On End-to-End Learning of Neural Goal-Oriented Dialog Systems

by

Janarthanan Rajendran

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2021

Doctoral Committee:

Professor Satinder Singh Baveja, Chair
Dr. Jonathan K. Kummerfeld
Professor Honglak Lee
Professor Richard L. Lewis

Janarthanan Rajendran
rjana@umich.edu
ORCID iD: 0000-0003-4886-6826

© Janarthanan Rajendran 2021

ACKNOWLEDGMENTS

I am very grateful for having had an incredible group of people guide and support me throughout my Ph.D. First, I would like to thank my advisor Satinder Singh. Through his continued guidance throughout my Ph.D., he taught me how to do good research as well as how to present it well.

I have been fortunate to have collaborated and worked with a lot of wonderful people throughout my Ph.D., along with my advisor. I would like to thank Jatin Ganhotra who has been instrumental in successfully completing my works presented in Chapters 3, 4 and 5. I would like to thank Jonathan K. Kummerfeld for his guidance and collaboration on the work presented in Chapter 6. I would also like to thank him for serving on my dissertation committee and providing valuable feedback on my thesis. I would like to thank Lazaros C. Polymenakos, Xiaoxiao Guo, and Mo Yu for their collaboration and guidance on the different works presented in this thesis.

I would like to thank my collaborators on projects on projects other than those included in this thesis, that I worked on during my Ph.D. I would like to thank Richard L. Lewis and Honglak Lee for their guidance and thought provoking discussions on several of these projects. I would also like to thank them both for serving on my dissertation committee and providing their valuable comments and feedback on my thesis. I would like to thank all my other wonderful collaborators: Vivek Veeriah, Alex Irpan, Eric Jang, Ethan A. Brooks, Laura Biester, Katie Matton, Emily M. Provost, Rada Mihalcea, Yujia Li, Richard Evans, Daniel Tarlow and Oriol Vinyals. It has been a pleasure working with them during my Ph.D.

Lastly, I would like to thank my family, labmates and friends for their loving support throughout my Ph.D. Thank you all!

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

LIST OF FIGURES vi

LIST OF TABLES viii

ABSTRACT x

CHAPTER

1. Introduction 1

- 1.1 Contributions 3
  - 1.1.1 NE-Table: A Neural Key-Value Table for Named Entities 3
  - 1.1.2 Learning With Multiple Answers 4
  - 1.1.3 Learning with Maximal User Task Success and Minimal Human Agent Use 5
  - 1.1.4 Learning to Learn From Related Tasks 6
- 1.2 Thesis Structure 7

2. Background 8

- 2.1 End-to-end Learning of Neural Goal-Oriented Dialog Systems . . 8
- 2.2 Goal-Oriented Dialog Tasks: bAbI 9
- 2.3 Memory Networks For End-To-End Goal-Oriented Dialog Tasks . 12

3. NE-Table: A Neural Key-Value Table for Named Entities 14

- 3.1 Introduction 15
- 3.2 Proposed Method 16
- 3.3 Experiments and Results 18
  - 3.3.1 Reading Comprehension Task 19
  - 3.3.2 Multiple-Attention Based Neural Retrieval Mechanism . 21
  - 3.3.3 Structured-QA from KB 24
  - 3.3.4 Goal-Oriented Dialog Tasks 26

3.4 Related Work . . . 35
3.5 Conclusion . . . 36

4. Learning With Multiple Answers . . . 37

4.1 Introduction . . . 38
4.2 Multiple-Utterance Problem in Goal-Oriented Dialog . . . 39
4.3 Issues With the Present Methods . . . 39
4.4 Proposed Method . . . 41
4.5 Permuted bAbI Dialog Tasks . . . 42
4.6 Experiments and Results . . . 44
  4.6.1 Baseline Method: memN2N . . . 45
  4.6.2 Mask End-to-End Memory Network: Mask-memN2N . 46
  4.6.3 Model Comparison . . . 46
  4.6.4 Ablation Study . . . 47
4.7 Related Work . . . 48
4.8 Conclusion . . . 49

5. Learning with Maximal User Task Success and Minimal Human Agent Use . . . 50

5.1 Introduction . . . 51
5.2 Proposed Method . . . 52
5.3 Modified-bAbI Dialog Tasks . . . 55
5.4 Experiments and Results . . . 57
  5.4.1 Baseline Method: M . . . 57
  5.4.2 Proposed Method: $M^* + T^*$ . . . 58
5.5 Related Work . . . 62
5.6 Conclusion . . . 64

6. Learning to Learn From Related Tasks . . . 66

6.1 Introduction . . . 67
6.2 Proposed Method . . . 68
6.3 Experiments and Results . . . 72
  6.3.1 MNIST Image Classification . . . 72
  6.3.2 Personalized Restaurant Reservation . . . 76
6.4 Related Work . . . 84
6.5 Conclusion . . . 86

7. Conclusion . . . 87

7.1 Summary of Contributions . . . 87
7.2 Future Work . . . 88
7.3 Concluding Remarks . . . 89

BIBLIOGRAPHY 90

LIST OF FIGURES

Figure

2.1 A dialog from the bAbI dialog tasks dataset. A user (in green) chats with a dialog system (utterances in blue and API calls in orange) to book a table at a restaurant. 11

2.2 A single layer version of end-to-end memory networks 13

3.1 For input question - *Who teaches EECS-545*, the *NE-Embedding Generation Module* (f_ϕ) takes the context embedding as input and generates a NE-Embedding for the NE *EECS-545*. The NE-Embedding is stored in *NE-Table* with its actual value *EECS-545*. The *NE-Retrieval Module* (g_θ) performs attention over the keys in *NE-Table* to retrieve the NE-value. We show a simple example here to illustrate f_ϕ and g_θ. Depending on the task, the context can vary and the *NE-Table* can have more entries. 17

3.2 *Left:* Two dialogs from bAbI task-1. A user (in green) chats with a dialog system (in blue) to book a table. Each dialog has its own separate *NE-Table* and a separate NE-Embedding is generated for the NE *London* though it appears in both dialogs. *Right:* Question from CBT. NE *Enrico* (highlighted in yellow) occurs twice in the context *S*, where a separate NE-Embedding is generated for each occurrence. 19

3.3 Results on CBT-NE OOV test sets 21

3.4 Multiple-attention based neural retrieval mechanism. The *KB-Retrieval Module* attends to the relevant rows and columns of the KB by generating attention key embeddings *ACC*, *ACR* and *ARR*. 23

4.1 Mask-memN2N - *Left*: A single layer version of memN2N. *Right*: Masking 40

4.2 Permuted-bAbI dialog tasks. A user (in green) chats with a dialog system (in blue) to book a table at a restaurant. At a given point in the dialog, the dialog system has multiple correct next utterances (in orange). The dialog system can choose either of the multiple correct utterances as the next utterance. The list of restaurants are returned from the API_call (in grey) also contain multiple restaurants with the same rating, giving the dialog system more options to propose to the user. 43

5.1 *Left*: A single layer version of memN2N (dialog model M) *Right*: Proposed Method 53

5.2 Modified-bAbI dialog tasks. A user (in green) chats with a dialog system (in blue) to book a table at a restaurant. We update each subtask in the original-bAbI dialog task with specific changes (in yellow). 56
6.1 (Part 1/2) MNIST image classification. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. The points in red and green correspond to the weights of the related task data points that have incorrect and correct labels respectively. *Right*: Histograms of the number of data points in the different interval of weights. The histograms in red and green correspond to the related task data points with incorrect and correct labels respectively. Refer to Figure 6.2 for Part 2/2. 74
6.2 (Part 2/2) MNIST image classification. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. The points in red and green correspond to the weights of the related task data points that have incorrect and correct labels respectively. *Right*: Histograms of the number of data points in the different interval of weights. The histograms in red and green correspond to the related task data points with incorrect and correct labels respectively. Refer to Figure 6.1 for Part 1/2. 75
6.3 A user (in green) chats with a dialog system (in blue) to book a table at a restaurant. *Left*: (Related Task) An example dialog from Original-bAbI dialog Task 5. *Right*: (Primary Task) An example dialog from Personalized-bAbI dialog Task 3. 77
6.4 (Part 1/2) Personalized Restaurant Reservation. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. *Right*: Histograms of the number of data points in the different interval of weights. Refer to Figure 6.5 for Part 2/2. 82
6.5 (Part 2/2) Personalized Restaurant Reservation. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. *Right*: Histograms of the number of data points in the different interval of weights. Refer to Figure 6.4 for Part 1/2. 83

LIST OF TABLES

Table

3.1 Test results on CBT-NE dataset 20
3.2 Test results on structured-QA task 25
3.3 Test results for KB-augmented-bAbI dialog tasks 1 and 2. Accuracy % for Test and Test-OOV (given in parenthesis). KB-Retrieval %: Retrieval accuracy for rows. Per-Dialog %: Percentage of dialogs where every dialog response is correct. Per-Dialog + KB-Retrieval %: Percentage of dialogs where every dialog response and information from KB retrieval are correct. 29
3.4 Test results for KB-augmented-bAbI dialog tasks 1 and 2. Accuracy % for Test and Test-OOV (given in parenthesis). ARR non-NE columns are price and number of people. ARR NE columns are cuisine and location. . 30
3.5 Test results for KB-augmented-bAbI dialog task 4. Accuracies in % for Test and Test Out-Of-Vocabulary (given in parenthesis). KB-Retrieval %: Retrieval accuracy for a particular cell. Per-Dialog %: Percentage of dialogs where every dialog response is correct. Per-Dialog + KB-Retrieval %: Percentage of dialogs where every dialog response and information from KB retrieval are correct. 33
3.6 Test results for KB-augmented-bAbI dialog task 4. Accuracies in % for Test and Test Out-Of-Vocabulary (given in parenthesis). 33
3.7 Performance comparison of our method in the KB-Augmented-bAbI dialog tasks, with a baseline method in the original-bAbI tasks. Accuracies in % for Test and Test Out-Of-Vocabulary (given in parenthesis). 35
4.1 Test results for our baseline end-to-end memory network model across the three datasets. Results (accuracy %) are given in the standard setup and out-of-vocabulary (OOV) setup. Results are given for both with and without match-type features. 45
4.2 Test results for various methods on permuted-bAbI dialog task. Results (accuracy %) are given in the standard setup and OOV setup; and both with and without match-type features. 47
4.3 Ablation study of our proposed method on permuted-bAbI dialog task. Results (accuracy %) are given in the standard setup, without match-type features. 48

5.1 Modified-bAbI test set statistics. The numbers shown represent number of dialogs in test data (Task 5) out of 1000 dialogs where a certain type of new user behavior is encountered. 57
5.2 Modified-bAbI test set statistics. The numbers shown represent number of dialogs in test data (Task 5) out of 1000 dialogs, where no new user behavior or one or more type of new user behavior is encountered. 57
5.3 Test results (accuracy %) for our baseline method (M:memN2N) across original and modified bAbI dialog tasks. 58
5.4 Test results for the different methods on the modified-bAbI dialog task. The numbers represent the mean and standard deviation of running the different methods across 5 different permutations of the test set. *User Accuracy*: Task success rate for the user; *Model ratio*: Percentage of time the transferrer chooses the model M; *Final Model Accuracy*: Accuracy of the model M at the end of testing. 59
5.5 Test results for $M + T^*$ method on different permutations of modified-bAbI dialog task's test set 63
5.6 Test results for $M^* + T^*$ method on different permutation of modified-bAbI dialog task's test set 63
5.7 Test results for $M_a^* + T^*$ method on different permutation of modified-bAbI dialog task's test set 64
6.1 Test results, % classification accuracy (mean and standard deviation over 5 runs). 76
6.2 Test results, % Per-turn retrieval accuracy (mean and standard deviation over 5 runs) in predicting the next dialog system utterance. 80
6.3 Test results, Per-turn % retrieval accuracy (mean and standard deviation over 5 runs) in predicting the next dialog system utterance. 81

ABSTRACT

Goal-oriented dialog systems assist users to complete tasks such as restaurant reservations and flight ticket booking. Deep neural networks have opened up the possibility of end-to-end learning of the entire goal-oriented dialog system directly from data. End-to-end learning enables automatic adaptation of the different parts of the dialog system accounting for how changes in one part affect the others. Since the entire dialog system is learned directly from the data, the design of the dialog system need not make any assumptions about the domain. This makes it possible to build dialog systems for new domains with different training data, without much domain-specific hand-crafting of the dialog system. With deep neural networks which can potentially capture the complexity of human dialog in natural language, learning neural goal-oriented dialog systems end-to-end holds the promise of bringing dialog systems into our everyday lives.

In this thesis, we identify some of the challenges in end-to-end learning of neural goal-oriented dialog systems and propose methods to address them. We look at four challenges:

1) The challenge posed by the presence of a large number of named entities in goal-oriented dialog tasks. We propose a method to build neural embeddings for named entities on the fly and store them in a key-value table with neural embeddings as keys and the actual named entities as values. The proposed method allows for comparison and retrieval, using neural embeddings as well as actual named entities, which leads to significant improvement in performance, especially in the presence of out-of-vocabulary named entities.

2) The challenge of performing supervised learning of goal-oriented dialog systems with multiple valid next utterances. We propose a method to learn to use different parts of the neural network to encode different predictions of the next utterances with learning of one not interfering with the learning of the others. Our experiments show considerable improvement in the generalization performance.

3) The challenge of handling new user behaviors during deployment of a trained dialog system. We propose a method that learns to anticipate failures and efficiently transfers dialogs to human agents in order to make sure the overall task success of the users remains high. Our experiments show that using our proposed method it is possible to achieve very high user task success while minimally using human agents.

4) The challenge of requiring large amounts of training data for each new dialog task of interest. We show that by selectively learning from a related task's data that is already available, we can improve the performance on a new task of interest that has only a limited amount of training data.

CHAPTER 1

Introduction

Natural language dialog is one of the primary means by which humans communicate with each other and perform many of our day-to-day activities. This includes talking with our family and friends, ordering lunch at a restaurant, discussing ideas in a meeting at work, booking a train ticket at the train station counter and resolving an issue with a computer software by talking to an agent at the support center. It is a long-standing goal of AI to have intelligent systems that can have seamless dialog with humans in natural language and thereby help with some of our day-to-day activities.

Dialog systems are broadly categorised as goal-oriented dialog systems and non-goal-oriented dialog systems. Goal-oriented dialog systems are systems that engage in a dialog to achieve an explicit goal. They are generally used to assist a user to complete certain tasks. Examples include dialog systems that assist with purchasing tickets and reservations for flights, trains, movies, restaurants, hotels etc. Non-goal-oriented dialog systems (also referred to as chit-chat systems) are systems that engage in dialog that are unstructured and spread across multiple domains. They mimic idle *chit-chat*, and are mainly used for entertainment.

In this thesis, we set our focus on goal-oriented dialog systems. A widely used framework for building goal-oriented dialog systems has been to have a pipeline with different modules (pipeline-methods) where the input of one module is dependent on the output of another module. The typical pipeline used for goal-oriented dialog systems has four modules: 1) Natural Language Understanding (NLU), which converts user utterance into useful features (e.g., pre-determined semantic slots), 2) Dialog State Tracking, which combines features of the current user utterance with dialog history and outputs the current dialog state, 3) Dialog Policy, which takes the current dialog state and selects the dialog system's next action, 4) Natural Language Generation (NLG), which maps the selected action to a natural language response for the user.

In the past, dialog systems (and the modules therein) were built mainly using handcrafted features and rules (Bobrow et al. 1977; Ward and Issar 1994; Goddeau et al. 1996). Handcrafting the right set of features and rules for a given dialog task requires extensive domain knowledge of the task. Given the complexity of human dialog in natural language, handcrafting features and rules that works well requires a lot of effort, making it a difficult and expensive processes. Even with best efforts, the dialog systems built this way were brittle and not robust in the real world. The extensive effort involved in building goal-oriented dialog systems with handcrafted features and rules also limited the ability to build dialog systems for new domains and tasks quickly.

With the progress in areas of statistical methods and machine learning, parts of the goal-oriented dialog system pipeline which were earlier fully handcrafted started getting replaced with statistical and machine learning models (Young et al. 2009; Williams 2013). Over the last decade and a half, neural network based learning methods, especially the ones using deep learning, have led to great progress in different fields where learning methods are applicable. This includes vision (Diba et al. 2017; Chen and Yuille 2014), language (Mikolov et al. 2013a; Bahdanau, Cho, and Bengio 2015b) and control (Mnih et al. 2015; Lillicrap et al. 2016). This progress is true for dialog systems as well. Chen, Liu, et al. (2017) provide a good survey on the recent advances in dialog systems. The recent success of deep learning in these fields can be attributed to a combination of several factors. Some of the important ones are: 1) an increase in availability of large amounts of training data, 2) developments in hardware and compute that has enabled training of large deep learning models with large amounts of data in reasonable time, 3) development of specialised deep learning libraries for programming languages, that have enabled easy and fast construction of deep neural network models and their training schemes and 4) improvement in deep neural network architecture and optimization methods.

All of the four modules in the pipeline-method of building goal-oriented dialog systems have undergone tremendous progress over the last decade with the use of deep neural network based learning methods. Some examples include: 1) natural language understanding: Tur et al. (2012) and Yao et al. (2014), 2) dialog state tracking: Henderson, Thomson, and Young (2013) and Mrkšić et al. (2015), 3) dialog policy learning: Cuayáhuitl, Keizer, and Lemon (2015) and Takanobu, Zhu, and Huang (2019), and 4) natural language generation: Wen, Gasic, et al. (2015) and Tran, Nguyen, and Nguyen (2017). Deep neural networks have enabled automatic learning of useful feature representation and complex dialog strategies.

When the learning of the different modules are independent as in pipeline-methods, two main limitations arise. First, it is hard to propagate end user feedback to all the different upstream modules. Second, interdependence of the different independently learned modules mandates careful adaptation. Since the input of one module depends on the output of another, modifying one module affects the performance of others. Any adaptation to one module due to changes in domain/data/training method needs careful adaptation of other modules to achieve global optimization. This often requires a lot of human effort.

Deep learning has opened up the possibility of performing end-to-end learning of the entire dialog system (all modules together) directly from data. End-to-end training allows the propagation of end user's feedback to all the upstream modules and also enables automatic adaptation of the different modules in the dialog system accounting for how changes in one module affects the others, thereby overcoming the two main limitations of pipeline-methods. Since the entire dialog system is learned directly from the dialog task's data, the design of the system need not make any assumption about the dialog domain and is hence applicable to new domains with different training data, without domain-specific hand-crafting. Learning neural network based goal-oriented dialog systems end-to-end has shown great promise and gained a lot of interest in recent years (Bordes, Boureau, and Weston 2017; Liu and Lane 2017; Wen, Vandyke, et al. 2017). With deep neural networks, that have the potential to capture the complexity of human dialog in natural language, end-to-end learning of neural goal-oriented dialog systems holds the promise of achieving the goal of seamless integration of dialog systems into our everyday lives. In this thesis we identify some of the challenges in end-to-end learning of neural goal-oriented dialog systems and propose methods to address them.

1.1 Contributions

We identify four important challenges in end-to-end learning of neural goal-oriented dialog systems: 1) The challenge posed by the presence of a large number of named entities in goal-oriented dialog tasks. 2) The challenge of performing supervised learning of goal-oriented dialog systems with multiple valid next utterances. 3) The challenge of handling new user behaviors during deployment of a trained dialog system. 4) The challenge of requiring large amounts of training data for each new dialog task of interest. The contributions of this thesis are outlined below.

1.1.1 NE-Table: A Neural Key-Value Table for Named Entities

The ability to effectively encode, compare and retrieve Named Entities (NE) plays a crucial role in the success of goal-oriented dialog tasks. The presence of a very large number of named entities (e.g., different restaurant names and location names) especially in large knowledge bases, presents a serious challenge to traditional ways of encoding them using neural networks. The traditional way of treating named entities as any other word and adding them to the vocabulary leads to an explosion in the vocabulary size and is unable to learn *good* neural embeddings for rarely occurring NEs. The proposed alternative of using random embeddings for all the named entities leads to a loss of semantics. The other proposed alternative is to replace all the named entities with their NE tags based on their type. This removes the ability to reference and distinguish the different individual NEs with the same NE type. In many scenarios it is easier and more accurate to work with the actual exact values of NEs rather than their neural embeddings, like providing a phone number to a user. None of the above neural methods have the ability to work with exact NE values directly.

We propose a novel method that addresses all the aforementioned issues. There are three aspects to our method.

- On-the-fly-generation: Neural embeddings for the NEs are generated on the fly using their context information. This avoids the explosion in vocabulary size, while still providing meaningful and distinguishable neural embeddings for the different NEs.
- Key-Value-Table: The generated embeddings are stored in a table (*NE-Table*), with embeddings as the keys (key-embeddings) and exact NEs as the values (NE-values).
- On-the-fly-Retrieval: The NE-values can later be retrieved from the *NE-Table* by attending over the key-embeddings, providing the ability to interact with exact NE values.

1.1.2 Learning With Multiple Answers

The most common method for end-to-end learning of neural goal-oriented dialog systems at the moment is to collect a dataset of humans performing the task of interest and use supervised learning to train the dialog model to mimic the human agents in the dataset. There could be multiple ways/strategies to complete a given task. When the dataset is collected from different agents performing the task, these different ways of solving the task get reflected in it. These variations could be as simple as the difference in the order in which the agent asks the information from the user, or as complex as following a completely different line of questions/answers to complete the task. For example, in an IT support scenario, there could be different strategies to resolve an issue with a given piece of software. Therefore, in a dialog, there can be multiple valid next utterances at any point, but we have access to only one of them in a particular dialog in the dataset. During supervised learning the dialog system is trained to predict only one of the multiple valid next utterances. This results in reducing the probability of other valid next utterances for that dialog and leads to contradicting updates during training.

We propose a method that learns different valid next utterances without the learning of one interfering with the learning of the others much. Our proposed method uses only certain parts of the dialog state vector to predict a particular valid next utterance in a given dialog. The update from a given dialog data point affects only the parts of the network that were responsible for the prediction of that particular next utterance. The dialog system can retain other parts of the state vector and values in the network that stored information about other valid next utterances. This is achieved by generating a mask which decides which parts of the dialog state vector should be used for predicting a particular next utterance. The mask is learned as a function of current dialog state and also the actual next utterance present in the given dialog data point. Once the dialog system has learned different dialog responses and behaviors from the dataset safely, we fine-tune the dialog system using reinforcement learning to choose between the learned different valid next utterances (without access to the actual next utterance for masking). The dialog system might settle on a unique behavior that it finds best for it to perform the task and use that behavior during test time as well.

1.1.3 Learning with Maximal User Task Success and Minimal Human Agent Use

The inability to gracefully handle new user behaviors during deployment has limited the use of end-to-end learned neural goal-oriented dialog systems in the real world. New behaviors can be for example a different way of a user asking/providing certain information or could also be as simple as a user utterance with Out-Of-Vocabulary (OOV) words. The dialog system fails when these new behaviors are beyond the generalization capabilities of the trained systems. This happens generally because of the limited coverage of training data collected. For an enterprise that uses this dialog system, these failures could affect their business, as the unreliability of the dialog system might result in losing both current and future users/customers. While the dialog system fails for new user behaviors, it can perform the task well for the majority of user behaviors if trained on reasonable training data. However, such failures have restricted the deployment of end-to-end learned neural goal-oriented dialog systems and have forced enterprises to either rely completely on human agents or on other restrictive systems where the users have very little freedom in dialog.

We propose a method that learns to anticipate failure and transfer the dialog to human agents to maximise user's task success. The proposed method learns to make this decision through trial and error (reinforcement learning) using the feedback that it receives during deployment. The proposed method also learns end-to-end online from the human agent's response to reduce the human agent's load over time. We set the following three goals for our method:

- Maximize task success rate for the users by transferring to a human agent in cases where the dialog system might fail.
- Minimize the use of human agents by transferring to a human agent only when it is essential.
- Learn online from the human agent's response to reduce the use of human agents over time.

The reward function for the decision maker that chooses between the dialog system and the human agent allows the designer to choose the trade off between overall task success and the load on human agents.

1.1.4 Learning to Learn From Related Tasks

End-to-end learning of neural goal-oriented dialog systems requires large amounts of training data. The generalization performance of the trained dialog system critically depends on the quantity and quality of the training data used. This means for each and every new goal-oriented dialog task of interest large amounts of data needs to be collected. While end-to-end learning of neural goal-oriented dialog systems have saved human effort from handcrafting features/rules and careful module adaptation as compared to rule based systems and pipeline-methods respectively, they have increased the human effort involved in data collection. Collecting data can be a costly and time consuming process.

Instead of collecting large amounts of data for each new dialog task of interest, we show that we can end-to-end learn neural goal-oriented dialog systems for a new task with only limited amounts of training data by utilising already available data from a related dialog task, task that has parts/subtasks that are similar/overlapping with the new task of interest. The key challenge is to identify which data points of the related task data to learn from and which data points to not learn from. We want the dialog system to learn from the related tasks data in a way that helps the dialog system improve its performance on the new task of interest and not degrade it. We use a meta-gradient based method to learn to assign weights to related task data points during training to select which data points of the related task data to learn from and which data points not to learn from so as to perform well on the new task of interest.

1.2 Thesis Structure

The rest of thesis is organized as follows. Chapter 2 provides some background on the goal-oriented dialog tasks and the neural architecture of the dialog models used in the thesis. Chapter 3 talks about the challenges in interacting with named entities in goal-oriented dialog system and presents our method of building a neural key-value table as a solution. Chapter 4 discusses the issues with using traditional supervised learning methods in the presence of multiple valid next utterances and presents our two phase method of masking and reinforcement learning to counter the issue. In chapter 5, we present our method for learning to minimally use human agents for dialog when essential to maximise user task success. In chapter 6, we present our work on learning to learn from related tasks to improve performance on a new task of interest with limited data. Chapter 7 concludes with the summary of the thesis' contribution and discussion of possible future work.

CHAPTER 2

Background

In this chapter, we provide the background necessary to understand the rest of the thesis. Section 2.1 provides an introduction to end-to-end learning of neural goal-oriented dialog systems. Section 2.2 introduces the dialog tasks that are used in this thesis to evaluate goal-oriented dialog systems followed by section 2.3, which describes the Memory Networks neural architecture that is used in this thesis to build neural dialog systems.

2.1 End-to-end Learning of Neural Goal-Oriented Dialog Systems

End-to-end learned neural goal-oriented dialog systems (commonly referred to as end-to-end goal-oriented dialog systems) allow training of the entire dialog system directly from the dialog data. This reduces the domain-specific handcrafting often involved in the pipeline-methods. For example, most traditional goal-oriented dialog systems are based on slot filling with handcrafted domain-specific slots for the dialog state. This is inherently limiting for the task itself and also makes it hard to adapt to new domains. Training directly from the data removes the need not make any assumptions about the domain and hence the learning method is applicable to new domains. End-to-end training also removes some of the other inherent limitations associated with the pipeline-methods: 1) limitations of not being able to pass end user's feedback to all upstream modules and 2) limitations of having to adapt all modules in the pipeline carefully (with human effort) to accommodate for any changes to one of the modules. End-to-end learning automatically takes care of adaptation of the different modules in the dialog system accounting for how changes to one module affects the others.

Building goal-oriented dialog systems using neural networks allows end-to-end training. Deep neural networks which are universal approximators (Hornik, Stinchcombe, and White 1989) have the potential to capture the complexity of human dialog in natural language. The increase in the availability of faster compute, larger training data, better programming libraries and improvements in deep learning architectures and optimization methods have all together paved the way for successfully learning neural end-to-end goal-oriented dialog systems. There are two broad ways in which present day neural end-to-end goal-oriented dialog systems are trained: 1) Supervised Learning (SL) and 2) Reinforcement Learning (RL). In the RL setting, the dialog system learns through trial and error with reinforcement (rewards at the end or at key dialog points) from a human or a human simulator. In many cases, it is difficult to define and award appropriate rewards. It is also difficult to learn language from scratch through scalar rewards. Training of dialog systems using RL alone generally requires a lot of training interaction. In order to handle these challenges, RL methods are almost always complimented with a SL phase of pre-training. In SL setting, a fixed set of dialog data is collected from human agents performing the task of interest and the dialog system is trained to imitate the human agents in the data. Bordes, Boureau, and Weston (2017) and Wen, Vandyke, et al. (2017) and Eric and Manning (2017) represent some of the important early end-to-end learning works which are based on supervised learning on a collected dataset. Zhao and Eskenazi (2016) and Liu and Lane (2017) represent some of the important early works on end-to-end learning of goal-oriented dialog system using reinforcement learning which allows for exploration of dialog control, and therefore more robustness.

Goal-oriented dialog systems also often require interaction with external knowledge sources. For example, for the task of restaurant reservation, the dialog system has to search for restaurants matching user preferences in an external knowledge base. Traditionally this is done through symbolic queries to knowledge bases, which makes the retrieval process non-differentiable, therefore unsuitable for end-to-end training. This is overcome by parameterizing the retrieval process using neural networks and making it differentiable. For example, Dhingra et al. (2017) and Eric and Manning (2017) learn a soft distribution over knowledge base entries and the dialog system uses that distribution for retrieval.

