Knowledge Base Adaptation for Task Oriented Dialog Systems

Thesis submitted by

Nikhil Gupta
2014CS50462

under the guidance of

Prof. Mausam

in partial fulfilment of the requirements
for the award of the degree of

Bachelor and Master of Technology

June 2019

Department Of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

THESIS CERTIFICATE

This is to certify that the thesis titled **Knowledge Base Adaptation for Task Oriented Dialog**, submitted by **Nikhil Gupta**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor and Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Mausam
Professor
Dept. of Computer Science
IIT-Delhi, 600 036

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Mausam for the continuous support of my Master's study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Master's study. He consistently allowed me to frame my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to thank Dinesh Raghu for his immense support and guidance during the past two years. He brought structure to both my research and thinking process, allowing me to deep dive and grow into this field of study. I will cherish our late-night sprints during our paper submissions and our stimulating discussions regarding minute details of our algorithm or codebase. Dinesh consistantly brought out the best in me while giving me my own space to work and think, leading me to believe I have become a better researcher by the end of it.

I would also like to acknowledge my friend and labmate at IIT Delhi: Ayush Bhardwaj for being my right-hand man throughout all my course work and assignments, and I am gratefully indebted to his very valuable comments on this thesis. A heartful thanks to Aishwarya Sharma at IIT (BHU) Varanasi for lending her brilliant proofreading skills to my publications.

I thank Danish Contractor, Gaurav Pandey and Sachindra Joshi for their comments on my drafted publications. I extend gratitude to support my work received via IBM AI Horizons Network grant, an IBM SUR award, grants by Google, Bloomberg and 1MG, and a Visvesvaraya faculty award by Govt. of India. I thank Microsoft Azure sponsorships, and the IIT Delhi HPC facility for computational resources.

Last but not the least, I would like to thank my family: my parents for giving birth to me at the first place and supporting me spiritually throughout my life. They have taught me to always put my best effort in all my endevours and have consistently set this example for me to follow. I am grateful to them for cultivating my curiousity and pushing me to be a good human being along my journey.

ABSTRACT

Dialog systems or chatbots are computer programs that can interact with humans either using a speech interface or text interface. Based on the application, dialogs systems can be divided into two categories: open domain and task oriented. Dialog systems that converse with an intention to accomplish a task such as recommending a restaurant or booking a flight tickets are task oriented dialog systems. Such systems gennerally need to consult a Knowledge Base (KB) with stored information to accomplish the task. End-to-end neural networks trained for these task-oriented dialogs are expected to be immune to any changes in this KB which can evolve with time. However, existing approaches breakdown when asked to handle such changes. The failure is mostly due to the inability to handle Out-of-Vocabulary (OOV) words and the inability to perform simple reasoning over the knowledge base results such as suggest without repetition and sorting based on a field.

We studied the correlation between the learned language model and the knowledge base incorporation and conjectured that we must strive to create a system that learns both mutually exculsive of the other, i.e. in a disentangled manner. We propose an encoder-decoder architecture (BoSsNet) with a novel Bag-of-Sequences (BoSs) memory, which facilitates such disentangled learning. Consequently, the KB can be modified with new knowledge without a drop in interpretability. We find that BoSsNet outperforms state-of-the-art models, with considerable improvements (>10%) on bAbI OOV test sets and other humsan-human datasets. We also systematically create adversarial attacks that introduce modifications to the KB in an attempt to measure the extent of disentanglement and show that BoSsNet remains robust to all such attacks.

Generally the training set dialogs in task-oriented systems use an explicit API for KB retreival which must be manually annotated within the dialogs themselves. We attempt to go one step further by reducing the amount of annotations needed to train such task-oriented dialogs by predicting the correct API without it being explicitly mentioned. We generate this API using an RL-based decoder. The results from this API are then used to generate responses by our response decoder which serve as feedback (reward) to train the RL decoder. We created a novel architecture that would be capable to mitigte two critical problems: data bias (multiple APIs fetch similar KB results) and length bias (preference of the RL decoder to predict short length APIs over longer ones, despite worse results). Our system was successfully able to achieve similar performance on the unannotated dataset as compared to the fully API annotated dataset.

TABLE OF CONTENTS

© 2019, *Indian Institute of Technology Delhi*

LIST OF TABLES

© 2019, *Indian Institute of Technology Delhi*

LIST OF FIGURES

© 2019, *Indian Institute of Technology Delhi*

Chapter 1

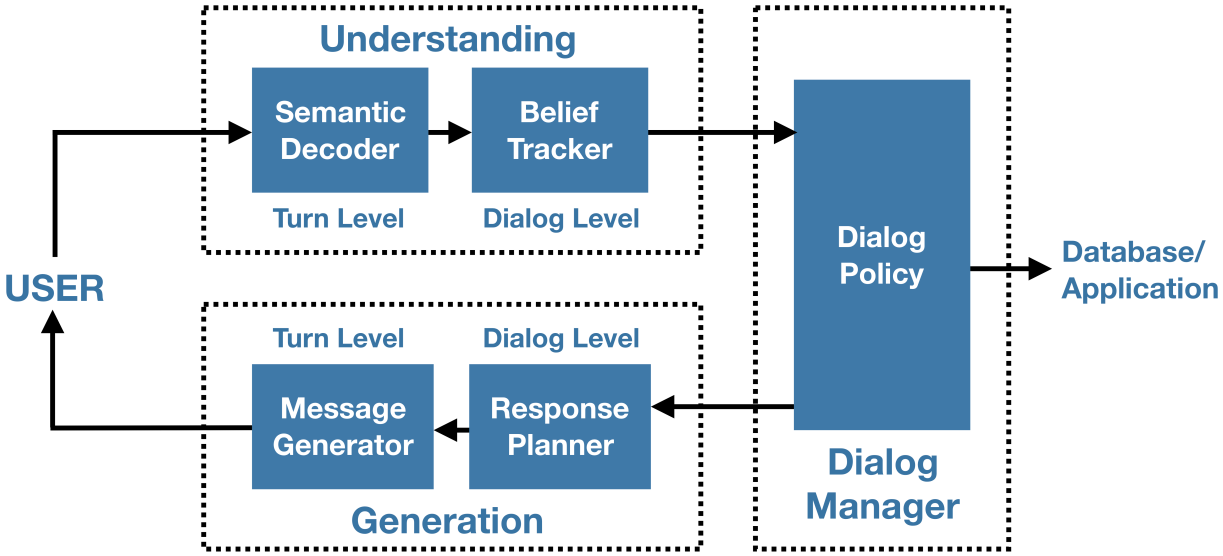
INTRODUCTION

1.1 Overview

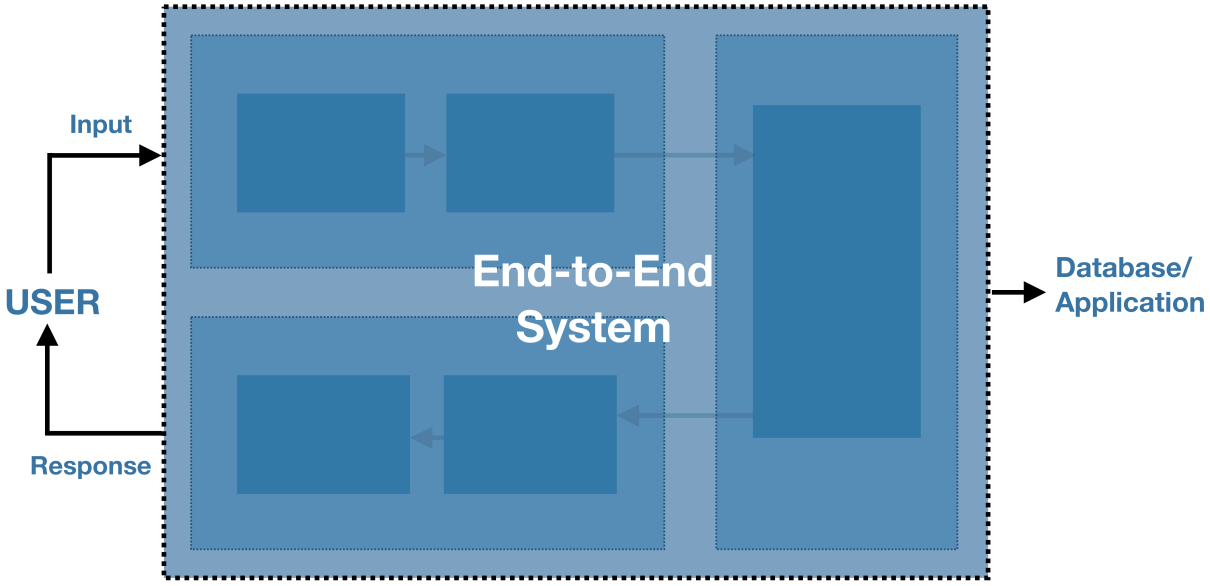
Dialog systems, also referred to as chatbots or virtual agents are systems that can converse with a human to help with their informational needs. Dialog systems are used in a wide range of applications such as personal assistants in mobile phones, technical support services, chit chat, product enquiries, IVR systems and entertainment. Some of the popular dialog systems include Apple's Siri, Google Now, Microsoft's Cortana, Amazon's Alexa and Google's Smart Reply. Most of the dialog systems that are being used for real word applications are hand crafted by a dialog designers and also very specific to a domain. Even though using such hand crafted rules provides the flexibility to build a interpretable dialog system, it requires a great amount of effort for the dialog designer to create one from scratch for a new domain. This extra effort also transcends to scenarios where the existing chatbot's capability has to be extended or improved.

End-to-End Learning: A complete dialog system is comprised of several sub-modules with different functionalities. Broadly we can classify these into three primary components: 1) natural language understanding (NLU) unit, 2) dialog manager and 3) natural language generator (NLG). The architecture is shown in Figure 1.1a. Each component can be either built using hand-crafted rules or can be a statistical machine learning model that learns from a provided set of examples. A complete dialog-system, which is a combination of these components, can then be partly hand-designed and partly data-driven. In some cases, the designer will choose to combine two or more components together because of the type of data that is avaliable. This bypasses the need for explicitly annotating the data to serve each component separately, but in exchange, the complexity of the module increases thereby demanding a lot more data to train. The approach where all the three components are combined together and examples of (user input in natural language, expected response in natural language) pairs are used to learn a dialog model, is referred to as *end-to-end* dialog models. Note that in end-to-end learning, the intermediate output space may be defined, but examples are not annotated with expected intermediate outputs. This architecture is depicted in Figure 1.1b.

Task-Oriented: Application of dialog systems can be broadly categorized into two types: open domain (non-task-oriented) dialog systems and task-oriented dialog systems. The main difference between the two approaches is that, the objective of the former is usually broad



(a) Multi-Component Dialog System



(b) End-to-end Dialog System

Figure 1.1: Types of Dialog System Architecture

© 2019, *Indian Institute of Technology Delhi*

and not well defined (abstract), where as the objective of the latter is narrow and well-defined. For example, the restaurant reservation domain dialogs falls under task-oriented systems. The goal could be, given a set of options (cuisine: *Chinese*, location: *East Delhi*), the system should be able to suggest the right restaurant that is acceptable by the user. A large portion of the research in data driven approaches for dialog modeling has been around open domain systems. Some examples include chit-chat and language learning. Applications such as restaurant reservation, flight booking, travel enquiry and bus enquiry belong to the task-oriented setting. One advantage of working with task-oriented dialogs is the ease of defining evaluation techniques to measure the performance of the system. The performance of the system can be measured by the percentage of conversations where the dialog system was able to help the user achieve their goal.

Question answering systems and task-oriented dialog systems are very similar to each other. The major difference between the two is that, in question answering all details necessary to achieve the task are provided in a single shot, where as in task-oriented dialog, the user conveys information spaced throughout the converstation. In the latter it is the job of the dialog system to collect the necessary information to complete the task. These architectures also provide the ability to carry over context when switching between tasks.

Knowledge Base in the Loop: These are the subset of dialog systems that require access to a knowledge base to respond to user input. Some examples of knowledge bases systems are: a bus inquiry system using a KB with bus running status, a restaurant reservation system using a KB filled with restaurant information, etc. There has been a large amount of work on using databases in systems that use hand-crafted features for modeling dialog. The community has recently (in 2017) started to explore the area of learning data-driven dialog models grounded by a knowledge base.

We interact with the Knowledge Base using specially designed APIs that are task specific. These APIs will have the ability to either retrieve information or update information in the KB. Traditionally, API calls have been simple template based responses that employ slot-filling techniques for each template entry. However, the retrieval power could be increased if the APIs would become more flexible and dynamic. This would enable the selection of more precisise results and even allow simple data manipulations, such as sorting, which could aid in training the system.

Typically, to train dialog systems associated with a knowledge base, we need to annotate the knowledge base incorporation in the dialog data itself. For example, if the system is to query the KB, then the corresponding API call must be included in the dialog explicitly. This makes it difficult to create a large training corpus as general ASR systems won't be able to recognise the API call being performed and the data curator must add it manually.

© 2019, *Indian Institute of Technology Delhi*

1.2 Problem Definition

The focus of this thesis will be to improve the ability of task-oriented dialog systems to generalise over its KB. In real-world applications, the KB information could change over time. For example, (1) a KB associated with a movie ticket booking system gets updated every week based on new film releases, and (2) a restaurant reservation agent, trained with the knowledge of eateries in one city, may be deployed in other cities with an entirely different range of establishments. In such situations, the system should have the ability to conform to new-found knowledge unseen during its training. Ideally, the training algorithm must learn to disentangle the language model from the knowledge interface model. This separation will enable the system to generalize to KB modifications, without a loss in performance.

Moreover, for achieving good progress towards the user's task, the agent must also retain the ability to draw inferences based on past utterances and the KB. Notably, we find that existing approaches either achieve this disentanglement or effective progress towards the task, but not both.

We will also focus on enabling the system to predict the correct API call without explicit annotations. This will help us twofold: (1) we will be able to leverage larger databanks for training end-to-end dialog systems which don't have this additional KB annotation, and (2) we will be able to predict more efficient APIs to solve the task than those originally used.

To summarize, the goal of our research is to build a dialog system that takes as input (1) a large set of task-oriented conversation logs between a user and an agent/domain expert, (2) a knowledge base used to generate agent response and learn a dialog system that can mimic the agent to accomplish a user's task. With this goal in mind we can broadly formulate two distinct problems as follows:

Problem 1 *Create an end-to-end dialog system that effectively disentangles the language and knowledge models.*

The system must be capable to handle changes in the database which we will simulate using systematic adversarial attacks which mutates KB information. We will test the effectiveness of the system on three standard datasets used for task-oriented dialog – bAbI [4], CamRest [31], and Stanford Multi-Domain Dataset [6]. Of these, the last two are real-world datasets. We show a sample dialog and its components in Figure **??**.

Problem 2 *Enable the end-to-end system to achieve comparable performance on datasets without explicit Knowledge Base API annotations.*

The mentions datasets will have to be augmented to remove the extra annotation depicting the API information and a *"predict API"* signal should be inserted in its exact place. The system will be evaluated using the same metrics as before and the objective will be to acheive the same performance as the model trained in the first problem definition.

© 2019, *Indian Institute of Technology Delhi*

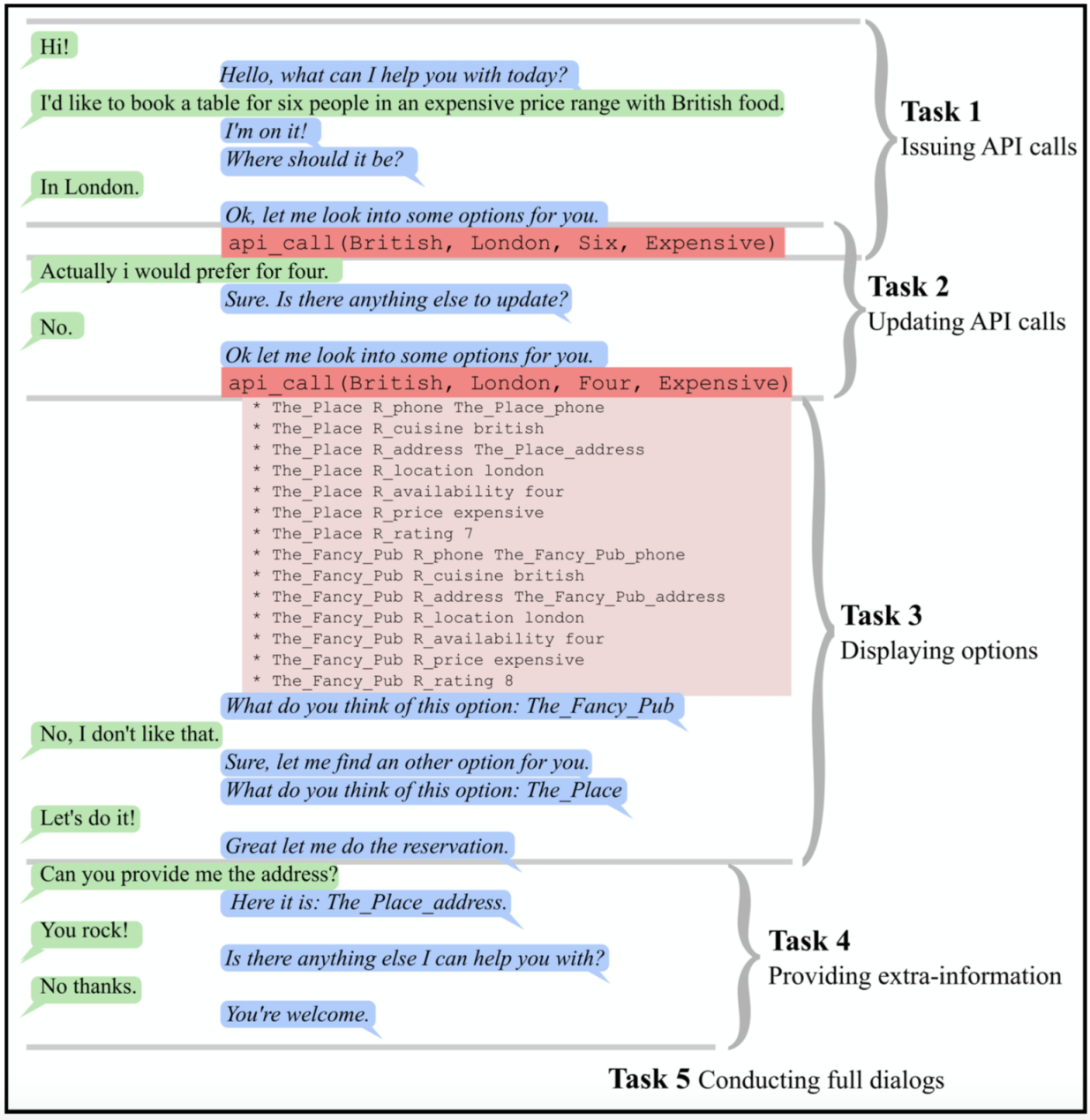


Figure 1.2: A breakdown of tasks in bAbI dialog. User queries are in green, system responses in blue, APIs in red along with their fetched KB results. Figure credits [3]

© 2019, *Indian Institute of Technology Delhi*

1.3 Motivation

In this section, we motivate our problem by clearly defining the gap between what the state-of-the-art, data-driven approaches for dialog learning are capable of and what it takes to build a usable end-to-end task-oriented dialog system that requires a knowledge base.

