Recursive Template-based Frame Generation for Task Oriented Dialog

Rashmi Gangadharaiah
AWS AI, Amazon
rgangad@amazon.com

Balakrishnan Narayanaswamy
AWS AI, Amazon
muralibn@amazon.com

Abstract

The Natural Language Understanding (NLU) component in task oriented dialog systems processes a user's request and converts it into structured information that can be consumed by downstream components such as the Dialog State Tracker (DST). This information is typically represented as a semantic frame that captures the *intent* and *slot-labels* provided by the user. We first show that such a shallow representation is insufficient for complex dialog scenarios, because it does not capture the recursive nature inherent in many domains. We propose a recursive, hierarchical frame-based representation and show how to learn it from data. We formulate the frame generation task as a *template-based* tree decoding task, where the decoder recursively generates a template and then fills slot values into the template. We extend local tree-based loss functions with terms that provide global supervision and show how to optimize them end-to-end. We achieve a small improvement on the widely used ATIS dataset and a much larger improvement on a more complex dataset we describe here.

1 Introduction

The output of an NLU component is called a semantic or dialog frame (Hakkani-Tür et al., 2016). The frame consists of *intents* which capture information about the goal of the user and *slot-labels* which capture constraints that need to be satisfied in order to fulfill the users' request. For example, in Figure 1, the intent is to book a flight (*atis_flight*) and the slot labels are the *from* location, *to* location and the *date*. The intent detection task can be modeled as a classification problem and slot labeling as a sequential labeling problem.

The ATIS (Airline Travel Information System) dataset (Hakkani-Tür et al., 2010) is widely used for evaluating the NLU component. We focus on complex aspects of dialog that occur in real-world

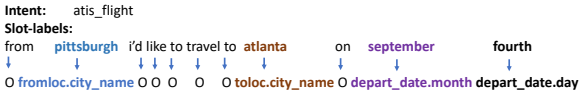

Figure 1: Flat structures used to represent Intents and slot labels in ATIS. 'O' for *Other* or irrelevant tokens.

scenarios but are not captured in ATIS or other alternatives such as, DSTC (Henderson et al., 2014) or SNIPS [1]. As an example, consider a reasonable user utterance, *"can i get two medium veggie pizza and one small lemonade"* (Figure 2A). The intent is *OrderItems*. There are two items mentioned, each with three properties. The properties are the *name* of the item (*veggie pizza, lemonade*), the *quantity* of the item (*two, one*) and *size* of the item (*medium, small*). These properties need to be grouped together accurately to successfully fulfill the customer's request - the customer would not be happy with one small veggie pizza.

This structure occurs to a limited extent in the ATIS dataset (Figure 2B), which has specific forms such as, *from_loc.city_name* and *to_loc.city_name*, which must be distinguished. However, the scale is small enough that these can be separate labels and multi-class slot-labeling approaches that predict each specific form as a separate class (Figure 1) have had success. In more open domains, this hierarchy-to-multi-class conversion increases the number of classes exponentially vs. an approach that appropriately uses available structure. Further, hierarchical relationships, e.g. between *fromloc* and *city_name*, are ignored, which limits the sharing of data and statistical strength across labels.

The contributions of this paper are as follows:

• We propose a recursive, hierarchical frame-based representation that captures complex relationships between slots labels, and show how to

[1] https://github.com/snipsco/nlu-benchmark/tree/master/2017-06-custom-intent-engines

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2059–2064

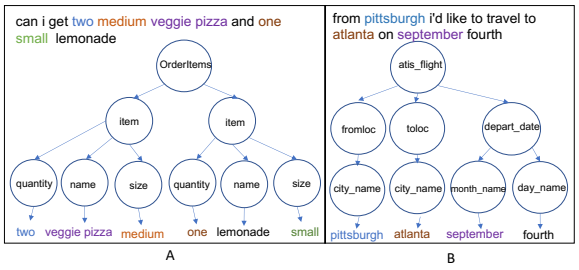

Figure 2: Hierarchical relationships between slot labels and intents. *A*: simulated dataset, *B*: ATIS dataset.

learn this representation from raw user text. This enables sharing statistical strength across labels. Such a representation (Figure 3) also allows us to include multiple intents in a single utterance (Gangadharaiah and Narayanaswamy, 2019; Kim et al., 2017; Xu and Sarikaya, 2013).

• We formulate frame generation as a *template-based* tree-decoding task (Section 3). The value or positional information at each terminal (represented by a $) in the template generated by the tree decoder is predicted (or filled in) using a pointer to the tokens in the input sentence (Vinyals et al., 2015; Jia and Liang, 2016). This allows the system to copy over slot values directly from the input utterance.

• We extend (local) tree-based loss functions with global supervision (Section 3.5), optimize jointly for all loss functions end-to-end and show that this improves performance (Section 4).

2 Related Work

Encoder-Decoder architectures, e.g. Seq2Seq models (Sutskever et al., 2014), are a popular class of approaches to the problem of mapping source sequences (here words) to target sequences (here slot labels) of variable length. Seq2Seq models have been used to generate agent responses without the need for intermediate dialog components such as the DST or the Natural Language Generator (Gangadharaiah et al., 2018). However, there has not been much work that uses deeper knowledge of semantic representations in task-oriented dialog. A notable exception is recent work by Gupta et.al (2018), who used a hierarchical representation for dialog that can be easily parsed by off-the-shelf constituency-based parsers. Neural constituency parsers (Socher et al., 2011; Shen et al., 2018) work directly off the input sentence, and as a result, different sentences with the same meaning end up having different syntactic structures.

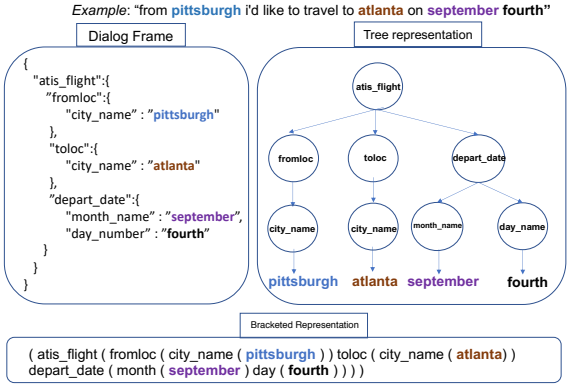

Figure 3: Representations proposed in this paper for an example from the ATIS dataset.

We define a recursive, hierarchical, frame-based representation allows us to exploit some of the structure in natural language while allowing end-to-end training. Our template-based generation is similar to sketch-based Seq2Tree decoding (Dong and Lapata, 2018) developed for SQL query generation, where the decoder predicts a rough sketch of the meaning, omitting low-level details such as arguments and variable names. Here, we generate templates that generalize slot values by their labels.

3 Proposed Approach

We learn to map a user's utterance $\mathbf{x} = \{x_1, x_2, ...x_n\}$ to a template-based tree representation (Figure 2), specifically the bracketed representation in Figure 3. We denote the symbols in the bracketed representation by $\mathbf{y} = \{y_1, y_2, ..y_m\}$. The translation from $\mathbf{x}$ to $\mathbf{y}$ is performed using four components that are jointly trained end-to-end, (1) an encoder, (2) a slot decoder, (3) a tree decoder (Figure 4) and (4) a pointer network. Each of these components is briefly explained below.

3.1 Encoder:

We use BERT (Devlin et al., 2019) as the encoder to obtain token embeddings which are fine-tuned during the end-to-end learning. This can be replaced with any other choice of embedding.

3.2 Slot Decoder:

The slot decoder accepts embeddings from the encoder, is deep, and has a dense final layer which predicts the slot label for each token position $\hat{\mathbf{a}} = \hat{a_1}, \hat{a_2}, ...\hat{a_n}$. The true slot label $\mathbf{a} = a_1, a_2, ...a_n$ is the general form of the label. For example, *city_name*, *month_name* and *day_name* are the general forms obtained

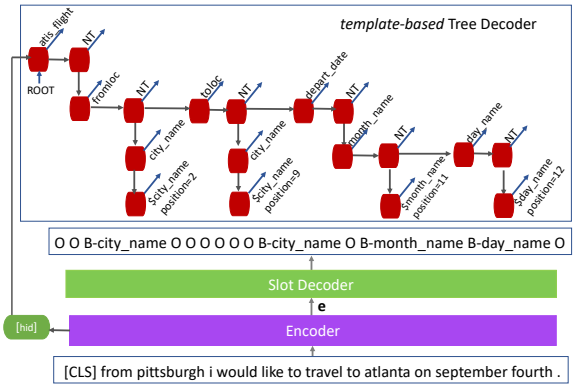

Figure 4: Proposed architecture.

from *fromloc.city_name*, *toloc.city_name*, *depart_date.month_name*, *depart_date.day_name*.

