Reducing Friction Between Conversational AI and the User

Nik F. Bailey
bail0301@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris , Morris, Minnesota, USA

Abstract

As conversational AI, such as the Amazon Alexa, Apple Siri, and Microsoft Cortana, become more prominent in the daily life of the individual, complications may occur as development of better AI takes place. One such problem that an everyday user of a conversational AI will have, is formulating a command in such a way for the conversational AI to perform the desired action. This friction between the user and the conversational AI can grow over time, and any friction between a product and a customer is unwanted. To reduce this friction, one approach involves absorbing Markov chains to group similar commands to the correct one and sending the result to the user. Another approach is to use query rewriting combined with stored feedback data of the individual to accurately interpret what action the user intends for the conversational AI to perform. In this paper, an explanation of both methods will further increase understanding on the subject and help to determine which method is more successful in reducing user friction.

Keywords: friction rate, conversational AI, Markov chains, query rewrite

1 Introduction

The problem presented in this paper is that conversational AI systems, like the Amazon Alexa, do not always understand the user's query. This causes the AI to give an incorrect or unwanted response, which leads to unwanted friction between the user and the AI. Two different approaches to reducing user friction are using Markov chains to absorb user input towards the correct output, and query rewriting to get the intended response. The research this paper describes focuses on the Amazon Alexa AI agent, and while the ideas and concepts can be applicable to other AI, from this point on when discussing AI we will be referring to the Amazon Alexa. After going through some necessary background information, we will discuss two alternatives to reducing friction between a user and an AI, and the results of both options.

2 Background

In this section, we will explore topics that are relevant to conceptually understand the solutions offered in this paper.

2.1 System Overview

Most conversational AI operate with a conventional spoken dialogue system. A conventional spoken dialogue system contains five components: Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), dialogue management (DM), a natural language generation system (NLG), and a text-to-speech system (TTS). When an individual interacts with the conversational AI, the audio goes through the ASR and gets converted into text. This text, or query, is passed through the NLU component and then the DM component, which extracts the user's intent. This interpretation of the user's query is called an NLU hypothesis, and helps determine what action to take in response. The action to perform is decided through the NLG component, and finally the TTS generates the audio response, which is sent to the device to complete that instance of interaction. An example image of what this system looks like can be seen in Figure 1 [5].

2.2 Feedback-Based Self-Learning using Absorbing Markov Chains

Markov Chains is a system of states where two states are connected by the probability to move from one state directly to the other. An example of what a Markov chain could represent is a baby, and what possible states the baby could be in. Some states that the baby could be in would be eating, sleeping, crying, or playing. A Markov Chain would not only show all the possible states, but would also express the probability of travelling from the current state to another state. In the baby example, the Markov Chain would be able to tell us the probability of the baby that's crying to move straight to playing without sleeping or eating. Markov Chains are used in a variety of applications. They are used in economics to predict market crashes and lengths of time between recessions and expansion. Markov Chains are also applicable in many of the sciences, like biology or physics or chemistry, and also in music software. [3]

Absorbing Markov Chains are Markov Chains that have one or more states where it is impossible to transition out of said state. States such as these are labelled absorbing states.

Transient States are states in a Markov Chain that have two important properties. The first property is that they can be reached in a finite number of steps. The other property, which distinguishes them from absorbing states, is that once

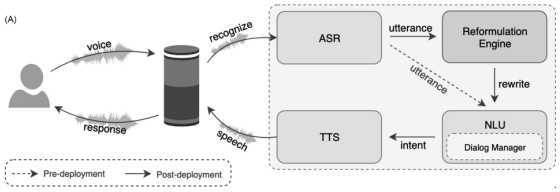

Figure 1. Process of a conversational AI responding to user input [4]

in a transient state, it is possible to leave again. Once a transient state is reached, and one transitions from it to another state, it is extremely unlikely to return back to that transient state.

Absorbing States are states in a Markov Chain that are impossible to leave. While there needs to be a positive probability chance of reaching said state, once in an absorbing state, there is no leaving, and while a Markov Chain can continue to process, it will remain in the absorbing state.

