\mathbf{h}_n^u, \mathbf{y}_n^u; \theta) = Q_g(\mathbf{h}_n^u, \mathbf{y}_n^u; \theta_g) + o^u Q_p(\mathbf{h}_n^u, \mathbf{y}_n^u; \theta_p). \tag{7.9}$$

For each training trajectory $< \mathbf{h}_n^u, \mathbf{y}_n^u, r_n^u, \mathbf{h}'^u_n > \in \mathcal{D}^u$ of user u, we can update the personal weight factor o^u for the corresponding user u as

$$o^u \leftarrow o^u + \alpha \Delta_{o^u} L(\mathcal{D}^u; \theta). \tag{7.10}$$

The user-level personal weight factors give more flexibility to the model by having a different personal weight variable for each user, so the dialogue policy can have different personalized behaviour for different users. However, more training dialogues are required to train the model.

7.3.3 Learning Word-level Personal Weight Function

In this part, we introduce how to learn a word-level personal weight function (personal word gating function) for each word in response generation. In order to transfer fine-granularity phrase-level knowledge across different users, we propose the word-level personal weight function. In an end-to-end personalized dialogue policy, a word-level personal weight function predicts the personal word gating variable $o_{n,t}$ for each generated word $y_{n,t} \in Y_n$ based on the hidden states in the recurrent models.

The motivation is explained as follows. Some phrases in the dialogue sentences are commonly seen in all dialogues, for example, "Would you like" and "please deliver to" are such common phrases in services domain. Some phrases in the dialogue are related to the personal information of a user, for example, "No.1199 Minsheng Road, Pudong District Shanghai" is a sensitive personal address that a user would not like to share. The common phrases can be shared across all users while the personal phrases should not be shared at all, but this two kind of phrases might appear together in the same dialogue sentence. Traditional transfer learning methods operate on the granularity of sentences, so they might have poor performance and might lead to personal information leakage. We propose a novel personal word gating mechanism for this problem, the personal word gating mechanism can predict whether the current word is a general word or a personal word, and switch to appropriate response generation model accordingly. The personal word gating variable can be seen as a kind of personal weight for each word, so here the personal word gating function is a kind of word-level weighting function which predicts the personal word gating variables. The personal word gating variable indicates whether the word should be affected by the user preference before this word is generated.

In an end-to-end personalized dialogue policy, the task is to generate the response sentence word by word. In the generation process, the state $\mathbf{h}_{n,t}$ is the current hidden state for the t-th word in the n-th system reply in the recurrent model. The possible actions are all the words y in

the vocabulary, and the ground-truth word is denoted as $y_{n,t}$. We can use a generation function $\pi(\mathbf{h}_{n,t}, y_{n,t})$ to model the generation probability as

$$P(\hat{y}_{n,t} = y_{n,t}|\mathbf{h}_{n,t}) = \pi(\mathbf{h}_{n,t}, y_{n,t}), \tag{7.11}$$

$$\pi(\mathbf{h}^u_n, \mathbf{y}^u_n; \theta) = (1 - O(\mathbf{h}^u_n; \theta_o))\pi_g(\mathbf{h}^u_{n,t}, y^u_{n,t}; \theta_g) + O(\mathbf{h}^u_n; \theta_o)\pi_p(\mathbf{h}^u_n, y^u_{n,t}; \theta_p), \tag{7.12}$$

where $\theta = \{\theta_g, \{\theta^u_p\}_u, \theta_o\}$ are the parameters to learn, θ_g is the general model shared across all users, θ^u_p is the personal generation model of user u and θ_o is the parameter for the personal word-gating function.

In the previous section, the personal word gating variable $\hat{o}^u_{n,t-1}$ is binary. If the predicted personal word gate variable $\hat{o}^u_{n,t-1}$ is 0, the current word $y_{n,t}$ will be generated by the common model, other wise $y_{n,t}$ will be generated by the personal model of user u, as

$$\pi(\mathbf{h}_{n,t}, y_{n,t}; \theta) = \begin{cases} \pi_g(\mathbf{h}_{n,t}, y_{n,t}; \theta_g) & \text{if } \hat{o}^u_{n,t-1} = 0 \\ \pi_p(\mathbf{h}_{n,t}, y_{n,t}; \theta^u) & \text{if } \hat{o}^u_{n,t-1} = 1 \end{cases}. \tag{7.13}$$

A training dialogue trajectory of length N is denoted as $\mathcal{T}^u = \{H^u_n, Y^u_n, r^u_n, H'^u_n\}^N_{n=1}$. We denote the return of the trajectory by $J^{\mathcal{T}^u} = \sum^N_{n=1} \gamma^{n-1} r^u_n$ where r^u_n is the reward obtained at the n-th dialogue turn. The model can be trained via policy gradient method, and the loss function is defined as the expected reward of the policy under all trajectories $\mathcal{T}^u$ in $\mathcal{D}^u$:

$$J(\theta) = -\int_{\mathcal{T}^u} p(\mathcal{T}^u; \theta) J^{\mathcal{T}^u} d\mathcal{T}^u, \tag{7.14}$$

$$J(\mathcal{D}^u; \theta) = -\mathbb{E}(P(\mathcal{T}^u \in \mathcal{D}^u; \theta) J^{\mathcal{T}^u}). \tag{7.15}$$

The word-level personal function can be learned by

$$\theta_o \leftarrow \theta_o + \alpha \Delta_{\theta_o} J(\mathcal{D}^u; \theta). \tag{7.16}$$

When we have a ground truth binary annotation $o^u_{n,t}$ for each word in the training sentences, the gradient can be calculated by

$$\Delta_{\theta_o} J(\mathcal{D}^u; \theta) = \mathbb{E}\{\sum_n \sum_t \Delta_{\theta_o} \log P(\hat{o}^u_{n,t} = o^u_{n,t}|\mathbf{h}^u_{n,t}; \theta_o) J^{\mathcal{T}^u}_n\}. \tag{7.17}$$

When we do not have ground truth annotations for each word in the training sentences, it is theoretical possible to learn the soft personal word label $o^u_{n,t}$ in the training set as a real value latent variable, so the variables to be learned becomes $\theta = \{\theta_g, \{\theta_p\}, \theta_o, \{o^u_{n,t}\}_{\forall u,n,t}\}$, then the gradient can be calculated by

$$\Delta_\theta J(\mathcal{D}^u; \theta) = \mathbb{E}\{\sum_n \sum_t \Delta_\theta \log P(y^u_{n,t}|\hat{o}^u_{n,t} = o^u_{n,t}, \mathbf{h}^u_{n,t}; \theta_g, \theta_p) P(\hat{o}^u_{n,t} = o^u_{n,t}|\mathbf{h}^u_{n,t}; \theta_o) J^{\mathcal{T}^u}_n\}.$$

However, in our experiments, since the number of parameters are too large, learning the ground-truth personal word labels as latent variables suffers from severe noise problem, and the performance is not good. We leave this part to the future work.

The word-level personal weight function is the most flexible model, since it is dependent on the hidden state for each generated words. It can enable the dialogue policy transfer learning to operate on the fine-granularity phrase level, where the common phrases generator model is shared across all users. However, training the word-level personal weight function requires the most annotation information, since the ground-truth personal gating variables for all words have to be annotated.

CHAPTER 8

CONCLUSION AND FUTURE WORKS

In this thesis, we tackle the problem of transferring task-oriented dialogue policies across users with different preferences, across domains with different slots and different speech-acts. In previous chapters, we propose three new models for three different scenarios and we summarize them into a unified transfer reinforcement learning framework. With the proposed unified transfer reinforcement learning framework, the dialogue policies in the source domain can be transferred to the target domain with little data, and it can help the target domain dialogue policy to reduce dialogue length and complete tasks faster, improve the response generation quality and reduce the amount of training data needed in the target domain.

8.1 Conclusion

In this thesis, we study the transfer reinforcement learning problem in task-oriented dialogue systems, and our contributions are summarized as follows.

We have systematically identified and formulated **three transfer reinforcement learning problems** in task-oriented dialogue systems.

1. In traditional rule-based dialogue system, the dialogue states, system speech-acts and the user speech-acts have to be manually defined, and it is difficult to reuse the system when the dialogue states and speech-acts are hard to define. In personalized task-oriented dialogue systems, the data collected from each user is usually insufficient for training a traditional learning-based dialogue system. A personalized dialogue system trained on a small dataset is likely to fail on unseen but common dialogues due to the over-fitting. Traditional transfer learning dialogue systems transfer dialogue data across users with different preferences, but they do not model the difference between different users. As a result, the transferred policy might harm the performance of the personalized dialogue systems in the target domain, e.g., making wrong recommendations. How to learn the dialogue states and the dialogue policy automatically from the source domain and adapt to the target user with little data? This question is studied in Chapter 4.