The decoder learns to predict Begin-Inside-Outside (BIO) tags, since this allows the tree decoder to focus on producing a tree form and requires the slot decoder to perform boundary detection. The slot decoder is trained to minimize a supervised loss,

$$\text{loss}_{SL} = -\frac{1}{n}\sum_{i=1}^{n} log\ \pi_{SL}(a_i|\hat{a}_{<i}, \mathbf{x}) \quad (1)$$

where, π_{SL} is the output of the softmax layer at output position i. $\hat{a}_{<i}$ represents slot labels predicted upto position $i-1$.

3.3 Template-based Tree Decoder

The tree decoder works top down as shown in Figure 4. Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) models are used to generate tokens and symbols. In the example shown in Figure 4, the decoder generates *atis_flight NT*. Here, the NT symbol stands for a non-terminal. When a non-terminal is predicted, the subsequent symbol or token is predicted by applying the decoder to the hidden vector representation of the non-terminal. Table 1 walks through this process with an example.

Each of the predicted NTs enter a queue and are expanded when popped from the queue. This process continues until no more NTs are left to expand. The loss function is,

$$\text{loss}_T = -\frac{1}{S}\sum_{s=1}^{S}\frac{1}{T_s}\sum_{t=1}^{T_s} log\ \pi_{TD}(z_t^s|z_{<t}^s, z^s, \mathbf{x}) \quad (2)$$

S refers to the size of the queue for a given training example. T_s refers to the number of nodes (or children) to be generated for a non terminal in the queue, z^s. z_t^s represents the t^{th} child of the non terminal z^s. $z_{<t}^s$ refers to left siblings of z_t^s. Children of z^s are generated conditioned on the hidden vector of z^s and the left siblings of that child.

The tree decoder is initialized with the [CLS] representation of the BERT encoder. The tree decoder generates templates which are then filled with slot values from the user's utterance. In the example, *atlanta* and *pittsburgh* are replaced by *$city_name*, *september* is replaced by *$month_name* and *fourth* is replaced by *$day_name* during training. The $ symbol indicates a terminal.

3.4 Pointer Network:

We predict positions for every terminal, pointing to a specific token in the user's utterance. We perform element-wise multiplication between the terminal node's hidden representation (h) and the encoder representations (e) obtained from the encoder. This is followed by a feed forward layer (g) and a dense layer to finally assign probabilities to each position (p) in the input utterance. That is,

$$p_t = \arg\max_i\ \text{softmax}(g(h(z_t^s) \odot e(x_i))) \quad (3)$$

The pointer network loss, loss_{PT}, is the categorical cross entropy loss between p_t and the true positions. The four components are trained jointly end-to-end to minimize a total loss,

$$\text{loss}_{-G} = \text{loss}_{SL} + \text{loss}_T + \text{loss}_{PT} \quad (4)$$

3.5 Global Context

We found that the tree decoder tends to repeat nodes, since representations may remain similar from parent to child. We overcome this by providing global supervision. This global supervision does not consider the order of nodes, but rather rewards predictions if a specific node is present or not in the final tree. If the model fails to predict that a node is present, the model is penalized based on the number of times it appears in the reference (or ground truth) tree.

Say, $z_1, ...z_K$ is the unique set of nodes present in the reference tree and $N(z_k)$ is the number of times node z_k occurs in the reference. The representation of the [CLS] token is used to predict the presence of these nodes with the loss function,

$$\text{loss}_G = -\sum_{k=1}^{K}\frac{N(z_k)}{\sum_j N(z_j)} log\ \pi_G(z_k|\mathbf{x}) \quad (5)$$

parent	children	Queue contents	Partially generated frame
head	ROOT NT_1	$[NT_1]$	ROOT ( )
NT_1	atis_flight NT_2	$[NT_2]$	ROOT (atis_flight ( ))
NT_2	fromloc NT_3 toloc NT_4 depart_date NT_5	$[NT_3, NT_4, NT_5]$	ROOT (atis_flight (fromloc ( ) toloc ( ) depart_date ( )))
NT_3	city_name NT_6	$[NT_4, NT_5, NT_6]$	ROOT (atis_flight (fromloc (city_name ( )) toloc ( ) depart_date ( )))
NT_4	city_name NT_7	$[NT_5, NT_6, NT_7]$	ROOT (atis_flight (fromloc (city_name ( )) toloc (city_name ( )) depart_date ( )))
NT_5	month_name NT_8 day_name NT_9	$[NT_6, NT_7, NT_8, NT_9]$	ROOT (atis_flight (fromloc (city_name ( )) toloc (city_name ( )) depart_date (month_name ( ) day_name ( ))))
NT_6	**$city_name**	$[NT_7, NT_8, NT_9]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name ( )) depart_date (month_name ( ) day_name ( ))))
NT_7	**$city_name**	$[NT_8, NT_9]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name (**$city_name**)) depart_date (month_name ( ) day_name ( ))))
NT_8	**$month_name**	$[NT_9]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name (**$city_name**)) depart_date (month_name (**$month_name**) day_name ( ))))
NT_9	**$day_name**	$[\emptyset]$	ROOT (atis_flight (fromloc (city_name (**$city_name**)) toloc (city_name (**$city_name**)) depart_date (month_name (**$month_name**) day_name (**$day_name**))))

