Neural Network based user input recommender for retrieval-based conversational interfaces

Luckyson Khaidem
Amadeus Software Labs
Bangalore, India
luckyson.khaidem@amadeus.com

Nitin Gupta
Amadeus Software Labs
Bangalore, India
nitin.gupta@amadeus.com

Hari Bhaskar Sankaranarayanan
Amadeus Software Labs
Bangalore, India
hari.sankaranarayanan@amadeus.com

***Abstract*—This paper discusses the quality issues faced by retrieval-based chat-bots and proposes a solution to improve conversational quality and overall satisfaction for end users. The problem with most chat-bots comes from the inability of the underlying intent classifier to correctly classify every user input text. Since all retrieval-based chat-bots are trained on a limited number of intents, it fails on open-ended conversations. As a result, the best way for such bots to succeed is to maintain the conversational focus around the areas in which the bot understands. The proposed solution revolves around the provision of guidance at every step of the conversation. The guidance is in the form of a recommendation model that attempts to guess the intention of a user as he types first few words by suggesting a list of relevant utterances. The solution makes use of similarity measurement between two pieces of text driven by multiple methods such as Siamese RNN and cosine similarity using word2vec embedding. The recommendation model prioritizes the relevancy of its suggestion by using the conversational context which is modeled as state transitions and past conversations. A conversational agent with the recommendation engine resulted in substantial increase in the average probability estimate of all intents correctly classified i.e. a more accurate conversational agent.**

Keywords—Siamese recurrent neural networks, word2vec, recommendation model, natural language processing

I. INTRODUCTION

Conversational agents or chatbots have become immensely popular in the research community and are widely used across web, mobile and enterprise applications. Well trained conversational agents can be used to interact intelligently with users to answer their questions. The potential that chatbots show is immense, however, most chatbots in production (which are mainly retrieval-based) falter because of many reasons. Use of language is filled with ambiguity and when talking to chatbot, users expect a level of understanding to that of human being which is not possible with present-day bots. Also, there's tendency of switching to multiple topics at the same time which is not easy for a bot to handle. Retrieval-based bots are generally trained on a limited set of training samples and work reliably if a user asks questions only around those training topics.

The paper proposes a model that aims to keep the conversational focus on the topics it understands through user input recommendations that are driven by neural models. Learning representation of linguistic items such as words and phrases in the form of floating point number vectors has become very effective approach [1] in most NLU tasks. Hence, the proposed model is based around neural models (such as word2vec, siamese recurrent network) that create the word or phrase representations for capturing similarities between user input phrases and samples within knowledge repository of conversational agents. We use distance metrics such as cosine distance and Manhattan distance to quantify these semantic similarities to generate the list of recommendations from the knowledge repository. Conversational state within the ongoing dialog and user's conversation history is used to prioritize these recommendations which are then provided as a means for guidance to users. We demonstrate this model by creating a generic travel agent chatbot application that answers travel related frequently asked questions.

The rest of the paper is organized as follows: we first give a brief summary of researches that have been done around conversational agents. The proposed model is described in section III. Experimental setup and results are provided in section IV And then we conclude our findings in section V.

II. RELATED WORKS

Conversational Agents or Dialogue systems interact with humans to provide useful information, answer queries or perform administrative tasks. Most conversational agents consist of a Natural Language Understanding module that processes user input text and extracts important information that is then sent to a Dialogue Manager that updates its internal state and performs some action based on a predefined business logic. In retrieval based chat-bots, predefined responses are retrieved from a knowledge repository based on the information (intent, contexts, entities) extracted by the NLU module from raw input text or using a simple rule-based expression matching. These systems do not generate new texts but simply picks from a fixed set.