Why cant already proven non-task-oriented models be used?

While end-to-end non-task-oriented (open domain) dialog systems are being used in real world applications [10], designers still prefer using hand crafted rules for modeling task-oriented dialogs. The main reason for success in non-task-oriented setting is that, models have largely been evaluated (and applied) in scenarios where, given the conversation so far the system is expected to predict just the immediate next response. Such modeling requires the system to learn a language model and a mapping from the context to the response. But in case of task-oriented settings, the requirements/expectations are much more. It is not just required to generate the next utterance based on the context, but understand the global picture of what the task is, what all details are necessary to finish the task, what is the optimal way or strategy to request for missing details, and then decide what the next utterance should be. Also, in chit chat bot (non-task-oriented) making a mistake in a single turn is not very costly, where as in flight booking system (task-oriented) making a single mistake (mis-interpreting a source city) could turn out to be very costly. Even though a large section of real world applications such flight booking, technical service support, basic medical consultations fall under this category, there has been very little focus so far.

In addition to task-oriented setting, when a knowledge base is added into the loop, the system should now also be able to learn when to query a knowledge base, how to incorporate the results from the knowledge base into the conversation. Also, such KB-centric task-oriented dialogs generally need specially annotated data which is not easily available as compared to unannotated dialogs which used to train open domain systems. This makes the problem much harder and harder to generalise. Hence models that have been proven to work on non-task-oriented systems cannot be used for task-oriented applications.

How is dialog data collected for data-driven modelling?

Most datasets used in open domain dialog systems are obtained using Automatic Speech Recognistion (ASR) techniques. Here a recording of a conversation between two individuals can easily be transcribed and added to the large data corpora. This ease of data collection and large datasets has invited a lot of research interest in open domain dialog. Task-oriented dialog requires specific annotation for it to be usable, such as interaction with the Knowledge Base via an API and the results themselves. Difficultly to annotate such instances automatically has led to scarce data reserves and has forced researchers to fabricate data of their own.

© 2019, *Indian Institute of Technology Delhi*

Limitations of state-of-the-art task-oriented approaches

There has been very little progress made end-to-end task-oriented dialogs, as most of the proposed approaches has been around using hand crafted rules to either build the entire system, or to define the structured space of each sub-components output. The works published so far on end-to-end task-oriented conversation have the following limitations:

1. Both the human and the agent utterances in the dataset are fabricated using rules. This simulated dataset is very simple compared to human to human conversation logs.
2. The learned model is unable to generalise over a Knowledge Base and performance suffers greatly whenever KB modifications are made. It has no support for field values that have never been encountered so far (OOVs).
3. The system performance is has acceptable accuracy on non-task-oriented evaluation metrics, but the performance is very poor on task-oriented evaluation metrics
4. The system is unable to learn simple patterns (that are exhibited in the train corpora) required to perform a task-oriented dialog, such as not repeating already suggested options and sorting based on a particular field.
5. The system proposed has a very limited view of the knowledge base. It assumes that it can only perform simple *SELECT* operation with *WHERE* clauses over the knowledge base

Why is gradient based policy optimization difficult?

Simple policy gradient methods like REINFORCE [34] use Monte Carlo samples from the current policy to perform an on-policy optimization of the expected return objective. This often leads to unstable learning dynamics and poor sample efficiency, sometimes even underperforming random search [19]. Learning through reflection on past experiences ("experience replay") is a promising direction to improve data efficiency and learning stability. MAPO [15] attempts to incorporate a memory buffer of promising trajectories within the policy gradient framework. It expresses the expected return objective as a weighted sum of an expectation over the trajectories inside the memory buffer and a separate expectation over unknown trajectories outside of the buffer.

But MAPO suffers from three main disadvantages:

Length Bias is the phonomenon of a policy to favor a shorter trajectory as compared to a longer one with equal or even greater rewards. This makes it hard to train the policy for longer and more complex queries.

Data Bias is an intrinsic property of the knowledge base on the data it contains. It refers to the situation when multiple APIs retrieve the same set of results from the KB for some inputs and behave very differently for a different set of inputs. This confuses the policy which is not able to learn the proper behavior of the API.

© 2019, *Indian Institute of Technology Delhi*

Language Bias is the due to the frequennt occurance of some inputs to the API which biases the policy to predict them. For example, *dontcare*, or null, is seen very often in the dataset as it is the default input. The policy would favor prediciting such null inputs even if they result to a low reward.

1.4 Contributions

To address our first problem statement, we propose BoSsNet, a novel network that effectively disentangles the language and knowledge models, and also achieves state-of-the-art performance on three existing datasets.

To achieve this, BoSsNet makes three design choices. First, BoSsNet uses an encoder-decoder architecture so that it can stitch the next utterance one word at a time as compared to a typical retrieval based model which selects a certain response from a fixed set of responses. Second, it encodes the conversational input as a *bag of sequences* (BoSs) memory, in which the input representation is built at two levels of abstraction. The *higher level* flat memory encodes the KB tuples and utterances to facilitate effective inferencing over them. The *lower level* encoding of each individual utterance and tuple is constructed via a sequence encoder (Bi-GRU). This enables the model to maintain the sequential context surrounding each token, aiding in better interpretation of unseen tokens at test time. Third, we augment the standard cross-entropy loss used in dialog systems with an additional loss term to encourage the model to only copy KB tokens in a response, instead of generating them via the language model. This combination of sequence encoding and additional loss (along with dropout) helps in effective disentangling between language and knowledge.

We find that BoSsNet is competitive or significantly better on standard metrics in all datasets as compared to state-of-the-art baselines. We also introduce *adversarial attacks* in which we systematically increase the percentage of previously unseen entities in the KB. We summarise these attacks as our *knowledge adaptability* (KA) evaluation. We find that BoSsNet is highly robust across all percentage levels in all the standard metrics. We also report a human-based evaluation and find that BoSsNet responses are frequently rated higher than other baselines.

To address the second problem statement, we extend BoSsNet and add a novel RL-decoder which generates the required API call at a fixed turn of the dialog. We train the system using a buffer containing valid API calls and do a mix of off-policy and on-policy sampling. We use a pseudo-reward for learning which is computed using a combination of precision and recall on the retrieved results and the gold system response KB entities. Our extended architecture which we term as BoSsNet-RL was successfully able to achieve similar accuracies to the original BoSsNet model trained on the fully annotated dataset.

© 2019, *Indian Institute of Technology Delhi*

Overall, our contributions are:

1. We propose BoSsNet, a novel architecture to disentangle the language model from knowledge incorporation in task-oriented dialogs.
2. We introduce a *knowledge adaptability* evaluation to measure the ability of dialog systems to scale performance to unseen KB entities.
3. Our experiments show that BoSsNet is competitive or significantly better, measured via standard metrics, than the existing baselines on three datasets.
4. We extend the BoSsNet architecture and add a novel RL-based API generator which matches state-of-the-art performance on modified datasets which lack explicit API annotation.

Our work addressing the first problem statement was published as a full paper in the Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019). `https://www.aclweb.org/anthology/papers/N/N19/N19-1126/`.

We release our code and *knowledge adaptability* (KA) test sets for further use by the research community. `https://github.com/dair-iitd/BossNet`. We also release the BoSsNet-RL code base along with the modified test sets. `https://github.com/NikhilGupta1997/BossNet-RL`.

© 2019, *Indian Institute of Technology Delhi*

Chapter 2

BACKGROUND

2.1 Related Work

Dialog systems can be broadly divided into two categories: open domain [29, 25] and task-oriented dialog systems. Open domain systems generate responses based on just the dialog history, whereas the task-oriented systems generate responses based on the dialog history and a KB associated with the task.

Task oriented dialogs systems can be further divided into two: modular and end-to-end trainable dialog systems. Modular dialog systems [30, 32, 33] require intermediate supervision on dialog transcripts to train each of its modules. Our work falls under end-to-end trainable dialog systems, which requires just the dialog transcripts and no intermediate supervision to train. We discuss end-to-end trainable neural models along two dimensions: (1) decoding: how the response is retrieved or generated and (2) encoding: how the dialog context (dialog history and KB tuples) are represented.

Most of the existing approaches [4, 16, 24, 35] *retrieve* a response from a pre-defined set. These methods are generally successful when they have to provide boilerplate responses – they cannot construct new responses or use words not seen during training. To improve upon these, generative approaches are used where the next response is *generated* one word at a time [7, 18]. These approaches also benefit the OOV problem by incorporating the ability to copy words from the input [28, 8]. This copy mechanism has also found success in summarization [20, 23] and machine translation [9]. BoSsNet is a copy incorporated generative approach.

For encoding, some approaches represent the dialog history as a sequence [7, 9]. Unfortunately, using a single long sequence for encoding also enforces an order over the set of KB tuples making it harder to perform inferencing over them. Other approaches represent the dialog context as a bag. Original Memory Networks [4] and its extensions encode each memory element (utterance) as an average of all constituent words – this cannot point to individual words, and hence cannot be used with a copy mechanism. Mem2Seq encodes each word individually in a flat memory. Unfortunately, this loses the contextual information around a word, which is needed to decipher an unseen word. In contrast, BoSsNet uses a bag of sequences encoding, where KB tuples are a set for easier inference, and also each utterance is a sequence for effectively learning when to copy. We go into further detail below highlighting the differences between the two models.

2.1.1 Comparison with Mem2Seq

The closest to our model is the recently introduced Mem2Seq [18], which also combines the multi-hop reasoning of memory networks with the generative decoding augmented with a copy mechanism. While similar in spirit to BoSsNet, there are significant differences in the two architectures. We illustrate these with Figure 2.1, which shows an example dialog along with an associated KB in both Mem2Seq and BoSsNet memories.

First, the memory in Mem2Seq is a flat representation which operates in part at utterance level and in part at word level. In particular, the individual words of a user/machine utterance are stored in their own memory cells, whereas a KB tuple gets only one memory cell – the aggregation is done in a bag of words fashion, using words in the entire tuple. For example, the memory element *m1* contains a full KB tuple and *m15* contains only the word "french" from the second user utterance.

This implies that Mem2Seq's copy mechanism can only point to a specific KB tuple, but cannot point to a specific constituent of the tuple. This forces Mem2Seq to make a second modeling choice, of allowing only the "object" entity of a tuple to be used in the next response. These choices result in three limitations of Mem2Seq, regarding the scenarios in which Mem2Seq can be useful.

First, Mem2Seq expects each tuple to be serialized by the KB API in way that the object always comes last. This may or may not always be in control of the neural model. Second, because each utterance word has a unique memory cell. it loses the context in which the word was mentioned. As a result, the model cannot learn that "french" and other cuisines are usually followed by the word "food". This causes poor generalization where the model has to understand an OOV word from the user utterance (e.g., a new cuisine or city name).

Finally, the model design dictates that the subject or the predicate from a KB fact can never be copied during the decode step. This adds severe limitations in practical settings. For example, the KB results in the original bAbI dataset [4] came in format of *attribute(restaurant_name, restaurant_attribute)*. Since restaurant name is never an object, Mem2Seq would never be able to copy it when making recommendations. As a result, experiments with Mem2Seq necessitated a training data preprocessing, in which all rating facts had to be inverted into *rating(restaurant_rating, restaurant_name)*, so that the name could become an object and could be copied.

We believe that this adds severe limitations on the scenarios in which Mem2Seq is applicable. We perform several experiments on the original (unprocessed) datasets to highlight the loss in performance due to these modeling choices. In contrast, BoSsNet uses a hierarchical memory that can copy *any* word from the whole dialog history (see Figure 2.1(c)). It models each cell as a *sequence* of words, which also enables it to learn contextual cues for each word, leading to better OOV generalization.

© 2019, *Indian Institute of Technology Delhi*

Restaurant	Cuisine	Rating	Phone
amour_bistro	french	5	ab_phone
le_cirque	french	3	lc_phone

Dialog Turn	Speaker	Utterance / KB Tuple
1	usr	hi
1	sys	hello
2	usr	can you book a table for six with french food

(a) Example KB and dialog history

m1	m2	m3	m4	m5		m15	m16	m17
amour_bistro r_phone ab_phone	5 r_rating amour_bistro	le_cirque r_phone lc_phone	3 r_rating le_cirque	t1 $u hi	…	t2 $u french	t2 $u food	sentinel

(b) Mem2Seq memory layout

m1	amour_bistro		r_phone		ab_phone	
m2	amour_bistro		r_rating		5	
m3	le_cirque		r_phone		lc_phone	
m4	le_cirque		r_rating		3	
m5	hi		$u		t1	
m6	hello		$s		t1	
m7	can	…	french	food	$u	t2

(c) BoSsNet memory layout

Figure 2.1: (a) an example dialog with history and KB tuples. (b) an illustration of Mem2Seq memory. (c) an illustration of BoSsNet memory

2.2 Components Survey

BoSsNet architecture has a *multi-hop encoder*, and a *pointer network* decoder with a *hierarchical attention* over the memory. The BoSsNet-RL architecture has an *RL-based API decoder* with mixed on-policy and off-policy sampling. We briefly survey these strands of related research.

Multi-hop Networks reason over a sequence of sentences fed as input. A hop refers to reading the sentences and generating a encoded-vector. Multi-hop refers to making multiple updates to the encoded-vector by iteratively reading the input. End to end memory network (MN) [26] represents the input as a set of sentences. Here the encoded-vector is updated by adding iterative reads. Query reduction network [24] reads the sentences sequentially using an RNN like unit called the QRN unit. Dynamic memory network [12] also reads the sentences sequentially, and also updates the encoded-vector using a recurrent cell. Gated memory network [16] uses a gating mechanism to update the encoded-vector.

© 2019, *Indian Institute of Technology Delhi*

MN [4], gated MN and QRN have been used to learn task-oriented dialogues. BoSsNet has two key differences from such architectures. First, existing models select a response from a predefined list of candidates (retrieval model), whereas BoSsNet has a decoder that generates the response one word at a time. Second, the memory in BoSsNet is hierarchical, i.e., each memory element is a sequence of words vectors rather than just a single utterance vector. This enables the generator to copy any word from the memory during generation.

Pointer Networks are sequence to sequence (Seq2Seq) models, where each token in the output sequence corresponds to a token at a certain position in the input sequence [28]. By enabling pointing in Seq2Seq models [5, 27], the effective decode vocabulary becomes the union of the fixed decode vocabulary and the vocabulary of the input sequence. Two main methods [8, 7] exist for incorporating pointing in standard Seq2Seq models – hard decision [20, 8, 7] and soft switch [23]. The former makes a hard choice between using the pointer distribution and the decode vocabulary distribution. It usually requires the hard decision to be labeled. The latter approach learns a soft interpolation between the two distributions without explicit labels. BoSsNet employs a soft switch in its decoder.

Eric and Manning [7] use a copy augmented Seq2seq model for learning task-oriented dialogues. This approach uses a hard decision to pick between the generate and pointer distributions. This model is explicitly trained to only point to words that are from the KB and generate the rest. This is the closest work to our approach, but has a flat memory and doesn't incorporate multi-hop reasoning.

Hierarchical Attention was first introduced for document classification [36]. Here, each document is represented as a set of sentences and each sentence as a set of words. For each sentence, an attention distribution is computed over words to identify informative words and compute a sentence representation. A similar approach identifies informative sentences to compute a document representation. Hierarchical attention has also been used for abstractive text summarization [20]. BoSsNet similarly computes two attention distributions over different levels of the input. A word-level distribution over the words in each utterance and an utterance-level distribution over all the input utterances. This a function of these two distributions is used when copying a word in the decode process.

Reinforcement Learning has been coupled with Seq2Seq architechures to solve tasks like program induction [14, 37], SQL query generation [15, 38], and dialog generation [13]. These systems generate output responses which are associated with a final reward. The REINFORCE [34] algorithm is used to train such systems from weak supervision and directly maximize the expected reward. Since learning from scratch is difficult for REINFORCE, it is augmented with an iterative maximum likelihood (ML) training process [14]. Memory Augmented Policy Optimization (MAPO) [15], leverages a memory buffer of promising trajectories to reduce the variance of policy gradient estimate. It uses distributed sampling from inside and outside of the memory buffer to scale up training.

© 2019, *Indian Institute of Technology Delhi*

BoSsNet-RL leverages these techniques to generate a valid API call to the Knowledge Base. However, it differs from the other methods as the decoder has to train using implicit reward which depends on the final system responses based on the KB results queried by the predicted API.

2.3 Preliminaries

In this section, we briefly describe the preliminaries over which the proposed BoSsNet architecture is built upon as explained in the previous section. This includes (1) the multi-hop encoder in MN and (2) a standard sequence decoder with attention [1]

Multi-Hop Encoder in MN

The multi-hop encoder as described in end-to-end memory network [26] takes as input a query $q \in \mathbb{R}^d$ and a memory $M = \{m_i; m_i \in \mathbb{R}^d\}$ and generates a reduced query $q_r \in \mathbb{R}^d$. Here d is the embedding dimension. Augmenting the query by attending it over the memory elements, to capture relevant information necessary to generate the response, is referred to as a *hop*. A single hop reduced query is computed as follows:

$$p_i = \text{softmax}(q^T m_i) \tag{2.1}$$

$$o = W_r \sum\nolimits_i p_i m_i \tag{2.2}$$

$$q_r = o + q \tag{2.3}$$

where $W_r \in \mathbb{R}^{d \times d}$. The hop step can be re-iterated, by assigning the output of the previous hop as the new input query (i.e.,) $q = q_r$. If the encoder has K hops, then the final output is represented as q_r^k. The multiple hops enable inference over multiple memory elements.

