End-to-End Task-Oriented Dialogue Agents

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

Abstract

We develop a task-oriented dialogue agent capable of responding to user queries with increasing levels of sophistication. Unlike chatbots, which simply seek to sustain open-ended conversation, task-oriented agents additionally aim to satisfy user goals in domain-specific topics. In doing so, task oriented dialogue often must model some form of user intent while also planning over multiple turns. Historically, the user belief state was explicitly maintained by the agent, but more recently, research has gone in the direction of representing the user intent through continuous vector embeddings obtained through neural networks. This paper replicates work in this direction, as well as adapts newer architectures that have yet to be widely applied to the problem of goal-oriented dialogue.

1 Introduction

Recent years has seen a tremendous growth in interest towards dialog agents, including voice-activated bots such as Siri, text-based chatbots on Slack, or email-based bots for scheduling meetings. Going beyond holding a conversation, a task-oriented dialog agent has the additional aim of helping 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 [12]. Thus, industry practitioners predominantly lean on conventional machine learning methods when deploying dialog agents in production [22].

However, as the abilities of dialog agents are incrementally improved, it is not 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, an agent must perform language modeling by producing a meaningful response given a user query. Additionally, since conversations can last multiple turns, the dialogue state must be stored as some user belief to keep track of the history of the discussion [23]. Finally, goal-oriented agents also include an information retrieval component for querying facts from a knowledge base (KB). Accordingly, this project incrementally applies a number of modern deep learning techniques to build an effective goal-oriented dialog system [6] for the task of restaurant recommendations.

2 Related Work

Research into goal-oriented dialog historically consisted of cobbling together a collection of modules to generate a user output with components commonly built for parsing raw input, tracking user beliefs, retrieving templates and generating language [23]. Moreover, such models stored and updated the user's intent explicitly throughout the conversation. More recently, the explosion of deep learning has spread into numerous academic disciplines and natural language processing is no exception [4]. These systems have incorporated neural networks approaches [22] as well as reinforcement learning [14] to the problem of neural text generation and intent tracking.

In particular, neural language modeling 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 [21]. The first network, referred to as the encoder, has taken on increasingly sophisticated forms ranging from vanilla RNNs to gated networks [19] and Transformer networks [20]. Similarly, the second network, referred to as the decoder, has also grown in complexity by adding adaptations such as attention [1], memory [2] and diversity scoring [13].

The most direct comparison comes from [6] who uses an end-to-end model for generating agent responses where they deviate from past attempts by using an end-to-end differentiable model that holds onto user intents through implicit vector embeddings. This builds directly on the work of [2] 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 [7].

3 Approach

Our baseline model is a basic sequence-to-sequence system consisting of an 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. Based on this last hidden state, the decoder outputs a token at each time step, where a token might actually be an API call to a knowledge base (KB), which continues until we reach an special end-of-sentence token. This output is then joined together to represent the agent response.

3.1 Cell Type

As the first extension, we replace the vanilla RNN with gated RNNs including GRUs [5] and LSTMs [9]. These units control the flow of information using gates that decide whether to directly pass along the training signal, which ameliorates the vanishing gradient problem. Improvements can sometimes be achieved by stacking GRU layers together, so we test this idea by tuning the number of layers within our RNN, as well as incorporating bi-directional layers.

3.2 Attention

For the next level, we implemented the attention mechanism1, which allows the decoder to access a weighted combination of all the encoded hidden states instead of only the last one [15]. Intuitively, the attention mechanism allows the decoder to "pay attention" to words many timesteps back based on its current hidden state, allowing direct access to the user input when it encoded. Calculating this weight is a function of the scoring mechanism employed and can possibly result in different performance2, so we test numerous popular options.

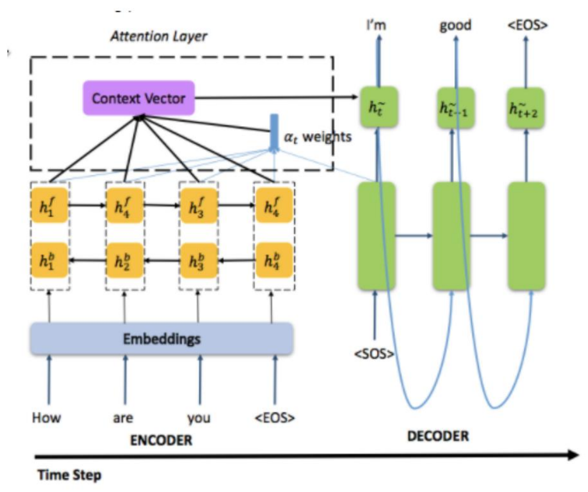


Figure 1: Seq2Seq architecture with attention

$$\text{score} = \begin{cases} \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \end{cases}$$

Figure 2: Scoring methods for attention

3.3 Copy Mechanism

The copy mechanism augments the system with the tokens found in the input sequence, which allows the decoder to generate those words in addition to words found the original vocabulary [8, 10]. By gaining such access, the decoder is better equipped to deal with out-of-vocabulary (OOV) issues when facing unknown tokens. For example, if a user requests a type of cuisine (i.e. Taiwanese) not previously encountered by the agent, it is now able to respond with a coherent "We do not have Taiwanese food available" rather than "We do not have <UNK> available".

Generate Mode

$$p(y_t, \mathsf{g}|\cdot) = \begin{cases} \frac{1}{Z} e^{\psi_g(y_t)}, & \psi_g(y_t = v_i) = \mathbf{v}_i^\top \mathbf{W}_o \mathbf{s}_t, \\ 0, & y_t \in \mathcal{X} \cap \bar{V} \quad (5) \\ \frac{1}{Z} e^{\psi_g(\text{UNK})} & y_t \notin \mathcal{V} \cup \mathcal{X} \end{cases}$$

Copy Mode

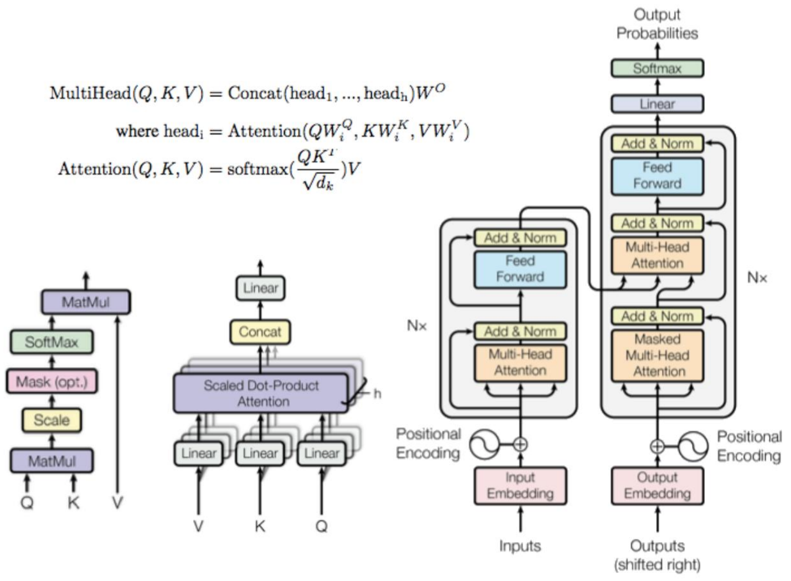
$$p(y_t, \mathsf{c}|\cdot) = \begin{cases} \frac{1}{Z} \sum_{j: x_j = y_t} e^{\psi_c(x_j)}, & \psi_c(y_t = x_j) = \sigma\left(\mathbf{h}_j^\top \mathbf{W}_c\right) \mathbf{s}_t, \\ 0 & \text{otherwise} \end{cases}$$

where:
y_t = predicted token at time t
x_j = word from input at time j
h_j = hidden state of encoder
s_t = hidden state of decoder
W_o, W_c = learned matrices

More concretely, the softmax is extended during each decoding step where the option to generate from the vocabulary and the option to copy from the input sequence compete directly with each other. This can be seen in the figure above 3.3, where the Generation Mode and Copy Mode share the same normalization constant Z when calculating their conditional probabilities. The ability to generate is decided by processing the decoder hidden state with W_o before selecting from the vocabulary v. In contrast, the ability to copy is decided by first pre-processing the encoder hidden state with W_c and a sigmoid before being compared against the decoder.

3.4 Transformer

While RNNs have displayed meaningful gains over previous non-neural attempts, new research has emerged that proposes simpler methods for text generation. Specifically, authors from [20] introduce a relatively radical idea that "attention is all you need" to perform machine translation, doing away with recurrent relations altogether. This novel Transformer network simplifies the architecture by relying solely on attention with basic feed-forward networks, and if implemented correctly, also significantly reduces training time by allowing for parallelized computation across multiple words. At a high level, the Transformer pushes the advantages of attention to the extreme by adding numerous skip-connections to every time-step that allow it to simultaneously look backward and forward at all words, so encapsulating knowledge in an ordered hidden state is no longer necessary.



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h) W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Figure 3: Transformer architecture with Multi-Head Attention and Scaled Dot-Product Attention

Scaled Dot-Product Attention generalizes attention by introducing a query Q, key K and value V, where the query is compared to the key in order to extract the value. In a typical attention setting, the query is the decoder hidden state, the key is the encoder outputs and the value is also the encoder output. Our model actually maintains this paradigm, but it should be noted the key could describe an item in the KB and differs from the item itself as studied in [16]. Multi-head attention works by first shrinking each of $[Q, K, V]$ into a smaller dimension, running the result through multiple feed-forward networks and concatenating the final output. The shrinking factor is often the same ratio as the number of heads to maintain the same output dimension.

Beyond the new attention mechanisms, the network also augments the inputs by imbuing them with a sense of where the word is located in the sentence. This is done by simply adding a static positional-embedding. Right before prediction, a position-wise feedforward network adds more parameters to improve the output, with LayerNorm and ResNet connections sprinkled throughout. While the network uses a copious amount of tricks, at the end of the day, it is still just an encoder coupled with a decoder and can be analyzed accordingly.

4 Experiments

4.1 Datasets

Our data sources come from the BaBI Dialog dataset, put together by Facebook and described in [2]. Before we are able to start training our model, a large portion of time and effort was spent pre-processing 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. For simplicity, we employed the built-in word embedding methods offered by PyTorch [3] since we have a relatively limited vocabulary size of 1,229 words.

Furthermore, 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, the embeddings of these key entities are augmented with additional match features indicating that they are important to the conversation [17]. 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.

In more detail, The BaBI dataset consists of six tasks, all surrounding the topic of helping a user find a restaurant to eat, given their stated desires. Example preferences include:

- 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) [23] 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.

4.2 Evaluation Metrics

To evaluate our performance against results of [2] and [6] 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 to form a full 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*.

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 weighted percent of n-gram overlap from the prediction and the target tokens. Next, we also visualize and analyze the attention weights of various implementations to gauge their through process. Finally, since our model is a conversational agent, we also evaluate the output by directly ”chatting” with the bot and measuring the quality of the response in terms of fluency and helpfulness.

4.3 Parameter Tuning

Our experiments were run on Titan Xp GPUs from Nvidia where most trials lasted around half an hour with a standard deviation of 4 minutes. Training on CPUs was noticeably slower, but surprisingly tractable with speeds clocking in at 40 minutes on average. We optimize for lower perplexity by using a negative log-likelihood loss function. During evaluation, performance exhibited high variance even for the same set of hyper-parameters, highlighting the unevenness of the landscape.

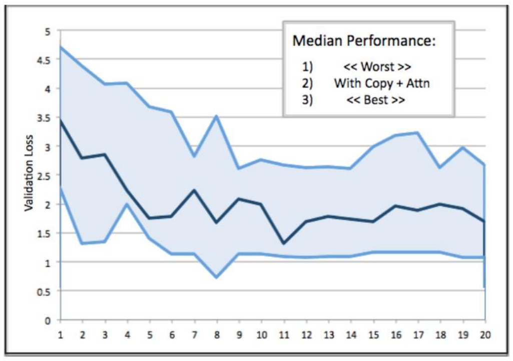
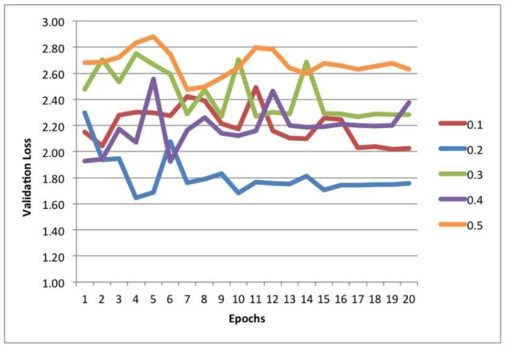


Figure 4: Validation loss across multiple initializations of a single model using the same params.

Dropout: Starting at 50% drop rate, we steadily lowered the amount of dropped units in increments of 10% from layer to layer and noticed an improvement in performance in validation loss until we reached 20% 5. In the figures below, the 5 epochs (out of 25) are truncated so we can more closely analyze the distinctions between the different curves. As we can see, continuing to push this down to 10% and no dropout causes the loss to go back up, so we settled on 20% as the right balance.



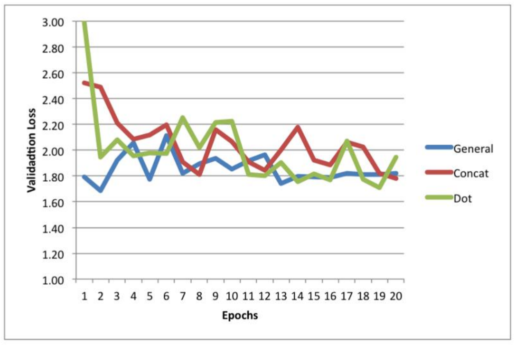


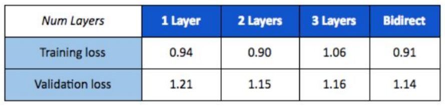
Figure 5: Truncated graph of dropout results

Figure 6: Truncated graph of attention results

Attention: In comparison to dropout, the type of attention method did not cause too much of a difference as all 3 styles performed roughly the same 6. Just as before, the results shown are averaged across three different runs, using the same settings with all other parameters held constant on Task 5. In the end, the best model happened to using dot attention, but these experiments suggest that this result might have just been due to a lucky run rather than some fundamental underlying cause.

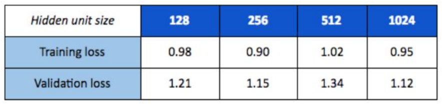
Layers: When deciding on the final size of the model, we note that models with more parameters have greater power to fit the data, but may cause overfitting as we trade-off between bias and variance. Additionally, larger models with often also take longer to train, so in the event that performance is roughly the same, we always choose the smaller network.

Based on preliminary results, we settled on a two layer network 7 with a hidden size of 256 8 which performed roughly as well as much larger options. As another experiment, we also exchanged the typical encoder with a bi-directional encoder. 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.



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 7: Tuning number of layers



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 8: Tuning hidden state dimensions

We also experimented with a number of other parameters and combinations of variables. The best model ended up with a hidden size of 256, 1 bi-directional layer, learning rate of 0.01 annealed at every 8 epochs, 20% drop rate, 0.01 weight decay, dot attention, and 0.6 teacher forcing ratio.

5 Results

5.1 Cell Type

Moving from a vanilla RNN into a gated cell showed a meaningful jump in improvement as the model was able to look further back in the utterance to get a signal. The difference between a GRU and LSTM was minimal, and since the GRU has fewer params we used that moving forward.

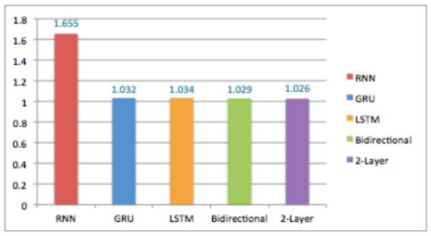


Figure 9: Training loss results on using gated cells and bidirectional layers

5.2 Attention and Copy Mechanism

While the type of attention did not cause a meaningful jump, the use of attention itself was critical for high performance 10. Accordingly, the addition of the copy mechanism also improved performance as expected, which on the graph is called ”Combined”. The low result seen on the graph is actually the copy mechanism by itself without any attention. This makes sense because because without attention neither the generate mode nor the copy mode have any idea how to look for what to output, and is strictly worse than the copy mechanism proposed in [8].

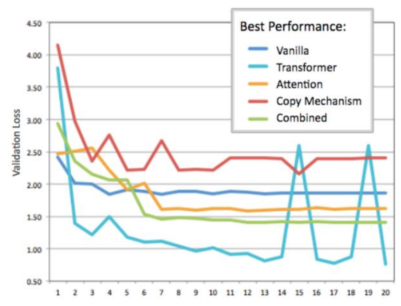


Figure 10: Validation loss for the best performing models at each feature set

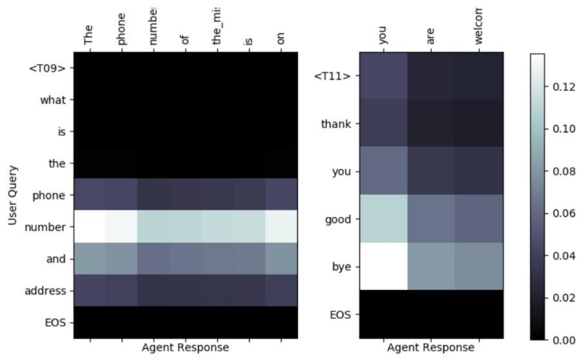


Figure 11: Visualizing the attention weights

When we visualize the attention weights 11, we notice that the network has learned to hone in on keywords when deciding what to output. For example, when it sees the "bye" token, the agent recognizes that the dialogue has ended and responds coherently. While it is great that the model has figured out some key words, we must admit that the dataset itself repeats certain phrases multiple times across many examples, oftentimes word for word. Thus, the good performance exhibited by the model may be a function of the limited domain rather than truly understanding the user.

Restaurant	BLEU	Per-Turn	Per-Dialog	Test Loss
Human Oracle	N/A	66.8%	0.0%	N/A
Bordes '16	N/A	41.0%	0.0%	N/A
Mihail '17	56.0	48.0%	1.5%	N/A
Vanilla RNN	27.5	3.4 %	0.0%	2.455
Bid w/ GRU	50.0	9.94%	0.0%	2.108
Seq2Seq + Attn Only	42.7	20.3%	0.0%	1.964
Seq2Seq + Copy Only	32.6	12.5%	0.0%	2.785
Seq2Seq + Attn + Copy	43.9	28.6%	1.1%	1.692
Transformer	47.5	0.0%	0.0%	0.984

Table 1: Evaluation on DSTC test data

5.3 Transformer

Turning our attention to the most complex model we notice random spikes in performance in the last couple of epochs 10, which we attribute to the momentum behavior of Adam [11]. Across multiple runs, the behavior of networks trained with Adam would be consistently erratic. In fact, for all other models, we ultimately turned to basic SGD as the optimizer of choice. With that said, the Transformer indeed ended with the lowest validation loss of all architectures.

So on the surface, it may seem like the Transformer architecture performs the best, but digging a bit deeper told a different story. Looking at actual agent responses revealed the model had learned to repeat the most common symbol, usually a period or question mark, followed by the <EOS> token. Since the <EOS> token is a part of every true response, the agent was able to achieve relatively high BLEU score due to good unigram and bigram counts. By keeping the response short, it accumulated very low loss since the system did not penalize any words beyond the end of the sentence. This behavior also explains why the per-turn accuracy was so low despite doing well on other metrics.

To manage this issue, we improved our masking of the <EOS> token by removing its impact when calculating BLEU score. Additionally, since the Transformer actually outputs a full sentence during each prediction, we modified the output to run for a given number of iterations and predicting an entire response at once. This improved matters by generating than just the top two tokens, but the sentences were often incoherent. Thus, a couple more tweaks and weeks of hyper-parameter tuning are still required. Overall, the Transformer seems like a promising direction for dialogue generation, but (as warned) is a bit finicky to get working properly.

6 Future Work

Based on our results, we believe that we have not matched the state-of-the-art performance, but we have certainly done much better than random chance. Through our ablation study, we are also glad to see that the extra features we added helped performance as expected. Finally, we believe that Transformer can be adapted for conversational agents, but will require a bit more tuning.

From this project, we experienced first hand the joys and pains of parameter tuning, and even when a model did happen to land in a preferable region of the optimization landscape, the model memorized the simplest thing possible to generate a reasonable response rather than understanding the user intent. Thus, for future work, we want to move away from the BaBI dataset and closer to dialogues that are more varied, with [18] as a great candidate. We also believe that explicitly forcing the agent to aim for user intent, possibly by incorporating a reinforcement learning policy gradient [14], can be a promising route. In the end, there is a lot of exciting progress still to be made in the area of building effective task-oriented dialog agents!

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

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

[4] 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.

[5] 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.

[6] 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.

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

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

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

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

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

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

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

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

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

[16] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

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

[18] P. Shah, D. Hakkani-Tür, G. Tür, A. Rastogi, A. Bapna, N. Nayak, and L. Heck. Building a conversational agent overnight with dialogue self-play. *arXiv preprint arXiv:1801.04871*, 2018.

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

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

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

[22] T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas, 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.

[23] 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.