2.2 Goal-Oriented Dialog Tasks: bAbI

To evaluate end-to-end goal-oriented dialog systems, Bordes, Boureau, and Weston (2017) proposed a testbed: bAbI dialog tasks, a set of five goal-oriented dialog tasks. Since its release, the bAbI dialog tasks have been used in several research works (Liu and Perez 2017; Wu, Socher, and Xiong 2019). bAbI dialog tasks are synthetically constructed tasks in the restaurant reservation domain. The goal of bAbI dialog tasks is not to improve the state-of-the-art in the narrow domain of restaurant reservation, but rather to test the strength and weakness of end-to-end goal-oriented dialog systems in a reproducible and lightweight way. The restaurant reservation task is split into several sub-tasks to test the different crucial capabilities required in a goal-oriented dialog system separately.

The dialogs are based on an underlying Knowledge Base (KB) with information about restaurants, such as the cuisine, location, price range, rating, address and phone number. Tasks 1 (Issuing API calls) and 2 (Updating API calls) test the dialog system to implicitly track dialog state, whereas Task 3 (Displaying options) and 4 (Providing extra information) check if the system can learn to use KB facts in a dialog setting. Task 5 (Conducting full dialogs) combines all tasks.

Task 1: Issuing API calls. A user's request for reservation has 0-4 of the required fields (cuisine, location, price range and number of people). The dialog system should ask for the remaining required information and make the appropriate API call.

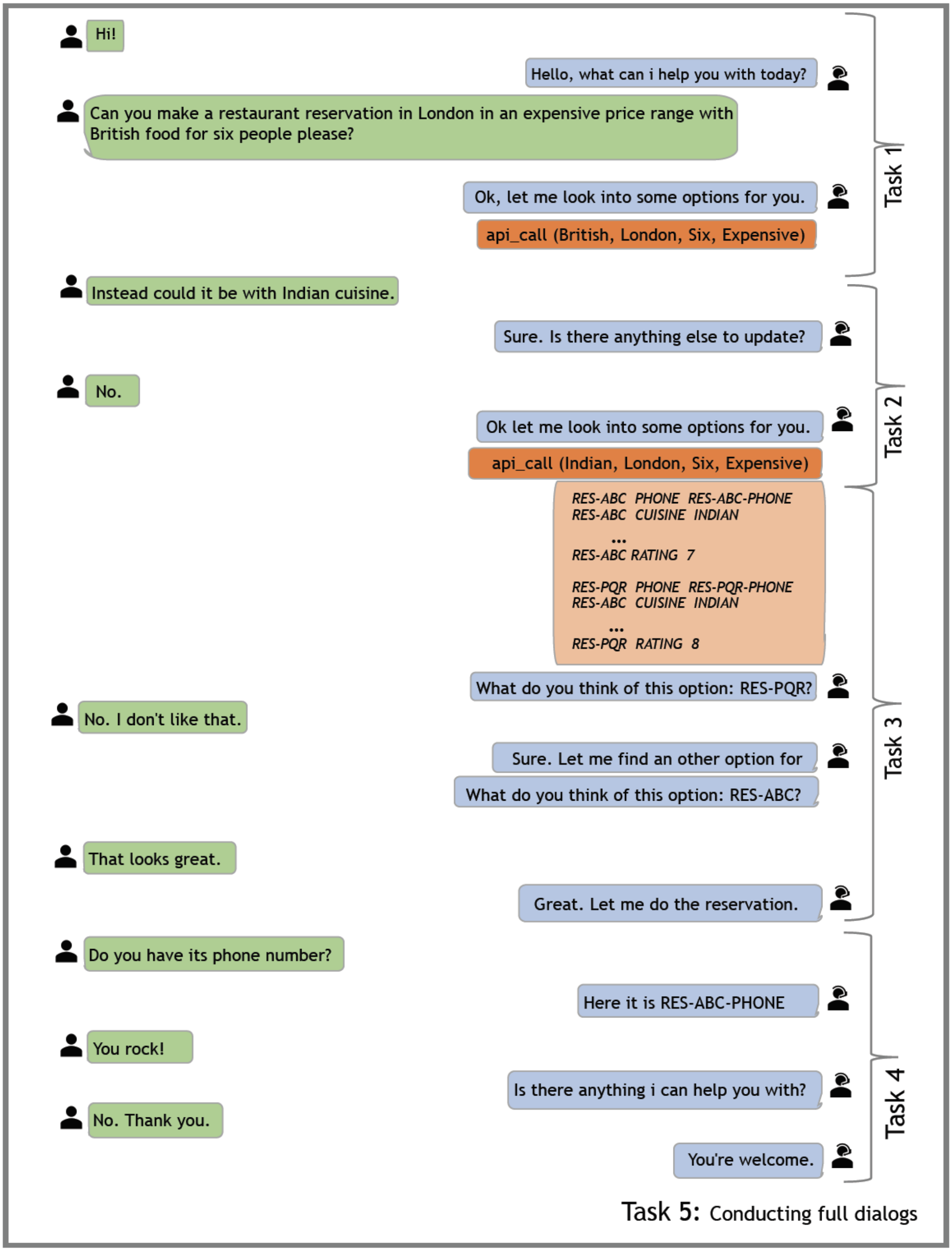
Task 2: Updating API calls. In this task, the user provides all the required information for the API call, and the task starts with the dialog system making the appropriate API call. The user might then later update some of the fields. The dialog system has to gather all the updates and make the corresponding API call with the updated information.

Task 3: Displaying options. The information about the restaurants corresponding to a given user query is provided as part of the dialog history. The dialog system has to recommend restaurants based on their ratings until the user accepts a restaurant.

Task 4: Providing extra information. The dialog starts as if the user has finalised the restaurant that they want to reserve. The information about the restaurant from the knowledge base is added as part of the dialog history. The dialog system should provide information about the restaurants address, phone number or both depending upon the user's query.

Task 5: Conducting full dialogs. Tasks 1-4 are combined to generate full dialog.

Figure 2.1 shows a simplified example of the different bAbI dialog tasks. The system is evaluated in a retrieval setting. At each turn of the dialog, the system has to select the correct response from a list of possible candidates (all system utterances that occur in the dataset). The dataset consists of 1000 dialogs each in train, validation, test and Out-Of-Vocabulary(OOV) test set. The OOV test set tests generalization to entities (restaurants,



Hi!

Hello, what can i help you with today?

Can you make a restaurant reservation in London in an expensive price range with British food for six people please?

Ok, let me look into some options for you.

api_call (British, London, Six, Expensive)

Instead could it be with Indian cuisine.

Sure. Is there anything else to update?

No.

Ok let me look into some options for you.

api_call (Indian, London, Six, Expensive)

RES-ABC PHONE RES-ABC-PHONE
RES-ABC CUISINE INDIAN
...
RES-ABC RATING 7

RES-PQR PHONE RES-PQR-PHONE
RES-ABC CUISINE INDIAN
...
RES-PQR RATING 8

What do you think of this option: RES-PQR?

No. I don't like that.

Sure. Let me find an other option for

What do you think of this option: RES-ABC?

That looks great.

Great. Let me do the reservation.

Do you have its phone number?

Here it is RES-ABC-PHONE

You rock!

Is there anything i can help you with?

No. Thank you.

You're welcome.

Figure 2.1: A dialog from the bAbI dialog tasks dataset. A user (in green) chats with a dialog system (utterances in blue and API calls in orange) to book a table at a restaurant.

cuisines, locations) that are not seen in the training data. In this thesis we use bAbI dialog tasks and some modified versions of them to evaluate many of our proposed methods.

2.3 Memory Networks For End-To-End Goal-Oriented Dialog Tasks

Memory networks, proposed by Weston, Chopra, and Bordes (2014) and extended for end-to-end training by Sukhbaatar et al. (2015), are a class of models that have an explicit external memory. In memory networks, the inputs are stored in an external memory. When a query comes in, the relevant information from the memory is retrieved by iteratively attending over the memory. The retrieved information is then used to predict the answer. Memory networks have been widely applied in a variety of natural language processing tasks such as language modelling, part-of-speech tagging, sentiment analysis, question answering and dialog. Works showing the application of memory networks on these varied tasks include Sukhbaatar et al. (2015), and Kumar et al. (2016).

Here we describe the end-to-end memory networks proposed by Sukhbaatar et al. (2015). They perform well (outperform end-to-end architectures based on Recurrent Neural Networks (RNN)) on the bAbI dialog tasks discussed in Section 2.2 and used in this thesis. The dialog systems in this thesis are built based on end-to-end memory networks. Figure 2.2 shows a single layer version of memory networks. First, the dialog history is stored in the external memory (both the user and system utterances). Every sentence from the dialog history is encoded into two separate neural embeddings shown as a_i and c_i in figure 2.2 and stored in different memory slots. The embedding representation of a sentence is calculated by adding the neural embeddings of all the words in that sentence.

The new user utterance acts as a query and is embedded into a neural embedding u. Based on the query embedding, relevant memory slots are selected. Inner product between u and the memory embeddings a_i followed by a softmax provides the attention weights (p_i) for the different memory slots:

$$p_i = \text{Softmax}(u^T a_i). \tag{2.1}$$

An output vector o is computed by the weighted sum of memory embeddings c_i with their corresponding weights p_i as shown below:

$$o = \sum_i p_i c_i. \tag{2.2}$$

The output embedding o is the overall context embedding which has selected relevant in-

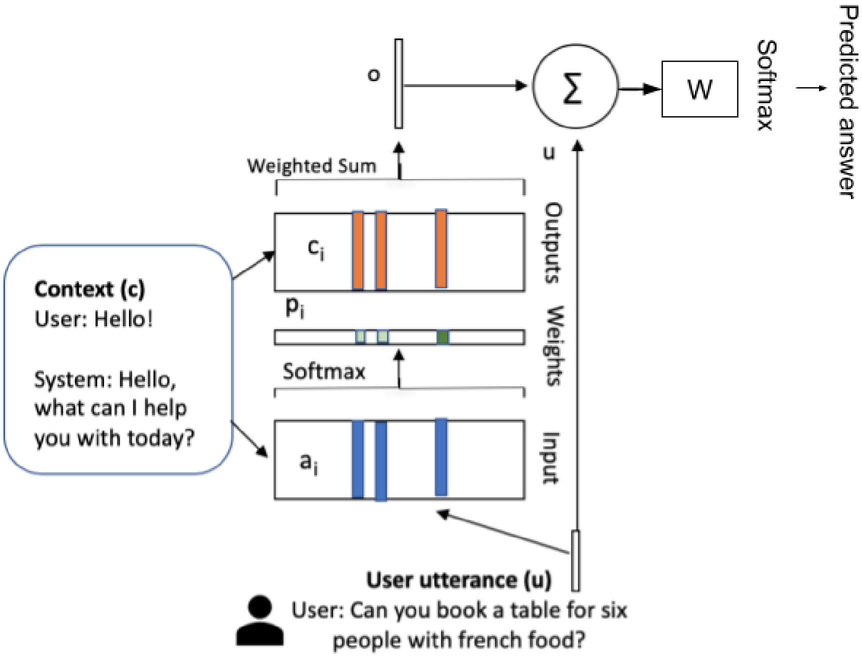


Figure 2.2: A single layer version of end-to-end memory networks

formation of the dialog history from the memory. The output embedding o is added to the query embedding (u) to obtain the dialog state embedding s,

$$s = o + u. \tag{2.3}$$

This process can be repeated multiple times (referred to as hops) to iteratively retrieve information from the memory by using the current dialog state embedding s as the new query embedding u. The last dialog state is then used to predict the next system response. The neural network parameters can either be shared or not shared across different hops. The entire model can be trained using back propagation by minimizing the standard cross-entropy loss between the predicted response and the correct ground truth response.

CHAPTER 3

NE-Table: A Neural Key-Value Table for Named Entities

The ability to effectively encode, compare and retrieve Named Entities (NE) plays a crucial role in the success of goal-oriented dialog tasks. The presence of a very large number of named entities (e.g., different restaurant names, location names and phone numbers) with rare or Out-Of-Vocabulary occurrence of any individual named entity (e.g., a particular phone number) presents a serious challenge to the traditional way of encoding them using neural networks. Adding them to the vocabulary leads to an explosion in vocabulary size, using random embeddings lead to loss of semantics, and replacing them with NE tags removes the ability to reference and distinguish the different individual NEs. We propose a novel method to tackles these challenges. Our proposed method encodes any NE on the fly using the context and stores them in a key-value table for comparison and retrieval. Our evaluation results show that the proposed method can be effective in dealing with both in-vocabulary and OOV NEs.

Published at: Recent Advances in Natural Language Processing (RANLP 2019).
Authors: Janarthanan Rajendran*, Jatin Ganhotra*, Xiaoxiao Guo, Mo Yu, Satinder Singh, and Lazaros C. Polymenakos.
Collaborations: IBM Research.

---

*Equal contribution

3.1 Introduction

Named Entities (NEs) play a crucial role in a goal-oriented dialog task's successful completion. Goal-oriented dialog tasks generally also involve interaction with external knowledge sources such as Knowledge Bases (KB) which could have a large number of NEs. For example, a dialog system that performs the task restaurant reservation has to deal with NEs such as restaurant names, restaurant locations, restaurant phone numbers and food item's names for successfully completing the task.

The presence of a very large number of named entities with rare or Out-Of-Vocabulary occurrence of any individual named entity (e.g., a particular phone number) presents a serious challenge to the traditional way of encoding them using neural networks. The traditional method is to treat NEs as any other word and add each and every NE (including those in the KB) to the vocabulary. For real world tasks, especially those with large KBs, this causes an explosion in the vocabulary size and hence the number of neural network parameters to learn. There is also the problem of not being able to learn *good* neural embeddings for individual NEs, as individual NEs (e.g., a particular phone number) generally occur only a few times in a dataset. It is also unclear how to represent new out-of-vocabulary named entities that arise during deployment. Past work has tried a few methods for addressing this challenge to neural methods posed by NEs. One method is to encode all the NEs with random embeddings and keep them fixed throughout (Yin, Lu, et al. 2016) (avoids explosion in number of neural network parameters to learn), but here we lose the meaning associated with the neural embeddings. This reduces the ability to compare and retrieve named entities effectively. We also risk interference and correlation with other named entities embeddings in unexpected ways. Another method is to first recognize the NE-type with either NE taggers (Finkel, Grenager, and Manning 2005) or entity linkers (Cucerzan 2007; Guo, Chang, and Kiciman 2013), and then replace them with NE-type tags. For example, all location names could be replaced with the tag *NE_location*. This prevents the explosion in vocabulary size; however, the system loses the ability to distinguish and reference different NEs of the same type. There is also the possibility of new NEs arising during test time. In fact, many of the Out-Of-Vocabulary (OOV) words that arise during test time in many goal-oriented dialog tasks (e.g., Bordes, Boureau, and Weston (2017)) are NEs. Furthermore, in many scenarios it is easier and accurate to work with the actual exact values of NEs rather than neural embeddings, like providing a phone number to a user. None of the above neural methods have the ability to work with exact NE values directly.

We propose a novel neural method that addresses all the aforementioned issues. There are three aspects to our method.

- On-the-fly-generation: Neural embeddings for the NEs are generated on the fly using their context information. This avoids the explosion in vocabulary size, while still providing meaningful and distinguishable neural embeddings for the different NEs.
- Key-Value-Table: The generated embeddings are stored in a table (*NE-Table*), with embeddings as the keys (key-embeddings) and exact NEs as the values (NE-values).
- On-the-fly-Retrieval: The NE-values can later be retrieved from the *NE-Table* by attending over the key-embeddings, providing the ability to interact with exact NE values.

Named Entities (NEs) play a crucial role in successful task completion not just in goal-oriented dialog tasks, but also in other Natural Language Processing (NLP) tasks such as Question-Answering (QA), reading comprehension and machine translation. For example, a QA system for retrieving information about courses offered at a university has to deal with NEs such as course numbers and instructor names. We demonstrate our proposed method on a reading-comprehension task, a simple structured Question-Answering (QA) task, and three goal-oriented dialog tasks. Our method achieves 10% increase in accuracy for Reading-Comprehension, 19% increase for structured-QA and around 90% increase for goal-oriented dialog tasks, with respect to their corresponding baselines.

3.2 Proposed Method

Our proposed method (Figure 3.1) has three major aspects to it.

On-the-fly-generation. Neural embeddings for the NEs are generated on the fly using their context information (shown as the *NE-Embedding Generation Module* in Fig 3.1), instead of adding them to the vocabulary. The context information depends on the task. For a dialog task, the context is the full dialog so far, including the present utterance which has the NE in it. For the QA task, context is the sentence in which the NE appears. For the Reading Comprehension task, the sentence where the NE occurs or potentially the full story can be used as the context. The context could also include the NE-type information when available. The *NE-Embedding Generation Module*, denoted (f_ϕ), takes the context embedding as input and outputs the NE-Embedding. In our experiments, f_ϕ is a multi-layer perceptron (MLP). The problem of explosion in vocabulary size is avoided, as NEs are not part of the vocabulary and the NE-Embeddings are generated on the fly. Our proposed method also generates unique embeddings for different NEs with the same NE-type. This

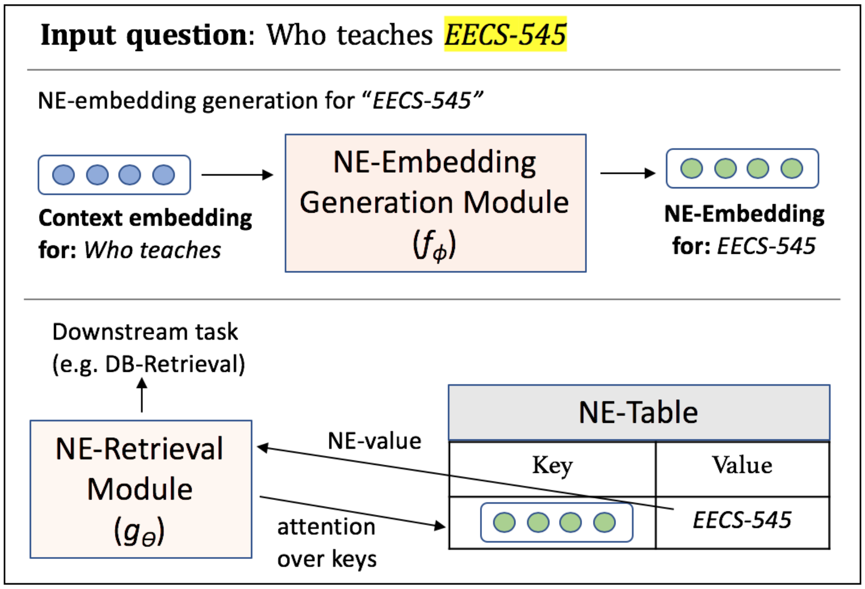

Input question: Who teaches *EECS-545*
NE-embedding generation for "*EECS-545*"
Context embedding for: *Who teaches*
NE-Embedding Generation Module (f_ϕ)
NE-Embedding for: *EECS-545*
Downstream task (e.g. DB-Retrieval)
NE-Retrieval Module (g_θ)
NE-value
attention over keys
NE-Table
Key
Value
EECS-545

Figure 3.1: For input question - *Who teaches EECS-545*, the *NE-Embedding Generation Module* (f_ϕ) takes the context embedding as input and generates a NE-Embedding for the NE *EECS-545*. The NE-Embedding is stored in *NE-Table* with its actual value *EECS-545*. The *NE-Retrieval Module* (g_θ) performs attention over the keys in *NE-Table* to retrieve the NE-value. We show a simple example here to illustrate f_ϕ and g_θ. Depending on the task, the context can vary and the *NE-Table* can have more entries.

is better than replacing a NE with its NE-type as that results in all NEs with the same NE-type having the same embedding and hence, losing the ability to distinguish different NEs with the same NE-type. The generated NE-Embeddings are meaningful as they are learned from the context, in comparison to fixed random embeddings and can also be used as the learned neural embedding for that NE word from thereon.

Key-Value-Table. As discussed in the previous section, there are many scenarios where it is easier and more accurate to work with the exact values of NEs rather than their neural embeddings, like providing a phone number to a user or searching for a instructor name over a KB. For this purpose, the generated NE-Embedding, along with its exact NE value is stored in a table, *NE-Table*, as a key-value pair, with the embedding as key (key-embedding) and the exact NE as value (NE-value).

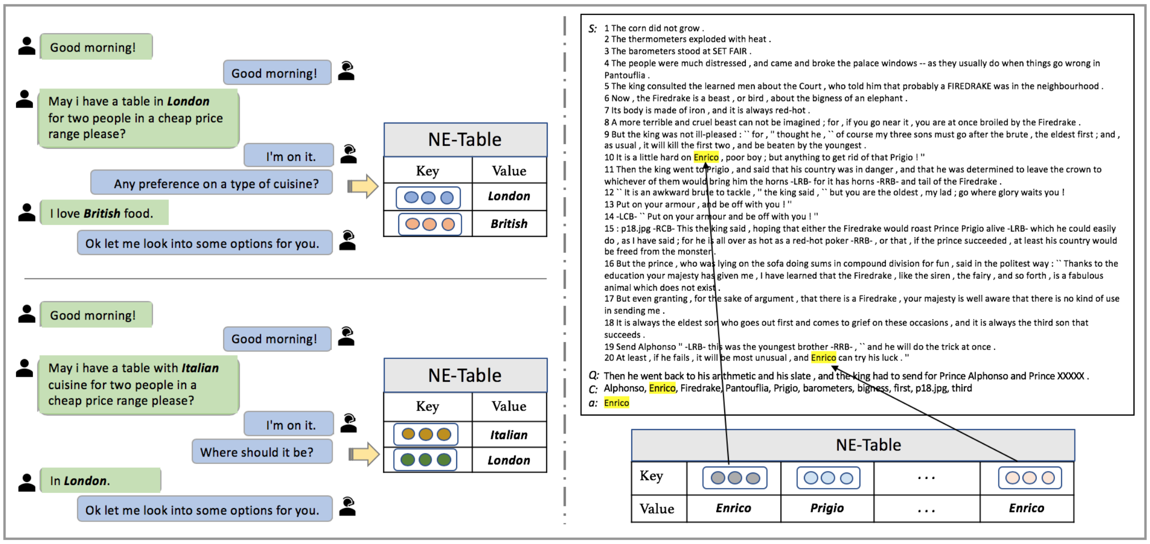
On-the-fly-Retrieval. The NE-value can later be retrieved from the *NE-Table* by performing attention over the key-embeddings in the *NE-Table*. This is performed by the *NE-Retrieval Module* (g_θ) shown in Figure 3.1. In our experiments, g_θ is an MLP. The input to *NE-Retrieval Module* also depends on the task. For dialog task, the dialog state vector is used, which has information of the dialog so far. For QA task, the encoding of the input question is used. For Reading Comprehension task, the full story is used as input to the retrieval module. The retrieved NE-value can be used in the output utterance (e.g., providing a phone number) or to do an exact match over values in a KB (e.g., searching for a instructor name in a KB).

While the matching of a NE-value retrieved from the *NE-Table*, with other NEs in the KB is performed through exact value match, the actual retrieval of that NE from the *NE-Table* happens using attention in the neural embedding space (using a dot product in our experiments). This allows the training of the *NE-Retrieval Module* using the supervision obtained from the downstream module (e.g., a KB retrieval module) that uses the retrieved NE-value. This also provides supervision for training the *NE-Embedding Generation Module*. Our intuition is that, this would encourage the *NE-Embedding Generation Module* to generate embeddings for the NEs such that the embeddings have relevant and enough information to allow the *NE-Retrieval module* to attend and retrieve them correctly when required later.

Since the embeddings are generated on the fly using the context, the proposed method works equally well for new NEs that come during test time as it would for the NEs present in the training data. We show examples for *NE-Table* for the dialog and Reading Comprehension task in Figure 3.2. A new, separate *NE-Table* is created for each data instance based on the task. For example, in the dialog task, each dialog will have its own separate *NE-Table*. Only the NEs that have appeared in the dialog so far will be present in its corresponding *NE-Table*. The same NE occurring in different dialogs will have different dialog-context-dependent embeddings in their corresponding *NE-Table*. Similarly, for the reading comprehension task, each story will have a separate *NE-Table* with the NEs present in that story and for the QA task, each question will have a separate *NE-Table*. Note that, a NE that occurs multiple times in the same dialog/story/question will also have multiple unique embeddings in the *NE-Table* because of differing contexts as shown in Figure 3.2 (right).

3.3 Experiments and Results

We evaluate our proposed method on three types of tasks: a reading-comprehension task, a structured-QA task and three goal-oriented dialog tasks. Our proposed method is generic and can be added to the state-of-the-art approaches for these tasks. But instead of implementing 3 separate specialized neural architectures, we chose the end-to-end memory network architecture from Sukhbaatar et al. (2015) described in Section 2.3 as the base


Good morning!

Good morning!

May i have a table in ***London*** for two people in a cheap price range please?

I'm on it.

Any preference on a type of cuisine?

I love ***British*** food.

Ok let me look into some options for you.

Good morning!

Good morning!

May i have a table with ***Italian*** cuisine for two people in a cheap price range please?

I'm on it.

Where should it be?

In ***London***.

Ok let me look into some options for you.

S: 1 The corn did not grow .
2 The thermometers exploded with heat .
3 The barometers stood at SET FAIR .
4 The people were much distressed , and came and broke the palace windows -- as they usually do when things go wrong in Pantouflia .
5 The king consulted the learned men about the Court , who told him that probably a FIREDRAKE was in the neighbourhood .
6 Now , the Firedrake is a beast , or bird , about the bigness of an elephant .
7 Its body is made of iron , and it is always red-hot .
8 A more terrible and cruel beast can not be imagined ; for , if you go near it , you are at once broiled by the Firedrake .
9 But the king was not ill-pleased : `` for , '' thought he , `` of course my three sons must go after the brute , the eldest first ; and , as usual , it will kill the first two , and be beaten by the youngest .
10 It is a little hard on Enrico , poor boy ; but anything to get rid of that Prigio ! ''
11 Then the king went to Prigio , and said that his country was in danger , and that he was determined to leave the crown to whichever of them would bring him the horns -LRB- for it has horns -RRB- and tail of the Firedrake .
12 `` It is an awkward brute to tackle , '' the king said , `` but you are the oldest , my lad ; go where glory waits you !
13 Put on your armour , and be off with you ! ''
14 -LCB- `` Put on your armour and be off with you ! ''
15 : p18.jpg -RCB- This the king said , hoping that either the Firedrake would roast Prince Prigio alive -LRB- which he could easily do , as I have said ; for he is all over as hot as a red-hot poker -RRB- , or that , if the prince succeeded , at least his country would be freed from the monster .
16 But the prince , who was lying on the sofa doing sums in compound division for fun , said in the politest way : `` Thanks to the education your majesty has given me , I have learned that the Firedrake , like the siren , the fairy , and so forth , is a fabulous animal which does not exist .
17 But even granting , for the sake of argument , that there is a Firedrake , your majesty is well aware that there is no kind of use in sending me .
18 It is always the eldest son who goes out first and comes to grief on these occasions , and it is always the third son that succeeds .
19 Send Alphonso '' -LRB- this was the youngest brother -RRB- , `` and he will do the trick at once .
20 At least , if he fails , it will be most unusual , and Enrico can try his luck . ''

Q: Then he went back to his arithmetic and his slate , and the king had to send for Prince Alphonso and Prince XXXXX .
C: Alphonso, Enrico, Firedrake, Pantouflia, Prigio, barometers, bigness, first, p18.jpg, third
a: Enrico

Figure 3.2: *Left:* Two dialogs from bAbI task-1. A user (in green) chats with a dialog system (in blue) to book a table. Each dialog has its own separate *NE-Table* and a separate NE-Embedding is generated for the NE *London* though it appears in both dialogs. *Right:* Question from CBT. NE *Enrico* (highlighted in yellow) occurs twice in the context *S*, where a separate NE-Embedding is generated for each occurrence.

architecture for our tasks. This allows us to evaluate the advantage gained by adding our method to the base architecture instead of trying to get state-of-the-art performance in a particular task/dataset.

3.3.1 Reading Comprehension Task

The Children's Book Test dataset (CBT), built from children's books from Project Gutenberg, was introduced by Hill et al. (2015) to test the role of memory and context in language processing and understanding. Questions are formed by enumerating 21 consecutive sentences, where the first 20 sentences form the *story* (*S*), and a *word* (*a*) is removed from the 21st sentence, which then implicitly becomes the *query* (*q*). The specific task is to predict the correct answer word (*a*) from a set of 10 candidate words (*C*) present in the story or the query. We test our proposed method on the NE questions subset of the CBT dataset.

We use the Window memory architecture proposed by Hill et al. (2015) for the CBT dataset as our baseline. The baseline method (W/O-NE-Table) does not distinguish NEs from normal words, and all words (including NEs) are part of the vocabulary. In Memory Networks (Sukhbaatar et al. 2015), each complete sentence of *S* is encoded and represented in a separate memory slot. For the CBT, this setting would yield exactly 20 memories for *S*. In Window memory, instead of a full sentence from the story, a phrase is encoded and represented in a separate memory slot. Each phrase s corresponds to a window of text from the story *S* centred on an individual mention of a candidate *c* in *S*. The window is constructed as span of words $w_{i-(b-1)/2}$ ... w_i ... $w_{i+(b-1)/2}$ where b is window size and $w_i \in C$ is an instance of one of the candidate words in the question. We perform two baseline evaluations: encoding the windows using a) Bag-of-Words (BoW) and b) LSTM (Hochreiter and Schmidhuber 1997).