Sequence Decoder with Attention

The sequence decoder predicts the token y_t in the output sequence $\langle y_1 y_2 \dots y_T \rangle$, given the decoder state at time t, s_t, and a set of input contexts $Z = \{z_i; z_i \in \mathbb{R}^d\}$. For simplicity, we denote this conditional distribution of generating the next word as just $P_g(y_t)$. To compute $P_g(y_t)$, first an attention distribution α^t is computed over the input contexts z_i using Loung attention [17].

$$u_i^t = s_t W_l z_i \tag{2.4}$$

$$\alpha_i^t = \frac{exp(u_i^t)}{\sum_k exp(z_k^t)} \tag{2.5}$$

where $W_l \in \mathbb{R}^{d \times d}$. Then, a context vector z_t^* is generated by performing a weighed sum of the input contexts z_i using the attention distribution α_i^t. The context vector concatenated with the decoder state s_t is then used to compute the *generate distribution* over the decode

© 2019, *Indian Institute of Technology Delhi*

vocabulary $\mathcal{V}$ at time t as follows:

$$P_g(y_t) = \text{softmax}(W_d[s_t; z_t^*] + b) \tag{2.6}$$

where $W_d \in \mathbb{R}^{|\mathcal{V}| \times 2d}$ and $b \in \mathbb{R}^{|\mathcal{V}|}$ are parameters to be learnt. $[;]$ indicates vector concatenation along the row.

During training, the objective is to minimize the average negative log-likelihood for all the words in the response. The total loss is computed by adding the loss of all the responses in the training data.

Augmented REINFORCE

Lets consider the setting where given a query x, the state, action and reward at each time step $t \in 0, 1, \ldots, T$ are (s_t, a_t, r_t). In a deterministic environment the state is defined by the query x and the action sequence: $s_t = (x, a_{0:t-1})$, where $a_{0:t-1} = (a_0, \ldots, a_{t-1})$ is the history of actions at time t. If the reward at time t is given by r_t, then the cumulative reward of a action sequence $a_{0:T}$ is defined as:

$$R(x, a_{0:T}) = \sum_t r_t \tag{2.7}$$

The agent's decision making procedure at each time is defined by a deterministic policy, $\pi_\theta(s, a) = P_\theta(a_t = a|x, a_{0:t-1})$, where θ are the model parameters. The probability of generating a sequence $a_{0:T}$ is:

$$P_\theta(a_{0:T}|x) = \prod_t P_\theta(a_t|x, a_{0:t-1}) \tag{2.8}$$

We can define our objective to be the expected cumulative reward and use policy gradient methods such as REINFORCE for training. The objective and gradient are:

$$J^{RL}(\theta) = \sum_x \mathbb{E}_{P_\theta(a_{0:T}|x)}[R(x, a_{0:T})] \tag{2.9}$$

$$\nabla_\theta J^{RL}(\theta) = \sum_x \sum_{a_{0:T}} P_\theta(a_{0:T}|x) \cdot [R(x, a_{0:T}) - B(x)] \cdot \nabla_\theta \log P_\theta(a_{0:T}|x) \tag{2.10}$$

where $B(x) = \sum_{a_{0:T}} P_\theta(a_{0:T}|x) R(x, a_{0:T})$ is a baseline that reduces the variance of the gradient estimation without introducing bias.

Following a similiar pattern to imitaion learning [22, 2] we can calculate a $a_{0:T}^{best}$ based on maximum likelihood and add them to the set of action sequences with a resonably large probability. We define a parameter α and with each iteration, we scale the probabilities of the on-policy action sequences by $(1-\alpha)$ and then add the calculated $a_{0:T}^{best}$ with a probability α. This augmented REINFORCE leads to faster convergence and stabilises training.

© 2019, *Indian Institute of Technology Delhi*

Chapter 3

APPROACH

In this chapter, we first describe the BoSs memory which contains the dialog history and KB tuples, followed by how the memory is consumed by the encoder and the decoder. We then define the loss function, which, along with dropout, enables disentangled learning of language and knowledge. Finally we go on to explain the RL-decoder and its integration in the BoSsNet-RL architecture.

3.1 The BoSsNet Architecture

The proposed Bag-of-Sequences Memory Network has an encoder-decoder architecture that takes as input (1) dialog history, which includes a sequence of previous user utterances $\{c_1^u, \ldots, c_n^u\}$ and system responses $\{c_1^s, \ldots, c_{n-1}^s\}$, and (2) KB tuples $\{kb_1, \ldots, kb_N\}$. The network then generates the next system response $c_n^s = \langle y_1 y_2 \ldots y_T \rangle$ word-by-word. The simplified architecture of BoSsNet is shown in Figure 3.1 and a more detailed architecture is shown in 3.4.

3.1.1 Bag-of-Sequences Memory

The context memory M contains the dialog history $\{c_1^u, c_1^s, \ldots, c_{n-1}^u, c_{n-1}^s\}$ and the KB tuples $\{kb_1, \ldots, kb_N\}$. Each utterance in the dialog history and each KB tuple is placed in a memory cell as seen in Figure 3.2b. As utterances and tuples are inherently a sequence, we represent each memory cell m_i as an ordered sequence of tokens $\langle w_i^1 w_i^2 \ldots w_i^{|m_i|} \rangle$. For an utterance, the word tokens are followed by a temporal indicator and a speaker indicator {$u (for user), $s (for system)}. For example, {`good, morning, #1, $s`} indicates this was the first utterance by the system. For a KB tuple, the tokens are sequenced as {*subject, predicate, object*} followed by temporal indicator and a kb indicator (`$db`).

Token representation is generated using a bidirectional GRU. Let the outputs of the forward and backward GRUs for the token w_i^j be denoted as $\overrightarrow{h_i^j}$ and $\overleftarrow{h_i^j}$ respectively. Then the token representation $\phi(w_i^j)$ is given by Eq. 3.1. Memory cell representation $\psi(m_i)$ is computed by concatenating the forward GRU output of its last token and the backward

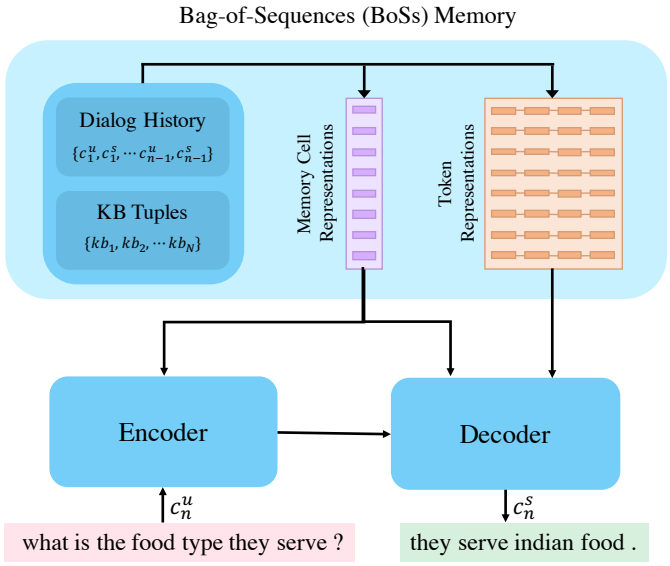


Figure 3.1: The dialog history and KB tuples stored in the memory have memory cell representations and token representations. The encoder understands the last user utterance using only the memory cell representations. The decoder generates the next response using both representations.

GRU output of its first token as in Eq. 3.2. This is further visualised in Figure 3.3

$$\phi(w_i^j) = [\overrightarrow{h_i^j}; \overleftarrow{h_i^j}] \tag{3.1}$$

$$\psi(m_i) = [\overrightarrow{h_i^{|m_i|}}; \overleftarrow{h_i^1}] \tag{3.2}$$

3.1.2 The BoSsNet Encoder

The encoder used in BoSsNet is similar to the multi-hop attention encoder with layer-wise weights proposed by [26]. The encoder in [26] uses two different embedding matrices, whereas we use just one to reduce the number of parameters. The encoder considers the last user utterance as the query $q = \psi(c_n^u)$ and computes the reduced representation q_r using the memory M as follows:

$$p_i = \text{softmax}(q^T \psi(m_i)) \tag{3.3}$$

$$o = W_r \sum\nolimits_i p_i \psi(m_i) \tag{3.4}$$

$$q_r = o + W_o q \tag{3.5}$$

where $W_r, W_o \in \mathbb{R}^{d \times d}$ are learnable parameters. The hop step can be re-iterated, by assigning the output of the previous hop as the new input query, i.e., setting $q = q_r$. The output of the encoder after K hops, q_r^k, is assigned as the initial state of the decoder.

© 2019, *Indian Institute of Technology Delhi*

db	resto_bombay_cheap_spanish_8stars R_location bombay
db	resto_bombay_cheap_spanish_8stars R_price cheap
db	resto_bombay_cheap_spanish_8stars R_rating 8
db	resto_bombay_cheap_spanish_8stars R_cuisine spanish
db	resto_bombay_cheap_spanish_6stars R_location bombay
db	resto_bombay_cheap_spanish_6stars R_price cheap
db	resto_bombay_cheap_spanish_6stars R_rating 6
db	resto_bombay_cheap_spanish_6stars R_cuisine spanish
user	good morning
system	hello what can i help you with today
user	can you book a table in bombay with spanish cuisine in a cheap price range
system	what do you think of this option : resto_bombay_cheap_spanish_8stars

(a) Simple Memory Representation (Sentence Level)

resto_bombay_cheap_spanish_8stars					R_location			bombay	
resto_bombay_cheap_spanish_8stars					R_price			cheap	
resto_bombay_cheap_spanish_8stars					R_rating			8	
resto_bombay_cheap_spanish_8stars					R_cuisine			spanish	
resto_bombay_cheap_spanish_6stars					R_location			bombay	
resto_bombay_cheap_spanish_6stars					R_price			cheap	
resto_bombay_cheap_spanish_6stars					R_rating			6	
resto_bombay_cheap_spanish_6stars					R_cuisine			bombay	
good					morning				
hello	what	can	i		help	you	with	today	
...	bombay	with	spanish	cuisine	in	a	cheap	price	range
what	do	you	think	of	:	resto_bombay_cheap_spanish_8stars			

(b) BoSs Memory Representation (Sentence + Word Level)

Figure 3.2: Types of Memory Architectures

© 2019, *Indian Institute of Technology Delhi*

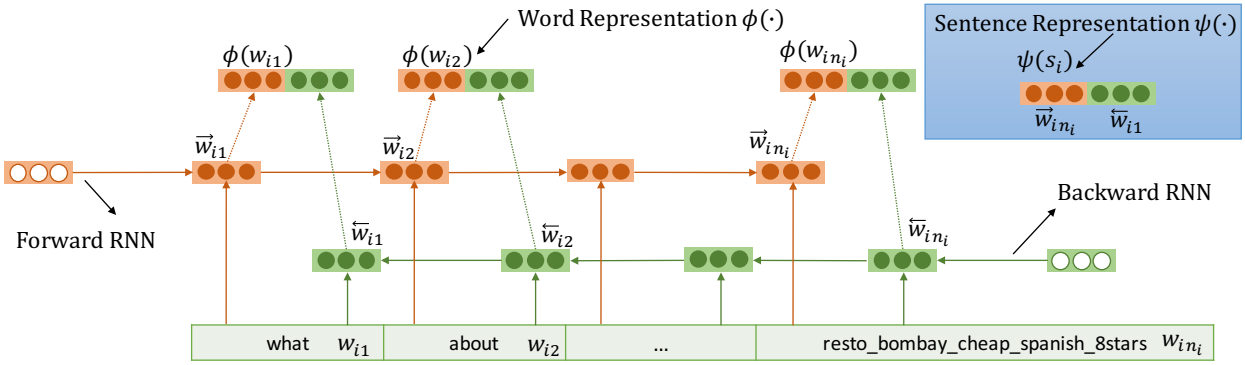


Figure 3.3: Each sentence is passed through a bi-directional GRU. We obtain word level embeddings $\phi(w_i)$ and sentence level embeddings $\psi(s_i)$ by concatinating the result of the front and back passes.

3.1.3 The BoSsNet Decoder

BoSsNet models a copy-augmented sequence decoder, which generates the response one word at a time. At any decode time step t, the decoder can either *generate* a word from the decode vocabulary or *copy* a word from the memory. Consequently, the decoder computes: (1) generate distribution $P_g(y_t)$ over the decode vocabulary, and (2) copy distribution $P_c(y_t)$ over words in the memory.

The generate distribution is computed using a standard sequence decoder [27] by attending [17] over the memory cell representations ψ. The copy distribution is generated by using a *two-level attention* over the BoSs memory. Given the decoder state s_t, it first computes attention α_t over the memory cells. Then it computes attention over the tokens in each memory cell m_i. Finally it multiplies both these attentions to compute $P_c(y_t)$ as follows:

$$\alpha_i^t = \text{softmax}(s_t \psi(m_i)) \tag{3.6}$$

$$e_{ij}^t = s_t \phi(w_i^j) \tag{3.7}$$

$$\beta_{ij}^t = \alpha_i^t * \frac{\exp(e_{ij}^t)}{\sum_k \exp(e_{ik}^t)} \tag{3.8}$$

$$P_c(y_t = w) = \sum_{ij:w_i^j = w} \beta_{ij}^t \tag{3.9}$$

The copy and generate distributions are combined using a soft gate $g_s^t \in [0, 1]$ as in [23]. g_s^t is a function of the decoder state at time t and the previous decoded word y_{t-1}.

$$g_s^t = W_g[s_t; y_{t-1}] + b \tag{3.10}$$

$$P(y_t) = g_s^t P_g(y_t) + (1 - g_s^t) P_c(y_t) \tag{3.11}$$

© 2019, *Indian Institute of Technology Delhi*

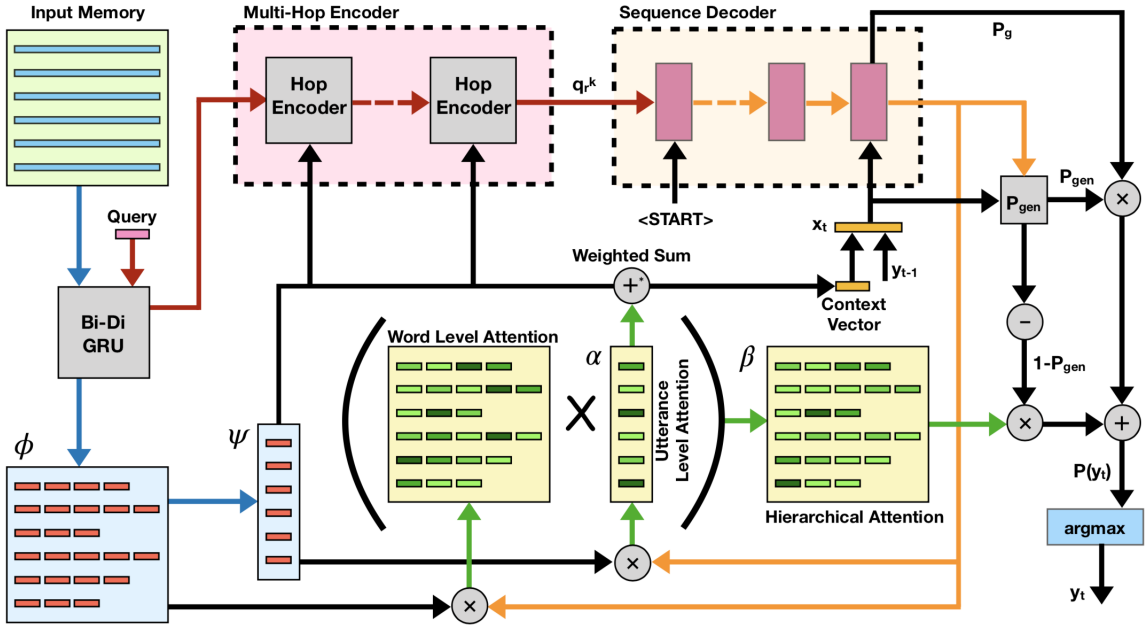


Figure 3.4: The detailed encoder-decoder architecture of BOSSNET depicting attention over its hierarchical memory and implementation of its copy mechanism.

3.1.4 Loss

The decoder is trained using cross-entropy loss. The loss per response is defined as:

$$\mathcal{L}_{ce} = -\sum_{t=1}^{T} \log\Big(g_s^t P_g(y_t) + (1 - g_s^t) P_c(y_t)\Big) \tag{3.12}$$

where T is the number of words in the sequence to be generated and y_t is the word to be generated at time step t. The decision to generate or copy is learnt implicitly by the network. However, to attain perfect disentanglement, the KB words should be copied, while the language should be generated. In other words, any word in the response that is present in the BOSS KB memory should have a low g_s. To obtain this behavior, we define a disentangle label D_l for each word in the response. This label is set to 1 if the word is present in the BOSS KB memory and 0 otherwise. We define a disentangle loss as follows:

$$\mathcal{L}_d = -\sum_{t=1}^{T} g_s^t \log D_l^t + (1 - g_s^t)\log(1 - D_l^t) \tag{3.13}$$

We randomly drop some words with disentangle label set to 1. This *Disentangle Label Dropout (DLD)* works in tandem with the disentangle loss and BOSS memory – it encourages the model to copy KB words whenever possible, based on their surrounding words. The overall loss is given as:

$$\mathcal{L} = \mathcal{L}_{ce} + \gamma \mathcal{L}_d \tag{3.14}$$

© 2019, *Indian Institute of Technology Delhi*

The relative weight of $\mathcal{L}_d$ in the overall loss is controlled using a hyper-parameter (γ). The dropout rate is also a hyper-parameter.

3.2 The BoSsNet-RL Architecture

The reinforment learning based API generation system builds upon the original BoSsNet architecture with the following modules: (1) API Explorer, (2) RL-decoder, and (3) Augmented REINFORCE. We explain these in the upcoming sections.

3.2.1 API Explorer

This module has the objective of aiding the exploration of valid APIs with a strong reward. Our actual reward for a generated API is implicit, because it depends on the responses of the response model based on the results queried by the API. Hence, we go on to define a psuedo reward for each API as the following:

Pseudo Reward The responses by the system depend on the results fetched by the API from the KB. The system will only be able to copy KB information into its response limited to these results. Hence we can measure the performance of a certain API by checking the overlap of the information required by the responses and that present in its results. We use the popular F1 metric which is a combination of precision and recall. For us recall is important because it measures the extent of overlap and the precision will make sure that the results retrieved say as concise as possible. We don't want to have a very generic API which results in a very large result set!. Our metric which we term as the *Psuedo Reward* is calculated as follows.

$$Reward = \frac{2 \cdot R \cdot P}{(R + P)} \tag{3.15}$$

where P = No. of KB items in both results and responses / No. of KB items in results and R = No. of KB items in both results and responses / No. of KB items in responses

Our API Explorer then tries to find (explore) valid APIs/queries with a large psuedo-reward to aid in augmented REINFORCE explained in the sections below.

3.2.2 RL-decoder

The RL-decoder is added as a second decoder to the BoSsNet architecture, and is only activated when the system receives a *Make API* command. It uses a different embedding matrix from the original encoder to encode the context which serves as the initial state to

© 2019, *Indian Institute of Technology Delhi*

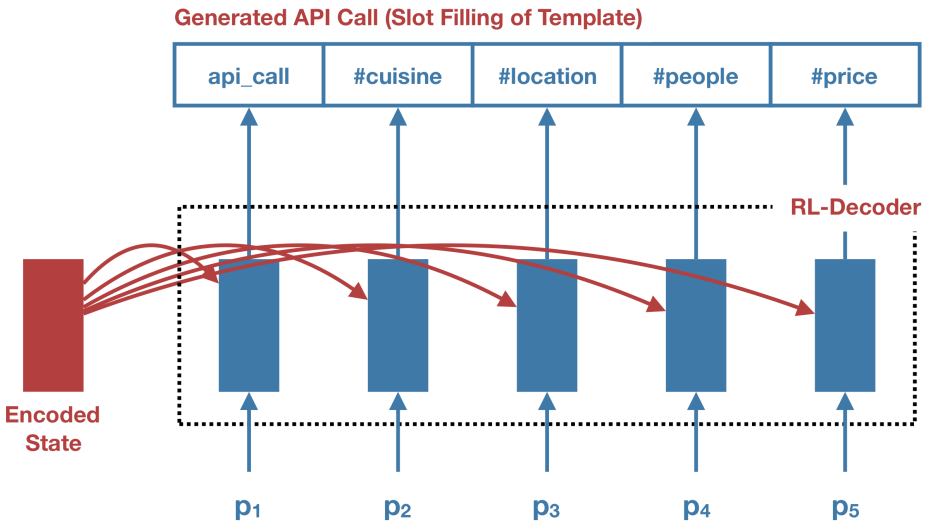


Figure 3.5: The decoder attempts to fill a predefined template of fixed length. It uses only the encoded state and current position p_t to decode to maintain independence between slots.

the decoder. The decoder uses a sequence independence approach to generate the output akin to a slot-filling technique with a fixed template as shown in Figure 3.5. Since the API has been templated it will always generate the same length trajectory and hence it mitigates the length bias issue. Also since each slot is being predicted independent of each other, the decoder will attempt to pick up as much informatiom as possible each time. This will mitigate the data bias issue which arises from the ability to choose what parameters to send in the API call, whereas we generally want to be as precisise as possible.

The initial state is passed on at each timestep of the decode process and only the current timestep is given as input, as in Eq. 3.16. This makes each decode step independent of the previously decoded sequence and it is completely focused on finding the correct information in the context to fill in its current slot/timestep. The decoder is implemented using beam-search and employs constraint decoding over a fixed grammar/template.

$$s_t = W_d[s; p_t] + b \tag{3.16}$$

where s is the initital state to the decoder and p_t is the position embedding for the current time step.

3.2.3 Augmented REINFORCE

The API explorer maintains a buffer of size M containing the set of promising APIs, denoted by $\mathcal{B} = \left\{a^{(i)}\right\}_{i=1}^{M}$, where $a \in \mathcal{A}$ is a valid API trajectory in our entire space of possible trajectories $\mathcal{A}$. We modify our expected objective of our policy π_θ, as done in [15], as a

© 2019, *Indian Institute of Technology Delhi*

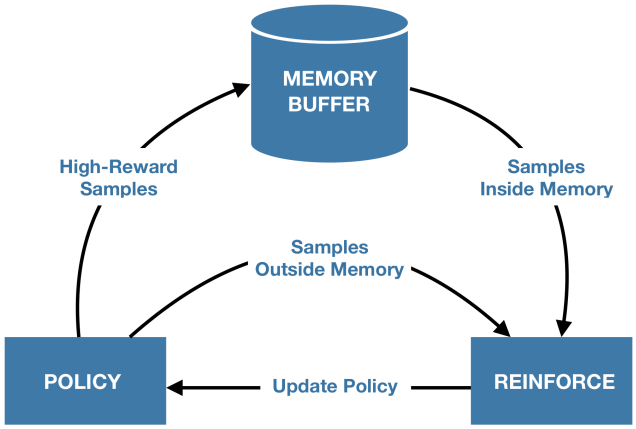


Figure 3.6: The policy update algorithm. A mix of on and off-policy samples are used to train the policy and any new promising trajectories are added to the buffer.

weighted sum of two terms: an expectation over the trajectories inside the memory buffer, and a separate expectation over the trajectories outside the buffer.

$$J^{RL}(\theta) = \alpha \sum_{a \in \mathcal{B}} \pi_\theta^+(a) R(a) + (1-\alpha) \sum_{a \in (\mathcal{A}-\mathcal{B})} \pi_\theta^-(a) R(a) \tag{3.17}$$

here π_θ^+ and π_θ^- denote the normalised probablity in and outside the buffer respectively.

$$\pi_\theta^+(a) = \begin{cases} \pi_\theta(a)/\alpha & \text{if } a \in \mathcal{B} \\ 0 & \text{if } a \notin \mathcal{B} \end{cases}, \quad \pi_\theta^-(a) = \begin{cases} 0 & \text{if } a \in \mathcal{B} \\ \pi_\theta(a)/(1-\alpha) & \text{if } a \notin \mathcal{B} \end{cases} \tag{3.18}$$

The first term of the expectation can be calculated over the entire buffer which is typically small in size. The second expection requires us to sample from π_θ, using beam search over our RL-decoder, and then employing rejection sampling w.r.t the examples in the buffer. This will ensure that the buffer and our on-policy samples are mutually exculsive. The entire pipeline is visualised in Figure 3.6.

Dealing with Language Bias Several default inputs such as *dontcare* or *null* occur very frequenntly in valid APIs. As a result the policy is biased to pick them on occasion. We attempt to mitigate this issue by dividing our dialogs into two sets: (1) confident and (2) confused. The confident examples are the dialogs with no null slots. Confused dialogs on the other hand have one or more null slots and hence multiple APIs giving the best set of results. The confident examples are trained in a similar setting to supervised learning by keeping only the one best API in the buffer. This leads to a robust system and it helps bias it to learn the confused examples correctly.

© 2019, *Indian Institute of Technology Delhi*

Chapter 4

IMPLEMENTATION

In this chapter we will discuss the implementation on various modules of the BoSsNet architecture. We start off by describing all the components we worked on and then explain in detail the parts which are relevant to this thesis.

4.1 Parts of the BoSsNet Architecture

We have created multiple modules, each with a different functionality and objective. We list the major modules in the BoSsNet architecture below:

1. **Main Controller** - This component manages the flow of data and information among the other components. It reads and cleans the data and manages the training and prediction of the dialogs via the memory network.
2. **Data Loader** - This is incharge of reading all the data from files and formatting it in an easy to use way.
3. **Data Batcher** - This module takes data from the data loader class and performs any preprocessing and indexing required by the Memory network to use the data for training.
4. **Memory Network** - This is the network of the encoder and decoder that both trains and predicts the responses for dialogs given the context and user query.
5. **Dynamic Decoder** - This is a built in library of tensorflow that was modified to allow batch decoding and beam search.
6. **Attention Mechanism** - This is a built in library of tensorflow that was modified to enable our multi-level attention mechanism.
7. **Beam Search** - This is the built in library of tensorflow that was modified to allow constraint decoding and the copy generate mechanism.
8. **Evaluator** - This class gives evaluation metric scores on the response predictions by comparing them with the gold examples.
9. **Logging** - This component maintains any errors in the code during runtime and allows for automatic deployment of multiple runs. It also generates log files giving detailed explanations of each run.

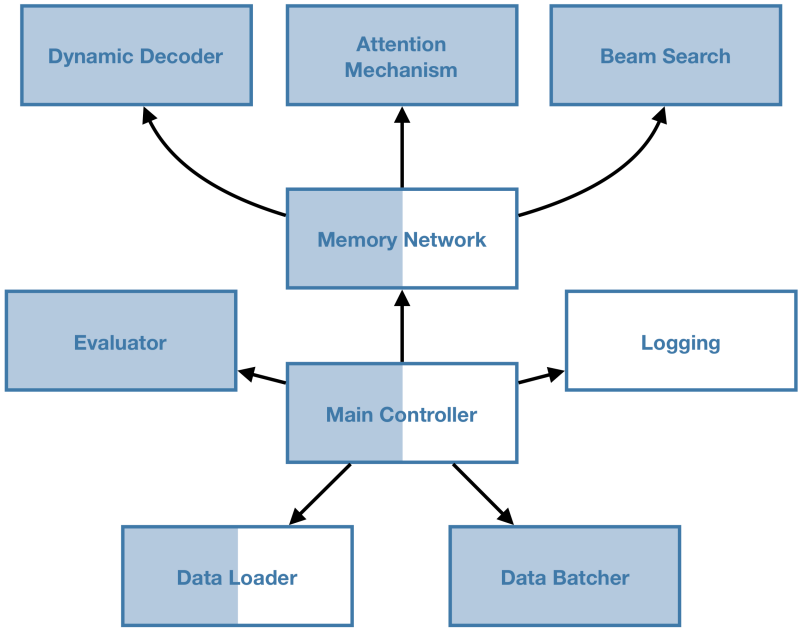


Figure 4.1: Components of the BoSsNet system and their interaction.

We depict the interaction between the multiple components in Figure 4.1. Parts in blue have been explained in this thesis. Half colored blocks mean part of the work regarding that component is explained and the rest is not in the scope of this thesis.

Main Controller

This is the main file that we run from the terminal interface. It manages the input parameters that are given by the user and makes sure they are accessible by any another part of the code in other files as well. It is incharge of calling the Data Loader and the Data Batcher to read the data, preprocess, and getting it ready to be fed as input to the main memory network.

It maintains the train, validation, and test sets separately, each with its different purpose. The validation set is used for early stopping and the state is stored for the best observed validation score using *tf.saver*.

Finally, it is incharge of collecting the responses for each dialog from the model and running the evaluation scripts to get the final metric scores to output to the user.

For readability of the processing and training, we use the *tqdm* library to easily visualise progress as well as get important information such as average loss and time-per-epoch

Data Loader

This class is responsible for reading the dialogs from corresponding files (train, validation, and test). It breaks each dialog on each turn into deparate dialogs. Each dialog is further broken into three parts: context, user query, and system response.

© 2019, *Indian Institute of Technology Delhi*

Data Batcher

The data batcher is responsible for all the preprocessing that needs to be done on the data. Each word has to be replaced with its corresponding index and we have to maintain list of OOV words as well. It removes punctuation and also adds special characters that will aid in training, such as EOS and UNK. All KB associated words are also tagged during the preprocessing which will be important when implementing disentanglement loss.

Further the data batcher is responsible for effectively partitioning the data into batches so that they can be sent into the memory network for training and prediction.

Memory Network

This is the main BoSsNet model containing the entire architecture as discussed in Chapter 3. It hosts the multi-hop encoder and the generative, copy-enabled decoder. It is implemented as a class with all the associated hyperparameters as private variables of this class. This enables us to easily store and transport any learned model, as it has all the information needed to run.

The model has two APIs exposed to the Main Controller: *batch_train* and *batch_fit*. The first one is responsible to train the weights of the model parameters and implements back-propagation over the whole network after calculating the loss. The gold responses are passed as input to the train API. The second one is responsible for predicting the correct responses, but does not perform any learning of the weights. The only inputs to this API is the current dialog context and the user query.

Dynamic Decoder

This component was modified from the original Tensorflow decode library. Its objective is to enable batch decoding in which each prediction of each batch element can be of different sizes. We modified it so that at each timestep it combines the generate vocab distribution and the attention distribution. The code listing for this weighted combination is given in Appendix B.1. We also modified the outputs from this library which now also outputs the p_gen values along with each weighted vocab distribution. This is important for calculating disentanglement loss.

Attention Mechanism

This component was modified from the original Tensorflow decode library. We changed the way that it computes attention so that it could handle multi-level attention over our multi-level BoSs memory. We changed the output of its APIs so that they could incorporate the new attention information. The code listing for the new attention mechanism is given in Appendix B.2. We show how to generate sentence level and word level attention seperately and how to combine them.

© 2019, *Indian Institute of Technology Delhi*

Beam Search

The beam search decoder is another built-in library in Tensorflow which we have modified to work with our modified dynamic decoder, attention mechanism, and copy-enabled output generation. The beam width can be any number as required and has been tested to beam width sizes of 64. It also does a weighted generate and attention combination, which now has to be applied onto multiple beams simulataneously, leading to increased complexity.

Evaluator

This is a script that takes the predicted and the gold responses and performs evaluations of serveral kinds. These evaluated scores are then sent to the main controller which displays them to the user.

Some metrics evaluated by the evaluator are:

- *BLEU Score* - This is a standard metric used on the real-world datasets where exact matches between predicted and gold responses is very rare. Hence BLEU helps to measure degree of closeness of the prediction.
- *F1 Score* - This is a combination of Recall and Precision over the predicted KB items in the responses.
- *Per-Response Accuracy* - For the fabricated datasets (bAbI) we look for an exact match between predicted and gold. The extent of exact matches is measures via this metric.
- *Per-Dialog Accuracy* - This measures how many dialogs had all perfect response predictions for all its turns.

4.2 Parts of the BoSsNetRL Architecture

There are mainly three main components that have been added to the BoSsNet architecture to make BoSsNet-RL. These are listed below.

1. **API Explorer** - This systematically searches the space of APIs and populates the buffer to be used by MAPO.
2. **Rewards** - This takes in a particular API as input along with the expected future responses and returns the reward/pseudo-reward.
3. **RL-Decoder** - This is the main API generator which we train to learn the ideal policy.
4. **Augmented REINFORCE** - This takes in results from both the API explorer populated buffer and the RL-decoder outputs and runs Augmented REINFORCE (MAPO) on top of them.

© 2019, *Indian Institute of Technology Delhi*

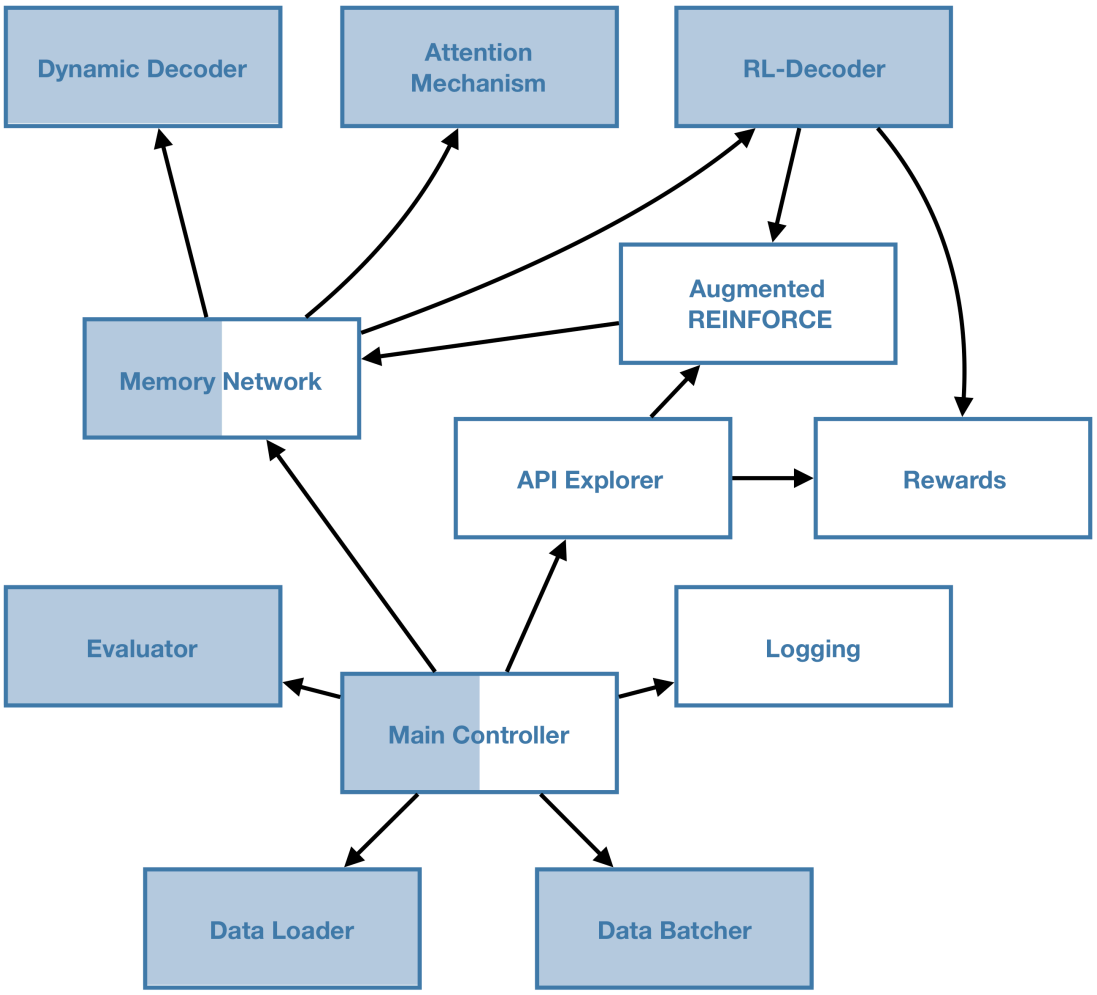


Figure 4.2: The components of the BoSsNet-RL system and their interaction.