Table 1: Actions taken to generate the frame representation of the sentence, *from pittsburgh i'd like to travel to atlanta on september fourth*. "NT" refers to non-terminals.

with overall loss,

$$\text{loss}_{\text{weighted G}} = \text{loss}_{-G} + \text{loss}_{G} \quad (6)$$

4 Datasets and Results

We start with ATIS, the only public dataset that has even a shallow hierarchy. The ATIS dataset contains audio recordings of people requesting flight reservations, with 21 intent types and 120 slot labels. There are 4,478 utterances in the training set, 893 in the test set and 500 utterances in the development set. We transform the ATIS dataset to the bracketed tree format (Figure 3).

We also evaluate the proposed approach using a simulated ordering dataset (example in Figure 3). The dataset contains 2 intents and 7 slot labels, 4767 training examples, 1362 test examples and 681 development examples. We manually created templates for every intent (i.e, OrderItems, GetTotal). An intent is randomly sampled, then a template along with a number of items and slot values for each of the properties of the items are randomly drawn to generate an utterance and a bracketed representation for the utterance [2].

[2]The modified ATIS and simulated datasets are available as part of Supplementary material.

4.1 Evaluating the proposed approach

We evaluate both the generalized and the specific forms generated by the proposed model (Figure 5) in Table 2. The exact match criteria requires that the predicted tree completely match the reference tree. As this metric does not assign any credit to partial matches, we also compare all parent child relationships between the reference and the predicted trees and compute micro-f1 scores (Lipton et al., 2014).

Specific:(atis_flight (fromloc (city_name ($city_name)) toloc (city_name ($city_name)) depart_date (month_name ($month_name) day_name ($day_name))))

Generalized: (atis_flight (fromloc (city_name (pittsburgh)) toloc (city_name (atlanta)) depart_date (month_name (september) day_name (fourth))))

Figure 5: Generalized and Specific bracketed forms for, *from pittsburgh i'd like to travel to atlanta on september fourth*.

To measure the benefit of the weighted G loss, we also evaluate an unweighted G loss function,

$$\text{loss}_{\text{unweighted G}} = \text{loss}_{-G} - \frac{1}{K}\sum_{k=1}^{K} log\ \pi_G(z_k|\mathbf{x}) \quad (7)$$

As seen in Table 2, the best performance both on f-measure and accuracy is obtained with the weighted G loss function.

Model	ATIS				Simulated			
	gen-acc	spec-acc	gen-f1	spec-f1	gen-acc	spec-acc	gen-f1	spec-f1
Proposed method,-G	61.74	59.53	88.50	87.29	91.48	90.75	99.64	98.63
Proposed method,+unweighted G	62.21	60.23	87.33	86.81	91.85	90.68	**99.97**	98.63
Proposed method,+weighted G	**72.00**	**70.54**	**89.32**	**88.87**	**92.14**	**91.12**	**99.97**	**98.76**

Table 2: *+/-G:* with or without the global context loss function. *gen:* generalized form metrics and *spec:*results with the specific form. *acc:*accuracy and *f1:* f1-score on parent child relationships.

4.2 Baseline: Extending flat representations with group information

We also compare with a reasonable baseline that extends the traditional flat structured frame (Figure 1) in a way that captures hierarchies. We learn to predict group information along with the slot labels (Baseline in Table 3) by appending indices to the labels that indicate which group the slot label belongs to. Consider, *i want to fly from milwaukee to orlando on either wednesday evening or thursday morning*. This example requires capturing two groups of information as shown in Figure 6. *Group0* contains all the necessary pieces of information for traveling on *wednesday evening* and *Group1* contains information for traveling on *thursday morning*. As shown, *milwaukee* and *orlando* are present in both the groups.

