Self-Attentional Models Application in Task-Oriented Dialogue Generation Systems

by

Mansour Saffar Mehrjardi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Dialogue generation systems (chatbots) are currently one of the most noted topics in natural language processing, and many companies are investing intensively on creating such systems for automation of tasks. Given the notable success of deep learning methods in the creation of state-of-the-art models in other domains such as computer vision, recommender systems, and natural language processing, there has been in recent years a surge in creating chatbot systems using deep learning methods. In the case of dialogue generation, deep learning provides methods to bootstrap a large corpus of data with minimal feature engineering, enabling the chatbot systems to learn the feature representation from data and create coherent and well-structured responses.

Task-oriented chatbots are a type of dialogue generation system which tries to help the users accomplish specific tasks, such as booking a restaurant table or buying movie tickets, in a continuous and uninterrupted conversational interface and usually in as few steps as possible. Given the abundance of such systems in industry and how lucrative chatbots can be in saving cost and increasing user satisfaction, we decided to focus our study on this type of chatbot systems.

Self-attentional models are a new deep learning paradigm for sequence modelling tasks. These models differ from the previous sequence modelling methods, e.g. models based on recurrent neural networks and convolutional neural networks, in a way that their architecture is only based on the attention mechanism. Transformer was the first self-attentional model which was re-

cently proposed by Google Brain and could beat the state-of-the-art models in neural machine translation tasks. This fact motivated us to explore the usage of self-attentional models for the task of creating task-oriented chatbots. We compare common sequence modelling models with the self-attentional models and provide a comprehensive analysis. Our results show that self-attentional models can create task-oriented chatbots with higher BLEU score compared to recurrent neural networks and also with faster training time.

Lastly, we address the problem of evaluating chatbots by implementing a user-simulation based evaluation method for task-oriented chatbots. Our evaluation method is a pipelined user simulator, i.e. containing natural language understanding, user simulator, and natural language generation components, similar to Microsoft's user simulators used to train reinforcement learning agents but conditioned on user profile and characteristics which make it more realistic.

If a machine is expected to be infallible, it cannot also be intelligent.

– Alan Turing, 1912-1954

Acknowledgements

I would like to begin by expressing my sincere gratitude to my supervisor Prof. Osmar Zaiane. Without his guidance and support, I would not have been able to accomplish fulfilling my masters goals.

Contents

List of Tables

List of Figures

Acronyms

ACT Adaptive Computation Time

AI Artificial Intelligence

AIML Artificial Intelligence Markup Language

ALICE Artificial Linguistic Internet Computer Entity

AMT Amazon Mechanical Turk

BLEU Bilingual Evaluation Understudy

BOW Bag of Words

BPTT Backpropagation Through Time

CNN Convolutional Neural Network

ConvS2S Convolutional Sequence-to-Sequence

CRF Conditional Random Field

DCN Deep Convex Network

DSTC Dialogue State Tracking Challenge

FAQ Frequently Asked Questions

GPU Graphical Processing Unit

GRU Gated Recurrent Unit

HMM Hidden Markov Model

KB Knowledge Base

LSTM Long Short-term Memory

M2M Machines Talking to Machines

MLP Multi-Layer Peceptron

NER Named-entity Recognition

NLG Natural Language Generation

NLI Natural Language Inference

NLP Natural Language Processing

NMT Neural Machine Translation

NLU Natural Language Understanding

POS Part-Of-Speech

RELU Rectified Linear Unit

RNN Recurrent Neural Network

SMB Small and Medium Business

SRL Semantic Role Labeling

SVM Support Vector Machine

WOZ Wizard of Oz!

Chapter 1

Introduction

Conversational agents are commonly one of the hot topics in AI community within both academia and industry. The prospect of conversational agents is of great interest to many companies including Google, Microsoft, Amazon, Apple, Alibaba, and many others. The AI-powered conversational agents have a wide range of applications which lead to the emergence of fields such as Conversational AI and Conversational Commerce [78]. We can categorize these AI-powered dialogue generation systems into two main categories of task-oriented and open-domain chatbots. Task-oriented chatbots aim to help the user complete a specific task. Google's Duplex [65] is an example of a task-oriented chatbot which helps users make reservations by calling the stores on behalf of the user. On the other hand, open-domain chatbots aim to have a conversation with the user covering a wide range of topics and keep the user in conversation as long as possible. A great example of an open-domain chatbot is Microsoft's Xiaoice [123] which has about 660 million users online. In this study, we only focus on task-oriented chatbots.

The general architecture of task-oriented chatbots falls into two major design architectures, namely pipelined and end-to-end [14]. The pipelined architecture is made up of different components for language understanding, decision making, and creating a human-understandable response, each of them can be trained individually and then plugged into the chatbot, while the end-to-end architectures try to achieve the same goal in a single network in which all component are trained together. There has been a wide number of research papers implementing variations of sequence-to-sequence models for the development of end-to-end networks and most of them rely on sequence-to-sequence architectures based on recurrent neural networks (RNNs). Transformer [109] is the new model proposed by Google AI team which is purely attention based and has beat the state-of-the-art for machine translation tasks. Furthermore, Universal Transformer [22] has been recently proposed which works well for language understanding tasks on which the Transformer model could not perform well. We will explore these two newly proposed models for the task of creating task-oriented chatbots and provide a comprehensive comparison between some of the most commonly used models used for training task-oriented chatbots and the two aforementioned models.

One of the main challenges for the researchers working on conversational agents is the evaluation of these models [69]. The evaluation methods for open-domain and task-oriented chatbots are usually different due to the fact that these chatbots have different goals. The goal for open-domain chatbots is to keep the user engaged as much as possible while the goal for the task-oriented one will be to get the specified task done both successfully and usually with the minimum number of conversation turns. Furthermore, current automatic evaluation metrics for task-oriented chatbots make the comparison of the pipelined and end-to-end trained chatbots hard since they do not take semantics of the sentences into consideration and also encourage the models to follow a specific flow of dialogue while there are many ways a task-oriented dialogue can be formed. We propose a new evaluation method for task-oriented chatbots by implementing a user simulator [93] which is conditioned on the user profile [95] and thus make it possible to simulate different user behaviours. The proposed evaluation method will also make it easier to compare pipelined and end-to-end trained chatbots.

1.1 Motivation

Given the success of deep learning models in other fields such as computer vision, there has been a significant increase in the usage of deep learning methods for different NLP tasks which have lead to the creation of state-of-the-art results in many of those domains. One of the NLP domains in which deep learning models have significant contribution is Conversational AI in which researchers and machine learning engineers have used deep learning methods for the task of creating conversational agents which can interact with people via auditory or textual methods. Conversational AI and Conversational Commerce [26] are new trends in industry and academia which try to address industry problems using the conversational agents. Task-oriented chatbots are one of the main types of conversational agents which are adopted by many Small and Medium-sized Businesses (SMBs) and large corporations to provide a better user experience for their customers.

Deep learning methods for dialogue generation systems make it possible to take advantage of conversations corpora such as chat logs or forum discussions and make the chatbot model learn directly from data with minimal feature engineering. Transformer and Universal Transformer are amongst the deep learning methods that have created state-of-the-art in some NLP tasks. Their usage for task-oriented dialogue generation systems is not yet explored comprehensively and one of the main contributions of this thesis to provide such a comprehensive comparison and analysis. Furthermore, evaluation of dialogue generation systems is still a controversial issue amongst the research community and even some researchers compared current evaluation methods of dialogue generation systems with human evaluation and demonstrated the deficiencies of current evaluation methods for dialogue generation systems [69]. Using a user simulator which can model user behaviour and profile can be a good approach for both training and evaluations of chatbots and one of the contributions of this study is to show that such method can provide reliable evaluations for task-oriented chatbots.

1.1.1 Conversational AI and Conversational Commerce

Conversational AI is the science of developing agents which are able to communicate with humans in a natural way [88]. Digital assistants such as Apple's Siri, Google Assistant, Amazon Alexa, and Alibaba's AliMe are examples of successful chatbots developed by large companies to engage with their customers. Most of these chatbots are generally task-oriented and serve a limited range of tasks, such as setting an alarm or typical information retrieval tasks. The goal for researchers both in academia and industry is to augment these chatbots with capabilities to have a chitchat with users but there is still a long way to reach the chatbot which is able to have natural everyday conversations with humans [64].

Conversational Commerce is another recently high-profile field that tries to tackle the business problem using Conversational AI and tries to connect shoppers and brands via textual or voice interface. Most of the chatbots developed in this field are usually task-oriented and sometimes have basic chitchat capabilities, and their final goal is to try to help users have a better experience while shopping for items online. Conversational Commerce is enabling brands to have to the long-desired one-on-one conversation and engagement with their customers in a continuous and uninterrupted way.

1.1.2 Applications of Chatbots

Chatbots usually fall into the open-domain and task-oriented groups. The goal for open-domain chatbots is to have a chitchat conversation with the users, and the user can talk about many different topics with the chatbot. The user can ask about the weather, and the chatbot must be able to not only answer appropriately but also has to accompany the answer to weather query with some response that could potentially lead to a more engaging conversation with the user. For example, if the user asks about the weather in Edmonton, a possible good answer from an open-domain chatbot would be: "*It's a little bit chilly outside. You may want to stay home and watch some horror movies instead!*". On the other hand, a task-oriented chatbot would answer this question with a response like "*Right now in Edmonton it's -10 Celsius and sunny*" which is not an engaging answer possibly leading to a prolonged conversation with the user.

Given the brief comparison between functionality and general purpose of task-oriented and open-domain chatbots, it is important to note that most of the chatbots developed by SMBs and large corporations are task-oriented. Given the complexity of open-domain chatbots, they are still far from the current capabilities of current techniques used in building dialogue generation systems [110]. However, researchers in academia and industry are trying to augment the capabilities of the task-oriented chatbots with limited chitchat functionality. This additional functionality makes the users want to have longer chats with the chatbot because of its more human-like interaction. Microsoft's Xiaoice [123] is a good example of how users get much more engaged when they feel that they have become friends with the chatbot. Amazon's Alexa Prize competition [88] is a well known competition which tries to promote the development of such chatbots.

Applications of task-oriented chatbots cover a wide range of domains. Typical examples are reservation chatbots, Frequently Asked Questions (FAQ) chatbots, and customer-service chatbots. Deployment of task-oriented chatbots could have lucrative advantages for most SMBs and also giant corporations. The reductions of cost for human labour in simple question answering tasks, getting insights and feedback from customer engagement with a chatbot, and better customer experience given the continuous and uninterrupted availability of a chatbot service are amongst the most important advantages of using a task-oriented chatbot. For example, consider the automation of a call centre using a task-oriented chatbot. A well-trained task-oriented chatbot can handle most of the FAQs by callers and also is able to redirect to an agent if it notices that cannot handle the call. This could reduce the number of employees in the call centre noticeably since most of the calls can be handled by the chatbot. Google Duplex [65], a chatbot service able to make calls on behalf of the user for mostly reservation tasks, eBay's ShopBot [86], a personal shopping assistant, and Automat's BeautyBot [71] are examples of successful task-oriented chatbots.

Chatbots can also be used for applications related to health. Chatbots developed in this domain are not necessarily as task-oriented one and tend to have more engaging interactions with the user by having task-related chitchats with the user but are not still considered open-domain chatbots. Babylon

Health [90] tries to automate the process of initial diagnosis and triage through a conversational interface. This can help many patients who have difficulty in accessing a doctor get online medical consultation and get medical assistance in an urgent situation. Woebot [29] is another use case of conversational agents in the health domain which tries to help users with their depression and provide therapy based on psychological methods.

1.1.3 Deep Learning for Dialogue Generation Systems

Deep learning methods try to learn a hierarchical representation of data which makes them a strong candidate for many NLP tasks such as Part-Of-Speech (POS) tagging, parsing, named-entity recognition (NER), question answering, and language understanding tasks. The power of deep learning in learning a meaningful representation of data without using much manual hand-crafted feature engineering makes it a strong candidate for developing dialogue generation systems [129][14].

Deep learning plays an important role in the development of both open-domain and task-oriented chatbots. Sequence-to-sequence learning methods using neural networks [102] have created state-of-the-art results in dialogue generation systems. Given the complexity of a chatbot which has to understand the user utterance, process the information given by the user and integrate it with other relevant information and at last create an appropriate response which is relevant and informative, makes the usage of deep learning methods in dialogue generation systems more appealing. Traditionally hand-crafted rules were used for both understanding user utterances and the response generation mechanism. Early chatbots like Eliza [114] and Parry [12] are examples of chatbots which use hand-crafted scripts to communicate with humans. Deep learning methods make the process of creating chatbots easier in a way that there is minimal need for usage of hand-crafted rules and deep learning methods can learn directly from the conversation's data.

Dialogue generation systems (both open-domain and task-oriented) can be modeled as a sequence modeling task in which the deep learning model is provided with the conversation corpus and learns the dependencies between the tokens and concepts in the corpus in order to be able to generate relevant and appropriate responses. The learning task is usually formed in a sequence-to-sequence learning method in which the input sequence is the context of the conversation up to some point in the past, and the target is the response which system should generate. The state-of-the-art deep learning models for sequence modeling have three main architectures which are recurrence-based, convolution-based, and self-attentional models which we will elaborate more in Chapter 2. The self-attentional models (Transformer and Universal Transformer) are relatively new models which purely work based on attention mechanism and have shown great results in a number of NLP tasks. We will explore the usage of these models for the task of task-oriented dialogue generation systems and provide a comprehensive comparison between these architectures for this specific task.

1.1.4 Self-attentional Models for Sequence Modeling

Transformer [109] and Universal Transformer [22] are self-attentional deep learning models for sequence modeling which purely rely on self-attention mechanism (see Chapter 2.3). The Transformer model could beat state-of-the-art model in neural machine translation (NMT) tasks and achieves a 2 points increase in BLEU score which is a significant performance increase in NMT task. The self-attention mechanism allows the model to learn long-term dependencies and learn different concepts available in the corpus in a way that the model was able to handle the coreference resolution [131] problem in the task of NMT [106]. The usage of the self-attention mechanism in the architecture of the Transformer model allows high parallelization capability to the model which can be trained faster compared to recurrence-based models.

Universal Transformer [22] is a variation of the Transformer model which tries to introduce a recurrence to the model so that it can perform better in some language understanding tasks. The original Transformer model did not have any recurrence and was not able to perform well in some language understanding tasks compared to recurrence-based models such as RNNs. The Universal Transformer model extends the Transformer model by applying the recurrence method not time-wise similar to RNN model, but token-wise which means the recurrence is applied on the different representation of a token in the different layers of the Universal Transformer and the model can decide where to stop the recurrence. Methods such as Adaptive Computation Time [37] are applied to the Universal Transformer model to help the model perform token-wise recurrence computation more efficiently and faster [99].

1.1.5 Evaluation of Task-oriented Chatbots

One of the main challenges that researchers and engineers face in the domain of dialogue generation systems is the evaluation of these systems. Due to the fact that these systems are supposed to act like humans and create an engaging and context-relevant response, it is challenging to automatically evaluate these systems. The evaluation metrics and methods used for the evaluation of task-oriented and open-domain chatbots have slight difference due to the fact that their final purpose is different. Task-oriented chatbots are supposed to complete a specific task in as few turns as possible and provide relevant responses while open-domain chatbots are intended to keep the user engaged and have a long conversation with the user.

The evaluation methods and metrics used in sequence modeling tasks such as BLEU [82] which is usually used in NMT tasks is also used in chatbot evaluation. The other methods that are usually in the evaluation of task-oriented chatbots are per-turn accuracy, per-dialogue accuracy, and entity F1 score. All of these metrics try to pick the best model which is able to output responses which are exactly similar to the training corpus and even if the chatbot generates another relevant response, which is different from what it was supposed to generate according to the training corpus, it will be penalized. This process can be alleviated through methods such as beam search [30][125] but still will not be solved entirely and needs more research.

All the aforementioned methods for the evaluation of task-oriented chatbots do not model the user error and characteristic in the evaluation process. Using a user-profile conditioned user simulator, which is able to simulate real users with different characteristics, seems a promising approach in order to better evaluate the task-oriented chatbots. We create such an evaluation method using the user simulator, natural language understanding (NLU), and natural language generation (NLG) modules and evaluate the trained chatbots by measuring metrics such as the number of turns in the conversation in order to provide an evaluation method which models the user characteristic in the process of evaluation.

1.2 Problem Formulation

In this study we try to address two problems. One is to investigate the usage of self-attentional models for creating end-to-end task-oriented chatbots. We setup a number of experiments on different datasets to check whether we can use the self-attentional models for this task and also benchmark their performance against state-of-the-art sequence modelling methods.

The second problem we address in this study is to evaluate the usage of profile-conditioned user simulator for assessment of task-oriented chatbots. As mentioned before, the goal for task-oriented chatbots is to complete the user task in as few turns as possible and thus we can formulate this problem into an optimization problem in which we try to optimize the number of turns in a task-oriented conversation with respect to a limited number of turns as our constraint.

1.3 Thesis Statement

In this study, we investigate the Transformer and Universal Transformer models for the aim of creating task-oriented chatbots. We also propose a pipelined evaluation method for task-oriented chatbots using a user simulator conditioned on the user profile. We claim that the self-attentional models can be used to train end-to-end task-oriented chatbots with comparable accuracy to recurrence-based sequence modelling methods and often in a faster way. We also claim that a profile-conditioned user simulator can be used as an evaluation method for task-oriented chatbots.

1.4 Thesis Contribution

The contributions of this study are in two categories. First, we use the Transformer and Universal Transformer models for the purpose of training task-oriented chatbots and provide comprehensive comparison and analysis for each of the models. In the second part, we propose a new evaluation method for task-oriented chatbots by implementing a user simulator which takes user profile (characteristics) into account. A detailed description of each of the contributions is as follows:

- **Transformer and Universal Transformer Models**: We train and evaluate task-oriented chatbots for each of these models and compare them with current state-of-the-art recurrence-based sequence modelling methods for developing task-oriented chatbots. We use three different sets of datasets from the restaurant and movie domain and evaluate the trained chatbots.

- **User Simulator Conditioned on User Profile**: We use the idea of Agenda-based user simulator [93] and combine it with the user profile in order to make the user simulator more realistic. We train an NLU module and also define a template-based NLG module for the proposed evaluation method in order to make it able to talk to chatbots and respond accordingly. This pipelined method can be used as an evaluation method for task-oriented chatbots.

1.5 Organization of the Dissertation

For each of the contributions mentioned in 1.4 we have a chapter. In Chapter 2 we will discuss the different deep learning architectures used in sequence modeling tasks. We also have a background chapter (Chapter 3) about methods for developing task-oriented chatbots and creating task-oriented training corpus. In Chapter 4 we focus on the evaluation of task-oriented chatbots and discuss the proposed evaluation methods. In Chapter 5, we will demonstrate the experiment results for both the self-attentional models usage in task-oriented chatbots and also the profile-conditioned user simulator. Chapter 6, conclusion and future work, addresses the possible ways to add contributions to this thesis. We also provide some examples of data preprocessing procedure in Appendix A and also explain the details of a rule-based chatbot that we use in our experiment in Appendix B. Having this explanation in mind, the six chapters in this study are as follows:

- **Chapter 1.** ***Introduction***: In this chapter we'll provide some background on Conversational AI and its impact on businesses. We also explain how current deep learning methods can help us create intelligent, context-aware chatbots and briefly mention our contributions in this study.

- **Chapter 2.** ***Preliminaries for Sequence Modeling with Self-attentional Models***: Detailed description of recurrent and convolutional methods for sequence modeling tasks, Detailed description, and analysis of Transformer and Universal Transformer models and their use cases for natural language processing (NLP) tasks.

- **Chapter 3.** ***Task-oriented Dialogue Generation Systems***: An overview of methods used for development of task-oriented chatbots for both end-to-end and pipelined architectures. We also discuss the different methods used in creating the training corpus for task-oriented chatbots.