Method	Test Accuracy (%)
W/O-NE-Table (BoW)	41.69
W/O-NE-Table (LSTM)	41.10
With-NE-Table (BoW)	51.28
With-NE-Table (LSTM)	51.08

Table 3.1: Test results on CBT-NE dataset

For each NE (identified by the Stanford Core NLP NER system (Manning et al. 2014)), the corresponding window is fed to an LSTM to create the context embedding. The context embedding is used as input to *NE-Embedding Generation Module* (f_ϕ), as shown in Figure 3.1, to generate the corresponding NE-Embedding, which is added to the *NE-Table*. The NE-Embeddings are also added to window memory, in place of the NEs. The query (*q*) embedding is used to attend over the memory (list of encoded window memory slots) to get relevant information from the memory. The internal state generated is given as input to the *NE-Retrieval Module* (g_θ), for retrieving the correct NE answer (*a*). Table 3.1 shows that replacing the baseline with our method achieves higher performance on both BoW and LSTM baseed baselines. We use a window size of 5 as in Hill et al. (2015). We think that since the window size is small, both BoW and LSTM encoding perform similarly.

The hyperparameters used for baseline *W/O-NE-Table* methods are as follows: hops = 1, embedding_size = 100, batch_size = 16. The *W/O-NE-Table (LSTM)* uses an LSTM for encoding windows with hidden_units = 100. They are trained using stochastic gradient descent (learning rate = 0.05), minimizing a standard cross-entropy loss between predicted answer $\hat{a}$ and the correct answer a. We use the same embedding matrix for encoding both story and the query.

For our *With-NE-Table*, we use the same hyperparameters as mentioned above. The *With-NE-Table (LSTM)* uses separate LSTM cells for 1) generating the context embedding for *NE-Embedding Generation Module* (f_ϕ) and 2) encoding the windows.

To further evaluate the impact of OOV NEs, we created additional OOV test sets by replacing NEs in the test set with new NEs not present in the train and validation sets. We generate 5 such OOV test sets with varying percentage of OOV NEs (20%, 40%,

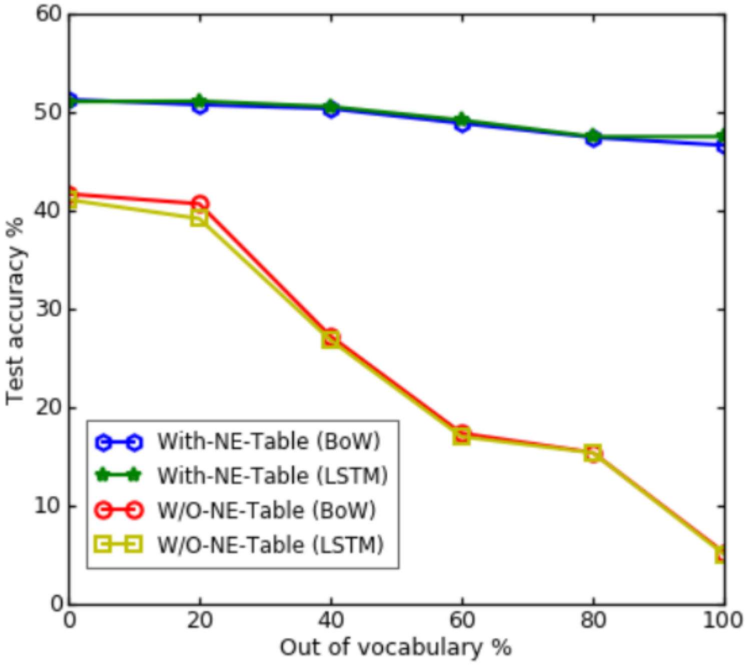


Figure 3.3: Results on CBT-NE OOV test sets

60%, 80% and 100%).The OOV test sets are available - `https://github.com/IBM/ne-table-datasets/`. Figure 3.3 shows the comparison of our method with the baselines on OOV test-sets. The baseline methods perform poorly as OOV% increases, decreasing to as low as 5% from 41%. We observe only a slight reduction in accuracy for the NE-Table methods from 51% to 46% because the new entities are also part of the windows, used to generate NE-Embeddings. These experiments illustrate that our methods performance is robust to OOV NEs.

The next two tasks, structured-QA and goal-oriented dialog involve retrieval from an external KB. This is performed by the *KB-Retrieval Module* (h_ψ), which uses a multiple-attention based neural retrieval mechanism. We describe this next and then present results on the 2 tasks.

3.3.2 Multiple-Attention Based Neural Retrieval Mechanism

3.3.2.1 Overview

In both structured-QA and goal-oriented dialog tasks, the external knowledge base is present in the form of a single knowledge base table. For example, in our structured-QA (which is about course offerings at a university) KB is in the form a single table where, each row corresponds to a course and the columns correspond to course attributes, such as course number, course name, instructor name, etc. Each column of the table has a column heading, which labels the attribute of that column. These headings are also part of the vocabulary. While the non-NEs present in the KB are part of the vocabulary and represented by their learned neural embeddings, the NEs are not part of the vocabulary and are represented by their exact values.

The *KB-Retrieval Module* performs attention over both attributes (columns) as well as rows to select the final cell(s) in 3 steps. In step 1, the column(s) that the final cell(s) belong to are selected by attention over the column heading embeddings. For the question *Who teaches EECS545?*, step 1 selects the column *'instructor name'*. In step 2, separate attention is performed over the column headings to select the columns, which are used to represent the rows (to retrieve the final cell) and column *'course number'* is selected. Step 3 is to do attention over the rows. For each non-NE column selected in step 2, the cell embeddings are added together along each row, to generate an embedding for each row. Attention is performed over these row embeddings to select row(s). For each NE-column selected in step 2, a NE-value is retrieved from the *NE-Table* to do an exact match search over that NE-column to select matching row(s). The intersection of these matching row(s) gives the final set of selected row(s), and their intersection with the set of column(s) selected in step 1 gives the retrieved cell(s). For our example, only one column is selected to represent the rows: *'course number'*, which is a NE-column. Therefore, a NE value is retrieved from the *NE-Table* (EECS545) and an exact match search is done over the *'course number'* column.

The input to the *KB-Retrieval Module* depends on the task. For the dialog task, the dialog state vector is used, which has the information of the dialog so far. For the QA task, the encoding of the input question is used. All the attention operations in our experiments are performed through dot product followed by a sigmoid operation, which allows for multiple selections. Note that *NE-Table* can potentially be used with other neural retrieval mechanisms. The multiple-attention mechanism described above is only one of the several neural retrieval mechanisms (Yin, Lu, et al. 2016).

3.3.2.2 Details

Figure 3.4 shows the schematic of the entire retrieval process. In order to retrieve a particular cell from the table, the system needs to find the correct column and row corresponding to it. The *KB-Retrieval Module* (h_ψ) does that by generating 3 different attention key embeddings (vectors): Attention over Columns for Columns (*ACC*), Attention over Columns for Rows (*ACR*), Attention over Rows for Rows (*ARR*).

The column(s) that the final retrieved cell(s) belong to, are selected by matching *ACC* key embeddings with the neural embeddings of the column headings (Course Number,

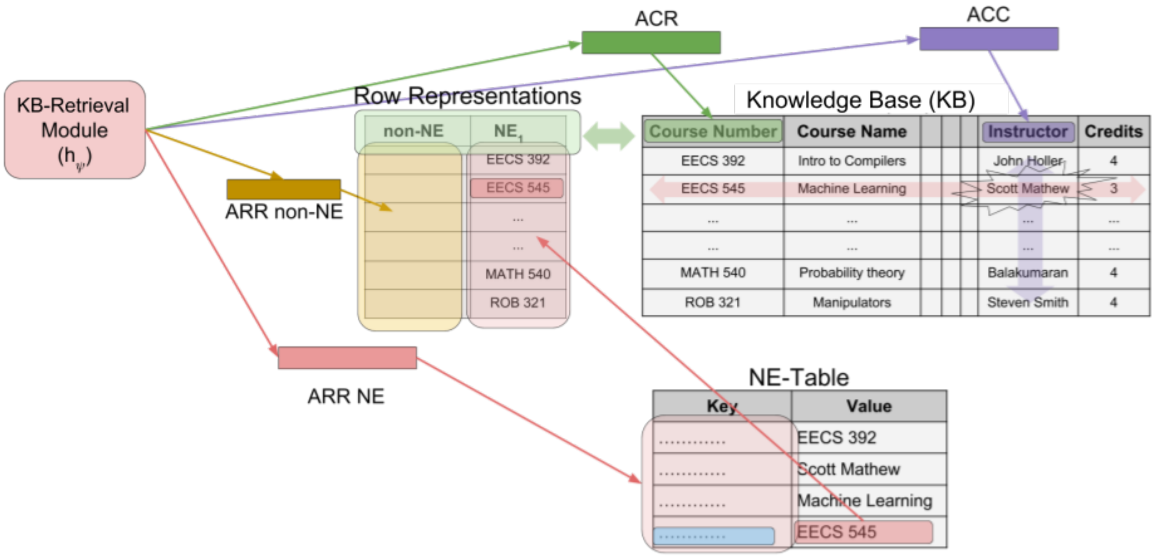


Figure 3.4: Multiple-attention based neural retrieval mechanism. The *KB-Retrieval Module* attends to the relevant rows and columns of the KB by generating attention key embeddings *ACC*, *ACR* and *ARR*.

Instructor, Credits etc). A separate *ACC* key embedding is generated for every column heading and matched with its embeddings to provide attention scores for all the columns. For the example, *Who teaches EECS545?*, the system would want to retrieve the name of the *Instructor*. Therefore, the *Instructor* column heading alone will have high attention score and be selected. In our experiments, the attention scores are computed through dot products followed by a sigmoid operation, which allows for multiple selections.

Now that the column(s) are chosen, the system has to select row(s), so that it can get the cell(s) it is looking for. Each row in the table contains the values (EECS545, Machine Learning, Scott Mathew etc) of several attributes (Course Number, Course Name, Instructor etc). But we want to assign attention scores to the rows based on particular attributes that are of interest (*Course Number* in this example). The column/attribute headings that the system has to attend to for selecting these relevant attributes are obtained by matching *ACR* (Attention over Columns for Rows) key embeddings with the neural embeddings of the different column headings.

The last step in the knowledge base retrieval process is to select the relevant rows using the *ARR* (Attention over Rows for Rows) key embedding. *ARR* is split into two parts *ARR NE* and *ARR non-NE*. In a general scenario, *ACR* can select multiple columns to represent the rows. For each selected column that is a NE column, a separate NE-value is retrieved from the *NE-Table* using a separate *ARR NE* embedding for each of them. These NE values are used to do exact match search along the corresponding columns (in the NE row representations) to select the matching rows. For the non-NE columns that are selected by *ACR*, their neural embeddings are combined together along each row to get a fixed vector representation for each row in the KB (weighted sum of their embeddings, weighted by the corresponding column attention scores). *ARR non-NE* is then used to match these representations for selecting rows. The intersection of the rows selected in the NE row representations and the non-NE row representations is the final set of selected rows.

In short, the dialog system can use neural embedding matching for non-NEs, exact value matching for NEs and therefore a combination of both to decide which rows to attend to. Depending on the number of columns and rows we match with, we select zero, one or more output cells. For our running example, *ARR NE* is used to match with the keys in the *NE-Table* to select the row corresponding to *EECS 545* and the value *EECS 545* is returned to do an exact match over the NE row representations (represented by the course number values). This gives us the row corresponding to *EECS 545* and hence the cell *Scott Mathew*. We could use our *NE-Table* idea with potentially many types of neural retrieval mechanisms to retrieve information from the KB. The multiple-attention based retrieval mechanism, described above, is only one such possible mechanism.

3.3.3 Structured-QA from KB

The task here is to retrieve an answer (single cell in a table) from KB in response to structured one line questions. We used the details of course offerings at University of Michigan to create structured Question-Answer (QA) pairs. The KB is a single table of 100 rows (Courses) and 4 columns (Course Number, Course Name, Department, Credits)[*], where course numbers and course names are treated as NEs. The QA pairs are generated through random sampling from the KB, following the format -

Q: `Col-1-type Col-1-value Col-2-type ?`

A: `Col-2-value`

with the following being a specific example:

Q: `Course Number EECS545 Credits ?` **A:** `4.`

500 QA pairs were created and split randomly between training and test set (400-100), where the random split results in some NEs (OOV) in the test set, not present in the training set. The task was specifically constructed to be simple to show the impact of OOV NEs on performance and evaluate our proposed method.

The experiments were performed on two methods. Both use a Recurrent Neural Network (RNN) to encode the question and use the multiple-attention based neural retrieval

---

[*] number of unique course numbers - 100, unique course names - 96, unique department names - 10 and unique credits - 4

mechanism to retrieve answers. The baseline method (*W/O-NE-Table*) does not distinguish NEs from normal words, and all words (including NEs) that occur in questions and KB are part of the vocabulary. The *With-NE-Table* uses our proposed method and builds *NE-table* (course numbers and course names are the NEs in this task). In the *With-NE-Table*, when a NE word is encountered, the hidden state of the RNN at the previous time step (word) is used as input to the *NE-Embedding Generation Module* (f_ϕ). The NE-Embedding generated by f_ϕ is then fed back to the RNN to continue encoding the question. The generated NE-Embedding is also stored in the *NE-Table* associated with this question. The final hidden state of the RNN obtained after encoding the full question is provided as input to the *KB-Retrieval Module* (h_ψ).

For our example (**`Q:`** `Course Number EECS545 Credits ?`), both methods perform attention over the column headings to identify the correct column *Credits* required for the answer. Then, both methods attend over column headings to find the column *Course Number* used for representing the rows. For *W/O-NE-Table*, since all course numbers are part of vocabulary, each row is represented by neural embeddings associated with course numbers and attention is done over the row embeddings. For *With-NE-Table*, since course numbers are NEs, each row is represented with exact course number values. A neural attention over *NE-Table* is performed to return the NE value, *EECS545*, which is then used to perform an exact match with the course number values.

Both the *With-NE-Table* and *W/O-NE-Table* use the following hyperparameters : embedding_size = 20, batch_size = 16. The RNN used for encoding the sentences has hidden_units = 20. They are trained using Adam (learning rate = 0.01, epsilon = 1e-8). Supervision is provided for KB-Retrieval attentions and standard cross-entropy loss is used.

Method	Test Accuracy (%)
W/O-NE-Table	81.0
With-NE-Table	100.0

Table 3.2: Test results on structured-QA task

Table 3.2 shows the retrieval accuracy for both methods. While the test accuracy for *With-NE-Table* is 100%, it drops to 81% for *W/O-NE-Table*. Further analysis shows that out of the 19% drop, 11% is due to OOV NEs encountered at test time. These OOV NEs are in the KB, and hence are part of the vocabulary for the *W/O-NE-Table* method, but have random embeddings which did not change during the training (as they were never encountered during the training). The rest 8% drop can be attributed to the method's inability to learn good embeddings for unique NEs that were rarely seen during training. However, these issues do not pose a problem for our *With-NE-Table* method, since we generate embedding for a NE on the fly for each question based on the context. This solves both the problems: 1) whether an NE occurred rarely or 2) it was not present in training data at all. The *With-NE-Table* method should also easily scale to large datasets with any number of NEs without drop in performance.

3.3.4 Goal-Oriented Dialog Tasks

The bAbI dialog tasks (referred to as original-bAbI dialog tasks from here on) dataset was introduced by Bordes, Boureau, and Weston (2017) (described in Section 2.2) as a testbed to break down the strengths and shortcomings of end-to-end goal-oriented dialog systems.

In the original-bAbI dialog tasks, KB-Retrieval is bypassed by providing all possible system utterances with all combinations of information *pre*-retrieved from the KB in a large candidate response list. We extend the original testbed and propose a new testbed: KB-Augmented-bAbI dialog tasks, by adding an actual external KB so that the system can also be tested on the ability to retrieve the required information from the KB. We evaluate our method on the KB-augmented versions of bAbI dialog task 1,2 and 4. The KB-augmented-bAbI dialog tasks dataset is available at - `https://github.com/IBM/ne-table-datasets/`.

For our experiments, we use an end-to-end memory network similar to Bordes, Boureau, and Weston (2017) (described in Section 2.3), except that we encode sentences using an RNN, while they use BoW encoding. The encoded sentences, which are part of the dialog history, are stored in the memory and the query (last user utterance) embedding is used to attend over the memory to get relevant information from the memory. The generated internal state is used to select the candidate response, and is also given as input to the *KB-Retrieval Module* (h_ψ). The KB is used to identify the NEs along with their types (if a word is present in a NE-column in the KB it is a NE; the column where it appears gives its NE-type).This simple method (based on exact match) though works for this dataset, may not necessarily be effective for other datasets, as plural or abbreviated NEs will not match.

The experiments are performed on two methods: *W/O-NE-Table* method (the baseline method) - All input words including NEs are part of the vocabulary. For NEs, however, their embedding given to the sentence encoder RNN is the sum of the NE word embedding and the embedding associated with its NE-type. *With-NE-Table* method (uses our proposed method) - When an NE is encountered in the dialog, the last hidden state of the RNN encoding the sentence is used as input to the *NE-Embedding Generation Module* (f_ϕ). The NE-Embedding generated is stored in the *NE-Table*. The generated NE-Embedding and the embedding associated with its NE-type are fed to the RNN.

Note that both the methods have access to the information of whether a given word is a NE and its NE-type. Supervision is provided for candidate response selection and all attention operations performed during *KB-Retrieval*, for both methods. Both the *With-NE-Table* and *W/O-NE-Table* methods use the following hyperparameters : hops = 3, embedding_size = 40, batch_size = 16. The RNN used for encoding the sentences has hidden_units = 40. Both methods are trained using Adam (learning rate = 0.001, epsilon = 1e-8) with standard cross-entropy loss.

3.3.4.1 KB-Augmented-bAbI Dialog Tasks 1 and 2

In the original-bAbI dialog task 1, the conversation between the system and the user involves getting information necessary to issue an *api_call*. In task 2, the user can ask the system to update his/her preferences (cuisine, location etc.). The system has to take this into account and make an updated *api_call*. In our KB-augmented version, once the system determines that the next utterance is an *api_call*, the system also has to actually retrieve the restaurant details from the KB (rows) which match user preferences. The system is evaluated on having conversation with the user, issuing *api_call* and retrieving the correct information from the KB. The KB is represented as a single table, where each row corresponds to a unique restaurant and columns correspond to attributes, e.g., cuisine, location etc. Examples of KB-augmented-dialog bAbI tasks 1 and 2 are provided below.

Original-bAbI dialog task 1:
U: Hi !
`S: Hello, what can I help you with today?`
U: I’d like to book a table for six people in an expensive price range with British food.
`S: I’m on it.`
`S: Where should it be?`
U: In London.
`S: Ok, let me look into some options for you.`
`S: api_call British, London, Six, Expensive`

KB-augmented-bAbI dialog task 1:
U: Hi !
`S: Hello, what can I help you with today?`
U: I’d like to book a table for six people in an expensive price range with British food.
`S: I’m on it.`
`S: Where should it be?`

U: In London.
S: Ok, let me look into some options for you.
S: api_call
S (Retrieved Information):
{The_Place, british, london, ...}
{The_Fancy_Pub, british, london, ...}

Original-bAbI dialog task 2:
U: Hi !
S: Hello, what can I help you with today?
U: I'd like to book a table for six people in an expensive price range with British food in London.
S: Ok, let me look into some options for you.
S: api_call British, London, Six, Expensive
U: Instead, could it be Indian cuisine.
S: Sure, is there anything else to update?
U: No.
S: Ok, let me look into some options for you.
S: api_call Indian, London, Six, Expensive

KB-augmented-bAbI dialog task 2:
U: Hi !
S: Hello, what can I help you with today?
U: I'd like to book a table for six people in an expensive price range with British food in London.
S: Ok, let me look into some options for you.
S: api_call
S (Retrieved Information):
{The_Place, british, london,...}
The_Fancy_Pub, british, london, ...}
U: Instead, could it be Indian cuisine.
S: Sure, is there anything else to update?
U: No.
S: Ok, let me look into some options for you.
S: api_call
S (Retrieved Information):

`{Suvai, Indian, london, ...}`
`{Madras_Masala, Indian, london, ...}`

Both *W/O-NE-Table* and *With-NE-Table* methods, first select the four relevant (cuisine, location, price range and number of people) columns to represent each row (restaurant). The *W/O-NE-Table*, then selects the rows using attention over the row embeddings obtained through the combined (additive) representation of the four selected attributes. The *With-NE-Table*, splits the row selection into two simpler problems. For cuisine and location (which are NEs), one NE value each is retrieved from the *NE-Table* and an exact match is performed in the KB. The neural embeddings of the non-NE attributes (price range and number of people) are added to perform attention for selecting rows. The final retrieved rows are the intersection of the rows selected by NE column and non-NE column based selections.

Task	Method	KB-Retrieval	Per-Turn	Per-Dialog	Per-Dialog + KB-Retrieval
1	W/O-NE-Table	10.2 (7)	100 (98.2)	100 (90.3)	10.2 (6.7)
	With-NE-Table	98.5 (99.0)	99.8 (99.8)	98.8 (99)	97.3 (98.0)
2	W/O-NE-Table	0.8 (1.0)	100 (100)	100 (100)	0.0 (0.1)
	With-NE-Table	99.6 (99.8)	100 (100)	100 (100)	99.2 (99.7)

Table 3.3: Test results for KB-augmented-bAbI dialog tasks 1 and 2. Accuracy % for Test and Test-OOV (given in parenthesis). KB-Retrieval %: Retrieval accuracy for rows. Per-Dialog %: Percentage of dialogs where every dialog response is correct. Per-Dialog + KB-Retrieval %: Percentage of dialogs where every dialog response and information from KB retrieval are correct.

The results for tasks 1 and 2 are shown in Table 3.3. The *With-NE-Table* method achieves close to 100% accuracy on both tasks, while *W/O-NE-Table* performs poorly. During KB retrieval, for *With-NE-Table*, two NEs are chosen from the *NE-Table* and exact matching is done over different cuisines and locations in the KB, but embeddings for these NEs are learned for *W/O-NE-Table*. This results in poor KB-Retrieval for *W/O-NE-Table* for less frequent and OOV location/cuisine values. Both methods perform well in *Per-dialog* accuracy as it does not involve KB retrieval. The system responses do not contain any NEs, but the system still needs to understand user utterances which might have NEs. The Per-Dialog accuracy is high for both methods on the normal test set. However, for task 1 OOV-test set, *W/O-NE-Table* method is affected by OOV-NEs (90.3%), while *With-NE-Table*'s performance is robust (99.0%).

Detailed results for tasks 1 and 2.

The detailed results for task 1 and task 2 are shown in Tables 3.4.

Task	Method	ACR	ARR Non-NE	ARR NE
1	W/O-NE-Table	100 (100)	9.0 (6.9)	-
	With-NE-Table	99.4 (98.1)	96.9 (96.7)	100,100 (100,100)
2	W/O-NE-Table	100 (100)	8.6 (7.6)	-
	With-NE-Table	100 (100)	99.1 (99.8)	100,100 (100,100)

Table 3.4: Test results for KB-augmented-bAbI dialog tasks 1 and 2. Accuracy % for Test and Test-OOV (given in parenthesis). ARR non-NE columns are price and number of people. ARR NE columns are cuisine and location.

With-NE-Table: For issuing an *api_call* in tasks 1 and 2, four argument values are required - cuisine, location, price range and number of people. We consider cuisine and location to be NEs. So whenever cuisine and location names occur in the dialog, a NE key is generated on the fly and is stored in the *NE-Table* along with the NE values.

- ACC: For tasks 1 and 2, ACC is not required as we are interested in retrieving rows (not cells of the table).
- ACR: ACR is used to select the columns required to represent the rows. These are four columns - NE columns (cuisine and location) and non-NE columns (price range and number of people)
- ARR-non-NE: Each row in the KB is represented by weighted vector (embedding) sum of its price range and number of people (embeddings). The relevant rows are selected using attention on the non-NE columns embeddings.
- ARR-NE: The method attends over the *NE-Table* by matching (dot product) its generated key with the keys present in the *NE-Table* to retrieve NE values. The selected NE values are then matched (exact-match) with cuisine and location values in KB to retrieve the relevant rows.
- The final retrieved rows are the intersection of the rows selected by ARR-non-NE and ARR-NE.

W/O-NE-Table: *ACR* is used to attend to the four relevant columns. However, each row is represented by the combined neural embedding representation of all four attribute values, cuisine, location, price range and number of people. *ARR non-NE* is used to retrieve the relevant rows.

From Tables 3.4 and 3.3, we can see that both the methods perform well in selecting the relevant columns, but *W/O-NE-Table* performs poorly in retrieving the rows, while *With-NE-Table* performs very well. This results in *With-NE-Table* method achieving close to 100% accuracy in KB retrieval while *W/O-NE-Table* performs poorly.

This is because, in *With-NE-Table*, the task of retrieving rows is split into two simpler tasks. The NEs are chosen from the *NE-Table*, and then exact matching is used (which helps in handling OOV-NEs as well). The non-NEs, price range and number of people, have limited set of possible values (low, moderate or expensive for price range and 2,4,6 or 8 for number of people respectively). This allows the system to learn good neural embeddings for them and hence have high accuracy in *ARR non-NE*. Whereas in *W/O-NE-Table*, *ARR non-NE* involves the neural representations of cuisine and location values as well, where a particular location and cuisine value will occur only a few number of times in the training dataset. In addition to that, new cuisine and location values can occur during the test time (Test OOV dataset, performance shown in parenthesis).

For the dialog part (which does not involve the KB retrieval aspect) of the KB-augmented tasks 1 and 2, the system utterances do not have any NEs in them. However, the user utterances contain NEs (cuisine and location that the user is interested in) and so the system has to understand them in order to select the right next system utterance. The accuracy in performing the dialog (by selecting responses from candidate set) is similar for both the methods on the normal test set. However, in the OOV-test set, for task 1, where the system has to maintain the dialog state to track which attribute values have not been provided by the user yet, *W/O-NE-Table* seems to get affected, while *With-NE-Table* is robust to that. While *W/O-NE-Table* gets a Per-Dialog accuracy of 90.3% in the OOV-test set, *With-NE-Table* is able to get 99%.

3.3.4.2 KB-augmented-bAbI Dialog Task 4

The original-bAbI dialog task 4 starts at the point where a user has decided on a particular restaurant. The system is given information (location, phone number, address etc.) about *only* that restaurant as part of the dialog history and the user can ask for its phone number, address or both. For a given user request e.g., address, the task is to select the correct response with the restaurant's address from a list of candidate responses. These candidate responses have phone number and address information for all the restaurants mentioned in the KB.

In our KB-augmented version of the task, even though the user has decided a particular restaurant, its corresponding information is not provided as part of dialog history. This makes the task harder but more realistic. Now, the system needs to search for phone number/address for the restaurant from the full KB while in the original task, the phone number/address is already provided as part of dialog history. In the KB-augmented version, the NEs in candidate responses are replaced with their NE-type tags. For example, *Suvai_phone* is replaced with *NE_phone*. The system has to select the candidate with correct NE-type tag and then replace the tag with the actual NE-value retrieved from the KB, similar to Williams, Asadi, and Zweig (2017). This setting is closer to how a human agent would do this task. An example dialog from the KB-augmented-bAbI dialog task 4 and it's comparison with the original-bAbI dialog task 4 is shown below.

Original-bAbI dialog task 4:
KB Information given :
NAME:The_Place, CUISINE:British,
LOCATION:London, PRICE RANGE:Expensive,
RATING:7, PHONE:The_Place_phone,
ADDRESS:The_Place_address, AVAILABILITY:six
U: Hello
`S: Hello, what can I help you with today?`
U: I'd like to book a table at The_Place.
`S: Great, let me do the reservation.`
U: May I have the address of the restaurant.
`S: Here it is, The_Place_address.`

KB-augmented-bAbI dialog task 4:
KB Information given : Entire KB
U: Hello
`S: Hello, what can I help you with today?`
U: I'd like to book a table at The_Place.
`S: Great, let me do the reservation.`
U: May I have the address of the restaurant.
`S: Here it is, NE_address.`
`S: The_Place_address (retrieved information)`

For *With-NE-Table*, the restaurant name that appears in the dialog would be stored in *NE-Table*. When the user asks for information such as phone number, the restaurant name stored in *NE-Table* is selected and used for retrieving its phone number from the KB. In

W/O-NE-Table, all input words (including NEs) are part of vocabulary and phone number is selected by neural embedding attention over all restaurants names.

Method	KB-Retrieval	Per-Turn	Per-Dialog	Per-Dialog + KB-Retrieval
W/O-NE-Table	0.0 (0.0)	100 (100)	100 (100)	0.0 (0.0)
With-NE-Table	100 (100)	100 (100)	100 (100)	100 (100)

Table 3.5: Test results for KB-augmented-bAbI dialog task 4. Accuracies in % for Test and Test Out-Of-Vocabulary (given in parenthesis). KB-Retrieval %: Retrieval accuracy for a particular cell. Per-Dialog %: Percentage of dialogs where every dialog response is correct. Per-Dialog + KB-Retrieval %: Percentage of dialogs where every dialog response and information from KB retrieval are correct.