Historically, there have been many attempts to improve NLU [2] and find a better representation of linguistic items. One of the earliest methods adopted was to identify keywords or a combination of them. This is the basis for script-based chatbots and systems like ELIZA. With the advancement of machine learning, there have been many developments ranging from statistical modeling of language [3], the creation of neural embeddings of linguistic items such as word2vec, doc2vec, and GloVE [4,5,6] to approaches using neural sequence models. Utilization of embeddings for answer retrieval using Locality Sensitive Hashing (LSH) Forest has also been explored [7]. A lot of effort has also gone into developing generative chatbots that build responses from scratch by treating it as a statistical machine translation task [8]. Such models have no explicit dialog structure. They are trained on human to human conversational corpora. Seq2seq models that use encoder-decoder recurrent neural

networks with attention mechanism have also been explored to generate responses in an end to end fashion [9]. However, generative chatbots are not matured enough to be deployed in production environments. Most chatbots in production are retrieval-based. Despite this, there are still a lot of quality issues that are still prevalent with retrieval-based conversational agents. Most of them come from the inability of the underlying intent classifier to correctly capture every variation of an expression.

III. Model Description

Most of deployed bots employ retreival-based models which use a repository of predefined responses to pick an appropriate response based on the input, like shown in Fig 1

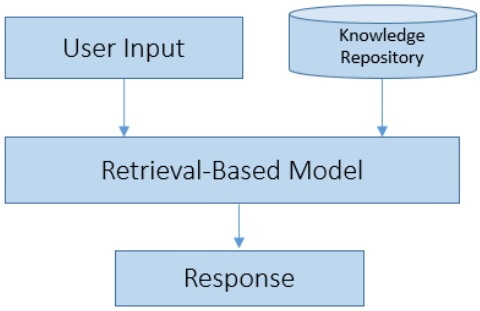

Fig. 1. Retreival-based Bots

Proposed model is a recommendation system that guesses the intention of the user by providing a list of utterances from the knowledge repository as the user starts typing the first three or more words in the chat box. This repository contains utterances and intents based on which the model is trained.

The model can be split into three main components. The first component is a similarity measurement unit which quantifies the semantic similarity between a user entered phrase and samples (utterances) within the knowledge repository. We tried two neural models to measure similarity; Manhattan Siamese Recurrent Network [10] and Skip Gram Word2vec model [4]. These neural models learn vector representation of phrases with which we can define similarity function to establish how similar two linguistic items are. The scores are used as a basis for filtering samples that are close to the user entered phrase in the semantic space. The second component is a simple module that performs prefix matching between the user entered phrases and samples from the repository. The third component maintains the current state of the conversation and a user's historical context. This component prioritizes the samples from the first and second component and combines them to form a comprehensive list of recommendations based on a logic that we have defined. The entire pipeline can be seen in figure 2.

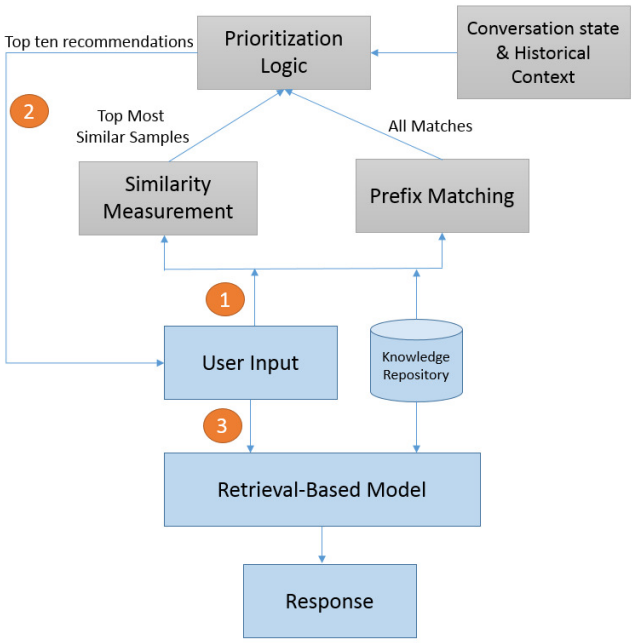

Fig. 2. Recommendation model pipeline

A. Similarity Measurement Unit

The first component quantifies the similarity of a user entered phrase against all the samples in the knowledge repository. It does this by mapping them to vector in a semantic space and calculating the similarity between these vectors by using distance metrics. Based on the similarity scores, we filter the top samples that are really close to the user entered phrase in the semantic space as candidates for recommendation. For creating this semantic space, we explored two neural models.

1) Manhattan Siamese Recurrent Neural Network

This model is based on the work carried out by *Mueller and Thyagarajan* [10]. The model contains two LSTM networks: $LSTM_a$ and $LSTM_b$. Each network processes a sentence in parallel. In Siamese architectures, the models in consideration share weights and hence $LSTM_a = LSTM_b$. The LSTM uses a sequence of word embedding to represent an input sentence and uses its final hidden state as a vector representation for each sentence. It learns a mapping from the space of variable length sequences of *din* - dimensional vectors into $\mathbb{R}^{drep}$. This means each sentence as a sequence of vectors $x_1, x_2, \ldots\ldots, x_T$ is fed to the LSTM. The training set is a collection of pairs of phrases or sentences labeled either as similar or non-similar. For example,

The dog is resting here. *The cat is sleeping there.* *1*

I am sleeping now. *I am going out for a walk.* *0*

The LSTM model produces vector representation for each sentence $ht \in \mathbb{R}^{drep}$. For a pair of sentence, Manhattan distance is calculated between their vector representations. If the learned vector representations for a sentence pair are ht^a and ht^b, then we define a similarity function G which uses the Manhattan distance of the vectors;

$$G(ht^a, ht^b) = \exp\left(\left|\left|ht^a - ht^b\right|\right|_1\right) \in [0,1] \quad (1)$$

On training, the model updates its weights to decrease *G* for dissimilar sentences and increase g for similar sentences. The model is trained on SNLI dataset with *drep* set to 60 (hidden units) and *relu* activation function resulting in an accuracy of 75% on 0.75 and 0.25 train-test split. The model is summarized in figure 2.

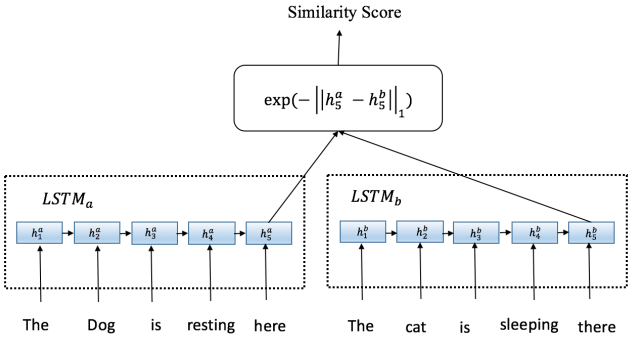

Fig. 3. Mahantann Siamese Recurrent Neural Network

2) Skip-gram word2vec

The skip-gram model is introduced in Mikolov et al. [4]. Word2vec uses a simple neural network with a single hidden layer which is trained to perform a certain task. However, the network is not used for the task it is trained for but rather the goal is to use update the weights in the hidden layer which is then used as the vector representation of words. The task is to predict context words given a target word. Let's take an example, *A quick brown fox jumped over the lazy fox*. We form context and target word pairs by taking a window size. For the sake of the simplicity, let's take a window size of 1 and a target word brown. It's context words are one word to it's left and one word to it's right. So the context-target pairs are

(brown, quick)
(brown, fox)

This way, we find context words for every word in the corpus for a selected window size. Then we train the shallow neural network to predict context words from target words. If the number of words in the vocabulary is *V* and we want to embed each word in a semantic space $\mathbb{R}^k$, then the input and output layer will have *V* neurons while the hidden layer will have *k* neurons. Hence, the input to hidden weight matrix is $W_1 \in \mathbb{R}^v \times \mathbb{R}^k$ and the hidden to output weight matrix is $W_2 \in \mathbb{R}^k \times \mathbb{R}^v$. Each word is represented as a one hot encoded vector $x \in \mathbb{R}^v$ where for the k^{th} word, $x_k = 1$ and $x_{k'} = 0$ for $k \neq k'$ then, we have

$$h = W_1^T x = W_{1\,(k,)}^T := V_{wk}^T \quad (2)$$

We simply copy the k^{th} row of W_1 to *h*. V_{wk} is the vector representation of the input word wk. And then we use the weight matrix W_2 from hidden to output to calculate the score u_j for each word in the vocabulary.