- **Chapter 4.** ***Evaluation of Task-oriented Dialogue Generation Systems***: An overview of metrics and methods used for the evaluation of task-oriented chatbots alongside the detailed description of implementing the profile-conditioned user simulator evaluation method.

- **Chapter 5.** ***Experiments and Results***: Training and data preparation process for training task-oriented chatbots is explained in this chapter. Experimental results of trained models are also included in this chapter.

- **Chapter 6.** ***Conclusion and Future Work***: In this chapter we explain what we learned and explain what are the possible ways to extend this study for future research.
- **Appendix A.** ***Data Preparation Process:*** A few samples of both raw and processed task-oriented dialogues to demonstrate the data preparation process for training end-to-end task-oriented chatbots.
- **Appendix B.** ***Rule-based Chatbot:*** Sample of a rule-based chatbot we used in our experiments.

Chapter 2

Preliminaries for Sequence Modeling with Self-Attentional Models

Sequence modeling is the use of machine learning and deep learning models for probabilistic modeling of a sequenced data. Language modeling is one the core tasks of sequence modeling in which we try to find a model which can learn the joint probability distribution of the tokens in the sequenced data (text data). The language model can be used to predict the next token (e.g., word) given the current context. Sequence modeling is widely used in many NLP fields such as machine translation, dialogue generation, NER, and text summarization. The core idea behind the sequence modeling is to learn the joint probability distribution of the tokens (a generative model) which also can be used as a discriminative model and conditioned on the context to predict the next tokens.

Many neural-based architectures have been proposed for the task of sequence modeling in NLP, and the state-of-the-art models mainly fall into three main categories which are recurrence-based models (e.g. RNNs), convolution-based models (e.g., Convolutional Sequence-to-Sequence (ConvS2S)), and self-attentional models (e.g., Transformer). In the recurrence-based model, the sequence is fed to the model sequentially, and the model tries to learn the dependencies between the tokens using the backpropagation through time (BPTT) method [118]. The convolution-based models try to use 1-D convolution in order to model the sequence in a hierarchical approach in which multiple layers of convolution are stacked on top of each other in order to model the whole sequence. Their architecture also allows high parallelization which results in faster training compared to recurrence-based models. The main problem of convolution-based models is that the convolution is done position-based (regardless of whether the tokens in those positions are related to each other) and therefore some of the semantic concepts may be missed during the training and the model will not be able to learn those concepts. Self-attentional models, such as Transformer, try to address this problem by incorporating the self-attention mechanism in sequence modeling. The self-attention mechanism allows the model to learn the intra-dependencies between the tokens and assign (relevance) weights to them so that the models are able to learn all the concepts in the sequence. This idea is depicted in Figure 2.1 [108][52].

Figure 2.1: Convolution-based (left) vs self-attention-based (right) sequence modeling [109]

The Transformer model [109] was first used in neural machine translation tasks and could lead to state-of-the-art results with significant enhancement compared to previous models. The main difference between the Transformer model and the other models used for sequence-to-sequence modeling task is that the Transformer model only relies on the attention mechanism and there is no recurrence or convolution as in the previous models. The Transformer architecture makes it an ideal model for parallelizing and thus the training could be done faster compared to recurrent and convolutional models used for sequence modeling tasks. The Transformer model was able to beat state-of-the-art in tasks such as machine translation, but in some language understanding tasks, it could not beat recurrence-based models [22]. The Universal Transformer model [22] tried to tackle this problem by incorporating the recurrence into the Transformer model in a depth-wise way instead of time-based recurrence and hence has the parallelization capability of Transformer as well as the recurrence of models like RNNs.

In the following sections, we will discuss different architectures used in sequence modeling tasks and explain their architectures as well as the building blocks and methods used in these models. In Section 2.1 and 2.2 we will discuss the sequence modeling models which use recurrence and convolution in their architecture respectively. In Section 2.3 we discuss the attention mechanism and its variations and then in Section 2.4 we discuss the usage of self-attentional models for sequence modeling task and their applications in different machine learning tasks.

2.1 Recurrence-based Sequence Modeling

Using the recurrence for sequence modeling is the most intuitive way to build a model. In recurrence-based sequence modeling, the tokens are fed into the model in a sequential way, and the model learns the dependencies between the tokens given the context from the past (and the future in case of bi-directional RNNs). RNNs and their variations such as Long Short-term Memory (LSTM) and Gated Recurrent Units (GRU) are the most widely used recurrence-based models used in sequence modeling tasks. The recurrence-based models are trained using the BPTT method in which the errors are backpropagated through time up to some step in the past, and the gradients are calculated accordingly. The exploding and vanishing gradient [85] problems are among the main limitations of RNNs which do not allow the model to learn long-term dependencies. LSTMs alleviate this problem by changing the architecture of the model in a way that information can flow through the model from the past directly, and the model can decide what parts of the information from the past to keep. In Section 2.1.1 we will discuss the RNN model and how it is trained using BPTT. We also discuss the bi-directional architecture of RNN models and later in Section 2.1.2 we discuss the LSTM model and how its architecture can handle vanishing gradient problem and learn long-term dependencies.

2.1.1 Recurrent Neural Networks

Recurrent neural networks are a family of neural networks designed to process the sequential data. Given the sequence $X = x^1, x^2, ..., x^t$, the RNN model tries to learn dependencies between the given tokens of the sequence and create a joint probability distribution of the whole sequence. This is shown in Equation 2.1. As noted in this formula, the joint probability can also be written in the form of multiplication of conditional probabilities the RNN model tries to learn the joint distribution by computing the probability of the current token given all (or a number of) its previous tokens.

$$P(x^1, x^2, .., x^t) = \prod_{i=2}^{t} P(x^i \mid x^1, ..., x^{i-1})P(x^1) \tag{2.1}$$

All sequence modeling tasks try to learn such a joint probability distribution between the tokens of the given sequence data. RNN models [44] try to model such a distribution using a sequence of hidden states (cells). The basic equation of an RNN model (and also all recurrence-based models) is shown in Equation 2.2 [35].

$$h^t = f(h^{t-1}, x^t; \theta) \tag{2.2}$$

The hidden state of an RNN (i.e., h^t) is supposed to store all the information of the sequence model from the beginning up to time step t. For updating the hidden state h^t, we will pass all the information from the previous tokens which are stored in h^{t-1} and the current token x^t to some function f which can learn the joint probability distribution given in Equation 2.1. Note that in the ideal case the model should be able to store all the information from previous states in its hidden state h^t, but given that all of these models are trained in a differential method (using BPTT in case of recurrence-based models) the computational limitations of the cells and problems like vanishing/exploding gradients make it impossible to learn these dependencies for long input sequences.

To further expand Equation 2.2, we can unfold the basic RNN model and write Equation 2.2 for an unfolded RNN in Equation 2.3 [35],

$$
\begin{aligned}
a^t &= b + Wh^{t-1} + Ux^t \\
h^t &= tanh(a^t) \\
o^t &= c + Vh^t \\
\hat{y}^t &= softmax(o^t)
\end{aligned} \tag{2.3}
$$

where W, U, V, b, c are learnable parameters for the model and the $softmax$ function is defined as in Equation 2.4 and the $tanh$ function is defined as in Equation 2.5.

$$
softmax(z)_j = \frac{\exp(z_j)}{\sum_{j=11}^{n} \exp(z_j)} \tag{2.4}
$$

$$
tanh(x) = \frac{2}{1 + \exp{(-2x)}} - 1 \tag{2.5}
$$

Note that some RNNs have output tokens in some time steps and therefore $\hat{y}^t$ is used to create the output token(s).

2.1.2 LSTM and Bi-Directional LSTM

LSTM networks are a special kind of RNN networks which can learn long-term dependencies [45]. RNN models suffer from vanishing gradient problem [7] which makes it hard for RNN models to learn long-term dependencies. LSTM model tackles this problem by defining gating mechanism which introduces input, output and forget gates and the model has the ability to decide how much of the previous information it needs to keep and how much of the new information it needs to integrate and thus this mechanism help the model keep track of long-term dependencies. LSTM transition formula is depicted in Equations 2.6, 2.7, and 2.8 [39][80].

$$
\begin{aligned}
i^t &= sigmoid(W_i x^t + U_i h^{t-1} + b_i)(\text{input gate}) \\
o^t &= sigmoid(W_o x^t + U_o h^{t-1} + b_o)(\text{output gate}) \\
f^t &= sigmoid(W_f x^t + U_f h^{t-1} + b_f)(\text{forget gate})
\end{aligned} \tag{2.6}
$$

The equations in 2.6 depict the three main gates in the LSTM network. The input gate checks how much the new token x^t is worth regarding the information it carries compared to previous information (stored in hidden state h^{t-1}). The forget gate function is to determine how much of the information from the past (stored in hidden state h^{t-1}) is worth keeping. Finally, the output gate decides how much of the information from both hidden state h^{t-1} and current token x^t should be used in updating the current cell state C^t. The cell state acts like a channel in which information can pass and each LSTM cell tries to modify this information if needed. The update formulas for the cell states is shown in Equation 2.7. Note that the $\odot$ sign is the Hadamard multiplication [100] sign (i.e. point-wise multiplication).

$$\begin{aligned} \tilde{C}^t &= tanh(W_c x^t + U_c h^{t-1} + b_c)(\text{updated cell state}) \\ C^t &= f^t \odot C^{t-1} + i^t \odot \tilde{C}^t(\text{final cell state}) \end{aligned} \tag{2.7}$$

Finally, the hidden state of the LSTM network is updated according to equation 2.8 in which the output and the final cell state are used in order to determine the hidden state of current time. Note that in Equation 2.7, the previous hidden state and the current token are used in order to decide whether to update the current cell state $\tilde{C}^t$. Then the LSTM uses the information from the input gate (which is basically a scalar) in order to decide how much of the information from the update cell state should be kept and the information from the forget gate is used to decide how much of the information from the previous cell state has to be thrown away.

$$h^t = o^t \odot tanh(C^t)(\text{updated hidden state}) \tag{2.8}$$

This architecture in the LSTMs can help the model learn long-range dependencies given the fact that the information can travel through the cell state and the LSTM model decides how to tweak that information and whether to add more information given the current token to the cell state or whether to forget the information from the past. In theory, this model should be able to model very long sequences, but in practice, due to the differential training of these models, there is still a limitation on how much information can be kept.

Bi-directional LSTMs [94] are a variation of LSTMs which proved to give better results for some NLP task [38]. The idea behind a Bi-directional LSTM is to give the network (while training) the ability to not only look at past tokens as LSTM but to future tokens, so the model has access to information both form past and future. In the case of a task-oriented dialogue generation systems, in some cases, the information needed so that the model learn the dependencies between the tokens comes from the tokens that are ahead of the current index, and if the model is able to take future in time tokens into accounts it can learn more efficiently.

2.1.3 Gated Recurrent Unit

GRU [16] is a variation of the LSTM network in which instead of three gates, we have two gates called reset gate and update gate. GRUs are computationally more efficient and faster compared to LSTM models. The architecture of a GRM model can be seen in Equation 2.9. The input gate and the forget gate of the LSTM are merged and the update state is created. There is also a reset gate which decides how much of the information from the previous states should be forgotten. The cell state is modeled differently in GRU and the equation to update it is simpler [80].

$$\begin{aligned}
z^t &= sigmoid(W_z x^t + U_z h^{t-1})(\text{update gate}) \\
r^t &= sigmoid(W_r x^t + U_r h^{t-1})(\text{reset gate}) \\
\tilde{h}^t &= tanh(r^t \odot U h^{t-1} + W x^t) \\
h^t &= (1 - z^t) \odot h^t + z^t \odot h^{t-1}
\end{aligned} \tag{2.9}$$

2.2 Convolution-based Sequence Modeling

Sequence modeling usually relies on recurrent models such as RNNs since their architecture is designed to process a sequence of data while the convolutional neural networks (CNN) [63] are mostly used in image analysis. RNNs are designed to handle variable length sequences while CNNs are designed to handle images with different size [35]. However, it is possible to use CNNs for sequence modeling tasks, and in fact in some cases, CNNs could outperform certain RNNs architectures and produce better results. The main advantage of using CNNs instead of RNNs for sequence modeling is the fact that CNNs can leverage the parallelization power of GPUs better than RNNs and also since the number of non-linearities in the model is fixed and independent of sequence length, the optimization process is easier compared to RNNs [33].

RNNs need to process the input in a sequential way while the CNNs can process different chunks of the input concurrently and thus can parallelize this process. Note that CNNs do not take at the input sequentially, and therefore they are not able to model dependencies between tokens that are far away from each other. The solution to this problem is to stack multiple layers of CNN layer on top of each other, so long-range dependencies can be seen and processed by the model. The stacking of layers in CNNs for sequence modeling allows the model to grow its receptive field, or in other words context size, and thus can model complex dependencies between different sections of the input sequence. In multi-layer CNN sequence modeling architecture, nearby tokens interact in lower layers while tokens which are far from each other will interact in the higher level layers. The receptive field size can be increased in an exponential way using dilated convolutions [130] so the number of stacked layers in the CNN architecture needed to model long sequences will be lowered considerably as depicted in Wavenet architecture in Figure 2.2 (it will be downsized in a logarithmic way).

Another advantage of using CNNs is that they can look at the whole sequence in multiple layer architectures which helps the model learn long-range dependencies in the higher level layers in which the order of tokens in the sequence does not matter while in RNN models this problem is present. Given the fact that in RNNs the sequence is processed in a sequential way, therefore, the model is not able to look at the tokens in the future. This issue is addressed in RNNs by training the model in both backward and forward fashion in which we train the model two times once feeding the model with the original sequence and once the reversed version of the sequence so that the RNN can look at the sequences both from left-to-right and from right-to-left. Another way to solve this issue is using bi-directional RNNs as described in Section 2.1.2. Another advantage of CNNs over RNNs is that CNNs use less

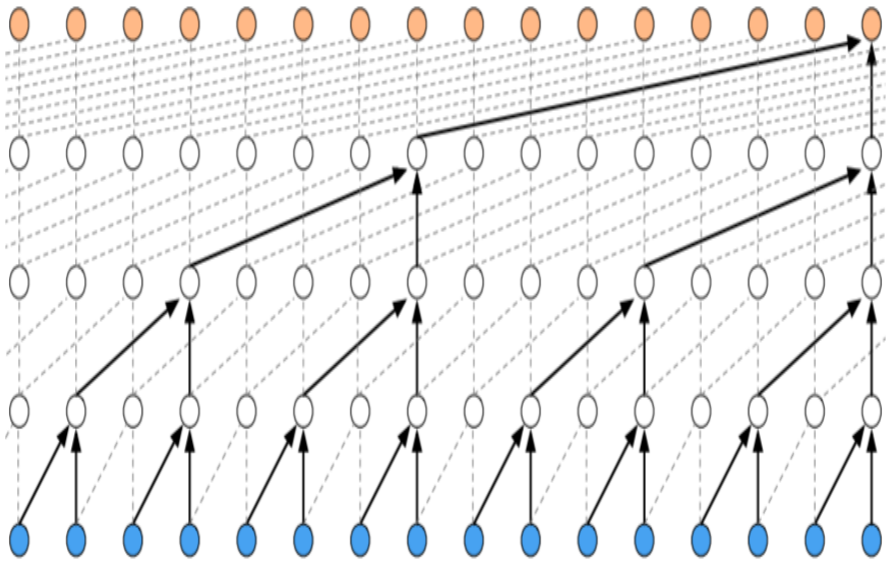

Figure 2.2: Wavenet architecture with dilated convolution [107]

memory compared especially in long sequences. This is due to the fact that RNNs need to store the partial derivatives of the whole sequence up to some point back in time, as used in BPTT [118] and truncated BPTT [49], while in CNNs this number is actually the depth of the CNN architecture which tends to be smaller for long sequences [6][54][32].

Given the aforementioned properties for the CNNs, they seem a good candidate to be used for calculating representations of the sequences and thus be used for sequence modeling. In NLP domain, CNNs have been used for sentence and document classification [57][128][55][51], semantic role labeling [18], text summarization [33], language modeling [20], and Neural Machine Translation [32][54][33]. In this study, recent usage of CNNs for NMT in ByteNet [54] and ConvS2S [32], and WaveNet [107] (used for audio synthesis), is of great interest since these models lead to the creation of self-attentional-based models like Transformer and thus in the following sections we will give an overview of these models.

2.3 Attention Mechanism

Attention mechanism is one of the most exciting and useful advances in deep learning which in some cases augmented the models [5] and also lead to models that instead of recurrence use attention mechanism [109]. Attention idea has been firstly used in computer vision applications such as in [62][24] and later was used in NLP domain, specially for machine translation task [5][73]. Before using attention mechanism, the final hidden state of the encoder, which is supposed to have the gist of the information in the whole sequence, was passed to the decoder and thus decoder could decide what token to output based on the information in this single hidden state. However, given a rather long sequence, the final hidden state of the encoder fails to capture all the information needed for appropriate modeling and also it is highly unlikely that information from initial tokens of the sequence which are far from the last hidden state is captured. Given the problems above, the idea of attention mechanism was used while training the decoder in sequence-to-sequence learning models. In each step in decoding, the attention mechanism looks at all the hidden states of the encoder and assigns a score to each of them by comparing them to target hidden state in the decoder and create a context vector which has all the relevant information needed for the model to decide which token to output in the decoding step. This context vector is then used with target hidden state to decide which token to output. Figure 2.3 depicts the attention mechanism used for the application neural machine translation [72].

2.3.1 Luong and Bahdanau Attention Mechanism

The method to calculate attention weights and context vector are shown in Equations 2.10 through 2.14. In the first step, the current hidden state in the decoder (h_t or target hidden state) is compared against all the hidden states in the encoder (h_e) and a score is assigned to each of the hidden states in the encoder. This is score is called attention weight and the formula to calculate attention weights is shown in Equation 2.10:

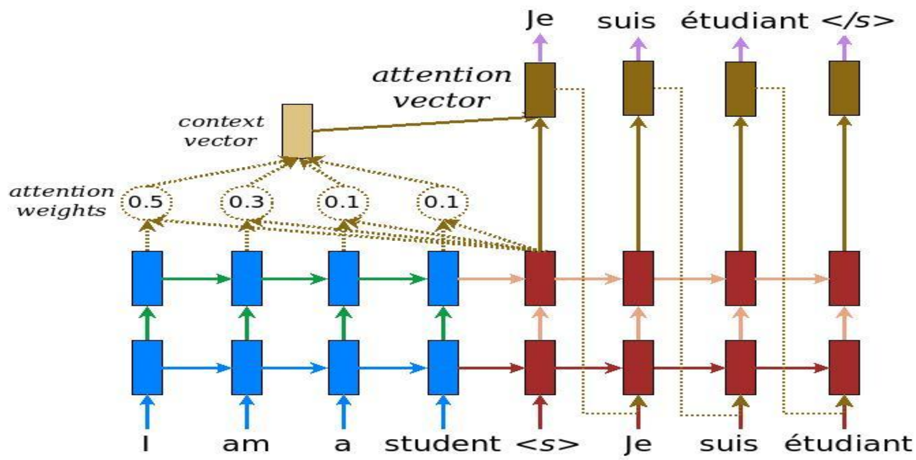

Figure 2.3: Attention mechanism in used in NMT [73]

$$\alpha_{te} = \frac{\exp(\text{score}(h_t, h_e))}{\sum_{e=1}^{E} \exp(\text{score}(h_t, h_e))} \tag{2.10}$$

where E is the number of hidden states in the encoder. After calculating the attention weights, a context vector is calculated which contains aggregated information from all the hidden states of the encoder weighted by their relevance. The formula to calculate the context vector is shown in Equation 2.11:

$$c_t = \sum_{e} \alpha_{te} h_e \tag{2.11}$$

The context vector then is then used with h_t to calculate the attention vector. The attention vector contains all the necessary information from the hidden states in the encoder as well as the current target hidden state and is passed to the decoder in order to continue the decoding process. The method to calculate the attention vector is shown in Equation 2.12

$$\alpha_t = \tanh(\mathrm{W}_c[c_t; h_t]) \tag{2.12}$$

It is important to mention that there are two methods to calculate the score (attention weight) used in Equation 2.10. The method shown in Equation 2.13 is called the Luong Attention method [73] and is in the form of multiplicative style which is similar to a one-layer multi-layer perceptron (MLP).

$$\text{score}(h_t, h_e) = h_t^T \mathrm{W} h_e \tag{2.13}$$

The second method to calculate the attention weights is called the Bahdanau Attention [5] and is in the form of additive style which is similar to a two-layer MLP.

$$\text{score}(h_t, h_e) = v_a^T \tanh(\mathrm{W}_a[c_t; h_t]) \tag{2.14}$$

where v_a and W_a are learnable parameters for the attention model.