Friction Rate is the likelihood of an automated system response causing friction with the user. When the user inputs an utterance or query, the Alexa system creates an output response based on the inputted utterance, along with contextual clues, such as if the Alexa is playing music or not. If the response is not what the user expects, then it will have a high friction rate. Friction can be caused by a variety of issues, such as not interpreting the utterance correctly (Alexa hears "*Play maj and dragons*" instead of "*Play imagine dragons*"), or a user not using an exact command ("*Don't play this song*" could mean "*Skip this song*" or "*Take this song off the playlist*"). Calculating the friction rate is helpful to determine if the response the system has come up with is in accordance with what the user intended, or if the system should scrap it and attempt a different response. [4]

2.3 Self-Learning Query Rewrite System

Neural networks are systems that are created to be structured and act similar to the human brain, where repeated activation of certain neural paths leads to learning. These networks are made up of different node layers. These layers include an input layer, an output layer, and one or more hidden layers. Each node, or 'neuron', connects to another node and has a set weight and threshold value. A node is activated if the output goes over the set threshold value, and the data is sent on to the next layer in the neural network. The more layers there are, the better the neural networks can refine information they get and the better they are to make accurate predictions, or accurate output, from the inputted data. [1]

Deep learning is a neural network with three or more layers. [2]

Hidden embedding is a layer inside a neural network. This layer is essential to letting the system process input information data efficiently, and also to learn about the relationships between inputs. The way this works is that it when given an input, it converts from having multiple dimensions to ideally one dimension. Think of it was points on a two dimensional graph. Instead of listing the points by their coordinates, we give each point a number, and if we want to refer to the point we refer to it by its given number. It seems unnecessary when there are only two dimensions present, but is extremely useful with multiple dimensions, and hundreds or thousands of data points.

Gradient Boosted Decision Trees (GBDT) are used to solve prediction problems. GBDT approaches a problem in a way that simplifies the objective and by doing so also reduces the number of iterations needed to obtain an optimal solution. A GBDT is a decision tree that is constructed to correct any errors of previous trees. The first decision tree is used, and the predicted results are compared to the actual, real world results. Any errors there are recorded, and a new decision tree is made, with adjustments made to help make it more accurate. This process is continued until the tree is always correct or if the error range is sufficiently small enough to be used.

Information Retrieval (IR) model extracts information out of different features of a query to learn more about what the query is about. For this research, there are the main types of IR features: text features, document features, and query-document features. Text features captures any differences between the original query and the rewrite. Document features provide information relative to the historical interaction of the NLU hypothesis, such as how many queries led to this NLU hypothesis, and the historical friction rate of each of those queries. Query-document features contain information about the relevance of the query such as the number of queries. [5]

Deep Structured Semantic Model (DSSM) is a deep neural network (DNN) modeling technique for representing text strings in a semantic space, and models similarities between two text strings. DSSM is used in a variety of applications, including web search ranking and information retrieval, question answering, ad selection and ad relevance, image captioning, and machine translation.

3 Feedback-Based Self-Learning using Absorbing Markov Chains

This section describes research that focuses on the method of using absorbing Markov Chains to reduce friction between

conversational AI and their users. When a query, or utterance is inputted to the system, instead of moving on to the NLU after going the the ASR, the utterance is sent to the reformulation engine, which uses Markov chains determine if a rewrite is necessary. That result is sent to the TTS to produce the output. The reformulation engine uses an online database to store the user data, which is updated regularly from recent Alexa log data. That log data is compared to the original utterance, testing the friction rates of utterance, and the one with the highest friction rate is prevented from uploading to the database. [4]

3.1 Dataset

The dataset is constructed from three months of Alexa log data from millions of users, to get a randomized collection of utterance data. That randomized data is then partitioned and sorted into groups of finite sets of successive utterances. These sets are then divided further into sessions, where divisions are made between utterances with a set time delay of forty-five seconds between them. This helps break the data into individual interactions between the user and their Alexa-enabled device. Because of the use of anonymous data, there is a wide variety of results from similar utterances. Note that interjecting utterances, utterances from the user that interrupt the current intent or stop the interaction before getting to the intent, are removed. Therefore, each session can be described as a linear directed chain of successive utterances. [4]

3.2 Absorbing Markov Chain

The interpretation space is made up of all the utterances that are involved with the dataset. Each utterance is seen as a node in the Markov model. In general, the interpretation space is the set of all transient states in the Markov model. The interpretation space is also known as the utterance space, or the space of all utterances. In its original form, it has extremely sparse connections between states. By using the domain and intent classifier, along with labeled data, the utterance space is condensed into a new interpretation space, grouping together similar utterances to connect it the same interpretations.