We observe the interaction between these added components and the previous architecture in Figure 4.2. Below we discuss modifications to previous components and newly added features.

Main Controller

There are several changes that had to be made to the main controller to handle the RL-decoding policy. First, we implemented a phase-wise training to help enable the RL-decoder and the response model to train independently. In this phase we have 4 phases:

1. **Phase 1** : This trains the response deocder only for dialogs that occur before any API is made.
2. **Phase 2** : This phase adds in the training for the RL-decoder. This is only used in case we use a shared embedding space for encoding the context for the response deocder and RL-decoder. If they use different embeddings, then Phase 2 can be directly merged with Phase 1.
3. **Phase 3** : In this phase we can now start training the system on responses after the

© 2019, *Indian Institute of Technology Delhi*

API call has been made. We include the results retrieved by the RL-decoder in Phase 2.

4. **Phase 4** : This phase calculates the actual reward based on the responses innn Phase 3 and retrains the RL-decoder accordinly. This reward actually measures how well the system was able to train on the retrieved results and will be able to capture the effects of how the results are stored in memory, like sorting based on a field.

These phases are run one after the other and more phases are activated after the previous phases near convergence. For example we start off with just Phase 1 and after it converges we now run Phase 1 and 2 together.

Memory Network

We added three main APIs in the memory network to enable the RL-decoder.

- *api_predict* will attempt to predict good API responses via the RL-decoder policy and utilises beam search to get a large set predictions.
- *api_train* takes in APIs and their rewards and then implements the augmented REINFORCE algorithm
- *api_prob* takes in an API and returns the probability of generating that API with the current policy.

RL-Decoder

The RL-decoder takes in the encoded state as the inital state. It maintains a time variable which corresponds to the current time step of the decoder. This time variable is sent through a position embedding and is then sent as input to the decoder. We set the next state of the decoder as the inital state so that we break the sequential behavior as required. This decoder uses the dynamic decoder with beam search to decode and attends over the input context via the attention wrapper.

© 2019, *Indian Institute of Technology Delhi*

Chapter 5

RESULTS

5.1 Setup

5.1.1 Datasets

We perform experiments on three task-oriented dialog datasets: bAbI Dialog [4], CamRest [31], and Stanford Multi-Domain Dataset [6].

bAbI Dialog consists of synthetically generated dialogs with the goal of restaurant reservation. The dataset consists of five different tasks, all grounded to a KB. This KB is split into two mutually exclusive halves. One half is used to generate the train, validation, and test sets, while the other half is used to create a second test set called the OOV test set.

CamRest is a human-human dialog dataset, collected using the Wiz-of-Oz framework, also aimed at restaurant reservation. It is typically used to evaluate traditional slot filling systems. In order to make it suitable for end-to-end learning, we stripped the handcrafted state representations and annotations in each dialog, and divided the 676 available dialogs into train, validation, and test sets (406, 135, and 135 dialogs, respectively).

Stanford Multi-Domain Dataset (SMD) is another human-human dialog dataset collected using the Wiz-of-Oz framework. Each conversation is between a driver and an in-car assistant. The other datasets consist of dialogs from just one domain (restaurant reservation), whereas SMD consists of dialogs from multiple domains (calendar scheduling, weather information retrieval, and navigation).

5.1.2 Knowledge Adaptability (KA) Test Sets

Each bAbI dialog task has an additional OOV test set, which helps to evaluate a model's robustness to change in information in the KB. A model that perfectly disentangles language and knowledge should have no drop in accuracy on the OOV test set when compared to the non-OOV test set. To measure the degree of disentanglement in a model, we generated 10 additional test sets for each real-world corpus by varying the percentage (in multiples of 10) of unseen entities in the KB. Our adversarial attacks systematically pick random KB entities and replace all their occurrences in the dialog with new entity names. We will refer to these generated dialogs as the *Knowledge Adaptability* (KA) test sets.

5.1.3 Baselines

We compare BOSSNET against several existing end-to-end task-oriented dialog systems. These include retrieval models, such as the query reduction network (QRN) [24], memory network (MN) [4], and gated memory network (GMN) [16].

We also compare against generative models such as a sequence-to-sequence model (Seq2Seq), a copy augmented Seq2Seq (Seq2Seq+Copy) [9], and Mem2Seq [18].[1] For fairness across models, we do not compare against key-value retrieval networks [6] as they simplify the dataset by canonicalizing all KB words in dialogs.

We noticed that the reported results in the Mem2Seq paper are not directly comparable, as they pre-processed[2] training data in SMD and bAbI datasets. For fair comparisons, we re-run Mem2Seq on the original training datasets. For completeness we mention their reported results (with pre-processing) as Mem2Seq*.

5.1.4 Evaluation Metrics

We evaluate BOSSNET and other models based on their ability to generate valid responses. The *per-response accuracy* [4] is the percentage of generated responses that exactly match their respective gold response. The *per-dialog accuracy* is the percentage of dialogs with all correctly generated responses. These accuracy metrics are a good measure for evaluating datasets with boilerplate responses such as bAbI.

To quantify performance on other datasets, we use *BLEU* [21] and *Entity F1* [7] scores. BLEU measures the overlap of n-grams between the generated response and its gold response and has become a popular measure to compare task-oriented dialog systems. Entity F1 is computed by micro-F1 over KB entities in the entire set of gold responses.

5.1.5 Human Evaluation

We use two human evaluation experiments to compare (1) the *usefulness* of a generated response with respect to solving the given task, and (2) the *grammatical correctness* and *fluency* of the responses on a 0–3 scale. We obtain human annotations by creating Human Intelligence Tasks (HITs) on Amazon Mechanical Turk (AMT). For each test condition (percentage of unseen entities), we sampled 50 dialogs from Camrest and SMD each, and two AMT workers labeled each system response for both experiments, resulting in 200 labels

[1] We thank the authors for releasing a working code at `https://github.com/HLTCHKUST/Mem2Seq`

[2] Mem2Seq used the following pre-processing on the data: 1) The subject (restaurant name) and object (rating) positions of the rating KB tuples in bAbI dialogs are flipped 2) An extra fact was added to the navigation tasks in SMD which included all the properties (distance, address, etc.) combined together as the subject and *poi* as the object. See Appendix.

© 2019, *Indian Institute of Technology Delhi*

per condition per dataset per system. We evaluate four systems in this study, leading to a total of 1600 labels per condition.

Response Relevance Test We show a sample of an Human Intelligence Task (HIT) on Amazon Mechanical Turk in Figure 5.1a. We randomize the responses generated by the three baseline models and BOSSNET on the same dialog and ask the user to tick all those response options that seem to capture the relevant information of the given sample response. A total of 200 such annotations were collected for Camrest and SMD each.

Response Grammar Test We show a sample of an Human Intelligence Task (HIT) on Amazon Mechanical Turk in Figure 5.1b. We randomize the responses generated by the three baseline models and BOSSNET on the same dialog and ask the user to rate each response based on the grammatical correctness and natural flow of the sentence. The rating ranges from 0-3 where 0 being the worst and 3 being the best. Note) the sentences were not asked to be rated with respect to each other, but instead as individual occurrences. A total of 200 such annotations were collected for Camrest and SMD each.

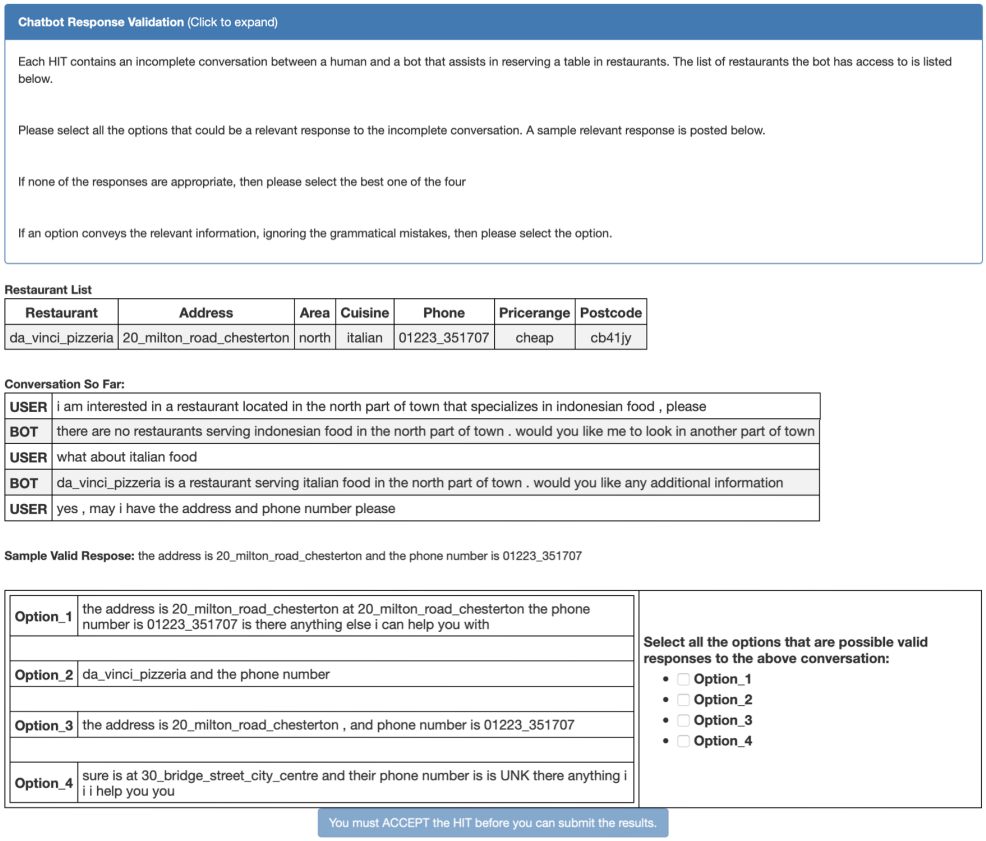
5.1.6 Training

We train BOSSNET using an Adam optimizer [11] and apply gradient clipping with a clip-value of 40. We identify hyper-parameters based on the evaluation of the held-out validation sets. We sample word embedding, hidden layer, and cell sizes from {64, 128, 256} and learning rates from {10^{-3}, $5{\times}10^{-4}$, 10^{-4}}. The hyper-parameter γ in the loss function is chosen between [0-1.5]. The Disentangle Label Dropout rate is sampled from {0.1, 0.2}. The number of hops for multi-hop attention in the encoder is sampled from {1, 3, 6}. We list out the complete set of hyperparameters used to train BOSSNET for the various datasets in Table 5.1.

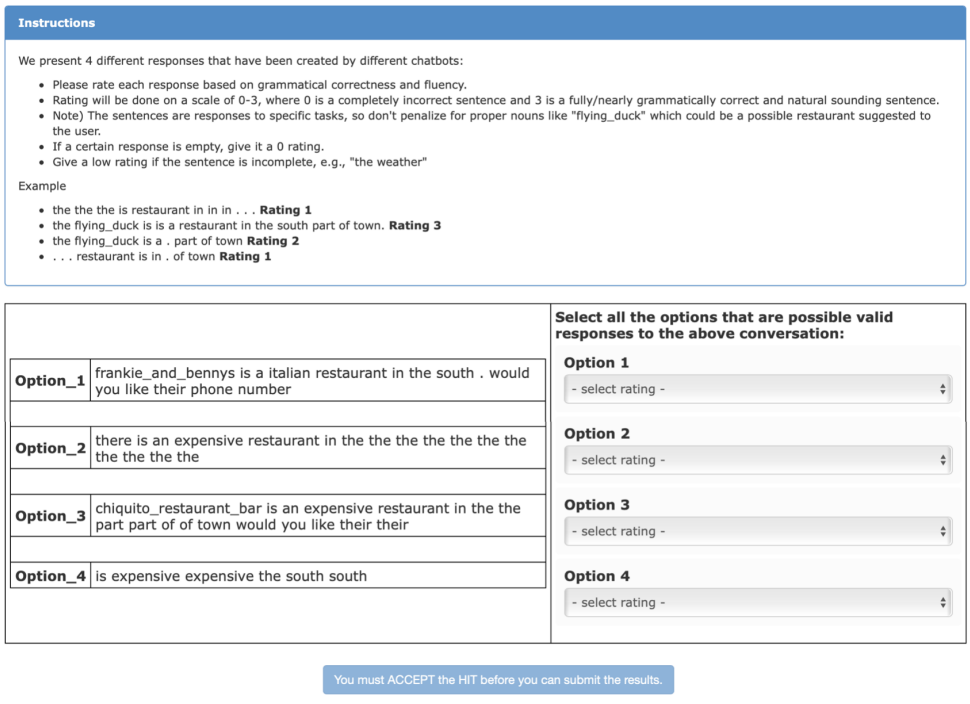
Task	Learning Rate	Hops	Embedding Size	Disentangle Loss Weight	DLD
T1	0.001	1	128	1.0	0.2
T2	0.001	1	128	1.0	0.2
T3	0.0005	3	128	1.5	0.2
T4	0.001	1	128	1.0	0.2
T5	0.0005	3	256	1.0	0.2
CamRest	0.0005	6	256	1.0	0.2
SMD	0.0005	3	256	1.0	0.1

Table 5.1: The hyperparameters used to train BOSSNET on the different datasets
.

© 2019, *Indian Institute of Technology Delhi*



(a)



(b)

Figure 5.1: A sample HIT on Amazon Mechanical Turk to (a) validate useful responses based on the given dialog context, and (b) validate grammatical correctness of different responses on a scale of 0-3

© 2019, *Indian Institute of Technology Delhi*

5.2 Experimental Results

Our experiments evaluate three research questions.

1. *Performance Study*: How well is BoSsNet able to perform the tasks of our three datasets as compared to the baseline models?
2. *Disentanglement Study*: How robust are the models in generalizing on the KA test sets?
3. *Ablation Study*: What is the performance gain from each novel feature in BoSsNet?

5.2.1 Performance Study

	Generative Models				
Task	**Mem2Seq***	**Seq2Seq**	**Seq2Seq+Copy**	**Mem2Seq**	BoSsNet
T1	100 (100)	100 (100)	100 (100)	100 (100)	100 (100)
T2	100 (100)	100 (100)	100 (100)	100 (100)	100 (100)
T3	94.7 (62.1)	74.8 (0)	85.1 (19.0)	74.9 (0)	**95.2 (63.8)**
T4	100 (100)	57.2 (0)	100 (100)	100 (100)	100 (100)
T5	**97.9 (69.6)**	97.2 (64.4)	96 (49.1)	97.7(66.3)	97.3 (65.6)
T1-OOV	94.0 (62.2)	81.7 (0)	92.5 (54.7)	94.0 (62.2)	**100 (100)**
T2-OOV	86.5 (12.4)	78.9 (0)	83.2 (0)	86.5 (12.4)	**100 (100)**
T3-OOV	90.3 (38.7)	75.3 (0)	82.9 (0)	75.2 (0)	**95.7 (66.6)**
T4-OOV	100 (100)	57.0 (0)	100 (100)	100 (100)	100 (100)
T5-OOV	84.5 (2.3)	67.4 (0)	73.6 (0)	75.6 (0)	**91.7 (18.5)**

Table 5.2: Per-response and per-dialog accuracies (in brackets) on bAbI dialog tasks of BoSsNet and other generative model baselines. We highlight the best accuracies achieved for each task.

Table A.2 reports the per-response and per-dialog (in parentheses) accuracies on the bAbI dialog tasks for the generative models. Due to brevity we show the results for the retrieval-based models in the Appendix. The multi-hop retrieval-based models such as QRN, MN and GMN perform well on the non-OOV test sets for tasks 1, 2, and 5, but fail to exhibit similar performance on the corresponding OOV test sets. This result is expected as these models are trained to retrieve from a pre-defined set of responses. Their poor non-OOV performance on tasks 3 and 4 is attributed to an error in the bAbI dataset construction, due to which, the non-OOV and OOV test conditions are the same for these tasks (see Appendix).

A simple generative model (Seq2Seq) achieves accuracies comparable to the multi-hop retrieval models. Enabling it with the ability to copy from the context (Seq2Seq+Copy) shows a considerable increase in performance, especially on the OOV test sets (and non-OOV tests for tasks 3 and 4).

© 2019, *Indian Institute of Technology Delhi*

	CamRest		SMD	
	BLEU	**Ent. F1**	**BLEU**	**Ent. F1**
Mem2Seq*	12.7	39	12.6	33.4
Seq2Seq	11.4	40.6	8.7	34.9
Seq2Seq+Copy	4.7	32.2	3.23	16.9
Mem2Seq	12.7	39	**10.3**	31.8
BoSsNet	**15.2**	**43.1**	8.3	**35.9**

Table 5.3: Performance of BoSsNet and baselines on the CamRest and SMD datasets

	CamRest		SMD	
	Info	**Grammar**	**Info**	**Grammar**
Seq2Seq	46	2.24	35	2.38
Seq2Seq+Copy	27	1.1	21	1.04
Mem2Seq	51	2.2	**38**	2.0
BoSsNet	**77**	**2.28**	36	**2.5**

Table 5.4: AMT Evaluations on CamRest and SMD

The strong performance of simple sequence encoders when compared with multi-hop encoders (in retrieval models) raises a question about the value of multi-hop inference. Mem2Seq answers this question, by obtaining improvements in several tasks, specifically on their OOV test sets. This clearly shows that multi-hop inference and the copy mechanism are essentials for task-oriented dialogs.

Despite gains from the Mem2Seq model, the performance difference between the non-OOV and OOV test sets remains large. BoSsNet succeeds to bridge this gap with its ability to better interpret unseen words, using their surrounding context. It obtains significant improvements on average of about 34% per-dialog accuracy and 10% per-response accuracy for the bAbI OOV test sets.