2.3.2 Self-Attention Mechanism

Self-attention or intra-attention is an attention mechanism in which we try to learn the relationships between the tokens in a sequence and learn a representation of the whole sequence. The self-attention mechanism has been widely used in many NLP tasks, including sentence representation [68], abstractive summarization [15], semantic role labeling (SRL) [103], and Natural Language Inference (NLI) [83][96].

Transformer [109] model is the first model that entirely relies on the self-attention mechanism for both encode and decoder, and that is why it is also referred to as a self-attentional model. In neural machine translation tasks, CNNs and self-attentional networks could beat the recurrent models (RNNs), and thus some speculate that the ability of the self-attention mechanism to learn long-range dependencies between the tokens which can help the model extract semantic features from the sequences [104].

2.3.3 Multi-head Attention Mechanism

Multi-head attention is another variation of attention mechanism in which different attention models (layers) are created and trained separately so each of them could focus on specific topics and concepts in the sequence. In the case of NLP, multi-head attention has been used for sentence representation [68] in which they defined multiple heads for sentence embedding. For example, consider the sentence "*I went to the restaurant where President Obama had lunch, who is the best president in the USA in my opinion, and ordered some cheap Italian food*". There are different topics discussed in this sentence which have different semantic meanings which are about the *Italian restaurant* and *president Obama*. A simple attention mechanism may not be able to capture all the topics discussed in a sentence or capture all the semantic concepts in a sequence if we want to generalize. Multi-head attention solves this problem by defining multiple heads for the attention mechanism, each of them having different matrices and variables to represent different topics, in hope to capture most of the topics in the sequence.

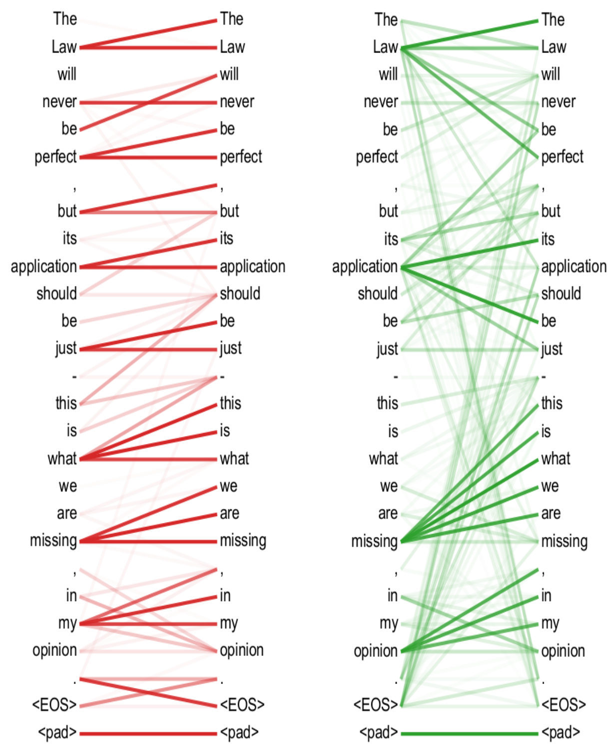

Figure 2.4: Multi-head (self-)attention in Transformer Model [109]

Figure 2.4 depicts the two different heads for the Transformer model in the same layer of the encoder (we will discuss the Transformer architecture later in the next section). This image has been rotated 90 degrees clockwise in order to ease readability. These two heads depict the self-attention weights learned in layer 5 (out of 6) of the encoder in a Transformer model trained for NMT task. The intensity of the colours shows how much each of the tokens (in the left side for each head) attend to the other tokens in the same sentence. The distribution of the colour intensities clearly shows that the model is focusing on two different topics (while in some cases it may be hard to interpret that topic based on the attention weights) each of them representing an abstract concept in the sentence. The multi-head attention can be applied to all the sequence modeling models with encoder-decoder architecture which utilize attention mechanism in order to help the model learn about different topics and semantic concepts in the sequence.

2.4 Self-Attentional based Sequence Modeling

Self-attentional models are a class of deep learning model which learn the representation of the sequence using the self-attention mechanism. Transformer and the Universal Transformer are current state-of-the-art models for sequence learning which completely rely on the self-attention mechanism for sequence modeling. The computational complexity (in terms of training time) for the three aforementioned sequence modeling methods (i.e., recurrence, convolutional and the self-attentional) is shown in Table 2.1 in which n is the number of tokens in the sequence, d is the dimensionality of the hidden state, and k is the kernel size in convolutional models.

Layer Type	Computational Complexity
Recurrent	$O(n.d^2)$
Convolutional	$O(k.n.d^2)$
Self-attentional	$O(n^2.d)$

Table 2.1: Comparison of time complexities for different layers used in sequence modeling [Adapted from [109]]

This table shows the per-layer time complexities for each of the layer types used in sequence modeling. In cases in which $n < d$, the self-attentional model is faster compared to other layers. This case happens in most of the applications for sequence modeling tasks since the typical numbers for hidden state dimensionality are most of the times larger than the sequence length [109].

In the following sections, we will discuss the Transformer and the Universal Transformer models in more detail. We will also discuss the applications of each of these models in different domains.

2.4.1 Transformer

Transformer is a relatively new model proposed by Vaswani et al. [109] from Google AI which could noticeably improve the state-of-the-art machine translation tasks. Transformer is the first model which uses pure attention mechanism, instead of recurrence, to learn the dependencies between input and output. It is based on the self-attention idea which is intended to learn a representation of a sentence by relating different positions of that sentence. Transformer also uses multi-head attention which intends to give the model the ability to look at different representations of the different positions of both the input (encoder self-attention), output (decoder self-attention) and also between input and output (encoder-decoder attention).

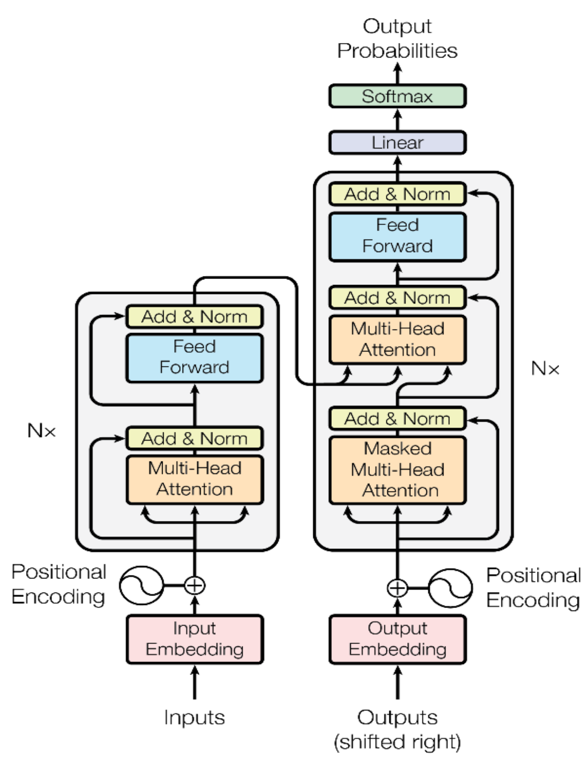

Figure 2.5: The Transformer model architecture [109]

Architecture of the Transformer Model

The encoder is comprised of N identical layers, each of them having a multi-head self-attention mechanism as the first part and the second part is a simple feed-forward network. Note that the model also uses residual connection [16] followed by layer normalization [4] after each part. The decoder has almost the same architecture as the encoder except that it has three main parts. The second and the third part are the same as the encoder, but the first part is a masked multi-head self-attention mechanism. The masked multi-head self-attention mechanism makes sure that decode behaves in an autoregressive way [36] meaning that the prediction for a specific position in the decoder only depends on the previously generated positions.

The self-attention mechanism used in Transformer is a special kind of self-attention called scaled dot-product attention which can be computed using Equation 2.15. The notation used in this equation tries to model the attention mechanism as a series of queries (packed in matrix Q), keys (packed in matrix K), and values (packed in matrix V) which then are used in this equation in order to compute the attention matrix. A rescaling is also done in order to handle large values of the QK^T where d_k is the dimension of both keys and queries.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.15}$$

In order to better learn the different concepts available in the data, multi-head attention is used instead of single-head attention. In multi-head self-attention mechanism used in the Transformer architecture, different and separate representations of the input are learned and then concatenated together in order to create the final representation. Equation 2.16 depicts this procedure in which h number of heads are learned and then concatenated in order to compute the final attention matrix.

$$\begin{aligned} &MultiheadAttention(Q, K, V) = concat(head_1, ..., head_h)W^O \\ &head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.16}$$

Note that W^K, W^Q, W^V are separately learnable parameters for each of the heads and the W^O is the final linear model which creates the final representation of the attention matrix.

The attention mechanism is used in three different ways in the Transformer architecture. First, the encoder has a self-attention mechanism in which the keys, queries, and values all come from the encoder and therefore, in a specific layer, each position can attend to all the positions in its previous layer. The same mechanism is also true for the decoder but with the small difference that each position, in one layer, can attend to all the positions in the previous layer up to the index of that position. As explained before, this is to make sure that the model behaves in an autoregressive way. Finally, the encoder-decoder attention mechanism is the third attention mechanism in the Transformer architecture in which the keys and values come from the last encoder layer but the queries are from the current decoder layer, and in each of the decoder layer, this attention matrix is calculated accordingly.

It is also important to notice that as in the convolutional models, there is no notion of time in the self-attentional models as well. The whole sequence is fed to the encoder and the decoder, and since there is no recurrence in training, the model looks at the whole sequence without considering that it is a sequential type of data. In order to help the model understand the notion of time, we embed time in the sequence by using positional embedding technique [33] in which add either constant numbers (representing time) or sinusoidal function with different frequencies in order to give the model a notion of time. In Transformer model, sine and cosine functions shown in Equation 2.17 are used for positional embedding where d_model is the dimension of input and output tokens in the model.

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.17}$$

Applications of the Transformer Model

Given the great success of Transformer model in machine translation task [109] and the fact that it offers a different training paradigm in which self-attention is the main mechanism used for training and its ability to learn long-term dependencies, many researchers tried to apply it on different learning tasks including computer vision and NLP tasks. In computer vision tasks, the idea of self-attention and Transformer model has been used in image generation [127][132][84], image classification [53], and video captioning tasks [133]. In the NLP domain, Transformer model has been used for mathematical language understanding [113], language modeling [19][91], machine translation [109][2], sentiment analysis [87], question answering [87][25][126], and text summarization [70].

2.4.2 Universal Transformer

Transformer model is a self-attentional feed-forward sequence model which does not have any recurrence. It turns out that the recurrence used in RNN models is essential for some tasks in NLP including language understanding task and thus the Transformer fails to generalize in those tasks. The Universal Transformer model [22] tries to tackle this problem by introducing recurrence in the Transformer model.

Architecture of the Universal Transformer Model

The Universal Transformer model is an encoder-decoder based sequence-to-sequence model which applies recurrence to the representation of each of the positions of the input and output sequences. The main difference between the RNN recurrence and Universal Transformer recurrence is that the recurrence used in Universal Transformer is applied on consecutive representation vectors of each token in the sequence (i.e., over depth) whereas in the RNN models this recurrence is applied on positions of the tokens in the sequence. This idea is depicted in Figure 2.6 in which h_m^t is the representation of token m in the recurrence step time t. This figure shows the depth-wise recurrence for two consecutive recurrence steps in which self-attention mechanism and transfer function are applied to the representations of each token. The transition function serves as the recurrence transformation and is either a separable convolution [17] or a fully-connected two-layer MLP with RELU activation function.

Figure 2.7 depicts the recurrent blocks of both encoder and decoder in

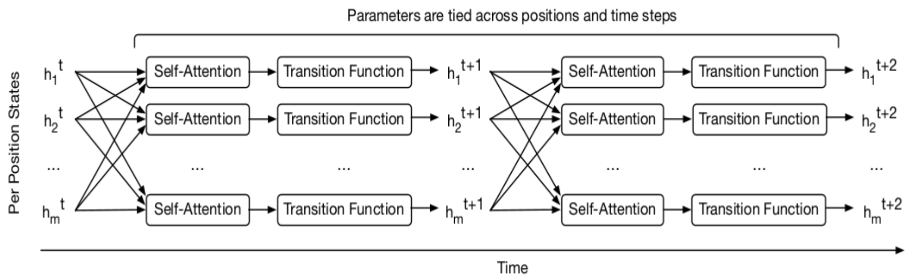

Figure 2.6: Depth-wise recurrence used in Universal Transformer model [22]

the Universal Transformer model. The main difference between the Universal Transformer model and the Transformer model (see Figure 2.5) is that in the Transformer model a constant number of stacked layers are used in both the encoder and the decoder while in the Universal Transformer model, the number of stacked layers can vary based on how many recurrence time steps the model is trained with. A variation of the Universal Transformer, called Adaptive Universal Transformer, applies the Adaptive Computation Time (ACT) [37] technique on the Universal Transformer model which makes the model train faster since it saves computation time and also in some cases can increase the model accuracy [22]. The ACT allows the Universal Transformer model to use different recurrence time steps for different tokens. The ACT technique is used in RNN model to modulate the computation time for each token based on the fact that in sequence modeling problems, some tokens are more ambiguous than others and thus the ACT technique helps the model have more focus on those tokens.

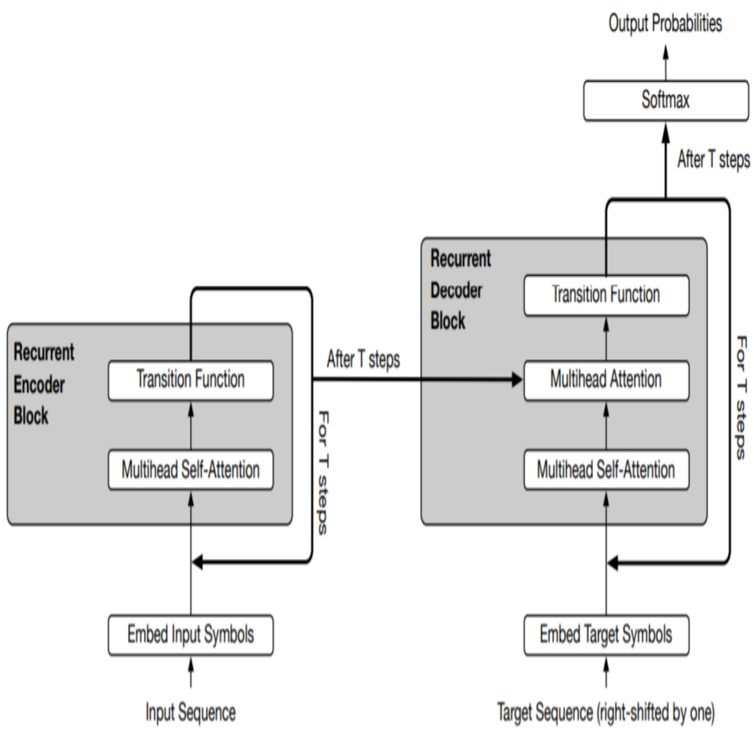

Figure 2.7: The Universal Transformer model architecture [22]

Chapter 3

Task-Oriented Dialogue Generation Systems

Task-oriented chatbots tend to help the user accomplish a specific task such booking a salon, restaurant, parking, etc., successfully and usually with the minimum number of interactions. These systems should be able to query databases or call an API dynamically in the conversation in order to complete the task. For example, a task-oriented chatbot designed to help customers make a reservation in restaurants should be able to query the database of restaurants (with their meta information) in order to look for what the user gave it as his/her preferences. These user preferences will be given to the chatbot in the form of domain-specific entities. Chatbot then queries the database in order to provide suggestions to the user and also can ask about more information in order to limit its search space and provide more specific suggestions. When all the required information is provided by the user, the chatbot can call an API in order to make the reservation process complete and notify the user if the reservation was made successfully or not. This example is one of the possible scenarios that could happen in reality and the number of possible scenarios heavily depends on the complexity of the task.

In the following sections, we will discuss the different architectures of task-oriented chatbots and also explain the taxonomy that is common among the task-oriented chatbots. In Section 3.1, we will explain the user intent and entities that are common among the task-oriented chatbots regardless of their architecture. Later in Section 3.2, we explain the different methods used for creating task-oriented chatbots and also explain different architectures used in each of those methods. Lastly, in Section 3.3, we will explore how the datasets used for training task-oriented chatbots are created and also explain the different methods used for dataset generation.

3.1 Taxonomy of Different Task-Oriented Dialogue Generation Systems

There are several ways to implement a task-oriented chatbot including complete rule-based chatbots, machine learning based methods, or hybrid methods which incorporate both machine learning and domain-specific rules. Regardless of how these chatbots are implemented, there are some common modules which these task-oriented chatbots have to implement in order to be able to comprehend the user provided information and generate an appropriate response. Generally speaking, the task-oriented chatbot should be able to first understand the user intent, e.g., *booking a restaurant* or *finding a movie*, and be able to discern the domain-specific entities provided by the user. The chatbot should also be able to keep track of user intent and entities throughout the conversation and process them in order to either retrieve relevant information from the database or ask relevant questions from the user to make the task complete in a reasonably limited time. Finally, the chatbot has to have a robust and well-defined policy system in order to take appropriate actions given the current state of the conversation. In the following sections, we explain the intent and entity concepts which are the main building blocks of a task-oriented chatbot.

3.1.1 Determining the User Intent

Task-oriented chatbots are designed to help the user accomplish a specific task in predefined domains. For example, given the restaurant domain, the user may have many intents such as *finding a restaurant*, *booking a table*, or *searching for a specific type of food* in mind and each of these intents can be defined as a task for the chatbot. Determining the user intent is one of the main challenges that a task-oriented chatbot has to do, and many big corporations have built systems which can be used as an intent classifier for task-oriented chatbots [10]. Luis [120] by Microsoft[1], Amazon's Lex[2], Google's Dialogflow[3], IBM's Watson[4], and Facebook's Wit[5] are the major Natural Language Understating (NLU) providers which users can use to do intent classification in their task-oriented chatbots. Rasa NLU[6] [8] is an open-source NLU platform which we will explore more later in Section 4.3.4.

3.1.2 Identifying the Needed Entities

One of the main components of task-oriented chatbots is its ability to recognize the domain-specific entities from user input (either text or auditory) which show the user preferences The possible entities for reserving a restaurant are *restaurant name*, *cuisine type*, *location* ,*date*, *time*, and *number of people*. These are required entities which the chatbot needs to know their value in order to complete the task and make a reservation. The user can also ask the chatbot about entities like *phone number*, *working hours*, and *address* of the restaurant that the chatbot made the reservation for. These entities are optional entities which are not necessarily required for the chatbot in order to make the reservation complete and usually are asked by the user after the task is complete. The distinction between required and optional entities are highly domain-specific but usually required entities are *informable* which means the user has to provide the value for these entities for the chatbot and the optional entities are *requestable* which means the user can request the value for these entities from the chatbot [67].

