End-to-End Task-Oriented Dialogue Agents

Derek Chen
Stanford University
Department of Computer Science
derekchen14@cs.stanford.edu

Alexia (Wenxin) Xu
Stanford University
Department of Computer Science
alexiaxu@cs.stanford.edu

Abstract

Goal-oriented dialog systems cover a wide range of applications to boost productivity including IT tech support, healthcare consulting and calendar management. Unlike chatbots, goal-oriented agents go beyond entertainment and instead have the aim of accomplishing a task for a user. In contrast to Question-Answer bots, these dialog agents must give their responses in the context of a natural conversation which often span multiple turns. Given the potential for great impact, this project works to reproduce the many components found in cutting edge task-oriented dialog agents. We are able to program most pieces and manage to get a working model, but currently fall short at reproducing the same level of results.

1. Introduction

The past five years has seen an explosion in dialog agents, including voice-activated bots such as Siri, text-based chatbots on Slack, or email-based bots for scheduling meetings. More so than conversational agents, task-oriented dialog agents help users complete tasks such as reporting the weather or navigating through traffic. As the prevalence of such personal assistants continues to grow, so too does the desire for increased capabilities. However, many of these so-called intelligent agents fall short of expectations, often failing to return any useful information to the user [11].

If the abilities of dialog agents were incrementally improved though, it would not be hard to imagine reaching a tipping point where a number of real-world tasks, such as call centers, are replaced or automated by such systems. The difficulty of this problems lies in three main areas. First off, an agent must handle language modeling by producing a meaningful response given a user query. Additionally, since conversations can last multiple turns, the user belief must be stored as some state to keep track of the history of the discussion [19]. Finally, goal-oriented agents also include an information retrieval component for querying facts from a knowledge base (KB). Along those lines, the main goal of this project is to apply techniques and ideas from class to build an effective task-oriented dialog system. We additionally also make progress on replicating certain pieces of state-of-the-art models built for goal-oriented dialog [5].

2. Related Work

The growth of deep learning has spread into almost every discipline in the past few years and natural language processing is no exception [3]. In particular, neural text generation has seen promising progress through the deployment of sequence-to-sequence models that take in a series of tokens and connect to a second network that outputs a different set of tokens [17]. Historically, goal-oriented dialog was a matter of combining a collection of modules to generate a user output with pieces commonly built for parsing, tracking user beliefs, template retrieval and language generation [19]. In more recent times, these systems have incorporated recurrent neural networks [18] as well as reinforcement learning [12].

The most direct comparison to our work comes from [5] who uses an end-to-end model for generating agent responses. In fact, our goal is largely to attempt to reproduce their work. The task outlined for the agent is recommending restaurants based on customer preferences which are elicited through natural language conversation with the user. This draws directly on the work of][1] who establish these as the BaBI Dialog tasks. In further work, Mihail et. al. apply additional methods to an in-car domain using Key-value Retrieval networks [6].

3. Datasets

Our data sources come from the BaBI Dialog dataset, put together by Facebook and described in [1], as well the In-Car dataset put together by Stanford NLP group and described in [6]. Before we are able to start training our model, a large portion of time and effort was spent preprocessing the data. Concretely, we needed to transform the input utterances into embedding vectors by tokenizing each sentence and then indexing into our custom vocabulary to extract a word embedding.

Our corpus consists of 1249 unique words and symbols from all various tasks, which includes 20 special tokens, such as indicators for <SILENCE> (which occurs quite often at the beginning of the conversation), SOS, EOS, api-call, <PHONE>, and <ADDR>. For simplicity, we employed the built-in word embedding methods offered by PyTorch [2]. Based on the limited size of our vocabulary, our past experience suggested that more complicated word embeddings such as GloVE or Word2Vec were unnecessary.

3.1. Facebook BaBI

The BaBI dataset consists of a series of six tasks of increasing difficulty all around the topic of helping a user find a restaurant to eat at given their stated desires. Some example preferences are:

- Type of cuisine (91 choices) - Japanese, French, Thai, Chinese
- Location (10 choices) - London, Tokyo, Miami, Berlin
- Price (3 choices) - cheap, moderate, expensive
- Rating (8 choice) - integers ranging from 1 to 8

Additionally, the knowledge base for certain scenarios also includes the phone number and address of the restaurants in case the user requested that information in follow-up questions.

The first five tasks were created using a collection of rules and thus can be solved completely by a rule-based agent if those rules are reverse engineered. However, using gradient-based methods to infer those rules is indirect, so performing well remains non-trivial. Concretely, the tasks require the agent to:

1. Issue API calls
2. Able to change their mind and update API calls
3. Return a list of restaurants and rank
4. Provide extra information (i.e. phone number or address)
5. Combine all requirements from tasks from 1-4

The sixth task is adapted from the Dialog State Tracking Challenge (DTSC) [19] and has the same premise but much more realistic conversations. Since Task 6 is the most applicable to real world scenarios, we decide to run our experiments primarily on this dataset.

3.2. Ford In-Car

In addition to the restaurant task, Mihail et al. also provides a set of training data extracted from actual in-car conversations passengers have with their car. Specifically, they fall into three categories where the user might ask about:

- *Weather*: Will it rain tomorrow in Los Angeles?
- *Navigation*: What is the route to the nearest gas station?
- *Scheduling*: Who am I scheduled to meet with on Monday?

Given the realistic nature of these dialogues, there is a large amount of variation and accomplishing high accuracy is quite difficult. Text examples of both tasks, as well as trained agent responses, can be found in the results section and the Appendix.

4. Models

The backbone of the project is a modular framework that decouples data loading from training and dialogue generation, allowing us to easily swap in new encoder and decoder components whenever a new feature is added.

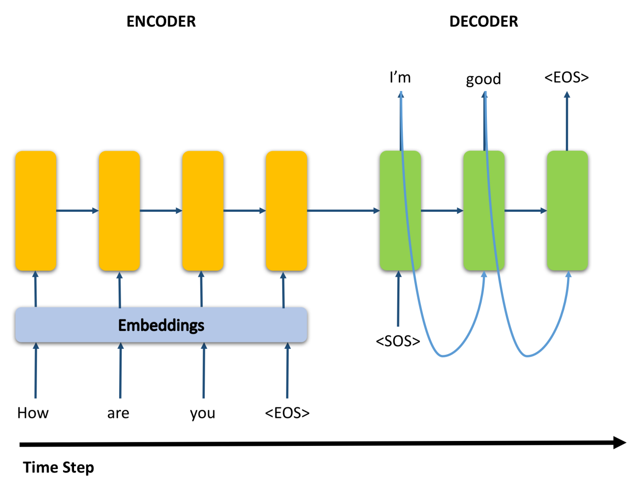


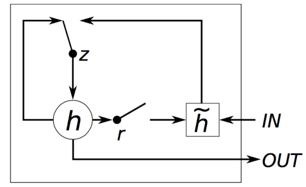
Figure 1. Example sequence-to-sequence architecture

4.1. Baselines

Before diving into the neural-based models, we briefly discuss our baseline which is achieved through TF-IDF matching to select a response for a query from a list of 2000+ candidate responses. More specifically, we set up a pipeline which starts with vectorizing each token into a dense word embedding. Afterwards, these embeddings are concatenated together to form the user input and fed into SK-Learn which tags the term frequencies of the tokens. Finally, we retrieve the candidate response with the most similar TF-IDF signature. Overall, we were able to achieve an per-turn accuracy of 2.1% and 0.0% per-dialog accuracy, which is on par with the 1.6% and 0.0% reported in the original paper [1].

4.2. Seq2Seq

Out baseline neural model is a vanilla sequence-to-sequence system as shown in 1, which consists of two main building blocks: the encoder and decoder. Generally speaking, the encoder is responsible for reading the user utterance at each time step and encoding the entire sequence into a latent representation. In our model, this hidden state implicitly holds the users intent on which restaurant they want eat at or what place they want to go in their car. Based on this last hidden state, the decoder outputs a token at each time step, which continues until we reach an special end-of-sentence token. This output is then joined together to represent the agent response.



$$\mu_i = \sigma(W^{(u)} x_i + U^{(u)} h_{i-1} + b^{(u)})$$

$$r_i = \sigma(W^{(r)} x_i + U^{(r)} h_{i-1} + b^{(r)})$$

$$h_i^{\sim} = tanh(W x_i + r_i \circ U h_{i-1} + b^{(h)})$$

$$h_i = u_i \circ h_i^{\sim} + (1 - u_i) \circ h_{i-1}$$

Figure 2. A GRU unit

For our particular model, each of the hidden units in both the encoder and decoder are gated recurrent units 2 more often referred to as the GRU [4]. In more detail, GRU units control the flow of information using gates, similar to the LSTM [8], but does so in a more computationally more efficient manner by only passing along a single output. Improvements can sometimes be gained by stacking GRU layers together, but as a primitive version, our encoder and decoder both contain only one layer. After brief tuning, we were able to fit the rule-based tasks in the Facebook BaBI dataset well with a perplexity of 0.51.

4.3. Attention

In order to reach better accuracy, we have also implemented the attention mechanism, which allows the decoder to access a weighted combination of all the encoded hidden states instead of only the last one [14]. This is commonly added to a model in order to alleviate the vanishing gradient problem. By adding attention mechanism, our model were able to fit the DTSC tasks well. By tuning the weight decay and dropout ratio, we achieved a training loss of 0.90 and a validation loss of 1.15.

4.4. Match Features

A key factor in the restaurant task is keeping track of the user preferences along the dimensions of the possible restaurant traits such as price range and location. Therefore, these key entities are augmented with additional match features indicating that they are important to the conversation [15]. Specifically, a eight dimensional one-hot vector is appended to the end of each token representing whether it is one of seven special entities. If the eight bit is flipped on, then it indicates that the token is *not* a special item. Adding this extra information often helps training, which we experience as well.

$$Z = \sum_{v \in \mathcal{V} \cup \{\text{UNK}\}} e^{\psi_g(v)} + \sum_{x \in X} e^{\psi_c(x)}$$

Figure 3. The new vocabulary Z is a combination of the original vocab being **g**enerated (denoted as ψ_g) plus the chance of **c**opying from the input (denoted as ψ_c).

4.5. Copy Mechanism

The copy mechanism augments the vocabulary of the system with the tokens found in the input sequence, which effectively allows the decoder to choose a from words only found in the input sequence, but not in the larger vocabulary [7, 9]. During decoding, the softmax is extended to allow it to choose from both the regular vocabulary or from the input sequence when generating the next token 3. Since most the majority of out-of-vocabulary (OOV) come from tokens found in the the user query, this method serves as a feasible solution for dealing with such issues. Ultimately, we are able to get make some progress on incorporating a copy mechanism, but did not have enough time left to run complete tests so this will not be included in the results.

As a side note, we call attention to the fact that we can apply artificial intelligence concepts from class by viewing the problem through the lens of Markov Decision Processes (MDPs). For the neural-based models, the states are tokens at timestep t where the goal of the language model is to generate the highest likelihood word w at the next timestep. Thus, we can formulate this as maximizing $P(w_{t+1}|w_t)$. Additionally, the hidden vector within the GRU can be viewed as modeling the user intents as a continuous state, so if the action is filling slots in a template, and then the goal of the decoder is to discover an optimal policy π^* for taking the best action given the state. To be clear, this is not how the models are actually trained, but simply that these ideas are applicable.

5. Experiments

Our experiments were run on GPUs found on Microsoft Azure with a standard NC6 instance. In our first round of tests, we trained our vanilla Seq2Seq model 7500 iterations (7.5 epochs) and validated every 150 iterations. Since we did not observe any obvious overfitting, we moved onto basic fine-tuning (shown in Figure 4). The curves show a steady drop in loss which reaches a steady state after roughly 3000 iterations. We optimize for lower perplexity by using a negative log-likelihood loss function.

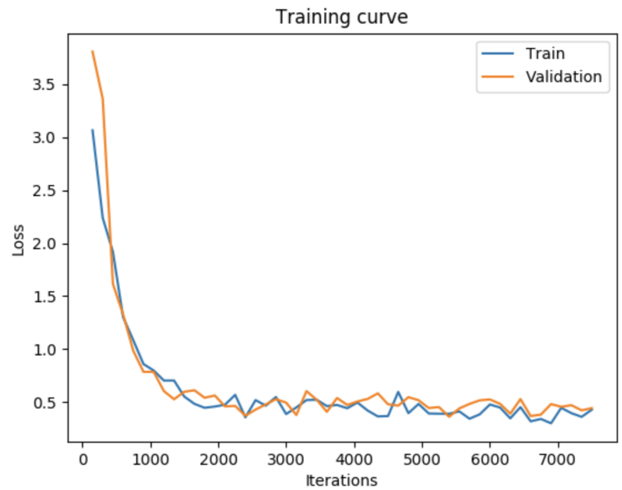


Figure 4. Training curve of the vanilla seq2seq model

5.1. Teacher Forcing

During the decoding process, the network takes the previous hidden state along with a new word token as input to produce the output token, which continues until a <EOS> token is generated. When performing inference, the output token in the previous timestep is fed in as the input token in the current timestep. However, during training there is an option of supervising the training for using the known, correct word as the input token regardless of whether or not the output token in the previous timestep was accurate. The tradeoff is that models that always have access to the correct answer during training tend to overfit. There is an intuitive interpretation that a student who peeks ahead when taking a practice exam doesn't perform quite as well during the actual exam.

As such, our model interpolates between the two extremes based on a parameter γ which is called the teacher forcing ratio. When γ is closer to 1, the model will more likely feed in the correct input tokens during training. Alternatively, when γ is closer to 0, the model will be more likely to feed the previously predicted output token as the input 5. After tuning over multiple trials, we end up with $\gamma = 0.6$ as our best overall weight.

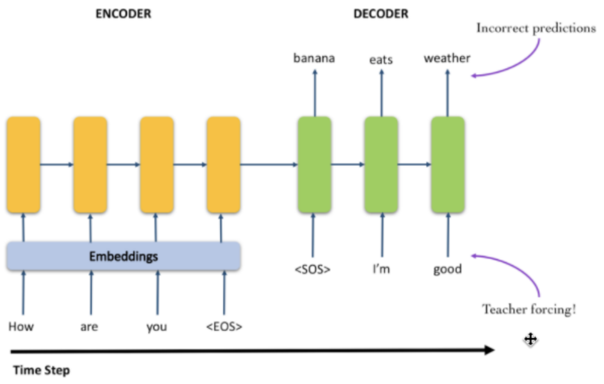


Figure 5. Example of Teacher Forcing

Num Layers	**1 Layer**	**2 Layers**	**3 Layers**	**Bidirect**
Training loss	0.94	0.90	1.06	0.91
Validation loss	1.21	1.15	1.16	1.14

Figure 6. Impact of layers on loss.

5.2. Layers

Another key aspect to getting our model to work properly was tuning the size of the model. Generally speaking, models with more parameters have greater power to fit the data, which is great up to a certain point. Once again there is a trade-off between bias and variance where huge networks can learn just about anything, but will fail to generalize to unseen data, not to mention taking forever to train.

For our purposes, we played with models of 1, 2 and 3 layers 6 as well as 128, 256, 512 and 1024 dimensions as the hidden size 7. Based on preliminary results, we settled on a two layer network with a hidden size of 256 which performed roughly 15% better than other options. As another experiment, we also exchanged the typical encoder with a bi-directional encoder 8. A single layer bidirectional encoder with 512-dim ended up performing the best, which is quite interesting since the forward and backward hidden sizes are 256 units. This effectively means we are back to the same 2-layer network we had just found earlier, with just one of the directions flipped. We use the bidirectional version moving forward.

Hidden unit size	**128**	**256**	**512**	**1024**
Training loss	0.98	0.90	1.02	0.95
Validation loss	1.21	1.15	1.34	1.12

Figure 7. Loss as a function of hidden unit size

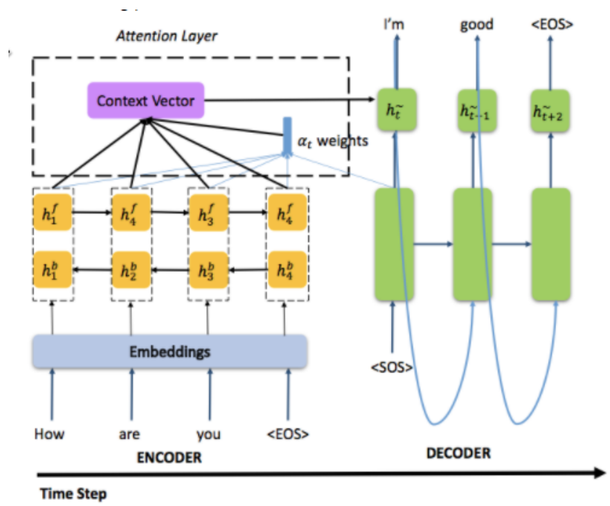


Figure 8. Diagram of Bidirectional Encoder Decoder

5.3. Parameter Tuning

To round out our experiments, we also run a series of tests for parameter tuning.

- *Cell Types*: We start by looking at different RNN cell types including Long Short-Term Memory (LSTM) [8] and Gated Recurrent Unit (GRU) [4] cells. Unsurprisingly, the vanilla RNN performed significantly worse than the gated RNNs with roughly twice the loss, likely due to the vanishing gradient problem. The performance between the LSTM and GRU cells performed statistically the same when we account for the variance across multiple trials, so we just pick the GRU moving forward since ran slightly faster.

- *Optimizer*: Going beyond Stochastic Gradient Descent (SGD), we also tried different optimizers that tune the learning rate λ during the course of training. Specially, we experimented with Adam [10] and RMS Prop, but they did not perform as well as we would have hoped. We ended up with SGD + Annealing, where we lower

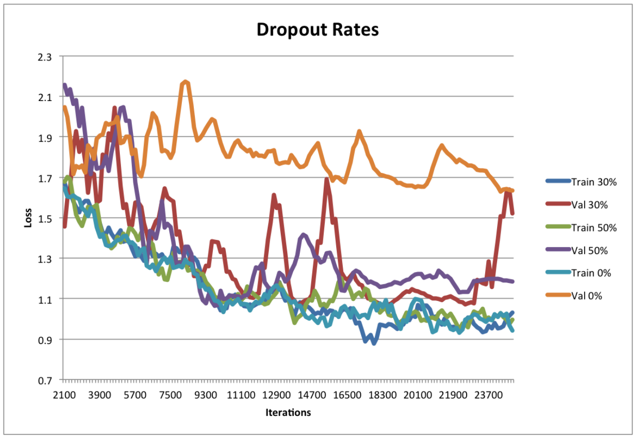


Figure 9. Truncated loss curves with varying dropout rates

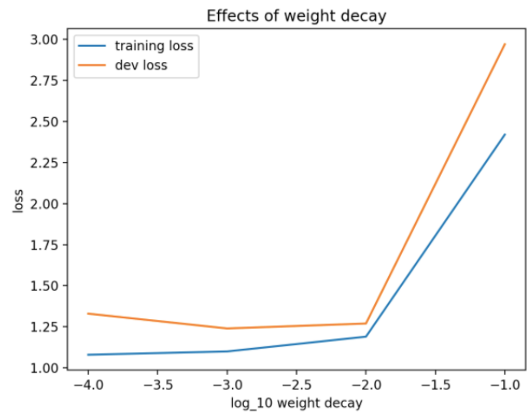


Figure 10. Effect on weight decay on loss

the learning rate to 20% of the previous rate a 1/3 checkpoints. Concretely, when training for 24,000 iterations we anneal the learning rate at 8k and 16k steps.

- *Dropout*: As we increased the amount of dropped units passed from layer to layer, we notice a improvement in validation loss with minimal impact on the training loss up until we reach roughly %50. Figure 9 shows the loss curves for select dropout rates, where the first thousand iterations are truncated so we can more closely analyze the distinctions between the different curves. We settle on 30% drop rate (70% keep rate), which still has slightly erratic behavior during validation, but we believe gives a good balance given the circumstances.
- *Weight Decay*: As another method to prevent overfitting, we add L2 regularizations in the form of weight decay parameter. After numerous trials we end up with $d = 0.001$ or $1e - 3$ as the ideal point 10.

6. Results

We first set the bar for how results should be interpreted and then go into quantitative and qualitative measures.

6.1. Oracle Results

Since our datasets provide a correct answer, a system could theoretically achieve 100% accuracy. However, we didnt believe this was a meaningful metric, so we base our oracle on human-accuracy. For the bAbI dataset, we achieved 66.9% per-turn accuracy and 22.0% per-dialog accuracy. The original paper received 41.1% and 0.0% accuracy respectively using memory networks. A follow-up paper by Eric et.al. achieve 48.0% and 1.5% respectively. A key reason for the gap between the oracle and baseline occurs because the baseline algorithm does not have any mechanism for modeling user belief or changes in state over time.

To understand the human agent scores, we first note that 66.9% is roughly 2/3 of the turns in an example. This is not a coincidence since most examples have three parts of which two parts are answered correctly. The first decision-point is what to place into the API call, which the human gets right 92% of the time. Next, after querying the API for restaurants, 10-20 results might show up and the right answer in the dataset represents a randomly chosen restaurant from the list.

Since the dialog context gives no information on which restaurant will be chosen, generating a response following an API call amounts to guessing a restaurant from the list. Thus, this second decision-point is often wrong. Finally, the customer often requests follow-up information like the address or phone number of the restaurant, which the human agent always gets correct. Ultimately, we should consider $\frac{2}{3}$rds accuracy as roughly the best that any agent can do when we look at the results below.

Restaurant	BLEU	Per-Turn	Per-Dialog
Human Oracle	N/A	66.8%	0.0%
Bordes '16	N/A	41.0%	0.0%
Mihail '17	56.0	48.0%	1.5%
Bid Seq2Seq	78.4	24.2%	0.0%
Bid Seq2Seq + Attn	82.5	27.3%	0.0%
Bid Seq2Seq + Attn + Match	82.2	28.6%	0.0%

Table 1. Evaluation on DSTC test data

In-car	BLEU	Per-Turn	Per-Dialog
Bid Seq2Seq	62.4	13.7%	0.0%
Bid Seq2Seq + Attn	68.9	16.9%	0.0%
Ensemble	69.2	15.3%	0.0%

Table 2. Evaluation on the In-Car test data

6.2. Quantitative Results

In order to compare against the results of [1] and [5] we adopt a few consistent metrics that we use across all our different runs.

To start we calculate the *per-turn accuracy* which is percent of the number of turns where the agent completely predicts all tokens, within all examples. Recall that a certain number of turns then add up into a dialog. Accordingly if all the turns within a dialog are accurate then that dialog is also accurate. The percent of these within all example is considered the *per-dialog accuracy*. Lastly, we also look to the BLEU score which is a standard NLP metric since it has been shown to correlate well with human judgments. At a high level, BLEU is calculated as a weight percent of n-gram overlap from the prediction and the target.

Based on these values we believe that we have not matched the state-of-the-art performance, but we have certainly done much better than random chance. Through a slight ablation study, we are also glad to see that the extra features we added helped performance,

6.3. Qualitative Results

In order to gain a better understanding of why the scores ended up where they did, we also attempted to chat with our agent as a sanity check metric of its abilities. To do so, we randomly printed out conversations the agent had with a user from our validation set to visualize the performance of the model. We see that the responses are reasonably logical despite displaying minor syntactical issues. We are especially impressed by the outcomes because sequence-to-sequence models do not have a candidate list to choose from, and instead generate responses word by word. Thus, even being able to form coherent sentences is often difficult, which our model was able to accomplish.

6.4. Restaurant

Generally speaking, despite low scores in terms of accuracy and BLEU, we are quite happy with the success of our models because they generate meaningful responses. Furthermore, even when the agent's response does not match the target label, we feel that the responses often perform well given the circumstances.

Success Example

```
Turn 1 >>>
user: < SILENCE >
predicted: Hello , welcome to the Cambridge
restaurant system . You can ask for
restaurants by area , price range or food
type . How may I help you ?
target: Hello , welcome to the Cambridge
restaurant system . You can ask for
restaurants by area , price range or food
type . How may I help you ?
Turn 2 >>>
user: im looking for a restaurant that
serves scandinavian food
predicted: api_call R_cuisine R_location
target: What kind of food would you like ?
Turn 3 >>>
user: scandinavian food
predicted: I 'm sorry but there is no
restaurant serving EOS food
target: I am sorry but there is no scand-
inavian restaurant that matches your request
Turn 4 >>>
user: how about a restaurant that serves
french food
predic: api_call french R_location R_price
target: api_call french R_location R_price
... (other turns truncated) ...
```

In this example, we note that the agent is able to successfully make the right API call given the user preference. We also want to note that the target represents the training signal that would get received by the agent. Additionally, notice that in Turn 2, the target answer requests for the restaurant cuisine type even though the user has already stated her desire. As such, gradients flowing from examples like Turn 2 cause noise.

Failure Example See 3

Notice how there are a large number of $I don't care$ responses, in addition to <SILENCE>. Another common refrain was the prevalence of $breathing$ or repeated no answers. In fact, we could almost characterize the user as being downright rude. (See appendix for more) As a result, given the dearth of information coming from the user, we feel it is no surprise that the model can often go off in unfruitful directions.

6.5. In-Car

Please see tables below in the Appendix 2. Sorry for the poor formatting. We note that <POI> represents the point-of-interest and <ADDR> represent a an api call to the address. While it is clear that the per-turn dialogues are indeed "incorrect", the semantics of the response generated by the agent are often right on point. Accordingly, based on our humble view, we believe the low scores for our agent is more an issue with poor ability to evaluate performance at scale [13] rather than actual poor performance.

7. Future Work

Given more time, we would clearly want to finish running all experiments to reach a more decisive conclusion about what methods and parameters worked the best. We were only able to get started on the In-Car dataset so putting more effort in this direction would be ideal. Adding other measures such as Entity F1 score to allow for more direct comparison is also low hanging fruit.

In terms of new ideas, there are two architecture changes in particular that seem extremely promising. On the one hand, we believe that the model would greatly improve with the ability incorporate longer term context by building out a hierarchical-recurrent encoder decoder network [16]. Essentially, there would be two layers of encoders, one for each turn and a longer one for the entire conversation, in order to capture long-term dependencies.

Additionally, the ability to incorporate Key-Value Retrieval networks as seen in a follow up paper would be a clear next step [6]. Finally, great progress has been made recently in using Reinforcement Learning to improve results [12]. In the end, there is a lot of exiting progress still to be made in the area of building effective task-oriented dialog agents!

References

[1] A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016.

[2] S. Chintala. An overview of deep learning frameworks and an introduction to pytorch. *pytorch.org*, 2017.

[3] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[5] M. Eric and C. D. Manning. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. *arXiv preprint arXiv:1701.04024*, 2017.

[6] M. Eric and C. D. Manning. Key-value retrieval networks for task-oriented dialogue. *arXiv preprint arXiv:1705.05414*, 2017.

[7] J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

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

[9] R. Jia and P. Liang. Data recombination for neural semantic parsing. *arXiv preprint arXiv:1606.03622*, 2016.

[10] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] B. Krause, M. Damonte, M. Dobre, D. Duma, J. Fainberg, F. Fancellu, E. Kahembwe, J. Cheng, and B. Webber. Edina: Building an open domain socialbot with self-dialogues. *arXiv preprint arXiv:1709.09816*, 2017.

Speaker	Utterances
User	<SILENCE>
Agent	Hello , welcome to the Cambridge restaurant system . You can ask for restaurants by area ...
User	im looking for a restaurant in any area that
Agent	What kind of food would you like?
User	no
Agent	You are looking for a restaurant serving any kind of food right?
User	no
Agent	What part of town do you have in mind?
User	i dont care
Agent	There are restaurants if you don't care about the area or the type of food . What price range would you like ?
User	i want lebanese food
Agent	Would you like something in the cheap , moderate , or expensive price range?
User	i dont care
Agent	Ok , a restaurant in any part of town is that right?
	... (other turns truncated) ...

Table 3. Failure example

[12] B. Liu and I. Lane. Iterative policy learning in end-to-end trainable task-oriented neural dialog models. *arXiv preprint arXiv:1709.06136*, 2017.

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

[14] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[15] J. Perez and F. Liu. Gated end-to-end memory networks. *arXiv preprint arXiv:1610.04211*, 2016.

[16] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. C. Courville. Multiresolution recurrent neural networks: An application to dialogue response generation. In *AAAI*, pages 3288–3294, 2017.

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

[18] T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[19] J. Williams, A. Raux, D. Ramachandran, and A. Black. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 404–413, 2013.

Speaker	Utterances
User	what will the temperature be this weekend ?
Agent	what city would you like to know the weather for ?
Target	what city do you want to know the temperature ?
User	grand rapids .
Agent	it is not raining in grand rapids today .
Target	it will be between 80-100f in grand rapids this weekend .
User	thanks .
Agent	you are welcome
Target	anytime !

Table 4. Weather example

Speaker	Utterances
User	what is the date of my next optometrist appointment and who will be attending ?
Agent	your next doctor appointment is on the 3rd at with .
Target	i have two , one on thursday with tom , and one on the 3rd with alex .
User	thanks .
Agent	you are welcome
Target	you are welcome driver.

Table 5. Scheduling example

Speaker	Utterances
User	i need a rest stop .
Agent	the nearest gas station is poi ,
Target	<POI> is 5 miles away .
User	farther than i wish but patience.. please send me the quickest way to get there , thank you !
Agent	i sent the info on your screen , drive carefully
Target	sure , it 's just 5 miles away at <ADDR> .
User	please give me a quick route . thank you
Agent	i sent the info on your screen
Target	setting route to <ADDR> now .
User	thank you car !
Agent	you 're welcome !
Target	you 're welcome , happy to help .

Table 6. Navigation example

A. More Failed Dialogues

This is a typical example of dialogue where the user gives very little information to the agent for making recommendations. Other dialogues may not have as many moments if silence, but seeing a large proportion of turns that offer no information was surprisingly common.

```
Turn 1 >>>
user: <SILENCE>
predicted: Hello , welcome to the Cambridge restaurant system . You can ask for restau
Turn 2 >>>
user: breathing
predicted: What kind of food would you like?
Turn 3 >>>
user: part of town cheap restaurant
predicted: There are restaurants in the south of town . What type of food do you want?
Turn 4 >>>
user: <SILENCE>
predicted: You are looking for a restaurant serving any kind of food right?
Turn 5 >>>
user: <SILENCE>
predicted: There are restaurants in the north of town serving any kind of food . What
Turn 6 >>>
user: <SILENCE>
predicted: Would you like something in the cheap , moderate , or expensive price range
Turn 7 >>>
user: <SILENCE>
predicted: api_call R_cuisine north cheap
Turn 8 >>>
user: <SILENCE>  restaurant_two_two is a nice place in the north of town
predicted: anything else  saigon_city is a nice place in the north of town
Turn 9 >>>
user: <SILENCE>
predicted: city_stop_restaurant is a nice place in the north of town
Turn 10 >>>
user: <SILENCE>
predicted: Sure , city_stop_restaurant is on city_stop_restaurant_address
Turn 11 >>>
user: <SILENCE>
predicted: Can I help you with anything else?
Turn 12 >>>
user: thank you good bye
predicted: you are welcome
```