Given a session and the interpretations within it, each interpretation is seen as a transient state and the paths that lead to it. Each successive pair of interpretations represents an edge on the graph. Between two connected transient states, x and y, the probability of transitioning from state x to state y is calculated by taking the total number of times state y is directly linked to state x, divided by the number of occurrences of state x throughout all sessions in the current dataset.

The absorbing states in the Markov chain model are determined to be either failure or success. By using implicit and explicit feedback, we can figure out the state of an absorbing state. Explicit feedback is where the user attempts to correct the interpretation, such as trying to get the same intent again, or when a user interjects the interaction, stopping the attempt to get the correct intent. Implicit feedback is where the user gives up, or abandons the session once the Alexa fails to handle the request made. With the feedback given, the absorbing state is assigned a failure status or a success status, where success is defined as the absence of failure. These states are added to the end of all sessions

The absorbing states in the Markov chain model are either a failure, or a success. A failure state is when the outputted response was not what the user wanted, and the user either tried again, or gave up. A success state is when the outputted response was what the user wanted. The absorbing states are not naturally part of the Markov chain, and have to be injected onto the chain

Although the dataset is extremely large, with millions of users' data, it is noted that most (97.3%) have path lengths of 5 or less. [4] This is attributed to the fact that if a given user utterance results in a defective output, users will try to reformulate their utterance a few times before getting the correct output or giving up. From this, they concluded that the dataset contains several clustered Markov chain structures.

3.3 Experimentation & Results

The current mainstream approach to learning tasks has been sequence-to-sequence architectures. The Markov chain model was compared against a long short-term memory-based (LSTM) model to determine which approach was better at developing accurate rewrites.

The LSTM model, specified as a pointer-generator model, was trained with rephrase data. The rephrase data used was taken within a span of three months, such that the first utterance was a failure, and the second utterance, or the rephrase, was a success. This data trained the pointer-generator model to produce the rephrase when given the faulty utterance.

Once the pointer-generator model had been trained, the research team annotated over five thousand unique utterance-rephrase pairs that were generated by the Markov chain model. The accuracy of the Markov chain model, based on these results, was estimated to be 93.4%, and had a calculated win-loss ratio, successful rewrites to failed rewrites, of 12.0. Using the same utterance-rephrase pairs, the pointer-generator model resulted in a significantly lower accuracy of 55.2%. This isn't surprising, as the Markov chain model has the advantage of utilizing the interpretation space to aggregate the utterances and aggregates all the months of data instead of just the rephrases, and also takes into account the frequency of transitions from utterance to rephrase.

There were some benefits to using the pointer-generator model, such as it has a higher recall and it can learn patterns in utterances. A significant difference that put the Markov chain model ahead of the pointer generator model is that it

can identify when an utterance is successful, and will signal to itself when to not start a rewrite. [4]

It is noted that when the Markov chain model fails, it generally does so when the utterance is rewritten to a generic term, such as "*play*" or "*shuffle my songs*". This happens when the original utterance fails, and the user tries several different approaches that end up losing information, and the utterance eventually gets aggregated into a generic utterance. This can be corrected by applying rules to the rewrite building, or by inserting a learning-based ranker after the Markov chain generation. Another common way that the Markov chain model fails, is when the system changes the intention of the original command by changing the song name or artist. This commonly happens when the original utterance used to fail, and so the user gave up and asked to play a similar song instead, or the same song with a different artist. [4]

Launching the model online, the performance of Markov chain model rewrites compared to no rewrites was monitored for two weeks. There was a noticeable 30% reduction in defect rate, or user dissatisfaction. In another separate nine week trial, as defect decreased, user engagement was going up. The win-loss ratio was calculated after 3 months, and was determined to be 11.8, very similar to the offline win-loss ratio.