The results for task 4 are shown in Table 3.5. We observe that both methods perform well in Per-dialog accuracy. The *W/O-NE-Table* method fails in KB-retrieval (0%) because it needs to learn neural embeddings for all restaurant names, while *With-NE-Table* performs well (100%) as it uses our proposed method to generate NE-Embeddings on the fly and use the actual NE values later for exact value matching over restaurant names in the KB.

Detailed results for task 4.

Detailed results for task 4 are shown in Tables 3.6.

Method	ACR	ACC	ARR Non-NE	ARR NE
W/O-NE-Table	100 (100)	100 (100)	0.0 (0.0)	-
With-NE-Table	100 (100)	100 (100)	-	100 (100)

Table 3.6: Test results for KB-augmented-bAbI dialog task 4. Accuracies in % for Test and Test Out-Of-Vocabulary (given in parenthesis).

With-NE-Table: In task 4, the user tells the system, the restaurant in which he/she wants to book a table. The restaurant name, which is a NE, is stored in the *NE-Table* along with it's generated key. When the user asks for information about the restaurant such as, phone number, the NE restaurant name stored in the *NE-Table* is selected and used for retrieving its corresponding phone number from the KB. For this particular case, *ACC* attends over the column *Phone* and *ACR* attends over *Restaurant Name*. Since the column selected by *ACR* is a NE column, the NE value (here the actual restaurant name given by the user) is retrieved using *ARR NE* from the *NE-Table*. The retrieved NE value is used to do an exact match over the KB column selected by *ACR* to select the rows. The cell that intersects the selected row and the column selected by *ACC* is returned as the retrieved information and used to replace the NE-type tag in the output response.

W/O-NE-Table: Here, all input words (including NEs) are part of the vocabulary and for NEs, their embedding given to the sentence encoder is the sum of the NE word embedding and the embedding associated with its NE-type. The candidate response retrieval (dialog) is same as the above method and the column attentions are also similar. However, the methods differ with respect to attention over rows. Since NEs are not treated special here, attention over rows happens through *ARR non-NE*. For this task, when *ACR* is selected correctly (restaurant name), each row will be represented by the neural embedding representation of its restaurant names. *ARR non-NE* generates a key to match these neural embeddings to attend to the row corresponding to the restaurant name mentioned by the user.

3.3.4.3 Comparison With Original-bAbI Dialog Tasks

We choose the best method (MemN2N + match-type features) from (Bordes, Boureau, and Weston 2017) (they use match-type features for dealing with entities) and update the baseline method by using RNN encoding for sentences (similar to *With-NE-Table*). Note that we achieve higher accuracy for our updated baseline method for original-bAbI tasks than reported in (Bordes, Boureau, and Weston 2017), which we attribute to the use of RNN for encoding sentences (they use BoW encoding).

For match-type features, (Bordes, Boureau, and Weston 2017) add special words (*R_CUISINE*, *R_PHONE* etc.), for each KB entity type (cuisine, phone, etc.) to the vocabulary. The special word (e.g., *R_CUISINE*) is added to a candidate if a cuisine (e.g., Italian) appears in both dialog and the candidate. For each type, the corresponding type word is added to the candidate representation if a word is found that appears 1) as a KB entity of that type, 2) in the candidate, and 3) in the input or memory. For example, for a task 4 dialog with restaurant information about *RES1*, *only* one candidate *"here it is RES1_phone"* will be modified to *"here it is RES1_phone R_PHONE"*. Now, if the user query is for the restaurant's phone number, using match-type features essentially reduces the output search space for the dialog model and allows it to attend to specific candidates better. Hence, match-type features can only work in a retrieval setting and will not work in a generative setting. Our *With-NE-Table* method will work in both retrieval and generative settings.

Table 3.7 compares the performance of the *With-NE-Table* method in the KB-augmented-bAbI dialog tasks with that of a baseline method on the original-bAbI tasks. Note that KB-augmented-bAbI dialog tasks require the dialog system to do strictly more work compared to the original-bAbI dialog tasks. Though not a strictly fair comparison for our method, we observe that the performance of our *With-NE-Table* method in KB-augmented-bAbI dialog tasks is as good as the performance of updated baseline method

Task	Method	Evaluation	Task 1	Task 2	Task 4
Original	Baseline	Per-Dialog	100 (100)	99.9 (50.6)	100 (100)
KB-Augmented	With-NE-Table	Per-Dialog + KB-Retrieval	97.3 (98.0)	99.2 (99.7)	100 (100)

Table 3.7: Performance comparison of our method in the KB-Augmented-bAbI dialog tasks, with a baseline method in the original-bAbI tasks. Accuracies in % for Test and Test Out-Of-Vocabulary (given in parenthesis).

in original-bAbI tasks. In addition to that, for bAbI task 2 OOV test set, *With-NE-Table*'s performance in the KB-augmented-bAbI dialog task, is actually much higher compared to the baseline method on the original-bAbI task (99.7% vs 50.6%).

3.4 Related Work

NE in QA. Neelakantan, Le, and Sutskever (2015) and Yin, Lu, et al. (2016) transform a natural language question/query to a program that could run on Knowledge Bases (KB), but those approaches are only verified on small or synthetic KBs. Other works dealing with large Knowledge Bases (KB) usually rely on entity linking techniques (Cucerzan 2007; Guo, Chang, and Kiciman 2013), which links entity mentions in texts to KB queries. Yih et al. (2015), Yin, Yu, et al. (2016), and Yu et al. (2017) compare the text spans in questions with KB entity names at the character-level for entity linking; after the linked entities have their properties extracted, the corresponding text spans are replaced with special NE tags for further text processing like KB relation extraction. Recently, Liang et al. (2017) extended end-to-end neural methods to Question Answering (QA) over KB, which could work for large KB and large number of NEs. However, their method still relies on entity linking to generate a short list of entities linked with text spans in the questions, in advance. Yin, Lu, et al. (2016) propose 'Neural Enquirer', a neural network architecture similar to the neural retrieval mechanism used in this chapter, to execute natural language queries on KB. They keep the randomly initialized embeddings of the NEs fixed as a method to handle NEs and OOV words.

NE in Dialog. There has been a lot of interest in end-to-end training of dialog systems (Vinyals and Le 2015; Serban, Sordoni, et al. 2016; Lowe et al. 2015; Kadlec, Schmid, and Kleindienst 2015; Shang, Lu, and Li 2015; Guo, Klinger, et al. 2017). Among recent work, Williams and Zweig (2016) use an LSTM model that learns to interact with APIs on behalf of the user; Dhingra et al. (2017) use reinforcement learning to build the KB look-up in task-oriented dialog systems. But the look-up actions are defined over each entity in the KB and is therefore hard to scale up. Most of these works actually do not discuss the issue of handling NEs though they are present. Williams, Asadi, and Zweig (2017) propose Hybrid Code Networks and achieve state-of-the-art on the bAbI dataset, but their approach involves a developer writing domain-specific software components.

NE in Reading Comprehension and Others. For certain tasks such as Machine Translation and summarization, neural copying mechanisms (Gulcehre et al. 2016; Gu et al. 2016) have been proposed as a means for handling OOV words. Our *NE-Table* method can be used along with such copying mechanisms for things like dialog generation.

3.5 Conclusion

We proposed a novel method for handling NEs in neural settings for goal-oriented dialog tasks. It is also applicable to other NLP tasks with NEs. Our experiments on the CBT dataset illustrate that the method with *NE-Table* perform better than methods without *NE-Table*, and clearly outperform the baseline methods on the OOV test sets. We observe similar results for our experiments on the structured-QA task and goal-oriented bAbI dialog tasks. We also show that our method can be used for NEs in the external KB provided. Overall, these experiments show that the proposed method can be useful for various NLP tasks where it is beneficial to work with actual NE values, and/or it is hard to learn good neural embeddings for NEs. An interesting future direction is to explore the use of pre-trained embeddings: word2vec (Mikolov et al. 2013b), ELMo (Peters et al. 2018) etc., to bootstrap the learned NE-embeddings in the NE-Table.

CHAPTER 4

Learning With Multiple Answers

In a dialog, there could be multiple valid next utterances at any point. The present end-to-end neural methods for dialog do not take this into account. They learn with the assumption that at any time there is only one correct next utterance. In this chapter, we focus on this problem in the goal-oriented dialog setting where there are different paths to reach a goal. We propose a new method, that uses a combination of supervised learning and reinforcement learning approaches to address this issue. We also propose a new and more effective testbed, permuted-bAbI dialog tasks by introducing multiple valid next utterances to the original-bAbI dialog tasks. We show that there is a significant drop in performance of existing end-to-end neural methods from 81.5% per-dialog accuracy on original-bAbI dialog tasks to 30.3% on permuted-bAbI dialog tasks. We also show that our proposed method improves the performance and achieves 47.3% per-dialog accuracy on permuted-bAbI dialog tasks.

Published at: Empirical Methods in Natural Language Processing (EMNLP 2018).
Authors: Janarthanan Rajendran*, Jatin Ganhotra*, Satinder Singh, and Lazaros C. Polymenakos.
Collaborations: IBM Research.

---

*Equal contribution

4.1 Introduction

The task of having a dialog with a human in natural language has some unique properties that many other machine learning tasks do not. For any given dialog state or input, multiple correct next utterances or answers may be possible; i.e., given the dialog so far, there are several different utterances which one can say next that would be valid. However, this property of dialog is not taken into account in the present way of end-to-end training of neural dialog systems from dialog data.

Present day neural dialog systems are trained using (broadly) either Supervised Learning (SL) or Reinforcement Learning (RL) or a mix of both (refer Section 2.1 for more discussion). In the RL setting, the dialog system learns through trial and error with reinforcement (rewards at the end or at key dialog points) from a human or a simulator. In the SL setting, a fixed set of dialog data is collected from human-human chatlogs and the dialog system is trained to imitate that data using some form of mini-batch gradient descent. It is not possible to know all of the valid next utterances for a given dialog state at any single time. A particular dialog in the dataset has access to only one of the valid next utterances given the dialog history and the current utterance. Another valid next utterance could be present in some other dialog in the dataset.

Since for a given dialog only one correct answer is available, the gradients (from a mini-batch of the dataset) are calculated based on the assumption that there is only one correct next utterance for the given dialog state. This results in reducing the probability of other valid next utterances for that dialog. While all this is true for dialog in general, in this chapter, we focus on the goal-oriented dialog setting.

We propose a novel method which handles the issue of learning multiple possibilities for completing a goal-oriented dialog task. Our presentation is organized as follows: In Sections 4.2 and 4.3, we define the multiple-utterance problem and point out the limitations of current learning methodologies.

The rest of the chapter is organized as follows. Section 4.4 describes our proposed method, which combines Supervised Learning and Reinforcement Learning approaches for handling multiple correct next utterances. In Section 4.5, we introduce permuted-bAbI dialog tasks, which is our proposed testbed. Section 4.6 details our experimental results across all datasets and all methods.

4.2 Multiple-Utterance Problem in Goal-Oriented Dialog

Goal-oriented dialog tasks, where there is an explicit goal that the system tries to achieve through the dialog, typically involve getting some information from the user, interacting with an external Knowledge Base (KB) and giving back information to the user. Simple examples (form filling) include restaurant reservation, hotel booking etc., whereas complex tasks could involve a combination of informative and form filling tasks (e.g., IT support, customer care etc.).

There could be multiple ways/strategies to achieve a given task. When a dataset is collected from different people performing the task, these different ways of solving the task get reflected in it. These variations could be as simple as difference in the order in which the system asks the information from the user, or as complex as following a completely different line of questions/answers to achieve the task. For example, in an IT support scenario, one may ask a sequence of standard questions or start from common problems and once eliminated, follow the standard set of questions. In the dataset, they turn into multiple valid next utterances for a given dialog so far.

Our objective is not to mimic all humans from whom the data was collected from or all the possible strategies, but rather to use that knowledge and learn to perform the task better and faster.

4.3 Issues With the Present Methods

Consider a goal-oriented dialog dataset for restaurant reservation where the dialog system has to acquire cuisine, location, number of people and price range information from the user before retrieving restaurant options. Consider two dialogs (A and B) in the dataset which have the same first system utterance (`S1a` is same as `S1b`). Let their dialog state vector after encoding the dialog until `S1` be s. This state vector s is what will be used for next utterance generation or retrieval. Their next utterance is different because of the variation in the order in which the information is asked from the user as shown below, i.e., `S2a` is different from `S2b`.

Dialog A
U: Hello !
`S1a:  Hello, what can I help you with today?`
U: Can you book a table for six people with French food?
`S2a:  Where should it be?`

U: In Bombay

`S3a:  Which price range are you looking for?`

U: I am looking for a cheap restaurant.

`S4a:  Ok, let me look into some options for you.`

Dialog B

U: Hello !

`S1b:  Hello, what can I help you with today?`

U: Can you book a table for six people with French food?

`S2b:  Which price range are you looking for?`

U: I am looking for a cheap restaurant.

`S3b:  Where should it be?`

U: In Bombay

`S4b:  Ok, let me look into some options for you.`

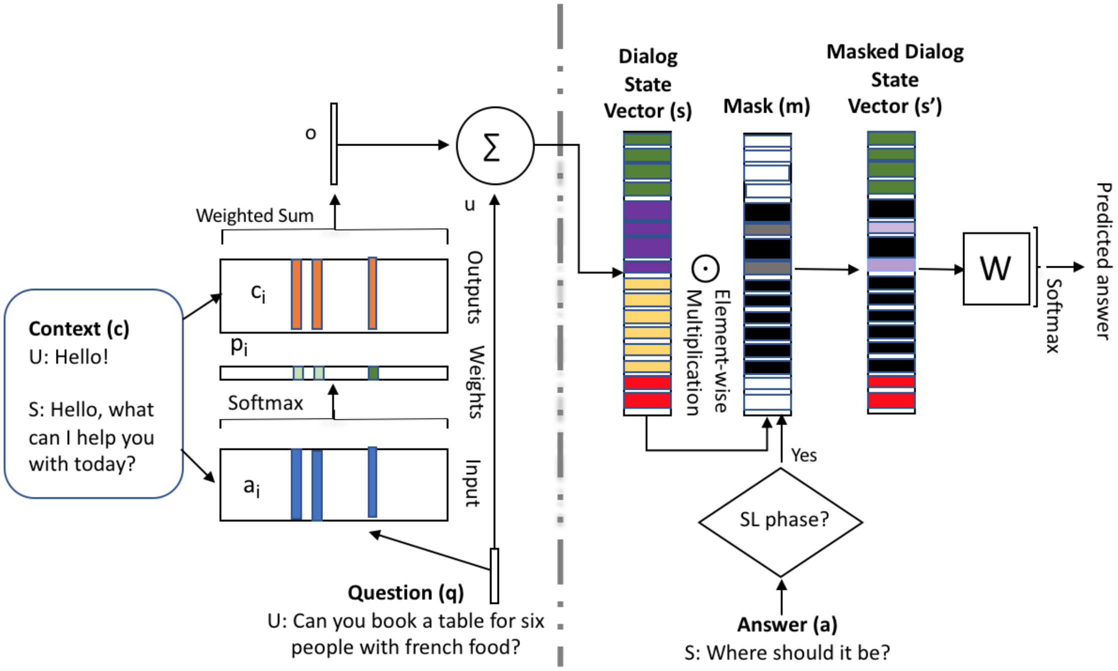


Figure 4.1: Mask-memN2N - *Left*: A single layer version of memN2N. *Right*: Masking

These two dialogs might be present in different places in the dataset. When dialog A is part of the mini-batch for which loss is calculated and parameters are updated, the dialog system is asked to produce `S2a` from s. Here, the loss could be negative log-likelihood, squared error or anything that tries to push s towards producing `S2a`. In this process, the probability of the dialog system producing `S2b`, an equally valid answer, is reduced. The reverse happens when the dialog system encounters a mini-batch with dialog B. This is true whether a softmax or a sigmoid non-linearity for each unit is used in the output layer. In essence, the system is expected to learn a one to many function, but is forced to produce only one of the valid outputs at one time (that is all we have at any one time), while telling that all other outputs are wrong.

Note that this could be a problem even when the dialogs are similar (semantically) in the beginning, but not the same exact dialog. For simplicity, we show an example where two dialogs have the same beginning and only 2 valid next utterances occur.

4.4 Proposed Method

The proposed method has two phases. In phase one, the dialog system tries to learn how to perform dialog from the dataset by trying to mimic it and in phase two it learns through trial and error. The former uses supervised learning and the latter uses reinforcement learning. Consider a dialog state vector s. This has all the information from the dialog so far and is used for next utterance generation or retrieval. Any neural method such as memory network (Weston, Chopra, and Bordes 2014), HRED (Sordoni et al. 2015) etc. can be used for encoding and producing the dialog state vector s. As discussed earlier for the state vector s, there could be multiple valid next utterances.

During the SL phase, at each data point, the dialog system is trained to produce the one next utterance provided in that data point and is penalized even if it produces one of the other valid next utterances. We avoid this by providing the dialog system, the ability to use only parts of the state vector to produce that particular next utterance. This allows only parts of the network to be affected that were responsible for the prediction of that particular answer. The dialog system can retain other parts of the state vector and values in the network that stored information about other valid next utterances. This is achieved by generating a mask vector m which decides which parts of the state vector s should be used for producing that particular answer. This is achievable, as m is learned as function of s and the actual answer a present in the given dialog data point.

In the RL phase, however, the dialog system is rewarded if it produces an answer that is among any of the set of valid correct answers. While in the SL phase the dialog system had access to the actual answer a at given time to produce the mask, in the RL phase the dialog system produces the mask by only using the dialog state vector s.

$$\begin{aligned}
&\text{Supervised Learning phase:} && (4.1)\\
&m = \sigma(W_s s + W_a a + b_{sl}),\\
&s' = m * s,\\
&\text{Reinforcement Learning phase:} && (4.2)\\
&m = \sigma(W_s s + W_r s + b_{rl}),\\
&s' = m * s,
\end{aligned}$$

where W's and b's are the parameters learned, σ is an element wise sigmoid non-linearity and s' is the masked dialog state vector that is used by the dialog system to perform the downstream task such as next utterance generation or retrieval. The parameters of the network that produce s and that follow s' are shared between the two phases. While there are different ways of combining the two phases during training, the RL phase which does not use the answer for its mask is what is used during testing. The masking approach described above is illustrated in Fig 4.1.

In our experiments, SL phase is performed first, followed by RL phase. In the SL phase, the dialog system learns different dialog responses and behaviours from the dataset. It has the ability to learn multiple possible next utterances without one contradicting/hindering the learning of the other much. In the RL phase, the dialog system might settle on a unique behaviour that it finds best for it to perform the task and uses that during test time as well.

4.5 Permuted bAbI Dialog Tasks

Bordes, Boureau, and Weston (2017) proposed bAbI dialog tasks, a testbed to break down the strengths and shortcomings of end-to-end dialog systems in goal-oriented applications (described in detail in Section 2.2). Bordes, Boureau, and Weston (2017) used natural language patterns to create user and system utterances. There are 43 patterns for the user and 20 for the system, which were combined with the KB entities to form thousands of different utterances. However, on a closer analysis of the testbed, we observe that even though there are thousands of different utterances, these utterances always follow a fixed deterministic order (predefined by the simulation). For example, for Task 1, the system follows a predefined order to ask for missing fields required to complete the dialog state. In Task 3, all restaurants retrieved have a unique and different rating. While this makes evaluation deterministic and easier, these hidden settings in the simulation create conversations,

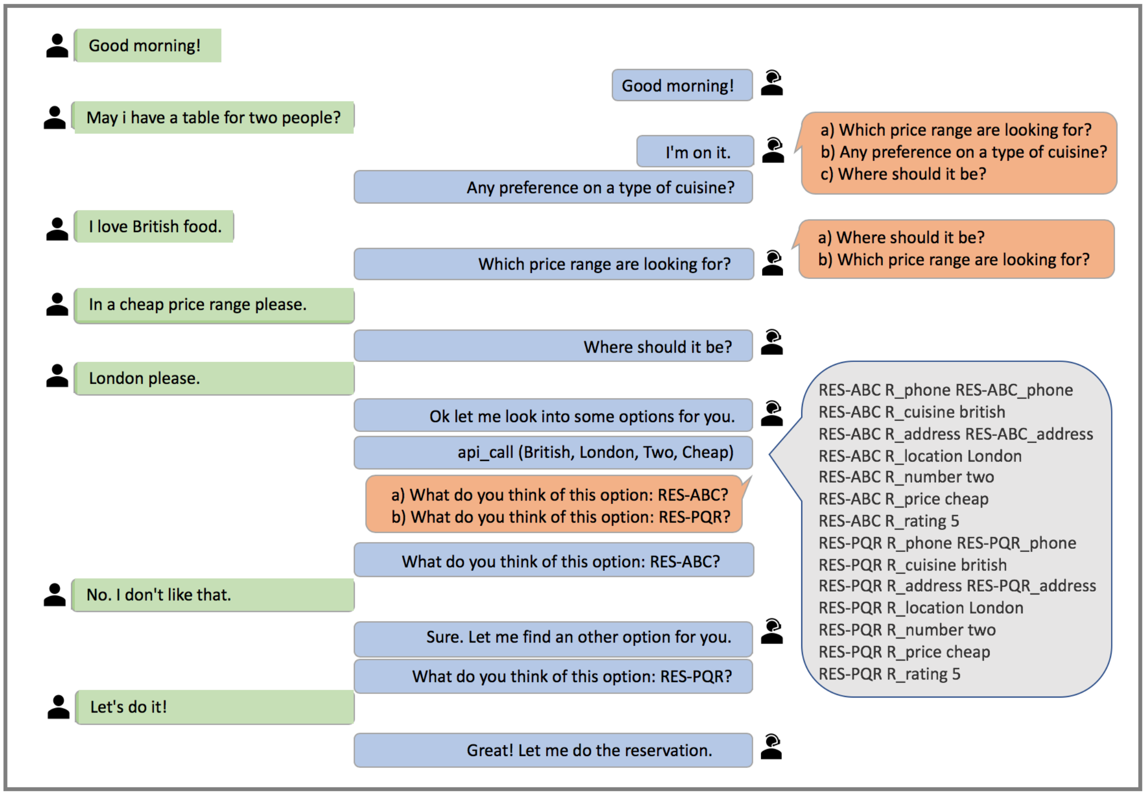

Good morning!
Good morning!
May i have a table for two people?
I'm on it.
a) Which price range are looking for?
b) Any preference on a type of cuisine?
c) Where should it be?
Any preference on a type of cuisine?
I love British food.
a) Where should it be?
b) Which price range are looking for?
Which price range are looking for?
In a cheap price range please.
Where should it be?
London please.
Ok let me look into some options for you.
api_call (British, London, Two, Cheap)
a) What do you think of this option: RES-ABC?
b) What do you think of this option: RES-PQR?
RES-ABC R_phone RES-ABC_phone
RES-ABC R_cuisine british
RES-ABC R_address RES-ABC_address
RES-ABC R_location London
RES-ABC R_number two
RES-ABC R_price cheap
RES-ABC R_rating 5
RES-PQR R_phone RES-PQR_phone
RES-PQR R_cuisine british
RES-PQR R_address RES-PQR_address
RES-PQR R_location London
RES-PQR R_number two
RES-PQR R_price cheap
RES-PQR R_rating 5
What do you think of this option: RES-ABC?
No. I don't like that.
Sure. Let me find an other option for you.
What do you think of this option: RES-PQR?
Let's do it!
Great! Let me do the reservation.

Figure 4.2: Permuted-bAbI dialog tasks. A user (in green) chats with a dialog system (in blue) to book a table at a restaurant. At a given point in the dialog, the dialog system has multiple correct next utterances (in orange). The dialog system can choose either of the multiple correct utterances as the next utterance. The list of restaurants are returned from the API_call (in grey) also contain multiple restaurants with the same rating, giving the dialog system more options to propose to the user.

which are simpler compared to real-world conversations for restaurant reservation without the issue of multiple next utterance in the dataset.

We propose permuted-bAbI dialog tasks (available at `https://github.com/IBM/permuted-bAbI-dialog-tasks`), an extension of original-bAbI dialog tasks, which make our proposed testbed more appropriate for evaluating dialog systems in goal-oriented setting. In *original-bAbI dialog tasks* at a given time in the conversation, there is **only one** correct system utterance. *Permuted-bAbI dialog tasks* allow **multiple** correct system utterances at a given point in the conversation.

We propose the following changes to original-bAbI dialog tasks. In Task 1, a user request defines a query that can contain from 0 to 4 of the required fields to make a reservation. The system asks questions to fill the missing fields and eventually generate the correct corresponding API call. However, the system asks for information in a *deterministic* order

-

Cuisine → *Location* → *People* → *Price*
to complete the missing fields. In permuted-bAbI dialog tasks, we don't follow a deterministic order and allow the system to ask for the missing fields in any order.

In Task 3, for the API call matching the user request, the facts are retrieved from the KB and provided as part of dialog history. The system must propose options to users by listing the restaurant names sorted by their corresponding rating (from higher to lower) until users accept. However, each restaurant has a different rating. In permuted-bAbI dialog tasks, multiple restaurants can have the same rating but the system must still propose the restaurant names following the decreasing order of rating, which allows multiple valid next utterances.

In Task 5, Tasks 1-4 are combined to generate full dialogs. In permuted-bAbI dialog tasks, we incorporate the changes for both Task 1 and Task 3 mentioned above to the final Task 5 (Conducting full dialogs).

Fig 4.2 shows a dialog sample from permuted-bAbI dialog tasks. We release two versions of permuted-bAbI dialog tasks 1) permuted-bAbI dialog task*, which contains all permutations (the full dataset). There are around 11,000 dialogs in each set and the exact number varies for train, validation, test and test-OOV sets; 2) permuted-bAbI dialog task, which contains 1000 dialogs randomly sampled (we used random seed = 599 for sampling) from permuted-bAbI dialog task*. We choose a random 1000 subset from each of train, validation, test and test-OOV sets to match the number of dialogs in original-bAbI dialog task. Another key point to choose a small subset and to not include all permutations in the training set is that it allows to mimic real-world data collection. For a real-world use-case, as the number of required fields and user options increase, the cost for gathering data covering all permutations will increase exponentially, and one can't guarantee that enough training examples for all permutations will be present in the collected dataset. Note that, since there are multiple correct next utterances, we also modify the evaluation criteria so that the system is rewarded if it predicts any of the multiple correct next utterances.

4.6 Experiments and Results

We use end-to-end memory networks (Sukhbaatar et al. 2015), an extension of Memory Networks proposed by Weston, Chopra, and Bordes (2014) as our base dialog model architecture. We perform experiments on the three datasets mentioned. We also perform experiments with match-type features proposed by Bordes, Boureau, and Weston (2017), which allow the dialog model to use type-information for entities like location, cuisine, phone number etc. The results for our baseline method, our proposed method and results

Dataset	No Match-Type		+ Match-Type	
	Per-Turn	Per-Dialog	Per-Turn	Per-Dialog
Original-bAbI	98.5	77.1	98.8	81.5
Permuted-bAbI*	96.4	58.2	96.9	63.9
Permuted-bAbI	91.8	22	93.3	30.3
OOV: Original-bAbI	65.6	0	78.3	0
OOV: Permuted-bAbI*	63.6	0	78.4	0
OOV: Permuted-bAbI	63.4	0.5	78.1	0.6

Table 4.1: Test results for our baseline end-to-end memory network model across the three datasets. Results (accuracy %) are given in the standard setup and out-of-vocabulary (OOV) setup. Results are given for both with and without match-type features.

on our ablation study are described below. The test results reported are calculated by choosing the model with highest validation per-turn accuracy.

4.6.1 Baseline Method: memN2N

We use end-to-end memory networks (memN2N) as our baseline method. A single layer version of the memN2N is shown in Fig.4.1. The detailed description of memN2N is provided in Section 2.3. The hyperparameters used for the baseline method are as follows: hops = 3, embedding_size = 20, batch_size = 32. The entire model is trained using stochastic gradient descent (SGD) (learning rate = 0.01) with annealing (anneal_ratio = 0.5, anneal_period = 25), minimizing a standard cross-entropy loss between $\hat{a}$ and the true label a. We add temporal features to encode information about the speaker for the given utterance (Bordes, Boureau, and Weston 2017) and use position encoding for encoding the position of words in the sentence (Sukhbaatar et al. 2015). We learn two embedding matrices A and C for encoding story, separate embedding matrix B for encoding query and weight matrices TA and TC for encoding temporal features. The same weight matrices are used for 3 hops.

Our results for our baseline method across the three datasets are given in Table 4.1.

The first 3 rows show the results for the three datasets in the standard setup, and rows 4-6 show results in the Out-Of-Vocabulary (OOV) setting. Per-response accuracy counts the percentage of responses that are correct (i.e., the correct candidate is chosen out of all possible candidates). Note that, as mentioned above in Section 4.5, since there are multiple correct next utterances, a response is considered correct if it predicts any of the multiple correct next utterances. Per-dialog accuracy counts the percentage of dialogs where every response is correct. Therefore, even if only one response is incorrect, this might result in a failed dialog, i.e., failure to achieve the goal of restaurant reservation. We report results both with and without match type features, shown in the last two columns.