2. In personalized task-oriented dialogue systems, some phrases in the dialogue sentences are commonly seen in all dialogues, for example, "Would you like" and "please deliver to" are such common phrases in services domains. Some phrases in the dialogues are related to the personal information of a user, for example, "No.1199 Minsheng Road, Pudong District Shanghai" is a sensitive personal address that a user would not like to share. The common phrases can be shared across all users while the personal phrases should not be shared at all. However, this two kind of phrases might appear together in the same dialogue sentence. Traditional transfer learning methods operate on the granularity of sentences, so they might have poor performance and might lead to the personal information leakage. How to transfer an end-to-end dialogue policy from the source domain to the target domain effectively? This question is studied in Chapter 5.

3. Existing cross-domain dialogue policy transfer models cannot transfer across domains with different speech-acts, and the current slot-mapping matrix is either based on common slots or built with human heuristic and requires all data rows in the database. However, in many situations there might not be any common slots and the full database access might not be available, so the existing methods can only be used in limited situations. How to transfer a task-oriented dialogue policy across domains with different slots and speech-acts? This question is studied in Chapter 6.

We propose **three models** for the above mentioned problems, including the PErsonalized Task-oriented diALogue (PETAL, Chapter 4) model, the Personalized Decoder model (Chapter 5) and the Policy tRansfer across dOMaIns and SpEech-acts (PROMISE) model (Chapter 6). We have summarized these models into a Unified Transfer Reinforcement Learning model (Chapter 7), and we give some general principles and methodologies for policy transfer across dialogues with different users, different domains and different speech-acts.

We have applied the above transfer reinforcement learning models in different applications, and we have conducted extensive experiments to verify the effective of the propose models compared with baseline models.

8.2 Future Works

In this section, we list some promising future directions that are not yet explored.

Fine-grain knowledge transfer without personal word gating annotation. Fine-granularity knowl-

edge transfer models are proven to perform better than the sentence-level knowledge transfer models. However, in the previous Chapter 5, in order to transfer fine-granularity knowledge, we will have to annotate the ground-truth personal word label for each word in the training dataset, which requires a lot of human efforts. Can we let the model learn to identify which word is personal and which word is not automatically? A general idea is to treat the ground-truth personal word labels in the training set as latent variables, and design an algorithm to learn it from the training dataset. The tf-idf features of each word should also help to distinguish whether a word is personal or not. If we can infer automatically which part of the sentence is personal and which part should be shared, this implies that this algorithm can automatically discover the domain-independent knowledge ready for transfer, and it will have great impact and it will allow transfer learning on many sequence-to-sequence-based models.

Co-training for transfer reinforcement learning. Current transfer learning in reinforcement learning considers only single direction transfer, from the source domain to the target domain. However, it is possible the enhanced target domain could also help the learning of the source domain. Co-training style transfer learning framework might be used in such situation, where the source domain and the target domain can help each other alternatively.

In current task-oriented dialogue systems, only the dialogue history can be used as the dialogue context. However, many environment factors will also affect the dialogue responses of a human being. In a different time and different places, people might have different kinds of responses to the same question. For example, in a formal conference, people might respond to question seriously. But in a party, people might joke about the question. The external factors such as the time, the location, the activity or even the weather should be considered in the dialogue context.

Current learning-based task-oriented dialogue systems usually focus on one single dialogue domains, and they cannot talk about something other than the current domain. However, in real dialogues, people might jump from topic to topic and talk about many tasks in the same dialogue. Designing a dialogue state for each possible tasks might work, but it will cost a lot of human effort. Designing a learning algorithm that can learn from dialogues with multiple domains will surely be of great help. There are a few difficult questions, e.g., how do we know the task has been changed, how to define a task? Can we use external knowledge such as knowledge-graph to help construct the dialogue states? Solving such questions might have great industrial and academic impact.

REFERENCES

[1] James F. Allen and C. Raymond Perrault. Analyzing intention in utterances. *Artif. Intell.*, 15(3):143–178, 1980.

[2] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[3] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 119–125. AAAI Press / The MIT Press, 2002.

[4] Gabor Angeli, Percy Liang, and Dan Klein. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512. Association for Computational Linguistics, 2010.

[5] Mehran Asadi and Manfred Huber. Effective control knowledge transfer through learning skill and representation hierarchies. In Veloso [151], pages 2054–2059.

[6] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] Jeesoo Bang, Hyungjong Noh, Yonghee Kim, and Gary Geunbae Lee. Example-based chat-oriented dialogue system with personalized long-term memory. In *Proceedings of International Conference on Big Data and Smart Computing*, pages 238–243, 2015.

[9] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[10] Mohsen Bayati, David F Gleich, Amin Saberi, and Ying Wang. Message-passing algorithms for sparse network alignment. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 7(1):3, 2013.