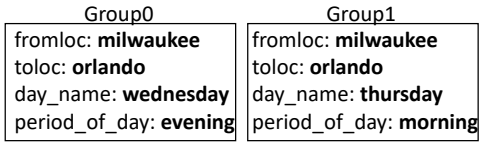

Figure 6: Example shows two groups of information.

We can represent the two *day_name*s (and *period_of_day*) with *B-atis_flight.depart_date.day_name**0*** and *B-atis_flight.depart_date.day_name**1***. We can then use *B-atis_flight.fromloc.city_name**01*** and *B-atis_flight.toloc.city_name**01*** to indicate that they belong to both the groups. Such an approach increases the number of unique slot labels, resulting in fewer training examples for each slot label, but allows multi-class classification methods from prior work to be used as is.

We then train and test the model using the approach that provided highest slot labeling scores which used BERT (Chen et al., 2019). We also convert the generated output of the hierarchical method proposed in this paper to the flat format above. Note, the f1 scores we obtain here are different from those reported in Table 2 as here we only consider the most specific label (eg. *B-atis_flight.toloc.city_name**01***) as the true slot label for a token versus the f1 measure over all the parent child relationships in Table 2. Since adding group information increases the number of unique slot labels, the results reported for the Baseline are different from what has been reported in (Chen et al., 2019).

We notice a large improvement with the proposed approach on the simulated dataset. This implies that modeling hierarchical relationships between slot labels via a tree decoder is indeed helpful. The small improvement we see on ATIS can be attributed to the fact that only a small fraction of the test data required grouping information ($\approx$ 1.7%).

5 Conclusion and Future Work:

With this preliminary work, we showed cases where traditional flat semantic representations fail to capture slot label dependencies and we highlighted the need for deep hierarchical semantic representations for dialog frames. The proposed recursive, hierarchical frame-based representation captures complex relationships between slots labels. We also proposed an approach using a template-based tree decoder to generate these hierarchical representations from users' utterances. We also introduced global supervision by extending the tree-based loss function, and showed that it is possible to learn all this end-to-end.

As future work, we are extending the proposed approach and test its efficacy on real human conversations. More broadly, we continue to explore strategies that combine semantic parsing and neural networks for frame generation.

Model	ATIS	Simulated
Baseline	87.51	32.85
Proposed method + weighted G	88.01	97.67

Table 3: Comparing slot-label f1 scores of the *Proposed approach* and *Baseline*.

References

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. BERT for joint intent classification and slot filling. *CoRR*, abs/1902.10909.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Rashmi Gangadharaiah and Balakrishnan Narayanaswamy. 2019. Joint multiple intent detection and slot labeling for goal-oriented dialog. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 564–569, Minneapolis, Minnesota. Association for Computational Linguistics.

Rashmi Gangadharaiah, Balakrishnan Narayanaswamy, and Charles Elkan. 2018. What we need to learn if we want to do and not just talk. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 3 (Industry Papers)*, pages 25–32.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

Dilek Hakkani-Tür, Gokhan Tür, Asli Celikyilmaz, Yun-Nung Vivian Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *Proceedings of The 17th Annual Meeting of the International Speech Communication Association (INTERSPEECH 2016)*. ISCA.

Dilek Hakkani-Tür, Gokhan Tür, and Larry P. Heck. 2010. What is left to be understood in atis? In *SLT*, pages 19–24. IEEE.

Matthew Henderson, Blaise Thomson, and Jason D. Williams. 2014. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 263–272, Philadelphia, PA, U.S.A. Association for Computational Linguistics.

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

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Byeongchang Kim, Seonghan Ryu, and Gary Geunbae Lee. 2017. Two-stage Multi-intent Detection for Spoken Language Understanding. *Multimedia Tools Appl.*, 76(9):11377–11390.

Zachary C. Lipton, Charles Elkan, and Balakrishnan Naryanaswamy. 2014. Optimal thresholding of classifiers to maximize f1 measure. In *Machine Learning and Knowledge Discovery in Databases*, pages 225–239, Berlin, Heidelberg. Springer Berlin Heidelberg.

Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. 2018. Straight to the tree: Constituency parsing with neural syntactic distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180, Melbourne, Australia. Association for Computational Linguistics.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 129–136, USA. Omnipress.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2692–2700, Cambridge, MA, USA. MIT Press.

Puyang Xu and Ruhi Sarikaya. 2013. Exploiting Shared Information for Multi-intent Natural Language Sentence Classification. In *Interspeech*.