$$u_j = {V_{wj}^2}^T h \quad (3)$$

Here ${V_{wj}^2}^T h$ is the j-th column of the matrix W_2, then we can use softmax function to obtain a posterior distribution of words, which is a multinomial distribution.

$$p(w_c|w_t) = \frac{\exp(u_c)}{\sum_{j=1}^{V} \exp(u_j)} \quad (4)$$

In this paper, we use Google's pre-trained word2vec which contains word vectors of 3 unique million words and phrases that they train on 100 billion words of Google news dataset. Given a sentence, we find its representation by simply finding the mean of the vector representation of its constituent words. If a sentence of *n* words is $W = \{w^1, w^2, \ldots\ldots, w^n\}$ where w^i is the i^{th} word in the sentence and $v^i \in \mathbb{R}^k$ is the word vector of the word w^i, then the representation of the sentence $s \in \mathbb{R}^k$ can be calculated as

$$s_i = \frac{\sum_{j=1}^{n} v_j^i}{n} \quad (5)$$

Given two sentence vectors s^a and s^b, we quantify their similarity by finding the cosine of the angle between the two vectors.

$$\cos\theta = \frac{s^a . s^b}{||s^a||.||s^b||} \quad (6)$$

The assumption is that higher value of the cosine of the angle between the sentence vectors, the more similar are the two sentences.

B. Prefix Matching

This is a simple component that filters every sample from the repository that has the user entered phrase as their prefix. We only use this filter if the the number of words in the user entered phrase is more than three. This is due to the fact that samples have more relevancy if number of words in their prefix being matched is of sufficient number. For example,

User entered phrase

I have lost my

Matched samples

I have lost my *baggage*

I have lost my *itinerary. What should I do?*

I have lost my *travel documents.*

These samples are sent down the pipeline to a prioritization logic such that more relevant samples appear first in the list.

C. Conversational State and Historical Context

We represent conversational flows as states and transitions. Each state is associated with a set of intents. For example, in travel agent chatbot application, a state "Send itinerary" can be associated with a set of intents in chatbot application. Another example can be a state "baggage" where a user can talk about topics ranging from baggage allowance to baggage loss. A conversation has a finite number of such states with transitions between them. It can be thought of as a directed graph where each state is a node and the transitions between them as the edges.

For all users, we maintain a history of their conversation and define the probability of a user going to a particular state. If in a chat-bot application, there are *n* states $\{S^i, S^2, \dots \dots, S^n\}$, then we calculate $p(S^i)\ \forall\ i \in [1, n]$. Calculating this probability is simple. A user may have multiple sessions within the applications at different points in time with each session having its own unique sequence of state transitions. Let us assume that a particular session has the state sequence $S^1, S^2, S^2, S^2, S^3, S^3, S^1, S^4, S^5$. If there are *m* sessions, there will be *m* such state sequences. Then,

$$p(S^i) = \frac{number\ of\ times\ S^i\ occurs\ across\ all\ sessions}{total\ number\ of\ states\ across\ all\ session} \quad (7)$$

These probability scores along with the current conversational state are used to prioritize the list of recommendations from the first two components. The idea behind this is to adjust the relevancy of the recommendations with the most relevant recommendation appearing first in the list and so on.

D. Prioritization

From the similarity measurement unit, we have a list of samples that are sorted based on their similarity scores measured against the user entered phrase. Each sample or utterance is associated with a state because using this utterance as input will lead to a transition into that state. The idea is to use the probability of each state to adjust the similarity scores.

The recommendations are a list of 3-tuple $< r^i, S^i, \psi^i >$ where r^i is the sample from the repository, S^i is the state associated with the sample, and ψi is the similarity score measured against the user entered phrase. We adjust the score ψ^i by using $p(S^i)$. States which are more likely to occur should have their input samples higher up in the relevancy in the recommendation list. To capture this idea, we define weighing factor

$$\Omega^i = \left(1\text{-}C(S^i)\right).\left(a + (b\text{-}a).p(S^i)\right) + C(S^i).b \quad (8)$$

where $b > a \geq 1$ and

$$C(S^i) = \begin{cases} 1, if\ S^i\ is\ the\ current\ state \\ 0\ , \quad if\ S^i is\ any\ other\ state \end{cases} \quad (9)$$

Using (8) and (9), we arrive to an adjusted score

$$\sigma^i = \Omega^i.\psi^i \quad (10)$$

We then sort the recommendations r^i based on the decreasing order of their adjusted score σ^i . (8) linearly shifts $p(S^i)$ between a and b thereby boosting similarity scores for samples with higher state probability. In our experimented. we set a = 1.0 and b = 1.2. We then selected the top 5 samples based on their adjusted score which is combined with 5 samples from the prefix matched samples. If the number of matched words in the prefix matched samples is more than five, then they precede the samples from the similarity measurement unit. Otherwise, the similar samples appear first in the recommendations.

IV. Expreiemental Setup and Results

To test the outcome of our model, we created a travel agent chat-bot application that answers frequently asked travel related queries such as baggage information, hotel bookings, payment queries, cancellations etc.

A. Travel agent chat-bot

We used Google's dialogflow to create the conversational agent that performs tasks such as intent classification, named entity recognition and context management. We created 95 intents and a sufficient number of sample utterances for each of them. We outline some of the intents and the sample utterances (non-exhaustive) for each of them in Table I below. This serves as the knowledge repository in our application. The dialog flow agent is trained with the training samples that we hand-created with the classification threshold set to 0.35. This means, given a user input, if the highest probability in the probability distribution across all intents is less than the threshold, the agent falls back to a default response because it is not able to classify an intent with the minimum confidence level.

TABLE I. KNOWLEDGE REPOSITORY

State	Intent	Samples
Start	Greeting	Hello, Hi,.....
Baggage	Baggage loss	I have lost my baggage, I lost my baggage while travelling with xyz airline,.....
Baggage	Baggage allowance	How much baggage can I carry with me?,...
Hotel	Change hotel booking	I want to change my hotel booking,...
Hotel	Cancel hotel booking	I want to cancel my hotel booking,...
Payment	Credit card	Can I pay with my credit card?,...
Payment	Security	Is my credit card information safe?,...

Cancellation	Refund on cancellation	Am I eligible to a refund if I cancel my flight?,...
Cancellation	Cancellation procedure	I have booked a ticket through your agency but I would like to cancel it now. How do i go about doing this?

Like most conversational frameworks, dialogflow's limitations lie in the inability of its underlying intent classification model to capture every variation of a sample utterance resulting in misclassifications and fallbacks. However, this is not to say that the agent is not able to handle all out of sample variation of the utterances in the knowledge repository. There are also cases where the agent correctly classifies these variations. Table II outlines some examples of misclassification that was discovered when testing the agent.

TABLE II. MISCLASSICATION EXAMPLES

Input utterance	**Correct intent**	**Predicted intent**
The airline I flew with has misplaced my baggage	Baggage loss	Fallback
I would like to cancel my flight. Will i get my money back?	Refund on cancellation	Cancellation procedure
What is the maximum weight of baggage I can carry with me?	Baggage allowance	Fallback

It is clear that there are already flaws in the agent. Next, we created a simple web UI that contains chat box that houses all the messages exchanged between user and the bot. A text box is placed at the bottom of chat box where user can enter their messages and send to agent by pressing the submit button located adjacent to text box. An image of the UI is shown in Figure 3.

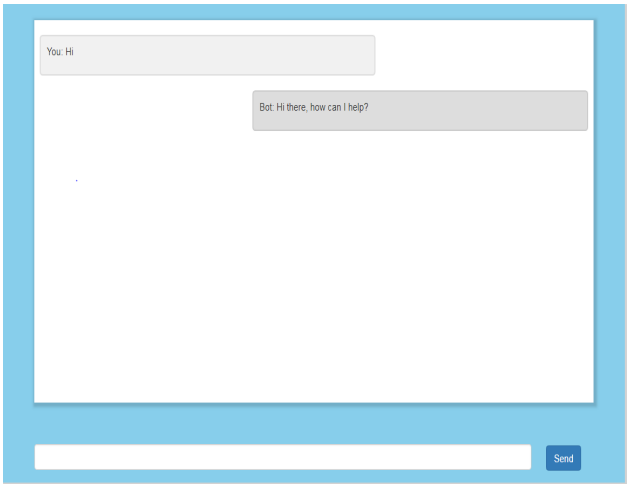

Fig. 4. Chatbot web UI

On submission, we make an API call to the dialogflow agent that we created using the user message as one of the parameters for the API call. The API returns among many things, the predicted intent and the bot response. The response is then populated in the chat box. This version of the chatbot without the recommendation model was deployed in a private network with several users for a certain period of time. The aim was to log state transitions within a conversational session and probability estimates of true intents for every user messages.

Then we introduced our recommendation model between the web client and the dialogflow agent. The recommendation model is decoupled from both the web client and the agent and is hosted as a separate service. The web client makes an asynchronous call to the recommendation API before sending the message to the dialogflow agent. As the user starts typing and crosses a minimum of three words, the web client triggers an ajax call for every additional word. The ajax call sends a request to the recommendation model with the user entered phrase as a parameter and retrieves a list of recommendations from the knowledge repository. The recommendations are populated in the web UI as shown in figure 4. A user can either click on one of the recommendations and send the message to the agent or type a complete message and hit the submit button. We created two versions of the recommendation model: one using Manhattan Siamese Network and the other using mean of word2vec representations for sentence similarity measurement. We tested them both separately to compare the quality of their recommendations.

B. Results

Measuring and evaluating the quality of recommendation models offline is a difficult task. An ideal approach to test our model is to deploy it in scale and keep track of the click rates on the recommendations. However, we hosted our application in a private network with a number of users testing it.

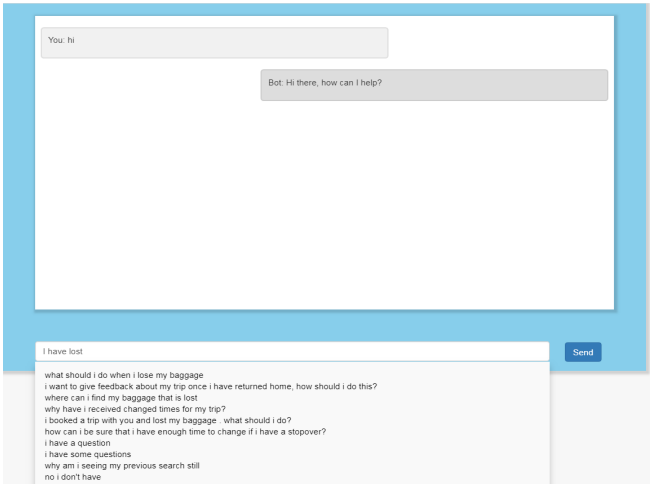

Fig. 5. Recommendations populated on the UI

We recorded the number of messages exchanged, misclassifications, fallbacks and number of clicks on recommendations. Tables III, IV and V outline the results from chatbot without the recommendation model, with the recommendation model using Siamese Network and with word2vec averaging respectively. It is evident from the results that versions of the chatbot with the recommendation model performs better with lesser misclassifications and

better user engagement. This means that user engages better with more intuitive recommendations.

TABLE III. WITHOUT THE RECOMMENDATION MODEL

Total messages sent	600
Total misclassifications	350
Total fallbacks	100
Error Rate %	58. 3 %

TABLE IV. SIAMESE NETWORK RECOMMENDATION MODEL

Total messages sent	650
Total misclassifications	295
Total fallbacks	90
Recommendation clicks	180
Recommendation clicks %	27.6 %
Error Rate %	45.3 %

It is important to note that a large number of fallbacks does not necessarily imply that a chatbot has quality issues. It is a desirable behavior to fallback when it receives a message it was not trained to answer rather than misclassifying it.

TABLE V. WORD2VEC AVERAGING RECOMMENDATION MODEL

Total messages sent	705
Total misclassifications	180
Total fallbacks	88
Recommendation clicks	356
Recommendation clicks %	50.4 %
Error Rate %	25.5 %

Recommendation clicks is the number of times users have chosen a sample from the recommendation instead of typing an entire message. The recommendation model with average word2vec representation of phrases performed better with only 25.5% error rate as compared to 45.3% misclassification by the recommendation model that uses siamese network for generating similarity scores. Almost 50% of the time, the test users clicked on one of the recommendations instead of typing out complete sentences when interacting with the model with word2vec averaging. What it means is that averaging word2vec in phrases creates more accurate semantic spaces for similarity measurements as proven by the following examples. Figure 5 shows the results obtained from both models.

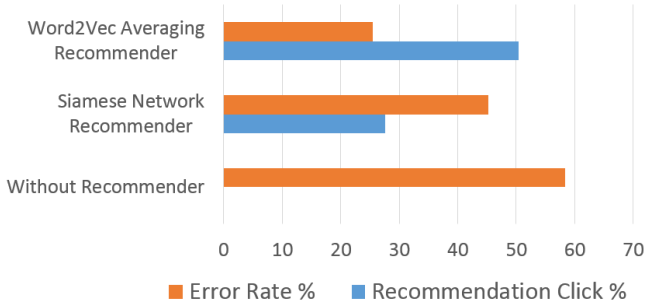

Fig. 6. Error Rates for Recommenders

The similarity measurement with word2vec averaging fetches recommendations that are more relevant to the context provided by the user entered phrases when compared to the ones that are retrieved with Siamese network. This could be due to the fact that the Siamese network is trained to compare a pair of complete sentences rather than phrases. Below shown are three examples tried on both recommenders

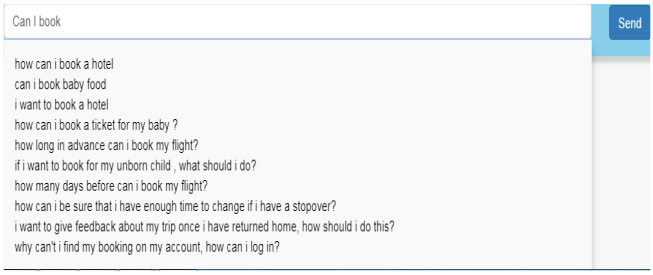

Fig. 7. Word2vec averaging - Example 1

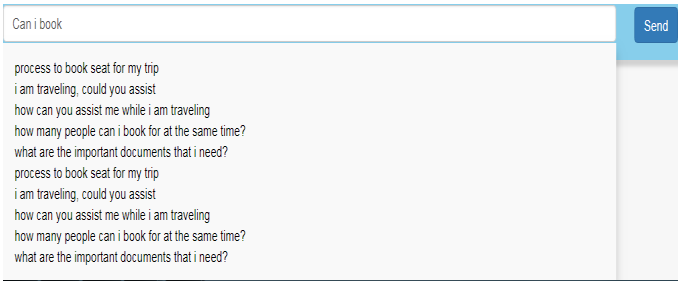

Fig. 8. Siamese network - Example 1

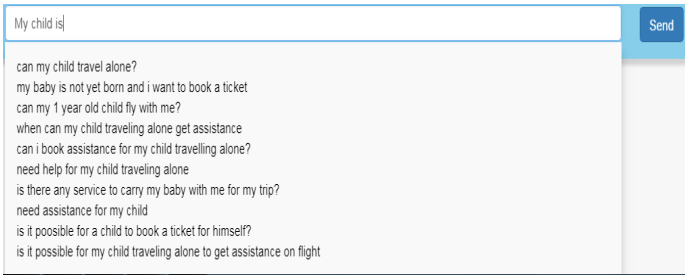

Fig. 9. Word2vec averaging - Example 2

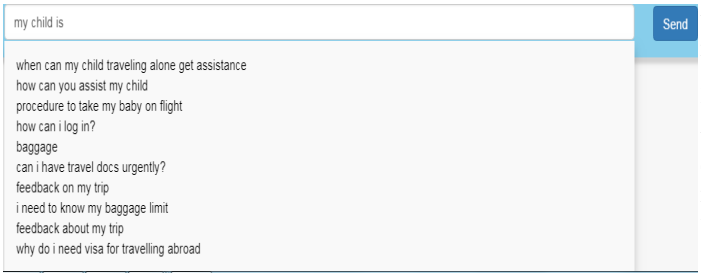

Fig. 10. Siamese network - Example 2

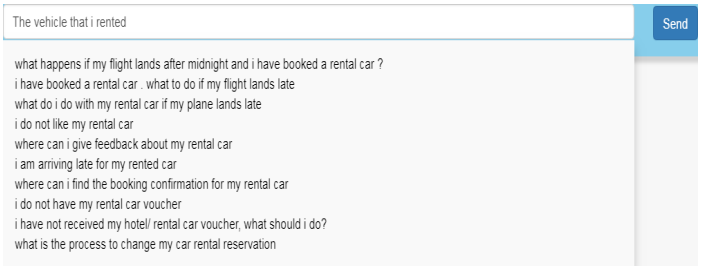

Fig. 11. Word2vec averaging - Example 3

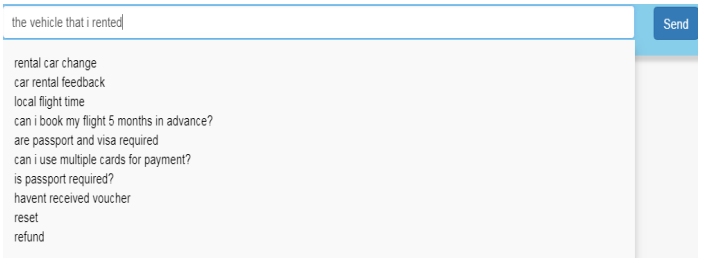

Fig. 12. Siamese network - Example 3

V. Conclusion

Most conversational agents are retrieval-based and are trained on limited number of intents and samples. They fail on open-ended conversations because most users are not aware that knowledge of the bots is limited. Most conversational agents are also incapable of understanding every variation of a sample utterance. Due to this, users are more likely drop out of conversations early on. To make chatbots more engaging and intuitive, it is imperative to have a recommendation model that suggests sample utterances that the bot understands. The proposed recommendation model - word2vec averaging with prioritizations has shown good performance. Relevant recommendations are retrieved with even very little context in the small phrases provided by the user. However, testing the model was a challenge. More exhaustive testing on larger corpus will help improve the model. This is can be done by deploying the model in scale with thousands of users interacting with it.

References

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Advances in neural information processing systems, 2013,

[2] M. Bates, "Models of natural language understanding," in Proceedings of the National Academy of Sciences of the United States of America, vol. 92. National Academy Press, 1995, pp. 9977–9982.

[3] C. D. Manning and H. Schutze, "Foundations of statistical natural language processing," The MIT Press, 1999.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space". ICLR Workshop, 2013.

[5] Quoc V. Le and Tomas Mikolov, "Distributed Representations of Sentences and Documents", ICML'14 Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 Pages II-1188-II-1196, 2014.

[6] Jeffrey Pennington, Richard Socher and Christopher D. Manning, "GloVe: Global Vectors for Word Representation", Proceedings of the Empiricial Methods in Natural Language Processing(EMNLP 2014) 12.

[7] Alexander Bartl and Gerasimos Spanakis, "A retrieval-based dialogue system utilizing utterance and context embeddings", 16th IEEE international conference on Machine Learning and Applications, 2017

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

[9] Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", ICLR 2015.

[10] Jonas Mueller and Aditya Thyagarajan, "Siamese Recurrent Architectures for Learning Sentence Similarity", In Proceedings of the 13th AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. 2786–2792