In Table 5.3, we report results on the real-world datasets. BoSsNet greatly outperforms other models in both Entity F1 metric and BLEU scores on CamRest. On SMD, BoSsNet achieves the best only in Entity F1. On further analysis of the generated responses we observe that BoSsNet responses often convey the necessary entity information from the KB. However, they consist of meaningful phrases with little lexical overlap with the gold response, reducing the BLEU scores. We investigate this further in our human evaluation.

Human Evaluation: We summarize the human evaluation results for real-world datasets in Table 5.4. BoSsNet shows the best performance on Camrest, and is judged useful 77 times out of 100. Also, it has the highest average grammatical correctness score of 2.28 (very close to Seq2Seq and Mem2Seq). BoSsNet performs on par with Mem2Seq and Seq2Seq in its ability to relay appropriate information to solve SMD dialog tasks, and has a slightly higher grammaticality score.

© 2019, *Indian Institute of Technology Delhi*

KB (restaurant|address)
r_bangkok_overpriced_thai_8|r_bangkok_overpriced_thai_8_addr
r_bangkok_overpriced_thai_7|r_bangkok_overpriced_thai_7_addr
r_bangkok_overpriced_thai_4|r_bangkok_overpriced_thai_4_addr
r_bangkok_overpriced_thai_2|r_bangkok_overpriced_thai_2_addr

usr-1	may i have a table in an *overpriced* price range for *nine* people with *thai* food in *bangkok* ?
sys-1	what do you think of : *r_bangkok_overpriced_thai_8* ?
usr-2	can you provide the address ?
Gold	here it is *r_bangkok_overpriced_thai_8_addr*
Seq2Seq+Copy	here it is *r_bangkok_overpriced_thai_4_addr*
Seq2Seq	here it is *r_london_moderate_spanish_6_addr*
Mem2Seq	here it is *r_bangkok_overpriced_thai_4_addr*
BOSSNET	here it is *r_bangkok_overpriced_thai_4_addr*

Table 5.5: Example from bAbI Task 5 KA test set with 100% OOV entities. Identifying the address of an unseen restaurant is challenging for all models.

5.2.2 Disentanglement Study

We use our generated knowledge adaptability (KA) test sets to measure the robustness of BOSSNET and the other baselines to changes in the KB. We perform this experiment on 4 different tasks, namely bAbI tasks 1 and 5, CamRest, and SMD.

Figures 5.2a and 5.3a show the per-response accuracies of the two bAbI dialog tasks plotted against the percentage of unseen entities in KA sets. From Figure 5.2a we observe that BOSSNET remains immune to any variablity in the KB content, whereas the performance of Mem2Seq and Seq2Seq models drops drastically due to their inability to capture semantic representations of the injected KB entities. We see a similar trend in Figure 5.3a, but here all the models show a drop in performance, with BOSSNET appearing the most steady. We explain this trend using the example dialog in Table 5.5. In the current dialog context, the system is required to provide the address of the selected restaurant, but since more than one restaurant in the KB is unseen, it becomes ambiguous for the network to identify the correct restaurant and infer its address. In the end, the system is forced to pick a random address – the probability of which being correct decreases as more restaurants become unseen.

The performance on the CamRest KA test sets is illustrated in Figures 5.4a and 5.4b. BOSSNET has the best performance with even a slight increase in both BLEU and Entity F1 metrics as more OOV content is injected in the dialog, probably because it is clear that it needs to copy when processing unseen entities. Seq2Seq+Copy is unable to perform well

© 2019, *Indian Institute of Technology Delhi*

	CamRest		SMD	
	Info	**Grammar**	**Info**	**Grammar**
Seq2Seq	26	2.28	22	**2.44**
Seq2Seq+Copy	22	1.22	16	1.04
Mem2Seq	35	2.06	26	1.9
BoSsNet	**80**	**2.44**	**51**	2.28

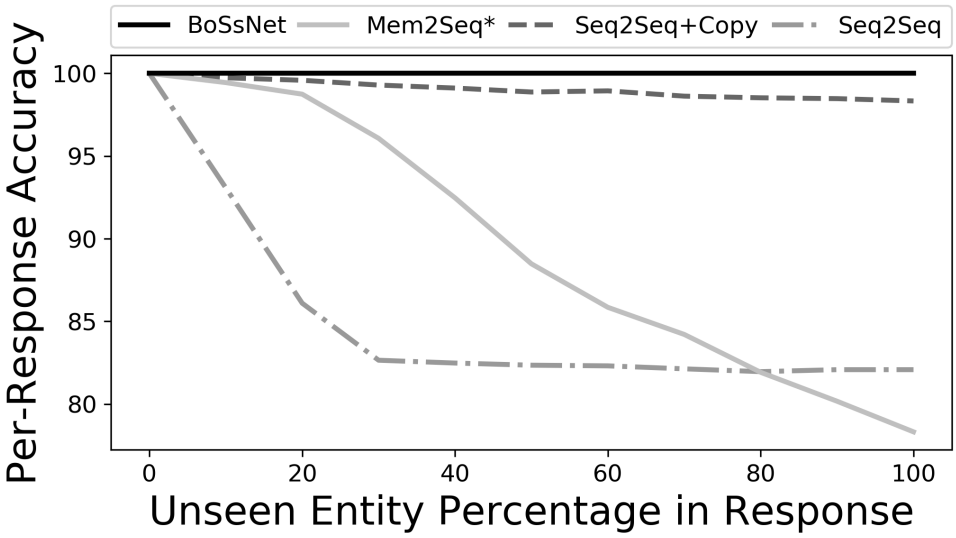
Table 5.6: AMT Evaluations on CamRest and SMD (50% unseen) KA datasets

in CamRest as the length of the input (dialog history + KB tuples) is long and the size of the training set is also small. We believe that Seq2Seq+Copy works best in an environment with an abundance of short dialog training data (e.g., bAbI task 1 in Figure 5.2a).

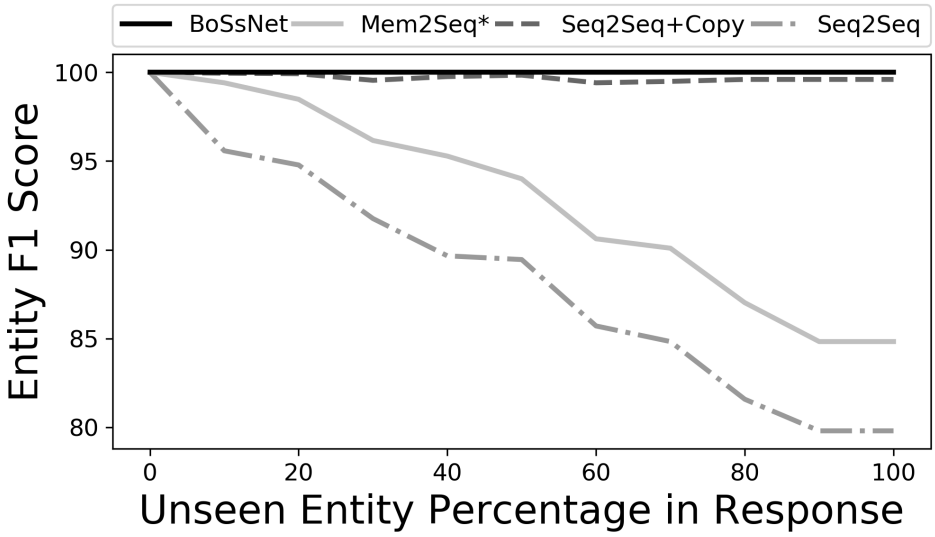
SMD consists of dialogs with a large KB and a highly varying response pattern. This makes it very difficult to learn the language model – reflected in the low BLEU scores for all the systems. BoSsNet still provides the best F1 entity score due to its ability to inference efficiently on the large KB (Figure 5.5a). Mem2Seq shows the best BLEU score performance on the original test set, but its performance drop of 42.5%, from 10.3 at 0% unseen to 5.93 at 100% unseen, is a lot heavier than that of BoSsNet which only drops 7.6% – 8.27 at 0% unseen to 7.64 at 100% unseen.

Human Evaluation: We summarize the human evaluation results for real-world datasets on the 50% unseen KA test set in Table 5.6. BoSsNet again outperforms the baselines and is labeled *successful* twice more often than the next best model on both Camrest and SMD. Seq2Seq appears to produce better sentence structures on the SMD dataset, primarily because it does not attempt to learn inference on the KB, allowing it to solely focus on learning the language model better.

© 2019, *Indian Institute of Technology Delhi*



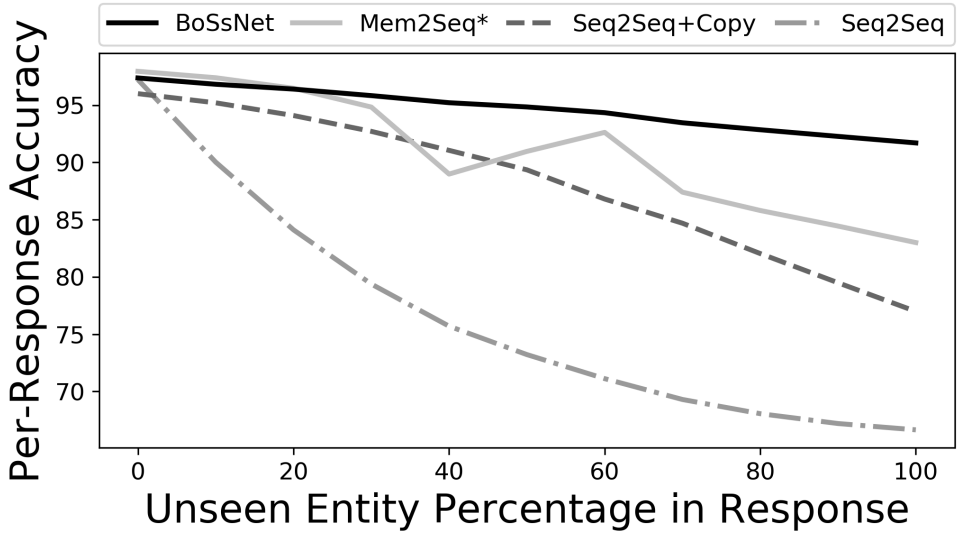
(a)



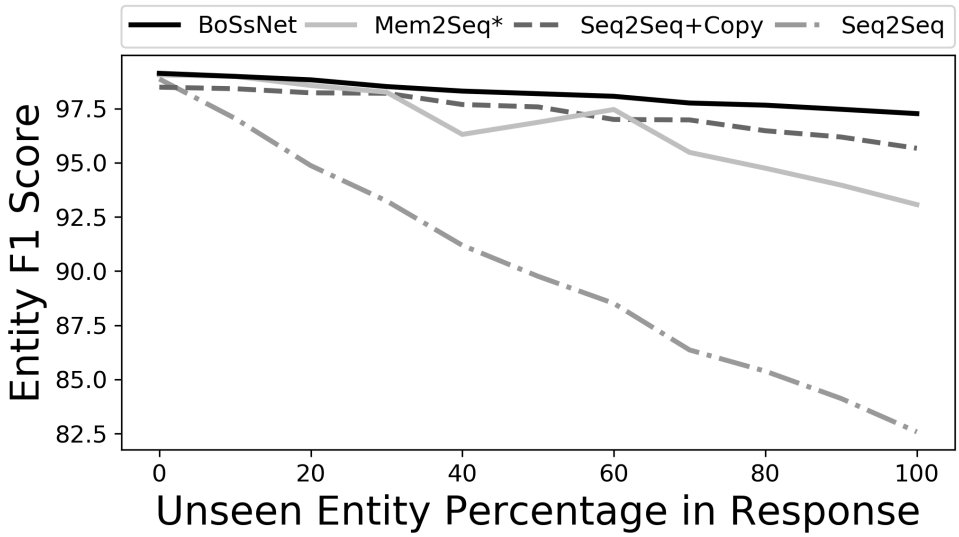
(b)

Figure 5.2: Task 1 Disentanglement Evaluation Plots

© 2019, *Indian Institute of Technology Delhi*



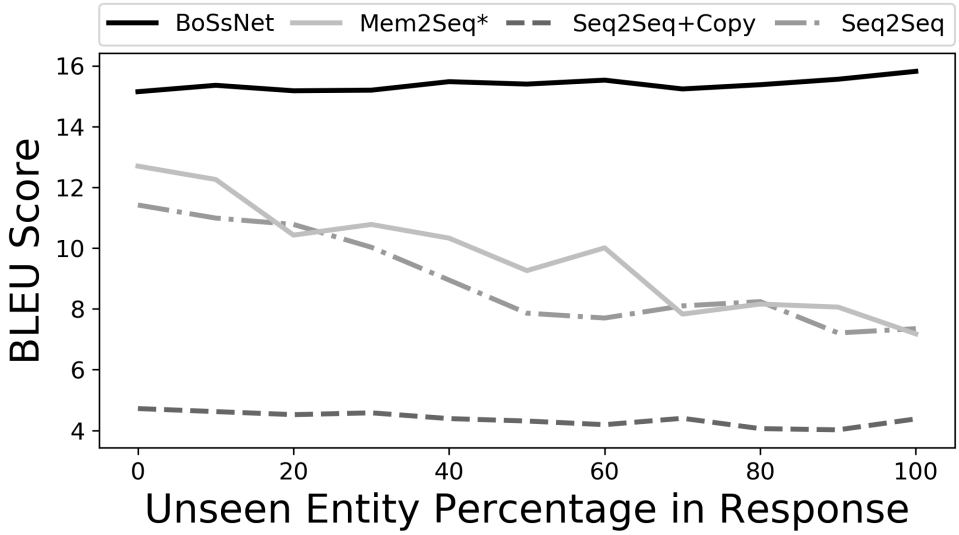
(a)



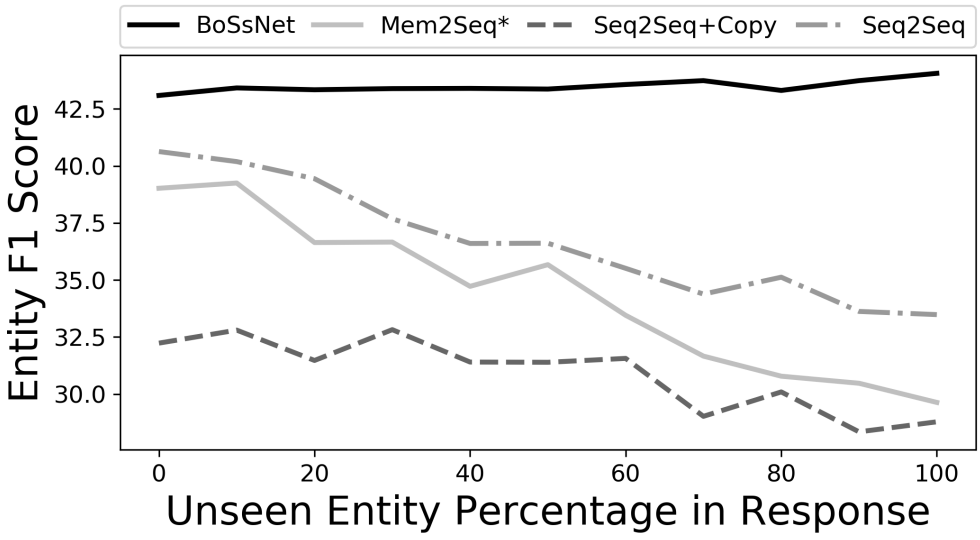
(b)

Figure 5.3: Task 5 Disentanglement Evaluation Plots

© 2019, *Indian Institute of Technology Delhi*



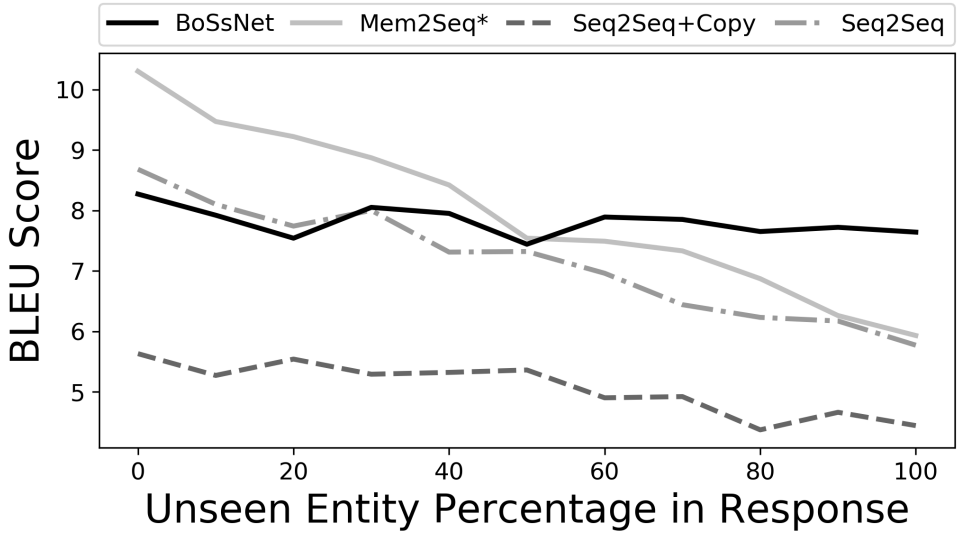
(a)



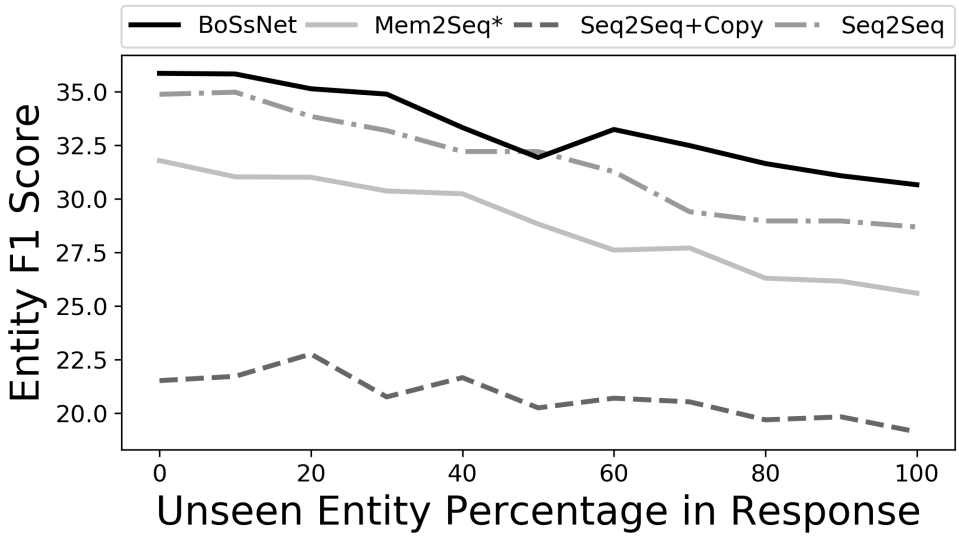
(b)

Figure 5.4: CamRest Disentanglement Evaluation Plots

© 2019, *Indian Institute of Technology Delhi*



(a)



(b)

Figure 5.5: SMD Disentanglement Evaluation Plots

© 2019, *Indian Institute of Technology Delhi*

5.2.3 Ablation Study

We assess the value of each model element, by removing it from BoSsNet. Table 5.7 reports the per-response accuracy scores for various configurations of BoSsNet on bAbI dialog tasks. It also reports the BLEU and entity F1 metric of various configurations on CamRest.

Without BoSs Memory: This configuration uses the Bag-of-Bags (BoB) Memory rather than BoSs memory. The BoB memory is a simplified representation, similar to the one in the original Memory Networks. Here the token representation is the vector embedding of the token with no influence from the surrounding words and the memory cell representation is the sum of all its token embeddings. As a result, each word w representation is influenced equally by all words in a memory cell, irrespective of its distance from w. This makes capturing context in the immediate neighbourhood harder. Inability to capture the correct context prevents the configuartion from generalizing to OOV test sets.

Without Disentangled Loss: Disentangled Loss ($\mathcal{L}_d$) plays an important role in enforcing that KB words be copied and other language be generated. By removing this loss component, it achieves better BLEU score in CamRest, but with a drop in Entity F1. Without the disentangled loss, the model sometimes learns to generate KB words. This severely affects OOV performance. As described earlier, an error in bAbI dataset construction tasks 3 and 4 effectively injects the validation set with a lot of OOVs. This anomaly in conjunction with the dropout (DLD), helps the configuration in achieving an acceptable performance for those tasks.

Without Disentangled Label Dropout: BoSsNet learns to generate language and copy KB words. Without DLD, the model learns to memorize words to be copied rather than learning the context under which a word should be copied. Hence, the performance on OOV test sets is much inferior compared to the non-OOV setting.

Overall, we notice that combining all three model elements is necessary in obtaining the best performance across all tasks.

Multi-Hop vs 1-Hop Encoders: Table 5.8 shows the performance of bAbI tasks and CamRest on two BoSsNet encoder settings. Multi-hops in encoder helps in bAbI task 3 and 5, as they require inferencing over the KB tuples (sorting restaurants by rating) to recommend a restaurant. We also see substantial improvements on CamRest in both BLEU and entity F1 metric.

© 2019, *Indian Institute of Technology Delhi*

	bAbI Dialog	**bAbI Dialog (OOV)**	**CamRest**
	Avg. Response Acc.	Avg. Response Acc.	Ent. F1
BoSsNet w/o BoSs Memory	85.54	77.14	29
BoSsNet w/o $\mathcal{L}_d$	97.2	84.3	40.1
BoSsNet w/o DLD	97.74	86.52	40.45
BoSsNet	98.5	97.48	43.1

Table 5.7: Ablation study: impact of each model element on BoSsNet

	bAbI Dialog Tasks					**bAbI Dialog Tasks (OOV)**					**CamRest**	
	T1	T2	T3	T4	T5	T1	T2	T3	T4	T5	BLEU	Ent. F1
1-Hop	100	100	92.3	100	90.5	100	100	91.4	100	89	10.5	36.9
Multi-Hop	100	100	95.2	100	97.3	100	100	95.7	100	91.7	15.2	43.1

Table 5.8: Ablation study: impact of hops in BoSsNet encoder

5.2.4 Qualitative Evaluation

We qualitatively compare the performance of BoSsNet with other baselines using examples.

Table 5.9, demonstrates the ability of BoSsNet to copy entities (restaurant name and address) in its response. The other baselines either generate unwanted or irrelevant entities in their response, or fail to copy altogether. BoSsNet also best captures the language model effectively with a slight paraphrasing of the gold response.

Table 5.10 contains only unseen entities. This example highlights the shortcomings of the Seq2Seq model as it ends up predicting a restaurant encountered during training. Mem2Seq copies a restaurant name without learning to sort the restaurants based on rating. BoSsNet, with its efficient memory addressing, is seen to be able to solve both issues.

5.2.5 Visual Evaluation

To visualize the benefit of two-level attention used on BoSs memory by the decoder, we compare attention weights for two models: our proposed *two-level attention* and a variant with just *one-level attention* (over all the words in the memory). In the example of a sample dialog from bAbI Task 3, shown in Figure 5.6, the decoder is aimed at predicting the second best restaurant *3 stars*, given that the restaurant with rating *8 stars* has already been suggested and rejected. We show attention only on the KB entries for brevity.

The models share some similarities in their distribution of attention. First, the attention weights are localized over the restaurant names, indicating the preference of the system to point to a specific restaurant. This is supported by the g_s values, 3.14 x 10^{-5} and 1.15 x 10^{-4} for two-level attention and one-level attention respectively, i.e., both models prefer to

© 2019, *Indian Institute of Technology Delhi*

KB (restaurant|cuisine|address|phone) *pizza_hut_fen_ditton|italian|newmarket_road_fen_ditton|01223_323737*	
usr-1	may i have information for an *italian* restaurant in the *east* part of town ?
sys-1	yes sure . there is only on *italian* restaurant called *pizza_hut_fen_ditton* in the *east* . what else do you want to know ?
usr-2	what is their address please ?
Gold	the address for *pizza_hut_fen_ditton* is *newmarket_road_fen_ditton* . can i help you with anything else ?
Seq2Seq+Copy	their address is at
Seq2Seq	*curry_prince* is at *451_newmarket_road_fen_ditton*
Mem2Seq	*pizza_hut_fen_ditton* is located at *7_barnwell_road_fen_ditton* is *7_barnwell_road_fen_ditton* and the phone number is *01223_244955*
BoSsNet	*pizza_hut_fen_ditton* is located at *newmarket_road_fen_ditton* is there anything else i can help you with

Table 5.9: An example of responses generated by BoSsNet and baselines on the CamRest test set. This example has no unseen entities.

KB (restaurant|cuisine|price_range|location|rating) *resto_tokyo_affordable_vietnamese_8stars|vietnamese|affordable|tokyo|8* *resto_tokyo_affordable_vietnamese_7stars|vietnamese|affordable|tokyo|7* *resto_tokyo_affordable_vietnamese_6stars|vietnamese|affordable|tokyo|6* *resto_tokyo_affordable_vietnamese_5stars|vietnamese|affordable|tokyo|5*	
usr-1	can you book a table in *tokyo* with *vietnamese* food in a *affordable* price range ?
sys-1	what do you think of this option : *resto_tokyo_affordable_vietnamese_8stars* ?
usr-2	no this does not work for me .
sys-2	what do you think of this option : *resto_tokyo_affordable_vietnamese_7stars* ?
usr-3	do you have something else ?
Gold	what do you think of this option : *resto_tokyo_affordable_vietnamese_6stars*
Seq2Seq+Copy	what do you think of this option : what ?
Seq2Seq	what do you think of this option : *resto_london_moderate_british_2stars* ?
Mem2Seq	what do you think of this option : *resto_tokyo_affordable_vietnamese_5stars* ?
BoSsNet	what do you think of this option : *resto_tokyo_affordable_vietnamese_6stars* ?

Table 5.10: An example of responses generated by BoSsNet and baselines on bAbI dialog Task-5. This example is from the KA test set with 100% unseen entities.

© 2019, *Indian Institute of Technology Delhi*

copy rather than generate. Moreover, entries with the same restaurant name have similar attention weights, reflecting the robustness of the distribution.

We also observe that two-level attention is able to perform the difficult task of *sorting* the restaurant entries based on decreasing order of rating (number of stars). It gives more weight to entries with a high rating (*3 stars* > *2 stars* > *1 star*) and suppresses the weights of any previously suggested restaurant.

The attention over memory cells provides BoSsNet with the ability to infer over multiple sets of tuples. The ability to sort the restaurants and reject a previously seen restaurant can be observed by the attention heat map of Memory cells. Attention over tokens on the other hand can push the attention weights towards either the subject or object in the KB tuple, based on the query's request. Thus using both in conjunction helps BoSsNet perform significantly better than the baselines and illustrates the importance of the BoSs memory in comparison to a flat memory layout.

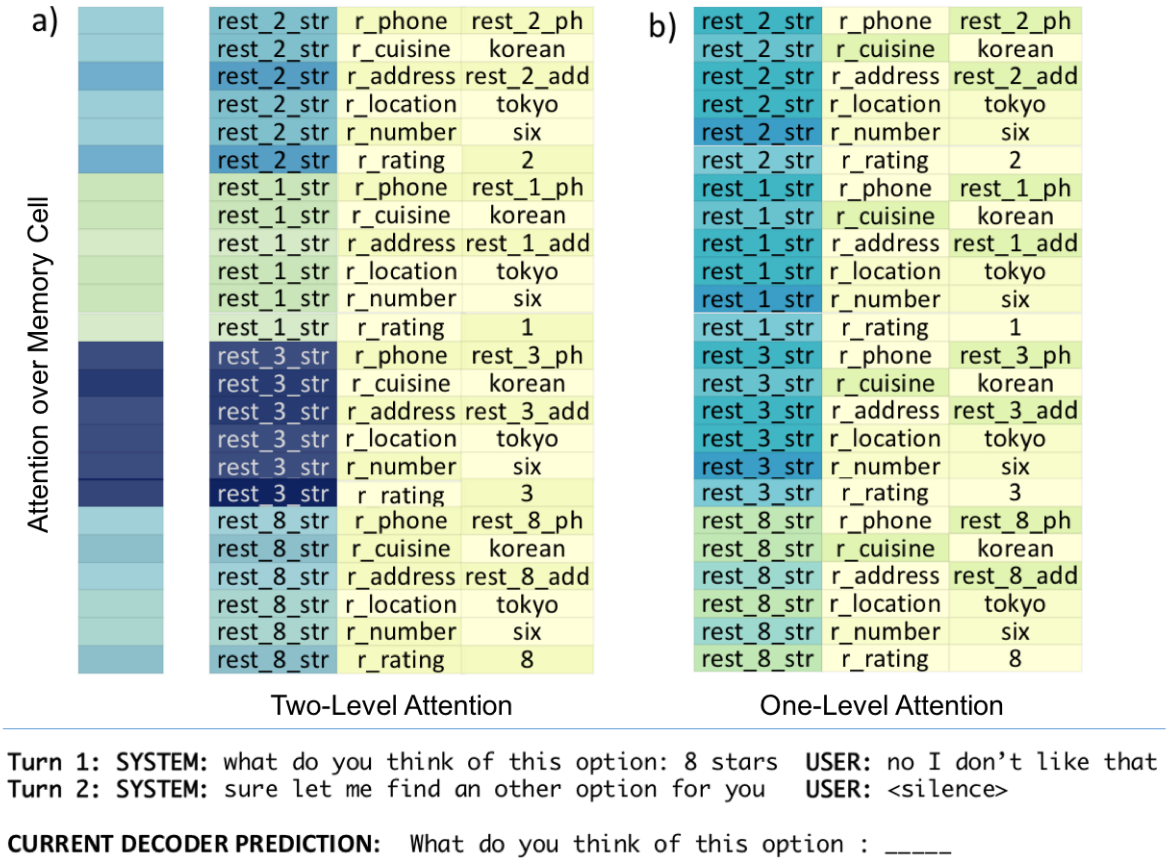


Figure 5.6: Visualization of attention weights on selected portions of memory in (a) BoSsNet with two-level attention vs (b) BoSsNet with one-level attention

© 2019, *Indian Institute of Technology Delhi*

5.3 BoSsNet-RL Experiments

5.3.1 Datasets

We modified two of the previously used datasets: (1) **bAbI Task 3** and (2) **Camrest** and used them for this experiment. We removed the API call and the knowledge base results in both of the datsets and added a single *make API* symbol in their place.

5.3.2 Baselines

We used two baselines for this task

Ground Truth in which we provide the true API call for each dialog instance. It is more of a upper limit on the performance for the problem, rather than a baseline.

Greedy in which we train the RL-decoder using supervised learning with the best APIs found during initial exploration as gold. We run exploration long enought to capture the highest reward APIs for each dialog, which is made feasible with our small search space.

5.3.3 Results

We train the RL-decoder using a two phase training mechanism. In the first phase we train just the RL-decoder till it nearly converges. In the second phase we start training the response encoder-decoder architecturee in tandem with the RL-decoder.

During Phase 1 of the training, the decoder mangages to converge to 100% perfect API prediction on the bAbI Task 3 train set in 10 epochs. We show the performance on the corresponding test sets in Table 5.11. From the table we observe that the Ground Truth manages to reach nearly 84.5 percent accuracy. This is lower than the 94.6% [3] we achieved with BoSsNet on the original Task 3 dataset. We attribute this difference to due hyperparameter tuning as we only had time for runs with one set of hyperparameters during the time of writing of this thesis.

Greedy mode is significantly lower in performance than the Ground Truth which can be attributed to data bias which plagues the greedy implementation. There is no provision for the greedy system to mitigate this bias as it only trains on APIs based on their reward. which might be equal for multiple APIs given our KB.

BoSsNet -RL shows promising results on the first run and very closely matches the ground truth accuracies in both non-OOV and OOV test sets. This reflects on the ability of

[3]This accuracy is caluculated by taking the responses of BoSsNet on Task 3 and removing all the turns in which it had to predict an API call. Hence it doesn't match the reported 95.2%

© 2019, *Indian Institute of Technology Delhi*

	Task 3	**Task 3 OOV**
Ground Truth	**84.5**	**81.17**
Greedy	74.9	74.28
BoSsNet-RL	82.98	79.41

Table 5.11: BoSsNet-RL Evaluation comparison with baselines on bAbI Task 3

the system to successfully overcome the biases which effect MAPO with a normal decoder.

We do not report results on the Camrest dataset as they are still running during the time of writing of this thesis. Upto date infromation will be displayed in the README of the BoSsNet-RL repository.

© 2019, *Indian Institute of Technology Delhi*

Appendix A

SUPPLEMENTARY

A.1 Evaluation Results with Baseline Comparison

	Generative Models				
Task	**Mem2Seq***	**Seq2Seq**	**Seq2Seq+Copy**	**Mem2Seq**	BoSsNet
T1	100 (100)	100 (100)	100 (100)	100 (100)	100 (100)
T2	100 (100)	100 (100)	100 (100)	100 (100)	100 (100)
T3	94.7 (62.1)	74.8 (0)	85.1 (19.0)	74.9 (0)	**95.2 (63.8)**
T4	100 (100)	57.2 (0)	100 (100)	100 (100)	100 (100)
T5	**97.9 (69.6)**	97.2 (64.4)	96 (49.1)	97.7(66.3)	97.3 (65.6)
T1-OOV	94.0 (62.2)	81.7 (0)	92.5 (54.7)	94.0 (62.2)	**100 (100)**
T2-OOV	86.5 (12.4)	78.9 (0)	83.2 (0)	86.5 (12.4)	**100 (100)**
T3-OOV	90.3 (38.7)	75.3 (0)	82.9 (0)	75.2 (0)	**95.7 (66.6)**
T4-OOV	100 (100)	57.0 (0)	100 (100)	100 (100)	100 (100)
T5-OOV	84.5 (2.3)	67.4 (0)	73.6 (0)	75.6 (0)	**91.7 (18.5)**

Table A.1: Per-response and per-dialog accuracies (in brackets) on bAbI dialog tasks of BoSsNet and other generative model baselines. We highlight the best accuracies achieved for each task.

	Retrieval Models			Generative Models
Task	**QRN**	**MN**	**GMN**	BoSsNet
T1	99.9 (-)	99.6 (99.6)	100 (100)	100 (100)
T2	99.5 (-)	100 (100)	100 (100)	100 (100)
T3	74.8 (-)	74.9 (2.0)	74.9 (0)	**95.2 (63.8)**
T4	57.2 (-)	59.5 (3.0)	57.2 (0)	**100 (100)**
T5	**99.6 (-)**	96.1 (49.4)	96.3 (52.5)	97.3 (65.6)
T1-OOV	83.1 (-)	72.3 (0)	82.4 (0)	**100 (100)**
T2-OOV	78.9 (-)	78.9 (0)	78.9 (0)	**100 (100)**
T3-OOV	75.2 (-)	74.4 (0)	75.3 (0)	**95.7 (66.6)**
T4-OOV	56.9 (-)	57.6 (0)	57.0 (0)	**100 (100)**
T5-OOV	67.8 (-)	65.5 (0)	66.7 (0)	**91.7 (18.5)**

Table A.2: Per-response and per-dialog accuracies (in brackets) on bAbI dialog tasks of BoSsNet and other retrieval model baselines

.

