

Santa Clara University

Scholar Commons

Computer Science and Engineering Master's
Theses

Engineering Master's Theses

6-12-2020

Deep Reinforcement Learning for Dialogue Systems with Dynamic User Goals

Glen Chandler

Follow this and additional works at: https://scholarcommons.scu.edu/cseng_mstr



Part of the [Computer Engineering Commons](#)

Santa Clara University

Department of Computer Science & Engineering

Date: June 12, 2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

Glen Chandler

ENTITLED

**Deep Reinforcement Learning for Dialogue Systems with
Dynamic User Goals**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF


MASTERS OF SCIENCE IN COMPUTER SCIENCE & ENGINEERING



Thesis Reviewer
Dr. Ying Liu



Thesis Advisor
Dr. Yi Fang

DocuSigned by:

96CE94A1A83A48A...

Chair of Department of Computer Science & Engineering
Dr. Nam Ling

Acknowledgments

First, I would like to thank Professor Fang for his generosity and encouragement in providing opportunities that fostered my ability to conduct research. He was very welcoming and I have greatly enjoyed working with him throughout my thesis. I would like to thank Professor Liu for being my thesis reader, as well as my advisor during my prior senior design project.

I would also like to thank my close friends who have always shared and listened, and helped me become who I am today. Finally, I would like to thank my parents for always reminding me they are proud as I pursued my bachelor's and master's degrees, it's easy to say I couldn't have done any of it without them.

ABSTRACT

Dialogue systems have recently become a widely used system across the world. Some of the functionality offered includes application user interfacing, social conversation, data interaction, and task completion. Most recently, dialogue systems have been developed to autonomously and intelligently interact with users to complete complex tasks in diverse operational spaces. This kind of dialogue system can interact with users to complete tasks such as making a phone call, ordering items online, searching the internet for a question, and more. These systems are typically created by training a machine learning model with example conversational data. One of the existing problems with training these systems is that they require large amounts of realistic user data, which can be challenging to collect and label in large quantities. Our research focuses on modifications to user simulators that "change their mind" mid-episode with the goal of training more robust dialogue agents. We do this by taking an existing dialogue system, modifying its user simulator, and observing quantitative and qualitative effects against a set of goals. With these results we demonstrate benefits, drawbacks, and tangential effects of using various rules and algorithms while recreating goal changing behavior.

Table of Contents

1	Introduction	1
1.1	Dialogue System Introduction	1
1.2	Overview and Common Issues	2
1.3	User Simulators	3
1.4	Paper Overview	5
2	Related Work	6
2.1	Early Dialogue Systems	7
2.2	Natural Language Understanding	8
2.3	Deep Reinforcement Learning	11
2.4	Machine Learning and Dialogue Systems	12
2.5	User Simulators	15
3	Approach	20
3.1	Natural Language Understanding	20
3.1.1	Probabilistic Models	21
3.1.2	Introduction to Recurrent Neural Networks	23
3.1.3	Types of Recurrent Neural Networks	24
3.1.4	Variations of Recurrent Neural Network	26
3.2	Deep Reinforcement Learning	27
3.2.1	Introduction to Reinforcement Learning	28
3.2.2	Q-Value Iteration	30
3.2.3	Q-Learning	32
3.2.4	Deep Q-Learning	33
3.3	End-to-end Goal Oriented Dialogue System	35

3.3.1	Overview	35
3.3.2	Semantic Frames	37
3.3.3	End-to-end Training	37
3.3.4	Dialogue State Tracker	39
3.3.5	Agent	40
3.3.6	User	41
3.4	Goal Shifting Theory	45
3.4.1	When to Goal Shift	45
3.4.2	Respond to Request	47
3.4.3	Respond to Inform	50
3.4.4	Implementation	50
4	Experimentation	52
4.1	Experiment Design	53
4.1.1	Goal Changing Implementation	53
4.1.2	Controlling Goal Change Events	54
4.1.3	Quantitative Metrics	55
4.2	Experiments	56
4.2.1	Control System	56
4.2.2	Experiments with Goal Change by Inform	60
4.2.3	Experiments with Goal Change by Request	64
4.2.4	Experiments with Mixed Model Goal Change	66
4.2.5	Testing Against Control Environment	68
5	Conclusion and Future Work	70
	Bibliography	74

List of Figures

3.1	Semantic frame depicting user intent, inform restriction slots, and request slots to be filled in later	36
3.2	Framework architecture depicted with dialogue flow and relevant modules	38
3.3	Decision tree diagram for user simulator response to agent request	47
3.4	Decision tree diagram for user simulator response to agent inform	49
4.1	Blue: Episode success rate within epoch, Orange: Average turn count in Epoch relative to maximum	57
4.2	Blue: Average number of turns relative to maximum, Orange: Total event count relative to maximum	58
4.3	Sum breakdown of various potential goal switch dialogues in control framework	59
4.4	Success rate of various start epoch times with 1% probability of inform goal change	61
4.5	Average turns and event 0 count relative to their respective maximums. Left: Goal change at epoch 10, Right: No Goal-change	62
4.6	Success rate of various start epoch times with 1% probability of request goal change	65
4.7	Left: Event0: 1%, Event1: 10%, Right: Event0: 2%, Event1: 10%	67
4.8	Success rate of three models on a 90% goal-change test environment . . .	68

List of Tables

4.1	Success rate of goal change by inform	60
4.2	Success rate of goal change by request	63
4.3	Success rate of goal change by inform / request	65

CHAPTER 1

Introduction

1.1 Dialogue System Introduction

Dialogue systems have recently become widely used in a variety of contexts across the world. While a great deal of dialogue system technology has been around since the 1960s, recent advancements in machine learning, cloud systems, mobile devices, and information retrieval have made this field nearly universal. Nearly all smartphones have some version of a voice assistant which can understand context and complete tasks from complex audio or text messages. Furthermore, standalone voice platforms such as Amazon's Alexa have allowed us to interact with device ecosystems as well as vast amounts of internet data to normalize a new era of human-computer interaction.

Another major factor that has led to the increased use of dialogue systems has been improved robustness. Many historical systems worked with brittle over-the-phone speech recognition to drive users through pre-defined system menus. Now, modern platforms are

able to perform complex semantic understanding and context awareness to handle a wide array of conversations. These improvements typically have come from advancements in natural language processing modules that learn from large databases to derive complex meaning from text. Furthermore, recent improvements in behavioral design have led to a greater ability for a dialogue agent to perform actions in an optimal way.

1.2 Overview and Common Issues

Our research focuses on facilitating the creation of more expansive systems that stray further from the early menu-based systems, and closer to natural human-to-human interaction. Ideally, dialogue systems should be able to understand any phrase or sentence in any conversational context. From the standard viewpoint of hard-coded dialogue flows, this challenge seems almost impossible. This is due to our potential conversational space which grows extremely quickly as we enable complex speech understanding and fully dynamic conversational structures.

This issue has led to the widespread adoption of model based conversational agents which seek to more deeply understand these continuous problem spaces. These models accomplish this by learning a connection between speech signals, context, and given

rewards. More specifically, reinforcement learning is often applied to teach a system to learn far-sighted, optimal behavior through trial and error. Advancements in deep neural networks have additionally allowed these types of systems to learn in spaces where states and actions are not statically defined.

However, as these mathematical models expand to more complicated domains, the demand for diverse, labelled data begins to expand as well. This data is needed to train models to interact with varying speech contexts and errors as well as dynamic conversational scenarios. Additionally, this data is very hard to collect and label, so many training platforms have introduced user simulators to automatically interact with models and generate training points to aid in this data scarcity problem.

1.3 User Simulators

User simulators can be an excellent approach when faced with a lack of data. This has the added benefit of state space exploration, which leads to more robust models with deeper understanding of tasks and possible user dialogues. However, the very first problem with user simulators is that they are an imperfect representation of actual human interaction. It is impossible to train a conversational model to perfectly handle

human inputs, especially if it was never actually trained with human data. So when we develop user simulators, our focus shifts towards creating a simulator that replicates the essential human behavior needed to effectively train our model.

This is an active research area that, like dialogue agents, is split into rule-based systems and model-based systems. Both systems attempt to generate fake user data by understanding the ways that users behave when interacting with dialogue systems. This can be done by hand-crafting rules or automatically learning them from data. We will be focusing on rule-based systems.

The specific user simulator behavior that we will be exploring is dynamic goal changing. User simulators typically operate by randomly selecting a specific goal every episode, which the model agent then tries to discover. User actions then attempt to discover information that satisfies its goal, while agent actions attempt to learn more about the user goal, in order to provide satisfactory results. Programming user simulators to occasionally "change their mind" about their goal can have large effects on the training of a deep reinforcement learning dialogue agent. There are many ways to simulate this behavior, so we will be exploring the effects of these various methods under different scenarios.

1.4 Paper Overview

The remainder of this document will provide background, implementation details, experimentation, and results. First, we will extensively discuss the history of dialogue systems, reinforcement learning, and the connection between the two. Then, we will explain our research approach which includes implementation details and theory of the major components of our platform. Next, we will go over the design of our experiments and their results. Finally, we will finish with our conclusion and suggestions of future work.

CHAPTER 2

Related Work

Ever since programming and automation became a part of usable products, it has been used as an attempt to make lives easier. Computers provide the ability to vastly outperform humans in the areas of speed, memory, and consistency. Today we have built so many layers on top of these general improvements that we are able to provide deep understanding and optimization to problem spaces by means of conversation. In this chapter, we will provide a review and history of dialogue system research that has led to a wide range of dialogue programs from simplistic rule based systems to complex and data-trained neural models.

The term "dialogue system" is a broad research field that encompasses multiple areas. For some, conversational agents are the focus, and for others it is a tangential application. Since we are focusing on a system that yields a self-training cycle, several of these fields will be at least somewhat relevant. Some of these areas include natural language understanding and generation, deep learning, reinforcement learning, expert systems,

and information retrieval. The primary backdrop for our research is model robustness and error for deep reinforcement learning algorithms, so this will be the focus of the rest of this chapter.

2.1 Early Dialogue Systems

One of the first dialogue systems was developed at MIT in 1966. This program was titled ELIZA and claimed to make human-computer interaction possible, demonstrating one of the first "chat bots". This research focused on phrase building through keyword extraction, context understanding, and heuristic dialogue flows [1]. Another one of the first examples of early dialogue systems was developed by the DARPA Project in 1977 [2]. This program was primarily meant to provide spoken word interfacing through telephone systems to interact with basic menus and information systems. The speech system focused on recognizing parts of audio inputs and converting partially completed sentences into meaningful interpretations. This research was also significant as it included an early dialogue manager to handle dialogue processing and network interfacing in parallel to speech recognition.

Around 1999, AT&T Labs produced meaningful research in the field of conversation

tracking and decision making. Specifically, they integrated data-driven reinforcement learning into dialogue systems at a time where generic Markov decision processes were facing challenges in state representation and rewards in data. They were able to produce policies and discover optimal behavior given a corpus of dialogue information [3]. Another significant dialogue system created during this time was ALICE, developed in 1995 to produce realistic dialogues, which also attempted to pass the Turing Test. ALICE made use of a complicated heuristic systems in addition to pattern mining, and used large conversational trees to encode contextual understanding for a given conversation. These early systems were marked by differing views of human vs machine conversation initiative, scalability issues in conversation trees, error-prone speech understanding, as well as processing and memory challenges [4].

