NUDGING NEURAL CONVERSATIONAL MODEL WITH DOMAIN KNOWLEDGE

Sungjin Lee

Microsoft Research, Redmond, WA, USA

ABSTRACT

Neural conversation models are attractive because one can train a model directly on dialog examples with minimal labeling. With a small amount of data, however, they often fail to generalize over test data since they tend to capture spurious features instead of semantically meaningful domain knowledge. To address this issue, we propose a novel approach that allows any human teachers to transfer their domain knowledge to the conversation model in the form of natural language rules. We tested our method with three different dialog datasets. The improved performance across all domains demonstrates the efficacy of our proposed method.

Index Terms— conversational agents, domain knowledge, natural language rule, neural conversational model

1. INTRODUCTION

Recently, conversational systems have been increasingly adopting neural approaches [1, 2, 3, 4, 5, 6, 7]. Neural approaches are attractive because they allow us to directly train a model on dialog examples with minimal labeling, which significantly reduces the development complexity compared to traditional approaches [8, 9]. Now neural networks are at the center of services like Conversation Learner [1] and Rasa [2] which allow developers to interactively build bots with much less hand-crafted features. For task-oriented conversational systems, however, neural approaches still have many challenges to overcome. Particularly, the overfitting problem of neural approaches can be severe when there is an insufficient amount of dialog examples available. It is mainly because the model over-optimize on training data by capturing spurious features instead of learning actually meaningful features for the task.

In this work, we address this problem with a new paradigm, so-called, *machine teaching* [10]. In machine teaching, the role of the teacher is to transfer knowledge to the learning machine so that it can generate a useful model. With their domain knowledge, the teachers can divine some features that are immune to overfitting because they are created independently of the training data. In Table 1, for example, the learning machine might pick the previous system response, i.e. `I'm on it`, as a key feature in making predictions on next response because it consistently shows up in the limited set of dialogs. But such spurious regularities won't last long as more dialogs become available. In contrast, any human teacher can tell that the user inputs should instead be the key feature despite their varying surface forms. Thus, if there is an easy way to transfer such knowledge, one can bias the model to rely more on semantically robust features.

...
System: I'm on it
User: Actually I would prefer for two people
System: Sure, is there anything else to update?
...
System: I'm on it
User: Actually I want French food
System: Sure, is there anything else to update?

Table 1. Example dialogs in a restaurant finding domain

As a first step toward this goal, we propose a novel conversational model which allows one to express domain knowledge in the form of *Natural Language Rules* (NLRs). [3] Specifically, our method takes as input two sets of rules, *u-rules* and *s-rules*. With *u-rules*, one can suggest what the system should say upon a particular user input, e.g, 'Actually I want $cusine food' → 'Sure, is there anything else to update?'. [4] Whereas, with *s-rules*, one can encourage the system to follow a typical ordering of system actions, e.g., 'what kind of food would you like?' → 'what area of town should I search?'. With the *NLR inferencer* (introduced in Section 2.2), we perform inference based on NLRs. The inference result is passed to the overall conversational model as features to give it a nudge to consider domain knowledge.

The rest of this paper is organized as follows. In Section 2 we describe our conversational model with an NLR inference module. In Section 3 we discuss our experiments and results. We finish with conclusions and future work in Section 4.

[1]https://labs.cognitive.microsoft.com/en-us/project-conversation-learner
[2]http://rasa.com/

[3]We think the use of natural language for knowledge description is important for a wide adoption because representing knowledge with formal languages is a non-trivial skill to master.
[4]In this example, we delexicalized the utterance by substituting the actual value (e.g. French) with its category.

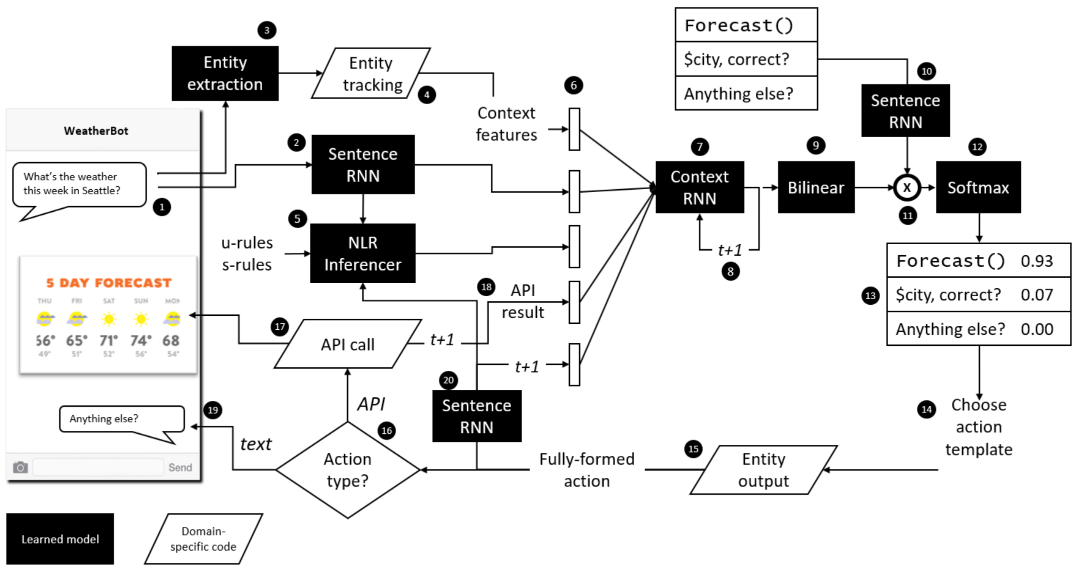

Fig. 1. The overall operational loop. Trapezoids refer to programmatic code provided by the software developer, and shaded boxes are trainable components.

2. CONVERSATION MODEL WITH NATURAL LANGUAGE RULES

In this section, we describe our task-oriented conversation model that makes use of domain-specific rules that are expressed in natural language.

2.1. Overall Architecture

Our conversational model builds upon the Hybrid Code Network (HCN) [6] which servers as a key component for the recent interactive bot development technologies such as Conversation Learner and Rasa. At a high level, our model have five components: sentence RNNs; a context RNN; domain-specific software; domain-specific action templates; and a conventional entity extraction module for identifying entity mentions in text. We use a pre-trained sentence encoder obtained from [11] which encodes a sentence with 4 layer LSTM models [12] with 1,000 hidden units for each layer. We take the hidden state from the top layer at the end of the sentence as the sentence representation. We call this sentence encoder *Neurocon* and use it for all sentence RNNs. Both the context RNN and the developer code maintain state. Each action template can be a textual communicative action or an API call. Figure 1 shows the overall operational loop. The cycle begins when the user provides an utterance, as text (step 1). Second, an utterance embedding is formed, using the sentence RNN (step 2). Third, an entity extraction module identifies entity mentions (step 3), for example, identifying "Seattle" as a $\langle \texttt{city} \rangle$ entity. The text and entity mentions are then passed to "Entity tracking" code provided by the developer (step 4), which grounds and maintains entities, for example, mapping the text "Seattle" to a specific row in a database. This code can optionally return "context features" which are features the developer thinks will be useful for distinguishing among actions, such as which entities are currently present and which are absent. An embedding for the previous system's action is generated (step 5). Then, we feed the NLR inferencer with the user utterance embedding, system action embedding and the two sets of NLRs, *u-rules* and *s-rules*. The NLR inferencer performs inference to yield a vector that represents a soft preference over the set of action templates based on the domain knowledge encoded in the NLRs (step 6). The feature components from steps 1-6 are concatenated to form a feature vector (step 7). This vector is passed to the context RNN. The context RNN computes a hidden state (step 8), which is retained for the next timestep (step 9), and passed to a bilinear layer which projects the hidden state to the response space (step 10), yielding a response embedding. A set of embeddings for each distinct system action template are generated, using the sentence RNN (step 11). This set of embeddings get ranked according to the similarity to the response embedding (step 12). With a softmax, a probability distribution over action templates is generated (step 13). From the resulting distribution (step 14), the best action is selected (step 15). The selected action is next passed to "Entity output" developer code that can substitute in entities (step 16) and produce a fully-formed action, for example, mapping the template "$\langle$city$\rangle$, right?" to "Seattle, right?". In step 17, control branches depending on the type of the action: if it is an API action, the corresponding API call in the developer code is invoked (step 18), for example, to render rich content to

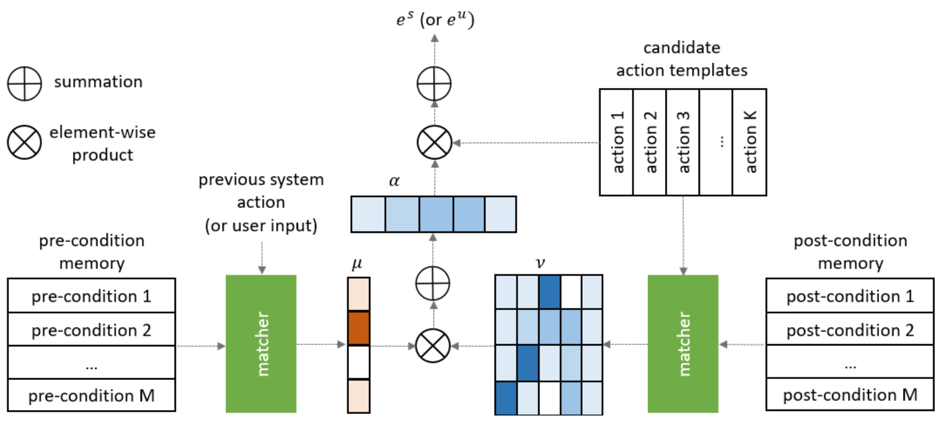

Fig. 2. Natural Language Rule-based inference process

the user. APIs can act as sensors and return features relevant to the dialog, so these can be added to the feature vector in the next timestep (step 19). If the action is text, it is rendered to the user (step 20), and cycle then repeats. Note that there are a few improvements in our model compared to the HCN model, we encode system actions using RNNs rather than just featurizing the system action taken with a binary vector which is all zero values except for the index of the taken action; we rank a set of candidate system actions by matching them with the context rather than performing classification without looking into the actions.

2.2. Natural Language Rule Inferencer

The Figure 2 depicts the entire process of NLR inference. There are two sets of rules *s-rules* and *u-rules*. A rule is a tuple of *(pre-condition, post-condition)* – specifically, the pre-condition and post-condition for *s-rules* (*u-rules*) are "previous system action" and "system action" ("user input" and "system action"), respectively. For each rule set, we store pre-condition embeddings and post-condition embeddings in the pre-condition memory and post-condition memory, respectively. We use the sentence RNNs (in Section 2.1) to encode pre/post-conditions. Then, we match the inferencer input, which is either the previous system action or the current user input depending on the rule set, against the pre-condition embeddings to generate a vector of matching scores, μ (red vector in Figure 2). This indicates how relevant each rule is given the inferencer input. At the same time, we match each of the post-condition embeddings against all candidate action templates to generate a matrix of matching scores, ν (blue matrix). Each element of this matrix represents how relevant a candidate action template is when the corresponding rule gets matched. Thus, the element-wise product of μ and ν yields a weighted matrix such that the summation of it over the rows (i.e., rules), results in a vector, α (blue vector), that represents how relevant a candidate action template is based on the inferencer input and the rules. Finally, we produce output vectors e_s and e_u based on *s-rules* and *u-rules*, respectively by taking a weighted average over the set of candidate action template embeddings with the weight being α. [5] Then, the NLR inferencer yields the final vector by concatenating e^s and e^u. For the matcher, we first measure cosine similarities and normalize them to probabilities through the softmax operation: $m = \mathit{softmax}(\mathit{cosine}(a, b)/\lambda)$, where λ is a scalar to adjust the sharpness of the resulting probability distribution m.

3. EXPERIMENTS

3.1. Data

We use three dialog datasets — `Weather`, `Navigate`, and `Schedule`. Basic statistics of the datasets are shown in Table 2 and Table 3. The datasets are three distinct domains of the recently released Stanford dialog data for an in-car assistant: weather information retrieval, point-of-interest navigation, and calendar scheduling, respectively [13]. Dialogs were collected through Amazon Mechanical Turk using a Wizard-of-Oz scheme. To create datasets that are appropriate for training and testing models, we first delexicalized the datasets based on dialog act annotations and performed random negative sampling from the set of distinct action temples to generate 9 distractors for the true action template for each turn. We created a small number of simple rules, spending time less than a half hour for each domain: 20 rules for `Weather`, 16 for `Navigate`, and 17 for `Schedule`. [6]

[5]Due to the limited coverage of rules, it is possible that there is no entry in the pre-condition memory (or post-condition memory) matching the inferencer input (or candidate action templates). To handle such cases, we introduce two "no match" bias parameters for each matching process and extend the set of candidate embeddings with a zero vector. When we compute the weighted average, the zero vector gets multiplied by the probabilities assigned to the bias terms.

[6]The rules are made available at `https://www.dropbox.com/s/qz6wgxngtzm0dzp/rules.zip?dl=0`.

Domain	Train	Dev	Test
`Weather`	797	99	100
`Navigate`	800	100	100
`Schedule`	828	103	104

Table 2. The size of datasets

Metric	Weather	Nav.	Sch.
Avg. turns per dialog	5.40	6.56	7.32
Avg. tokens per user turn	5.51	7.06	10.06
Avg. tokens per system turn	5.70	7.88	16.14
distinct action templates	187	259	158

Table 3. Data statistics. Nav. and Sch. stand for Navigation and Schedule, respectively.

3.2. Results

To evaluate our proposed model, we conducted ablation tests in Table. 4 – NLR: the proposed model, NLR-S: without *s-rules*, NLR-U: without *u-rules*, NLR-SU: no NLR inference. A more detailed description on the model parameters and training process can be found in Section 3.3. The proposed model outperforms the baseline, NLR-SU, by about 6% - 20% in terms of *Recall@1*. Across all the domains, the performance consistently increases as more domain knowledge gets added. To investigate how much sample complexity is reduced by incorporating domain knowledge, we plot performance curves depending on the training data size in Figure 3. NLR reaches about the same performance with only 100 - 300 dialogs that the baseline model, NLR-SU, achieves with full training data (around 800 dialogs). Finally, as the computation of cosine similarity in the NLR inferencer depends on the quality of sentence embeddings, we report performance with different sentence encoders in Table 5 – NC: Neurocon, ST: Skip-Thoughts [14], WE: we train sentence RNNs with pre-trained word embeddings, SC: we train both sentence RNNs and word embeddings from scratch. The performance consistently increases as we use more pre-trained components. Although NC and ST both are pre-trained sentence encoders, the performance gap between between NC and ST indicates that, for conversational models, a sentence encoder trained on conversation data is better than one trained on plain text like books.

3.3. Training Details

For the sentence RNNs that we trained for the WE and SC models, we use a bidirectional LSTM-RNN with 100 hidden units for each direction. For the SC model, the word embedding weight matrix was initialized with the GloVe embeddings with 100 dimension [15]. For the ST model, we obtained the Skip-Thoughts model [14] at the author's repository [7] which was trained on the BookCorpus data.

[7]https://github.com/ryankiros/skip-thoughts

Domain	NLR	NLR-S	NLR-U	NLR-SU
`Weather`	**71.96**	61.62	67.90	52.77
`Navigate`	**62.69**	57.80	59.94	48.62
`Schedule`	**66.17**	63.68	61.69	58.21

Table 4. Ablation test results in *Recall@1*.

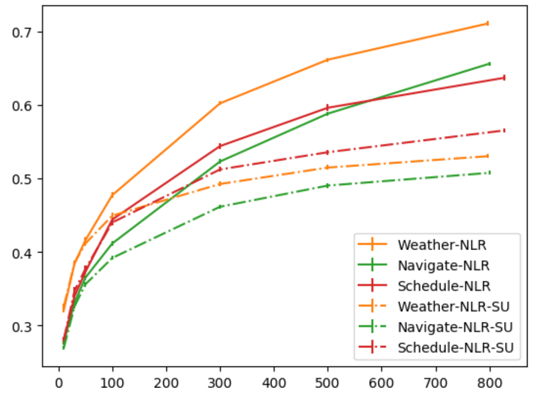

Fig. 3. Performance curves

Domain	NC	ST	WE	SC
`Weather`	**71.96**	52.77	45.76	42.07
`Navigate`	**62.69**	51.99	41.90	35.19
`Schedule`	**66.17**	46.77	46.77	38.31

Table 5. Experimental results with different encoders.

Specifically, we used the uni-skip model which uses a GRU-RNN [16] encoder with 2,400 hidden units. For the context encoder, we use an LSTM-RNN with 200 hidden units. We initialized all the weights of the LSTM-RNN using the *Xavier* uniform distribution [17]. For the matcher, we initialized λ to 0.1. We use the Adam optimizer [18], with gradients computed on mini-batches of size 1 and clipped with norm value 5. The learning rate was set to 1×10^{-3} throughout the training and all the other hyperparameters were left as suggested in [18]. We performed early stopping based on the performance of the evaluation data to avoid overfitting.

4. CONCLUSION

We have presented a novel approach to improve the data-intensiveness problem of neural conversational models. We tackled the problem with the NLR inferencer that allows one to transfer domain knowledge in the form of natural language rules. We tested our method with multiple dialog datasets. The improved performance across all domains demonstrates the efficacy of our proposed method.

5. REFERENCES

[1] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan, "A neural network approach to context-sensitive generation of conversational responses," *arXiv preprint arXiv:1506.06714*, 2015.

[2] Oriol Vinyals and Quoc Le, "A neural conversational model," *arXiv preprint arXiv:1506.05869*, 2015.

[3] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models.," in *AAAI*, 2016, pp. 3776–3784.

[4] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young, "A network-based end-to-end trainable task-oriented dialogue system," *arXiv preprint arXiv:1604.04562*, 2016.

[5] Antoine Bordes and Jason Weston, "Learning end-to-end goal-oriented dialog," *arXiv preprint arXiv:1605.07683*, 2016.

[6] Jason D Williams, Kavosh Asadi, and Geoffrey Zweig, "Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning," *arXiv preprint arXiv:1702.03274*, 2017.

[7] Tiancheng Zhao, Allen Lu, Kyusong Lee, and Maxine Eskenazi, "Generative encoder-decoder models for task-oriented spoken dialog systems with chatting capability," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, 2017, pp. 27–36.

[8] Kristiina Jokinen and Michael McTear, "Spoken dialogue systems," *Synthesis Lectures on Human Language Technologies*, vol. 2, no. 1, pp. 1–151, 2009.

[9] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams, "Pomdp-based statistical spoken dialog systems: A review," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1160–1179, 2013.

[10] Patrice Y Simard, Saleema Amershi, David M Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, et al., "Machine teaching: A new paradigm for building machine learning systems," *arXiv preprint arXiv:1707.06742*, 2017.

[11] Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan, "A persona-based neural conversation model," *arXiv preprint arXiv:1603.06155*, 2016.

[12] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] Mihail Eric and Christopher D Manning, "Key-value retrieval networks for task-oriented dialogue," *arXiv preprint arXiv:1705.05414*, 2017.

[14] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler, "Skip-thought vectors," in *Advances in neural information processing systems*, 2015, pp. 3294–3302.

[15] Jeffrey Pennington, Richard Socher, and Christopher D Manning, "Glove: Global vectors for word representation," *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, vol. 12, pp. 1532–1543, 2014.

[16] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[17] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[18] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *The International Conference on Learning Representations (ICLR).*, 2015.