[1]https://www.luis.ai
[2]https://aws.amazon.com/lex
[3]https://dialogflow.com
[4]https://www.ibm.com/watson/services/natural-language-understanding
[5]https://wit.ai
[6]https://github.com/RasaHQ

3.2 Architecture of a Standard Task-Oriented Dialogue Generation System

There are multiple ways to implement a task-oriented chatbot. Task-oriented chatbots can be designed using a set of hand-crafted and carefully-designed rules like Parry [12] or Eliza [97]. Another way that has recently been popular for designing task-oriented chatbots is corpus-based which means the chatbot can be trained with a relatively large corpus of conversational data(e.g., chat logs). It is worth mentioning that most systems usually use a combination of these two techniques since some domain-specific rules need to be applied for a task-oriented chatbot and it is not probably efficient for the system to learn those domain-specific rules from the corpus. Deep learning methods are one of the best ways to implement corpus-based chatbots and many architectures, and many of the state-of-the-art methods for building task-oriented chatbots are heavily based on deep learning methods.

Corpus-based architecture for task-oriented chatbot design usually falls into two main categories which are end-to-end and pipelined architectures. In end-to-end design, all the components of the chatbot, i.e., NLU, dialogue management, and NLG are learned in an end-to-end way, and some of these components can be implicitly implemented using deep learning methods such as encoder-decoder based models. In contrast, in pipelined chatbots, each of these components is trained separately, and then they are combined in order to build the final chatbot system.

3.2.1 Rule-based Task-Oriented Chatbots

In rule-based chatbot design, domain-specific rules and pattern matching are used for both NLU and dialogue management components of the task-oriented chatbot. The chatbot runs the user query against a set of pre-defined patterns and extracts the information needed in order to understand what user intends to say and then tries to respond based on other sets of rules that are task-specific. It is needless to mention that the complexity of the rules defined for the chatbot to follow in order to provide an appropriate response given the current state of the conversation heavily depends on the complexity of the task and that is the origin of one of the main problems related to rule-based chatbots. Given a complex task, it takes a considerable amount of man-hours to define all the rules and their relationships.

Artificial Intelligence Markup Language (AIML) [112] is one the commonly used techniques for the development of rule-based conversational agents. AIML is an XML-based markup language which tries to helps the task of dialogue modeling which follows a stimulus-response approach [74]. AIML consists of special tags which can define how the chatbot can respond based on the user utterance. Each of the user utterances (stimulus) will be modeled using the tags in AIML, and the chatbot can respond to those utterances using the pre-defined responses. Artificial Linguistic Internet Computer Entity (ALICE) [112] was the first chatbot designed using the AIML language that could pass the Loebner Prize [75] in, a Turing Test for computer programs, in 2000 and 2001. It is worth mentioning that the learning method of ALICE is based on active learning approach in which a human, called botmaster, tries to improve the responses and query understanding of the chatbot and make modifications to the template and patterns in the AIML code. Another successful chatbot (open-domain chatbot) developed using AIML is Mitsuku[7] developed by Steve Worswick which could won the Loebner Prize in 2013, 2016, 2017, and 2018 [111][1].

The aforementioned chatbots, ALICE and Mitsuku, are on the list of Leobner Prized chatbots and both of them were developed using the AIML method. This shows that the AIML method could lead to the creation of a chatbot that can respond to humans appropriately with consideration of the context of the conversation and can even able to pass the Turing Test. However, the complexity of the task, thus the complexity of the AIML code, and the amount of man-hours needed to such a chatbot is not feasible for many people and corporations and therefore there is a need for automation of the learning process for conversational agents. The corpus-based methods can help us with this problem and provide methods and models which can learn how to understand user

[7] https://www.pandorabots.com/mitsuku/

queries and respond accordingly by learning from chat logs and conversational datasets.

3.2.2 Corpus-based Task-Oriented Chatbots

In contrast to rule-based chatbots which are created by hand-crafted domain-specific rules, the corpus-based chatbots learn to converse with the users by learning the conversations from a corpus. The corpus can be chat logs of a call centre which contains the typical conversations between the agents and the customers (usually FAQs) or from a live conversational interface available in many websites in which a real agent has a conversation with the customers. Given the need for a corpus which contains the domain-specific conversations and the fact that the chatbot will be trained using the corpus dataset, the quality and relevance of the corpus play an important role in building a robust and functional task-oriented chatbot. However, in some cases, the access to such corpus for specific domains is not possible, or such a corpus does not exist. One of the active research areas in task-oriented dialogue generation systems is about methods to create domain-specific datasets which can be used for training of task-oriented chatbots. In Section 3.3 we will explain some of the most common methods researchers and engineers use to create these datasets.

3.2.3 End-to-End Architectures for Task-Oriented Dialogue Generation Systems

End-to-end architectures are one of the most used architecture for research in the field of conversational AI. The advantage of using end-to-end architecture is that one does not need to explicitly train different components for language understanding and dialogue management and then concatenate them. This makes the task of creating the chatbot easier in a way that one needs to only define the architecture once and then train the model in an end-to-end way. The downside of using end-to-end method is that typically one needs more training data in order to train a reasonably good chatbot. For the sake of conversational agents, using end-to-end training is the dominant way in open-domain chatbots, and also many of models proposed to train task-oriented chatbots are trained in an end-to-end way.

Many models are proposed to train chatbots in an end-to-end way. All the sequence learning models (i.e., recurrence-based, convolution-based, and self-attentional models) can also be used for training end-to-end task-oriented chatbots. Our main contribution in this study is the comparison of the most widely used sequence learning architectures for the task of creating end-to-end task-oriented chatbots and later in Chapter 5. We will discuss how we can configure and prepare task-oriented conversation data to be used in sequence learning models. Apart from the sequence learning models, other architectures that are specifically designed for task-oriented dialogue generation are also proposed. Network-based end-to-end trainable model [117] is one the most popular end-to-end architectures used for training task-oriented chatbots. The architecture of this model is depicted in Figure 3.1.

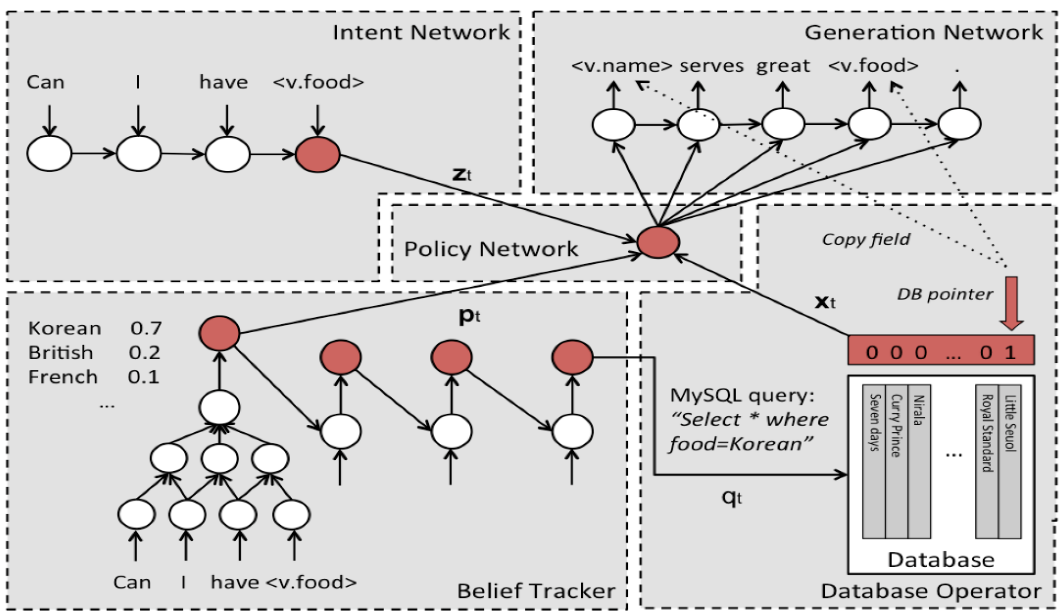

Figure 3.1: Architecture of network-based end-to-end trainable task-oriented chatbot [117]

Each of components in Figure 3.1 correspond to one of the main tasks that the chatbot has to do. The model tries to learn the user intent and also extract and keep track of different entities in the user utterance and then make a decision based on the current state of the conversation. It is also able to query a database and create natural utterance. The generation network is similar to NLG in pipelined architecture and the intent network, and belief tracker serves as the NLU component, and finally, policy network serves as the dialogue management components in the pipelined architecture. The difference between this architecture and the pipelined architecture is that this network-based model is trained end-to-end and the error can back-propagate through all of the components. This makes the task of training harder since all of these components are trained all together, and therefore much more data is needed to train it compared to pipelined architectures.

Another popular method for training chatbots is based on memory networks [101]. Memory networks augment the neural networks with task-specific memories which the model can learn to read and write. Memory networks have been used in [9] for training task-oriented in which they store dialogue context in the memory module, and then the model uses it to select a system response (also stored in the memory module) from a set of candidates. A variation of Key-value memory networks [79] has been used in [28] for the training task-oriented chatbots which stores the knowledge base in the form of triplets (which is (subject,relation,object) such as (yoga,time,3pm)) in the key-value memory network and then the models tries to select the most relevant entity from the memory and create a relevant response. This approach makes the interaction with the knowledge base smoother compared to other models.

Another approach for training end-to-end task-oriented dialogue systems tries to model the task-oriented dialogue generation in a reinforcement learning approach in which the current state of the conversation is passed to some sequence learning network, and this network decides the action which the chatbot should act upon. End-to-end LSTM based model [121], and the Hybrid Code Networks [119] can use both supervised and reinforcement learning approaches for training task-oriented chatbots. Figure 3.2 depicts the architecture of the Hybrid Code Network model.

In the Hybrid Code Network model architecture, all the information from the past (or up to some point in the past) is passed to an RNN network. The RNN network is then trained in order to select the best action given the cur-

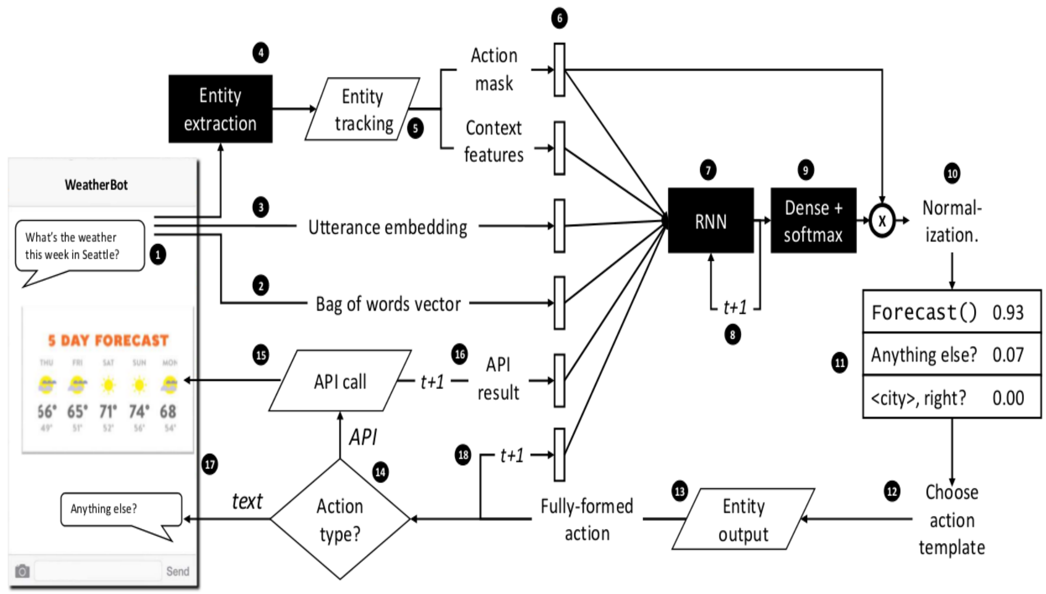

Figure 3.2: Architecture of Hybrid Code Network model [119]

rent state of the conversation. This approach is very similar to reinforcement learning method in which a policy network is trained to select the best action (here chatbot action) given the current state (where the state is the current state of the conversation). This approach makes it possible to train the model both in supervised and also reinforcement learning ways. The use of reinforcement learning can help the model learn with fewer data points, but usually, user feedback or a user simulator is needed in order to learn the best policy.

3.2.4 Pipelined Architecture for Task-Oriented Dialogue Generation Systems

In contrast to end-to-end architectures in which all the components of the task-oriented chatbot, including NLU, dialogue management, and NLG are trained all together, and these components are implicit in the model architecture, the pipelined architecture tries to implement each of those components individually and then stick them back together in order to make a complete task-oriented chatbot system. The general architecture for a pipelined task-oriented chatbot is depicted in Figure 3.3.

In the following sections, we will explain how each of these components

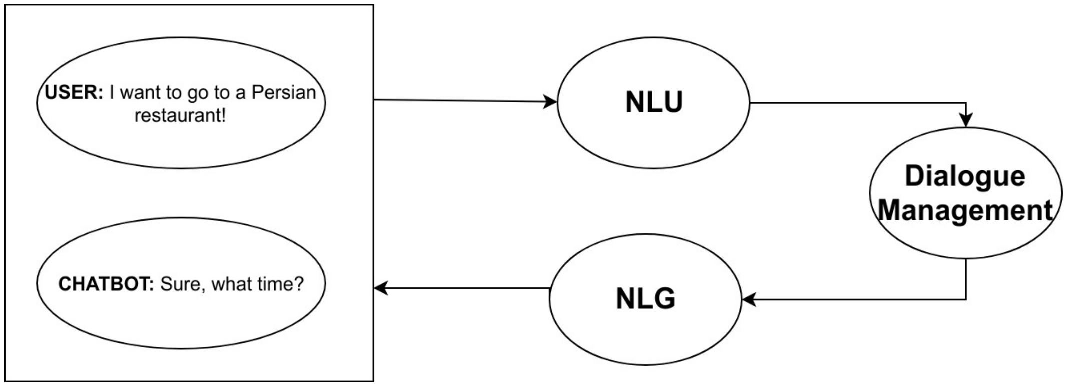

Figure 3.3: Pipelined architecture for task-oriented dialogue generation systems

contributes to the final task-oriented chatbot and what are the available models and methods for each of the components.

Natural Language Understanding

NLU component is the component responsible for understanding the user utterances and extract specific information from it. NLU is somewhat similar to dialogue generations systems in a sense that it has a wide spectrum of implementations ranging from simple rule-based and pattern matching method to complex deep neural networks. The rule-based methods mostly rely on regular expression based methods in which specific patterns are used to parse the sentence and extract specific information from the sentence.

NLU in task-oriented chatbots is supposed to extract two types of information from the sentences which are user intent and name of the entities (slots) specific to the task as depicted in Figure 3.4. Note that NLU is a general term, and different type of chatbots and query systems will have different type of NLUs since there is need to extract more information, such as semantic roles or sentiment, from the input sentences and some NLU service providers such as IBM Watson NLU[8] do provide this extra information. Most of the task-oriented chatbots will work fine if the NLU provides the user intent and slot name because these two pieces of information are essential for the chatbot in order to take the right action in dialogue management component.

[8] https://www.ibm.com/watson/services/natural-language-understanding/

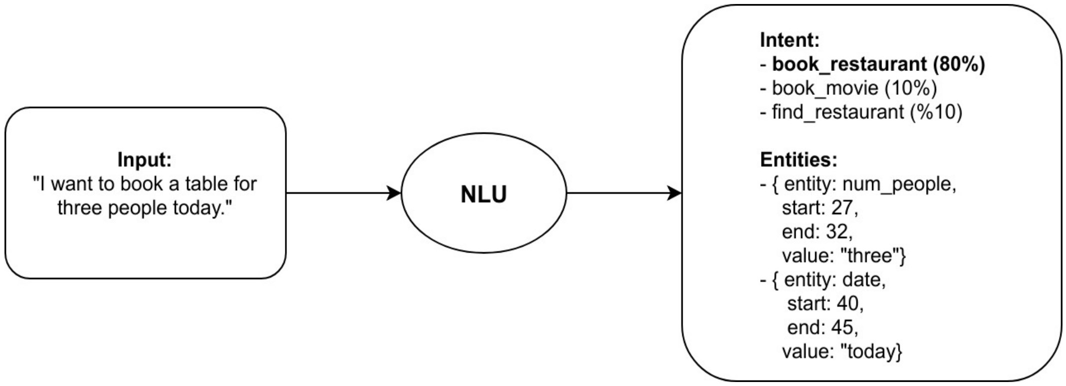

Figure 3.4: Typical NLU system for a task-oriented dialogue generation system

The user intent classification part of the NLU can be implemented using a broad spectrum of techniques from classical machine learning methods like SVM (Support Vector Machine) [77] to deep learning methods such as CNNs [41] and DCNs (Deep Convex Networks) [23][105]. For entity extraction, classical methods like regular expression, CRF-based (Conditional Random Fields) methods [61] and also LSTM-based methods such as LSTM-CRF [48] can be used.

Dialogue Management

Dialogue management is the task of using all the information of the current state of the conversation in order to find the best action that the chatbot has to take. This action can be querying an API or a knowledge based (KB), asking the user more information, or making sure that the user gave you the correct information. The model used in dialogue management has to be able to first extract and gather all the necessary information from the user and possibly other sources and have a reasonable policy to decide what action to be taken. The methods for dialogue management range from rule-based methods [34] to deep learning based models [43]. End-to-end LSTM based model [121], and the Hybrid Code Networks [119] discussed in Section 3.2.3 can also be used in dialogue management since they can learn a policy based on all the information of the current state of the conversation. Rasa Core[9] [8], which is

[9]https://www.rasa.com/docs/core/

one the most widely used open-source dialogue management frameworks, uses an ensemble of machine learning [121] and rule-based methods for training a policy network.

Natural Language Generation

NLG is the generation of a natural language sentence given the current state of the conversation. It can also mean converting the chatbot action into a natural language sentence. The generation of such a natural language sentence can be done using template-based methods, N-gram models [21], or deep sequence generation models such as using CNN [115] and LSTM [116] for NLG.

3.3 Task-Oriented Dialogue Datasets

As mentioned in Section 3.2.2, the quality of the corpus used for training corpus-based task-oriented chatbots is of great importance since it directly affects the robustness of the trained chatbot. In case such a high-quality corpus does not exist for a specific domain, there are multiple ways to create them. The data generation techniques for task-oriented chatbots fall into main three categories which are machine-to-machine (M2M), human-to-machine, and human-to-human methods. In the following section, we will briefly explain how each of these methods is used to create domain-specific datasets.

3.3.1 Dialogue Action

Dialogue action is one of the main constituents of a task-oriented dialogue dataset. Dialogue actions convey either the user's or the system's (chatbot) intent in a specific turn and can semantically represent the sentence used in that specific turn [42]. Typical examples of a dialogue action are *INFORM* which means that either the user or the system wants to inform a slot (entity) value to the other speaker, *REQUEST* which means a slot value is asked from the other speaker. Other commonly used dialogue actions in the task-oriented conversations are *GREETING*, *CONFIRM*, *REQUEST*, *OFFER*, *SELECT*, *NEGATE*, and *DENY* [95]. A complete list of dialogue actions can be found in the Dialogue State Tracking Challenge book [42]. It is important to note that dialogue actions are not natural language sentences, but are able to model the semantic state of a specific turn in a task-oriented conversation.

3.3.2 Methods for Creation of Task-Oriented Dialogue Datasets

Existing task-oriented dialogue datasets can be categorized into three main groups based on how they were created, and the three main groups are machine-to-machine, human-to-machine, and human-to-human methods. In the following sections, we will discuss these three data creation methods and provide a list of available datasets for each of them.