2.2 Natural Language Understanding

Recurrent Neural Networks (RNNs) are one of the most important technologies developed in the scope of dialogue systems. While this technology has primarily been used for Natural Language Understanding (NLU), it has also been used in a variety of other applications, including some user simulators. RNNs operate on sequential data and

learn the relationships between items in a stream of information. This can be incredibly useful for tagging words in a sentence, or even learning behavior from sequences of dialogue actions. RNNs have the ability to encode sequential information into hidden states which represent contextual information. These states can then be applied to RNN weights, whose outputs correspond to various subsequences of information. This method of training can make an RNN an incredibly useful tool for dealing with dialogue systems in many contexts.

Some of the first Recurrent Neural Networks were developed at University of California, San Diego and Carnegie Mellon University in 1993 alongside general research for back propagation in neural networks. This team explored different gradient methods, gradient descent optimization, and applied these methods to neural networks containing explicit or implicit hidden layers. This is relevant to the difference today between typical neural networks that use predefined structures, and recurrent networks that rely on self propagation of hidden states. This was one of the first papers demonstrating the use of hidden node information in a neural network and how it could be optimized using gradients that were applied to a large set of data types [\[5\]](#).

In 2010, Tomas Mikolov and a team from the Brno University of Technology as well

as Johns Hopkins University applied Recurrent Neural Networks to language models to receive a large improvement in accuracy as well as reduction in error rate. They specifically used RNNs for sequential information prediction which is now especially relevant to language models and conversational data. This model is built upon previous research which used typical neural networks to understand dialogue using fixed length contexts [6]. However, with RNNs, Mikolov et al. were able to more realistically represent context lengths, and train models that ultimately improved upon typical n-gram models [7].

In 2016, Maluuba Research labs expanded on the use of sequence-to-sequence RNNs in order to improve user simulators' flexibility in general context understanding. This work introduced a data-driven model which used the encoder-decoder network to take as input dialogue contexts and output a sequence of dialogue acts in relation to the user goal. This method also allowed the current goal fulfillment to be internally represented, allowing the system to generate sequences of user responses in relation to conversational progress rather than just an end state. This research is a modern example of RNNs being applied to sequential dialogue information rather than just NLU [8].

2.3 Deep Reinforcement Learning

Reinforcement learning is one of the oldest and most relevant research areas in our work. In this section we will be discussing deep reinforcement learning research to better understand how it has been applied to dialogue systems, and how it is one of the most prevalent approaches to goal oriented dialogue systems. This section will start with early, non-specific, examples of deep reinforcement learning, and cover common issues with deep reinforcement learning. This will provide an introduction into section 3.2 where we discuss the eventual use of deep reinforcement learning in dialogue systems.

In 2013, Mnih et al. proposed a convolutional neural network model which was able to discover Q-learning policies which mapped raw input to actions in a non-modified environment. This was significant as it demonstrated the ability to train a reinforcement learning policy with a high dimensional input rather than the typical lower-dimensional, hand crafted signals. This application of deep neural networks to reinforcement learning also demonstrated the ability of such a system to learn behavior through trial and error, partially relieving the need for labelled data [9]. These two features of Mnih et al.'s research provides the basic ability to train an optimal task-oriented dialog agent with continuous, unlabeled datasets, such as the outputs generated by our user simulator.

Google DeepMind introduced double Q-learning to a Deep Q Network (DQN) in 2016 in order to provide a solution to overestimation of action values. Double Q-learning in this context includes a secondary Q function that exclusively is used for value estimation, while the other Q function is used for action selection. Then, one Q value can be used in the other's update step to balance iteration and fight over-estimation. This paper is an example of constant improvements being made to DQNs in order to address several issues involved with imperfect training [10]. A similar multi-network implementation was introduced by DeepMind in 2016 which drastically improved performance and reduced training time for continuous state reward problems. Since various training models are executed on different versions of the environment, they are able to modify the same Q-learning policy one step at a time with more bandwidth from various algorithms [11].

2.4 Machine Learning and Dialogue Systems

As mentioned in section 2.1, dialogue systems were first implemented with rule-based algorithms that provided value through heuristics. Eventually, statistical models and reinforcement learning were applied to these systems to learn optimal policies through modelling rather than expert rules. As deep reinforcement learning became more

prevalent, dialogue systems became an obvious application as it worked with difficult to define features, rewards, and actions. In this section we will go over the joint history of machine learning and dialogue systems, and how they have become deeply intertwined.

In the year 2000, AT&T laboratories created a system that applied Q-learning and a new evaluation framework to learn performance functions based on conversation experience and dialogue history. This system was demonstrated with a spoken dialogue system that accessed email through speech. This early research demonstrated a dialogue system's ability to learn an optimal policy for agent initiative, and other conversational actions to satisfy a goal [12].

One of the most important areas of research for dialogue systems was the idea of training dialogue policies through fixed datasets. In 2005, the University of Edinburgh developed a hybrid supervised learning and reinforcement learning approach. They built their system with large representations of states and policies and introduced reinforcement learning since their original policy too narrowly explored their large state space. With this combined approach, the team was able to use reinforcement learning to optimize dialogue reward, while supervised learning was used to apply the learned policy to the data with relevant scope. This resulted in an optimized system that could learn policy

efficiently through experience within a labelled data space [13].

In 2016, a research team from University of Montreal explored data-based generative models using Recurrent Neural Networks. This research primarily focused on the goal of creating realistic dialogues with improved performance over the typically used n-gram models. Movie script data was primarily used to train a hierarchical encoder-decoder RNN. Bootstrapping with word embeddings was also explored, as well as bi-directional RNNs to mitigate training bias towards the edges of utterances. Hidden states were summarized by taking representations of both directional chains and feeding them into the context RNN, which kept track of past utterances to summarize a dialogue. This research ultimately demonstrated the many diverse uses of RNNs in the context of generating and understanding speech details and context, and laid groundwork for future research that explores using these types of models to more deeply understand, generate, and tag words that can be used to train more robust and complex dialogue agents [14].

The next paper is the primary source that we will be using for our testbed. The research put out by the Microsoft research team in 2018 starts with the goal of end-to-end training for the purpose of task oriented dialogue systems. This attempts to address the issue where individually trained models propagate error to other modules in the secondary

training or testing cycles. Additionally, this research introduces modules into the training cycle such as: dialogue managers which interact with databases, error controllers which introduce specific kinds of errors, user simulators, and finally the DRL agent. The framework proposes features which address common problems with other DRL training systems such as inflexible question types, and simplistic user goals. The experimentation also explores learning curves based on frame-level and natural-language-level errors, as well as general accuracy and effectiveness based on the entire end-to-end training pipeline. This research introduces a framework for controlling and co-training various modules that ultimately affect the accuracy of the DRL agent based on cross-module interaction, module error and interaction data robustness [15].

2.5 User Simulators

User simulators have been seldom researched relative to the rest of dialogue systems as they are often a secondary piece of larger training pipeline. User simulators also typically fall into the category of agenda-based heuristic simulators or data-driven model simulators. Both of these approaches have advantages and disadvantages, which we will try to talk about in this section. User simulators offer a potentially infinite amount

of training data which is especially useful for the kinds of DRL models that we are experimenting on. However, these user simulators often struggle to match the realistic nature and robustness of real user dialogue datasets, so we will discuss research that explores solutions as well as alternative methods.

In 2005 a team from University of Edinburgh researched what they called "learning user simulations" for information state update dialogue systems. These information state update systems represent the dialogue process as states with each turn of dialogue. In this way, the user simulator created here would be represented in the information state, and its effectiveness could be reflected during evaluation. The specific user simulators explored were stochastic processes, generated from the current conversational state, as well as n-gram generation methods. With these methods, effectiveness was measured with relation to the number of filled slots, confirmed slots, and number of actions performed by the system. This showed an early hybrid between data and rule based approaches, using supervised learning processes with a dialogue corpus [16].

In 2006 a team from University of Edinburgh researched advanced n-gram models for user simulators. The advanced n-gram models sought to build upon previous n-gram models in order to provide functionality for goal-based user simulators. Typically n-gram

user simulators make actions based on given speech and current tasks. With advanced n-grams, actions are made by the user based on the current progress of the goal. This is very relevant to our research, as it is one of the earlier user simulators that make goal-oriented decisions, and attempts to evaluate the performance accordingly [17].

Jost Schatzmann et al. researched quantitative evaluation of general user simulators in 2005, as a potential solution to the inability to evaluate stochastic models. The team introduces a method to evaluate domain-independent dialogues using the DARPA communicator corpus, showing that they were able to discern between user simulators and actual human users. The research measured probabilistic models as well as deterministic models. Multiple methods of evaluation were used including precision and recall to measure which models produced the most relevant results with respect to different agent models. These models include the Bigram, Levin and Pietquin models. Ultimately, the evaluation looked at types of actions and used these statistical distributions to differentiate human and machine behavior. This simple approach can be a powerful method of user simulator effectiveness as a user simulator non-differentiable from a real human can provide more sensible and robust responses, which may train a more effective agent [18].

In 2017, Xiujun LI et al. introduced a user simulator framework to be applied to goal-oriented DRL dialogue systems, as well as online training. This user simulator is built to interact within the movie-booking space and is connected to a user goal list as well as a movie database. The agent attempts to find a movie that fits user restrictions such as time, location, etc., from which the user will respond by either informing this question or asking its own question. The entire training system [15] benefits from a warmup process which allows the agent to be pre-trained based on random actions before applying its own policy. This type of user simulator is then able to encode important behavior that can pre-prompt a DRL dialogue agent to be further trained with real user input. This type of user simulator makes a case for rule-based systems as starting points over data-based models as they are able to define important behavior that may be absent, mislabelled, or under-represented in real data [19].

A team of industry and academic researchers published a paper in 2018 describing end-to-end training of a model-based user simulator in task oriented settings. In this paper, Gur et al. designed a hierarchical sequence-to-sequence RNN to generate user dialogue from a starting user goal, as well as represent dialogue history and current conversational state. This model avoids the need for a state tracker or error model controller by introducing

model regularization techniques to balance the trained user simulator. The model is then evaluated on the movie booking space and trained against various model-based agents. The user model is first trained using a dataset of potential user goals, which is fed to an agenda based model that generates a full example dialogue at the start of each episode. This research demonstrates an effective use of a sequence-to-sequence RNN which is trained in a self-sustaining way. This type of model introduces realistic robustness while implementing novel regularization techniques to keep the model from over-optimizing [20].

CHAPTER 3

Approach

In this section, we will be describing implementation details relevant to the general research field of task-oriented dialogue systems. This will build a foundation for discussing the specifics of a training pipeline such as the one used for our research. After forming an understanding of this current dialogue system, we will discuss some of the implications and challenges with module errors and training side effects. We will conclude this section by proposing our theory for a potential goal shifting user simulator and the predicted effects.

3.1 Natural Language Understanding

We believe it is important to discuss Natural Language Understanding first as it is one of the first things that a user sees (or hears) when interacting with a dialogue system. NLU has gone through its own research arc, and is fairly decoupled from the specific dialog agent research that we are working on. However, NLU research is very important

to dialog agent research, not only because its accuracy directly informs actions, but because the most recently researched Recurrent Neural Network (RNN) models provide insight into developing action-to-state and state-to-action systems that can be used in complicated neural networks and algorithms. In this section, we will discuss this underlying technology, starting from the foundation of NLU, and relate it to hypothetical future systems that may benefit from our research.

3.1.1 Probabilistic Models

When discussing Natural Language Understanding, a good place to start is probabilistic dialogue models. These models are used to predict the probability of a word occurring given the sequence of words that have occurred before it. This can be used for auto-completion in sentences, as well measuring the probability of observing an entire sequence.

Typically formula is expressed as:

$$\prod_{t=1}^T p(x^{(t)} | x^{(t-1)}, \dots, x^{(1)}) \quad (3.1)$$

where x_t is the word at position t in a sequence. Through the chain rule, this formula

expands out to:

$$p(x^{(T)} \dots x^{(1)}) = p(x^{(T)} | x^{(T-1)}, \dots, x^{(1)}) * \dots * p(x^{(2)} | x^{(1)}) * p(x^{(1)}) \quad (3.2)$$

where each probability represents a conditional probability of smaller and smaller subsequences. This formula can then be computed with fairly straightforward probability estimations. A common way to estimate these conditional probabilities is to simply count word co-occurrence given a corpus.

Once you have a way of computing the probability of a word given a sequence, you can begin to build up a more complete formula that is used to predict labels of words given a sequence of words:

$$p(y' | x') = \left(\prod_i^n p(s_i | w_1, \dots, w_i) \right) * p(i_m | y') \quad (3.3)$$

In this formula, w_i is i th word in a given utterance x' . The semantic frame being predicted, y' contains slots i through n represented as s_i . Now instead of predicting the next word given a sequence of words, we predict the slot type at word i given the sequence from word x_1 to x_i . Computing all slot probabilities as in formula 3.1, then multiplying

this by the conditional probability $p(i_m|y')$ which computes the intent type given our predicted semantic frame. Combining all probability estimations to create formula 3.3 will allow us to label all slots in a sentence as well as predict the conversational intent of the sentence itself.

3.1.2 Introduction to Recurrent Neural Networks

Recurrent Neural Networks (RNN)s allow us to leverage the essential math used in neural networks to learn and represent continuous information with states, as well as digest and produce intermediate information given these states. However, when dealing with sequences of information (which is especially relevant to dialogue systems) we often have trouble training with and representing information over a continuous set of inputs or outputs. RNNs seek to use recurrence within neural network nodes to represent history, and create an applied chain of information that may represent some sequence of information.

This self referencing idea allows us to "unwrap" a history of inputs from a single node into a metaphorical chain of nodes where each sequential state is fed into the next. This way, when a node's previous hidden state is fed back into itself during the next sequential

input, there is some hidden representation of the input sequence which persists each time a new input is entered. This is very relevant to formula 2 since each previous input creates an output which is then fed into the next sequential probability computation. This allows us to create a conditional probability chain very similar to what we see in formula 2. Since each sequential state is represented by a node's input, hidden state, and output, we can also apply a version of back-propagation to automatically learn optimal weights and compute conditional probabilities.

RNNs use a process called back-propagation through time (BPTT), which is similar to typical back propagation where an end node's error is propagated backwards through the network and applied to gradients along the way. The major difference in this case is that "backwards" is relative to our sequence (which often implies time).

3.1.3 Types of Recurrent Neural Networks

One of the major benefits of RNNs is the flexibility of using inputs and outputs. When we "unroll" a node we can essentially match each sequential input to its respective sequential output. These outputs are functions of that given input, as well as the hidden state which represents the history of all the inputs before it. So given that we potentially

have a sequence of inputs and a respective sequence of outputs, we can be selective about which to use when our RNN approximates a function.

Seq-to-seq RNNs are networks where the input is a sequence and the output is also a sequence. An example of a seq-to-seq application would be a system that takes weather data as inputs and outputs temperature as an output. The inputs are contextually based on sequence, and each output should correspond to a given day. If we modified our model slightly to predict only a single day's temperature given a month's worth of weather data, this would be called a seq-to-vector model. This is because we are predicting only a single value given a sequence of information. In this case, our output's hidden state and corresponding output would embed the entire sequence history of weights, biases, and inputs. Depending on which inputs and outputs we find relevant, we can also implement vector-to-seq and vector-to-vector RNNs. The last type of RNN we should mention is the encoder-decoder network. This network is composed of nodes trained on separate datasets, where the final hidden state of one node embeds latent information which is then transferred into the other node. The most famous example of this is language translation, where a sequence of words is processed into a vector (seq-to-vector), then passed as input into a separate node which generates another sequence of words from

the hidden state (vector-seq).

3.1.4 Variations of Recurrent Neural Network

One of the first problems with RNNs is that the propagation of information through hidden states is inherently lossy. This can be especially problematic if we are representing information in a seq-to-vector manner. If we are using the final result as the output of our network, this may be biased towards information that appears later in the sequence. A solution proposed for this problem is called Long Short-Term Memory (LSTM) [21]. LSTM layers are composed of LSTM cells which include a short-term state and a long-term state. The inner-workings of an LSTM cell include a forget gate, which drops memories to make room for new ones, an input gate which selects new memories to be added, and an output gate. The input gate selects memories from both the current input and previous short-term state, adds it to the result of the forget gate, which then passes its result to the output gate. The output gate takes the results from the forget gate, input gate, as well as the previous short-term state, and produces the next short-term state. This new short term state is also the output of the cell at that time, and is fed into the next LSTM along with the new long-term state. All of this functionality ultimately

gives RNNs the ability to learn which latent information to store long term, forget, or pass onto future node iterations. This ability drastically improves the ability for RNNs to recognize patterns in complex continuous information such as audio, video, and text.

3.2 Deep Reinforcement Learning

While this research is based on the user simulator module of this framework, we are really trying to improve the dialogue agent at its core. The dialogue agent that is used in our framework is a Deep Reinforcement Learning (DRL) algorithm which utilizes deep learning networks to optimize the goal function that is used for reinforcement learning. As our agent tries new behavior, it will learn its own policy based on a reward function, then learn to interact with state data which can be flexibly defined.

By the end of the training process, our agent will be able to take complicated state representations as input into a policy function, produce meaningful actions, and retrain itself based on the received rewards. Our research goal is to improve our dialogue agent's ability to handle situations where a user changes its goal. We do this by making small changes to the user simulator and allowing the agent to learn from these newly introduced experiences. However, in order to best do this, we must first understand the DRL model

that we are seeking to improve.

3.2.1 Introduction to Reinforcement Learning

At its core, reinforcement learning is based on states, actions, and rewards. If we view our environment as a Markov decision process (MDP) where we map all state, action pairs to a reward, and consider the probability that an agent will take any given action, we can begin to model behavior. This allows us to develop a policy where we seek to maximize our long term reward by optimizing some distribution of actions upon arriving at a given state.

If we introduce a value function into our framework which models the average long-term reward of being at a given state, we can optimize our model through traditional dynamic programming. This essentially means processing the long term value of all states through recursive evaluations, then taking actions based on these evaluations. This value iteration formula is defined as:

$$V_{i+1}(s) = \max_a \left[\sum_{s'} P_a(s, s') (R_a(s, s') + \gamma V_i(s')) \right] \quad (3.4)$$

In this formula, we use the probability $P_a(s, s')$ to give a probability weight to any

action a followed from state s . Given this probability and the reward observed if such an action is taken, we can compute the immediate weighted reward of taking an action from a given state s . Since we are optimizing for the long term reward, we add the value function of the resultant state s' to our immediate reward. In this formula, γ represents the learning rate, which affects how fast our model will converge.

The sum of all of the weighted rewards will give us an estimation for the "value" of a given state. If we continue to compute value functions and use them recursively, we can understand a value space by continuing this process until the difference between a previous value and the next value converges to a small enough error in order complete training.

Once we have a way of representing the given state values, state rewards, and action probability distributions, we can start to learn an optimal policy. The policy equation $\pi(s) = a$ simply returns an action to take given a current state. If replace the action a with policy $\pi(s)$ in equation 3.4, we can now evaluate a value based on a specific policy rather than the computed maximum valued action. Next, we can fix a given value function, then introduce the concept of policy iteration where we recursively re-compute which action to take at a given state using the value function and previous policy. These

concepts give us the following equations:

$$V_{i+1}(s) = \sum_{s'} P_{\pi_i(s)}(s, s')(R_{\pi_i(s)}(s, s') + \gamma V_i(s')) \quad (3.5)$$

$$\pi_{i+1}(s) = \max_a \left[\sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V_{i+1}(s')) \right] \quad (3.6)$$

Similarly to value iteration, we can continue this recursive process until convergence.

However, in the policy iteration case, convergence occurs when $\pi_{i+1}(s) == \pi_i(s)$ for all

s .

3.2.2 Q-Value Iteration

Now that we've developed our environment in a way that allows us to learn the long-term value and optimal action policies given a state, we can begin to explore model-free learning. Model-free learning now assumes that we no longer have a predefined model of transition probabilities or rewards. This is useful for modeling an agent that learns through trial and error, which is what reinforcement learning is based off of.

To learn in this less explicitly defined environment, we can use Q-value iteration which

seeks to estimate the value of a state-action pair to aid in situations where we do not know the immediate reward of taking a given action at a state s . This modified vision of our environment forces us to learn based on the distant potential actions that may occur after taking an initial action rather than learning based on immediate actions before we've taken them. We can model this new situation with the following equation:

$$Q_{i+1}(s, a) = \sum_{s'} P(s, a, s') (R(s, a, s') + \gamma * \max_{a'} Q_i(s', a')) \quad (3.7)$$

The main difference between this equation and equation 3.4 is that now we now consider the maximum secondary action a' after we arrive at the next state s' which we were brought to from taking the given initial action a . Similar to typical value iteration, Q-value iteration is recursive as it factors in many $Q_i(s', a')$ values when computing $Q_{i+1}(s, a)$. Once our Q-value estimation converges, learning a policy is even more straightforward than before. All we need to do is find a maximum $Q(s, a)$ value given a state s over all possible actions a . In conclusion, our modified value function finds an optimal policy using only a given state and potential actions, without knowing the subsequent rewards or state transition probabilities of taking said action. With this, we can create a model that learns through trial and error, without any pre-existing

understanding of the space around it.