From Table 4.1, we observe that the baseline method performs poorly on permuted-bAbI dialog tasks (both full dataset and 1000 random dialogs). For permuted-bAbI dialog task*, the baseline method achieves 58.2% on per-dialog accuracy, but the number decreases to only 22% for permuted-bAbI dialog task (1000 random dialogs). This implies that only 1 out of every 4 dialogs might be successful in completing the goal. The results improve slightly by using match-type features. These results clearly demonstrate that the end-to-end memory network dialog model does not perform well on our proposed testbed.

4.6.2 Mask End-to-End Memory Network: Mask-memN2N

Our method, Mask-memN2N, shown in Fig 4.1, is built on the baseline memN2N method described above, except for an additional masking performed to the dialog state vector. The SL phase is performed for the first 150 epochs. The best performing model chosen based on validation accuracy is used as a starting point for the RL phase. All parameters except for the network that produces the masks are shared between the two phases. During the SL phase, the mask parameters of the RL phase are pre-trained to match the mask produced in SL phase using an L2 loss. Through this approach, when the model transitions in the RL phase, it does not need to explore the valid masks and hence, the answers from scratch. Instead, its exploration will now be more biased towards relevant answers. For the RL phase, we use REINFORCE (Williams 1992) for training the system. An additional loss term is added to increase entropy. We use the same hyperparameters as the baseline method mentioned above. The additional hyperparameters are as follows: L2 loss coefficient = 0.1 for pre-training the RL phase mask, entropy with linear decay from 0.00001 to 0, positive reward = 5 for every correct action and negative reward = - 0.5 for an incorrect action.

4.6.3 Model Comparison

Our results for our proposed method and comparison with other methods for permuted-bAbI dialog task are given in Table 4.2. Table 4.2 follows the same format as Table 4.1, except we show results for different methods on permuted-bAbI dialog task. We show results for three methods - memN2N, memN2N + All-Answers and our proposed method, Mask-memN2N.

In the memN2N + All-Answers method, we extend the baseline memN2N method and though not realistic, we provide information on all correct next utterances during training,

Method	No Match-Type		+ Match-Type	
	Per-Turn	Per-Dialog	Per-Turn	Per-Dialog
memN2N	91.8	22	93.3	30.3
memN2N + All-Answers†	88.5	14.9	92.5	26.4
Mask-memN2N	93.4	32	95.2	47.3
OOV: memN2N	63.4	0.5	78.1	0.6
OOV: memN2N + All-Answers†	60.8	0.5	74.9	0.6
OOV: Mask-memN2N	63.0	0.5	80.1	1

Table 4.2: Test results for various methods on permuted-bAbI dialog task. Results (accuracy %) are given in the standard setup and OOV setup; and both with and without match-type features.

instead of providing only one correct next utterance. The memN2N + All-Answers method has an element-wise sigmoid at the output layer instead of a softmax, allowing it to predict multiple correct answers. This model serves as an important additional baseline, and clearly demonstrates the benefit of our proposed approach.

From Table 4.2, we observe that the memN2N + All-Answers method performs poorly, in comparison to the memN2N baseline both in standard setup and OOV setting, as well as with and without match-type features. This shows that the existing methods do not improve the accuracy of a dialog system even if all correct next utterances are known and used during training the dialog model. Our proposed method performs better than both the baseline methods. In the standard setup, the per-dialog accuracy increases from 22% to 32%. Using match-type features, the per-dialog accuracy increases considerably from 30.3% to 47.3%. In the OOV setting, all methods perform poorly and achieve per-dialog accuracy of 0-1% both with and without match-type features. These results are similar to results for original-bAbI dialog Task 5 from Bordes, Boureau, and Weston (2017) and our results with the baseline method.

Overall, Mask-memN2N is able to handle multiple correct next utterances present in permuted-bAbI dialog task better than the baseline methods. This indicates that permuted-bAbI dialog task is a better and effective evaluation proxy compared to original-bAbI dialog task. This also shows that we need better neural approaches, similar to our proposed method, Mask-memN2N, for goal-oriented dialog in addition to better testbeds for benchmarking goal-oriented dialogs systems.

4.6.4 Ablation Study

Here, we study the different parts of our method for better understanding of how the different parts influence the overall performance. Our results for ablation study are given

Method	Per-Turn	Per-Dialog
Mask-memN2N	93.4	32
Mask-memN2N (w/o Entropy)	92.1	24.6
Mask-memN2N (w/o L2 Mask Pre-Training)	85.8	2.2
Mask-memN2N (Reinforcement Learning Phase Only)	16.0	0

Table 4.3: Ablation study of our proposed method on permuted-bAbI dialog task. Results (accuracy %) are given in the standard setup, without match-type features.

in Table 4.3. We show results for Mask-memN2N in various settings - a) without entropy, b) without pre-training mask c) reinforcement learning phase only.

Adding entropy for the RL phase seems to have improved performance a bit by assisting better exploration in the RL phase. When we remove L2 mask pre-training, there is a huge drop in performance. In the RL phase, the action space is large. In the bAbI dialog task, which is a retrieval task, it is all the candidate answers that can be retrieved which forms the action set. L2 mask pre-training would help the RL phase to try more relevant actions from the very start.

From Table 4.3 it is clear that the RL phase individually does not perform well; it is the combination of both the phases that gives the best performance. When we do only the RL phase, it might be very tough for the system to learn everything by trial and error, especially because the action space is so large. Preceding it with the SL phase and L2 mask pre-training would have put the system and its parameters at a good spot from which the RL phase can improve performance. Note that it would not be valid to check performance of the SL phase in the test set as the SL phase requires the actual answers for it to create the mask.

4.7 Related Work

The idea of allowing the system to learn to attend to different parts of the state vector at different times depending on the input that the proposed model uses has been used in different settings before. To name a few, Bahdanau, Cho, and Bengio (2015a) use it for Machine Translation (MT), where the MT system can attend to different words in the input language sentence while producing different words in the output language sentence. Xu, Ba, et al. (2015) use it for image caption generation where the system attends to different parts of the image while generating different words in the caption. Madotto and Attardi (2017) use it for Question Answering (QA), where the QA system attends and updates different parts of the Recurrent Neural Network story state vector based on the sentence the system is reading in the input story.

In recent years, several corpora have been made available for building data-driven dialog systems (Serban, Lowe, et al. 2018). However, there are not as many good resources to train and test end-to-end models in goal-oriented scenarios. Some datasets are proprietary (e.g., Chen, Hakkani-Tür, et al. (2016)) or require participation to a specific challenge and signing a license agreement (e.g., DSTC4 (Kim et al. 2017)). Several datasets have been designed to train or test dialog state tracker components (Henderson, Thomson, and Young (2014), El Asri et al. (2017)), which are unsuitable for training end-to-end dialog systems, either due to limited number of conversations or due to noise. Recently, some datasets have been designed using crowdsourcing (Hixon, Clark, and Hajishirzi (2015), Wen, Gašić, Mrkšić, Su, et al. (2015), Su et al. (2015)) e.g., Amazon Mechanical Turk, CrowdFlower etc., but dialog systems built for them are harder to test automatically and involve another set of crowdsource workers for comparing them. Datasets designed using Wizard-of-Oz setting (Eric and Manning (2017), Wen, Vandyke, et al. (2017)) showed promise, but they were limited in scale. A recent dataset, Multi-Domain Wizard-of-Oz (MultiWOZ) is one of the largest goal-oriented dialog dataset. It was first introduced in Budzianowski et al. (2018), followed by corrections and improvements done by Ramadan, Budzianowski, and Gasic (2018) and Eric, Goel, et al. (2019). It has human-human written conversations spanning over six domains: booking, restaurant, hotel, attraction, taxi and train.

4.8 Conclusion

We propose a method that uses masking to handle the issue of making wrong updates at different times because of the presence of multiple valid next utterances in a dataset, but having access to only one of them at any time. The method has a SL phase where the mask uses the answer as well, and an RL phase, where the system learns to generate the mask solely from its dialog state vector. We modify the original-bAbI dialog task to create permuted-bAbI dialog task (which we have made publicly available), that has the issue of multiple next utterances as would be the case with any dataset created from human-human dialogs. Our experiments show that there is a significant drop in performance of the present neural methods in the permuted-bAbI dialog task compared to their performance on the original-bAbI dialog task. The experiments also confirm that the proposed method is a step in the right direction for bridging this gap.

CHAPTER 5

Learning with Maximal User Task Success and Minimal Human Agent Use

Goal-oriented dialog systems could reduce the workload of human agents for customer service and also reduce wait time for users. End-to-end learned neural goal-oriented dialog systems have shown great promise in recent years. However, their inability to handle new user behavior at deployment has limited their usage in real world. In this chapter, we propose an end-to-end trainable method for neural goal-oriented dialog systems which handles new user behaviors at deployment by transferring the dialog to a human agent intelligently. The proposed method has three goals: 1) maximize user's task success by transferring to human agents, 2) minimize the load on the human agents by transferring to them only when it is essential and 3) learn online from the human agent's responses to reduce human agents load further. We evaluate our proposed method on a modified-bAbI dialog task that simulates the scenario of new user behaviors occurring at test time. Experimental results show that our proposed method is effective in achieving the desired goals.

Published at: Transactions of the Association for Computational Linguistics (TACL 2019).

Authors: Janarthanan Rajendran, Jatin Ganhotra, and Lazaros C. Polymenakos.
Collaborations: IBM Research.

5.1 Introduction

End-to-end learned neural dialog systems show huge potential for various goal-oriented dialog tasks such as restaurant reservation, flight ticket booking and hotel reservation. However, their use in the real world has been limited. One main reason is their inability to handle new user behavior at deployment.

As discussed in Section 2.1, there are two main methods to build neural end-to-end goal-oriented dialog systems. In the first method, large amounts of human-human chat logs of a particular task are collected and then the dialog system is trained to mimic the chat logs using Supervised Learning (SL) (Bordes, Boureau, and Weston (2017)). In the second method, the dialog system is trained to complete the task against a human or a simulator (Zhao and Eskenazi (2016)). The training is done using Reinforcement Learning (RL) by providing reward for task completion and also intermediate rewards for pre-identified sub-task completion. This is often accompanied by a SL pre-training as in Liu and Lane (2017).

Dialog systems built using either of these methods would fail in the presence of new user behaviors during deployment, which were missing during training. Failure here refers to the inability of the dialog system to complete the task for the user. The new behaviors can be a different way of a user asking/providing certain information or could also be as simple as an user utterance with Out-Of-Vocabulary (OOV) words. The failure happens when these new behaviors are beyond the generalization capabilities of the trained systems. This happens generally because of the limited coverage of training data collected. For a real-world use-case, it is difficult to collect chat logs and/or to build a user simulator that covers all possible user behaviors, which implies that users with new behaviors are bound to come by during deployment. The new user behaviors that we refer to and focus here are only those that are valid and acceptable, i.e., these user behaviors should ideally have been part of the training data (chat logs in the case of SL and user simulator behavior pattern in the case of RL).

For an enterprise that uses this dialog system, these failures could affect their business directly. In addition to losing customers who faced these failures, the enterprise also loses future users/customers, as it also affects the perceived reliability of the dialog system and hence, the enterprise's reputation. While the dialog system fails for new user behaviors, it can perform the task well for majority of user behaviors. However, these failures have restricted the deployment of neural end-to-end goal-oriented dialog systems and have forced the enterprises to either rely completely on human agents or on other restrictive systems where the users have very little freedom in dialog.

There have been recent works on using user feedback (Liu and Lane (2017)) and active human teaching (Liu, Tür, et al. (2018)) during deployment to improve robustness to new user behaviors. While they have produced better systems, they are not yet good enough to be deployed in the real world. In this chapter, we propose a method which can be used in addition to the aforementioned ideas and enables a dialog system to perform well in the real world.

We propose an end-to-end trainable method, in which the dialog system can automatically identify a new user behavior during deployment that the system might fail at and transfer the task to a human agent, such that the user's task is completed without any issue. At the same time, the dialog system also learns from the human agent's response to handle that new user behavior in future. Our method also allows one to choose the trade-off between maximizing user's task success and minimizing the workload on human agents.

We set the following three goals for our method:

- Maximize task success rate for the user by transferring to a human agent in cases where the dialog system might fail
- Minimize the use of human agent by transferring to the human agent only when it is essential
- Learn online from the human agent's response to reduce the use of human agent over time

The rest of the chapter is organized as follows. Section 5.2 describes the proposed method. In Section 5.3, we introduce modified-bAbI dialog tasks, which simulate new user behaviours occurrence at deployment and serve as testbed for evaluating our proposed method. Section 5.4 details our experimental results on modified-bAbI dialog tasks. Section 5.5 discusses related work and Section 5.6 concludes.

5.2 Proposed Method

Consider a neural dialog model M trained for a goal-oriented dialog task. We also have a human agent H who is trained for the same task. Both M and H can take the dialog so far as input and produce the response for the user utterance (u). There is a neural transferrer T which uses the dialog state vector (s) from model M as input and decides whether to use the model M to provide response to the user or to transfer to the human agent H who could then provide the response to the user. The proposed method is shown in Fig 5.1 (right).

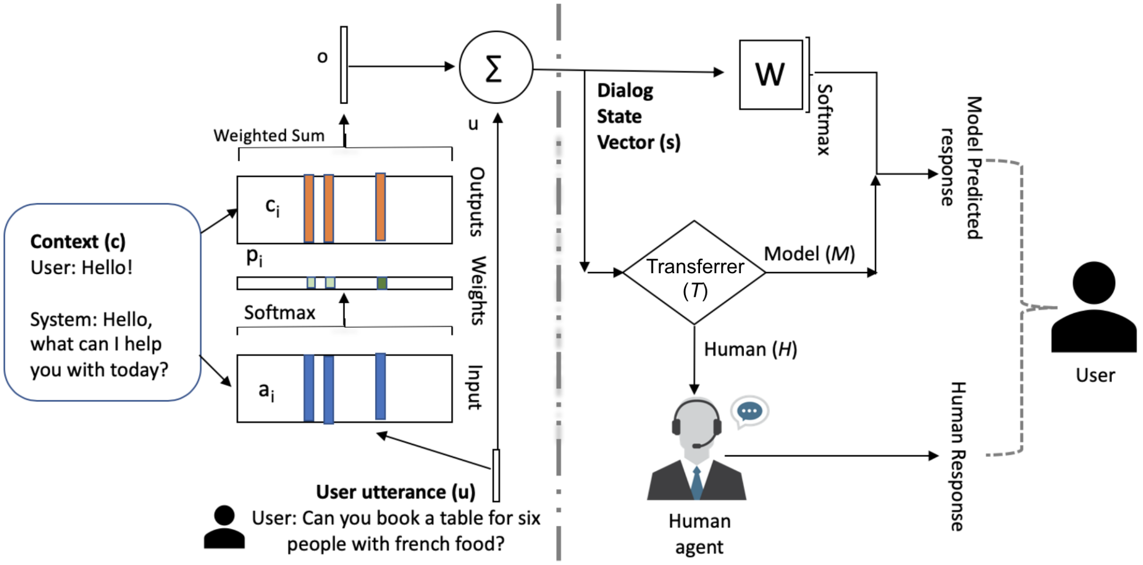


Figure 5.1: *Left*: A single layer version of memN2N (dialog model M) *Right*: Proposed Method

In a real world setting, we cannot expect the same exact user utterances to come during deployment, that the model came across during its training. Therefore, for a new dialog, it is not possible to know beforehand if model (M) would provide a correct response or not. The transferrer (T) has to learn this through trial and error and generalize. Therefore, the transferrer is trained using Reinforcement Learning (RL).

The transferrer is provided a high reward if it chooses the model M and the model produces a correct/valid response. The transferrer is however provided a relatively smaller reward if it chooses the human agent instead. We assume that the human agent's response is always correct. If the transferrer chooses the model M and the model provides an incorrect response, the transferrer (T) is penalized heavily. The validation, if a response is correct or not is provided by the user as feedback. The transferrer is trained using RL to make decisions (take actions) in order to maximize the above reward function. The reward function helps achieve two of our aforementioned goals -

- *Maximize task success rate for the user*: The reward function encourages the transferrer to learn the dialog scenarios in which the model M might fail and choose a human agent instead. Therefore, the transferrer helps to avoid sending an incorrect response to the user from the dialog model.

- *Minimize human agent use*: The reward function also encourages the transferrer to learn, identify and choose the model M in cases where the model has a high chance of providing the correct response, as the transferrer gets a higher reward compared to choosing a human agent. This minimizes the use of human agent to only when it is essential.

Here is an example reward function, which would achieve the desired goals:

- +1 : if human H is chosen
- +2 : if model M is chosen and the model's response is correct
- -4 : if model M is chosen and the model's response is incorrect

The reward function allows the designer to choose the trade off between maximizing the user task completion vs minimizing the human agent's workload. For example, when the model (M) is chosen, increasing the positive reward if the model's response is correct and reducing the penalty when the model's response is incorrect would encourage the overall system to use model (M) more to respond to the user.

The gradient updates obtained for the transferrer through the rewards received are also propagated back to the model M through the dialog state vector. This trains the model M to incorporate a) essential information about the dialog so far and b) the model's confidence in producing the correct response, in the dialog state vector, such that the transferrer can utilize it to make the right decision.

Whenever the transferrer chooses the human agent (H), the dialog interaction (including the human response) is also added to the training data of the model (M) and the model is updated online using supervised learning (SL). This helps achieve our third goal:

- *Reduce human agent use over time*: The online update allows the dialog model M to respond to the corresponding new user behavior and provide the correct response if a similar dialog scenario occurs in the future. This also enables the transferrer to reduce its dependence on the human agent (H) over time.

The transferrer keeps changing during its lifetime to adapt to the changes in the model M. Note that a human agent is involved only when the transferrer transfers the dialog to a human. The idea is generic enough to be used with any neural dialog model (M), e.g., HRED (Sordoni et al. 2015), end-to-end memory network (Sukhbaatar et al. 2015) etc. In our experiments, we use the end-to-end memory network for our model M and a Multi-Layer Perceptron (MLP) for the transferrer T. Additional details on the model architecture and training are provided in Section 5.4.

5.3 Modified-bAbI Dialog Tasks

bAbI dialog tasks (referred to as original-bAbI dialog tasks) were proposed by Bordes, Boureau, and Weston (2017) as a testbed to evaluate the strengths and shortcomings of end-to-end dialog systems in goal-oriented applications (described in detail in Section 2.2).

We propose modified-bAbI dialog tasks, an extension of original-bAbI dialog tasks (Bordes, Boureau, and Weston 2017). We modify the original-bAbI dialog tasks, by removing and replacing certain user behaviors from the training and validation data. The test set is left untouched. This simulates a scenario where some new user behaviors arise during the test (deployment) time that were not seen during the training and hence allows us to test our proposed method. This also mimics real-world data collection via crowdsourcing in the sense that certain user behavior is missing from the training data. Fig 5.2 shows a dialog sample from modified-bAbI dialog tasks.* We propose the following changes to the training and validation data of the original-bAbI dialog tasks:

- In Task 1, a user places a request that can contain from 0 to 4 of the required fields to book a table. The system asks questions to retrieve values for the missing fields and generate the correct corresponding API call. In modified-bAbI dialog tasks, the user doesn't provide the value for the price range by himself/herself and only provides that information when asked by the system.

- In Task 2, the user can request the system to update any of his/her preferences (cuisine, location, price range and number of people). In modified-bAbI dialog tasks, the user doesn't update his/her location preference.

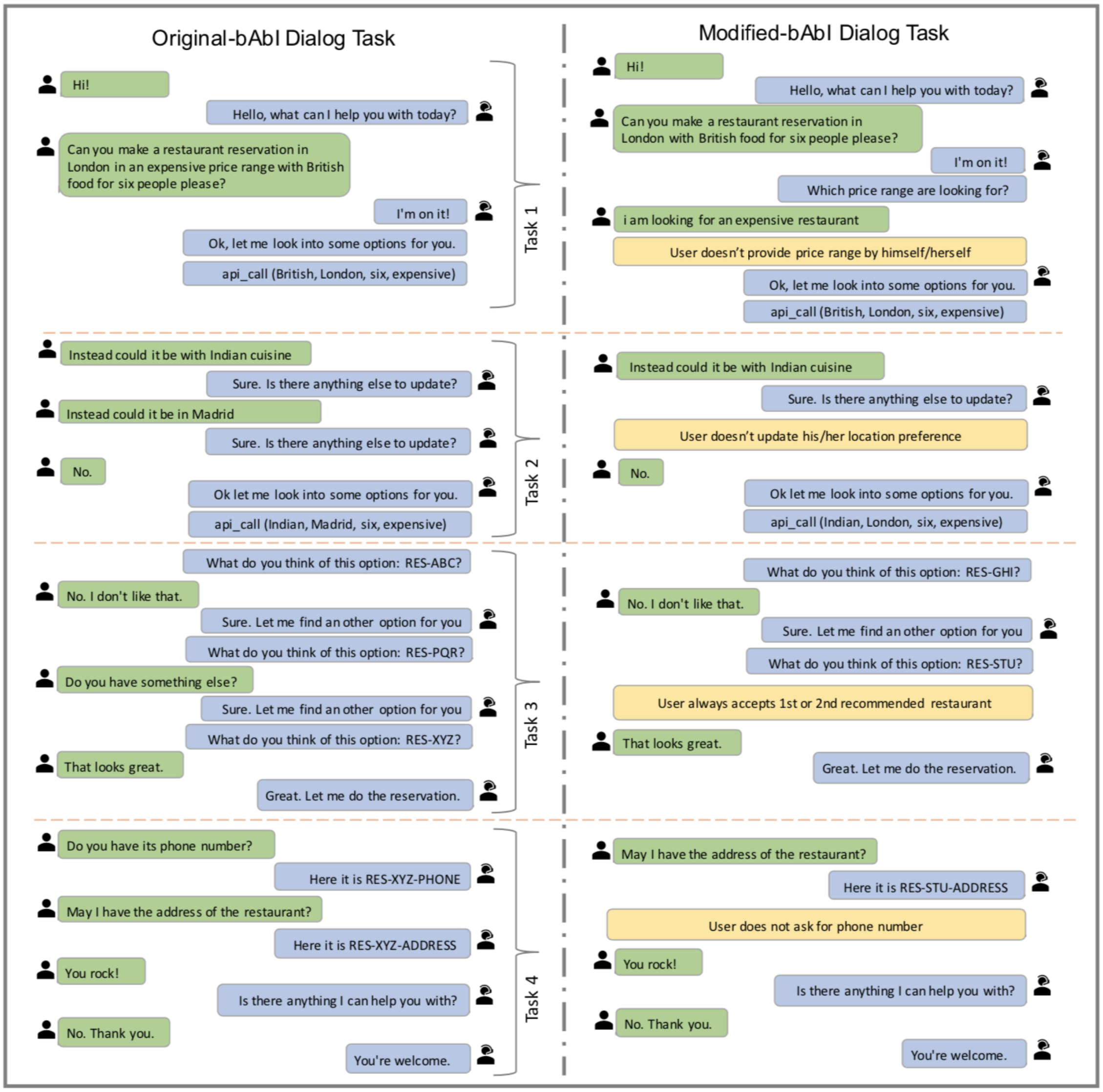
- In Task 3, for the API call matching the user request, information retrieved from the KB is provided as part of dialog history. The system must propose options to the user by listing the restaurant names sorted by their corresponding rating (in decreasing order). The system keeps proposing a new restaurant until the user accepts. In modified-bAbI dialog tasks, the user always accepts first or second recommended restaurant.

- In Task 4, the user can ask for phone number or address for the selected restaurant. In modified-bAbI dialog tasks, the user does not ask for phone number.

We incorporate the changes mentioned above to the final Task 5 (Conducting full dialogs). We perform experiments on modified-bAbI dialog task-5. The statistics for new user behavior in the test set (which was left untouched) are shown in Table 5.1 and 5.2.

---

*modified-bAbI dialog tasks - `https://github.com/IBM/modified-bAbI-dialog-tasks`



Original-bAbI Dialog Task

Task 1

Hi!
Hello, what can I help you with today?
Can you make a restaurant reservation in London in an expensive price range with British food for six people please?
I'm on it!
Ok, let me look into some options for you.
api_call (British, London, six, expensive)

Task 2

Instead could it be with Indian cuisine
Sure. Is there anything else to update?
Instead could it be in Madrid
Sure. Is there anything else to update?
No.
Ok let me look into some options for you.
api_call (Indian, Madrid, six, expensive)

Task 3

What do you think of this option: RES-ABC?
No. I don't like that.
Sure. Let me find an other option for you
What do you think of this option: RES-PQR?
Do you have something else?
Sure. Let me find an other option for you
What do you think of this option: RES-XYZ?
That looks great.
Great. Let me do the reservation.

Task 4

Do you have its phone number?
Here it is RES-XYZ-PHONE
May I have the address of the restaurant?
Here it is RES-XYZ-ADDRESS
You rock!
Is there anything I can help you with?
No. Thank you.
You're welcome.

Modified-bAbI Dialog Task

Hi!
Hello, what can I help you with today?
Can you make a restaurant reservation in London with British food for six people please?
I'm on it!
Which price range are looking for?
i am looking for an expensive restaurant
User doesn't provide price range by himself/herself
Ok, let me look into some options for you.
api_call (British, London, six, expensive)

Instead could it be with Indian cuisine
Sure. Is there anything else to update?
User doesn't update his/her location preference
No.
Ok let me look into some options for you.
api_call (Indian, London, six, expensive)

What do you think of this option: RES-GHI?
No. I don't like that.
Sure. Let me find an other option for you
What do you think of this option: RES-STU?
User always accepts 1st or 2nd recommended restaurant
That looks great.
Great. Let me do the reservation.

May I have the address of the restaurant?
Here it is RES-STU-ADDRESS
User does not ask for phone number
You rock!
Is there anything I can help you with?
No. Thank you.
You're welcome.

Figure 5.2: Modified-bAbI dialog tasks. A user (in green) chats with a dialog system (in blue) to book a table at a restaurant. We update each subtask in the original-bAbI dialog task with specific changes (in yellow).

Behavior Type	# Dialogs
Task-1	494
Task-2	539
Task-3	561
Task-4	752

Table 5.1: Modified-bAbI test set statistics. The numbers shown represent number of dialogs in test data (Task 5) out of 1000 dialogs where a certain type of new user behavior is encountered.

# New Behavior	0	1	2	3	4
# Dialogs	20	178	350	340	112

Table 5.2: Modified-bAbI test set statistics. The numbers shown represent number of dialogs in test data (Task 5) out of 1000 dialogs, where no new user behavior or one or more type of new user behavior is encountered.

5.4 Experiments and Results

5.4.1 Baseline Method: M

A dialog model M is trained on the modified-bAbI dialog task and is used for deployment. The model is not updated during test/deployment. An end-to-end memory network (Sukhbaatar et al. 2015) (described in detail in Section 2.3) architecture is used for the model M.

A single layer version of the memN2N model is shown in Fig.5.1 (left). The model is trained using SL on the training data. The trained model is then used during deployment. In our case, deployment is same as testing the model on the test data. Our results for the baseline method for the original and modified-bAbI dialog tasks are given in Table 5.3.

The hyperparameters used for the training the memN2N model in our baseline method are as follows: hops = 3, embedding_size = 20, batch_size = 32. The entire dialog model is trained using stochastic gradient descent (SGD) with learning rate = 0.01 and annealing (anneal_ratio = 0.5, anneal_period = 25), by minimizing the standard cross-entropy loss between the predicted response and the correct response. We learn two embedding matrices A and C for encoding context (input and output representations) and a separate embedding matrix B for encoding the query. We use position encoding for encoding word position in the sentence (Sukhbaatar et al. 2015). We also add temporal features to encode information about the speaker for the given utterance (user/system), similar to Bordes, Boureau, and Weston (2017) and weight matrices TA and TC are learned for encoding temporal features.

The same weight matrices mentioned above are reused for the 3 hops. The test results reported for the baseline method are calculated by choosing the model with highest validation per-turn accuracy.

Per-turn accuracy measures the percentage of responses that are correct (i.e., the correct candidate is selected from all possible candidates). Note that, as mentioned above in Section 5.3, we only modify the training and validation sets, and use the same test set. Per-dialog accuracy measures the percentage of dialogs where every response is correct. Therefore, even if only one response in a dialog is incorrect, this would result in a failed dialog, i.e., failure to achieve the goal of booking a table.

Dataset	Per-Turn	Per-Dialog
Original-bAbI	98.5	77.1
Modified-bAbI	81.7	3.7

Table 5.3: Test results (accuracy %) for our baseline method (M:memN2N) across original and modified bAbI dialog tasks.

From Table 5.3, we observe that the baseline method of using the trained memN2N dialog model performs poorly on modified-bAbI dialog tasks, which has new user behaviors at test time that the model has not seen during training. For modified-bAbI dialog tasks, the baseline method achieves 81.7% on per-turn accuracy and the per-dialog accuracy decreases to 3.7%. This implies that majority of dialogs would be unsuccessful in completing the goal. These results clearly demonstrate that the baseline method (end-to-end memory network) does not perform well on our proposed testbed, which simulates new user behaviours occurrence at deployment.

5.4.2 Proposed Method: $M^* + T^*$

We use the same memN2N dialog model used in the baseline method for the model M here. However, in our proposed method, we also have a transferrer T which takes as input the dialog state vector s and makes a decision on whether to use the model to respond to the user or to transfer the dialog to a human agent. For our experiments, T is a Multi-Layer-Perceptron (MLP) which outputs a probability distribution over the two actions. The sampled action is performed and the user feedback is used to determine the reward obtained, which is then used to train the transferrer and the model.