Machine-to-Machine

In machine-to-machine (M2M) technique a simulation is conducted using a user simulator and a system simulator. The created conversations between the simulators are then converted to natural language either by either predefined rules [9] or turkers (usually using Amazon Mechanical Turk[10]) [95] to create utterances that are natural. The datasets created using the machine-to-machine technique can be easily adapted to cover a particular domain, and since the conversation templates are generated by the simulators, the quality of the which conversation templates can be checked before it is converted to natural language looking utterances. It is also possible to control the flow of the conversations since they are created by simulators. However, using this method for dialogue generation can introduce bias in the dataset in a way that the distribution of the created conversations is different from conversations in reality in which real users converse with real agents. This problem can be alleviated by conditioning the user simulator on user profile attributes which try to define some properties, such as *verbosity*, *politeness*, etc., for the user simulator to make it sound like more natural and realistic. We will be using the datasets publicly provided by [95] and therefore we will discuss how these datasets were created.

[10]https://www.mturk.com

In the method proposed by Shah et al. [95], the data generation system is a pipeline of different components as depicted in Figure 3.5. In task specification part, a sample database which contains the information for the all the domains that the conversations are intended to cover will be used to sample entity values. Each domain has its own specific entities which are also called slots. For example, the possible slots used in movie domain are *movie name*, *theater*, *date*, *time*, and *location*. and for the restaurant domain the possible slots are *restaurant name*, *cuisine*, *price range*, *date*, *time*, and *location*. The database in the task specification part should have possible values for each of these slots in each domain so that they can be sampled to create scenarios. As an example, the possible values, called $\mathcal{S}$, for slot *date* available in the database could be $\mathcal{S} = \{tomorrow, today, \text{next week}, \text{march 2nd}, \text{don't care}\}$.

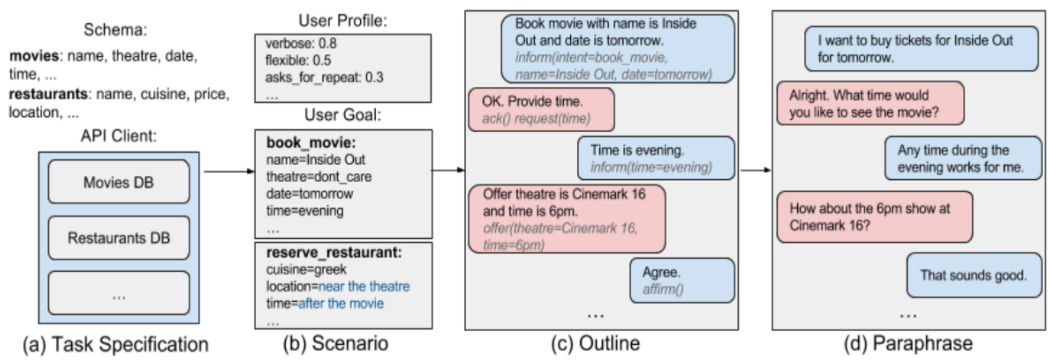

Figure 3.5: Pipleline used in M2M for outline and paraphrase generation [95]

In scenario generation part, a scenario is created which later will be used to create a conversation. The scenario consists of a user profile and user goal. As discussed earlier, the user profile contains some attributes related to the users which can be used in user simulator in order to make it more realistic. The user goal is a set of sampled slot value pair for each of the domains that will be used in the conversation. As illustrated in Figure 3.5, the user goal can have multiple intents, each of them belonging to a different domain, and these intents with their sampled slot value pairs in outline generation.

In outline generation part, we will pass a scenario to the outline generation system in which it will be used in a self-play conversation modeling. In the self-play model, a user simulator and a system simulator are used to create an outline. The outline contains the template of a conversation between the user and the system and is intended to model a real conversation that happens in those domains. The user simulator is based on the Agenda-based user simulator [93] but is conditioned on the user profile, and the system simulator is simply implemented using finite-state machines [47]. In this phase, each of the scenarios will be used to create an outline, and later the generated outline will be passed to paraphrasing phase.

In paraphrasing phase, the generated outlines will be passed to turkers in a crowd-sourcing platform such as Amazon Mechanical Turk. The turkers will be instructed to paraphrase the outline sentences into natural language sentences. This process not only helps with creating natural language sentences but also adds variation to the sentences. The crowd-sourced dataset which is now paraphrased can is supposed to cover most of the conversation scenarios that happen in reality. The sim-M (movie domain) and sim-R (restaurant domain) datasets[11] created in [95] are examples of machine-to-machine dataset generation.

Human-to-Machine

In the human-to-machine method for data collection, an initial system is created so that people can chat with it and create more dialogues. In this approach, a system is created using either rule-based methods or trained with a domain-specific corpus and then is deployed to interact with real users and therefore create more task-oriented dialogue data for that specific domain [89][11]. The problem with this method is that the creation of the initial system which is used to interact with the users heavily impacts the quality of the data. The DSTC2 & 3[12] [42] datasets are the most widely used datasets which are created using the human-to-machine method.

[11] Available at https://github.com/google-research-datasets/simulated-dialogue

[12] Available at http://camdial.org/ mh521/dstc/

Human-to-Human

Human-to-human dataset generation is perhaps the most obvious and efficient way for the creation of task-oriented datasets. In this method, a scenario for the task-oriented conversation is created in which a user has to complete specific tasks (such as buying a book and reserving a restaurant table), and then these scenarios are used in crowd-sourcing platforms such as AMT in order to create natural language sentences for the scenarios. One of the most widely used methods for creating human-to-human datasets using AMT is called WOZ (Wizard of Oz) [56] which is an interactive way to design task-oriented datasets in which a sample of conversation is given to a turker and the turker is asked to add one turn to the conversation and then the conversation is passed to another turker. Using the WOZ technique, one can create a diverse set of task-oriented datasets which can also have multiple-domains (i.e., a dataset with movie and restaurant domains). MultiWOZ dataset[13] [11] and the dataset created by Eric and Manning in [28][14] are examples of datasets created using the human-to-human approach. The Restaurant Game [81] is one of the methods which user behaviour and thus realistic datasets can be generated for creating task-oriented chatbots.

[13] Available at http://dialogue.mi.eng.cam.ac.uk/index.php/corpus/

[14] Available at https://nlp.stanford.edu/blog/a-new-multi-turn-multi-domain-task-oriented-dialogue-dataset/

Chapter 4

Evaluation of Task-Oriented Dialogue Generation Systems

Evaluation of chatbots refers to the process in which a chatbot agent is tested against a number of metrics in which its ability in different phases of language understanding and relevant response generation are tested. The chatbots are usually supposed to act like humans and create an engaging and context-relevant response, it is challenging and hard to evaluate these systems. Chatbots can be either evaluated automatically or manually. In the manual evaluation, the conversation history of a chatbot with real users (or a user simulator) is passed to evaluators (e.g., Amazon MT workers) and its quality is evaluated based on domain-specific metrics. In this study, we are more interested in automatic evaluation metrics, and specifically, we will focus on evaluation metrics for task-oriented chatbots as manual evaluation is costly and time consuming.

Regardless of the type of the chatbot (i.e., task-oriented or open-domain) and its architecture (i.e., end-to-end or pipelined), chatbots have to go through three phases for response generation. The first phase is understanding user utterance and extracting the necessary and relevant information from the user utterance. The second phase is processing all the information from the current state of the conversation and the information from the user utterance and choose an appropriate action (e.g., issuing an API call, query a database, ask the user more information, etc.). The third phase for the chatbot is the NLG part in which the chatbot creates an appropriate response in the form of natural language which is understandable by the user. It is important to note that the aforementioned three phases are more related to pipelined chatbots in which for each of these phases there is a trained module and the final chatbot is created by concatenating these modules (see Section 3.2.4). In the end-to-end chatbot architecture, these phases are implicitly learned by the model.

The evaluation metrics and methods used for the evaluation of task-oriented and open-domain chatbots have slight difference due to the fact that their final purpose is different. Task-oriented chatbots are supposed to complete a specific task in as few turns as possible and provide relevant responses while open-domain chatbots are intended to keep the user engaged and have a long conversation with the user. The evaluation methods and metrics used in sequence modeling tasks such as BLEU [82] which is usually used in machine translation tasks is also used in chatbot evaluation. The other methods that are usually in the evaluation of task-oriented chatbots are per-turn accuracy, per-dialogue accuracy, and entity F1 score. It is important to notice that the evaluation metrics of task-oriented chatbots do not model user error and characteristic in the evaluation process. Using a user-profile conditioned user simulator which is able to simulate a real user with different characteristics seems a promising approach in order to better evaluate the task-oriented chatbots. We create such an evaluation method using the user simulator, NLU, and NLG modules and evaluate the trained chatbots by measuring metrics such as the number of turns in the conversation in order to provide a proper evaluation method which models the user characteristic in the process of evaluation.

The outline of this chapter is as follows. In Section 4.1 we explain with detail the evaluation metrics that we will later use in our experiments. Later in Section 4.3 we explain the profile-conditioned user simulator evaluation method and explain its different components.

4.1 Evaluation Metrics for Task-oriented Dialogue Generation Systems

Evaluation of dialogue generation systems is a hard task [69][14] due to the fact that the generated dialogues have to be both structurally and semantically correct and coherent while the latter is a very hard task. For this study, we use four common metrics used in the evaluation of task-oriented chatbots [27]. Given the fact that end-to-end architectures of chatbots are language generation models, thus we can use the evaluation metrics used in the evaluation of language generation models for evaluation of chatbots. BLEU metric is one of the most popular evaluation methods for such systems and thus we will use it for the evaluation phase.

As explained in Section 3.1.2, one of the main tasks of task-oriented chatbots is to extract the domain-specific entities from the user utterance and thus it is important to evaluate the chatbot's ability for this task. Entity F1 score is the evaluation metric that we will use for the evaluation of end-to-end trained chatbots for this task. The other two methods that we will use for evaluation of chatbots are per-turn and per-dialogue accuracy which measure how chatbot can generate response according to the training corpus. In the following sections, we will dive more into these evaluation methods and also discuss their drawbacks.

4.1.1 BLEU

We use the BLEU (bilingual evaluation understudy) [82] metric which is commonly used in machine translation tasks. BLEU metric can be used to evaluate dialogue generation models as in [66][92][27]. BLEU metric is a word-overlap metric which computes the co-occurrence of N-grams in the reference and the generated response and also applies the brevity penalty which tries to penalize far too short responses which are usually not desired in task-oriented chatbots. We compute the BLEU score using all generated responses of our systems.

The formula for calculating BLEU score is shown in Equation (4.1):

$$BLEU = \min(1, \frac{\text{output length}}{\text{reference length}}) * (\prod_{i=1}^{4} \text{precision}_i)^{\frac{1}{4}} \quad (4.1)$$

where the first term, $\min(1, \frac{\text{output length}}{\text{reference length}})$, is brevity penalty and penalizes the answers which are far too short. The second term in Equation (4.1), $(\prod_{i=1}^{4} \text{precision}_i)^{\frac{1}{4}}$, is the product of precision for all N-grams of size 1 to 4.

BLEU metric was mainly designed for evaluation of machine translation tasks [125], but it has been used for evaluation almost all the NLP tasks which are in the form of language generation. Using BLEU score makes sure that the generated response is identical to the target response in the test dataset, but the model could generate a response which is entirely different from the target response though still a valid response. The usage of beam search can alleviate this problem [122]. In beam search decoding, the model considers the K most probable tokens and computes the multiplied log probabilities of tokens for each path and this process is repeated for each path until end of sentence token is reached and the output sentence (i.e. system response in case of a task-oriented chatbot) is generated (see 5.3 for more detail).

4.1.2 Per-Turn Accuracy

Per-turn accuracy measures the similarity of the system generated response versus the target response. Eric and Manning [27] used this metric to evaluate their systems in which they considered their response to be correct if all tokens in the system generated response matched the corresponding token in the target response. This metric is a little bit harsh, and the results may be low since all the tokens in the generated response have to be exactly in the same position as in the target response.

As an example, consider the target response "*The reservation for Pom restaurant at 2 pm is confirmed*" which is a turn in a conversation about reserving a restaurant. If the chatbot manage to create the exact same response as the target response then it is deemed to have created the correct response. Note that even if the chatbot creates the response with missed, additional, or reordered tokens which may have the same meaning as the target response it won't be deemed as the correct response.

4.1.3 Per-Dialogue Accuracy

We calculate per-dialogue accuracy as used in [27][9]. For this metric, we consider all the system generated responses and compare them to target responses. A dialogue is considered to be true if all the turns in the system generated responses match the corresponding turns in the target responses.

Consider a sample target conversation $X = \{x_1, x_2, ..., x_k\}$ in which x_i is the ith turn in the conversation with total of k turns. Consider the case in which the chatbot creates the response $X' = \{x'_1, x'_2, ..., x'_k\}$. The response, here the whole conversation X', is considered is to be correct if every token in x'_i is in the exact same position as in the x_i for i=1,...,k. Note that the order of turns and the number of turns do matter in this metric and that is why it is considered a very harsh metric.

4.1.4 Entity F1 Score

Datasets used in task-oriented tasks have a set of entities which represent user preferences. For example, in restaurant domain chatbots common entities are meal, restaurant name, date, time and the number of people. These are usually the required entities which are crucial for making reservations, but there could be optional entities such as location or rating. Each target response has a set of entities which the system asks or informs the user about. Our models have to be able to discern these specific entities and inject them into the generated response. To evaluate our models we could use named-entity recognition evaluation metrics [50]. F1 score is the most commonly used metric used for evaluation of named-entity recognition models which is the harmonic average of precision and recall of the model. We calculate this metric by micro-averaging over all the system generated responses.

$$F - Score = 2 * (\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}) \tag{4.2}$$

4.2 User Simulators For Task-Oriented Chatbots

All the aforementioned evaluation methods of task-oriented chatbots try to assess the chatbot's ability in learning based on the training corpus, and the evaluation is done using the test dataset. However, given the functionality of a task-oriented chatbot in serving users for doing specific tasks, it makes sense to employ either real users or a user simulator for the evaluation of the chatbot. Utilizing real users for the task of chatbot evaluation can be costly, and in such cases, it may be useful to have a user simulator which can emulate the real users as much as possible. Building such a user simulator is a hard task due to the fact that the final user simulator has to be able to emulate the real user in a realistic way which consists of user behaviour and user characteristics. A user simulator is considered to be the alternative which tries to imitate the behaviour of chatbot users while conversing with the task-oriented chatbot.

There are generally two methods for creating user simulators for task-oriented chatbots which are agenda-based simulators and model-based simulators [31]. In agenda-based user simulator we use hand-crafted rules to simulate the interaction of a user with the task-oriented chatbot while in the model-based methods we rely on some corpus to learn the behaviour of the user. It is also possible to use a mix of both techniques for creating the user simulator.

Agenda-based user simulator [93] is one of the most widely used methods for implementation of user simulators as used in Microsoft's TC-bot [67]. The advantage of using an agenda-based user simulator is that there is no need for a dataset to train the user simulator and we can add as many hand-crafted rules as we need to the simulator to cover every possible case. This is needed for simulators used in industry chatbots since we need to handle every possible case (and corner cases) in order to make sure the chatbot works well. On the other hand, this process can be tedious and time-consuming since these rules are hand-crafted.

Model-based methods usually rely on learning the used behaviour based on some corpus [13]. End-to-end user simulators have recently been proposed which try to train end-to-end sequence learning models on the task-oriented corpus and emulate the user [60] [3]. Using end-to-end sequence learning models with later variable models has also been proposed to add user behaviour to a user simulator [40]. The advantage of using model-based user simulators is that we can learn the user behaviour from some training corpus without the need to create hand-crafted rules as used in agenda-based simulators. On the other hand, it is likely that some cases (most probably corner cases) are not available in the corpus and thus the chatbot is not evaluated on how it handles those situations.

We propose a new user simulator for task-oriented chatbots which has a pipelined architecture similar to Microsoft's user simulators used to train reinforcement learning agents in [67] but in our architecture we try to incorporate the user profile by conditioning the agenda-based user simulator on a user profile and thus try to make it easier to incorporate user behaviour and user characteristic into the user simulator.

4.3 Profile-Conditioned User Simulator

Our proposed method is based on a pipelined architecture which makes the task of adding user behaviour and user characteristic to the simulation easier. The architecture of our model is depicted in Figure 4.1.

There are five components in the profile-conditioned simulator. The user simulator, NLG, NLU, user goal, and user profile are implemented and trained independently and then concatenated together in order to create the user simulator. The mechanism of the proposed user simulator is as follows. The user simulator used utilizes a user profile and a user goal in order to create annotations. An annotation consists of a dialogue action and a list of entities with their values which are chosen randomly. An example of an annotation is *INFORM(cuisine=Persian,date=tomorrow)* which can be translated into natural language like "*I want to go to a Persian restaurant tomorrow*". The annotation is then passed to our template-based NLG and converted into nat-

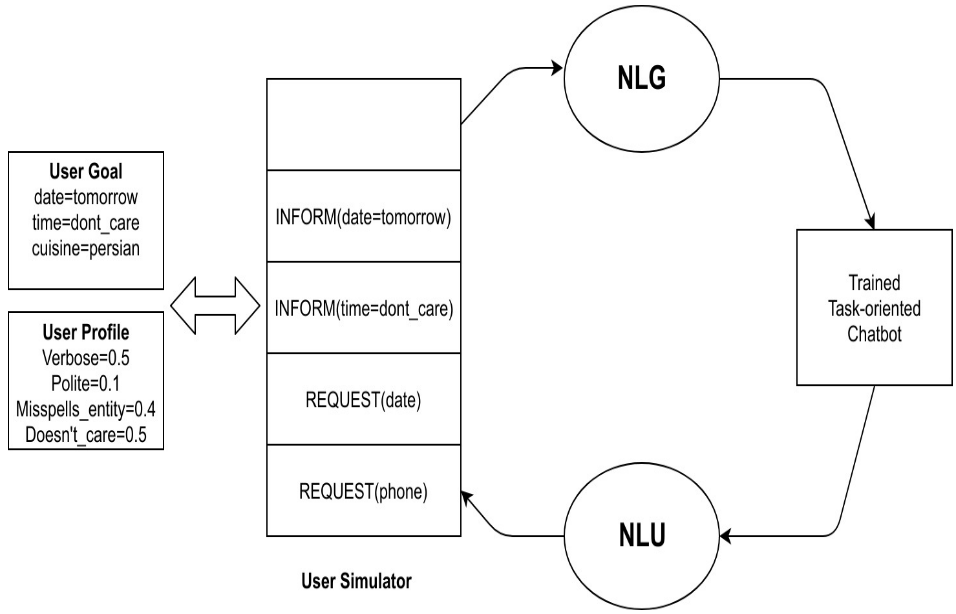

Figure 4.1: Architecture of a pipelined profile-conditioned user simulator

ural language. The generated sentence which conveys what the user simulator intends to convey to the chatbot is then passed to a trained task-oriented chatbot, and the response from the chatbot is passed to the NLU. The NLU component is the only component in our system that needs to be trained in order to identify the intent and the entities from the chatbot response. Based on the intent and list of entities extracted from the chatbot response, an annotation is created an passed to the user simulator in order to update its state. This process is repeated until the conversation ends. The end of the conversation is reached either when the user is satisfied and decides to end the conversation, or a maximum number of turns is reached. The latter is implemented in order to avoid infinite loops in the conversation in which a number of turns are repeated infinitely.

In the following sections, we will detail each of these components individually and explain the methods used for their implementation.

4.3.1 User Profile

User profile is a simple key-value data structure which stores the behavioural characteristics of the user. For instance, *verbosity*, *politeness*, *openness*, and *misspells_entities* are sample characteristics of a user which depict different ways in which the user can express its intention to the chatbot and also if he is likely to make mistakes in uttering the entities. These characteristics are completely domain-based and are intended to model different behaviours of real-user as much as possible in order to create a robust chatbot which can handle different types of users.

Verbosity, politeness, and misspelling are some examples of user characteristics which we can use in the user profile. If a user is verbose, then their utterance is longer compared to others and also it contains more dialogue actions and entities. As an example, a verbose user may say *"I want to book a table for two people tomorrow at Fancy Restaurant"* while a non-verbose one will only say *"I want to book a table for two people"*. A polite user tends to say greetings in the beginning of the conversation and also says goodbye when the conversations ends. The misspelling characteristic of the user makes them prone to misspell entities when they converse with the chatbot. Note that all these characteristics can be formulated into a probabilistic structure in which we can assign probability to each of the user characteristics. The higher the probability, the more the user shows that specific characteristic.

For example in Figure 4.1, the user profile shows the verbosity of user set to 0.5. This is a random number chosen from the uniform distribution on interval [0,1]. It is likely that this user creates long utterances with more than one dialogue action in the conversation. A verbosity 0 means that the user only pops one annotations from the agenda and a verbosity of 1 means that the user always pops the maximum allowed number of annotations from the agenda.

The user is also likely to misspell some entities when conversing with the chatbot with a chance of 0.4. When simulating this user, we misspell each entity with a chance of 0.4 by either adding extra characters or changing some random characters.

4.3.2 User Goal

User goal consists of a list of entities with their respective values. A sample user goal is shown in Figure 4.1. The user goal is a representation of user preference and can be used in the user simulator in order to either respond to some of the questions that the system asks the user in order to determine his preferences (*REQUEST* dialogue action from the chatbot).

In order to make the user goal more realistic, we add multiple entity types to it. The entity types that we use are *fixed*, *flexible*, *multiple-values*, and *open*. The fixed entities are the ones in which user only wants that specific value for that entity. The flexible entities means that the user preference is the value specified for that entity but is OK with other values in case that a specific value is not available. The multiple-value entity means that the user has a list of values for that entities which any of them can be used and the open entity means that the user does not care about the value of that entity.

In Figure 4.1, the user goal has three entities with their respective values. The time entity is of type open values since its value is "*dont_care*". The date and cuisine entities can be of type fixed or flexible since they only have on value (i.e. "*tomorrow*" for date and "*persian*" for cuisine). Note that when running the user simulator, all the entities will be assigned some type randomly and then some random values will be assigned to them based on their type.

4.3.3 Agenda-based User Simulator

The agenda-based user simulator [93] is one of the most widely used user simulators. Agenda is a stack-like data structure which stores annotations. The agenda is a representation of a user goal and its constraints. A sample of agenda-based user simulator is depicted in Figure 4.1. At the top of this agenda-based user simulator, the annotation *INFORM(date=tomorrow)* is available. If this annotation is popped from the user simulator, then it is passed to the NLG and then to the chatbot. We can also pop multiple annotations from the agenda based on the verbosity of the user. Also, it is important to notice that the agenda can be updated based on the answers from the chatbot. This is due to the fact that the agenda of a user simulator should contain all preferences and constraints that the user has in his mind at the current turn in the conversation.

When initializing the user simulator, we create an agenda for each intent that the user wants to cover. We then randomly sample the user goal for each intent and choose some of the entities with their respective values for each of them. Note that for those entities that are not available in the agenda we refer to the user goal in case we need their values during the conversation.

4.3.4 Rasa NLU

Rasa NLU[1] [8] is one of the most widely used open-source NLU frameworks. It provides two main functionalities for training an NLU model for task-oriented chatbots which are user intent classification and entity extraction. For user intent classification, one or a combination of different models such as keyword-based (similar to regex), SVM-based classification with Bag of Words (BOW) features, MITIE intent classification [58], and Facebook's StarSpace model for multilabel classification [124] are used.

For entity extraction, a number of named-entity recognition methods such as regex-based entity extraction, Facebook's duckling[2], CRF-based entity extraction [76], MITIE entity extractor [58], and Spacy's Named-entity Recognition (NER) model [46] is available to be used for entity extraction.

4.3.5 NLG

For NLG in our user simulator, we use a template-based NLG method in which the dialogue action and the list of the entities are converted into natural language. For example, an annotation such as *INFORM(date=tomorrow,time=2 pm)* will be converted into "*tomorrow at 2pm*".

We wanted to make it easier for the research community to use our profile-conditioned user simulator in their research and thus decided to create a framework, called ChatSim, for evaluation of task-oriented chatbots so that both

[1] https://github.com/RasaHQ
[2] https://github.com/facebook/duckling

researchers and practitioners can plug-in their task-oriented chatbot and get some preliminary evaluation of their chatbot in a few minutes.

We will finalize this chapter by explaining the ChatSim library and its components and use cases and will move into the experiments and results in the next chapter.

4.4 ChatSim; A Simulation-based Evaluation Library for Task-oriented Chatbots

ChatSim[3] is a architecture-agnostic evaluation library for task-oriented chatbots which uses different user simulators for the evaluation of task-oriented chatbots. The architecture-agnostic aspect of ChatSim means that it can be used for the evaluation of both end-to-end and also pipelined task-oriented chatbots. Its architecture is modular which consists of an NLU, an NLG, a user simulator, a moderator, and a task-oriented chatbot. The modulatiry of ChatSim allows the user to use different models for the NLG, NLU, and the user simulator components.

Since most of the components of the ChatSim are already explained in Section 4.3, we only briefly dive into its components in the next section. Then we explain the workflow of ChatSim and enumerate its use cases. Finally, we mention some future work on ChatSim and the components/features we could add to it in order to make it more useful for both researchers and practitioners.

4.4.1 ChatSim Components

As mentioned earlier, the components of ChatSim are an NLU, an NLG, a moderator, a user simulator, and the chatbot which we want to evaluate. The NLU component can be both a rule-based system or a machine learning based model. We plan to use Rasa NLU for the initial version of ChatSim, but users can train and add their custom NLU system to the framework if needed. The NLG component can also be both template-based (which we use for the initial version of ChatSim) or a neural sequence-to-sequence model.

[3] https://github.com/msaffarm/chatsim

The moderator is a new component we added to ChatSim in order to moderate the conversation flow and the interaction between the user simulator and the task-oriented chatbot. It makes sure that the conversation between the user simulator and the chatbot is sensible (e.g. the conversation does not turn into a conversation loop) and also controls the maximum number of turns in a single conversation between the user simulator and the chatbot. It also collects statistics about the performance of the chatbot with respect to different evaluation metrics. These metrics could be the average number of turns, and success rate of the task-oriented chatbot. User defined metrics can also be added to this component so that the moderator can keep track of them when running the conversation simulations.

The user simulator is probably the most important component in this framework which serves as the user chatting with the task-oriented chatbot. We use the profile-conditioned user simulator as our first simulator in this library, but like other components it can be replaced with custom simulators.

The chatbot which we want to evaluate can be both an end-to-end or a pipelined task-oriented chatbot. For end-to-end chatbots, we pass the response from the chatbot to the NLU system so that the dialogue action (see 3.3.1) and the entities can be extracted from the response and be passed to the user simulator. For the pipelined task-oriented chatbots, we bypass the NLU component since the response from the chatbot already contains the dialogue action and entities and thus can be directly passed to the user simulator. Note that we are using the profile-conditioned user simulator for this case and since the profile-conditioned user simulator takes dialogue action and entities as input to its agenda, we need to convert the response from the end-to-end chatbot into the format that is understandable by the user simulator. In case a different simulator is used, e.g. an end-to-end user simulator which takes a sequence as input rather than the dialogue action and entities, different strategies need to be implemented in order to make the whole simulation pipeline work. We consider this issue as a future work that needs to be addressed and integrated into the ChatSim.

4.4.2 ChatSim Workflow and Use Cases

The workflow of the simulation used in ChatSim follows the same workflow depicted in Figure 4.1. The moderator creates a user simulator with some user profile and simulates the conversation between the user simulator and the chatbot for a couple of times. It gets the utterance from the user simulator and passes it to the NLG module to create a natural language version of that utterance. That utterance is then given to the chatbot to process and the response from the chatbot is passed to the NLU in order to break it down to dialogue action and entities. The dialogue action and entities are passed to the user simulator so it can update its agenda based on that and create a response.

This cycle continues until either the chatbot can successfully fulfill the needs of the user simulator so that the user simulator ends the conversation or a maximum number of turns is reached and the conversation will be terminated. Note that the task-oriented chatbots are supposed to satisfy the needs of the user with as few turns as possible and therefore the intuition behind the latter criteria to end the conversation is to make sure that the conversation does no take much time to be handled by the task-oriented chatbot. The moderator runs this simulation for a couple of times by initializing the user simulator with different agendas, and the same personality, and collects statistics related to evaluation metrics.

We plan to make ChatSim a general evaluation platform for task-oriented chatbots so that users can easily plug in their chatbots into ChatSim and get evaluation metrics in a few minutes. Users are able to test their chatbots against different user personalities which makes it more realistic and enables them to find out about the possible weaknesses of their chatbots when specific user types are using their chatbot. The use cases of ChatSim would be all the possible use cases of task-oriented chatbots in different domain such a restaurant reservation chatbots, movie booking chatbots, flight booking chatbots, and so on.

4.4.3 ChatSim Desiderata

Our plan to further improve ChatSim is to add more models for different components such as NLU, NLG, and the user simulator and also add user custom evaluation metrics. For the NLU, we could add the feature that user can train neural models which can determine multiple intents in the utterance. For the NLG, we could give the user the ability to train a neural sequence-to-sequence model. We could also add an end-to-end user simulators to ChatSim so that users can train their own user simulator using their own corpus. This feature gives the users the ability to use specific training corpus (e.g., chat logs) for the user simulator which might be close to their target chatbot users.

Chapter 5

Experiments and Results

In this chapter we will discuss both the experiments procedures and the results for each of the contributions (see 1.4) and provide insights about the results. The outline of this chapter is as follows. In Section 5.1 we will talk about the datasets used in this study and also explain the data preparation process. In Section 5.2 we will discuss the training procedure for running the experiments and the detail of the training and inference process. Later in Section 5.4 we will discuss the experiments in detail and discuss the results.

5.1 Data Preparation

5.1.1 Datasets

For this study we use three datasets to compare self-attentional models with other sequence modeling models for the task of task-oriented chatbots. The first dataset that we use is the Dialogue State Tracking Competition 2 (DSTC2) dataset[1] [42] which is the most widely used dataset for research on task-oriented chatbots. Table 5.1 shows the statistics about the training, test, and development sets in the DSTC2 dataset.

Dataset	Number of Slots	Train	Dev	Test
Restaurant	8	1618	1117	500

Table 5.1: Statistics of DSTC2 Dataset

We also used two other datasets recently open-sourced by Google Re-

[1] http://camdial.org/ mh521/dstc/

search [95] which are M2M-sim-M (dataset in movie domain) and M2M-sim-R (dataset in restaurant domain)[2]. M2M stands for Machines Talking to Machines which refers to the framework with which these two datasets were created. In this framework, dialogues are created via dialogue self-play and later augmented via crowdsourcing (see 3.3.2). Table 5.2 shows both movie and restaurant datasets information which we will use to train and evaluate the models.

Dataset	Number of Slots	Train	Dev	Test
Restaurant	9	1116	349	775
Movie	5	384	120	264

Table 5.2: Statistics of M2M Dataset

The M2M dataset has more diversity in both language and dialogue flow compared to the the commonly used DSTC2 dataset which makes it appealing for the task of creating task-oriented chatbots. This is also the reason that we decided to use M2M dataset in our experiments to see how well models can handle a more diversed dataset. Table 5.3 shows the comparison between the M2M and DSTC2 datasets.

Metric	DSTC2 (Train)	M2M Restaurant (Train)
Unique uni-grams/total tokens	0.0049	0.0092
Unique bi-grams/total tokens	0.0177	0.0670
Unique transitions/total turns	0.0982	0.2464

Table 5.3: Comparison of M2M and DSTC2 datasets

5.1.2 Data Preprocessing

We use neural encoder-decoder architectures and model the dialogue generation problem as a sequence-to-sequence learning problem. Consider a sample dialogue $\mathcal{D}$ in the corpus which consists of a number of turns exchanged between the user and the system. $\mathcal{D}$ can be represented as the sequence $\{(u_1, s_1), (u_2, s_2), ..., (u_k, s_k)\}$ where k is the number of turns in this dialogue. At each time step in the conversation, we encode the conversation turns up

[2]https://github.com/google-research-datasets/simulated-dialogue

to that time step, which is the context of the dialogue so far, and the system response after that time step will be used as the target. For example, given we are processing the conversation at time step i, the the context of the conversation so far will be $\{(u_1, s_1, u_2, s_2, ..., u_\mathrm{i})\}$ and the model has to learn to output $\{(s_\mathrm{i})\}$ as the target. Check Appendix A to see some samples of both raw and processed dialogues.

5.2 Training

We used tensor2tensor library[3] [108] in our experiments for training and evaluation of sequence modeling methods. We use Adam optimizer [59] and conduct early stopping [35] to avoid overfitting in our experiments for training each of the models. We also used grid search for hyperparameter tuning for all of the trained models. The hidden state size for all models was set to 128 and the batch size was set to 4096 for all the experiments.

In order to alleviate the response generation problem, as we will discuss in Section 5.3, we use beam search technique for each beam size $\alpha \in \{1, 2, 4\}$ and pick the best result. Note that when $\alpha = 1$, we are using the original greedy search method for generation task.

5.2.1 Sequence-to-Sequence Models

We used the following sequence-to-sequence models for the task of creating end-to-end task-oriented chatbots. We trained 7 different sequence-to-sequence models in our experiments which are:

- LSTM model
- LSTM model with attention mechanism
- Bidirectional LSTM model (Bi-LSTM)
- Bidirectional LSTM model with attention mechanism
- Transformer model

[3] https://github.com/tensorflow/tensor2tensor

- Universal Transformer model
- Universal Transformer model with ACT mechanism

5.2.2 Optimizers

We use Adam optimizer [59] for training the models. We set $\beta_1 = 0.9$, $\beta_2 = 0.997$, and $\epsilon = 10^{-9}$ for the Adam optimizer and started with learning rate of 0.2 with noam learning rate decay schema [109]

5.2.3 Hyperparameter Tuning

We used grid-search for tuning parameters of all the models. For LSTM-based models we tuned number of hidden layers, and dropout. For self-attentional models we tuned number of layers, number of attention heads, and dropout. For the Universal Transformer with ACT module we also tuned the ACT type.

5.2.4 Regularization Methods

We used dropout [98] in [0.7,0.9] range for all the trained models. We also used early stopping [35] while training the models in order to avoid overfitting.

5.3 Inference

We ran the inference phase for each of the models using the best model selected by early stopping criteria and also used beam search as well as greedy search for decoding. The reason for using beam search in the inference phase is explained in the following section.

5.3.1 Beam Search

Most of the sequence-to-sequence learning models used in NLP learn a probabilistic distribution over the data and if they are used in a text generation task such as machine translation, image caption generation, language modeling and also dialogue generation, the model tries to do a search over the vocabulary tokens (e.g., words) based on the learned probability distribution over the whole vocabulary set. This procedure can be done using two methods which are greedy search and beam search. In greedy search method, the model picks the most probable token at each time step given the tokens decoded so far while in the beam search method the model tries to conduct a search over the possible most probable sequence of tokens. For each time step in beam search, the model selects the K most probable tokens and computes the multiplied log probabilities of tokens for each path and this process is repeated for each path until end of sentence token is reached [30].

Beam search has proved to be an essential part in generative NLP task such as neural machine translation [125]. In the case of dialogue generation systems beam search could help alleviate the problem of having many possible valid outputs which do not match with the target but are valid and sensible outputs. Consider the case in which a task-oriented chatbot, trained for restaurant reservation task, in response to user utterance "*Persian food*", generates the response "*what time and day would you like reservation for?*" but the target defined for the system is "*would you like fancy restaurant?*". The response generated by chatbot is a valid response which asks the user about other possible entities but does not match with the defined the target. This problem which is not unique to dialogue generation systems and other generative models have the same issue can be alleviated through usage of beam search.

5.4 Experiments and Results

The experiments for training end-to-end task-oriented chatbots are availabe open-source in *chatbot-exp repository*[4] in github. All the instructions to replicate the results can be found there.

5.4.1 Comparison of Models for Training Task-Oriented Chatbots

The results of running the experiments for the aforementioned models is shown in Table 5.4. The bold numbers show the best performing model in each of

[4]https://github.com/msaffarm/chatbot-exp

the evaluation metrics. As discussed before, for each model we use different beam sizes (bs) in inference time and report the best one.

Dataset	Model	BLEU	Per-Turn	Per-Diag	Entity F1
test	LSTM(bs=1)	5.75	17.70	0.0	5.63
	LSTM+Att.(bs=2)	30.84	18.08	0.15	32.16
	Bi-LSTM(bs=2)	30.38	18.04	0.0	24.34
	Bi-LSTM+Att.(bs=2)	38.64	26.04	0.62	43.52
	Transformer(bs=2)	**51.83**	**39.02**	1.7	**64.20**
	UT(bs=2)	48.17	35.76	**2.93**	61.56
	UT+ACT(bs=2)	39.40	30.00	0.15	61.49
dev	LSTM	16.13	10.33	0.0	6.54
	LSTM+Att.	31.05	18.68	0.31	32.59
	Bi-LSTM	30.92	19.07	0.31	25.91
	Bi-LSTM+Att.	39.12	27.28	0.96	44.15
	Transformer	**54.18**	**41.09**	0.62	**66.02**
	UT	47.89	35.57	**1.23**	61.33
	UT+ACT	39.27	29.30	0.31	62.50

Table 5.4: Evaluation of Models on DSTC2 dataset for both test and development datasets (bs: The best beam size in inference; UT: Universal Transformers)

Our findings presented in Table 5.4 show that self-attentional models outperform common recurrence-based sequence modelling methods in all of the evaluation metrics by a high margin. This shows that the use of self-attentional models (both Transformer and Universal Transformer) can improve the results in training end-to-end task-oriented chatbots and thus they can be both used separately or used as the building blocks of more complex models for training end-to-end chatbots as a replacement for recurrence-based blocks such as LSTM.

The results of the Transformer model outperforms other self-attentional models, suggesting its effectiveness in this task, but comparing the results of the Transformer model and the Universal Transformer shows that the Universal Transformer can generate very good results for this task, close to those of Transformer, but not exceeding them.

Another observation from the results of Table 5.4 is that using attention mechanism on top of recurrence-based models such as LSTM and Bi-LSTM can significantly increase the performance of the models, which highlights the power of the attention mechanism used in sequence modelling methods. Later in Section 5.4.3, we further investigate the effectiveness of the attention mechanism used in sequence modelling tasks both in recurrence-based and self-attentional models.

Moreover, we tested the models on the two aforementioned datasets, which are M2M movie (M2M-M) and M2M restaurant (M2M-R) datasets. The motivation for us to run the experiments on these datasets was to check whether the check the effect of dataset on training end-to-end task-oriented chatbots and see of self-attentional models supremacy is consistent when used on other datasets and the results are not corpus-biased. The results of our experiments are shown in Table 5.5.

Dataset	Model	BLEU	Per-Turn	Per-Diag	Entity F1
M2M-R (test)	LSTM(bs=2)	6.00	2.3	0.0	7.99
	LSTM+Att.(bs=1)	7.9	1.84	0.0	16.77
	Bi-LSTM(bs=1)	8.15	1.8	0.0	19.61
	Bi-LSTM+Att.(bs=1)	8.3	0.97	0.0	24.12
	Transformer(bs=1)	10.28	1.76	0.0	36.92
	UT(bs=2)	9.15	1.88	0.0	25.44
	UT+ACT(bs=2)	8.54	1.43	0.0	23.12
M2M-M (test)	LSTM(bs=4)	7.7	3.36	0.0	31.07
	LSTM+Att.(bs=2)	8.3	3.27	0.0	31.18
	Bi-LSTM(bs=2)	9.6	2.09	0.0	28.09
	Bi-LSTM+Att.(bs=2)	10.62	2.54	0.0	32.43
	Transformer(bs=1)	11.95	2.36	0.0	39.89
	UT(bs=2)	10.87	3.15	0.0	34.15
	UT+ACT(bs=2)	10.48	2.46	0.0	32.76

Table 5.5: Evaluation of models on M2M restaurant (M2M-R) and movie (M2M-M) dataset for test datasets (bs: The best beam size in inference; UT: Universal Transformers)

As evident in this table, the numbers drop significantly compared to the DSTC2 dataset numbers. We think this is due to the fact that the diversity of M2M dataset is much higher than the DSTC2 dataset while the corpus size (i.e. train set size) is lower. The lack of enough training data, given the more diversity in M2M dataset in both bi-grams and possible scenarios, leads to lower numbers in the evaluation metrics. However, the self-attentional model still outperform other models in the BLEU and Entity-F1 metrics.

We also examined the results (chat corpus) of the trained models and noticed an interesting fact. In some cases, the model creates a plausible response but the target response is a different one in the M2M training corpus and thus the evaluation metrics cannot accurately measure the performance of the trained chatbot. As we mentioned this problem in the inference section (see **??**), one of the possible solutions to this problem is the usage of beam search but our experiments showed that this is not enough. This motivated us to create the user-simulation evaluation framework which tries to capture the alternative valid responses by the chatbot instead of punishing it responding with utterances which are not the same as the target response but still are valid ones.

5.4.2 Time Performance Comparison

Table 5.6 shows the time performance of the models on DSTC2 dataset. Note that in order to get a fair time performance comparison, we trained the models with the same batch size and on the same GPU. These numbers are for the best performing model (in terms of evaluation loss and selected using the early stopping method) for each of the sequence modelling methods. Time to Convergence (T2C) shows the approximate time that the model was trained to converge. We also show the loss in the development set for that specific checkpoint.

Model	T2C (sec)	Dev Loss
LSTM	1100	0.89
LSTM+Att	1305	0.62
Bi-LSTM	1865	0.60
Bi-LSTM+Att	2120	0.49
Transformer	**612**	**0.31**
UT	1939	0.36
UT+ACT	665	0.33

Table 5.6: Comparison of convergence performance of the models on DSTC2 dataset

According to the results in Table 5.6 which shows the time performance of the models, the Transformer model beats all the models in terms of convergence time. Comparing the results of the Universal Transformer and Universal Transformer with adaptive computation time (ACT) technique shows the effectiveness of the ACT technique in terms of reducing the convergence time, but it can not beat the other self-attentional models in terms of evaluation metrics as evident in Table 5.4.

5.4.3 Effect of (Self-)Attention Mechanism

Self-attentional models rely on self-attention mechanism for sequence modelling. Recurrence-based models such as LSTM and Bi-LSTM can also be augmented in order to increase their performance, as evident in Table 5.4 which shows the increase in the performance of both LSTM and Bi-LSTM when augmented with attention mechanism. This leads to the question whether we can increase the performance of recurrence-based models by adding multiple attention heads, similar to the multi-head self-attention mechanism used in self-attentional models, and outperform the self-attentional models. In other words, can we beat the Transformer model by adding more attention heads to the Bi-LSTM model (while keeping it computationally reasonable to train) for the task of training and end-to-end task oriented chatbot?

To investigate this question, we ran a number of experiments in which we added multiple attention heads on top of Bi-LSTM model and also tried a different number of self-attention heads in self-attentional models in order to compare their performance for this specific task. Table 5.7 shows the results of these experiments. Note that the models in Table 5.7 are actually the best models that we found in the experiments in Table 5.4 and we only changed one parameter for each of them, i.e. the number of attention heads in recurrence-based models and the number of self-attentional heads in self-attentional models, keeping all other parameters unchanged. We also report the results of models with beam size of 2 in inference time. This is to make sure that the only parameter that can affect the results is the number of attention heads (NH in Table 5.7). We increased the number of attention heads in the Bi-LSTM model up to 64 heads to see its performance change. Note that increasing the number of attention heads makes the training time intractable and

Dataset	Model	BLEU	Per-Turn	Per-Diag	Entity F1
test	Bi-LSTM+Att.[NH=1]	38.64	26.04	0.62	43.52
	Bi-LSTM+Att.[NH=4]	42.23	29.01	0.92	48.06
	Bi-LSTM+Att.[NH=8]	42.61	28.18	0.77	49.90
	Bi-LSTM+Att.[NH=16]	**43.11**	**30.34**	**0.61**	**50.87**
	Bi-LSTM+Att.[NH=32]	**48.62**	**36.46**	**1.85**	**59.8**
	Bi-LSTM+Att.[NH=64]	**47.33**	**33.17**	**1.23**	**56.49**
	Transformer[NH=1]	**45.90**	**36.64**	**1.7**	**57.55**
	Transformer[NH=4]	**51.83**	**39.02**	**1.7**	**64.20**
	Transformer[NH=8]	51.37	39.45	3.24	62.38
	UT[NH=1]	43.02	31.20	1.54	60.10
	UT[NH=8]	48.17	35.76	2.93	61.56
	UT+ACT[NH=1]	34.98	25.66	0.46	51.32
	UT+ACT[NH=8]	36.29	24.97	0.31	55.27
dev	Bi-LSTM+Att.[NH=1]	39.12	27.28	0.96	44.15
	Bi-LSTM+Att.[NH=4]	40.47	27.64	0.93	48.10
	Bi-LSTM+Att. [NH=8]	42.78	28.36	0.31	50.05
	Bi-LSTM+Att.[NH=16]	**42.88**	**30.36**	**0.93**	**52.09**
	Bi-LSTM+Att.[NH=32]	**49.36**	**38.24**	**0.61**	**61.26**
	Bi-LSTM+Att.[NH=64]	**47.28**	**33.12**	**0.93**	**56.86**
	Transformer[NH=1]	**47.86**	**38.33**	**1.85**	**60.37**
	Transformer[NH=4]	**54.18**	**41.09**	**0.62**	**66.02**
	Transformer[NH=8]	51.54	39.42	1.54	63.56
	UT[NH=1]	43.01	32.12	1.58	60.42
	UT[NH=8]	47.89	35.57	1.23	61.33
	UT+ACT[NH=1]	35.74	26.46	0.31	52.71
	UT+ACT[NH=8]	38.95	27.10	0.31	57.02

Table 5.7: Evaluation of effect of self-attention mechanism using DSTC2 dataset (Att: Attention mechanism; UT: Universal Transformers; ACT: Adaptive Computation Time; NH: Number of attention heads

time consuming while the model size would increase significantly as shown in Table 5.8. Furthermore, by observing the results of the Bi-LSTM+Att model in Table 5.7 (both test and development set) we can see that Bi-LSTM performance is decreases and thus there is no need to increase the attention heads further.

Our findings in Table 5.7 show that self-attention mechanism can outperform recurrence-based models even if the recurrence-based models have multiple attention heads. If we compare the results of the highlighted records in Table 5.7, we can see that the Bi-LSTM model with 64 attention heads cannot beat the best Transformer model with NH=4 (the best model trained on DSTC2 dataset) and also its results are very close to the Transformer model with NH=1. This observation clearly depicts the power of self-attentional based models and demonstrates that attention mechanism used in self-attentional models as the backbone for learning, outperforms recurrence-based models even if they are augmented with multiple attention heads.

Model	T2C (sec)	Dev Loss
Bi-LSTM+Att.[NH=1]	2120	0.49
Bi-LSTM+Att.[NH=4]	3098	0.47
Bi-LSTM+Att.[NH=8]	3530	0.44
Bi-LSTM+Att.[NH=16]	3856	0.44
Bi-LSTM+Att.[NH=32]	7320	0.36
Bi-LSTM+Att.[NH=64]	9874	0.38
Transformer[NH=1]	375	0.33
Transformer[NH=4]	612	0.31
Transformer[NH=8]	476	0.31

Table 5.8: Comparison of convergence performance of the models

5.4.4 Profile-Conditioned User Simulator

We used a trained end-to-end chatbot on movie domain, and a rule-based one in the evaluation of profile-conditioned user simulator. Our current NLU system needs full-annotated training data, i.e. data with annotated dialogue actions and entities, and thus we had to use the chatbots trained on M2M dataset. The DSTC2 dataset lacks the dialogue action annotation and thus could not be used for the purpose of training the NLU system in the profile-conditioned user simulator. Therefore, we used the M2M-M (M2M dataset in movie domain) dataset and used it for training the chatbots in our evaluation process. All of our trained models (i.e. the NLU and the chatbot) are integrated with the ChatSim library.

Note that we train the NLU model using the same dataset with which we train the chatbot. This is to make sure that the trained NLU module is not of source a error while the chatbot utterances are passed to it and that it can fully understand the chatbot utterances and extract the dialogue action and entities from it.

We ran 100 simulations using different goals and one specific personality for the user simulator and measured two metrics for the chatbot which are the mean number of turns in conversations and the success rate of the chatbot. Given the randomness in user profile, the experiments were ran 10 times and mean and standard deviation of metrics were calculated. The user goals were chosen randomly from slots in the M2M-M dataset and their respective values. These slots are time, date, movie, number of tickets (people), and the theatre name. For each of these slots we selected some random values from the database and used it in our simulations. Furthermore, the user profile for this specific experiment is ”*verbosity=0.3, politeness=0.9, misspell_entities=0.1, agenda_size=0.85*“.

We evaluated two different chatbots trained on the M2M-M dataset. The details about these chatbots are as follows:

- We selected the best self-attentional model trained on M2M-M dataset. This model is a Transformer model which its automatic evaluation results are depicted in Table 5.5. Note that this chatbot is an end-to-end model which needs a lot of training data and since the M2M-M dataset is a small dataset we expect the chatbot not to perform very well.

- In order to show that the evaluation system can compare two different chatbots with different capabilities, we decided to create a rule-based chatbot and evaluate it using our evaluation framework. The rule-based chatbot uses domain-specific (movie domain in our case) database and ontology in order to make the right decision given the current state of the conversation. Note that the decision making part of this chatbot is rule-based rather than an end-to-end model. We expect this chatbot to perform very well given the fact that it’s a rule-based one in which we can hard-code all the required rules. The rules used for this chatbot are in shown in Appendix B.

We ran experiments for each of aforementioned chatbots. The maximum allowed number of turns per conversation was set to 15 for this experiment and if the number of turns in a conversation exceeded this threshold, then it would be assumed a failed conversation. The results are depicted in Table 5.9.

Model	Avg. Number of Turns per Conversation	Success Rate
Transformer	8.2±0.93	9.7±1.15%
Rule-based	9.5±1.17	100±0.0%

Table 5.9: Chatbot evaluation results using profile-conditioned user simulator

Note that the Transformer chatbot is not able to keep track of the entities properly. That is the reason that the success rate of the chatbot is low. Most of the times, the chatbot failed to create a reservation based on the user preferences and even if the chatbot made the reservation it was considered a failed conversation since the reservation was not compatible with user preference. This is due to the fact that the chatbot does not have access to a database and can not connect to it and also it can not keep track of the user preferences properly. Also, it is not able to handle misspelling in the entity values. These are the reason why the average number of turns in the Transformer chatbot is lower than the rule-based one. However, in some cases the chatbot managed to create a correct reservation based on the user preferences.

We use anecdotal evaluation to show that our evaluation framework can do the assessment of task-oriented chatbots properly. As a future work, it would be very insightful to conduct a user study to investigate the correlation between the user evaluation metrics and the ChatSim's metrics.

Here's a sample of a successful conversation for the Transformer chatbot:

User: hi number of people is 1

System: what date and time and movie ?

User: date is tomorrow time is 2 pm

System: what date and time and movie ?

User: movie is avatar and theater is angelika

System: 2 pm is unavailable , choose between 6 pm and 7.15 pm?

User: time is 6 pm

System: purchase confirmed for 1 ticket to avatar at 6 pm at angelika theater for tomorrow .

User: thank you

Here's a sample of a failed conversation for the Transformer chatbot:

User: hi number of people is 1

System: what date and time and movie ?

User: date is saturday time is 6 pm

System: what date and time and movie ?

User: movie is avatar and theater is angelika

System: 6 pm is unavailable , choose between 2 pm and 7.15 pm?

User: time is 2 pm

System: purchase confirmed for 1 ticket to avatar at 6 pm at angelika theater for saturday .

User: good bye

In this case the user terminates the conversation since the chatbot fails to make the right reservation.

Here is a sample of a successful conversation between the rule-based chatbot and the user simulator:

User: hi number of people is 1 theater is aquarius

System: what date ?

User: date is today

System: what time ?

User: time is 2 pm

System: what is the movie ?

User: movie is avatar

System: Do you confirm movie is avatar date is today for 2 pm at aquarius for 1 person ?

User: that is right

System: purchase made successfully

User: thank you

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we investigated the usage of self-attentional models for training end-to-end task-oriented chatbots and showed that self-attentional models can be used for this task. We compared self-attentional models against state-of-the-art recurrence-based sequence modelling methods which are commonly used for training task-oriented chatbots on three different datasets (DSTC2, M2M-M, and M2M-R) and evaluated them both using common automatic evaluation metrics used in task-oriented chatbots and also their training time performance. Furthermore, we investigated the effectiveness of self-attention mechanism versus recurrence-based sequence modelling for training end-to-end task-oriented chatbots. Our findings show that:

- Transformers and Universal-Transformers are indeed effective at generating appropriate responses in task-oriented chatbot systems. In fact, their performance is even better than the typically used deep learning architectures both in evaluation metrics used for the evaluation of task-oriented chatbots and also their training time performance.

- It is not always recommended to use end-to-end models for training task-oriented chatbots. As mentioned before, end-to-end models need a considerable amount of training data in order to learn be able to learn all the patterns in the data. This is not always possible since in many cases we do not have enough training data and training end-to-end models will not result in chatbots with good functionality. The alternative to end-to-end models is pipelined and rule-based chatbots which need less training data and specific rules can be hard-coded in them.

- Self-attention mechanism contributes most to the performance of self-attentional models (at least for the task of creating end-to-end task-oriented chatbots). Moreover, recurrence-based models can not beat self-attentional models even if they are augmented with multiple attention heads.

Moreover, we introduced the profile-conditioned user simulator as an evaluation method for task-oriented chatbots. We open-sourced ChatSim which is an architecture-agnostic, simulation-based evaluation platform for task-oriented chatbots and added the profile-conditioned user simulator to it. We evaluated a trained end-to-end chatbot with the profile-conditioned user simulator and showed that this method can be used as an evaluation method for task-oriented chatbots.

6.2 Improving Task-Oriented Dialogue Generation Systems

In future work, it would be interesting to compare the performance of self-attentional models (specifically the winning Transformer model) against other end-to-end architectures such as the Memory Augmented neural networks [28]. The current methods that we used in this thesis do not have explicit memories to be used as a knowledge base and thus it would be a great future work to augment self-attentional models with memory and thus increase their capabilities in both keeping track of the entity values. Another possible future work is to augment the self-attentional models with components which can query some database and be able to retrieve information from them. This allows seamless integration of these models with any database on custom domains and thus can dramatically increase their usability.

Another possible future work is to investigate the reason why self-attention mechanism can beat the recurrence-based models even if we add multiple layers of attention on top of the recurrence-based models. Self-attentional models can model long sequences very well since they don't suffer from problem like the vanishing/exploding gradients. On the other hand, the attention mechanism allows the recurrence-based model to capture the information in the long sequences. It would be interesting to fully understand the functionality of the attention mechanism and know why it is more useful when used in self-attentional models compared to its usage as an augmentation on recurrence-based models.

6.3 Improving Profile-Conditioned User Simulator

Adding some more granularity levels to entity types can also make the user simulator more realistic. One idea for increasing the granularity level of entities is adding *optional* or *required* types to the entities. The optional entity is a type of entity in which its value is not necessary for the conversation to be finished and does not affect the success rate of the conversation. For example, phone number of the restaurant can be an optional entity in restaurant booking chatbots which the user can ask the chatbot but is not essential for the task of booking a restaurant. On the other hand, required entities are necessary for task completion and thus needs to have proper values. The date of booking a restaurant is an example of a required entity for a restaurant booking chatbot.

6.4 Improving the ChatSim Library

For improving the ChatSim library all the desiderata mentioned in Section 4.4.3 are potential future work. Adding a user interface on top of the library could also improve its capabilities and make it easier for the users to experiment with the library and integrate it with their custom dashboards used in their own chatbot platforms. Integration of ChatSim with Rasa X[1] could

[1] https://rasa.com/docs/rasa-x/

also be a possible future work. Rasa X is a powerful tool for developing task-oriented chatbots with a user-friendly user interface and since Rasa is the leading chatbot platform used by many chatbot developers.

References

[1] S. A. Abdul-Kader and J. Woods, "Survey on chatbot design techniques in speech conversation systems," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 7, 2015.

[2] K. Ahmed, N. S. Keskar, and R. Socher, "Weighted transformer network for machine translation," *ArXiv preprint arXiv:1711.02132*, 2017.

[3] L. E. Asri, J. He, and K. Suleman, "A sequence-to-sequence model for user simulation in spoken dialogue systems," *ArXiv:1607.00070*, 2016.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *ArXiv preprint arXiv:1607.06450*, 2016.

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

[6] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *ArXiv preprint arXiv:1803.01271*, 2018.

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

[8] T. Bocklisch, J. Faulker, N. Pawlowski, and A. Nichol, "Rasa: Open source language understanding and dialogue management," *Preprint on Arxiv arXiv:1712.05181*, 2017.

[9] A. Bordes, Y.-L. Boureau, and J. Weston, "Learning end-to-end goal-oriented dialog," *ArXiv preprint arXiv:1605.07683*, 2016.

[10] D. Braun, A. Hernandez-Mendez, F. Matthes, and M. Langen, "Evaluating natural language understanding services for conversational question answering systems," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, 2017, pp. 174–185.

[11] P. Budzianowski, T.-H. Wen, B.-H. Tseng, I. Casanueva, S. Ultes, O. Ramadan, and M. Gašić, "Multiwoz-a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling," *ArXiv:1810.00278*, 2018.

[12] V. Cerf, "Parry encounters the doctor," Tech. Rep., 1973.

[13] S. Chandramohan, M. Geist, F. Lefevre, and O. Pietquin, "User simulation in dialogue systems using inverse reinforcement learning," in *Interspeech 2011*, 2011, pp. 1025–1028.

[14] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems: Recent advances and new frontiers," *ACM SIGKDD Explorations Newsletter*, vol. 19, no. 2, pp. 25–35, 2017.

[15] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *ArXiv preprint arXiv:1601.06733*, 2016.

[16] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *ArXiv preprint arXiv:1409.1259*, 2014.

[17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *ArXiv preprint arXiv:1610.02357*, 2016.

[18] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[19] Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Language modeling with longer-term dependency," 2018.

[20] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," *ArXiv preprint arXiv:1612.08083*, 2016.

[21] E. M. De Novais, T. D. Tadeu, and I. Paraboni, "Improved text generation using n-gram statistics," in *Ibero-American Conference on Artificial Intelligence*, Springer, 2010, pp. 316–325.

[22] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Universal transformers," *ArXiv preprint arXiv:1807.03819*, 2018.

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

[24] M. Denil, L. Bazzani, H. Larochelle, and N. de Freitas, "Learning where to attend with deep architectures for image tracking," *Neural computation*, vol. 24, no. 8, pp. 2151–2184, 2012.

[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv preprint arXiv:1810.04805*, 2018.

[26] M. v. Eeuwen, "Mobile conversational commerce: Messenger chatbots as the next interface between businesses and consumers," Master's thesis, University of Twente, 2017.

[27] 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.

[28] ——, "Key-value retrieval networks for task-oriented dialogue," *ArXiv preprint arXiv:1705.05414*, 2017.

[29] K. K. Fitzpatrick, A. Darcy, and M. Vierhile, "Delivering cognitive behavior therapy to young adults with symptoms of depression and anxiety using a fully automated conversational agent (woebot): A randomized controlled trial," *JMIR mental health*, vol. 4, no. 2, 2017.

[30] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," *ArXiv preprint arXiv:1702.01806*, 2017.

[31] J. Gao, M. Galley, L. Li, *et al.*, "Neural approaches to conversational ai," *Foundations and Trends in Information Retrieval*, vol. 13, no. 2-3, pp. 127–298, 2019.

[32] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *ArXiv preprint arXiv:1611.02344*, 2016.

[33] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *ArXiv:1705.03122*, 2017.

[34] D. Goddeau, H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchai, "A form-based dialogue manager for spoken language applications," in *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, IEEE, vol. 2, 1996, pp. 701–704.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[36] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv preprint arXiv:1308.0850*, 2013.

[37] ——, "Adaptive computation time for recurrent neural networks," *Arxiv preprint arXiv:1603.08983*, 2016.

[38] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[39] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[40] I. Gur, D. Hakkani-Tur, G. Tur, and P. Shah, "User modeling for task oriented dialogues," *ArXiv preprint arXiv:1811.04369*, 2018.

[41] H. B. Hashemi, A. Asiaee, and R. Kraft, "Query intent detection using convolutional neural networks," in *International Conference on Web Search and Data Mining, Workshop on Query Understanding*, 2016.

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

[43] M. Henderson, B. Thomson, and S. Young, "Deep neural network approach for the dialog state tracking challenge," in *Proceedings of the SIGDIAL 2013 Conference*, 2013, pp. 467–471.

[44] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Networks*, vol. 4, no. 1, pp. 61–66, 1991.

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

[46] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1373–1378. [Online]. Available: `https://aclweb.org/anthology/D/D15/D15-1162`.

[47] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation," *Acm Sigact News*, vol. 32, no. 1, pp. 60–65, 2001.

[48] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *ArXiv preprint arXiv:1508.01991*, 2015.

[49] H. Jaeger, *Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik Bonn, 2002, vol. 5.

[50] R. Jiang, R. E. Banchs, and H. Li, "Evaluating and combining name entity recognition systems," in *Proceedings of the Sixth Named Entity Workshop*, 2016, pp. 21–27.

[51] R. Johnson and T. Zhang, "Deep pyramid convolutional neural networks for text categorization," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017, pp. 562–570.

[52] Ł. Kaiser. (2017). Tensor2tensor transformers new deep models for nlp, [Online]. Available: `https://nlp.stanford.edu/seminar/details/lkaiser.pdf` (visited on 12/19/2018).

[53] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," *ArXiv:1706.05137*, 2017.

[54] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *ArXiv preprint arXiv:1610.10099*, 2016.

[55] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *ArXiv:1404.2188*, 2014.

[56] J. F. Kelley, "An iterative design methodology for user-friendly natural language office information applications," *ACM Transactions on Information Systems (TOIS)*, vol. 2, no. 1, pp. 26–41, 1984.

[57] Y. Kim, "Convolutional neural networks for sentence classification," *ArXiv preprint arXiv:1408.5882*, 2014.

[58] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1755–1758, 2009.

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

[60] F. Kreyssig, I. Casanueva, P. Budzianowski, and M. Gasic, "Neural user simulation for corpus-based policy optimisation for spoken dialogue systems," *ArXiv preprint arXiv:1805.06966*, 2018.

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

[62] H. Larochelle and G. E. Hinton, "Learning to combine foveal glimpses with a third-order boltzmann machine," in *Advances in neural information processing systems*, 2010, pp. 1243–1251.

[63] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[64] H. J. Levesque, *Common sense, the turing test, and the quest for real ai: Reflections on natural and artificial intelligence*. MIT Press, 2017.

[65] Y. Leviathan. (2018). Google duplex: An ai system for accomplishing real-world tasks over the phonea, [Online]. Available: `https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html` (visited on 10/31/2018).

[66] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan, "A diversity-promoting objective function for neural conversation models," *ArXiv preprint arXiv:1510.03055*, 2015.

[67] X. Li, Z. C. Lipton, B. Dhingra, L. Li, J. Gao, and Y.-N. Chen, "A user simulator for task-completion dialogues," *ArXiv:1612.05688*, 2016.

[68] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *ArXiv preprint arXiv:1703.03130*, 2017.

[69] C.-W. Liu, R. Lowe, I. V. Serban, M. Noseworthy, L. Charlin, and J. Pineau, "How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation," *ArXiv preprint arXiv:1603.08023*, 2016.

[70] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *ArXiv preprint arXiv:1801.10198*, 2018.

[71] L. Loubet. (2018). Beauty.bot launches as source of everything worth knowing about beauty bots, [Online]. Available: `https://www.automat.ai/2018/02/everything-worth-knowing-beauty-bots/` (visited on 10/31/2018).

[72] M. Luong, E. Brevdo, and R. Zhao, "Neural machine translation tutorial," *Https://github.com/tensorflow/nmt*, 2017.

[73] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *ArXiv:1508.04025*, 2015.

[74] M. d. G. B. Marietto, R. V. de Aguiar, G. d. O. Barbosa, W. T. Botelho, E. Pimentel, R. d. S. França, and V. L. da Silva, "Artificial intelligence markup language: A brief tutorial," *ArXiv preprint arXiv:1307.3091*, 2013.

[75] M. L. Mauldin, "Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition," in *AAAI*, vol. 94, 1994, pp. 16–21.

[76] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, Association for Computational Linguistics, 2003, pp. 188–191.

[77] M. Mendoza and J. Zamora, "Identifying the intent of a user query using support vector machines," in *International Symposium on String Processing and Information Retrieval*, Springer, 2009, pp. 131–142.

[78] C. Messina. (2015). Conversational commerce, messaging apps bring the point of sale to you, [Online]. Available: `https://medium.com/chris-messina/conversational-commerce-92e0bccfc3ff` (visited on 07/13/2019).

[79] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston, "Key-value memory networks for directly reading documents," *ArXiv preprint arXiv:1606.03126*, 2016.

[80] C. Olah. (2015). Understanding lstm networks, [Online]. Available: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (visited on 01/08/2019).

[81] J. Orkin and D. Roy, "The restaurant game: Learning social behavior and language from thousands of players online," *Journal of Game Development*, vol. 3, no. 1, pp. 39–60, 2007.

[82] 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 association for computational linguistics*, Association for Computational Linguistics, 2002, pp. 311–318.

[83] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," *ArXiv preprint arXiv:1606.01933*, 2016.

[84] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, and A. Ku, "Image transformer," *ArXiv preprint arXiv:1802.05751*, 2018.

[85] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[86] R. Pittman. (2016). Say "hello" to ebay shopbot beta, [Online]. Available: `https://www.ebayinc.com/stories/news/say-hello-to-ebay-shopbot-beta/` (visited on 10/26/2018).

[87] A. Radford, "Improving language understanding by generative pre-training," 2018.

[88] A. Ram, R. Prasad, C. Khatri, A. Venkatesh, R. Gabriel, Q. Liu, J. Nunn, B. Hedayatnia, M. Cheng, A. Nagar, *et al.*, "Conversational ai: The science behind the alexa prize," *ArXiv preprint arXiv:1801.03604*, 2018.

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

[90] S. Razzaki, A. Baker, Y. Perov, K. Middleton, J. Baxter, D. Mullarkey, D. Sangar, M. Taliercio, M. Butt, A. Majeed, *et al.*, "A comparative study of artificial intelligence and human doctors for the purpose of triage and diagnosis," *ArXiv preprint arXiv:1806.10698*, 2018.

[91] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, "Character-level language modeling with deeper self-attention," *ArXiv preprint arXiv:1808.04444*, 2018.

[92] A. Ritter, C. Cherry, and W. B. Dolan, "Data-driven response generation in social media," in *Proceedings of the conference on empirical methods in natural language processing*, Association for Computational Linguistics, 2011, pp. 583–593.

[93] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, "Agenda-based user simulation for bootstrapping a pomdp dialogue system," in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, Association for Computational Linguistics, 2007, pp. 149–152.

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

[95] P. Shah, D. Hakkani-Tür, G. Tür, A. Rastogi, A. Bapna, N. Nayak, and L. Heck, "Building a conversational agent overnight with dialogue self-play," *ArXiv preprint arXiv:1801.04871*, 2018.

[96] T. Shen, T. Zhou, G. Long, J. Jiang, S. Wang, and C. Zhang, "Reinforced self-attention network: A hybrid of hard and soft attention for sequence modeling," *ArXiv preprint arXiv:1801.10296*, 2018.

[97] H.-y. Shum, X.-d. He, and D. Li, "From eliza to xiaoice: Challenges and opportunities with social chatbots," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 10–26, 2018.

[98] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[99] M. D. Stephan Gouws. (2018). Moving beyond translation with the universal transformer, [Online]. Available: `https://ai.googleblog.com/2018/08/moving-beyond-translation-with.html` (visited on 12/20/2018).

[100] G. P. Styan, "Hadamard products and multivariate statistical analysis," *Linear algebra and its applications*, vol. 6, pp. 217–240, 1973.

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

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

[103] Z. Tan, M. Wang, J. Xie, Y. Chen, and X. Shi, "Deep semantic role labeling with self-attention," *ArXiv preprint arXiv:1712.01586*, 2017.

[104] G. Tang, M. Müller, A. Rios, and R. Sennrich, "Why self-attention? a targeted evaluation of neural machine translation architectures," *ArXiv preprint arXiv:1808.08946*, 2018.

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

[106] J. Uszkoreit. (2017). Transformer: A novel neural network architecture for language understanding, [Online]. Available: `https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html` (visited on 12/20/2018).

[107] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio.," in *SSW*, 2016, p. 125.

[108] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, "Tensor2tensor for neural machine translation," *CoRR*, vol. abs/1803.07416, 2018. [Online]. Available: `http://arxiv.org/abs/1803.07416`.

[109] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

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

[111] J. Wakefield. (2015). Intelligent machines: Chatting with the bots, [Online]. Available: `https://www.bbc.com/news/technology-33825358` (visited on 11/10/2018).

[112] R. S. Wallace, "The anatomy of alice," in *Parsing the Turing Test*, Springer, 2009, pp. 181–210.

[113] A. Wangperawong, "Attending to mathematical language with transformers," *ArXiv preprint arXiv:1812.02825*, 2018.

[114] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[115] T.-H. Wen, M. Gasic, D. Kim, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young, "Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking," *ArXiv preprint arXiv:1508.01755*, 2015.

[116] T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young, "Semantically conditioned lstm-based natural language generation for spoken dialogue systems," *ArXiv preprint arXiv:1508.01745*, 2015.

[117] T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young, "A network-based end-to-end trainable task-oriented dialogue system," *ArXiv preprint arXiv:1604.04562*, 2016.

[118] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[119] J. D. Williams, K. Asadi, and G. Zweig, "Hybrid code networks: Practical and efficient end-to-end dialog control with supervised and reinforcement learning," *ArXiv preprint arXiv:1702.03274*, 2017.

[120] J. D. Williams, E. Kamal, M. Ashour, H. Amr, J. Miller, and G. Zweig, "Fast and easy language understanding for dialog systems with microsoft language understanding intelligent service (luis)," in *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2015, pp. 159–161.

[121] J. D. Williams and G. Zweig, "End-to-end lstm-based dialog control optimized with supervised and reinforcement learning," *ArXiv preprint arXiv:1606.01269*, 2016.

[122] S. Wiseman and A. M. Rush, "Sequence-to-sequence learning as beam-search optimization," *ArXiv preprint arXiv:1606.02960*, 2016.

[123] M. A. Writer. (2018). Much more than a chatbot: China's xiaoice mixes ai with emotions and wins over millions of fans, [Online]. Available: `https://news.microsoft.com/apac/features/much-more-than-a-chatbot-chinas-xiaoice-mixes-ai-with-emotions-and-wins-over-millions-of-fans/` (visited on 09/12/2019).

[124] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston, "Starspace: Embed all the things!" *ArXiv preprint arXiv:1709.03856*, 2017.

[125] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *ArXiv preprint arXiv:1609.08144*, 2016.

[126] C. Xiong, V. Zhong, and R. Socher, "Dcn+: Mixed objective and deep residual coattention for question answering," *ArXiv:1711.00106*, 2017.

[127] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, "Attngan: Fine-grained text to image generation with attentional generative adversarial networks," *ArXiv preprint*, 2017.

[128] W. Yin and H. Schütze, "Multichannel variable-size convolution for sentence classification," *ArXiv preprint arXiv:1603.04513*, 2016.

[129] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *Ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[130] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *ArXiv preprint arXiv:1511.07122*, 2015.

[131] D. Zelenko, C. Aone, and J. Tibbetts, "Coreference resolution for information extraction," in *Proceedings of the Conference on Reference Resolution and Its Applications*, 2004.

[132] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *ArXiv:1805.08318*, 2018.

[133] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, and C. Xiong, "End-to-end dense video captioning with masked transformer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8739–8748.

Appendix A

Data Preparation Process

In this appendix we present two sample dialogues from the datasets that we used for training the models and demonstrate the process which we use to create the training dataset fed into end-to-end models.

A.1 DSTC2 Dataset Sample

Original Dialogue:

User: i'm looking for a restaurant in any area that serves belgian food
System: I'm sorry but there is no restaurant serving belgian food.
User: how about gastro of town please
System: I'm sorry but there is no restaurant serving belgian food.
User: there a restaurant that serves gastropub
System: Could you please repeat that?
User: i'm looking for a restaurant that serves gastropub
System: Ok, a restaurant in any part of town is that right?
User: yes
System: There are # restaurants serving gastropub food in any part of town . What price range would you like?
User: i don't care
System: the cow pizza kitchen and bar serves gastropub food.
User: what is the address
System: Sure , the cow pizza kitchen and bar is on Corn Exchange Street.

User: and the phone number

System: The phone number of the cow pizza kitchen and bar is 01223 308871.

User: thank you good bye

Processed Dialogue:

Input 1: i'm looking for a restaurant in any area that serves belgian food

Target 1: I'm sorry but there is no restaurant serving belgian food.

Input 2: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please

Target 2: I'm sorry but there is no restaurant serving belgian food.

Input 3: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please I'm sorry but there is no restaurant serving belgian food. there a restaurant that serves gastropub

Target 3: Could you please repeat that?

Input 4: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please I'm sorry but there is no restaurant serving belgian food. there a restaurant that serves gastropub Could you please repeat that? i'm looking for a restaurant that serves gastropub

Target 4: Ok, a restaurant in any part of town is that right?

Input 5: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please I'm sorry but there is no restaurant serving belgian food. there a restaurant that serves gastropub Could you please repeat that? i'm looking for a restaurant that serves gastropub Ok, a restaurant in any part of town is that right? yes

Target 5: There are # restaurants serving gastropub food in any part of town . What price range would you like?

Input 6: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please I'm sorry but there is no restaurant serving belgian food. there a restaurant that serves gastropub Could you please repeat that? i'm looking for a restaurant that serves gastropub Ok, a restaurant in any part of town is that right? yes There are # restaurants serving gastropub food in any part of town . What price range would you like? i don't care

Target 6: the cow pizza kitchen and bar serves gastropub food

Input 7: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please I'm sorry but there is no restaurant serving belgian food. there a restaurant that serves gastropub Could you please repeat that? i'm looking for a restaurant that serves gastropub Ok, a restaurant in any part of town is that right? yes There are # restaurants serving gastropub food in any part of town . What price range would you like? i don't care the cow pizza kitchen and bar serves gastropub food

Target 7: Sure , the cow pizza kitchen and bar is on Corn Exchange Street.

Input 8: i'm looking for a restaurant in any area that serves belgian food I'm sorry but there is no restaurant serving belgian food. how about gastro of town please I'm sorry but there is no restaurant serving belgian food. there a restaurant that serves gastropub Could you please repeat that? i'm looking for a restaurant that serves gastropub Ok, a restaurant in any part of town is that right? yes There are # restaurants serving gastropub food in any part of town . What price range would you like? i don't care the cow pizza kitchen and bar serves gastropub food Sure , the cow pizza kitchen and bar is on Corn Exchange Street. and the phone number

Target 8: The phone number of the cow pizza kitchen and bar is 01223 308871.

A.2 M2M-sim-M Dataset Sample

Original Dialogue:

User: hi, buy 3 movie tickets for tomorrow.
System: what movie do you want to see and what theater do you want to go to?
User: i would like to see ae dil hai mushkil at the cinelux plaza theatre.
System: at what time?
User: 8:00 pm.
System: did you say 8:00 pm.
User: yes.
System: your purchase of 3 tickets for the 8:00 pm showing of ae dil hai mushkil on march 08 th at the cinelux plaza theatre is confirmed.
User: bye.

Processed Dialogue:

Input 1: hi, buy 3 movie tickets for tomorrow.
Target 1: what movie do you want to see and what theater do you want to go to?

Input 2: hi, buy 3 movie tickets for tomorrow. what movie do you want to see and what theater do you want to go to? i would like to see ae dil hai mushkil at the cinelux plaza theatre.
Target 2: at what time?

Input 3: hi, buy 3 movie tickets for tomorrow. what movie do you want to see and what theater do you want to go to? i would like to see ae dil hai mushkil at the cinelux plaza theatre. at what time? 8:00 pm.

Target 3: did you say 8:00 pm.

Input 4: hi, buy 3 movie tickets for tomorrow. what movie do you want to see and what theater do you want to go to? i would like to see ae dil hai mushkil at the cinelux plaza theatre. at what time? 8:00 pm. did you say 8:00 pm. yes.

Target 4: your purchase of 3 tickets for the 8:00 pm showing of ae dil hai mushkil on march 08 th at the cinelux plaza theatre is confirmed.

A.3 M2M-sim-R Dataset Sample

Original Dialogue:

User: hi, i want to make a restaurant reservation.

System: okay, where do you want to go, and how many people will there be?

User: the sushi boat for 6.

System: for what date?

User: this friday.

System: at what time?

User: 7 pm.

System: so you want a reservation for the sushi boat for this friday at 7 pm for 6 people?.

User: that's right

System: alright. your reservation is confirmed for 6 people at the sushi boat friday at 7 pm

User: thanks!

Processed Dialogue:

Input 1: hi, i want to make a restaurant reservation.

Target 1: okay, where do you want to go, and how many people will there be?

Input 2: hi, i want to make a restaurant reservation. okay, where do you want to go, and how many people will there be? the sushi boat for 6.

Target 2: for what date?

Input 3: hi, i want to make a restaurant reservation. okay, where do you want to go, and how many people will there be? the sushi boat for 6. for what date? this friday.

Target 3: at what time?

Input 4: hi, i want to make a restaurant reservation. okay, where do you want to go, and how many people will there be? the sushi boat for 6. for what date? this friday. at what time? 7 pm.

Target 4: so you want a reservation for the sushi boat for this friday at 7 pm for 6 people?.

Input 5: hi, i want to make a restaurant reservation. okay, where do you want to go, and how many people will there be? the sushi boat for 6. for what date? this friday. at what time? 7 pm. so you want a reservation for the sushi boat for this friday at 7 pm for 6 people?. that's right

Target 5: alright. your reservation is confirmed for 6 people at the sushi boat friday at 7 pm

Appendix B

Rule-based Chatbot

B.1 Rules Used in Rule-based Chatbot

The rule-based chatbot used in Section 5.4.4 has the following rules:

1. Ask the user about the movie domain entities one by one. The entities are date, time, movie, number of people, and the name of theater.

2. Once all the entities are filled with valid values, send a confirmation to the user with the registered values and wait for their confirmation.

3. Once the confirmation is done, send a notification of success to the user.

Note that the assumption is that all the values which user says for the entities are available in the database and if there is a misspelling, the chatbot will handle it without notifying the user.