[11] R Bellman. Dynamic programming princeton university press princeton. *New Jersey Google Scholar*, 1957.

[12] Richard Bellman. A Markovian decision process. Technical report, DTIC Document, 1957.

[13] Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(04):431–455, 2008.

[14] Alan W Biermann and Philip M Long. The composition of messages in speech-graphics interactive systems. In *Proceedings of the 1996 International Symposium on Spoken Dialogue*, pages 97–100, 1996.

[15] Dan Bohus and Alex Rudnicky. A k hypotheses+ otherbelief updating model. In *Proc. of the AAAI Workshop on Statistical and Empirical Methods in Spoken Dialogue Systems*, volume 62, 2006.

[16] Hélene Bonneau-Maynard, Sophie Rosset, Christelle Ayache, Anne Kuhn, and Djamel Mostefa. Semantic annotation of the french media dialog corpus. In *Ninth European Conference on Speech Communication and Technology*, 2005.

[17] Antoine Bordes and Jason Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016.

[18] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of 19th International Conference on Computational Statistics*, pages 177–186, 2010.

[19] James L Carroll and Kevin Seppi. Task similarity measures for transfer in reinforcement learning task libraries. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 803–808. IEEE, 2005.

[20] Inigo Casanueva, Thomas Hain, Heidi Christensen, Ricard Marxer, and Phil Green. Knowledge transfer between speakers for personalised dialogue management. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2015.

[21] Asli Celikyilmaz and Dilek Hakkani-Tur. Convolutional neural network based semantic tagging with entity embeddings. *genre*, 2010.

[22] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[23] Deborah A Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the workshop on Human Language Technology*, pages 43–48. Association for Computational Linguistics, 1994.

[24] George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[25] Lucie Daubigney, Matthieu Geist, Senthilkumar Chandramohan, and Olivier Pietquin. A comprehensive reinforcement learning framework for dialogue management optimization. *IEEE Journal of Selected Topics in Signal Processing*, 6(8):891–902, 2012.

[26] Lucie Daubigney, Matthieu Geist, and Olivier Pietquin. Off-policy learning in large-scale pomdp-based dialogue systems. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4989–4992. IEEE, 2012.

[27] Yuxin Deng and Wenjie Du. The kantorovich metric in computer science: A brief survey. *Electr. Notes Theor. Comput. Sci.*, 253(3):73–82, 2009.

[28] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.*, 13:227–303, 2000.

[29] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[30] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM, 2006.

[31] David J. Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49(2-3):325–346, 2002.

[32] Michel Galley, Chris Brockett, Alessandro Sordoni, Yangfeng Ji, Michael Auli, Chris Quirk, Margaret Mitchell, Jianfeng Gao, and Bill Dolan. deltaBLEU: A discriminative metric for generation tasks with intrinsically diverse targets. *arXiv preprint arXiv:1506.06863*, 2015.

[33] M Gašić, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. On-line policy optimisation of bayesian

spoken dialogue systems via human interaction. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8367–8371. IEEE, 2013.

[34] M Gašić, Dongho Kim, Pirros Tsiakoulis, and Steve Young. Distributed dialogue policies for multi-domain statistical dialogue management. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5371–5375. IEEE, 2015.

[35] M Gašic, N Mrkšic, L Rojas Barahona, PH Su, D Vandyke, and TH Wen. Multi-agent learning in multi-domain spoken dialogue systems.

[36] M Gašić, N Mrkšić, PH Su, David Vandyke, Tsung-Hsien Wen, and S Young. Policy committee for adaptation in multi-domain spoken dialogue systems. 2015.

[37] Milica Gašic, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. POMDP-based dialogue manager adaptation to extended domains. In *Proceedings of the 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2013.

[38] Milica Gasic, Dongho Kim, Pirros Tsiakoulis, Catherine Breslin, Matthew Henderson, Martin Szummer, Blaise Thomson, and Steve J. Young. Incremental on-line adaptation of POMDP-based dialogue managers to extended domains. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association*, pages 140–144, 2014.

[39] Milica Gašic, Nikola Mrkšic, L Rojas Barahona, PH Su, D Vandyke, and TH Wen. Multi-agent learning in multi-domain spoken dialogue systems. In *The Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS), Workshop on Machine Learning for Spoken Language Understanding and Interaction*, 2015.

[40] Milica Gašić and Steve Young. Effective handling of dialogue state in the hidden information state pomdp-based dialogue manager. *ACM Transactions on Speech and Language Processing (TSLP)*, 7(3):4, 2011.

[41] Milica Gašić and Steve Young. Gaussian processes for pomdp-based dialogue manager optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):28–40, 2014.

[42] Aude Genevay and Romain Laroche. Transfer learning for user adaptation in spoken dialogue systems. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 975–983, 2016.

[43] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[44] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE, 1996.

[45] Yulan He and Steve Young. Spoken language understanding using the hidden vector state model. *Speech Communication*, 48(3):262–275, 2006.

[46] Matthew Henderson, Milica Gašić, Blaise Thomson, Pirros Tsiakoulis, Kai Yu, and Steve Young. Discriminative spoken language understanding using word confusion networks. In *Spoken Language Technology Workshop (SLT), 2012 IEEE*, pages 176–181. IEEE, 2012.

[47] Matthew Henderson, Blaise Thomson, and Jason Williams. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2014.

[48] Matthew Henderson, Blaise Thomson, and Jason D Williams. The third dialog state tracking challenge. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 324–329. IEEE, 2014.

[49] Matthew Henderson, Blaise Thomson, and Steve Young. Deep neural network approach for the dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 467–471, 2013.

[50] Matthew Henderson, Blaise Thomson, and Steve Young. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 360–365. IEEE, 2014.

[51] Matthew Henderson, Blaise Thomson, and Steve 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)*, pages 292–299, 2014.

[52] Takuya Hiraoka, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Reinforcement learning of cooperative persuasive dialogue policies using framing. In *Proceedings of the 25th International Conference on Computational Linguistics*, pages 1706–1717, 2014.

[53] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[54] Ronald A Howard. *Dynamic programming and Markov processes*. Wiley for The Massachusetts Institute of Technology, 1964.

[55] Derek Hao Hu, Vincent Wenchen Zheng, and Qiang Yang. Cross-domain activity recognition via transfer learning. *Pervasive and Mobile Computing*, 7(3):344–358, 2011.

[56] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.

[57] Minwoo Jeong and Gary Geunbae Lee. Multi-domain spoken language understanding with transfer learning. *Speech Communication*, 51(5):412–424, 2009.

[58] Filip Jurčíček, Blaise Thomson, and Steve Young. Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as pomdps. *ACM Transactions on Speech and Language Processing (TSLP)*, 7(3):6, 2011.

[59] Seokhwan Kim and Rafael E Banchs. Sequential labeling for tracking dynamic dialog states. In *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 332, 2014.

[60] Yonghee Kim, Jeesoo Bang, Junhwi Choi, Seonghan Ryu, Sangjun Koo, and Gary Geunbae Lee. Acquisition and use of long-term memory for personalized dialog systems. In *Proceedings of International Workshop on Multimodal Analyses Enabling Artificial Agents in Human-Machine Interaction*, pages 78–87, 2014.

[61] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] Victor Klee and George J Minty. How good is the simplex algorithm. Technical report, WASHINGTON UNIV SEATTLE DEPT OF MATHEMATICS, 1970.

[63] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *NIPS*, volume 13, pages 1008–1014, 1999.

[64] Ravi Kondadadi, Blake Howald, and Frank Schilder. A statistical nlg framework for aggregated planning and realization. In *ACL (1)*, pages 1406–1415, 2013.

[65] George Konidaris and Andrew G. Barto. Building portable options: Skill transfer in reinforcement learning. In Veloso [151], pages 895–900.

[66] George Konidaris, Scott Kuindersma, Roderic A. Grupen, and Andrew G. Barto. Autonomous skill acquisition on a mobile manipulator. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[67] George Konidaris, Ilya Scheidwasser, and Andrew Barto. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May):1333–1371, 2012.

[68] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In *Proc. Ecol. Inference Conf.*, 2011.

[69] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3675–3683, 2016.

[70] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001.

[71] Aravind S Lakshminarayanan, Ramnandan Krishnamurthy, Peeyush Kumar, and Balaraman Ravindran. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*, 2016.

[72] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551. ACM, 2008.

[73] Byung-Jun Lee and Kee-Eung Kim. Dialog history construction with long-short term memory for robust generative dialog state tracking. *Dialogue & Discourse*, 7(3):47–64, 2016.

[74] Sungjin Lee. Structured discriminative model for dialog state tracking. In *Proceedings of the SIGDIAL 2013 Conference*, pages 442–451, 2013.

[75] Fabrice Lefèvre, Milica Gašić, F Jurčíček, Simon Keizer, François Mairesse, Blaise Thomson, Kai Yu, and S Young. k-nearest neighbor monte-carlo control algorithm for pomdp-based dialogue systems. In *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 272–275. Association for Computational Linguistics, 2009.