The following scenarios arise during deployment depending on the transferrer's action and the model's prediction:

Method	User Accuracy		Model Ratio	Final Model Accuracy	
	Per-Turn	Per-Dialog		Per-Turn	Per-Dialog
M	81.73 ± 0	3.7 ± 0	100.0 ± 0	81.73 ± 0	3.7 ± 0
Reward: 1, 2, -4					
$M + T^*$	92.85 ± 1.58	33.48 ± 10.59	51.97 ± 8.22	81.73 ± 0	3.7 ± 0
$M^* + T^*$	96.28 ± 1.16	54.5 ± 10.72	64.06 ± 4.65	90.83 ± 0.82	14.82 ± 3.7
$M_a^* + T^*$	96.19 ± 1.21	54.44 ± 11.40	61.14 ± 6.9	88.98 ± 0.34	10.26 ± 1.39
Reward: 1, 3, -3					
$M + T^*$	91.31 ± 1.15	26.50 ± 7.57	58.82 ± 4.62	81.73 ± 0	3.7 ± 0
$M^* + T^*$	94.67 ± 1.20	43.48 ± 8.80	70.33 ± 2.13	89.27 ± 0.74	12.84 ± 2.22
$M_a^* + T^*$	94.08 ± 1.0	38.8 ± 8.15	69.69 ± 6.14	88.75 ± 0.91	11.62 ± 2.61

Table 5.4: Test results for the different methods on the modified-bAbI dialog task. The numbers represent the mean and standard deviation of running the different methods across 5 different permutations of the test set. *User Accuracy*: Task success rate for the user; *Model ratio*: Percentage of time the transferrer chooses the model M; *Final Model Accuracy*: Accuracy of the model M at the end of testing.

- The transferrer could choose a human agent H to respond to the user. Since we use the test data of the modified-bAbI dialog task as a way to test deployment performance, we already have the ground truth labels for the different dialog scenarios that arise during the deployment. We use these ground truth labels as the human response.

- The transferrer chooses the model M and the model produces a correct or incorrect response. In real world, this validation/feedback on whether the response was correct or not is obtained from the user during deployment. For our experiments, we use the ground truth labels for test data to provide the validation from the user. In a sense, we mimic an ideal user using the test data ground truth labels.

We have two versions of the proposed method: a) the model trained on the training data is kept fixed during deployment and only the transferrer is updated ($M + T^*$) b) both the model and the transferrer are updated ($M^* + T^*$) during deployment. For both versions, the transferrer T is initialized randomly and is updated only during deployment. We use REINFORCE (Williams (1992)) for training the transferrer using the rewards obtained.

For $M^* + T^*$, the model is updated using the following three ways:

- The gradients obtained for the transferrer T are passed through the dialog state vector s to the model.

- The human responses provided for cases where the transferrer transferred the dialog to a human agent, are added to the training data to augment it and are also used to update the model using supervised learning.

- Dialogs are sampled from the augmented training data and are used to update the model using supervised learning to avoid forgetting.

The $M^* + T^*$ method uses a fixed number of samples, e.g., 2 batches for our experiments, from augmented training data to update the dialog model. We also implement and evaluate a variant of $M^* + T^*$ method: $M_a^* + T^*$ where the number of samples are decided based on the model M's performance on validation data. During deployment, after each batch of test data, the model is evaluated on the validation data. The difference between the current validation per-turn accuracy ($v_{acc}^{current}$) and the best validation per-turn accuracy so far (v_{acc}^{best}) estimates the loss in information learned from original training data during deployment. This is used to determine the number of batches ($b \geq 0$) for updating the model, as per the equation:

$$b = \alpha * (v_{acc}^{current} - v_{acc}^{best}). \tag{5.1}$$

The $M_a^* + T^*$ (a - adaptive) method tries to update the model with the right number of dialog samples from the augmented data.

We use the same hyperparameters as the baseline method mentioned above for training the model M. The transferrer T is trained using REINFORCE (Williams 1992) with a learning rate of 0.01. In $M + T^*$ after every batch of test data, the transferrer MLP is updated. In $M^* + T^*$ after every batch of the test data (deployment), along with the transferrer MLP update, the model is also updated in the three ways discussed. In $M^* + T^*$ the update using the human responses is done multiple times after every batch (3 in our case). For the update with the training data, two batches of training data are randomly sampled after every batch of test data.

We perform our experiments on two sets of reward functions. The first reward function is as follows:

- +1 : if human H is chosen
- +2 : if model M is chosen and the model's response is correct
- -4 : if model M is chosen and the model's response is incorrect

The results are shown in Table 5.4. The test data is provided sequentially, which mimics the deployment in a real-world setting. Since the performance depends on the order in which the test dialogs are provided, we evaluate our proposed method on 5 different permutations of the test set. We present the mean and standard deviation of the performance measures across the 5 permutations.

The performance measures used for evaluation are:

- *User Accuracy*: Task success rate as perceived by the user, irrespective of whether the response is provided by the human agent H or the model M. This measures our goal #1 - Maximize task success rate
- *Model ratio*: Percentage of time the transferrer T provides the model response to the user, i.e., human agent is not involved. This measures our goal #2 - Minimize human agent use
- *Final Model Accuracy*: Accuracy of the model M on the test data at the end of testing. This is obtained by evaluating the model M on the test data again after the testing phase is over. This measures our goal #3 - Reduce human agent use over time, by online learning of the model.

While the baseline method M gets a per-turn user accuracy of 81.73%, using and learning a transferrer ($M + T^*$) achieves user accuracy of 92.85%, an increase of more than 10 percentage points. If the model is also updated during the deployment ($M^* + T^*$), we observe further increase in per-turn accuracy (96.28%). While $M + T^*$ achieves better performance by using the model 51.97% of the time, $M^* + T^*$ achieves high accuracy by using model even more (64.06%), thereby reduces the human agent's workload. This is attributed to the improvement in the model during the deployment for the $M^* + T^*$ method. This is supported by the improvement in the model accuracy, going from 81.73% at the start of test to 90.83% by the end (shown as the final model M's accuracy in Table 5.2). We observe that $M_a^* + T^*$ does not provide an improvement, but performs similar to $M^* + T^*$ on all performance measures. The numbers reported are calculated as the running average of the different performance measures by evaluating on the fixed size test data (1000 dialogs) once sequentially. We expect an improvement in the various performance measures over longer periods of deployment (test time).

The reward function determines the trade off between the user's task success rate and the human agent's load. We perform additional experiments by modifying the reward function to showcase this trade off. For example, if we want to reduce the load on the human agent further, we can increase the reward provided when the model M is chosen and the model's response is correct and decrease the penalty when the model is chosen and model's response is incorrect. One such reward function is as follows:

- +1 : if human H is chosen
- +3 : if model M is chosen and the model's response is correct
- -3 : if model M is chosen and the model's response is incorrect

The results for the new reward function are shown in the last three rows of Table 5.4. In comparison with performance measures for reward function (1,2,-4), for both methods - $M + T^*$ and $M^* + T^*$, we observe a small drop in the user accuracy and a significant increase in model ratio, which showcases our intended goal in altering the reward function.

Table 5.5, 5.6, and 5.7 shows the results for ($M^* + T^*$), ($M^*_a + T^*$) and ($M + T^*$) methods respectively on all the 5 individual permutations of the modified-bAbI dialog task test set.

5.5 Related Work

The most common methods for end-to-end learning of neural goal-oriented dialog systems have been to either use supervised learning to train the dialog model on collected chat logs of the dialog task (Bordes, Boureau, and Weston (2017); Eric and Manning (2017); Wen, Vandyke, et al. (2017)) or use reinforcement learning to train against a user simulator or human agents, mostly with supervised learning pre-training (Li, Chen, et al. 2017; Henderson, Lemon, and Georgila 2008; Williams, Asadi, and Zweig 2017; Zhao and Eskenazi 2016).

The most common way to cope with new user behaviors during deployment has been to use reinforcement learning and learn from the failures using user feedback (Williams and Zweig 2016). These user feedback are generally binary (positive / negative) signals provided at the end of the dialog. Liu, Tür, et al. (2018) on top of learning from user feedback, also ask the users to correct the mistake by providing the correct system response. The dialog system is trained using both imitation learning (on the correct response) and reinforcement learning (from user feedback) to learn from the mistakes. The main drawback with these methods is that they do not really handle new user behaviors when they happen, i.e., they do not avoid task failures, rather they only provide a means to learn from the failures. Therefore, there might be lot of task failures for the users while the dialog system is learning to correct its predictions. Our proposed method on the other hand learns to proactively transfer the dialog to human agents anticipating a mistake in prediction, thereby avoiding some potential task failures for the users.

There has also been recent work on actively using human in the loop to teach and assist the learning of neural dialog systems prior to deployment.Asghar et al. (2017) use human agents to provide feedback by selecting the best response among the top responses predicted by the agent at any moment. Li, Miller, et al. (2017) use human agents to provide both numerical rewards as in conventional reinforcement learning and also textual feedback in natural language with richer information during training. Liu and Lane (2017) instead

Method	User Accuracy		Model Ratio
	Per-Turn	Per-Dialog	
M	81.73	3.7	100.0
Reward: 1, 2, -4			
$M + T^*$	92.44	33.1	50.98
	92.57	32.7	53.22
	93.96	42.0	44.63
	94.7	43.0	45.82
	90.59	16.6	65.24
Reward: 1, 3, -3			
$M + T^*$	90.41	21.1	54.67
	92.25	36.1	54.79
	92.12	32.5	58.07
	89.75	18.2	65.69
	92.05	24.6	60.92

Table 5.5: Test results for $M + T^*$ method on different permutations of modified-bAbI dialog task's test set

Method	User Accuracy		Model Ratio	Final Model Accuracy	
	Per-Turn	Per-Dialog		Per-Turn	Per-Dialog
M	81.73	3.7	100.0	81.73	3.7
Reward: 1, 2, -4					
$M^* + T^*$	97.49	65.2	62.15	91.33	18.2
	96.4	55.1	60.45	90.58	12.7
	94.47	38.3	71.13	89.54	9.7
	97.07	62.9	60.23	91.65	18.4
	95.99	51.0	66.35	91.06	15.1
Reward: 1, 3, -3					
$M^* + T^*$	92.77	29.2	73.7	88.52	11.4
	94.23	41.3	68.6	88.79	12.3
	95.22	46.5	70.1	89.19	11.3
	95.47	49.2	68.45	89.39	12.5
	95.68	51.2	70.8	90.46	16.7

Table 5.6: Test results for $M^* + T^*$ method on different permutation of modified-bAbI dialog task's test set

Method	User Accuracy		Model Ratio	Final Model Accuracy	
	Per-Turn	Per-Dialog		Per-Turn	Per-Dialog
M	81.73	3.7	100.0	81.73	3.7
Reward: 1, 2, -4					
$M_a^* + T^*$	96.15	51.8	64.64	88.52	9.9
	94.69	43.4	66.96	89.16	8.8
	96.52	56.5	56.57	88.75	9.6
	97.99	72.9	51.26	89.37	12.5
	95.63	47.6	66.27	89.12	10.5
Reward: 1, 3, -3					
$M_a^* + T^*$	92.51	26.9	79.16	87.81	9.9
	95.23	49.3	65.67	88.8	12.0
	94.03	36.4	70.06	87.82	8.2
	94.55	41.4	63.03	89.73	14.8
	94.1	40.0	70.57	89.57	13.2

Table 5.7: Test results for $M_a^* + T^*$ method on different permutation of modified-bAbI dialog task's test set

of using a human agent in the loop learn and improve a user simulator. They achieve this jointly optimizing the dialog agent and the user simulator using deep reinforcement learning.

While these approaches have focused on different ways to improve the neural goal-oriented dialog systems and maximize user success rate by a) improving the model or, b) better ways of online learning or, c) through human teaching; the problem of handling new user behaviors during deployment has not been handled yet. Our proposed method directly optimizes for maximum user success and provides a framework where existing techniques for model learning, online learning and human teaching can be used in tandem, to enable the end-to-end goal-oriented dialog systems ready for real-world use.

5.6 Conclusion

We introduce a new method for designing and optimizing goal-oriented dialog systems geared for real-world use. Our method allows the designer to determine the trade-off between the desired user's task success and human agent workload. Specifically, the proposed method allows us to maximize user success rate by minimally using human agents instead of the dialog model for cases where the model might fail. Our evaluation on the modified-bAbI dialog task shows that our proposed method is effective in achieving the desired goals. We believe this opens up a new and promising research direction that would spark an increase in the use of end-to-end learned neural goal-oriented dialog systems in the real world soon.

There are several limitations to our current evaluation, which we discuss below and hope is overcome in future works. For our experiments, we use an artificially constructed dataset, modified-bAbI dialog tasks, which incorporates two essential assumptions: a) a perfect human-agent and b) correct user feedback. For actual real-world deployments with real users, while the former assumption might still hold true, the latter might not always be true. While we use simple techniques for the different components in our method, they can be replaced with more sophisticated state of the art techniques for improved performance. For example, while we use REINFORCE, an on-policy method for training the transferrer, it would be interesting to try off-policy reinforcement learning techniques to use the samples more effectively. We could also try state of the art online learning methods to see how they affect the performance. In our experiments, the learning of the transferrer T starts from scratch during the deployment. An interesting future work is to explore ways of pre-training the transferrer before deployment, so that the learning of T can happen faster, with fewer samples during deployment. Another interesting direction is to draw on ideas from novelty detection methods to see if they can help the transferrer to generalize better.

CHAPTER 6

Learning to Learn From Related Tasks

For each new goal-oriented dialog task of interest, large amounts of data need to be collected for end-to-end learning of a neural dialog system. Collecting dialog data is a costly and time-consuming process. Instead of collecting large amounts of data for each new dialog task of interest, in this chapter we show that we can learn end-to-end goal-oriented dialog systems for a new task of interest with only limited amounts of training data by utilising already available data from a related dialog task. Using a meta-learning based method that selectively learns from the related dialog task data, our evaluation results show significant improvement in performance on the new task of interest.

Collaborators: Jonathan K. Kummerfeld and Satinder Singh.

6.1 Introduction

End-to-end learning of neural goal-oriented dialog systems has allowed learning and improving the dialog systems directly from data. It has reduced the human effort involved in designing dialog systems compared to pipeline-methods which require careful human adaptation of the different modules. Since the entire dialog system is learned directly from the dialog task's data, the design of the system need not make any assumption about the dialog domain and hence applicable to new domains with different training data, without much domain-specific hand-crafting, compared to rule based dialog systems. However, the human effort in data collection has significantly increased, as successful end-to-end training of neural goal-oriented dialog systems require large amounts of training data. As we discussed in Section 2.1, one common method for training neural end-to-end goal-oriented dialog systems is Supervised Learning (SL). For supervised learning of dialog systems, first a dataset of dialogs from human agents performing the task of interest is collected. The dialog system is then trained to imitate the human agent's dialogs in the collected dataset (Bordes, Boureau, and Weston 2017; Eric and Manning 2017; Wen, Vandyke, et al. 2017). The performance of the trained dialog system during deployment critically depends on the quantity and the quality of the data used for training. Typically, the higher the quantity and the quality of the data used for training, the better the generalisation performance of the trained dialog system can be. Therefore, for each new dialog task of interest, large amounts of new data has to be collected. It often costs a lot of money, effort and time to collect large amounts of data.

Instead of collecting large amounts of data for each new dialog task of interest, in this chapter we show that we can learn an end-to-end goal-oriented dialog system for a new task with only limited amounts of training data by utilising already available data from a related dialog task. Related dialog tasks are tasks that have parts/subtasks that are similar to / overlap with the new task of interest. For example, consider a scenario where we have collected data for, and built a dialog system to perform the task of hotel reservation and are now interested in building a system for the task of restaurant reservation. The dialog data used for training the hotel reservation system could have parts that might be useful to learn from, for the new task of restaurant reservation. E.g., the parts involving greeting and obtaining a user's name/contact information. They could also have parts that are contradictory for supervised learning of restaurant reservation. E.g., hotel reservation might require the dialog system to ask for the user's duration (number of days) of stay while the restaurant reservation might require the dialog system to ask for the particular day and time of table reservation. There could also be a lot of irrelevant information in the related tasks' data that would be best to ignore for a learning system with limited capacity. E.g., in the hotel reservation task, the user might enquire about the availability of fitness facilities and pool in the hotel which are irrelevant for the task of restaurant reservation. Data from the related tasks could also provide rich information about different user behaviors and natural language in general of both users and agents.

Tapping into and learning from the related tasks' data that is already available can potentially allow us to build dialog systems with improved performance on the new task of interest with only a limited amount data collected, saving us time, effort and money in data collection. One setting where the related tasks are naturally present is where the new task of interest is a modified/updated way of performing the task which was done earlier. For example, these modification can include things like asking for the email ID of the user instead of their mailing street address.

The key challenge in applying supervised learning on the data from a related task for the new task of interest is to identify which data points of the related task's data to learn from and which data points not to learn from. We want the dialog system to learn from the related task's data in a way that helps the dialog system improve its performance on the new task of interest and not degrade it (negative transfer (Wang et al. 2019)). In this chapter, we show that by selectively learning from the related task data, we can significantly improve the performance of end-to-end learned neural goal-oriented dialog system for a new task of interest with only limited amount of training data. For selectively learning from the related task data, we use a method that uses meta-gradients to automatically meta-learn which data points of the related task data to learn from and which data points not to learn from so as to perform well on the new task of interest. The rest of the chapter is organized as follows. Section 6.2 describes the proposed method. In Section 6.3, we describe the evaluation settings, experiments and results. Section 6.4 discusses the related work and Section 6.5 concludes the chapter.

6.2 Proposed Method

Let T^P be the new task of interest (primary task) for which we have collected a limited amount of training data. Let T^R be the related task for which we have relatively large amounts of data already available. We are interested in building a dialog system for the task T^P. The data points are of the form (context c, dialog system's next utterance a) pairs, where context has the history of the dialog so far including the recent user utterance. We learn a dialog model M parameterized by θ that takes as input the context c and predicts the next dialog system utterance a.

Algorithm overview. The algorithm is specified in Algorithm 1. Each iteration of the training is comprised of the following three major steps. 1) The dialog model is updated using a batch of data points from the primary task. 2) The dialog model is updated using a batch of data points from the related task, where each related task data point's training loss is weighted between $(0, 1)$. 3) The related task data point's weights are updated. These three steps are repeated at each iteration. We describe each step in detail below.

Updating the dialog model using primary task data points. We sample a batch of data points $\{\ldots, (c_i^P, a_i^P), \ldots\}$ from the primary task T^P. Let $L(M_\theta(c_i), a_i)$ represent the supervised learning prediction loss between $M_\theta(c_i)$: the next utterance predicted by the dialog model and a_i: the ground truth next utterance. Model M_θ is updated using the supervised learning prediction loss of the primary task data points L^P as shown below:

$$L^P(\theta) = \sum_i L\left(M_\theta(c_i^P), a_i^P\right) \tag{6.1}$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta L^P(\theta), \tag{6.2}$$

where α is the learning rate and $\nabla_\theta L^P(\theta)$ represents the gradient of the loss $L^P(\theta)$ with respect to θ.

Updating the dialog model using weighted related task data points. We sample a batch of data points $\{\ldots, (c_i^R, a_i^R), \ldots\}$ from the related task T^R. The supervised learning prediction loss $L(M_\theta(c_i), a_i)$ for each data point in the batch is weighed by a scalar weight $w_i \in (0, 1)$ corresponding to each of the data points. The scalar weight for each of the related task data point is obtained as a function of that particular data point. Let P parameterized by η be the module that outputs the weights. The weight for a related task data point is calculated as shown below:

$$w_i(\eta) = \sigma(P_\eta(c_i^R, a_i^R)), \tag{6.3}$$

where σ is a sigmoid function used to normalize the output of P to $(0, 1)$. Model M_θ is updated using the weighted prediction loss of related task data points L^R as shown below:

$$L^R(\theta, \eta) = \sum_i w_i(\eta) L\left(M_\theta(c_i^R), a_i^R\right) \tag{6.4}$$

$$\theta \leftarrow \theta - \beta \nabla_\theta L^R(\theta, \eta), \tag{6.5}$$

Algorithm 1

```
Input: learning rate parameters $\alpha$, $\beta$ and $\gamma$
Init: initialize $\theta$ and $\eta$ with random values
repeat
    Updating dialog model using primary task data
    Sample a batch of primary task data: $\{\dots, (c_i^P, a_i^P), \dots\}$
    Calculate loss: $L^P(\theta) = \sum_i L\left(M_\theta(c_i^P), a_i^P\right)$
    Update dialog model $M_\theta$: $\theta \leftarrow \theta - \alpha \nabla_\theta L^P(\theta)$
    Updating dialog model using weighted related task data
    Sample a batch of related task data: $\{\dots, (c_i^R, a_i^R), \dots\}$
    Calculate weights: $w_i(\eta) = \sigma(P_\eta(c_i^R, a_i^R))$
    Calculate loss: $L^R(\theta, \eta) = \sum_i w_i(\eta) L\left(M_\theta(c_i^R), a_i^R\right)$
    Update dialog model $M_\theta$: $\theta \leftarrow \theta - \beta \nabla_\theta L^R(\theta, \eta)$
    Updating related task data points' weights
    Sample a batch of related task data: $\{\dots, (c_i^R, a_i^R), \dots\}$
    Calculate weights: $w_i(\eta) = \sigma(P_\eta(c_i^R, a_i^R))$
    Calculate related task loss: $L^R(\theta, \eta) = \sum_i w_i(\eta) L\left(M_\theta(c_i^R), a_i^R\right)$
    Simulate dialog model update $M_\theta$: $\theta' = \theta - \beta \nabla_\theta L^R(\theta, \eta)$
    Sample a batch of primary task data: $\{\dots, (c_i^P, a_i^P), \dots\}$
    Calculate primary task loss: $L^P(\theta') = \sum_i L\left(M_{\theta'}(c_i^P), a_i^P\right)$
    Update related task data points' weights: $\eta \leftarrow \eta - \gamma \nabla_\eta \theta' \nabla_{\theta'} L^P(\theta')$
until done
```

where β is the learning rate and $\nabla_\theta L^R(\theta, \eta)$ represents the gradient of the loss $L^R(\theta, \eta)$ with respect to θ. The weights w_i allow for selectively using data points from the related task data for updating the dialog model.

Updating the related task data points' weights. In this key step of our proposed method, we update the related task data points' weights $w_i(\eta)$. The update increases the weights of related task data points that improve the dialog model's performance on the primary task (our task of interest) when learned from and decreases the weights of those that degrade the dialog model's performance on the primary task.

We first sample a batch of related task data points and simulate how the model parameters θ would change if we update the model with a batch of related task data points with the current assignment of weights provided by P_η:

$$L^R(\theta, \eta) = \sum_i w_i(\eta) L\left(M_\theta(c_i^R), a_i^R\right) \tag{6.6}$$

$$\theta' = \theta - \beta \nabla_\theta L^R(\theta, \eta). \tag{6.7}$$

We then evaluate how the updated model $M_{\theta'}$ performs on the primary task to decide how to change P_η that assigned weights to the related task data points that resulted in $M_{\theta'}$:

$$L^P(\theta') = \sum_i L\left(M_{\theta'}(c_i^P), a_i^P\right), \tag{6.8}$$

where $L^P(\theta')$ is the supervised learning loss of the updated model $M_{\theta'}$ on a new batch of data points sampled from the primary task. The parameters of P_η are updated as shown below:

$$\begin{aligned} \eta \leftarrow & \eta - \gamma \nabla_\eta L^P(\theta') && (6.9) \\ = & \eta - \gamma \nabla_\eta \theta' \nabla_{\theta'} L^P(\theta'), && (6.10) \end{aligned}$$

where γ is the learning rate and $\nabla_\eta L^P(\theta')$ represents the gradient of the loss $L^P(\theta')$ with respect to η. The gradient, $\nabla_\eta L^P(\theta')$ is split into products of two gradients $\nabla_\eta \theta'$ and $\nabla_{\theta'} L^P(\theta')$ using the chain rule. $\nabla_\eta \theta'$ can be calculated using meta-gradients as follows:

$$\begin{aligned} \nabla_\eta \theta' = & \nabla_\eta(\theta - \beta \nabla_\theta L^R(\theta, \eta)) && (6.11) \\ = & \nabla_\eta\left(\beta \nabla_\theta L^R(\theta, \eta)\right) && (6.12) \\ = & \nabla_\eta\left(\beta \nabla_\theta\left(\sum_i w_i(\eta) L(M_\theta(c_i^R), a_i^R)\right)\right) && (6.13) \\ = & \beta \sum_i \nabla_\eta w_i(\eta) \nabla_\theta L(M_\theta(c_i^R), a_i^R). && (6.14) \end{aligned}$$

Discussion. The proposed method therefore learns a dialog model from the primary task data points and also selectively from the related task data points. The proposed method meta-learns, at different points in training of the dialog model, which related task data points to learn from (and also to what degree $(0, 1)$). The weight assigned to a particular related task data point can therefore vary across training.

For simplicity, we described our proposed method with one update using primary task data points, one update using weighted related task data points and one update of related task data point weights in each iteration. But in practice, we make multiple updates to the related task data points' weights (η parameters) within each iteration. Also, for each η update we simulate how the model parameters θ change over multiple gradient updates (instead of just one as described in equations 6.6 and 6.7). This allows for a better estimate of how the updates using related task data points with the current parameters η affect the updated dialog model's performance on the primary task. Note that our proposed method is agnostic to the exact neural network architecture of the model M and weight module P. Also, while we focus on settings with a single related task, the proposed method naturally extends to settings with more than one related tasks.

While in our proposed method, we achieve selective learning by meta-learning the weights of the related task data points, there are other techniques in the machine learning literature that could potentially be used for learning/assigning these weights to the related task data points. Some of these techniques are discussed in Section 6.4.

6.3 Experiments and Results

We first illustrate our proposed method on a simple image classification task with a hand designed related task that allows us to verify if the proposed method can learn meaningful/expected weights for the related task data points. We then evaluate our proposed method on the task of personalized restaurant reservation.

6.3.1 MNIST Image Classification

The purpose of this experiment is to illustrate in a very simple setting how our proposed method works. For this, we design the experiment such that there is a clear expectation on which related task data points should have high weights and which should have low weights for it to be useful for the primary task.

The primary task of interest is the classification of hand-written digits from the MNIST dataset (LeCun, Cortes, and Burges 2010). The related task is created by taking the primary task data and changing the label to an incorrect one for 75% (37,500 data points) of the training data. So among the related task data points, 25% (12,500 data points) of them are useful for the primary task while 75% (37,500 data points) of them with incorrect labels are not. We also add an indicator to every related task data point that tells whether that data point has correct label or not.

Our proposed method has three high level steps at each iteration 1) updating the model with primary task data, 2) updating the model with weighted related task data and 3) updating the weights of the related task data points. To study closely the effect of our learned related task data point weights on the performance, in this experiment we perform only the last two steps. i.e., we update the model only using the weighted related task data. No updates are made to the model using the primary task data. Primary task data is however used while calculating the meta-gradients for step 3, for updating the weights of the related task data points. In order for the dialog model to perform well on the primary task, the data points with incorrect labels in the related task data should be weighed less compared to the data points that have the correct label.

6.3.1.1 Architecture and Training Details

The MNIST dataset contains 60,000 training images and 10,000 testing images. Among the training images, we use 50,000 for training and 10,000 for validation. The MNIST hand-written digit images are resized to 28 x 28 images and the pixel values are normalized to [0, 1]. The images are flattened to a 1-D array of 784 features (28 x 28). We use a logistic regression as our classification model (M_θ) and a perceptron with sigmoid non-linearity at the output for the weight generation module (P_η). The weight generation module takes as input the image, its label and the indicator that tells if that label is correct or not for that image, and produces a scalar weight between $(0, 1)$ as the output. The training uses a batch size of 256, with Adam optimizer (learning rate = 0.001, epsilon = 1e-8).

6.3.1.2 Results

Figure 6.1 (left) and Figure 6.2 (left) shows the weights assigned for the different related task data points by our proposed method during different stages of training. The points in red and green correspond to the weights of data points that have incorrect and correct labels respectively. Figure 6.1 (right) and Figure 6.2 (right) shows the histograms of number of data points in the different interval of weights. The histogram in red and green correspond to the related task data points with incorrect and correct labels respectively. We observe that, our proposed method with the meta-gradient based update to the weight generation module learns to give high weights to the data points with correct labels and low weights to the data points with incorrect labels. We observe similar weight assignment over multiple runs with different random seeds. In the last epoch of training, the average weight given to the related task data points with correct labels is 0.9747 ± 0.0004 and the average weight given to those with incorrect labels is 0.0074 ± 0.0002 (mean and standard deviation over 5 runs). Note that the method starts with random weights and updates the weights during training. The average weight across all the training epochs, given to the related task data points with correct labels is 0.8558 ± 0.0020 and to the related task data points with incorrect labels is 0.1508 ± 0.0041.

Table 6.1 shows the performance of the classification model with different types of weighting for the related task data points.

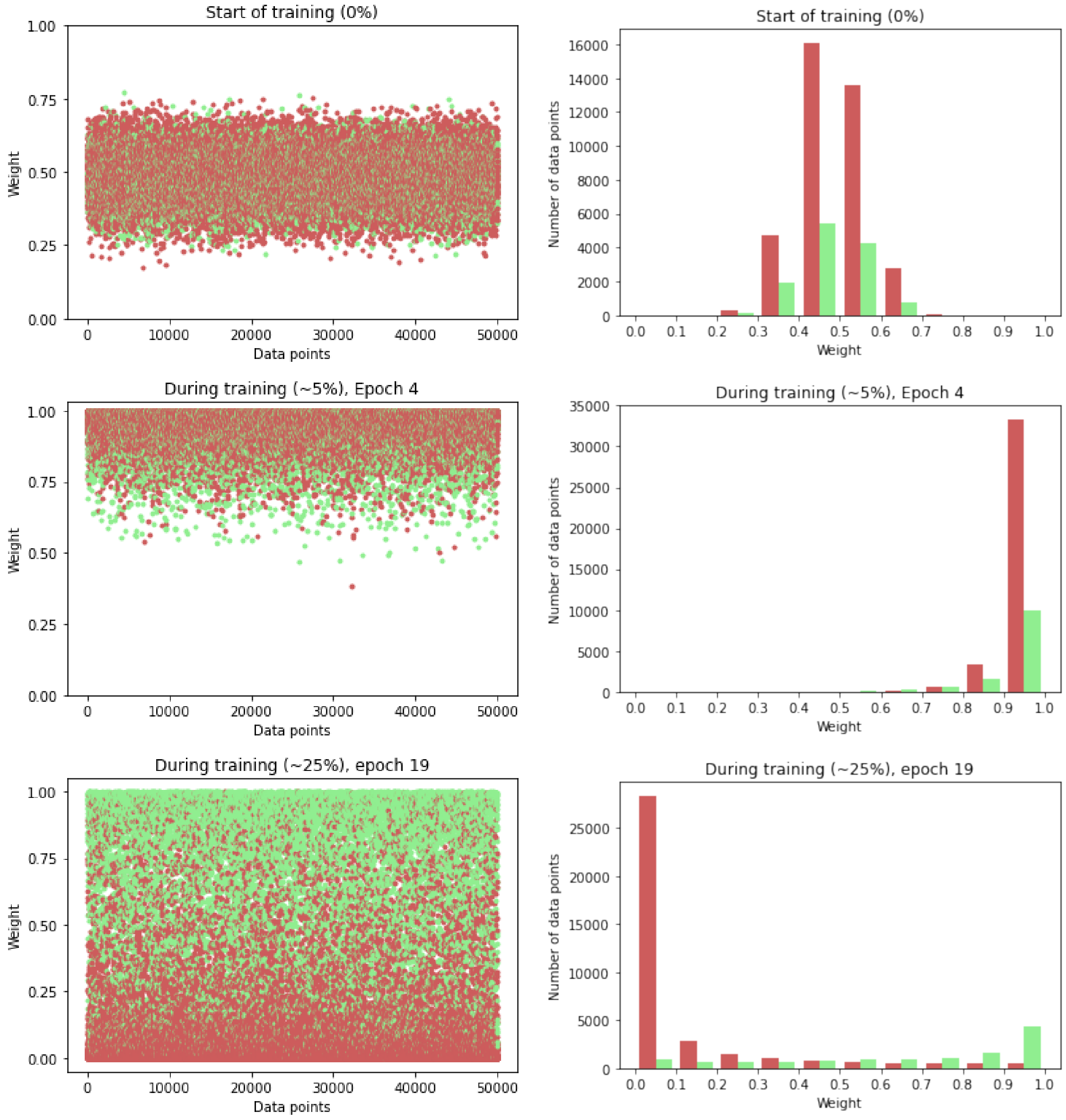


Figure 6.1: (Part 1/2) MNIST image classification. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. The points in red and green correspond to the weights of the related task data points that have incorrect and correct labels respectively. *Right*: Histograms of the number of data points in the different interval of weights. The histograms in red and green correspond to the related task data points with incorrect and correct labels respectively. Refer to Figure 6.2 for Part 2/2.

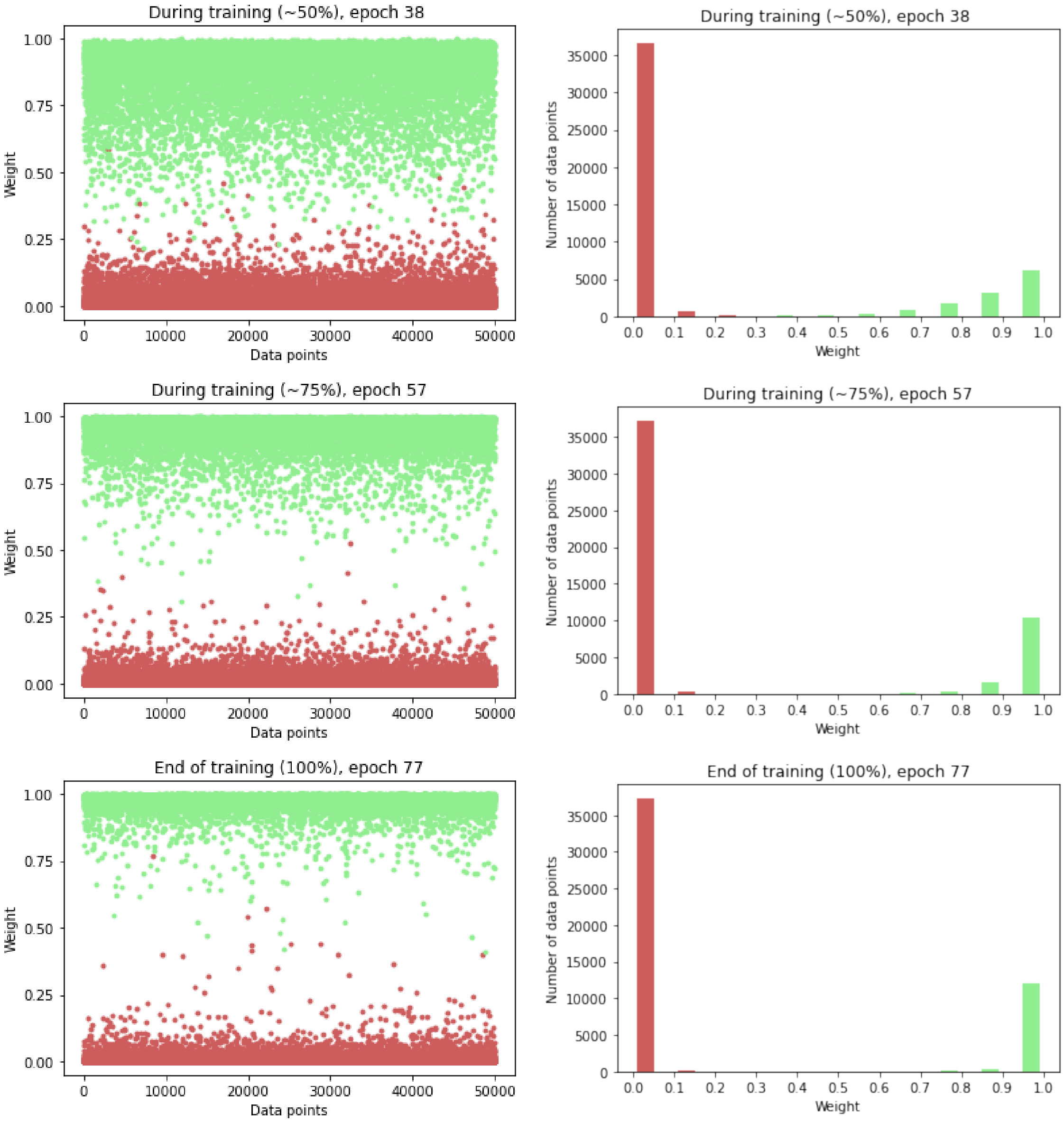


Figure 6.2: (Part 2/2) MNIST image classification. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. The points in red and green correspond to the weights of the related task data points that have incorrect and correct labels respectively. *Right*: Histograms of the number of data points in the different interval of weights. The histograms in red and green correspond to the related task data points with incorrect and correct labels respectively. Refer to Figure 6.1 for Part 1/2.

Method: Weighted Related Weights	% Accuracy
1 for All	21.63 ± 3.81
Random-Fixed	20.81 ± 4.46
Random-Changing	20.40 ± 4.42
Learned (Proposed Method)	87.86 ± 0.17
1 for Correct and 0 for Incorrect Labels (Perfect)	90.32 ± 0.33

Table 6.1: Test results, % classification accuracy (mean and standard deviation over 5 runs).

Random-Fixed: Assign a random weight between $(0, 1)$ for each of the related task data points. The weight for each of the data point is sampled once at the start of the training and then fixed throughout the training.

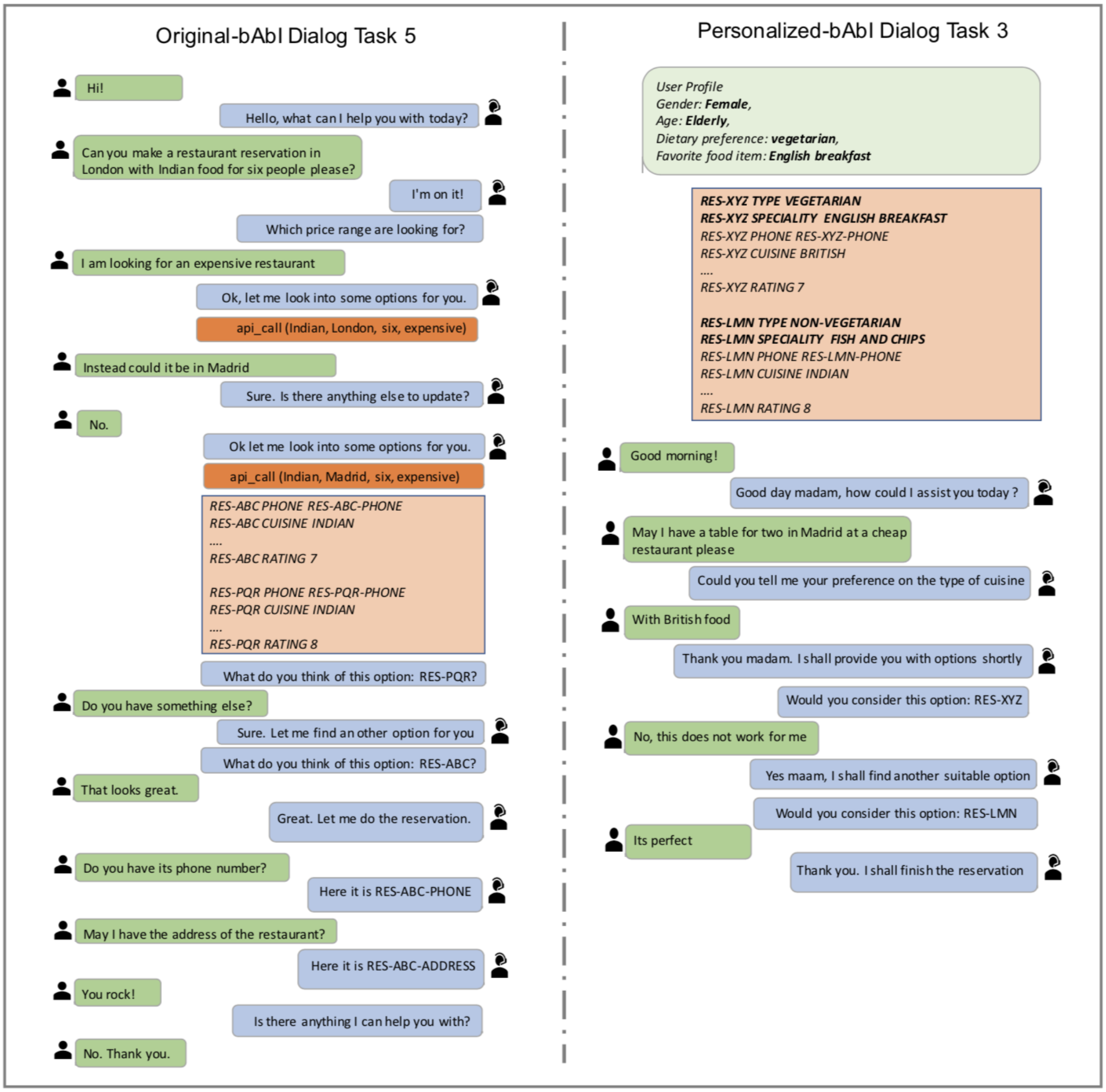
Random-Changing: Assign a random weight between $(0, 1)$ for each of the related task data points. Each time a data point is used, a new random weight is sampled for that data point.

From the results, we observe that selectively learning from the related task data points with learned weights (row 4) performs much better than the methods that use all the data points uniformly (row 1) or assigns random weights to the data points (rows 2 and 3). The proposed method's performance is very close to the method which has access to the perfect weights for the related task data points from the start (row 5) and throughout the training. We attribute this small gap mainly to the lingering effects of incorrect weights used for learning from the related task data points in the early stages of training in the proposed method.

6.3.2 Personalized Restaurant Reservation

Personalizing dialog system responses based on the user that the dialog system is interacting with will be a key step in seamless integration of dialog systems into our everyday lives. Recognizing this, Joshi, Mi, and Faltings (2017) proposed the first open dataset for training end-to-end dialog systems where the dialog system responses are based on the profile of the user. Their dataset is set in the domain of restaurant reservation, built as an extension of the bAbI dialog tasks released by Bordes, Boureau, and Weston (2017) (original-bAbI dialog tasks).

Original-bAbI dialog tasks is a testbed to evaluate the strengths and shortcomings of end-to-end dialog systems in goal-oriented applications (described in detail in Section 2.2). Figure 6.3 (Left) shows an example of Task 5 from the original-bAbI dialog tasks. In



Original-bAbI Dialog Task 5

Hi!

Hello, what can I help you with today?

Can you make a restaurant reservation in London with Indian food for six people please?

I'm on it!

Which price range are looking for?

I am looking for an expensive restaurant

Ok, let me look into some options for you.

api_call (Indian, London, six, expensive)

Instead could it be in Madrid

Sure. Is there anything else to update?

No.

Ok let me look into some options for you.

api_call (Indian, Madrid, six, expensive)

RES-ABC PHONE RES-ABC-PHONE
RES-ABC CUISINE INDIAN
....
RES-ABC RATING 7

RES-PQR PHONE RES-PQR-PHONE
RES-PQR CUISINE INDIAN
....
RES-PQR RATING 8

What do you think of this option: RES-PQR?

Do you have something else?

Sure. Let me find an other option for you

What do you think of this option: RES-ABC?

That looks great.

Great. Let me do the reservation.

Do you have its phone number?

Here it is RES-ABC-PHONE

May I have the address of the restaurant?

Here it is RES-ABC-ADDRESS

You rock!

Is there anything I can help you with?

No. Thank you.

Personalized-bAbI Dialog Task 3

User Profile
Gender: ***Female****,*
Age: ***Elderly****,*
Dietary preference: ***vegetarian****,*
Favorite food item: ***English breakfast***

RES-XYZ TYPE VEGETARIAN
RES-XYZ SPECIALITY ENGLISH BREAKFAST
RES-XYZ PHONE RES-XYZ-PHONE
RES-XYZ CUISINE BRITISH
....
RES-XYZ RATING 7

RES-LMN TYPE NON-VEGETARIAN
RES-LMN SPECIALITY FISH AND CHIPS
RES-LMN PHONE RES-LMN-PHONE
RES-LMN CUISINE INDIAN
....
RES-LMN RATING 8

Good morning!

Good day madam, how could I assist you today ?

May I have a table for two in Madrid at a cheap restaurant please

Could you tell me your preference on the type of cuisine

With British food

Thank you madam. I shall provide you with options shortly

Would you consider this option: RES-XYZ

No, this does not work for me

Yes maam, I shall find another suitable option

Would you consider this option: RES-LMN

Its perfect

Thank you. I shall finish the reservation

Figure 6.3: A user (in green) chats with a dialog system (in blue) to book a table at a restaurant. *Left*: (Related Task) An example dialog from Original-bAbI dialog Task 5. *Right*: (Primary Task) An example dialog from Personalized-bAbI dialog Task 3.

Joshi, Mi, and Faltings (2017)'s extension of the original-bAbI dialog tasks (referred to as personalized-bAbI dialog tasks from here on), in addition to the goal of the original-bAbI dialog tasks, the dialog system should also use the additional user profile information provided, to personalize the response styles and reasoning over the Knowledge Base (KB). The user profile consists of the user's age (young, middle-aged, elderly), gender (male, female), dietary preference (vegetarian, non-vegetarian) and favorite food item (Fish and Chips, Biryani, etc). The style of the dialog system's response depends on the age and the gender of the user. In Task 3, from the restaurants retrieved through the API call (that match the user's preferences), the dialog system now has to sort and suggest restaurants based not just on the restaurant's rating, but also based on the user's dietary preference and favorite food item. For this, the personalized-bAbI dialog tasks have additional information of restaurant type (vegetarian or non-vegetarian) and restaurants speciality (Fish and Chips, Biryani etc) of each restaurant added to the KB. Figure 6.3 (Right) shows an example of Task 3 from the personalized-bAbI dialog tasks.

For our experiments, we use Task 3 from the personalized-bAbI dialog tasks as our primary task of interest T^P, and Task 5 of the original-bAbI dialog tasks as the related task T^R. 100% (1000 dialogs) of the training dialogs of the original-bAbI dialog Task 5 are available as the related task data. For the primary task, we simulate the limited data availability setting by restricting the number of training and validation dialogs available. We look at three data settings with 5% (50 dialogs), 10% (100 dialogs) and 15% (150 dialogs) of the primary task training and validation dialogs from the dataset made available. We use 100% (1000 dialogs) of the test dialogs from the dataset for testing.

Let us look at the similarities and differences between the related task (Task 5 of the original-bAbI) and the primary task (Task 3 of the personalized-bAbI). The related task has parts in its dialog, such as the greetings and getting information from the user, that are semantically similar to that of the primary task. They are not exactly the same due to the differences in response style (the style differs based on the user profile in the primary task). Due the presence of different response styles, the vocabulary of the primary task is also much larger and different than that of the related task. The related task also has parts in its dialog that are semantically different, such as the ordering of the restaurants to suggest to the user. In the primary task the ordering should be based on the restaurant's rating, user's dietary preference and favorite food, while its based just on the restaurant's rating in the related task. There are also parts in the related task dialog that are not relevant to the primary task. These includes the parts corresponding to Task 2 and Task 4 of the related task. i.e., parts corresponding to updating API calls and providing extra information such as the restaurant's direction or contact information. These sub tasks are not part of the primary task.

6.3.2.1 Architecture and Training Details

As noted earlier, our proposed method is agnostic to the dialog model's (M_θ) architecture. In our experiments we use the same dialog model architecture as used in Joshi, Mi, and Faltings (2017), end-to-end memory networks (Sukhbaatar et al. 2015) (described in detail in Section 2.3). The sentences in the dialog are encoded using BoW (Bag of Words) encoding. The encoded sentences, which are part of the dialog history, are stored in the memory and the query (last user utterance) embedding is used to attend over the memory (3 times) to get relevant information from the memory. The generated internal state is used to select the candidate response from the list of candidates. The entire network is trained end-to-end using cross-entropy loss of the candidate selection.

We use an end-to-end memory network for the module P_η (that produces the weights for each of the related task data points) as well. In this case, the internal state generated after attending over the memory is used to generate the scalar $(0, 1)$ weight.

At each iteration of training, we make one update to the dialog model using primary task data (Equation 6.2), one update to the dialog model using weighted related task data (Equation 6.5) and 10 updates to the related task data points' weights, η (Equation 6.10) using meta-gradients. For each of the meta-gradient update we simulate how the model parameters θ changes over 5 gradient updates of weighted related task data points. We use the same hyper-parameters used by Joshi, Mi, and Faltings (2017) for both our end-to-end memory networks: embedding size = 20, batch size = 32, optimizer = Adam (learning rate = 0.001, epsilon = 1e-8).

6.3.2.2 Results

Table 6.2 shows the performance of our proposed method along with the baselines of not using related task and other conventional ways of using the related task data.

Primary: The dialog model is trained using only the primary task data.

Primary + Related Pre-Training: The dialog model is pre-trained with the related task data and then fine-tuned with the primary task data.

Primary + Related: The dialog model is trained using both related task and primary task data points.

Primary + Auxiliary Related (Multi-Task): The dialog model has two prediction heads, one for the primary task, and one for the related task, with a shared end-to-end memory network body that generates the internal state used for selecting the candidate response. This is similar to the conventional way of performing multi-task learning. This can also be interpreted as using related task prediction as an auxiliary task.

Primary + Weighted Auxiliary Related (Proposed Method): The dialog model has two prediction heads, one for the primary task and one for the related task, with a shared end-to-end memory network body that generates the internal state used for selecting the candidate response. The prediction loss of the related task is weighted by the weights learned by our proposed method during training.

Method	Number of primary task dialogs		
	50	100	150
Primary	54.7 ± 1.3	59.3 ± 0.5	61.1 ± 0.5
Primary + Related Pre-Training	32.8 ± 3.5	42.1 ± 4.7	47.8 ± 0.8
Primary + Related	37.1 ± 4.1	50.9 ± 1.7	58.6 ± 0.7
Primary + Auxiliary Related (Multi-Task)	51.2 ± 2.4	58.2 ± 1.2	60.6 ± 0.7
Primary + Weighted Auxiliary Related (Proposed Method)	57.7 ± 1.6	64.6 ± 0.8	67.1 ± 0.6

Table 6.2: Test results, % Per-turn retrieval accuracy (mean and standard deviation over 5 runs) in predicting the next dialog system utterance.

From the results in Table 6.2, we observe that, in all the three data settings, the conventional methods of using the related task data that we compare with (rows 2,3,4) lead to a reduction in performance (negative transfer) compared to not using the related task data points for training (row 1: Primary). The highest reduction in performance comes from pre-training with the related task data points (row 2: Primary + Related Pre-Training). We hypothesise that, due to the differences (vocabulary, contradicting and irrelevant sub-tasks) in the primary and related task, starting from the pre-trained network weights obtained using related task data leads to a worse local minima during fine-tuning with primary task data compared to starting from a randomly initialized network weights and training with primary task data alone. Learning from both the primary task data and related task data simultaneously with shared network body, and separate prediction heads (row 3: Primary + Auxiliary Related) seems to reduce the amount of net negative transfer, as the network weights do not specialize just for the related task. Our proposed method (row 4: Primary

+ Weighted Auxiliary Related) which weights the auxiliary related task update between $(0, 1)$ and selectively learns from them avoids negative transfer and provides a $\approx 6.5\%$ increase in performance across all three data settings, compared to the method with a uniform weight of 1 (row 3: Primary + Auxiliary Related). The net improvement in performance, compared to not using the related task data for training (row 1: Primary) is around $\approx 3\%$, $\approx 5\%$ and $\approx 6\%$ for the data setting of 50, 100, and 150 primary dialogs respectively. The increase in improvement with the increase in number of primary task data available indicates that the related task data (fixed in size) can be utilised better with more primary task data. There could be two explanations for this. 1) More primary task data facilitate our meta-gradient based proposed method to learn better weights. 2) More primary data leads to a dialog representations learned by the neural network that better utilises related task data and generalizes to the primary task. Further study is required to disentangle and understand these two possibilities. Also, note that the performance of the proposed method using 100 primary task dialogs is better than that of the method that learns from just the primary task dialogs with 150 dialogs. While selectively learning can always help with avoiding negative transfer by lowering the weights for data points that lead to negative transfer, the improvement in performance (compared to not using related task data) possible by using the related task data points will depend on the relationship between the primary task and the related task.

Method: Primary + Weighted Auxiliary Related	Number of primary task dialogs		
Weights	50	100	150
$0 \equiv$ Primary	54.7 ± 1.3	59.3 ± 0.5	61.1 ± 0.5
$1 \equiv$ Primary + Auxiliary Related	51.2 ± 2.4	58.2 ± 1.2	60.6 ± 0.7
Random-Fixed	50.7 ± 2.0	58.7 ± 0.8	61.2 ± 1.0
Random-Changing	52.3 ± 0.9	58.7 ± 1.0	59.8 ± 0.8
Learned (Proposed Method)	57.7 ± 1.6	64.6 ± 0.8	67.1 ± 0.6

Table 6.3: Test results, Per-turn % retrieval accuracy (mean and standard deviation over 5 runs) in predicting the next dialog system utterance.

To verify that it is not any random weighting of the auxiliary related task that leads to the improvement in performance, Table 6.3 compares the performance of the method: Primary + Weighted Auxiliary Related with different types of weighting for the related task data points. It is clear from Table 6.3 that not any random weighting can lead to the improvement in performance that we observe when we learn the weights using our proposed method.

Figure 6.4 (left) and Figure 6.5 (left) show the weights assigned for the different related

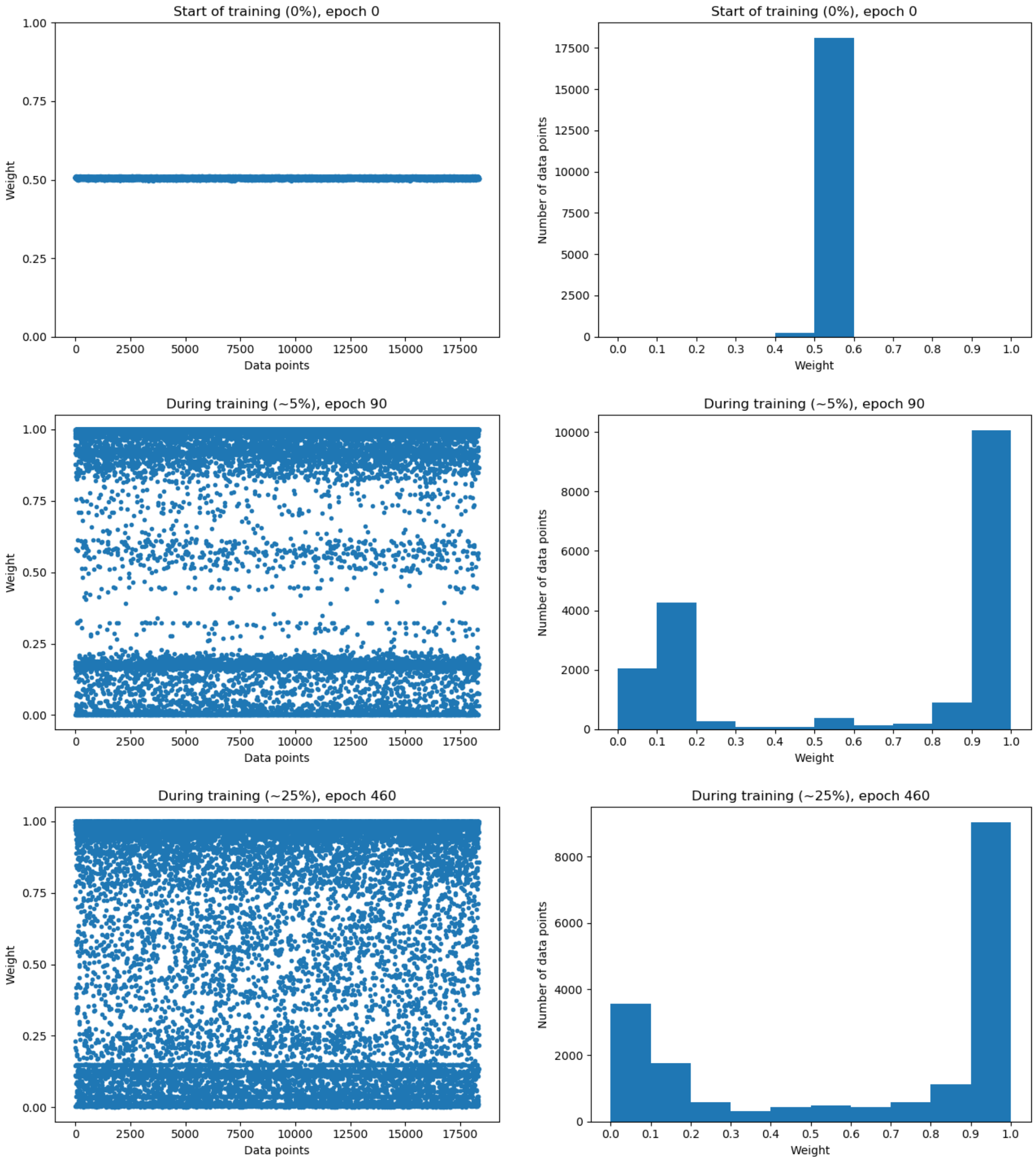


Figure 6.4: (Part 1/2) Personalized Restaurant Reservation. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. *Right*: Histograms of the number of data points in the different interval of weights. Refer to Figure 6.5 for Part 2/2.