3.2.3 Q-Learning

Q-value iteration is a useful model of thinking and can bring us very close to creating an agent that can autonomously learn an action policy in a loosely defined environment. However, Q-value iteration operates on the basis of expected values where state transition probabilities serve as weights for observed rewards. Often it is the case where these probabilities are hard to estimate. When this happens, temporal difference (TD) learning can be used. TD learning essentially updates the estimate of the value function by adding the difference between an estimation and the previous value, then multiplying this error by a learning rate.

$$Q_{i+1}(s, a) = (1 - \alpha)Q_i(s, a) + \alpha(r + \gamma * \max_{a'} Q_i(s', a')) \quad (3.8)$$

$$Q(s, a) \leftarrow_{\alpha} r + \gamma * \max_{a'} Q(s', a') \quad (3.9)$$

As we saw in equation 3.7, our Q-value iteration function essentially uses an expected value estimation with a previous version of the function to find the best next value estimate. In this sense, the estimated value is a summation of all possible values weighted by the probability of occurrence. Since we do not know the probability of an s' state occurring, we are only left with the individual reward value of that next state $r + \gamma * \max_{a'} Q_i(s', a')$. Although we can no longer use this estimation for expected value, we can compute the error between this estimation of our function, and the previously trained function, hence the error $r + \gamma * \max_{a'} Q_i(s', a') - Q_i(s, a)$. With this new error, we can employ a TD learning method in order to iteratively adjust our Q-value estimation over periods of training, which is represented more clearly in equation 3.9. Using this error, we are able to essentially execute gradient descent to learn the Q-value of an environment without knowing anything in advance besides a set of discrete state-action pairs.

3.2.4 Deep Q-Learning

While Q-learning can be very effective in autonomously estimating Q-values of an environment without any prior model, it is restrained to a discrete representation of

possible states and actions. This serves as a huge barrier when considering domains with near-continuous states and actions such as computer vision and natural language. Typically hand-crafted features provided to a DRL model to work with, but these representations often lose information and can be biased. Additionally, even if features can be hand-crafted, the size of the entire state space often explodes exponentially making it very hard for a DRL model to train with.

Neural networks can help us a great deal in this situation as they allow us to automatically learn weights that represent the relationship between inputs and outputs over time, both of which can be represented flexibly. Additionally, neural networks already use error gradients to perform back-propagation, which we are able to provide to the method introduced in equation 3.9. With all of these concepts, we are now able to combine Q-value theory, TD learning, and deep neural networks to teach an agent to automatically learn an optimal policy through trial and error with nothing but continuous variables representing states, actions, and rewards (often a continuous function).

Learning optimal behavior in a dialogue system setting has often been challenging. The seemingly infinite number of possible state-action representations can make feature engineering challenging for simple data-based models and stochastic rule-based mod-

els consequently more attractive. Once Deep Q-Networks (DQN) began to be used, reinforcement learning was able to be applied to images and text input.

3.3 End-to-end Goal Oriented Dialogue System

In this section, we will be describing the high level structure of our development platform, and how all modules of our system play a part in training our DRL agent. The dialogue system used in our research is based off of the end-to-end task-completion neural dialogue system developed by a Microsoft research team in 2017 [15]. At a glance, this framework includes four major modules: a dialogue manager (DM), error model controller (EMC), natural language understanding (NLU) unit, natural language generation (NLG) unit, and user simulator. The DM is composed of the dialogue state tracker (DST) and DRL agent.

3.3.1 Overview

This framework was specifically introduced with the goal of creating an end-to-end training pipeline which trains all modules simultaneously while training episodes run. This mainly has the purpose of avoiding downstream error propagation which typically

```
{  
  'intent': 'request',  
  'inform_slots': {  
    'city': 'seattle'  
  },  
  'request_slots': {  
    'theater': 'UNK',  
    'starttime': 'UNK'  
  }  
}
```

Fig. 3.1: Semantic frame depicting user intent, inform restriction slots, and request slots to be filled in later

occurs when modules are trained individually and used to train subsequent modules.

This tightly coupled framework also allows for modules to benefit in certain ways from training off of one another. This is partially thanks to tangential system behaviors such as user initiated dialogues, which allow dynamic feedback to the DRL agent to be conveyed in a near real-time manner during training. Additionally, modules such as the DST provide the ability to track conversation history as well as prepare states with internal and external information.

3.3.2 Semantic Frames

In Fig. 3.1 we can see an example semantic frame which represents a dialogue action with an intent, inform slots, and request slots. The intent represents whether this message is meant to request information, or inform information. Both the user and agent are able to request, inform, or do both. The inform slots represent what restrictive information the module is attempting to convey to the other module. The request slots represent the unknown information which the requesting module hopes to learn. This semantic frame format is what the natural language modules either generate text from, or convert text into. The user goal is composed of a set of request slots that the user hasn't given a value to yet, as well as known inform slots that the user wishes to tell the agent over the course of the conversation.

3.3.3 End-to-end Training

Fig. 3.2 illustrates the training process as well as the flow of information in our end-to-end framework. An episode is initialized by sampling a user goal from a database. This user goal is given to the rule-based user simulator which can then initiate a dialogue with the agent. In the full system, semantic frames containing information restraints

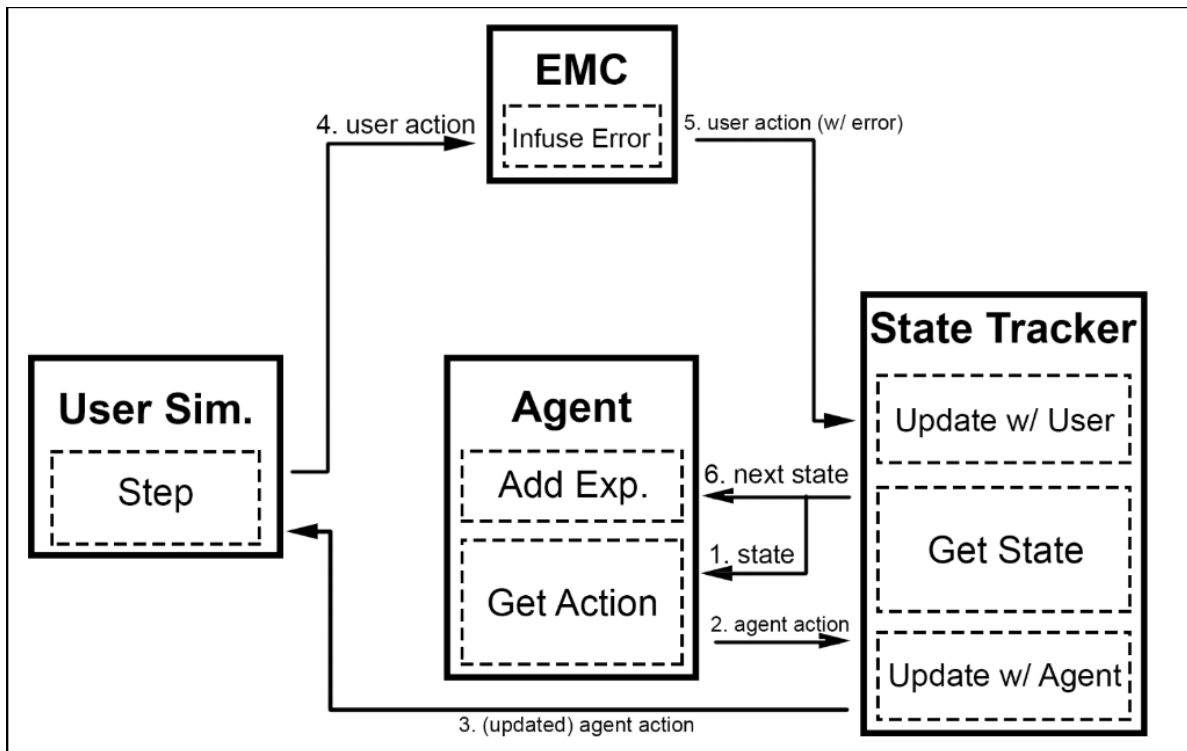


Fig. 3.2: Framework architecture depicted with dialogue flow and relevant modules

and requests are sent forward into the rest of this system. These semantic frames can be encoded by the natural language generator, then decoded by the natural language understanding module, which ideally represents human error in understanding intents and details in language. Once the sentence has been decoded back into a semantic frame, it is given to the dialogue manager which begins to unpack information which can be added to the DST's understanding of the conversation history. The dialogue manager then generates a matrix representation of the recent conversation history, round number, as well as some potential database matches. The DRL agent can now take this matrix representation, and generate a response agent action. This semi-completed agent action

is filled in with further information by the DM to be sent to the user simulator. The user simulator finally processes the incoming semantic frame, and generates its own response based on the user agenda, which finally completes the framework cycle.

Conversational error can be simulated by using intermediate natural language representations, or explicitly modifying either the semantic frame information using the EMC module. When the EMC module modifies semantic frame information, it can explicitly change the message intent or information slots. We can train the DQN as often as we like by taking the difference between a given Q-value target estimation and a prediction from the current DQN. This particular framework uses batch training by amassing data from a number of observed episodes and performing multiple training steps at once. Part of what makes this an end-to-end system is that the rewards that are used for generating our target Q-value are given by the user-simulator based on a simple metric that can be modified.

3.3.4 Dialogue State Tracker

The DST performs several functions that are necessary for understanding the context of a conversation, but unreasonable for the DRL agent to handle on its own. The first

function is to query the database for any potential movie matches that can be served to the user. The DST does this by continuously building an internal understanding of the dialogue and using restraints that it has noted over the course of the conversation to perform a query that satisfies these restraints. Next, the DST can reflect its own representation of the conversation based on the database query and the resulting user response. Finally, the DST can prepare state representations to the user. This is specifically done by taking the request and inform slots from the most recent user and agent actions, the current round number, as well as a list of the most recent DB matches. Once all of this information is collected, it is combined into a matrix and given as input into the DRL agent. Once the DRL produces an action frame, the DST can then fill in appropriate request and inform slot values based on its internal representation of the user goal and database query results.

3.3.5 Agent

As described in section 3.2, our goal-oriented dialogue agent is implemented with a DQN and is able to use near-continuous input and output spaces to learn an optimal Q-value representation of the given dialogue environment. With DRL, this agent is able to learn

an optimal policy by choosing the action from a given state that maximizes the Q-value as it is currently trained.

Much of the work in our own research is based on the surrounding framework that supports DRL and not the network itself. The network is actually a relatively compact DQN with only one hidden layer and a standard RELU activation function. This framework includes a large number of hyper-parameters such as batch training size, experience replay size, optional natural language modules, maximum episode length, and more. These parameters are all meant to be a part of the end-to-end training framework and are all directly or indirectly affecting the training effectiveness of the DRL model.

3.3.6 User

The user simulator is a relatively simple module with significant effects on the training of our agent. There is a great deal of room for research on user simulators, as there is no real standard for how to design a user simulator. Some of the major ways to implement a user simulator are through rule-based systems, reinforcement learning, RNNs, and other supervised learning methods. Ultimately user simulators serve the purpose of providing a realistic representation of a human user so that dialogue agents (especially

reinforcement learning agents) have an entity to train against. It is the job of the user simulator designer to represent behavior that isn't just realistic but useful to the training and success of the dialogue agent.

Our user simulator is an agenda based module that responds to agent actions based on a set of defined rules and stochastic behavior. There has recently been research, as discussed in section 2.5, exploring data-based models that are more generalizable, realistic, and robust than rule-based agenda models. However, rule-based user-simulators are still considered to be useful in domain-specific contexts (as in our framework) and as a method for bootstrapping data-based models.

The user simulator in our framework has a very specific behavior and holds the primary responsibility of responding to the agent's database query suggestions in a realistic manner, while initiating conversation to continue an episode's dialogue. Through these rule-based actions, the user simulator seeks to teach the DRL agent to optimize basic behaviors such as timely requests and informs, non-repetitiveness, improved context awareness, among other behaviors that the user simulator can explicitly or implicitly encourage.