[76] Esther Levin, Roberto Pieraccini, and Wieland Eckert. Learning dialogue strategies within the Markov decision process framework. In *Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 72–79, 1997.

[77] Jiwei Li, Will Monroe, Alan Ritter, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[78] Lihong Li, Jason D Williams, and Suhrid Balakrishnan. Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection. In *INTERSPEECH*, pages 2475–2478, 2009.

[79] Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc., 1995.

[80] Bing Liu and Ian 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.

[81] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pages 97–105, 2015.

[82] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[83] François Mairesse, Milica Gasic, Filip Jurcícek, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. Spoken language understanding from unaligned data using discriminative

classification models. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4749–4752. IEEE, 2009.

[84] François Mairesse and Marilyn A Walker. Trainable generation of big-five personality styles through data-driven parameter estimation. In *ACL*, pages 165–173, 2008.

[85] François Mairesse and Marilyn A Walker. Controlling user perceptions of linguistic style: Trainable generation of personality traits. *Computational Linguistics*, 37(3):455–488, 2011.

[86] François Mairesse and Steve Young. Stochastic language generation in dialogue using factored language models. *Computational Linguistics*, 2014.

[87] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 361–368. Morgan Kaufmann, 2001.

[88] M. Melekopoglou and A. Condon. On the complexity of the policy iteration algorithm for stochastic games. Technical report, Computer Sciences Department, University of Wisconsin Madison., 1990.

[89] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128. IEEE, 2002.

[90] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cutdynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pages 295–306. Springer, 2002.

[91] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2015.

[92] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775, 2013.

[93] Angeliki Metallinou, Dan Bohus, and Jason Williams. Discriminative state tracking for spoken dialog systems. In *ACL (1)*, pages 466–475, 2013.

[94] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[95] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[96] Teruhisa Misu, Kallirroi Georgila, Anton Leuski, and David Traum. Reinforcement learning of question-answering dialogue policies for virtual museum guides. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 84–93. Association for Computational Linguistics, 2012.

[97] Teruhisa Misu, Komei Sugiura, Kiyonori Ohtake, Chiori Hori, Hideki Kashioka, Hisashi Kawai, and Satoshi Nakamura. Dialogue strategy optimization to assist user's decision for spoken consulting dialogue systems. In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 354–359. IEEE, 2010.

[98] Lili Mou, Yiping Song, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. Sequence to backward and forward sequences: A content-introducing approach to generative short-text conversation. *arXiv preprint arXiv:1607.00970*, 2016.

[99] Nikola Mrkšić, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašić, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Multi-domain dialog state tracking using recurrent neural networks. *arXiv preprint arXiv:1506.07190*, 2015.

[100] Scott Niekum and Andrew G. Barto. Clustering via dirichlet process mixture models for portable skill discovery. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 1818–1826, 2011.

[101] Alice H Oh and Alexander I Rudnicky. Stochastic language generation for spoken dialogue systems. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational systems-Volume 3*, pages 27–32. Association for Computational Linguistics, 2000.

[102] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[103] Weike Pan, Evan Wei Xiang, Nathan Nan Liu, and Qiang Yang. Transfer learning in collaborative filtering for sparsity reduction. In *AAAI*, volume 10, pages 230–235, 2010.

[104] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[105] Ronald Parr and Stuart J. Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, pages 1043–1049. The MIT Press, 1997.

[106] Marcello Pelillo. Replicator equations, maximal cliques, and graph isomorphism. In *Advances in Neural Information Processing Systems*, pages 550–556, 1999.

[107] Theodore J Perkins, Doina Precup, et al. Using options for knowledge transfer in reinforcement learning. *University of Massachusetts, Amherst, MA, USA, Tech. Rep*, 1999.

[108] Janarthanan Rajendran, Aravind S Lakshminarayanan, Mitesh M Khapra, P Prasanna, and Balaraman Ravindran. Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain. *arXiv preprint arXiv:1510.02879*, 2015.

[109] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[110] Adwait Ratnaparkhi. Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech & Language*, 16(3):435–455, 2002.

[111] Balaraman Ravindran and Andrew G. Barto. Relativized options: Choosing the right transformation. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 608–615. AAAI Press, 2003.

[112] Christian Raymond and Giuseppe Riccardi. Generative and discriminative algorithms for spoken language understanding. In *INTERSPEECH*, pages 1605–1608, 2007.

