nBot

Nalin Chhibber
Student ID: 20715659, MMath CS(HCI)

Abstract

The primary aim of this paper is to describe popular techniques that researchers have been using to build conversational agents and cover a sizable body of literature related to the area. Additionally, this paper replicates the results by (Lowe et al. 2015) for Unstructured Multi-Turn Dialogue Systems. Based on the findings, the paper also introduces a new conversational agent: nBot. nBot is trained on a limited dataset of manually constructed sentences and built with a combination of both retrieval-based and generative techniques. It uses Long Short Term Memory to remember the context of conversation.

Introduction

As we are gradually incorporating technology in our lives, we are also trying to simplify the interaction technique by getting rid of unused mediums from the space of complicated interfaces. Taking the example of a simple phone, we started using them with rotary dials, then replaced those with numeric buttons and moved on to using touch screens. Voice user interaction is the next logical step and we have already started seeing a glimpse of it in Siri and Google Assistant. While most of the existing modes of human-computer interaction (touch/type/click) require users to adapt with the interface(screen/keyboard/mouse), voice user interfaces on the other hand, comes natural to humans and hence can be used as one of the most promising medium to engage them in a productive interaction. There has been a lot of optimism in the thought that near future will witness a rapid growth in human-computer-interaction using voice. This has not only led to an increased demand of conversational agents but also shown an increase in various chatbot development frameworks. Brands are increasingly using chatbots to engage their customers. Within just a couple years, we have seen a different evolution in the design of conversational agents from chatbots in Facebook Messenger, to Siri in iphones, to Microsoft's Cortana, Google Home and Amazon Alexa. While we cannot predict when Watson or Siri will start working like Jarvis, there is a still a lot that we can do with the normal scripted conversational agents.

This paper has implemented a simple conversational agent, nBot using a combination of retrieval-based and generative models. It uses a recurrent neural network with 4 hidden layers of LSTM cells to understand the context and respond to user utterances. It has been trained on a limited dataset of manually constructed conversations over 5000 epochs.

Rest of the paper describes taxonomy of models and related work around the design of conversational bots. It then details the implementation of nBot, followed by discussions on future work.

Taxonomy of models

Retrieval based vs Generative vs Pattern based

Challenge of building a conversational bot can be addressed using a retrival-based, generative or pattern-based model. Retrieval-based models are those which have a repository of pre-defined responses to answer a user utterance. Alicebot and Cleverbot are examples of such types. On the other hand, generative models are those which can generate responses they have never seen before. Microsoft's Tay bot is an example of generative model. Generative models are usually based on machine translation techniques but instead of translating from one language to another, they translate from input uttrance to output response. Both approaches have some obvious pros and cons. Retrieval based techniques dont make gramatical mistakes and are easier to train due to repository of handcrafted rules. However they may be unable to handle unseen cases for which no appropriate response exists. For the same reason these models cant refer back to contextual entity information. Generative models overcome this limitation and can refer back to entities in the input. However these models are harder to train, likely to make gramatical mistakes, produce inconsitant or irrelevant responses and require huge amounts of training data.

Another technique is to use pattern-based heuristics for selecting a response. In these techniques, when the agent receives a message, it goes through all the patterns until it finds a pattern which matches user utterance. If the match is found, the chatbot uses the corresponding template to gen-

erate a response. The problem with pattern-based heuristics is that patterns should be programmed manually which is not easy, especially if the agent has to correctly distinguish hundreds of intents. Users can express the same intent in many ways or use similar words in different contexts. Machine learning can help us train an intent classificaltion algorithm to pick patterns in the data. Most of the existing chatbot frameworks(wit.ai, dialogflow, Microsoft LUIS) are based on this technique.

Open-domain vs Close-domain

In open-domain, conversations dont have a well defined goal or intention and can jump to any topic from dating to space travel to politics. Conversations on social media sites like Reddit and Twitter are typically open domain. In closed domain, space of possible input utterance is limited to a specific topic(education, casual conversation, technical support). It is realtively easy to model a conversational agent in close domain. Figure 1 illustrates relative difficulity in designing chatbots with different techniques.

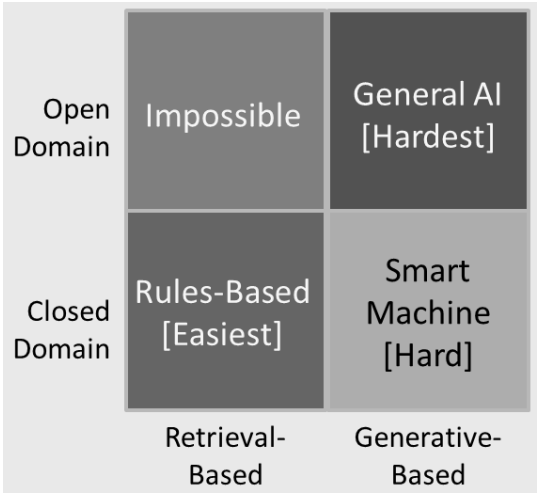


Figure 1: Chatbot Conversation Framework(Kojouharov 2016)

Online vs Batch Learning

Batch learning generates the best predictor by learning on the entire training data set at once. Online learning on the other hand uses new responses to update the best predictor for future responses at each step. Online learning can dynamically adapt to new patterns in the conversation as it is generated as a function of time. One problem with online learning is to control unwanted or potentially harmful information. One significant instance is when Microsoft's Twitter chatbot: Tay, started posting a deluge of incredibly racist messages in response to questions. Tay was designed to learn from conversations and get progressively "smarter" but online troublemakes and trolls persuaded it to blithely use racial slurs and even outright call for genocide (TechCrunch 2016). Therefore online learning should be assisted by other techniques to prevent them from going off the rails.

End-to-end vs Distributed Learning

Distributed learning systems require multiple stages of processing and follow a set pipeline for solving tasks at hand from feature extraction to learning the desired result. End-to-End learning on the other hand omits any hand-crafted intermediatery algorithms and directly learns the solution to a given problem from sampled dataset. This could involve concatenation of different neural networks(CNNs, RNNs) which are trained simultaneously. The idea is to let the network go from"raw-est" possible data to "final-most" output.

Related work

Typical dialogue systems consist of a Speech Recognizer, Language Interpreter, State Tracker, Response Generator, Natural Language Generator, and Speech Synthesizer. They can further be distinguished as goal driven systems, or non-goal driven systems. Initial work on goal driven dialogue systems primarily used rule-based systems with the distinction that machine learning techniques were used to classify the intention (or need) of the user, as well as to bridge the gap between text and speech(Serban et al. 2015). Research in this area started to take off during the mid 90s, when researchers began to formulate dialogue as a sequential decision making problem based on Markov decision processes. (Young et al. 2013) (Singh et al. 2000) (Pieraccini et al. 2009). With these systems, each of the components were trained separately to do their own task and the chatbot used to collectively use results from each to give the response. Later, with the improvements in deep learning algorithms, conversational models started using an end-to-end approach and use Neural Networks to predict the output of given input utterance. Though simple Feedforward Neural Networks with one hideen layer are regarded as universal function approximators, they are not well suited to remember timeseries data and thus cannot refer the information from past conversation. A Recurrent Neural Network (RNN) solves this problem by looping back part of the output back into the network, allowing information to persist. RNNs reads the input sentence, one token at a time, and predicts the output sequence, one token at a time. However, even RNNs are not too good in remembering long term dependencies and tend to forget the context if the gap between the relevant information and the point where it is needed becomes very large. Long Short Term Memory networks or LSTM are special types of RNNs and they are capable of learning long-term dependencies (Hochreiter and Schmidhuber 1997a). Besides the neural network itself, another thing that determines the performance of a conversational agent is the language model which gives the probability distribution over a sequences of words. This language model is based on the input to the neural network which involves elements of Natural Language Processing.

Traditional NLP models typically use word as an atomic unit. However, words are discrete in nature and it seems nonsensical to feed word indexes to Neural Networks. A typical approach is to map a discrete word to a dense, low-dimensional, real-valued vector, called an embedding.

A popular word embedding model is word2vec which uses a distributed representation of word to capture meaningful syntactic and semantic regularities (Mikolov et al. 2013a) (Mikolov et al. 2013b) (Mikolov, Yih, and Zweig 2013). In (Le and Mikolov 2014), authors extends the same idea to also compute distributed representations for sentences, paragraphs, and even entire documents. These vector representations are used in building the neural language model and help RNNs learn the context over time, given a word. We can choose how large these vectors should be(nBot uses vectors of size 300 dimensions). Both the embedded context and response are fed into the same Recurrent Neural Network word-by-word. The RNN generates a vector representation that, loosely speaking, captures the "meaning" of the context and response, and overtime learns to output the correct message. Word Embedding is typically done in the first layer of the Neural Network : Embedding layer, that maps a word (index to word in vocabulary) from vocabulary to a dense vector of given size. Alternatively, it is also possible to use a pre-trained word embeddings model with existing structure of Recurrent Neural Network.

Some other deep Learning architectures like Sequence to Sequence(Sutskever, Vinyals, and Le 2014) are uniquely suited for generating text and researchers are hoping to make rapid progress in this area. Though nBot does not use sequence-to-sequence framework, it is important to mention its relevance in this paper given its performance and acceptability amongst AI researchers. Sequence-to-sequence framework consists of two RNNs (Recurrent Neural Network) : an Encoder and a Decoder. The encoder takes a sequence(sentence) as input and processes one symbol(word) at each timestep. Its objective is to convert a sequence of symbols into a fixed size feature vector that encodes only the important information in the sequence while losing the unnecessary information. (Vinyals and Le 2015) have used sequence-to-sequence framework to build a Neural Conversational Model(NCM). They trained their model on both a domain specific IT helpdesk dataset, and a noisy open-domain movie transcript dataset. Sample conversations generated by the interaction between human actor and the NCM reports lower perplexity as compared to n-grams model and even outperformed CleverBot in a subjective test involving human evaluators to grade the two systems. However, as with any generative model, primary limitation of NCM is short and at times inconsistent responses. Moreover, the objective function of sequence-to-sequence framework is not designed to capture the objective of conversational models(though the underlying architecture can be leveraged for machine translation, and question answering applications).

Rest of this section is focused on discussing "The Ubuntu Dialogue Corpus" paper and replicating their results. Ubuntu Dialog Corpus is a dataset containing almost 1 million multi-turn dialogues, with a total of over 7 million utterances and 100 million words (Lowe et al. 2015). The dataset has both the multi-turn property of conversations in the Dialog State Tracking Challenge datasets, and the unstructured nature of interactions from microblog services such as Twitter. To justify the usefulness of proposed dataset for research in neural architectures for dialogue managers, authors provided a performance benchmarks for two neural learning algorithms(RNNs and LSTMs), as well as one naive baseline(TF-IDF). Prior to applying each method, they performed standard pre-processing of the data by tokenizing, stemmimg, and lemmatizing the utterances using NLTK. They also used generic tags for various word categories(like names, locations, organizations, urls). Additionally, authors set aside 2% of the Ubuntu Dialogue Corpus conversations (randomly selected) to form a test set that can be used for evaluation of response selection algorithms. Compared to the rest of the corpus, this test set has been further processed to extract a pair of (context, response, flag) triples from each dialogue. The *flag* is a Boolean variable indicating whether or not the response was the actual next utterance after the given context, *response* is a target (output) utterance which we aim to correctly identify, and finally *context* consists of the sequence of utterances appearing in dialogue prior to the response. In order to evaluate each of the three methods, authors used Recall@k metric. Here the agent is asked to select the k most likely responses, and test example is marked correct if the true response is among these k candidates. So a larger k means that the task becomes easier. In a 1-in-10 utterance classification, if we set k=10 we get a recall of 100% because we only have 10 responses to pick from. If we set k=1 the model has only one chance to pick the right response.

TF-IDF: Term frequency-inverse document frequency is a statistic that intends to capture how important a given word is to some document, which in our case is the context (Ramos and others 2003). It is a technique often used in document classification and information retrieval. The 'term-frequency' term is simply a count of the number of times a word appears in a given context, while the inverse document frequency term puts a penalty on how often the word appears elsewhere in the corpus. The final score is calculated as the product of these two terms, and has the form:

$$tfidf(w, d, D) = f(w, d) * \log \frac{N}{|d \in D : w \in d|}$$

where f(w, d) indicates the number of times word w appeared in context d, N is the total number of dialogues, and the denominator represents the number of dialogues in which the word w appears. For classification, the TF-IDF vectors are first calculated for the context and each of the candidate responses. Given a set of candidate response vectors, the one with the highest cosine similarity to the context vector is selected as the output. For Recall@k, the top k responses are returned.

RNNs: Recurrent neural networks are a variant of neural networks that allows for time-delayed directed cycles between units(Medsker and Jain 1999). This leads to the formation of an internal state of the network, h_t , which allows it to model time-dependent data. The internal state is updated at each time step as some function of the observed

variables x_t , and the hidden state at the previous time step h_{t-1}. W_x and W_h are matrices associated with the input and hidden state.

$$h_t = f(W_h W_{t-1} + W_x x_t)$$

LSTM: In addition to the RNN model, original authors considered the same architecture but changed hidden units to long-short term memory (LSTM) units(Hochreiter and Schmidhuber 1997b). LSTMs were introduced in order to model longerterm dependencies. This is accomplished using a series of gates that determine whether a new input should be remembered, forgotten (and the old value retained), or used as output. The error signal can now be fed back indefinitely into the gates of the LSTM unit. This helps overcome the vanishing and exploding gradient problems in standard RNNs, where the error gradients would otherwise decrease or increase at an exponential rate. In training, we used 1 hidden layer with 200 neurons.

Following tables illustrate the perfomance for above three models based on Recall@k metric from original paper and replicated results wrt TF-IDF and LSTM. For comparison, completely random predictor got a score of 9.3%, 19.4% and 49.2% for Recall@1, Recall@2 and Recall@5 respectively.

Method	TF-IDF	RNN	LSTM
Recall@1	41.0%	40.3%	60.4%
Recall@2	54.5%	54.7%	74.5%
Recall@5	70.8%	81.9%	92.6%

Table 1: Results for the three algorithms using various recall measures

Method	TF-IDF	LSTM
Recall@1	49.5%	49.84%
Recall@2	59.7%	68.74%
Recall@5	76.6%	91.44%

Table 2: Replicated results for the three algorithms using various recall measures after 2440 steps

The replicated model gives following response after training. Spelling mistakes are due to the quality of the corpus.

[Context] be i abl to use dhcpd toissu dhcp to all subnet except the one it be in ? __eou__ __eot__ great question ! sound implaus but i ll check __eou__ __eot__ thank . i ve be read the configur guid and ca n't find if i can or not . i could off cours tri . __eou__ __eot__ the way i read it , for a multi-hom server , it ll onli server sub-net for which there be a subnet claus in the config __eou__ __eot__
[Answer] so i would put a shared-network group with the subnet i want in there , exclud the one it be attach to ? __eou__

The AI and Deep learning blog by (Britz 2016) helped a lot while replicating the results of original authors. Inside the recurrent neural network, input context(c) is multiplied with a matrix M to "predict" a response r'. Input context(c) is a 300-dimensional vector, M is a 300*300 dimensional matrix, and the result is another 300-dimensional vector, which can be interpreted as a generated response. The matrix M is learned during training. The model then measures similarity of the predicted response r' and the actual response r by taking the dot product of these two vectors. A large dot product means the vectors are similar and that the response should receive a high score. I then applied a sigmoid function to convert that score into a probability.

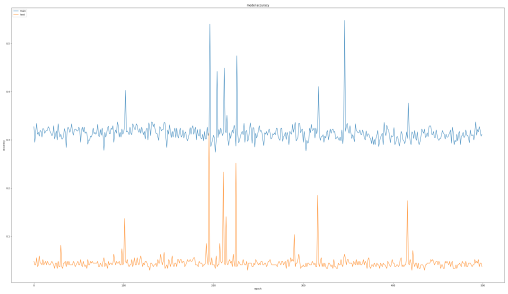
Implementation

As mentioned earlier, nBot uses a pre-trained word embeddings model from Associated Press News for training. This model is trained on keras using tensorflow as a backend. It uses a Recurrent Neural Network consisting of 4 hidden layers with LSTM cells each having an output dimension of 300. In order to train the neural network, it is important that the input is compatible with the hidden layers. For this, I did a preprocessing on input data where I created two lists for input queries and output responses. Every sentence was clipped to 15 words and each word was represented as a word vector of length 300. For every word in the corpus, I tried to find a vector for it from the pretrained word2vec model which effectively modeled a sentence to be a vector of word-vectors. This vector of word vectors is eventually fed to the recurent neural network. Weight initialization as well as the choice of optimizer are critical for training the RNNs. I have used "adam optimizer" with "cosine proximity" as the loss/cost function. Further, the model has used "glorot normal" initializer to set the initial random weights of Keras layers. This implementation is build on top of keras with tensorflow backend.

One catch with using Neural Networks is that they can only be applied to problems where inputs and outputs can sensibly be encoded with vectors of fixed dimensionality. Therefore, I padded . Following is the sample conversation generated by nBot after training for 5000 epochs. These alien words are due to the presence of padding elements in input.

User:hi
nBot:how are you doing ? kleiser kleiser karluah karluah karluah kleiser kleiser kleiser kleiser kleiser
User: Did I ever live?
nBot:it depends how you define life ballets kleiser karluah karluah karluah kleiser kleiser kleiser kleiser
User: Who are you?
nBot: what ? who is but a form following the function of what kleiser kleiser karluah

Though nBot is a functional conversational agent, it is still pretty naive in handling existing utterances and gives glibberish on receiving unknown/new inputs.



Discussion

Despite

Context Awareness

In real conversations, people don't start thinking from scratch every second to come come up with a new response. They build the conversation based on relevant concepts they might have discussed in past. Therefore, to produce sensible responses systems may need to incorporate both linguistic context and physical context. The most common approach is to embed the conversation into a vector, but doing that with long conversations is challenging in simple recurrent neural networks. Experiments by (Serban et al. 2016) and (Yao, Zweig, and Peng 2015) both go into that direction. Besides maintaining the context of topic, a smart conversational agent should also incorporate other kinds of contextual data such as date/time, location and information about a user.

Consistent Personality

When generating responses the agent should ideally produce consistent answers to semantically identical inputs. For example, the response to inputs like "How old are you?" and "What is your age?" should be same. This might sound simple, but incorporating such fixed knowledge or "personality" into models is very much a research problem. Since most of the models are trained on data from multiple different users, they tend to give inconsistent responses to the same user utterance. Works like (Li et al. 2016) has proven to be first steps into the direction of explicitly modeling a personality in a conversational bot.

Evaluation of Models

The ideal way to evaluate a conversational agent is to measure whether or not it is fulfilling its task.However such labels are expensive to obtain as there is no well-defined goal. Common metrics such as BLEU(Bilingual Evaluation Understudy) that are based on text matching and used for Machine Translation arent well suited because sensible responses can contain completely different words or phrases. In (Liu et al. 2016), authors have concluded that none of the commonly used metrics really correlate with human judgment.

Intention and Diversity

One of the prime issues with most fo the retrieval-based and generative models is lack of diversity in responses. Google's smart reply is a good example of this whose earlier version used to respond with "I love you" to almost anything. This is partly due to how these systems are trained, both in terms of data and in terms of actual training objective/algorithm. Some researchers have tried to artificially promote diversity through various objective functions (Li et al. 2015). Humans conversations are typically specific to the input and carry an intention. Since generative models arent trained to have specific intentions they lack this kind of diversity.

Choice of corpus

A corpus is a principled collection of texts, written or spoken, which is stored on a computer. It must represent something and its merits are often judged based on how representative it is. There are many corpora available and some can be bought, some are free and some are not publicly available. There is no one corpus to suit all purposes and choice of corpora is really important while training a conversational agent for a specific purpose (O'keeffe, McCarthy, and Carter 2007). For an effective research in dialog systems, corpus should have following qualities:

- Two-way (or dyadic) conversation, as opposed to multi-participant chat.
- Large number of conversations.
- Many conversations with several turns(more than 3).
- Task-specific domain, as opposed to chatbot systems.

Most of these requirements are satisfied by the Ubuntu Dialogue Corpus introducted by (Lowe et al. 2015). However, the dataset is more suited for a technical-support domain and not specifically made to train a practical conversational agent. Some popular corpus that have been used to train conversational bots in past are as follows:

- NPS Chat Corpus: Consists of 10,567 posts in release 1.0. It is a part of python NLTK package.
- NUS Corpus: Collection of SMS messages compiled by NLP group at National University of Singapore.
- Ubuntu Dialogue Corpus: Consists of 1,000,000 examples based on chat logs from the Ubuntu channels on a public IRC network.
- Cornell Movie-Dialogs Corpus: Contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts: 220,579 conversational exchanges between 10,292 pairs of movie characters.
- Microsoft Research Social Media Conversation Corpus: Collection of 12,696 Tweet Ids representing 4,232 three-step conversational snippets extracted from Twitter logs.

This paper replicate the results of (Lowe et al. 2015) for Unstructured Multi-Turn Dialogue Systems which uses Ubuntu Dialog Corpus. UDC is a big corpus and training a conversational agent on so much data with limited computing resources (without gpu) is chalenging. For these reasons, nBot

is trained on a manual corpus based on general conversational dialogs. to get better results after training nBot on a bigger and feature rich corpus in future.

Limitations

Current model has naively assumed that a speech-to-text recogniser can be bolted in front of the system in order to build a voice-driven assistant. However, the distribution of utterances changes dramatically according to the nature of the interaction. Besides, the current accuracy of nBot is not quite impressive due to two primary reasons. First, it has been trained on a small dataset of 86 statements divided into 12 blocks to simulate context. Second, nBot has not been trained for sufficiently large time. Present version has been trained for 5000 epochs which itself took a couple hours.

Future Work

With more modifications, nBot might be used to check effectiveness of certain interaction modes in voice-only conversations. Author(s) have submitted an ORE application to carry out a field experiment in 2018. Since the current implementation is still a textual

References

Britz, D. 2016. Deep learning for chatbots, part 2 implementing a retrieval-based model in tensorflow. Accessed: 2017-12-9.

Hochreiter, S., and Schmidhuber, J. 1997a. Long short-term memory. *Neural Comput.* 9(8):1735–1780.

Hochreiter, S., and Schmidhuber, J. 1997b. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kojouharov, S. 2016. Ultimate guide to leveraging nlp & machine learning for your chatbot. Accessed: 2017-12-9.

Le, Q., and Mikolov, T. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1188–1196.

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

Li, J.; Galley, M.; Brockett, C.; Spithourakis, G. P.; Gao, J.; and Dolan, B. 2016. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*.

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

Lowe, R.; Pow, N.; Serban, I.; and Pineau, J. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909*.

Medsker, L., and Jain, L. C. 1999. *Recurrent neural networks: design and applications*. CRC press.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Mikolov, T.; Yih, W.-t.; and Zweig, G. 2013. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, 746–751.

O'keeffe, A.; McCarthy, M.; and Carter, R. 2007. *From corpus to classroom: Language use and language teaching*. Cambridge University Press.

Pieraccini, R.; Suendermann, D.; Dayanidhi, K.; and Liscombe, J. 2009. Are we there yet? research in commercial spoken dialog systems. In *Text, Speech and Dialogue*, 3–13. Springer.

Ramos, J., et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, 133–142.

Serban, I. V.; Lowe, R.; Henderson, P.; Charlin, L.; and Pineau, J. 2015. A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*.

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

Singh, S. P.; Kearns, M. J.; Litman, D. J.; and Walker, M. A. 2000. Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, 956–962.

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

TechCrunch. 2016. Microsoft silences its new ai bot tay, after twitter users teach it racism. Accessed: 2017-12-9.

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

Yao, K.; Zweig, G.; and Peng, B. 2015. Attention with intention for a neural network conversation model. *arXiv preprint arXiv:1510.08565*.

Young, S.; Gašić, M.; Thomson, B.; and Williams, J. D. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE* 101(5):1160–1179.

Appendix

- Code for nBot: https://github.com/Nalinc/CS-686-Project
- Corpus used in training: https://github.com/Nalinc/CS-686-Project/blob/master/src/conversation.json
- Replicated results: