Dialogue System for Restaurant Reservations using Hybrid Code Network

Charles Akin-David
Stanford University
aakindav@stanford.edu

David Xue
Stanford University
dxue@stanford.edu

Evelyn Mei
Stanford University
evelyn66@stanford.edu

Abstract

Current techniques for end-to-end learning of recurrent neural networks (RNNs) are data-intensive and require thousands of dialogues to learn simple behaviors. However, Williams et. al (2017) introduced Hybrid Code Networks (HCNs) which combine an RNN with domain-specific knowledge encoded as software and action templates. Compared to existing end-to-end approaches, HCNs considerably reduce the amount of training data required, while retaining the key benefit of inferring a latent representation of dialogue state. In this project, we built upon the HCN model proposed by Bordes et al. (2016), and built a dialogue system for taking restaurant reservations.

1 Introduction

Modern task-oriented dialogue systems are challenging to build. Since they are application-specific, a major bottleneck is the limited availability of training data. Compared to existing end-to-end approaches, the Hybrid Code Network (HCN) considerably reduce the amount of training data required, while retaining the key benefit of inferring a latent representation of dialog state. In this project, we analyzed and improved on the HCN model proposed by Bordes et al. (2016), and built a dialogue system for taking restaurant reservations.

2 Background and Related Work

The originator of modern task-based dialogue system is the influential GUS system for travel planning (Bobrow et al., 1977). It is a Frame-based Mixed Initiative system. Each frame is a collection of slots and each slot can take certain pre-defined values. Unlike a simple slot-filling system, which continuously ask questions corresponding to unfilled slots, users can take some degree of conversational initiative in choosing what to talk about.

Subsequently, many different approaches to building dialogue systems emerged. Since they are developed for a variety of tasks in different dialogue contexts, it was very hard for us to figure out which method was more effective. Fortunately, we found this paper by Antoine Bordes and his colleagues published in 2016 (Bordes and Weston, 2016). They introduced an open dataset and task set, and evaluated a number of existing end-to-end goal-oriented dialogue learning systems.

They first analyzed training data and wrote a series of rules that fire for triggers like word matches, positions in the dialog, entity detections, and dialogue state update, etc. With these, they developed a Rule-based system and used it as a tool to predict in which circumstances machine learning algorithms might fail.

Then they built a classical Information Retrieval (IR) model. One of the variants that they tried was Nearest Neighbor. Using word overlap as the scoring method, they found most similar conversation in training set and output the response from that example. Classical IR models are often used as standard baselines and they often perform reasonably well on dialogue tasks.

A stronger baseline is a Supervised Embedding Model. By scoring (conversation history, response) pairs, the model learns to predict the next response given the previous conversation. This method can be thought of as a classical IR model as well, but the matching function is learnt.

Bordes and his colleagues (2016) chose Memory Network as their end-to-end model baseline. It stores historical dialogues and short-term context to reason about the required response. They used the MemN2N architecture proposed by Sukhbaatar and his colleagues (2015). MemN2N outperformed all aforementioned models in nearly all the tasks in the task set. Therefore, we decided to include MemN2N in our model, which we will describe in more details later.

Rather than a generic dialogue system, the system that we are building is task-oriented. Since it is application-specific, limited availability of training data is a major challenge. In order to address this problem, Jason D. Williams and his colleagues proposed Hybrid Code Networks (HCNs), which combine an RNN with domain-specific knowledge encoded as software and system action templates (Williams et al., 2017). Their system workflow is depicted in Figure 1.

In this architecture, entity extraction, RNN and softmax components are learned. In particular, entity extraction follows a predefined list of values. Moreover, all text responses follow templates.

Williams and his colleagues reported that HCNs achieved the same performance as existing recurrent end-to-end neural networks with considerably less training data. HCNs exceed the performance of existing sequence-to-sequence models. In addition, they match performance of past models using an order of magnitude less data (200 vs. 1618 dialogues), which is crucial in practical settings where collecting realistic dialogues for a new domain can be expensive (Williams et al., 2017). This is a very promising study and we believe this framework can be potentially valuable in our project.

3 Approach

The HCN model is long short-term memory (LSTM) based (Figure 2). The feature vector is a concatenation of five components extracted from user input: a bag of words feature vector, an utterance embedding, entities extraction, context features, and actions required. The LSTM computes a hidden state vector, which is retained for the next timestep, and is passed to a dense layer with a softmax activation, with output dimension equal to the number of distinct action templates. The final output of the network is a distribution over action templates. An action mask then checks if preconditions are met and filters out non-permitted actions. The result is normalized back into a probability distribution again. We can sample an action from this distribution or always choose the option with the highest probability. Actions are either text output or API calls.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Figure 2: LSTM Equation

4 Experiments

Following past work, we report average turn accuracy i.e., for each turn in each dialogue, present the (true) history of user and system actions to the network and obtain the networks prediction as a string of characters. The turn is correct if the string matches the reference exactly, and incorrect if not. We also report dialog accuracy, which indicates if all turns in a dialog are correct.

For the dataset, we use the bAbI dialogue dataset (Bordes and Weston, 2016), which includes two end-to-end dialogue learning tasks in the restaurant domain. This dataset consists of synthetic, simulated dialogue data, with highly regular user behavior and constrained vocabulary. Dialogues include a database access action which retrieves relevant restaurants from a database, with results included in the dialogue transcript. We trained on 43000 utterances which included 1000 full dialogues.

4.1 Baseline

As a baseline, we used an implementation of the system described by Antoine Bordes and Jason Weston in the paper *Learning End-to-End Goal-Oriented Dialogue* (2016). The original baseline system was set in the context of restaurant reservation, whose tasks require manipulating sentences and symbols in order to properly conduct conversations, issue API calls, and use the outputs of such calls. The baseline included an Entity

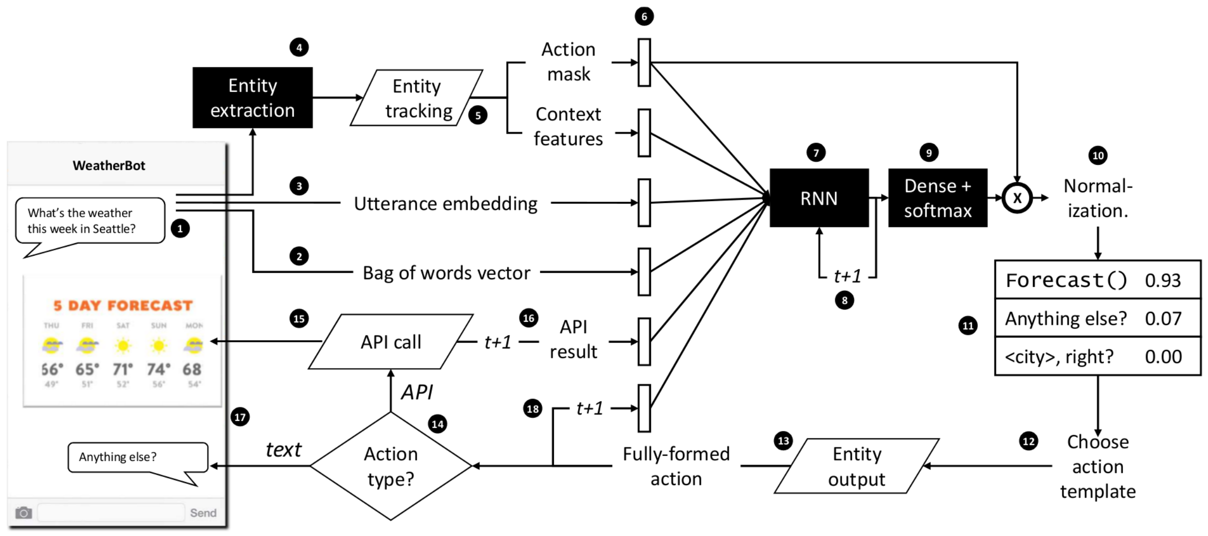
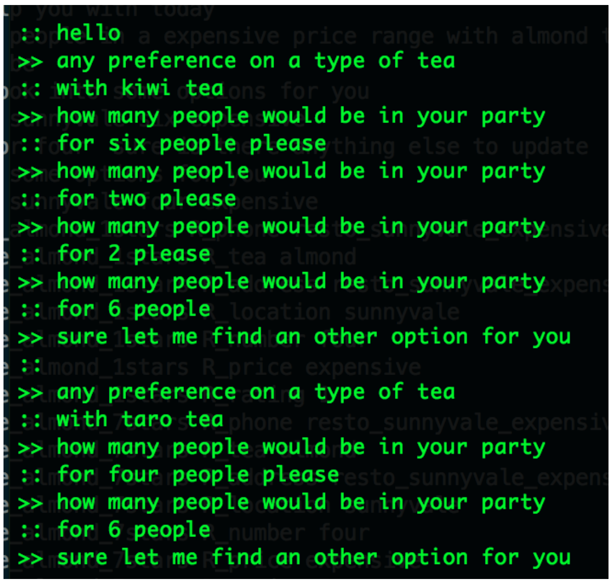


Figure 1: HCN Diagram
Operational loop. Trapezoids refer to programmatic code provided by the software developer, and shaded boxes are trainable components. Vertical bars under 6 represent concatenated vectors which form the input to the RNN. (Williams et al., 2017)

Tracker, Utterance Embedding, and Bag of Words vector all fed into LSTM neural network with the initial parameters shown Figure 3.



```
:: hello
>> any preference on a type of tea
:: with kiwi tea
>> how many people would be in your party
:: for six people please
>> how many people would be in your party
:: for two please
>> how many people would be in your party
:: for 2 please
>> how many people would be in your party
:: for 6 people
>> sure let me find an other option for you
::
>> any preference on a type of tea
:: with taro tea
>> how many people would be in your party
:: for four people please
>> how many people would be in your party
:: for 6 people
>> sure let me find an other option for you
```

Figure 3: Baseline System Example Dialogue

4.2 Hyper-parameter Tuning

As can be observed in Figure 3, the baseline system is imperfect and there is room for considerable improvement. The first experiment that we conducted was to improve on the network structure.

With improved hyper-parameters, we reached 100% validation accuracy in just 1 epoch. Though it doesnt necessarily mean the model is better, it is a significant saving of time and computation power.

The parameter we choose to edit were Hidden Layers, which is the number of hidden layers in the LSTM. Features, which is composed of 4 (entities) + 300 (embedding dimension) + 85 (vocab size). Classes, which were the number of possible responses provided by the system. Epochs, which represents the number of epochs our model took to reach >0.99 accuracy on the development set. Dataset was the number of dialogues the model was trained on.

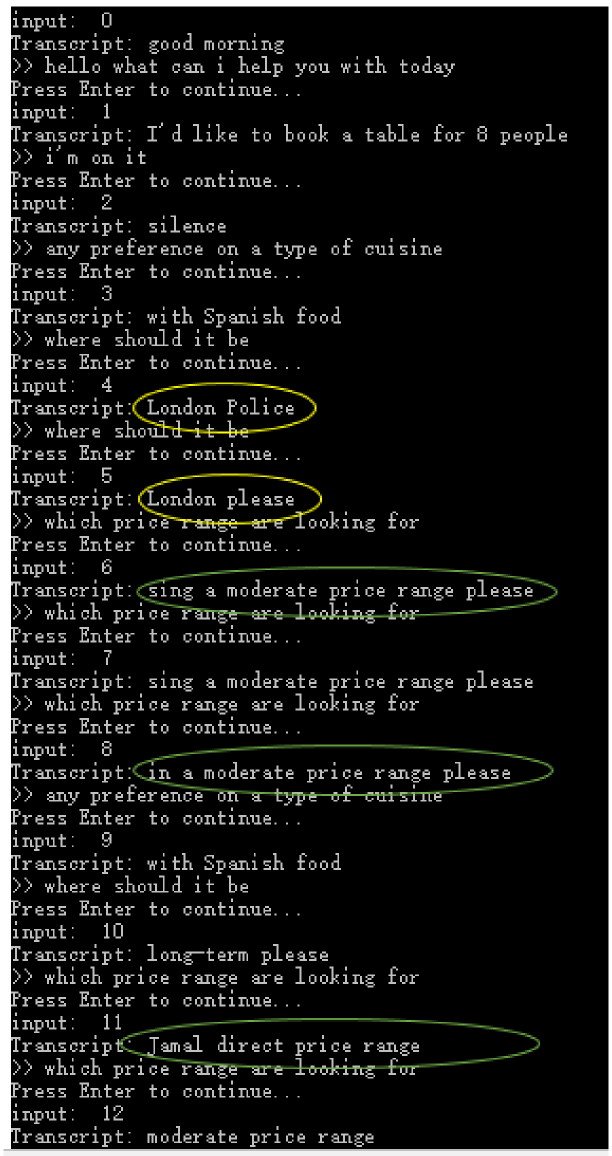
Differences between the Baseline and Final LSTM are shown in Table 1.

LSTM Parameters	Baseline	Final Model
Hidden Layers	128	256
Features	389	389
Classes	16 (action size)	16 + grounding
Epochs	13	1
Learning Rate	1e-1	1e-3
Optimizer	AdaDelta	Adam
Dataset	250	1000

Table 1: Baseline vs. Our Final LSTM

4.3 Automatic Speech Recognition

In order to build a more humanized system, we wanted the system to have real-time interaction with users and be able to understand user's speech input. Therefore, we have integrated Google Speech API. The API provides perfect transcription in most cases. However, when it misinterprets user utterance, the system gets stuck (see Figure 4).

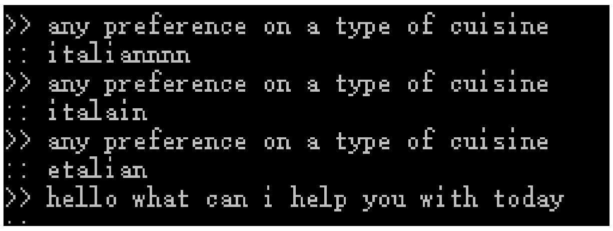


```
input:  0
Transcript: good morning
>> hello what can i help you with today
Press Enter to continue...
input:  1
Transcript: I'd like to book a table for 8 people
>> i'm on it
Press Enter to continue...
input:  2
Transcript: silence
>> any preference on a type of cuisine
Press Enter to continue...
input:  3
Transcript: with Spanish food
>> where should it be
Press Enter to continue...
input:  4
Transcript: London Police
>> where should it be
Press Enter to continue...
input:  5
Transcript: London please
>> which price range are looking for
Press Enter to continue...
input:  6
Transcript: sing a moderate price range please
>> which price range are looking for
Press Enter to continue...
input:  7
Transcript: sing a moderate price range please
>> which price range are looking for
Press Enter to continue...
input:  8
Transcript: in a moderate price range please
>> any preference on a type of cuisine
Press Enter to continue...
input:  9
Transcript: with Spanish food
>> where should it be
Press Enter to continue...
input:  10
Transcript: long-term please
>> which price range are looking for
Press Enter to continue...
input:  11
Transcript: Jamal direct price range
>> which price range are looking for
Press Enter to continue...
input:  12
Transcript: moderate price range
```

Figure 4: Live-interaction with the system using Google Speech API

From the transcripts that it provides, we suspect that the Google Speech API relies on pre-trained word collocation more than what it actually hears. In a typical user utterance like London please, the API transcribed London police as well as Long-term please, which are phrases more commonly seen in natural dialogues.

Nevertheless, using Speech API rather than keyboard input avoids mis-spellings, which used to be another major source of error (Figure 5).



```
>> any preference on a type of cuisine
: : italiannnn
>> any preference on a type of cuisine
: : italain
>> any preference on a type of cuisine
: : etalian
>> hello what can i help you with today
```

Figure 5: Misspelling causes the system to break, avoided using Google Speech API

4.4 Response Grounding

To further improve the dialogue system, we sought out ways to optimize and improve our turn accuracy. We believed that if the system was able to directly respond to a user whenever a specific entity was said, then the user would have confidence that the system understands their request. This belief lead us to implement response grounding by adding a post-processing mechanism that would directly respond to the user whenever one of the four entities needed in order to make an API call was spoken.

The Entity Tracker was mainly utilized to accomplish this. We tracked the entity dictionary and omitted an additional response from the dialogue system whenever a user said a specific entity. We also added special responses to confirm an entity change in the case when a user changes their mind about a specific entity. The system would ask the user if they would like to change their entity from its previous value to the new one entered/spoken. Figure 6 shows an example of dialogue with system with grounding.

The four entities we used were cuisine type, restaurant location, party size, and restaurant price range (see Table 2).

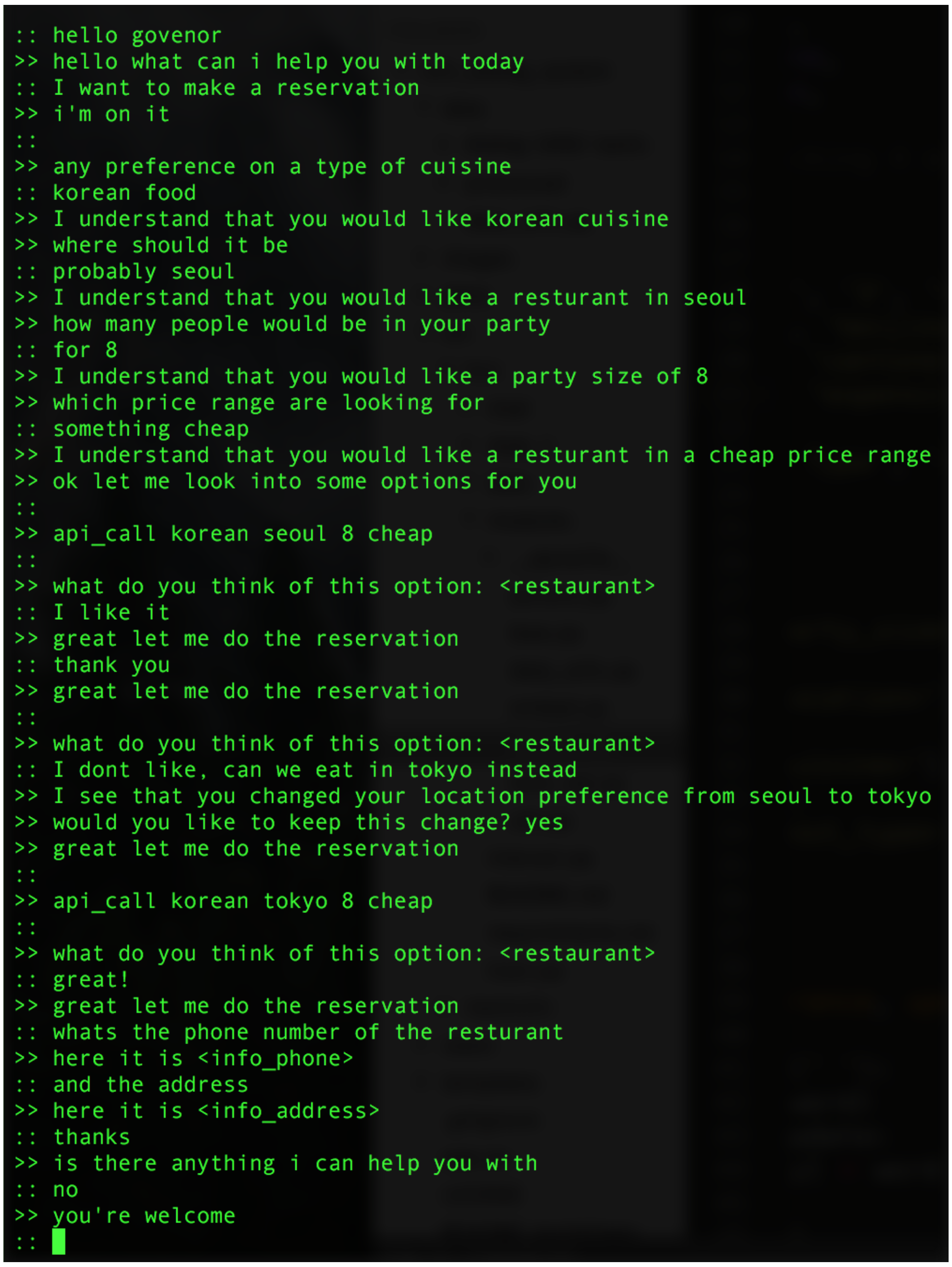
4.5 Results

To analyze the performance of our model, we ran two experiments each computing different scores using an untouched test set.

We analyzed our model on two metrics, turn accuracy and dialogue accuracy. As defined in the

Entity Tag	Description	Examples
cuisine	type of cuisine	british, cantonese, french, indian, italian, korean, spanish
location	place of restaurant	london, bangkok, rome, madrid, london, seoul, tokyo, beijing
party_size	size of reservation party	1, one, 2, two, 3, three, 4, four, 5, five, 6, six, 7, seven, 8, nine
rest_type	restaurant price range	cheap, expensive, moderate, costly, inexpensive, average

Table 2: Entities tracked.



```
:: hello govenor
>> hello what can i help you with today
:: I want to make a reservation
>> i'm on it
::
>> any preference on a type of cuisine
:: korean food
>> I understand that you would like korean cuisine
>> where should it be
:: probably seoul
>> I understand that you would like a resturant in seoul
>> how many people would be in your party
:: for 8
>> I understand that you would like a party size of 8
>> which price range are looking for
:: something cheap
>> I understand that you would like a resturant in a cheap price range
>> ok let me look into some options for you
::
>> api_call korean seoul 8 cheap
::
>> what do you think of this option: <restaurant>
:: I like it
>> great let me do the reservation
:: thank you
>> great let me do the reservation
::
>> what do you think of this option: <restaurant>
:: I dont like, can we eat in tokyo instead
>> I see that you changed your location preference from seoul to tokyo
>> would you like to keep this change? yes
>> great let me do the reservation
::
>> api_call korean tokyo 8 cheap
::
>> what do you think of this option: <restaurant>
:: great!
>> great let me do the reservation
:: whats the phone number of the resturant
>> here it is <info_phone>
:: and the address
>> here it is <info_address>
:: thanks
>> is there anything i can help you with
:: no
>> you're welcome
::
```

Figure 6: Dialogue example with response grounding

Experiment section, turn accuracy is the percent of exact matches between the model's prediction and the true response provided in the test set. The turn is correct if the string matches the reference exactly, and incorrect if not. Dialogue accuracy indicates if all turns in a dialogue are correct. We computed this by seeing what percent of all dialogues had a turn accuracy of 100%. We ran experiments over 200 separate dialogues found in the test set.

We averaged the turn accuracy over all 200 dialogues. Our improved model achieved an average turn accuracy of 96% and a dialogue accuracy of 40%. The baseline model achieved an average turn accuracy of 92% and dialogue accuracy of 33%.

Training our system on the full training set of 1000 dialogues instead of 200 improved its performance over the baseline's. In addition, we found that using the Adam optimizer instead of AdaDelta gave better results for our task of determining realistic responses for restaurant reservation requests. Also, having grounding in addition to the LSTM responses helped to make the overall dialogue system more robust since it is now able to inform users when important key words were captured. This allows users to continue to interact with the system, even if the response generated by the LSTM, doesn't exactly flow with the conversation.

5 Conclusion and Future Work

This paper has validated and improved the Hybrid Code Networks for end-to-end learning of task-oriented dialogue systems. With a relatively small training dataset, we have achieved 96% turn accuracy and 40% dialogue accuracy. We have also extended its functionality by integrating speech input, providing grounding and entity change confirmation in the work flow so as to improve system flexibility and enhance user experience.

However, there are still room for improvement. Some test cases show that the system tends to forget previous inputs. Therefore, we suggest modifying the LSTM in a way that puts more weights on history rather than the most recent input. Another improvement that we thought of is to reduce the pool of possible responses that the LSTM can choose from based on filled entity slots, such that the system will not repeatedly ask for entities that are already filled.

Another area that future scholars could work on is improving the bAbl dataset. Currently all dialogue in the dataset is synthesized. Therefore, human users inputs could be very different from what was used to train the system. As a suggestion for future work, more diverse training examples, especially human-generated examples should be included in the dataset.

References

Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. 1977. Gus, a frame-driven dialog system. *Artif. Intell.* 8(2):155–173. https://doi.org/10.1016/0004-3702(77)90018-2.

Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *CoRR* abs/1605.07683.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. Weakly supervised memory networks. *CoRR* abs/1503.08895.

Jason D. Williams, Kavosh Asadi, and Geoffrey Zweig. 2017. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. *CoRR* abs/1702.03274. http://arxiv.org/abs/1702.03274.

Acknowledgments

We want to thank our professor, Andrew Maas, and head TA, Jiwei Li, for all their advice and support throughout the project. We would also like to thank Suriyadeepan Ramamoorthy for the initial implementation of the HCN Baseline Model.

Appendix

In order to run our model, please ensure that you have **python3** installed. As well as the **google.cloud** python module, for the Google Speech API.