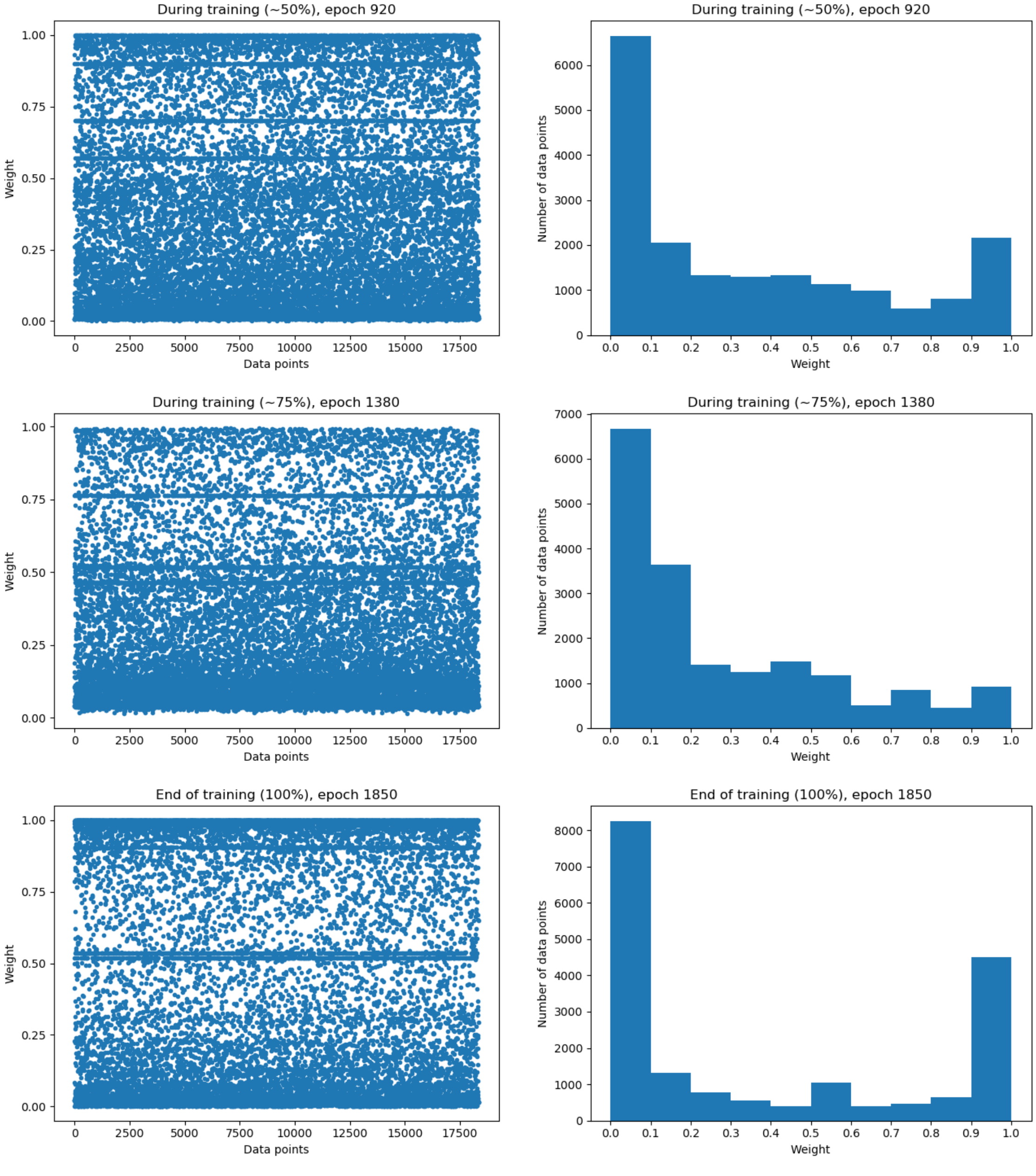


Figure 6.5: (Part 2/2) Personalized Restaurant Reservation. *Left*: The weights assigned for the different related task data points by our proposed method during different stages of training. *Right*: Histograms of the number of data points in the different interval of weights. Refer to Figure 6.4 for Part 1/2.

task data points by our proposed method during different stages of training. Figure 6.4 (right) and Figure 6.5 (right) shows the histograms of number of data points in the different interval of weights. The weight assignment and the histograms at different stages of training look different for a different run of the experiment with a different random seed. Unlike the simple MNIST image classification (Section 6.3.1), here it is not clear which related task data points should have high weights and which data points should not. Upon a closer investigation of the weights assigned to individual data points across training, we observe that some of them vary widely across training. The optimal weights for the related task data points at any given stage of training can be different from the optimal weights for them at a different stage of training, i.e., the optimal weights for the related task data points are non-stationary, as the optimal weights depend on the current state (parameters of the dialog model M) of the dialog system. For example, some data points of the related task which are quite different from the primary task might still be useful to learn from at the early stages of training to help with learning better representations for the vocabulary. Some data points that the dialog system has already learned from might get lower weights at later stages of learning so as to avoid overfitting and thereby helping with the prediction of other data points.

6.4 Related Work

Recognizing the difficulty in collecting large amounts of data for every new dialog task of interest, there have been several papers that look at ways to transfer knowledge from data from other tasks to cope with limited availability of training data for the task of interest. Zhao, Wu, et al. (2020) split the dialog model such that most of the model can be learned independent of the task of interest. They use large number of other dialogs tasks and unstructured documents to train these parts of the model. Only a small part of the dialog model with a small number of parameters needs to be trained with the limited number of dialogs from the task of interest. Wen, Gašić, Mrkšić, Rojas-Barahona, et al. (2016) pre-train the model with counterfeited data generated from different tasks while Lin et al. (2020) use pre-trained language models as initialization for the encoder and the decoder. Both works then fine-tune the dialog model with the data from the task of interest.

Recently, there have been works that explore ways to automatically learn some aspects of the transfer process using meta-learning. Xu, Zhu, et al. (2020) looks at the problem of learning a joint dialog policy using reinforcement learning in a multi-domain setting which can then be transferred to a new domain. For this, they decompose the state and action representation into features that correspond to low level components that are shared across domains, facilitating cross-domain transfer. They also propose a MAML (Model Agnostic Meta Learning) (Finn, Abbeel, and Levine 2017) based meta-learning extension that learns to adapt faster to a new domain. Madotto, Lin, et al. (2019), Mi et al. (2019), Qian and Yu (2019) and Dai et al. (2020) also look at multi-domain settings. They all use MAML based meta-learning method to learn an initialization that adapts fast with few dialog samples from a new task. Dai et al. (2020), in addition to learning an initialization using MAML that adapts fast to a new task, also learn whether to use the model's prediction or to shift to a human agent. All of the above papers look at settings where there is access to a large set of training tasks. The meta-learning system learns to transfer knowledge to a new task at test time by learning how to do that for different tasks during training. Though an individual task might have only a limited amount of dialog data, they need a lot of such tasks during training. In contrast, in this chapter we look at a setting where the task from which we want to transfer knowledge from and the task that we want to transfer knowledge to are the only tasks that we have access to at the training time. Any learning on how to best transfer knowledge has to happen from just these two tasks. None of the above methods are applicable to this setting. In our setting, the meta-learning agent has to transfer knowledge from the related task to the primary task while also learning how to do it well with just those two tasks to train on.

Learning a task, while simultaneously meta-learning certain aspects of the learning process has been done successfully in some supervised learning and reinforcement learning settings recently. Wu, Ren, et al. (2018) and Wichrowska et al. (2017) use meta-learning to adapt the hyperparemeters such as learning rate and and even learn the entire optimizers themselves during training for supervised learning tasks such as image classification. Given a single task, Zheng, Oh, and Singh (2018) successfully meta-learn intrinsic rewards (starting from a random values for the intrinsic rewards) that help the agent perform well on that task. Xu, Hasselt, and Silver (2018) use meta-gradients to learn reinforcement learning training hyper-parameters such as discount factor and bootstrapping parameters that work well for a given task. The meta-gradient technique used in our proposed method is closely related to my prior work Rajendran et al. (2020). There, we learn intrinsic rewards for a reinforcement learning agent acting in given domain, such that learning with those intrinsic rewards in that domain, improves the performance of the agent in the task of interest in a different domain.

In this chapter, our focus is on showing that the we can improve the performance of end-to-end learned neural goal-oriented dialog systems on the primary task of interest with only limited training data available, by selectively learning from a related task with relatively more training data. While we use a meta-learning based method for learning the weights for the related task data points, there are other techniques in the machine learning literature, especially in computer vision literature, that can potentially be used to learn the weights. A large section of these recent techniques are based on learning an adversarially trained discriminator for estimating the weights of related image classification task data points (Zhao, Zhang, et al. 2018; Cao et al. 2018; Sankaranarayanan et al. 2018; Wang et al. 2019). Jiang and Zhai (2007) use a combination of several domain adaptation heuristics to assign weights and evaluate on NLP tasks. Moon and Carbonell (2017) cluster the related task data points and learn attention weights for the clusters. An interesting future direction would be to study which weighting methods are best suited for end-to-end learning of neural goal-oriented dialog systems using related tasks and under what settings.

6.5 Conclusion

End-to-end learning of neural goal-oriented dialog systems requires large amounts of data for training. Collecting data is a costly and time consuming process. In this chapter we proposed a method that can utilise a related task's data when available so as to learn the new task of interest with only a limited amount of data. The related task data might however have some data points that are irrelevant or worse detrimental to learn from for the new task of interest. Our proposed method uses meta-learning to automatically learn which of the related task data points to selectively learn from. Our evaluation on learning the personalized restaurant reservation task by using additional data from a related restaurant reservation task shows that you can obtain a significant increase in performance by selectively learning from related task data using our proposed method.

CHAPTER 7

Conclusion

In this chapter, I conclude my thesis by first summarizing the contributions of this thesis, then discussing possible future work.

7.1 Summary of Contributions

In this thesis, we have identified and made progress in tackling four challenges in end-to-end learning of neural goal-oriented dialog systems.

NE-Table: A neural key-value table for named entities. For interacting with the large number of named entities that we encounter in goal-oriented dialog tasks, we proposed a method to build neural embeddings for them on the fly using their context, avoiding an explosion in vocabulary size. The context information enables distinguishing the different named entities from one another. By storing them in a key-value table with neural embedding as keys and the actual named entities as values, the proposed method allows for comparison and retrieval, using neural embeddings as well as actual named entities. This led to significant improvement in performance, especially in the presence of Out-Of-Vocabulary named entities.

Learning with multiple answers. For tackling the challenge of supervised learning with multiple valid next utterances but with access to only one of them, we proposed a method to learn to use different parts of the neural network to encode different predictions of next utterances. This helped with the learning of one while not interfering with the learning of others. Our experiments show considerable improvement in generalization performance.

Learning with maximal user task success and minimal human agent use. For the challenge of handling new user behaviors during deployment, we proposed a method to learn to efficiently transfer dialogs to human agents in order to make sure overall user task success remains high. The proposed method also allows for learning in an online fashion, on how to respond to the new user behaviors by looking at the human agent's responses. This helps reduce human agent's work load over time. Our experiments show that it is possible to achieve very high task success by minimally using human agents in the loop.

Learning to learn from related tasks. While neural network based dialog systems can achieve very high performance, they need a lot of quality data to learn from. We show that by selectively learning from a related task's data that is already available, we can improve the performance of a new task of interest that has only limited amount of training data.

7.2 Future Work

Actively learning from human agents. End-to-end training of goal-oriented dialog systems allows learning directly from data and user feedback. An interesting future direction is to explore ways to actively learn from the human agents during training of the dialog system, i.e., prior to deployment. The dialog system should identify parts of the dialog task that it needs help with and actively seek the human agent's help for those parts during training. This can potentially result in a dialog system with better performance at deployment. This could also reduce the amount of training data that needs to be collected, by removing redundancy in data collection and training. Overall, it would provide a means to make the training more effective and efficient.

Learning to learn from related data. Data useful for a dialog task of interest (related data) could be present in different formats. The related data need not necessarily be just in the form of another dialog task (as it is in Chapter 6). The related data could include, for example, natural language instructions on how to perform the task of interest, or dialog data from a related dialog task along with a description of how the new dialog task of interest is different from the related dialog task. In order to successfully utilise such related data, the system needs to identify what to learn from the related data. An interesting future direction is to explore ways to automatically learn what to learn from the related data. Apart from allowing us to use related data that are not dialog tasks, the system could potentially allow extraction of more useful information from a related dialog task than a fixed supervised learning prediction loss allows.

Leveraging pre-trained language models for dialog systems. End-to-end learned dialog systems use the dialog data to learn the natural language, its syntax, and semantics as a part of learning the dialog task of interest. In recent years, large language models such as GPT-3 (Brown et al. 2020) and BERT (Devlin et al. 2019) showing good understanding of the natural language have been successfully built. An interesting future direction is to explore ways to leverage such powerful language models for end-to-end learned neural goal-oriented dialog systems. Using pre-trained language models for encoding and generating the user/system dialog utterances can potentially reduce the amount dialog data required for training the dialog systems. They could also potentially facilitate learning of dialog systems using human feedback in natural language.

Lifelong learning dialog systems. To prevent performance degradation after deployment, dialog systems have to adapt to changes over time in user preferences, behaviors, and the tasks themselves. For example, a personalized movie recommendation system for a family has to identify when and how a person's preferences have changed and change the recommendations accordingly. If a new member joins the family, the system has to quickly learn the preferences of the new member in order to provide good recommendations. In the case of a flight ticket booking system, if there are changes to the information that need to be provided to and obtained from the user (e.g., information pertaining to health and safety, as was the case during the COVID-19 pandemic), the dialog system has to quickly learn to do that. Dialog systems that perform multiple tasks, have to be able to learn new tasks after deployment. While learning a new task, we want the dialog system to not forget the existing tasks, and use the knowledge of how to perform the existing tasks to learn the new task with minimal additional training. An interesting future direction is to identify and address the challenges in building neural end-to-end goal-oriented dialog systems that continually learn throughout their lifetime.

7.3 Concluding Remarks

In this thesis we identified some important challenges in end-to-end learning of neural goal-oriented dialog systems and made progress in tackling them. We believe the contributions of this thesis are also valuable for other researchers and engineers working on end-to-end goal-oriented dialog systems. By making progress in these challenges, we believe we have taken a step towards achieving the goal of seamless integration of goal-oriented dialog systems into our everyday lives.

BIBLIOGRAPHY

BIBLIOGRAPHY

Asghar, Nabiha, Pascal Poupart, Xin Jiang, and Hang Li (2017). "Deep Active Learning for Dialogue Generation". In: *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (SEM)*.

Bahdanau, D., K. Cho, and Y. Bengio (2015a). "Neural machine translation by jointly learning to align and translate". In: *International Conference on Learning Representations (ICLR)*.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015b). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations (ICLR)*.

Bobrow, Daniel G., Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd (1977). "GUS, a frame-driven dialog system". In: *Artificial Intelligence*.

Bordes, Antoine, Y-Lan Boureau, and Jason Weston (2017). "Learning end-to-end goal-oriented dialog". In: *International Conference on Learning Representations (ICLR)*.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin.

Budzianowski, Paweł, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Ultes Stefan, Ramadan Osman, and Milica Gašić (2018). "MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Cao, Zhangjie, Lijia Ma, Mingsheng Long, and Jianmin Wang (2018). "Partial Adversarial Domain Adaptation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*.

Chen, Hongshen, Xiaorui Liu, Dawei Yin, and Jiliang Tang (2017). “A Survey on Dialogue Systems: Recent Advances and New Frontiers”. In: *Proceedings of the Special Interest Group on Knowledge Discovery and Data Explorations (SIGKDD).*

Chen, Xianjie and Alan L Yuille (2014). “Articulated Pose Estimation by a Graphical Model with Image Dependent Pairwise Relations”. In: *Advances in Neural Information Processing Systems (NeurIPS).*

Chen, Yun-Nung, Dilek Hakkani-Tür, Gökhan Tür, Jianfeng Gao, and Li Deng (2016). “End-to-End Memory Networks with Knowledge Carryover for Multi-Turn Spoken Language Understanding.” In: *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH).*

Cuayáhuitl, Heriberto, Simon Keizer, and Oliver Lemon (2015). “Strategic Dialogue Management via Deep Reinforcement Learning”. In: *Workshop on Deep Reinforcement Learning, Advances in Neural Information Processing Systems (NeurIPS).*

Cucerzan, Silviu (2007). “Large-Scale Named Entity Disambiguation Based on Wikipedia Data”. In: *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CNLL).*

Dai, Yinpei, Hangyu Li, Chengguang Tang, Yongbin Li, Jian Sun, and Xiaodan Zhu (July 2020). “Learning Low-Resource End-To-End Goal-Oriented Dialog for Fast and Reliable System Deployment”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).*

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT).*

Dhingra, Bhuwan, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng (2017). “Towards End-to-End Reinforcement Learning of Dialogue Agents for Information Access”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL).*

Diba, A., V. Sharma, A. Pazandeh, H. Pirsiavash, and L. Van Gool (2017). “Weakly Supervised Cascaded Convolutional Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

El Asri, Layla, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman (2017). “Frames: a corpus for adding memory to goal-oriented dialogue systems”. In: *Proceedings of the 18th Annual Meeting of the Spec ial Interest Group on Discourse and Dialogue (SIGDIAL).*

Eric, Mihail, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyag Gao, and Dilek Hakkani-Tur (2019). “MultiWOZ 2.1: Multi-Domain Dialogue State Corrections and State Tracking Baselines”. In: *arXiv preprint arXiv:1907.01669.*

Eric, Mihail and Christopher D Manning (2017). “Key-Value Retrieval Networks for Task-Oriented Dialogue”. In: *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning (2005). “Incorporating non-local information into information extraction systems by gibbs sampling”. In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of International Conference on Machine Learning (ICML)*.

Goddeau, D., H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchai (1996). “A form-based dialogue manager for spoken language applications”. In: *Proceeding of 4th International Conference on Spoken Language Processing (ICSLP)*.

Gu, Jiatao, Zhengdong Lu, Hang Li, and Victor O.K. Li (2016). “Incorporating Copying Mechanism in Sequence-to-Sequence Learning”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Gulcehre, Caglar, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio (2016). “Pointing the Unknown Words”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Guo, Stephen, Ming-Wei Chang, and Emre Kiciman (2013). “To Link or Not to Link? A Study on End-to-End Tweet Entity Linking”. In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Guo, Xiaoxiao, Tim Klinger, Clemens Rosenbaum, Joseph P Bigus, Murray Campbell, Ban Kawas, Kartik Talamadupula, Gerry Tesauro, and Satinder Singh (2017). “Learning to Query, Reason, and Answer Questions On Ambiguous Texts”. In: *International Conference on Learning Representations (ICLR)*.

Henderson, James, Oliver Lemon, and Kallirroi Georgila (2008). “Hybrid Reinforcement/Supervised Learning of Dialogue Policies from Fixed Data Sets”. In: *Computational Linguistics*.

Henderson, Matthew, Blaise Thomson, and Steve Young (Aug. 2013). “Deep Neural Network Approach for the Dialog State Tracking Challenge”. In: *Proceedings of the 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Henderson, Matthew, Blaise Thomson, and Steve Young (2014). “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)*.

Hill, Felix, Antoine Bordes, Sumit Chopra, and Jason Weston (2015). “The goldilocks principle: Reading children’s books with explicit memory representations”. In: *International Conference on Learning Representations (ICLR)*.

Hixon, Ben, Peter Clark, and Hannaneh Hajishirzi (2015). “Learning knowledge graphs for question answering through conversational dialog”. In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation*.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). “Multilayer feedforward networks are universal approximators”. In: *Neural Networks*.

Jiang, Jing and ChengXiang Zhai (2007). “Instance Weighting for Domain Adaptation in NLP”. In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*.

Joshi, Chaitanya K., Fei Mi, and Boi Faltings (2017). “Personalization in Goal-Oriented Dialog”. In: *Workshop on Conversational AI, Advances in Neural Information Processing Systems (NeurIPS)*.

Kadlec, R., M. Schmid, and J. Kleindienst (2015). “Improved Deep Learning Baselines for Ubuntu Corpus Dialogs”. In: *Workshop on Machine Learning for SLU and Interaction, Advances in Neural Information Processing Systems (NeurIPS)*.

Kim, Seokhwan, Luis Fernando D’Haro, Rafael E Banchs, Jason D Williams, and Matthew Henderson (2017). “The fourth dialog state tracking challenge”. In: *Dialogues with Social Robots*.

Kumar, A., O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher (2016). “Ask Me Anything: Dynamic Memory Networks for Natural Language Processing”. In: *Proceedings of International Conference on Machine Learning (ICML)*.

LeCun, Yann, Corinna Cortes, and CJ Burges (2010). “MNIST handwritten digit database”. In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*.

Li, Jiwei, Alexander H. Miller, Sumit Chopra, Marc’Aurelio Ranzato, and Jason Weston (2017). “Dialogue Learning With Human-In-The-Loop”. In: *International Conference on Learning Representations (ICLR)*.

Li, Xiujun, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz (2017). “End-to-End Task-Completion Neural Dialogue Systems”. In: *Proceedings of the 8th International Joint Conference on Natural Language Processing (IJCNLP)*.

Liang, Chen, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao (2017). “Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL).*

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2016). “Continuous control with deep reinforcement learning.” In: *International Conference on Learning Representations (ICLR).*

Lin, Zhaojiang, Andrea Madotto, Genta Indra Winata, and Pascale Fung (2020). “MinTL: Minimalist Transfer Learning for Task-Oriented Dialogue Systems”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Liu, Bing and Ian Lane (2017). “Iterative policy learning in end-to-end trainable task-oriented neural dialog models”. In: *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU).*

Liu, Bing, Gokhan Tür, Dilek Hakkani-Tür, Pararth Shah, and Larry Heck (2018). “Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems”. In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL).*

Liu, Fei and Julien Perez (2017). “Gated End-to-End Memory Networks”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL).*

Lowe, Ryan, Nissan Pow, Iulian Serban, and Joelle Pineau (2015). “The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems”. In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL).*

Madotto, Andrea and Giuseppe Attardi (2017). “Question Dependent Recurrent Entity Network for Question Answering”. In: *arXiv preprint arXiv:1707.07922.*

Madotto, Andrea, Zhaojiang Lin, Chien-Sheng Wu, and Pascale Fung (July 2019). “Personalizing Dialogue Agents via Meta-Learning”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL).*

Manning, Christopher, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky (2014). “The Stanford CoreNLP natural language processing toolkit”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL).*

Mi, Fei, Minlie Huang, Jiyong Zhang, and Boi Faltings (2019). “Meta-Learning for Low-resource Natural Language Generation in Task-oriented Dialogue Systems”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence, (IJCAI).*

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013a). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013b). "Distributed representations of words and phrases and their compositionality". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (2015). "Human-level control through deep reinforcement learning". In: *Nature*.

Moon, Seungwhan and Jaime Carbonell (2017). "Completely Heterogeneous Transfer Learning with Attention - What And What Not To Transfer". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*.

Mrkšić, Nikola, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young (July 2015). "Multi-domain Dialog State Tracking using Recurrent Neural Networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*.

Neelakantan, Arvind, Quoc V Le, and Ilya Sutskever (2015). "Neural programmer: Inducing latent programs with gradient descent". In: *International Conference on Learning Representations (ICLR)*.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer (2018). "Deep Contextualized Word Representations". In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Qian, Kun and Zhou Yu (2019). "Domain Adaptive Dialog Generation via Meta Learning". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Rajendran, Janarthanan, Richard Lewis, Vivek Veeriah, Honglak Lee, and Satinder Singh (2020). "How Should an Agent Practice?" In: *Proceedings of 34th AAAI Conference on Artificial Intelligence (AAAI)*.

Ramadan, Osman, Paweł Budzianowski, and Milica Gasic (2018). "Large-Scale Multi-Domain Belief Tracking with Knowledge Sharing". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Sankaranarayanan, S., Y. Balaji, C. D. Castillo, and R. Chellappa (2018). "Generate to Adapt: Aligning Domains Using Generative Adversarial Networks". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Serban, Iulian Vlad, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau (2018). “A survey of available corpora for building data-driven dialogue systems”. In: *Dialogue and Discourse journal*.

Serban, Iulian Vlad, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau (2016). “Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models.” In: *Proceedings of 30th AAAI Conference on Artificial Intelligence (AAAI)*.

Shang, L., Z. Lu, and H. Li (2015). “Neural Responding Machine for Short-Text Conversation”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Sordoni, Alessandro, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie (2015). “A hierarchical recurrent encoder-decoder for generative context-aware query suggestion”. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM)*.

Su, P.-H., D. Vandyke, M. Gašić, D. Kim, N. Mrkšić, T.-H. Wen, and S. Young (2015). “Learning from Real Users: Rating Dialogue Success with Neural Networks for Reinforcement Learning in Spoken Dialogue Systems”. In: *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*.

Sukhbaatar, Sainbayar, arthur szlam arthur, Jason Weston, and Rob Fergus (2015). “End-To-End Memory Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Takanobu, Ryuichi, Hanlin Zhu, and Minlie Huang (2019). “Guided Dialog Policy Learning: Reward Estimation for Multi-Domain Task-Oriented Dialog”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Tran, V., V. Nguyen, and L. Nguyen (2017). “Enhanced semantic refinement gate for RNN-based neural language generator”. In: *Proceedings of the 9th International Conference on Knowledge and Systems Engineering (KSE)*.

Tur, Gokhan, Li Deng, Dilek Hakkani-Tür, and Xiaodong He (2012). “Towards Deeper Understanding Deep Convex Networks for Semantic Utterance Classification”. In: *IEEE International Confrence on Acoustics, Speech, and Signal Processing (ICASSP)*.

Vinyals, O. and Q. Le (2015). “A Neural Conversational Model”. In: *Workshop on Deep Learning, International Conference on Machine Learning (ICML)*.

Wang, Z., Z. Dai, B. Póczos, and J. Carbonell (2019). “Characterizing and Avoiding Negative Transfer”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ward, Wayne and Sunil Issar (1994). "Recent Improvements in the CMU Spoken Language Understanding System". In: *Proceedings of Workshop on Human Language Technology (HLT)*.

Wen, T.-H., M. Gasic, D. Kim, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young (2015). "Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking". In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Wen, Tsung-Hsien, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young (2016). "Multi-domain Neural Network Language Generation for Spoken Dialogue Systems". In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Wen, Tsung-Hsien, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young (2015). "Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Wen, Tsung-Hsien, David Vandyke, Nikola Mrkšić, Milica Gasic, Lina M. Rojas Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young (2017). "A Network-based End-to-End Trainable Task-oriented Dialogue System". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Weston, Jason, Sumit Chopra, and Antoine Bordes (2014). "Memory networks". In: *International Conference on Learning Representations (ICLR)*.

Wichrowska, Olga, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein (2017). "Learned Optimizers that Scale and Generalize". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Williams, Jason (Aug. 2013). "Multi-domain learning and generalization in dialog state tracking". In: *Proceedings of the 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Williams, Jason D, Kavosh Asadi, and Geoffrey Zweig (2017). "Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Williams, Jason D. and Geoffrey Zweig (2016). "End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning". In: *arXiv preprint arXiv:1606.01269*.

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning*.

Wu, Chien-Sheng, Richard Socher, and Caiming Xiong (2019). “Global-to-local Memory Pointer Networks for Task-Oriented Dialogue”. In: *International Conference on Learning Representations (ICLR).*

Wu, Yuhuai, Mengye Ren, Renjie Liao, and Roger Grosse. (2018). “Understanding Short-Horizon Bias in Stochastic Meta-Optimization”. In: *International Conference on Learning Representations (ICLR).*

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio (2015). “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *Proceedings of International Conference on Machine Learning (ICML).*

Xu, Yumo, Chenguang Zhu, Baolin Peng, and Michael Zeng (2020). “Meta Dialogue Policy Learning”. In: *arXiv preprint arXiv:2006.02588.*

Xu, Zhongwen, Hado P van Hasselt, and David Silver (2018). “Meta-Gradient Reinforcement Learning”. In: *Advances in Neural Information Processing Systems (NeurIPS).*

Yao, K., B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi (2014). “Spoken language understanding using long short-term memory neural networks”. In: *IEEE Workshop on Spoken Language Technology (SLT).*

Yih, Wen-tau, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao (2015). “Semantic parsing via staged query graph generation: Question answering with knowledge base”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL).*

Yin, Pengcheng, Zhengdong Lu, Hang Li, and Kao Ben (2016). “Neural Enquirer: Learning to Query Tables in Natural Language”. In: *Proceedings of the Workshop on Human-Computer Question Answering*.

Yin, Wenpeng, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze (2016). “Simple Question Answering by Attentive Convolutional Neural Network”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*.

Young, Steve, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu (Dec. 2009). “The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management”. In: *Computer Speech and Language*.

Yu, Mo, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou (2017). “Improved Neural Relation Detection for Knowledge Base Question Answering”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL).*

Zhao, Han, Shanghang Zhang, Guanhang Wu, José M. F. Moura, Joao P Costeira, and Geoffrey J Gordon (2018). “Adversarial Multiple Source Domain Adaptation”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhao, Tiancheng and Maxine Eskenazi (2016). “Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning”. In: *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.

Zhao, Xueliang, Wei Wu, Chongyang Tao, Can Xu, Dongyan Zhao, and Rui Yan (2020). “Low-Resource Knowledge-Grounded Dialogue Generation”. In: *International Conference on Learning Representations (ICLR)*.

Zheng, Zeyu, Junhyuk Oh, and Satinder Singh (2018). “On Learning Intrinsic Rewards for Policy Gradient Methods”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.