4 Self-Learning Query Rewrite System

The User Feedback Search based Query Rewrite (UFS-QR) system takes a query that is inputted into the system by the user, and determines if a rewrite should be triggered. If so, a rewrite is created and eventually sent to the NLU, restoring the flow of data and giving the user a response to their query. Any bad rewrites are disabled from continuing through the system. What makes a bad rewrite is the comparison of the friction rates of the rewrite and the original query; the one with the predicted higher friction rate is disabled from continuing. The UFS-QR system contains an index, a retrieval layer, and a ranking layer. The index is constructed from an offline process, the retrieval layer performs a nearest neighbor search, and the ranking layer ranks the rewrite candidates, returning the top rewrite as the final rewrite. Each of these parts of the system will be described in this paper. [5]

4.1 Indexing

The UFS-QR index is made up of a global index and a personalized index. The global index has rewrite candidates for all user interactions, and is generated from anonymized interactions between users and the conversational AI. The global index is sorted by the intended purpose of each query. Queries with the same intent are grouped together. So for queries whose intent is to play a specific song, such as "*Play Bad Blood by Taylor Swift*", they are grouped together based on that intent. [5]

The personalized index is constructed similarly to the global index, but is unique to the individual user. Taking from the individual interaction history, this index is vital as the context to a query may change based on the individual. An example of this is that a user could say "*Turn on the moon light*", and while for one user that could mean to turn on a lamp named "The moon", but for other users that could be a shorthand way of asking to play "Moonlight Sonata" [5]. The personalized index is made up of successful queries from the user within a thirty day time period, and is updated daily.

4.2 Retrieval Layer

Candidate rewrites for the query are scored using a neural network scoring function. If the potential rewrite scores well enough, it is added to an index of rewrite candidates. Using a system of query embedding, the query embedding is close to its related rewrites. A k-nearest-neighbor (kNN) index is used to quickly grab the top K rewrites. Here, K is all the rewrite candidates in the space. To promote the diversity of retrieved rewrite candidates, a query is passed on to a deep neural network (DNN) and a convolutional neural network (CNN) encoder.

The rewrite candidates are sorted by each encoder, and the results are interleaved. If a rewrite candidates is already in the list, the duplicate is removed. Once all candidates are interleaved, then starting from the top, we take the top K candidates, or in the case there are less than K candidates, the whole list. This improves coverage of candidates, as duplicates don't take up a spot on the list that a distinct rewrite candidate could take. These final candidates are given to the ranking layer. [5]

4.3 Ranking Layer

The ranking layer takes the query and the top K candidates from the retrieval layer. The ranking layer has two main components: a neural feature extractor, and a Gradient Boosted Decision Tree (GBDT). This allows for flexibility to add more semantic information, gives the model extendability to add various features on a global and personal level, and helps with generalization of data.

The Neural Feature Extractor acts as a deep learning model that provides hidden embedding as features for GBDT. Through the neural feature extractor, a ranking score is determined for the rewrites. Each query has a positive rewrite and a negative rewrite, where the positive rewrite will have a higher ranking score than the negative rewrite. Once the neural feature extractor model is trained, the features are extracted to be used within the GBDT models.

Along with the features extracted from the neural feature extractor, a group of conventional IR features are also extracted. All features are then concatenated together to form an input to the GBDT model. The GBDT model ranks the rewrites by which helps to reduce user friction the most, the most helpful ones going to the top. [5]

4.4 Rewrite Selection

For a global UFS-QR search, the top one returned rewrite from the ranking layer is used and replaces the original query in the conversational AI system.

For a personalized UFS-QR search, the goal is to find the most similar query from the individual user's personalized index. This is done by using the retrieval and ranking model, selecting the top one rewrite. Once both the global and personalized UFS-QR return a rewrite candidate, the personalized rewrite candidate is prioritized, to provide any personalization of query rewriting. [5]

4.5 Experimentation & Results

The training data used to train the model is from actual user data. The data picked was held to the following criteria: given two consecutive queries, the queries were spoken within a few seconds of each other (they were part of the same conversation), and using a friction detection model, the first query led to friction and the second was successful. The aim of these criteria is to grab query-rephrase pairs where the user repeated or rephrased the original query. They also used the ASR *n*-best, where the first query's *n*-best queries include the second query's ASR 1-best. The reasoning behind this is that while there may be an ASR error in the first query's 1-best, the correct query that the user intended is often in the first query's ASR *n*-best list. Looking at this helps find the rephrase pairs where the first query was likely an ASR error of the second query. User data was analyzed, and the data that was used for training was forty million query-rephrase pairs. Some training data examples are *play ambient mean -> play envy me*, and *play blues radio news -> play blue news radio*. In the first example, the rewrite corrected an ASR error, and in the second rewrite, the query was rearranged for better clarity. [5] They then obtained positive and negative rewrites of the training data. Positive rewrites are where the NLU hypothesis of the reference rewrite matches the NLU hypothesis of a proposed rewrite, otherwise it is deemed a negative rewrite. Correct rewrites are positive examples, negative rewrites are negative examples. From the forty million pairs, about twenty million triplets of "<query, positive_sample, negative_sample>" are sampled to train the ranking models. [5]