	CamRest		SMD	
	BLEU	**Ent. F1**	**BLEU**	**Ent. F1**
Mem2Seq*	12.7	39	12.6	33.4
Seq2Seq	11.4	40.6	8.7	34.9
Seq2Seq+Copy	4.7	32.2	3.23	16.9
Mem2Seq	12.7	39	**10.3**	31.8
BoSsNet	**15.2**	**43.1**	8.3	**35.9**

Table A.3: Performance of BoSsNet and baselines on the CamRest and SMD datasets

	CamRest		SMD	
	Info	**Grammar**	**Info**	**Grammar**
Seq2Seq	46	2.24	35	2.38
Seq2Seq+Copy	27	1.1	21	1.04
Mem2Seq	51	2.2	**38**	2.0
BoSsNet	**77**	**2.28**	36	**2.5**

Table A.4: AMT Evaluations on CamRest and SMD

	CamRest		SMD	
	Info	**Grammar**	**Info**	**Grammar**
Seq2Seq	26	2.28	22	**2.44**
Seq2Seq+Copy	22	1.22	16	1.04
Mem2Seq	35	2.06	26	1.9
BoSsNet	**80**	**2.44**	**51**	2.28

Table A.5: AMT Evaluations on CamRest and SMD (50% unseen) KA datasets

	bAbI Dialog	bAbI Dialog (OOV)	CamRest
	Avg. Response Acc.	Avg. Response Acc.	Ent. F1
BoSsNet w/o BoSs Memory	85.54	77.14	29
BoSsNet w/o $\mathcal{L}_d$	97.2	84.3	40.1
BoSsNet w/o DLD	97.74	86.52	40.45
BoSsNet	98.5	97.48	43.1

Table A.6: Ablation study: impact of each model element on BoSsNet

	bAbI Dialog Tasks					bAbI Dialog Tasks (OOV)					CamRest	
	T1	T2	T3	T4	T5	T1	T2	T3	T4	T5	BLEU	Ent. F1
1-Hop	100	100	92.3	100	90.5	100	100	91.4	100	89	10.5	36.9
Multi-Hop	100	100	95.2	100	97.3	100	100	95.7	100	91.7	15.2	43.1

Table A.7: Ablation study: impact of hops in BoSsNet encoder

© 2019, *Indian Institute of Technology Delhi*

A.2 Dataset Preprocessing and Faults

A.2.1 Mem2Seq Preprocessing

(a) Original bAbI Data

Subject	Predicate	Object
resto_rome_cheap_indian_6stars	R_phone	resto_rome_cheap_indian_6stars_phone
resto_rome_cheap_indian_6stars	R_cuisine	indian
resto_rome_cheap_indian_6stars	R_address	resto_rome_cheap_indian_6stars_address
resto_rome_cheap_indian_6stars	R_location	rome
resto_rome_cheap_indian_6stars	R_number	eight
resto_rome_cheap_indian_6stars	R_price	cheap
resto_rome_cheap_indian_6stars	R_rating	6
resto_rome_cheap_indian_7stars	R_phone	resto_rome_cheap_indian_7stars_phone
resto_rome_cheap_indian_7stars	R_cuisine	indian
resto_rome_cheap_indian_7stars	R_address	resto_rome_cheap_indian_7stars_address
resto_rome_cheap_indian_7stars	R_location	rome
resto_rome_cheap_indian_7stars	R_number	eight
resto_rome_cheap_indian_7stars	R_price	cheap
resto_rome_cheap_indian_7stars	R_rating	7

Subject	Predicate	Object
resto_rome_cheap_indian_6stars	R_phone	resto_rome_cheap_indian_6stars_phone
resto_rome_cheap_indian_6stars	R_cuisine	indian
resto_rome_cheap_indian_6stars	R_address	resto_rome_cheap_indian_6stars_address
resto_rome_cheap_indian_6stars	R_location	rome
resto_rome_cheap_indian_6stars	R_number	eight
resto_rome_cheap_indian_6stars	R_price	cheap
6	R_rating	resto_rome_cheap_indian_6stars
resto_rome_cheap_indian_7stars	R_phone	resto_rome_cheap_indian_7stars_phone
resto_rome_cheap_indian_7stars	R_cuisine	indian
resto_rome_cheap_indian_7stars	R_address	resto_rome_cheap_indian_7stars_address
resto_rome_cheap_indian_7stars	R_location	rome
resto_rome_cheap_indian_7stars	R_number	eight
resto_rome_cheap_indian_7stars	R_price	cheap
7	R_rating	resto_rome_cheap_indian_7stars

(a) Pre-Processed bAbI Data

Figure A.1: Pre-processing of bAbI dialog data used in Mem2Seq paper

Mem2Seq paper used the following pre-processing on the data:

1. The subject (restaurant name) and object (rating) positions of the rating KB tuples in bAbI dialogs are flipped, while the order remains the same for other tuples remains the same. This pre-processing is illustrated in Figure A.1

2. an extra fact was added to the navigation tasks in In-Car Assistant with all the properties (such as distance, address) combined together as the subject and *poi* as the object. This pre-processing is illustrated in Figure A.2

© 2019, *Indian Institute of Technology Delhi*

(a) Original SMD Navigate Data

Subject	Predicate	Object
the_westin	distance	2_miles
the_westin	traffic_info	moderate_traffic
the_westin	poi_type	rest_stop
the_westin	address	329_el_camino_real
toms_house	distance	1_miles
toms_house	traffic_info	heavy_traffic
toms_house	poi_type	friends_house
toms_house	address	580_van_ness_ave

Subject	Predicate	Object
2_miles moderate_traffic rest_stop	poi	the_westin
the_westin	distance	2_miles
the_westin	traffic_info	moderate_traffic
the_westin	poi_type	rest_stop
the_westin	address	329_el_camino_real
1_miles heavy_traffic friends_house	poi	toms_house
toms_house	distance	1_miles
toms_house	traffic_info	heavy_traffic
toms_house	poi_type	friends_house
toms_house	address	580_van_ness_ave

(a) Pre-Processed SMD Navigate Data

Figure A.2: Pre-processing of SMD Navigate data used in Mem2Seq paper

The pre-processing has major impact on the performance of Mem2Seq, as it can only copy objects of a KB tuple, while the subject and relation can never be copied.

A.2.2 bAbI Dataset Faults

The KB entities present in validation and non-OOV test sets for task 3 and 4 do not overlap with those in the train set. This effectively means that non-OOV and OOV test conditions are the same for tasks 3 and 4. This explains the low performance of baseline models on task 3 and 4 non-OOV test sets.

A.3 Example Predictions of BoSsNet and Baselines

Examples from SMD is shown in Table A.10 respectively. Examples from KA test set with percentage of unseen entites set to 50 from CamRest and SMD are shown in Table A.9 and

© 2019, *Indian Institute of Technology Delhi*

Table A.11 respectively. Examples from KA test set with percentage of unseen entites set to 100 from bAbI dialog Task 1 is shown in Table A.8.

usr-1	good morning
sys-1	hello what can i help you with today
usr-2	can you book a table with *korean* cuisine for *five* in *beijing*
sys-2	i ' m on it . which price range are looking for ?
usr-3	i am looking for a *overpriced* restaurant
Gold	api_call *korean beijing five overpriced*
Seq2Seq+Copy	api_call *korean beijing five overpriced*
Seq2Seq	api_call *french london four moderate*
Mem2Seq	api_call *overpriced korean korean overpriced*
BoSsNet	api_call *korean beijing five overpriced*

Table A.8: Example from bAbI dialog Task 1 with 100% OOV.

KB (restaurant|cuisine|address|phone|postcode) *prezzo|ethiopian|22_ atlantis_ road|01799_ 521260|cb30ad*	
usr-1	i want a *moderately_ expensive ethiopian* restaurant in the *south* side of town
sys-1	i ' m sorry . i can not find a *moderately_ expensive ethiopian* restaurant in the *south* side of town
usr-2	i would like to find a *moderately* priced restaurant in the *west* part of town
sys-2	*prezzo* is a moderately priced *ethiopian* restaurant in the *west* part of town . there is also *indian* and *british* food available in that area and price range . would you like more info ?
usr-3	yes can i get thier address and postcode ?
Gold	of course , *prezzo* ' s address is *22_ atlantis_ road* and postcode is *cb30ad*
Seq2Seq+Copy	postcode a is is *01223350688* UNK there get i for for today
Seq2Seq	the address is *83_ regent_ street* and their postcode is *cb58aq*
Mem2Seq	the phone address is *cb21uj*
BoSsNet	the address is *22_ atlantis_ road at 22_ atlantis_ road* the postcode is *cb30ad* is there anything else i can help you with

Table A.9: Example from Camrest with 50% OOV. The OOV entities present in the dialog are {*ethiopian*, *22_ atlantis_ road*}

© 2019, *Indian Institute of Technology Delhi*

KB (location|distance|traffic info|address) *teavana|4_ miles|no_ traffic|145_ amherst_ st* *cafe_ venetia|5_ miles|moderate_ traffic|269_ alger_ dr* *town_ and_ country|3_ miles|moderate_ traffic|383_ university_ ave*	
usr-1	where can i get tea ?
sys-1	there is 2 tea shops near you *teavana* is *4_ miles* away and *cafe_ venetia* is *5_ miles* away which one do you want ?
usr-2	which one is the quickest is there any traffic ?
Gold	*teavana* is the fastest journey because there is *no_ traffic* on the way there . *cafe_ venetia* has *moderate_ traffic* should we go to *teavana*
Seq2Seq+Copy	there is
Seq2Seq	there is *moderate_ traffic* on the way to *palo_ alto_ medical_ foundation*
Mem2Seq	there is *moderate_ traffic* on our way
BoSsNet	*teavana* there is a *moderate_ traffic* nearby but i might find another route to t*own_ and_ country town_ and_ country* address is *383_ university_ ave* i sent it on your screen

Table A.10: Example from SMD

KB (location|day|weather) *mountain_ view|sunday|foggy* *mountain_ view|monday|overcast* *pittsburgh|sunday|overcast* *pittsburgh|monday|warm* *downtown_ chicago|sunday|foggy* *downtown_ chicago|monday|warm*	
today|monday	
usr-1	is the weather *cloudy* right now ?
sys-1	what city do you want the weather for ?
usr-2	*pittsburgh*
Gold	there are no clouds in *pittsburgh* right now
Seq2Seq+Copy	which is no
Seq2Seq	it is currently raining in *san_ francisco* right now
Mem2Seq	it will not be cloudy in in
BoSsNet	it will be *cloudy* on *sunday* in *pittsburgh*

Table A.11: Example from SMD with 50% OOV. The OOV entity present in the dialog is {*pittsburgh*}

© 2019, *Indian Institute of Technology Delhi*

Appendix B

IMPORTANT CODE SNIPPETS

B.1 Combining the Generate and Copy Distribution

```
def _calc_final_dist(self, next_outputs, attn_dists, p_gens, oov_ids,
    oov_sizes, decoder_vocab_size, batch_size):
  """Calculate the final distribution, for the pointer-generator model

  Args:
    vocab_dists: The vocabulary distributions. (batch_size, vsize)
    attn_dists: The attention distributions. (batch_size, attn_len)
    p_gens : The soft switch weight for copy and generate probabilities
    oov_ids : List of ids of the OOV words in the context
    oov_sizes : Size of each OOV set for each example
    decoder_vocab_size : Size of deocde vocab = vsize
    batch_size : Number of examples in the input

  Returns:
    final_dists: The final distributions. List length max_dec_steps of
     (batch_size, extended_vsize) arrays.
  """
  with tf.variable_scope('final_distribution'):
    # Obtain Weighted Vocab Distribution
    vocab_dists = tf.nn.softmax(next_outputs.rnn_output)
    vocab_dists = tf.multiply(vocab_dists, p_gens)

    # Obtain Weighted Attention Distribution
    def one_minus_fn(x): return 1 - x
    p_gens = tf.map_fn(one_minus_fn, p_gens)
    attn_dists = tf.multiply(p_gens, attn_dists)

    # Concatenate some zeros to each vocabulary dist, to hold the
     probabilities for in-article OOV words
    max_oov_len = tf.reduce_max(oov_sizes, reduction_indices=[0])
    extended_vsize = decoder_vocab_size + max_oov_len
    extra_zeros = tf.zeros((batch_size, max_oov_len))
    vocab_dists_ext = array_ops.concat([vocab_dists, extra_zeros], axis=1)

    # Create Index which maps each word in the context to a particular cell
    batch_nums = tf.range(0, limit=batch_size)
    batch_nums = tf.expand_dims(batch_nums, 1)
    attention_ids = tf.reshape(oov_ids, [batch_size, -1])
```

```
    attn_len = tf.shape(attention_ids)[1]  # number of states we attend over
    batch_nums = tf.tile(batch_nums, [1, attn_len])
    indices = tf.stack((batch_nums, attention_ids), axis=2)

    # We project the attention weights to each index cell using scatter_nd
    shape = [batch_size, extended_vsize]
    attn_dists_projected = tf.scatter_nd(indices, attn_dists, shape)

    # Add the vocab distributions and the copy distributions together to get
    the final distributions
    final_dists = math_ops.add(vocab_dists_ext, attn_dists_projected)
    return BasicDecoderOutput(final_dists, next_outputs.sample_id)
```

B.2 Multi-Level Attention Mechanism

```
def _luong_score(query, keys, scale):
  """Implements Luong-style (multiplicative) scoring function at Sentence
   Level

  Args:
    query: Tensor, shape `[batch_size, num_units]` to compare to keys.
    keys: Processed memory, shape `[batch_size, max_time, num_units]`.
    scale: Whether to apply a scale to the score function.

  Returns:
    A `[batch_size, max_time]` tensor of unnormalized score values.

  Raises:
    ValueError: If `key` and `query` depths do not match.
  """
  # Check key and query depths
  depth = query.get_shape()[-1]
  key_units = word_keys.get_shape()[-1]
  if depth != key_units:
    raise ValueError("Size of query and key don't match")

  query = array_ops.expand_dims(query, 1)

  # Calculate Attention
  score = math_ops.matmul(query, keys, transpose_b=True)
  score = array_ops.squeeze(score, [1])

  return score

def _luong_word_score(query, word_keys, scale, batch_size):
  """Implements Luong-style (multiplicative) scoring function at Word Level

  Args:
```

© 2019, *Indian Institute of Technology Delhi*

```
33      query: Tensor, shape `[batch_size, num_units]` to compare to keys.
34      word_keys: Processed word level memory, shape `[batch_size, max_time,
        num_units]`.
35      scale: Whether to apply a scale to the score function.
36
37    Returns:
38      A `[batch_size, max_time]` tensor of unnormalized score values.
39
40    Raises:
41      ValueError: If `key` and `query` depths do not match.
42    """
43    # Check key and query depths
44    depth = query.get_shape()[-1]
45    key_units = word_keys.get_shape()[-1]
46    if depth != key_units:
47      raise ValueError("Size of query and key don't match")
48
49    query = array_ops.expand_dims(query, 1)
50
51    # Optimisation to transpose over 3D tensor instead of 4D tensor.
52    save_shape = word_keys.get_shape().as_list()
53    word_keys_small = tf.reshape(word_keys, [batch_size, -1, save_shape[2]*
        save_shape[3]])
54    word_keys = tf.transpose(word_keys_small, [1, 0, 2])
55    word_keys = tf.reshape(word_keys, [-1, batch_size, save_shape[2],
        save_shape[3]])
56
57    # Calculate Attention
58    context_fn = lambda key: array_ops.squeeze(math_ops.matmul(query, key,
        transpose_b=True), [1])
59    scores = tf.map_fn(context_fn, word_keys)
60    scores = tf.transpose(scores, [1, 0, 2])
61
62    return scores
```

© 2019, *Indian Institute of Technology Delhi*

REFERENCES

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] J. Berant and P. Liang. Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558, 2015.

[3] A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016.

[4] A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. In *International Conference on Learning Representations*, 2017.

[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] M. Eric, L. Krishnan, F. Charette, and C. D. Manning. Key-value retrieval networks for task-oriented dialogue. In *Dialog System Technology Challenges, Saarbrücken, Germany, August 15-17, 2017*, pages 37–49, 2017.

[7] M. Eric and C. D. Manning. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *arXiv preprint arXiv:1701.04024*, 2017.

[8] J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

[9] C. Gulcehre, S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149. Association for Computational Linguistics, 2016.

[10] A. Kannan, K. Kurach, S. Ravi, T. Kaufman, B. Miklos, G. Corrado, A. Tomkins, L. Lukacs, M. Ganea, P. Young, and V. Ramavajjala. Smart reply: Automated response suggestion for email. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (2016).*, 2016.

[11] D. P. Kingma and J. L. Ba. Adam: Amethod for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations*, 2014.

[12] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387, 2016.

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

[14] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *55th Annual Meeting of the Association for Computational Linguistics, ACL 2017*, pages 23–33. Association for Computational Linguistics (ACL), 2017.

[15] C. Liang, M. Norouzi, J. Berant, Q. V. Le, and N. Lao. Memory augmented policy optimization for program synthesis and semantic parsing. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9994–10006. Curran Associates, Inc., 2018.

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

[17] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics, 2015.

[18] A. Madotto, C. Wu, and P. Fung. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *Proceedings of 56th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2018.

[19] H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[20] R. Nallapati, B. Zhou, C. dos Santos, C. Gulcehre, and B. Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290. Association for Computational Linguistics, 2016.

[21] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on as-*

© 2019, *Indian Institute of Technology Delhi*

sociation for computational linguistics, pages 311–318. Association for Computational Linguistics, 2002.

[22] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[23] A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083. Association for Computational Linguistics, 2017.

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

[25] I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*, pages 3776–3784, 2016.

[26] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

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

[28] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[29] O. Vinyals and Q. Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[30] T. Wen, D. Vandyke, N. Mrkšíc, M. Gašíc, L. Rojas-Barahona, P. Su, S. Ultes, and S. Young. A network-based end-to-end trainable task-oriented dialogue system. In *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017-Proceedings of Conference*, volume 1, pages 438–449, 2017.

[31] T.-H. Wen, M. Gasic, N. Mrkšić, L. M. Rojas Barahona, P.-H. Su, S. Ultes, D. Vandyke, and S. Young. Conditional generation and snapshot learning in neural dialogue systems. In *EMNLP*, pages 2153–2162, Austin, Texas, November 2016. ACL.

[32] J. D. Williams, K. Asadi, and G. Zweig. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 665–677, 2017.

© 2019, *Indian Institute of Technology Delhi*

[33] J. D. Williams and S. Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007.

[34] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[35] C.-S. Wu, A. Madotto, G. Winata, and P. Fung. End-to-end recurrent entity network for entity-value independent goal-oriented dialog learning. In *Dialog System Technology Challenges Workshop, DSTC6*, 2017.

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

[37] W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015.

[38] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

© 2019, *Indian Institute of Technology Delhi*

CITATIONS

1. Yavuz et Al. **DEEPCOPY: Grounded Response Generation with Hierarchical Pointer Networks** *NIPS* (2018).

2. Ebrahimi et Al. **Reasoning over RDF Knowledge Bases using Deep Learning** *arXiv* (2018).

3. Singh et Al. **Towards VQA Models That Can Read** *CVPR* (2019).

4. Golchha et Al. **Courteously Yours: Inducing courteous behavior in Customer Care responses using Reinforced Pointer Generator Network** *NAACL-HLT* (2019).