[113] Hang Ren, Weiqun Xu, and Yonghong Yan. Markovian discriminative modeling for cross-domain dialog state tracking. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 342–347. IEEE, 2014.

[114] Alan Ritter, Colin Cherry, and William B Dolan. Data-driven response generation in social media. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 583–593, 2011.

[115] Ariel Rosenfeld and Sarit Kraus. Providing arguments in discussions on the basis of the prediction of human argumentative behavior. *ACM Transactions on Interactive Intelligent Systems*, 6(4):30, 2016.

[116] Ariel Rosenfeld and Sarit Kraus. Strategical argumentative agent for human persuasion. In *Proceedings of the 22nd European Conference on Artificial Intelligence*, 2016.

[117] Jost Schatztnann, Matthew N Stuttle, Karl Weilhammer, and Steve Young. Effects of the user model on simulation-based learning of dialogue strategies. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 220–225. IEEE, 2005.

[118] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. *arXiv preprint arXiv:1507.04808*, 2015.

[119] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *arXiv preprint arXiv:1507.04808*, 2015.

[120] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.

[121] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. *arXiv preprint arXiv:1605.06069*, 2016.

[122] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. *arXiv preprint arXiv:1503.02364*, 2015.

[123] Yangyang Shi, Martha Larson, and Catholijn M Jonker. Recurrent neural network language model adaptation with curriculum learning. *Computer Speech & Language*, 33(1):136–154, 2015.

[124] Satinder P. Singh. Reinforcement learning with a hierarchy of abstract models. In William R. Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 202–207. AAAI Press / The MIT Press, 1992.

[125] Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems. In *Nips*, pages 956–962, 1999.

[126] Jinhua Song, Yang Gao, Hao Wang, and Bo An. Measuring the distance between finite markov decision processes. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 468–476. ACM, 2016.

[127] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*, 2015.

[128] Amanda Stent and Martin Molina. Evaluating automatic extraction of rules for sentence plan construction. In *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 290–297. Association for Computational Linguistics, 2009.

[129] Amanda Stent, Rashmi Prasad, and Marilyn Walker. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42nd annual meeting on association for computational linguistics*, page 79. Association for Computational Linguistics, 2004.

[130] Pei-Hao Su, Milica Gasic, Nikola Mrksic, Lina Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689*, 2016.

[131] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[132] Kai Sun, Lu Chen, Su Zhu, and Kai Yu. The sjtu system for dialog state tracking challenge 2. In *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 318, 2014.

[133] Kai Sun, Qizhe Xie, and Kai Yu. Recurrent polynomial network for dialogue state tracking. *arXiv preprint arXiv:1507.03934*, 2015.

[134] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, 2014.

[135] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[136] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[137] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[138] Ben Tan, Yangqiu Song, Erheng Zhong, and Qiang Yang. Transitive transfer learning. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1155–1164, 2015.

[139] Ben Tan, Erheng Zhong, Evan Wei Xiang, and Qiang Yang. Multi-transfer: Transfer learning with multiple views and multiple sources. *Statistical Analysis and Data Mining*, 7(4):282–293, 2014.

[140] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[141] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[142] Cynthia A Thompson, Mehmet H Goker, and Pat Langley. A personalized system for conversational recommendations. *Journal of Artificial Intelligence Research*, 21:393–428, 2004.

[143] Blaise Thomson, Jost Schatzmann, and Steve Young. Bayesian update of dialogue state for robust dialogue systems. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4937–4940. IEEE, 2008.

[144] Blaise Thomson and Steve Young. Bayesian update of dialogue state: A pomdp framework for spoken dialogue systems. *Computer Speech & Language*, 24(4):562–588, 2010.

[145] Volker Tresp. A bayesian committee machine. *Neural Computation*, 12(11):2719–2741, 2000.

[146] Paul Tseng. Solving h-horizon, stationary markov decision problems in time proportional to log (h). *Operations Research Letters*, 9(5):287–297, 1990.

[147] Gökhan Tür. Model adaptation for spoken language understanding. In *ICASSP (1)*, pages 41–44. Citeseer, 2005.

[148] Gokhan Tur. Multitask learning for spoken language understanding. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE, 2006.

[149] Gökhan Tür, Anoop Deoras, and Dilek Hakkani-Tür. Semantic parsing using word confusion networks with conditional random fields. In *INTERSPEECH*, pages 2579–2583, 2013.

[150] Stefan Ultes, Lina M. Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gasic, and Steve Young. PyDial: A Multi-domain Statistical Dialogue System Toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[151] Manuela M. Veloso, editor. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.

[152] Marilyn A Walker, Jeanne C Fromer, and Shrikanth Narayanan. Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Con-*

ference on Computational Linguistics-Volume 2, pages 1345–1351. Association for Computational Linguistics, 1998.

[153] Marilyn A Walker, Rebecca Passonneau, and Julie E Boland. Quantitative and qualitative evaluation of darpa communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 515–522. Association for Computational Linguistics, 2001.

[154] Marilyn A Walker, Owen C Rambow, and Monica Rogati. Training a sentence planner for spoken dialogue using boosting. *Computer Speech & Language*, 16(3):409–433, 2002.

[155] Marilyn A Walker, Amanda Stent, François Mairesse, and Rashmi Prasad. Individual and domain adaptation in sentence planning for dialogue. *Journal of Artificial Intelligence Research*, 30:413–456, 2007.

[156] Hao Wang, Shunguo Fan, Jinhua Song, Yang Gao, and Xingguo Chen. Reinforcement learning transfer based on subgoal discovery and subtask similarity. *IEEE/CAA Journal of Automatica Sinica*, 1(3):257–266, 2014.

[157] Ye-Yi Wang and Alex Acero. Discriminative models for spoken language understanding. In *INTERSPEECH*, 2006.

[158] Zhuoran Wang, Hongliang Chen, Guanchun Wang, Hao Tian, Hua Wu, and Haifeng Wang. Policy learning for domain selection in an extensible multi-domain spoken dialogue system. In *EMNLP*, pages 57–67, 2014.

[159] Zhuoran Wang and Oliver Lemon. A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed information. In *Proceedings of the SIGDIAL 2013 Conference*, pages 423–432, 2013.

[160] Zhuoran Wang, Tsung-Hsien Wen, Pei-Hao Su, and Yannis Stylianou. Learning domain-independent dialogue policies via ontology parameterisation. In *16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 412, 2015.

[161] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[162] Duncan J Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.

[163] Ying Wei, Yu Zheng, and Qiang Yang. Transfer knowledge between cities. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1905–1914, 2016.

[164] Tsung-Hsien Wen, Milica Gasic, Dongho Kim, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *arXiv preprint arXiv:1508.01755*, 2015.

[165] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. Conditional generation and snapshot learning in neural dialogue systems. *arXiv preprint arXiv:1606.03352*, 2016.

[166] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[167] Tsung-Hsien Wen, Milica Gašic, Nikola Mrkšic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. Toward multi-domain language generation using recurrent neural networks.

[168] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. Multi-domain neural network language generation for spoken dialogue systems. *arXiv preprint arXiv:1603.01232*, 2016.

[169] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

[170] Tsung-Hsien Wen, Aaron Heidel, Hung-yi Lee, Yu Tsao, and Lin-Shan Lee. Recurrent neural network based language model personalization by social network crowdsourcing. In *INTERSPEECH*, pages 2703–2707, 2013.

[171] Jason Williams. Multi-domain learning and generalization in dialog state tracking. In *Proceedings of SIGDIAL*, volume 62, 2013.

[172] Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 404–413, 2013.

[173] Jason D Williams. The best of both worlds: unifying conventional dialog systems and pomdps. In *INTERSPEECH*, pages 1173–1176, 2008.

[174] Jason D Williams. Integrating expert knowledge into pomdp optimization for spoken dialog systems. In *Proceedings of the AAAI-08 Workshop on Advancements in POMDP Solvers*, volume 2, page 25, 2008.

[175] Jason D Williams. Incremental partition recombination for efficient tracking of multiple dialog states. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5382–5385. IEEE, 2010.

[176] Jason D Williams. Web-style ranking and slu combination for dialog state tracking. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 282–291, 2014.

[177] Jason D Williams and Geoffrey Zweig. End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning. *arXiv preprint arXiv:1606.01269*, 2016.

[178] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[179] Puyang Xu and Ruhi 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*, pages 78–83. IEEE, 2013.

[180] Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi. Spoken language understanding using long short-term memory neural networks. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 189–194. IEEE, 2014.

[181] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding. In *INTERSPEECH*, pages 2524–2528, 2013.

[182] Majid Yazdani and James Henderson. A model of zero-shot learning of spoken language understanding.

[183] Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174, 2010.

[184] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

[185] Steve Young, Jost Schatzmann, Karl Weilhammer, and Hui Ye. The hidden information state approach to dialog management. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–149. IEEE, 2007.

[186] Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008.

[187] Lukas Zilka and Filip Jurcicek. Incremental lstm-based dialog state tracker. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 757–762. IEEE, 2015.