The user simulator operates through the step method, which takes an agent action as

input, and returns a user action and reward. Once the agent action has been received, the user simulator will extract the intent from the semantic frame and either respond to a request, respond to an inform, or respond to a "match found" message. Within these subroutines, the user simulator will either populate its own request slots, inform slots, or both.

When responding to a request message, the user action is fairly simple. Since the user has a list of inform slots in its goal, all it needs to do is respond with the correct slot value if it possesses it. Otherwise, the user should continue the dialogue with its own question if this is a user goal request slot that it plans to populate in the future. If this is not an important piece of information, the user can simply respond with an "I do not care" message. When responding to an agent inform, the user simulator must check the inform slots of the movie suggestion by checking compatibility with its goal. Since the agent is making suggestions based on incomplete information, certain details of a movie suggestion may be incompatible with the user goal. The user can then respond with corrections to these slot values with its own inform message, continuing the dialogue.

The reward function used in our platform is fairly simple and essentially focuses on providing a negative reward for a failed dialogue, and a positive reward for a successful

dialogue. The function linearly scales with the number of rounds, where positive rewards are scaled by double the factor as negative rewards. This reward, however, is only delivered upon completion of the episode, while a reward of -1 is accumulated at each intermediate step to eventually punish longer conversation length. These rewards are embedded in all state transition data points and cached in the experience replay buffer to be used for training at the appropriate time. This represents a sequence of small negative rewards that will lead to either a large penalty or large reward, which the DRL agent can use for performing deep Q-learning.

Dialogues officially end when the agent finds that there are no possible matches, the user picks a match, or the maximum round number has been reached. Maximum number of rounds is an important hyper-parameter as it protects against infinitely looping behavior and encourages the agent to find a match as quickly as possible. This number is an area of study, however, since it effectively determines how much time an agent has to find a match, where this variable urgency can produce different behavior.

3.4 Goal Shifting Theory

Our research here is ultimately focused on improving DRL agent performance. However, instead of modifying the agent itself, we are modifying the end-to-end training framework used by a Microsoft research team in 2017 [15]. There are many modules included in the framework that track conversation history, access database information, and even simulate users. Our research seeks to improve this entire system by adding small, specific changes to the user simulator with the goal of representing goal-changing behavior during training episodes. Since many areas of the framework are interlinked, this requires a deep understanding of how information is stored and transmitted module to module. In this section, we will discuss the theory behind a few potential implementations of goal shifting behavior in a minimally invasive, but impactful way.

3.4.1 When to Goal Shift

While changing a user goal is functionally quite simple, there are a few major challenges involved in actually implementing user goal shifting. First, we need to execute goal shifting in a manner that realistically reflects human user behavior. Additionally, we need to execute goal shifting in a way that works with our current framework and doesn't

cause catastrophic contradictions or assertion failures that break our system. Changing the goal is as simple as choosing a slot in the user goal, and changing the corresponding value to something else in our dataset of potential slot values. However this often requires more nuanced work to reflect a realistic goal change in context.

It is important to note that this type of slot value change already occurs in the error model controller. However, this change is only temporary and represents a communication error which does not actually persist to the user goal. The other major difference is that the EMC module's value change method can occur with any outgoing user message regardless of context, while user goal shifting should only happen in certain cases that accurately reflect a human user "changing their mind".

Before talking about specific dialogue scenarios where it may be appropriate for a goal change to happen, it is important to commit to a working definition of what a goal change is in the context of our framework.

A goal change is when the user simulator has previously initiated and responded to an action, then permanently changes a detail in its goal that may contradict information it has previously conveyed.

This definition helps us find and simplify situations where a goal change may be

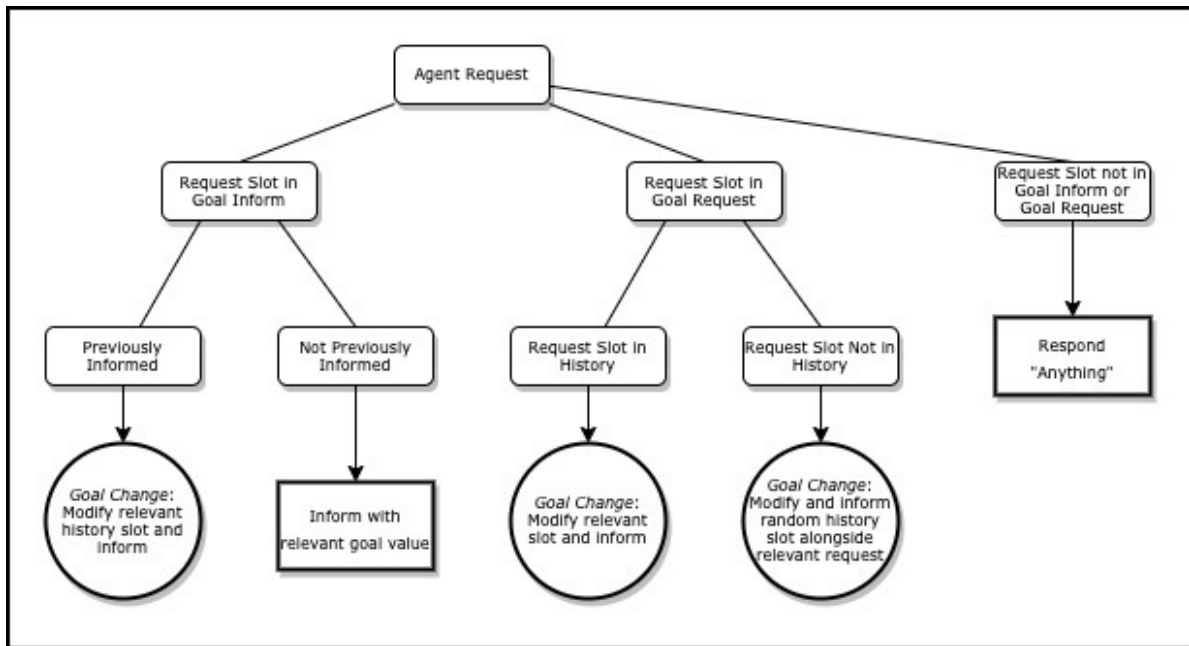


Fig. 3.3: Decision tree diagram for user simulator response to agent request

appropriate. It is important to only change information that has been previously conveyed, because changing goal information that was previously unknown to the agent does not introduce any new user behavior and will lead to the agent most likely training in a fashion similar to before. It is also important to notify the agent of this change in some way, since hiding a goal change event would only result in more difficult to satisfy user goals, without providing any meaningful information for the agent to learn from.

3.4.2 Respond to Request

Fig. [3.3](#) depicts a breakdown of actions made by the user simulator when presented with an agent request action. The purpose of agent requests are primarily to probe a singular

slot from the user to help reveal some information about the user goal. For example the agent may say "how many tickets do you need", which is a request from the agent to the user for the value of the ticket slot. While this is a simple request, the context the user simulator may be in makes the implication a little bit more complicated. The first filter that is applied to an incoming request is whether or not the slot is in the user's goal. If it is not, the agent can respond with the "anything" keyword, since that is a condition that the user does not care about. If it is in the user goal, the user simulator then identifies if it is a goal inform or request slot.

If the goal is in the user's goal inform list and it has been informed before, we know that the agent has been told this value in the past. If the user now modifies its goal slot value and informs this slot in response with the new value, we can trigger a contradiction on the agent side, representing a goal-changing event. On the other hand, if the goal is in the user's list of goal requests, we can assess if the slot has been requested by the user before and trigger a potential goal change event in either case. In the first case where the user has requested this slot in the past, we can check if there is a corresponding value, change its value in the user goal, then inform the agent with the new value. If the user has not requested this slot in the past itself, there is no previous value to contradict.

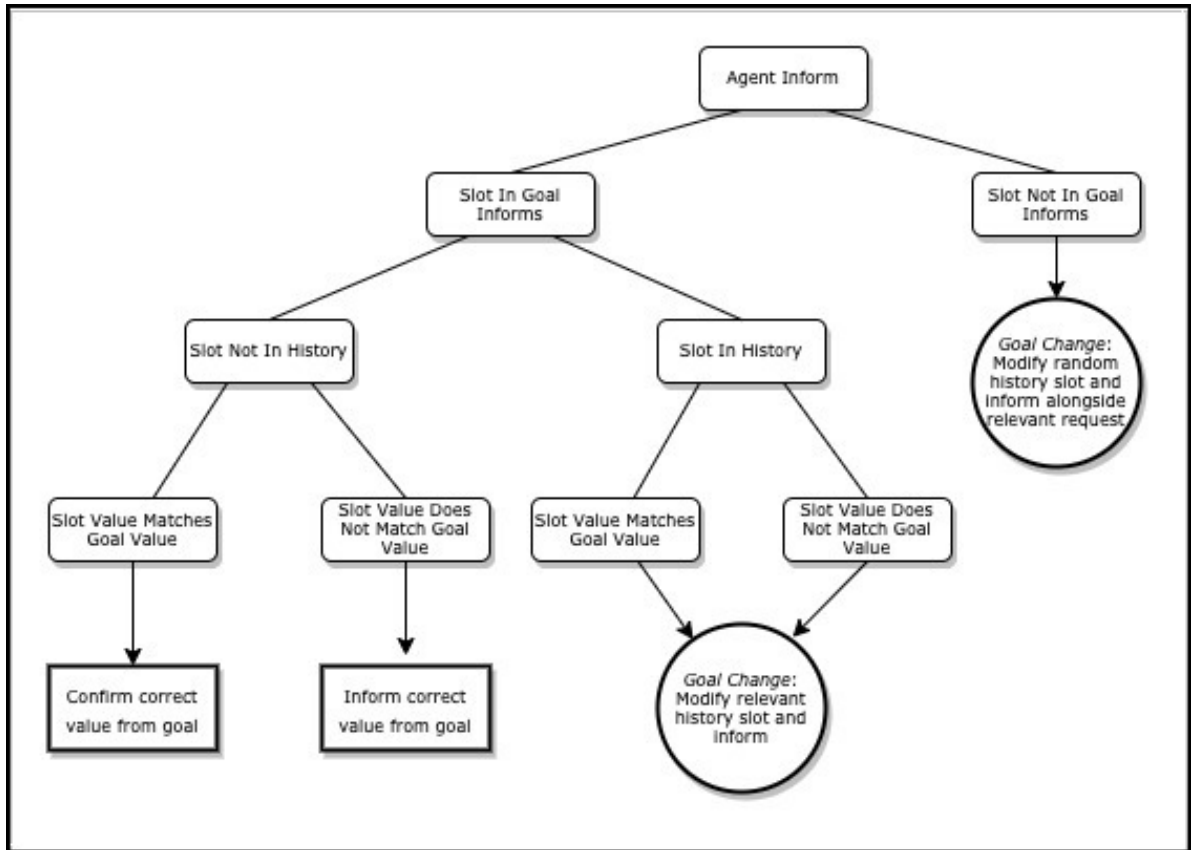


Fig. 3.4: Decision tree diagram for user simulator response to agent inform

However, we can still trigger a goal change event here by identifying a random goal inform value that is in the conversation history, and respond back to the agent with the relevant goal request slot and modified goal inform. This simulates a scenario where the user signals a goal change in a request, but for a slot not relevant to the agent's initial request.

3.4.3 Respond to Inform

Responding to an agent inform also presents potential cases for triggering goal change events. Fig. 3.4 shows a breakdown of responses to an agent inform message, in which the user is receiving information from the agent and has a few options regarding how to respond. The first check is to see if the inform slot is present in the user's goal informs. If it is not present, we know that the inform is not relevant to the user, so we can simply initiate a goal change by responding with a random slot request in relation to a modified historical slot value inform slot. Otherwise, if the agent's inform slot is present in the user goal and that user goal has been informed previously, we can trigger another goal change event by responding to the agent inform with a user correction of the same inform slot. In the case that the agent's inform slot is not present in the user's history, we should respond with either a correction or confirmation without a goal change, since there is no prior information to contradict.

3.4.4 Implementation

This theoretical exploration of goal-change events in conversational contexts helps us better understand the nature of goal-changing and how it may be applied in our system.

Through this framework, goal-change events are broken down into two main types: informs that explicitly notify the agent of inform slot value changes, and information requests that implicitly contradict inform slot values. These events can occur in different conversational contexts, and at different periods of the conversation, but in all situations seek to permanently change personal goal information. This is all meant to be done while conveying this change to the agent in a realistic manner.

In order to properly implement these changes in the given framework, we simply need to identify places in our code where our conceptual goal-change contexts occur. Once we've identified these contexts, we need to first change the user's goal, as well as latent representations of the goal that may cause contradictions later in the code. For example, the user has a historical representation of slot values, as well as a queue of planned slots that the simulator may request or inform. In order to fully change a goal, the historical slot values must often be changed, as well as the future slots to avoid confusing behavior and inconsistent information from the user. Once the user-goal implementation is completed, the user should be able to inform and request new information in a consistent way, which the agent can ideally identify and learn from in order to optimize behavior in this new environment.

CHAPTER 4

Experimentation

In this section, we explore the effects of user simulator goal shifting on end-to-end goal oriented dialogue systems. We specifically seek to observe how goal shifting affects training and testing DRL agents with quantitative and qualitative measures. In section 4.1 we will discuss our testbed setup which we use to automate goal shifting within our training framework for purposes of testing and diversity. In section 4.2, we will discuss some quantitative measures primarily relating to how and when goal shifting is introduced in the training process. We hope this will show some of the impact in training time and total accuracy when introducing this new user behavior in our end-to-end training framework. Next, we will discuss qualitative effects of training DRL agents against users that shift their goal mid episode. We hope to gather some insights on how agents react to goal changing, and some of the learned techniques that they employ.

4.1 Experiment Design

4.1.1 Goal Changing Implementation

In this section, we explore the effects of user simulator goal shifting on end-to-end goal oriented dialogue systems. We specifically seek to observe how goal shifting affects training and testing DRL agents with quantitative and qualitative measures. In section 4.1 we will discuss our testbed setup which we use to automate goal shifting within our training framework for purposes of testing and diversity. In section 4.2, we will discuss some quantitative effects relating to how and when goal shifting is introduced during the training process. We hope this will show some of the impact in training time and total accuracy when introducing this new user behavior into the end-to-end training framework. Next, we will discuss qualitative effects of training DRL agents against users that shift goals mid episode. We hope to gather some insights on how agents react to goal changing, and some of the learned techniques that they employ.

4.1.2 Controlling Goal Change Events

In order to test multiple different goal shifting events, we identify locations where an event should occur and call a function which will occur with a given probability. If a goal random value is selected relative to the probability currently selected, then a goal shift event specific to that dialogue context will occur. While we are able to set the probability of this goal shift event happening during a certain dialogue event, we cannot set the probability of the dialogue context itself taking place. While user actions are deterministic given a specific agent action and state, these variables are hard to control and are based upon latent information such as the weights of our DRL agent as well other random events that may have occurred to change the trajectory of training. Fixing agent actions would be self-defeating since these actions are meant to train a model policy through diverse experience.

In order to attempt to control how often goal shifting events occur, we count how often a dialogue context event occurs and tweak our event-specific goal shift probability to reach an effective total frequency of goal shifting. However, as our agent trains and the distribution of its actions shift, the likelihood of an event occurring that might trigger a goal shift changes as well. This would hopefully cause convergent dialogue agent behavior,

but can also lead to undesirable behavior as agents may simply avoid actions which lead to goal shifting, or unrealistically exploit this behavior in a way that maximizes training reward.

4.1.3 Quantitative Metrics

At a high level the primary quantitative measure we are observing is ticket booking success rate. This is because goal-shifting is meant to introduce a major complication into a given episode that provides a realistic challenge to a dialogue agent. This should allow the newly trained agent to handle goal shifting situations more robustly. However, even as we consider how human agents should handle these situations, we see that it is not so straightforward.

The dialogue agent in this framework is connected to a dialogue manager that handles the history of the conversation, intended responses, as well as access to a database. Since the dialogue agent doesn't encode state information on its own, it relies on these external modules to prepare state representations that it can use to generate actions. Due to this interwoven framework design, we think another important quantitative metric to understand is simply how often certain goal shift events occur in relation to a given goal

shift probability and training stage. This tells us quantitative, fine-grained information about how goal-shifting user behavior affects agent actions, which may shed light onto the sorts of behavior that agents are capable of learning from interacting with our new user simulator.

4.2 Experiments

4.2.1 Control System

The first quantitative experiments we conducted were on the control system with no goal changes. This was done to identify general trends and behaviors in the system in order to appropriately compare behavior of the modified system.

Fig. [4.1](#) illustrates the existing inverse relationship between success rate and average number of turns. This relationship appears in the unmodified system as well as modified system, and holds true for most if not all configurations. This is primarily due to the fact that the reward function provided by the user is fully based on episode length, where an episode success yields a positive factor relative to the number of turns, and a failure yields a punishment also relative to the number of turns. Additionally each

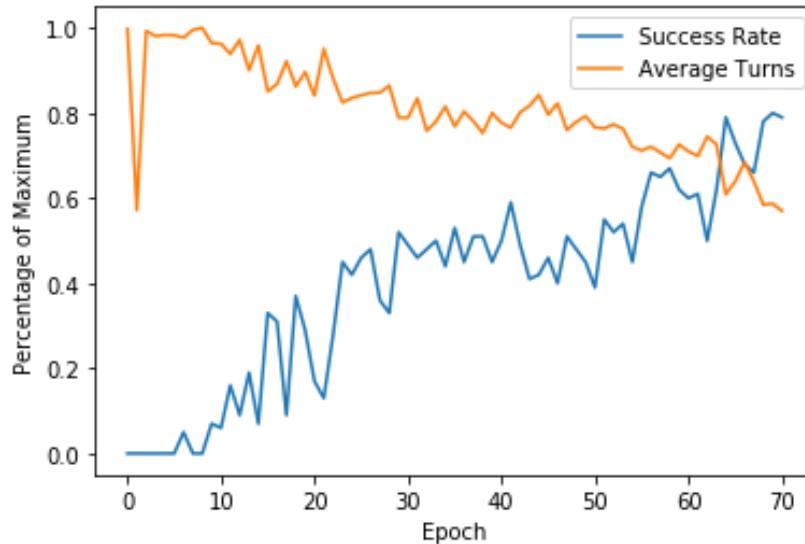


Fig. 4.1: Blue: Episode success rate within epoch, Orange: Average turn count in Epoch relative to maximum

step is discounted by a constant negative value punishment. If a DRL model were to be trained based on this requirement, it is likely that the final model is able to finish dialogues in fewer steps. Since success rate is a metric for what percentage of episodes completed the task of booking a movie, we can hypothesize that optimal behavior might be whichever behavior results in shorter conversations.

Fig. 4.2 illustrates a correlation between the average number of turns in an episode and the occurrence of potential goal-change events. It is important to note that total event count is not actually a count of goal change events but of events such as those described in section 3.4 where a goal change could potentially occur. What Fig. 4.2 suggests is that the count of these types of events follows the same pattern as the average number of

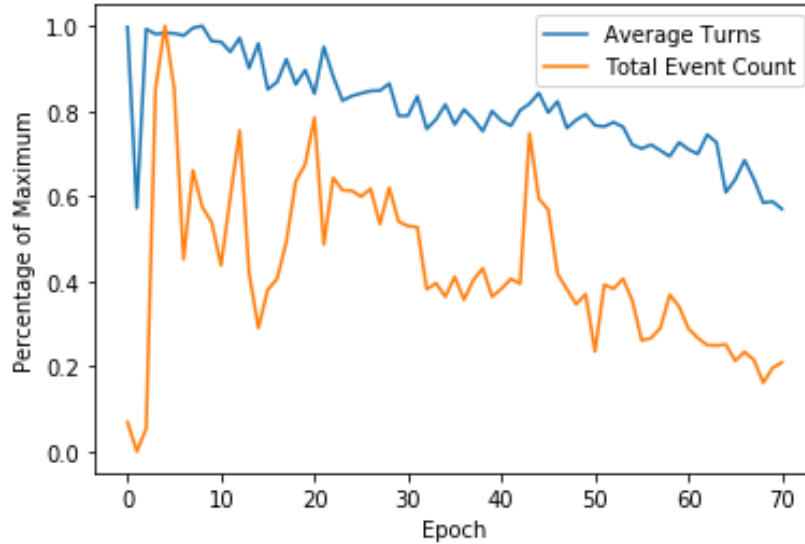


Fig. 4.2: Blue: Average number of turns relative to maximum, Orange: Total event count relative to maximum

turns, and thus is also inversely related to success rate, as shown in Fig. 4.1. This makes intuitive sense as these kinds of events take up a certain percentage of the action space, and as the number of available turns shrinks, so should the count of these potential goal-change events. However, as our model trains, we can see that there are instances where the number of events drastically grows or shrinks at a much larger scale relative to the average number of turns. This indicates that the relationship in Fig. 4.2 is not as tight and that this event count can also be affected by different policies the agent learns when exploring the action space. With this frame of thinking, we can identify moments where the agent learns a more successful policy based on goal changing by observing an increase in event count as well as an increase in success rate, or even a decrease in

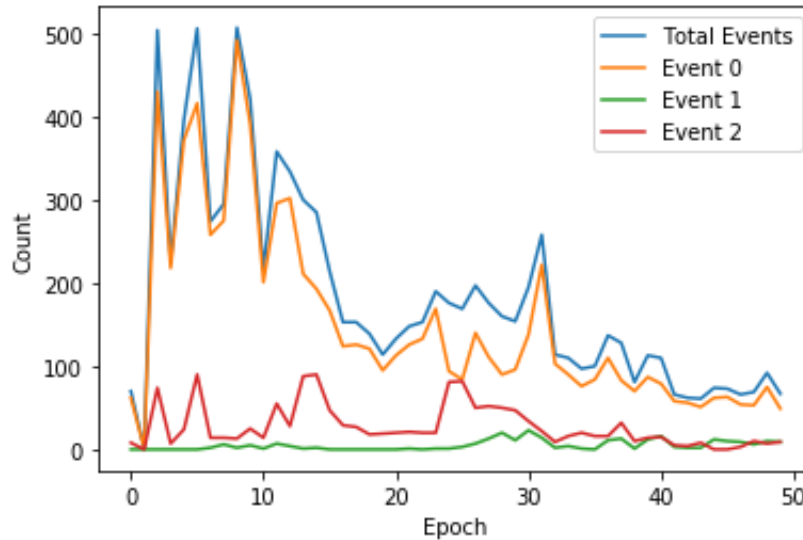


Fig. 4.3: Sum breakdown of various potential goal switch dialogues in control framework average number of turns.

Fig. 4.3 depicts the breakdown of the total events shown in Fig. 4.2 broken down by type of event. Event 0 corresponds to when the agent's request semantic frame includes a slot that the user can inform from its goal informs. Event 1 corresponds to when the agent's request semantic frame includes a slot that is in the user's goal request list as well as history list. Event 2 corresponds to events where the agent's request slot is in the user's goal request slot list and slot list of future actionable slots. In Fig. 4.3, we can see that as a sum, these event counts are inversely proportional to success rate, but we can also see that these event counts are diverse in proportion of the total as well as relationship to each other. For instance, event 2 appears to be inversely proportional with event 0,

Event 0		
	1% Chance	2% Chance
Epoch 0	0.09	0.25
Epoch 10	0.52	0.03
Epoch 30	0.61	0.49
Epoch 50	0.87	0.83

Table 4.1: Success rate of goal change by inform

while event 1 is small and makes up a relatively small proportion of events occurring. This graphic is important because it may help identify which events contribute to longer average episode lengths and give us a reference point for which goal-changing episodes have the most effect on success rate among other factors.

4.2.2 Experiments with Goal Change by Inform

Table 4.1 represents the dual effects of goal change event probability and start epoch of goal-changing activity, specifically on goal-changes occurring in the inform context. The first immediate result to see from this is that as goal changing is introduced to a model that has already been trained on a large amount of non goal changing epochs, it is more likely to sustain its accuracy. This at least shows that the addition of goal-changing behavior is not catastrophic to the training of the model, but may hinder future training. Fig. 4.4 more clearly demonstrates this behavior where introducing the goal-change too early can result in unstable behavior which actually causes the model to decline

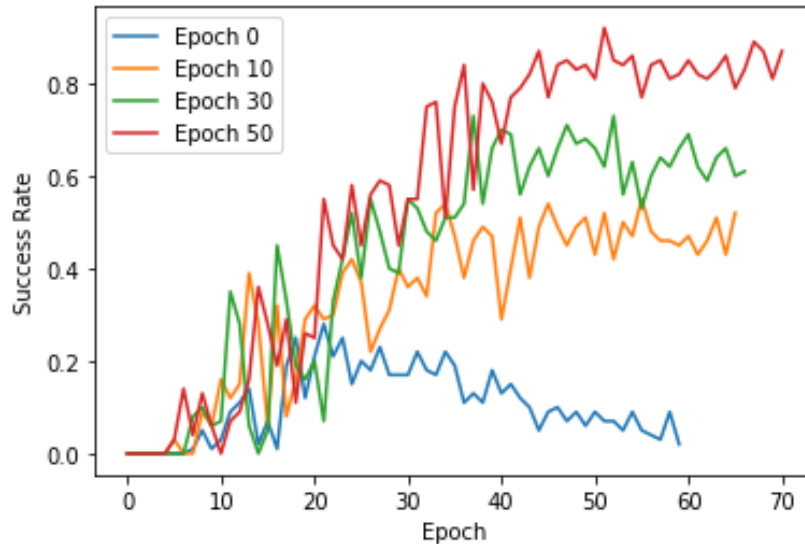


Fig. 4.4: Success rate of various start epoch times with 1% probability of inform goal change

in accuracy. In other cases, introduction of behavior to a more mature DRL agent typically results in the accuracy being held constant without significantly dropping. This suggests firstly that goal-change behavior poses a challenge that the model is not able to adequately overcome, which also blocks future optimal behavior. However, Fig. 4.4 also illustrates that it is possible to introduce this behavior without causing the model to degenerate if presented late enough in training.

Fig. 4.5 primarily shows the relationship between the model success rate, average number of turns and the goal-change context occurring. It is important to note that the dialogue context occurring does not necessarily mean the actual goal-change will occur, since this is based on a random probability. More specifically, this figure helps show how often

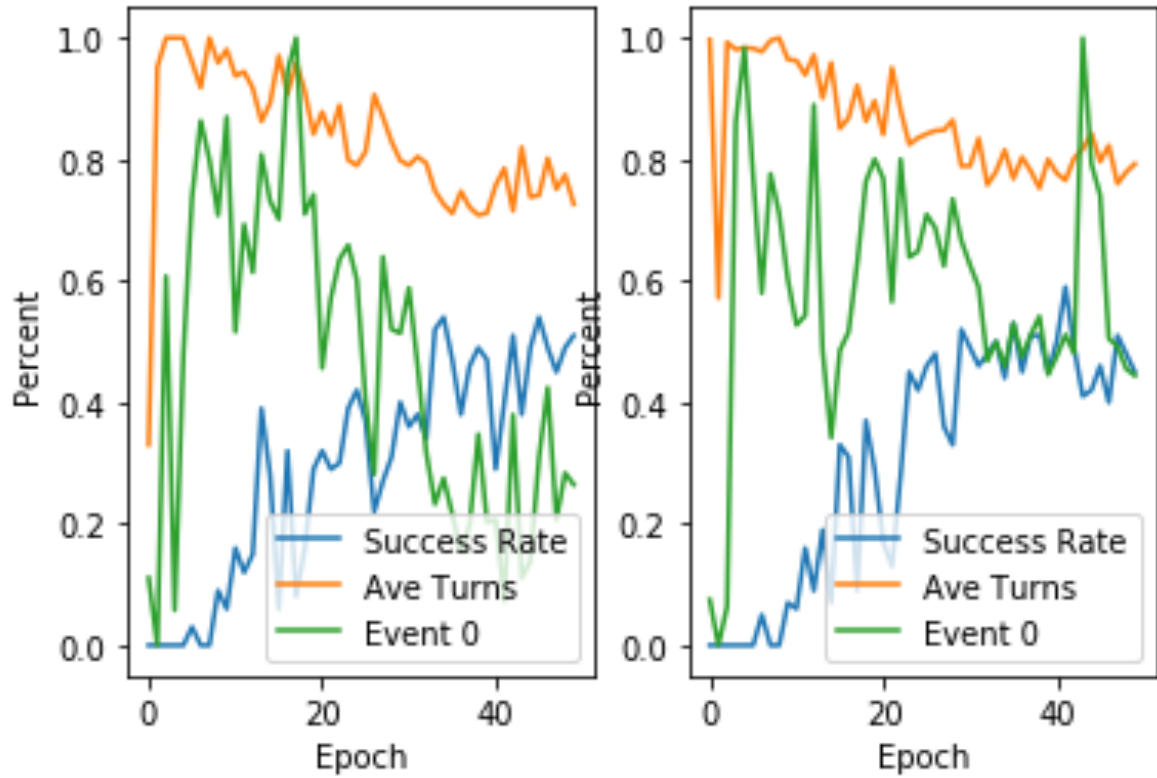


Fig. 4.5: Average turns and event 0 count relative to their respective maximums. Left: Goal change at epoch 10, Right: No Goal-change

Event 1		
	1% Chance	2% Chance
Epoch 0	0.68	0.77
Epoch 10	0.73	0.77
Epoch 30	0.75	0.81
Epoch 50	0.73	0.71

Table 4.2: Success rate of goal change by request

a model may initiate behavior which can lead to a goal-change event. What we see is that under a control scenario, the context we chose for goal-changing occurs nearly directly proportional to the average number of turns. However, when we introduced the goal-changing behavior with a 1% chance, we saw that the dialogue context itself of a goal-change event dropped further in frequency relative to its maximum. In this specific scenario, we also see that this is observed at a stage of training where the control DRL model and goal-change model are performing with similar success rates. This suggests that the DRL model may be learning to avoid conversational dialogues which may potentially lead to goal-change events. This is significant because, depending on how important the conversational context is to general dialogue completion, goal-changing can negatively impact success rate by preventing the agent from learning fundamental behavior.

4.2.3 Experiments with Goal Change by Request

Table 4.2 demonstrates the effects of goal change event probability and starting epoch of goal-changing activity, in relation to goal-changes occurring in the request context.

In these experiments we use notably higher goal-change probabilities than in Table 4.1.

This is because, as we saw in Fig. 4.3, request events which repeat historical information occur at a much lower frequency. Using a probability that it is too low in this situation could result in the model never actually experiencing a goal-change event and presenting a misleadingly high accuracy value. In Table 4.2 we see that although the training accuracies are lower than the baseline control final accuracy (roughly 0.84), this seems to not be due to the epoch time of goal-change event introduction, or percentage likelihood of goal-change event activation. This is likely because the model sees these contexts so infrequently that it has enough room to train on normal behavior outside of goal change events. This result is contrary to the other goal-change contexts which may happen so frequently that it takes over normal behavior that would normally allow the model to continue training.

Fig. 4.6 depicts the relationship suggested by Table 4.2 where low frequency goal-changing events have little impact on success rate over time, allowing the DRL model

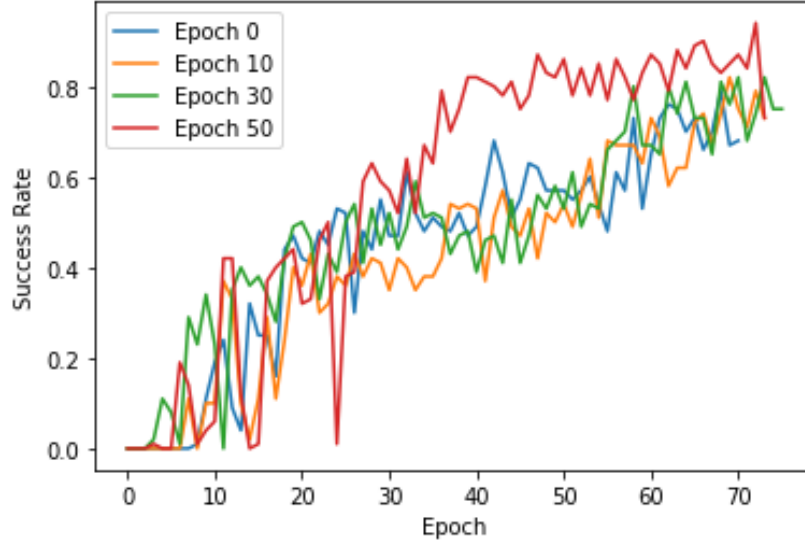


Fig. 4.6: Success rate of various start epoch times with 1% probability of request goal change

Event 2		
	1%,10% Chance	2%,10% Chance
Epoch 0	0.12	0.25
Epoch 10	0.56	0.04
Epoch 30	0.51	0.54
Epoch 50	0.84	0.57

Table 4.3: Success rate of goal change by inform / request

to continue learning on normal behavior. It is possible to see however, that the early introduction of goal-changing events at epochs 0, 10 and 30 cause a slower rate of learning, where training sequences that introduce goal-changing at epoch 50 are able to discover more successful long term policies.

4.2.4 Experiments with Mixed Model Goal Change

Table 4.3 depicts data from the method of using both types of goal-changing events in an episode. This can be beneficial as the total amount of goal-change events increases while minimizing impact on normal behavior in each conversational context. This essentially introduces more goal-change events by expanding into more areas of the conversational space, rather than increasing the amount of goal change events in a single conversational context. This can potentially make a user simulator more realistic in a way where consistent conditions in different dialogue contexts cause a reasonable amount of goal change events. This is likely an improvement over having a specific context cause most of the events, while another similar context causes very few.

Fig. 4.7 shows a comparison between various goal-change start epochs, compared between a 1% probability of goal-changing and a 2% probability of goal changing. We first see some obvious trends that we've noted earlier such higher overall success due to late goal-change start epoch, and lower instantaneous goal-change chance resulting in higher long term success. We can also see that with a 2% chance of goal-changing events, the start epoch of 50 begins to drastically drop in success rate once goal change events begin, while the 1% chance on the left is able to maintain success rate without dropping.

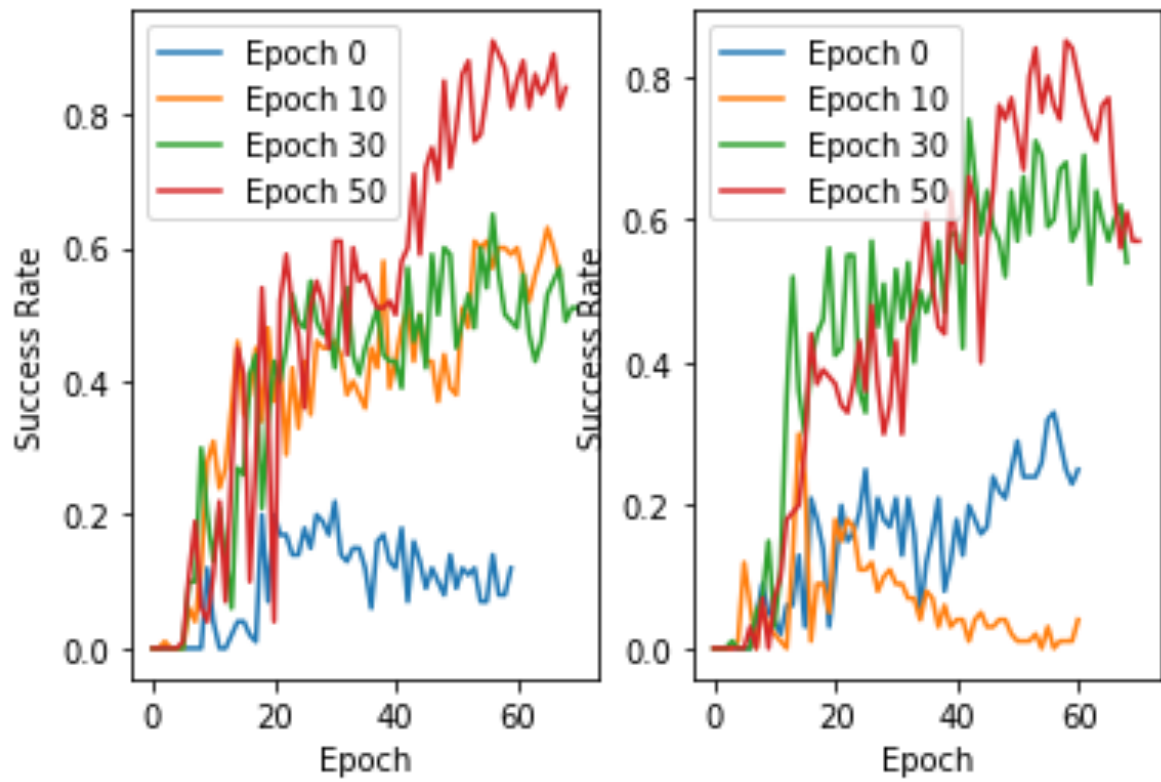


Fig. 4.7: Left: Event0: 1%, Event1: 10%, Right: Event0: 2%, Event1: 10%

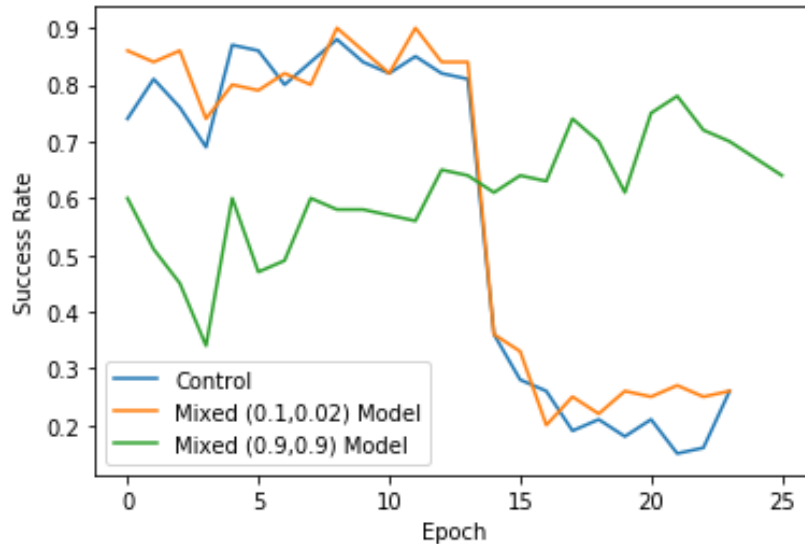


Fig. 4.8: Success rate of three models on a 90% goal-change test environment

However, Fig. 4.7 does also depict a few exceptions that indicate complications in our model. First, on the right side of the figure we see that starting epoch of 10 becomes unstable and converges to nearly 0% success once goal change events begin, while starting epoch of 0 manages to learn and plateau without falling. These general observations suggest that there are stages of training at which various policies are being explored where introduction of goal-change events can be either catastrophic or nearly unnoticed.

4.2.5 Testing Against Control Environment

Fig. 4.8 shows a depiction of a test setup where a selected model trained with a small percentage of goal-changing events is compared against a model trained with a high

number of goal-change events, as well as the control. These DRL agents are tested by further training them over a small number of epochs against an aggressive goal-changing user-simulator that changes its goal 90% of the time when in a goal-changing context. The first 10 epochs establish a baseline where the model does not initiate any goal-change events, and the last 10 epochs show the performance of the model against the new aggressive goal-changing. The model which was trained with a small number of goal change events was able to perform comparably to the control model with a high success rate in the beginning, but once goal-changing began, both of these models severely and consistently plummeted in success rate. By contrast, the model trained with a high percentage of goal change events, while initially under performing the other two models, was able to remain consistent and even slowly improve over the rest of the training epochs while the other models dropped significantly in success rate.

CHAPTER 5

Conclusion and Future Work

To summarize our work, we modified the user simulator of an existing end-to-end goal-oriented DRL training framework so that it may change its goal with different frequencies and in different scenarios. We identified places in the existing rule-simulator where goal change events may occur, and discussed the rationality and realism of these goal changes. With these goal change events identified, we created a testbed which allowed us to control the frequency at which various goal change events occurred, and at which training episode.

With this testbed, we conducted various experiments by sampling combinations of goal-change type, start epoch, and frequency. We also observed information such as success rate, average episode length, and conversational event distribution. Since our testing variable space is very large, we tried to make educated guesses as to which combinations to use, and arrived at a few significant observations.

Before conducting experiments on different methods of goal-changing, we analyzed the

control model using our testbed. We found that the inverse relationship between success rate and average number of turns in goal-oriented dialogue systems is very strong. In addition to this, we also found that the number of conversational contexts that may lead to goal-change events is also related to the average episode length. Furthermore, we saw that the most frequently occurring type of potential goal-change dialogues were agent requests on goal slots followed by user informs. We found this behavior to be common in the agent's standard policy of rapid information collection.

The experiments we conducted on goal change events were on two types of dialogues: actions where the user would respond to an agent request with information from its recently modified goal, and actions where the user would respond to a request with its own request alongside information from a recently modified goal. We first saw that goal-change behavior had the potential to be very intrusive on basic goal-oriented training. We then noted that early introduction of goal-change behavior in our framework often led to trained models with low overall success rates. Ultimately the timing and frequency of the same goal change would affect if a model's success rate would deteriorate, remain constant, or improve (which was rare). We also found that a larger amount of potential goal change scenarios with smaller likelihood of individual event triggers was a more

realistic way to provide a significant amount of goal-change training events. We finally found that with this type of goal change, the model would sometimes avoid dialogue contexts that could potentially lead to goal shifting. We were not able to confirm an improvement in overall success rate by training with these methods.

Finally, we tested various models against a more aggressive user simulator which changed its goal nearly every chance it had. We saw that the model trained with some goal-change events which performed well on its own training data, did not outperform the control model in this more challenging environment. However, we saw that the model trained in a high goal-change frequency environment was able to see nearly the same success rate in an environment with no goal-change events as compared to an environment with nearly all goal-change events.

In future work, we will explore further potential segmented training methods, as well as mixed models to provide more diverse model responses in what are effectively contradictory training environments. Additionally, we will try experimenting with different potential episode lengths, since many episode failures in our training resulted from max episode length being reached. This is additionally important as we have confirmed that goal-change events lead to longer average episode lengths. Furthermore, we will

investigate modified data-oriented additions to user simulators that make more natural and realistic decisions based on goal slot importance. Additionally, we will seek to use this model to bootstrap RNN conversational models that have been recently used for data-based user simulators.

Bibliography

- [1] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9:36–45, 1966.
- [2] G. Pirani. *Advanced Algorithms and Architectures for Speech Understanding*. Research Reports Esprit. Springer Berlin Heidelberg, 2013.
- [3] Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, pages 956–962, 2000.
- [4] James Glass. Challenges for spoken dialogue systems. In *Proceedings of the 1999 IEEE ASRU Workshop*, 1999.
- [5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
- [7] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.
- [8] Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. *Interspeech 2016*, pages 1151–1155, 2016.
- [9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.
- [10] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.
- [11] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

- [12] Marilyn A Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416, 2000.
- [13] Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *Ninth European Conference on Speech Communication and Technology*, 2005.
- [14] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [15] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 733–743, 2017.
- [16] Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *Ninth European Conference on Speech Communication and Technology*, 2005.
- [17] Kallirroi Georgila, James Henderson, and Oliver Lemon. User simulation for spoken dialogue systems: Learning and evaluation. In *Ninth International Conference on Spoken Language Processing*, 2006.
- [18] Jost Schatzmann, Kallirroi Georgila, and Steve Young. Quantitative evaluation of user simulation techniques for spoken dialogue systems. In *6th SIGdial Workshop on DISCOURSE and DIALOGUE*, 2005.
- [19] Xiujun Li, Zachary C Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*, 2016.
- [20] Izzeddin Gür, Dilek Hakkani-Tür, Gokhan Tür, and Pararth Shah. User modeling for task oriented dialogues. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 900–906. IEEE, 2018.
- [21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.