For an offline evaluation, test data was created that had a similar procedure as that of getting data for training. Immediately following the time period for grabbing training data, test data cases are randomly grabbed within the next time period. Human annotators select the true rephrase pairs from the data. The test cases come from the baseline model, Markov chain model, and were specifically cases where the Markov chain model was not able to create rewrites for that case. For the UFS-QR system, potential rewrite pairs are selected that fulfill the following conditions: The second utterance follows the first utterance by a short time window, and the second utterance can be observed to have a successful interaction based off the determination of the friction estimated model. Evaluation is performed on the NLU hypothesis level, using Precision@N (P@N), which measures if at least one rewrite in the first N candidates has a NLU hypothesis matched to the second utterance.

For the online analysis, the UFS-QR, both personalized and global, is launched through and A/B testing setup, and the defect rate decrease is measured.

Performance is in terms of improvement compared to the baseline. For offline experimentation, the Markov chain model is considered the baseline, and performance is rated at 0%. Introducing the retrieval layer in UFS-QR received a relative improvement of 15.1% at P@1 and 25.3% at P@10. Inserting the retrieved candidates into the ranking neural model received a relative improvement of 36.2%. Implementing the full UFS-QR system, there was a significant relative improvement at P@1 of 142.2%. The significant boost is attributed to the GBDT model, where the combination of semantic comparison captured by the ranking neural models and the feedback signal carried in the IR features where responsible for being able to connect queries to the correct rewrite, even when they were syntactically distant. [5]

The global and personalized UFS-QR were also tested against each other. The global UFS-QR system is set as the baseline, with P@1 performance reported at 0%. Personalized test data contains utterances that are specific to the individual, i.e. utterances that regard the users specific devices. Global UFS-QR rarely has confident rewrites for these test cases. Personalized UFS-QR improved the global UFS-QR by a relative 64% at P@1. [5]

The online evaluation was done over a week, with English speaking users in the US. Comparing the performance of global UFS-QR rewrites against no rewrites, the defect rate was reduced to 13%. Launching the personalized UFS-QR on top of the global UFS-QR resulted in the defect rate being reduced by 4%. The personalized UFS-QR also reduced user rephrases by 4.33%. When both global and personalized UFS-QR systems were fully launched in production, the total number of accurate rewrites was increased by 46%.

5 Compare and Contrast

The Markov chain model reported significant improvements when actual Amazon Alexa users used their system compared to a more commonly used pointer-generator model. This model rarely failed to produce a correct response for the user, and when it did it was from general user queries or because of loss of information from reiterating the same query multiple times. The Search-Based query rewrite system also showed to largely improve user satisfaction with the user's Alexa product, and also had the benefit of being able to produce correct responses to queries the Markov model missed.

References

[1] IBM. [n. d.]. *What are neural networks?* https://www.ibm.com/topics/neural-networks

[2] IBM. [n. d.]. *What is deep learning?* https://www.ibm.com/topics/deep-learning

[3] Victor Powell. [n. d.]. *Markov Chains Explained Visually.* https://setosa.io/ev/markov-chains/

[4] Yi Yi Benjamin Yao Chenlei Guo Ruhi Sarikaya Pragaash Ponnusamy, Alizera Roshan Gias. 2022. Feedback-based self-learning in large-scale conversational AI agents. *AI Magazine* 42 (January 2022), 14 pages. Issue 4. https://doi.org/10.1609/aaai.12025

[5] Xiaojiang Huang Xing Fan, Eunah Cho and Chenlei Guo. 2021. Search based Self-Learning Query Rewrite System in Conversational AI. *2nd International Workshop on Data-Efficient Machine Learning (DeMal)* (August 2021), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn