Confirmation Generation for Almond Virtual Assistant

Silei Xu
SUID: silei
Computer Science Department
Stanford University

Background

Virtual assistants are built to scale with a large open-world repository of skills. For example, the Amazon Alexa virtual has 40,000 skills and Google Home has 1 million functions. They accept natural language utterances as intents and dispatches them to the named services.

The open-source Almond virtual assistant (Campagna et al. 2017) lets user issue compound commands in natural language and automatically translates them to programs in a formal language called ThingTalk. Compound commands combine APIs from an open-source, crowdsourced Thingpedia currently comprising 64 devices and services, with 212 functions. Fig. 1 shows how a natural-language sentence can be translated into a ThingTalk program, using the set of services in Thingpedia.

Problem

Whereas today's commercial virtual assistants can only understand skills that are hard-coded, future virtual assistants will let users specify new commands that compose skills from different service providers. Almond, as a pioneer in this area, translates natural language to a formal language called ThingTalk. ThingTalk can compose compound commands by combining different APIs from a repository called Thingpedia, and applying filters and parameter passing on top. For example, as shown in Figure 1, the virtual assistant first retrieve a cat picture from The Cat API, and then post the returned result to Facebook.

The increase of complexity and expressiveness of virtual assistant's capability also increases the complexity of parsing. Sometimes two commands with different semantics may be just too similar in natural language to distinguish without reading user's mind. Further, unlike question-answering problems, virtual assistant commands make side effect to the world, e.g., posting on user's Facebook. If an mistake is made in the question-answering problem, user can simply try again. While if a virtual assistant makes an mistake and executes the wrong command, it might cause some non-reversible effect. Thus, a confirmation is needed before the command is executed.

Github: `https://github.com/rayslxu/tt2nl`

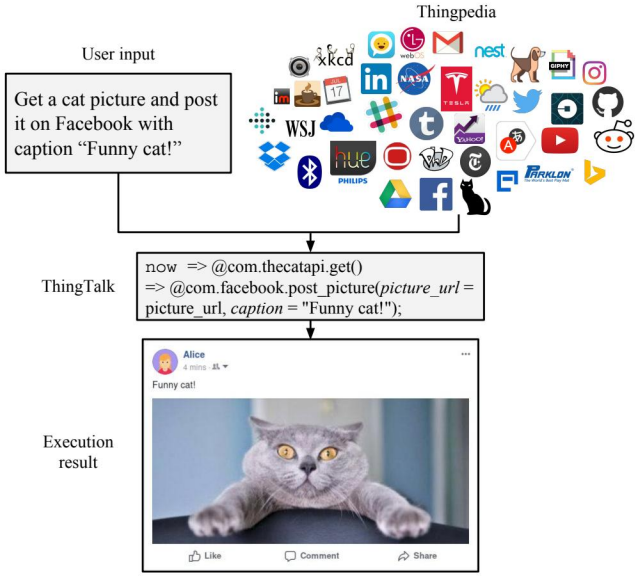

Figure 1: An example of translating and executing compound virtual assistant commands.

In this project, I will try to solve the reverse problem of the common virtual assistant parsing problem: translate the formal language and generate a natural language sentence for confirmation.

Dataset

ThingTalk Programs

ThingTalk programs have a single control construct, with up to three clauses, each calling an API in Thingpedia:

$$when \Rightarrow get \Rightarrow do$$

WHEN specifies the time or event that triggers the operation, GET (optional) retrieves data and DO performs a side effect. WHEN defaults to `now` which indicates the execution should take place once, now; the keyword `monitor` indicates that the program reacts to changes in the data. DO defaults to `notify`, meaning that the result will be returned to the user. One or more parameters can be passed between the functions. Results can be filtered with one or more predicates

Natural language	Program
my Dropbox files	@com.dropbox.list_folder()
my Dropbox files in alphabetical order	@com.dropbox.list_folder(*order_by* = name_increasing)
my Dropbox files that changed most recently	@com.dropbox.list_folder(*order_by* = modified_time_decreasing)
my Dropbox files that changed this week	@com.dropbox.list_folder(*order_by* = modified_time_decreasing) `filter` *modified_time* $\geq$ `start_of_week`
my Dropbox files larger than $file_size	@com.dropbox.list_folder() `filter` *file_size* $\geq$ $file_size
files in my Dropbox folder $folder	@com.dropbox.list_folder(*folder_name* = $folder)
when I modify a file in Dropbox	`monitor` @com.dropbox.list_folder()
when I create a file in Dropbox	`monitor` @com.dropbox.list_folder() `on new` *file_name*

Table 1: Examples of developer-supplied annotations for the @com.dropbox.list_folder function in Thingpedia.

based on equality, comparison, or containment.

Contributors to Thingpedia are responsible for supplying representative natural-language utterances for each function. A single Thingpedia function can be used for different purposes depending on the parameters and filters applied to it. For example, the @com.dropbox.list_folder function can be used to list files in different orders, filter on size or modification time, and react to file creation and modification (Table 1).

Synthetic Programs

The number of ThingTalk programs grows quickly with the number of functions, because different parameters, filters, operators can be used for a single function and multiple functions can be combined with different parameter passing.

To cover the program space, we generated 65,349 different programs by randomly applying the ThingTalk grammar on Thingpedia functions. As shown in the first column of Table 2, there are 238 possible primitive programs with no input parameters: there are 57 DO functions, the 66 GET functions can be used in the WHEN or GET clause, and some GET functions have several variants to monitor different parameters. We combine primitives to derive 22,326 compound programs in which only constant parameters are used. We add parameter passing and filters by sampling them to create more complex programs.

Natural Language Dataset

The ThingTalk dataset consists of three components (Table 2). The *Synthetic*, consisting of 573,878 sentences, is constructed by combining the natural language utterances of functions provided by the developers for each program generated. The utterances are composed via a set of templates, consisting of 56 rules for the main WHEN-GET-DO construct and 27 rules for filters, to produce reasonably natural sentences (Fig. 2(a), (b)). To further increase the lexical variety of the sentences, we randomly substitute some of the words

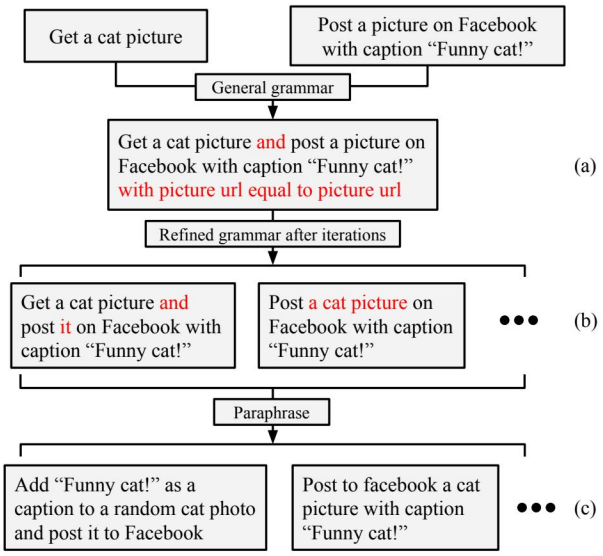

Figure 2: Natural language collection flow

in the generated sentences using PPDB (Ganitkevitch, Van Durme, and Callison-Burch 2013).

To obtain more natural and diverse sentences, we create the *Paraphrase* component. We sample 6,200 sentences from the Synthetic and ask Mechanical Turk workers to paraphrase them (Fig. 2(b)). For each sampled sentence, we ask 3 workers to provide 2 paraphrases each. Only workers with 99% approval rate are employed, and the paraphrases are accepted only if three other workers agree that the original meaning is retained. We reject sentences automatically if constants and named entities are removed. We collect 24,566 paraphrase sentences over 3,410 programs. This set includes 2,139 unique words, compared to just 815 in the Synthetic set.

Splitting to Train, Dev, and Test Set

The synthetic sentences are generated by combining natural language utterances with fixed templates, which turn out to be clunky and hard to understand in a lot of cases. Thus 1000 sentences are randomly sampled from the paraphrase set for the dev and test set, respectively. The rest including the entire synthetic set will be used for training.

Seq2Seq Model

The model is based on a sequence-to-sequence neural network with attention. The ThingTalk is encoded using word embeddings projected onto a low-dimensional space and passed into a bidirectional recurrent neural network. We then concatenate the forward and backward hidden layers to get the final hidden states.

The encoding is then passed to a recurrent decoder. Attention proposed in (Luong, Pham, and Manning 2015) is used. The model greedily takes the action with the highest score, and feeds its embedding back into the decoder at the next step.

Complexity	Synthetic		Paraphrase	
	Programs	Sentences	Programs	Sentences
Primitives	238	15,166	96	1,860
+1 parameter	674	51,446	164	2,283
+ $\geq$ 2 parameters	902	6,822	114	899
+1 filter	11,592	158,835	1,378	9,600
+ $\geq$ 2 filters	3,317	23,127	20	147
Total primitives	16,723	255,396	1,772	14,789
Compounds	22,326	102,449	851	5,369
+ parameter passing	6,068	118,021	121	722
+1 filter	15,039	66,067	555	3,117
+ $\geq$ 2 filters	2,944	7,987	86	431
+ filter & parameter passing	2,249	23,958	25	138
Total compounds	48,626	318,482	1,638	9,777
Total	**65,349**	**573,878**	**3,410**	**24,566**

Table 2: Characteristics of the ThingTalk Dataset. Primitives refer to commands that only use one API, compounds use two. Filters are predicates applied to the result of an API call.

A masked cross-entropy loss is used to train the model. A BLEU score is calculated to evaluate the performance of the model. If the BLEU score stops improving for 10 epochs, the training early stops.

Both the encoder and the decoder use Adam optimizer to speed up the training.

Experiments

The model is implemented using PyTorch framework with learning rate 0.01, batch size 128, and 64 hidden units. The code can be found at `https://github.com/rayslxu/tt2nl`. To tune the hyperparameters, a set of experiments is conducted to evaluate the performance under different settings. In the experiments, we trimmed the dataset to a subset containing only sentences shorter than 10 tokens. Since the sentences are relative short, we only consider unigrams and bigrams when calculating BLEU score and method 7 introduced in (Chen and Cherry 2014) is used as the smoothing function as it correlates better with human judgment.

We first evaluate how different learning rate affect the training. Figure 3 and 4 depict the masked entropy loss and BLEU score within the first 300 epochs of training, respectively. With learning rate 0.1, the loss function fails to converge and the BLEU score remains low even though the loss decreases. Training with learning rate 0.01 and 0.001 both gives good results while the former reaches the optimal within fewer epochs. Thus, we choose 0.01 as the default learning rate.

We then evaluate how different number of hidden units affect the training. The results are shown in Figure 3 and 4. The training loss goes down to around 2 for all 4 cases. As expected, the more hidden units, the fewer epochs are needed to reach the optimal. However, with the same loss, the BLEU scores with different number of hidden units vary. And even the loss function remains roughly the same, i.e., only decreases by around 0.01 every epoch, the BLEU score

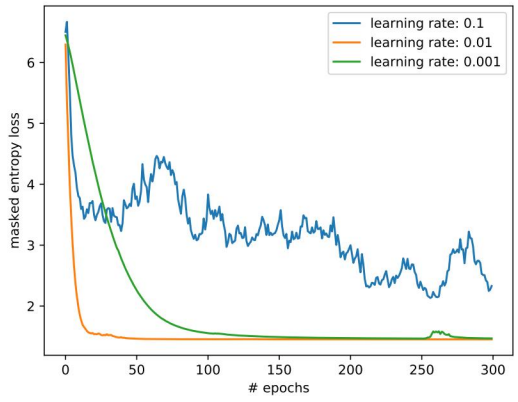

Figure 3: Masked entropy loss with different learning rate

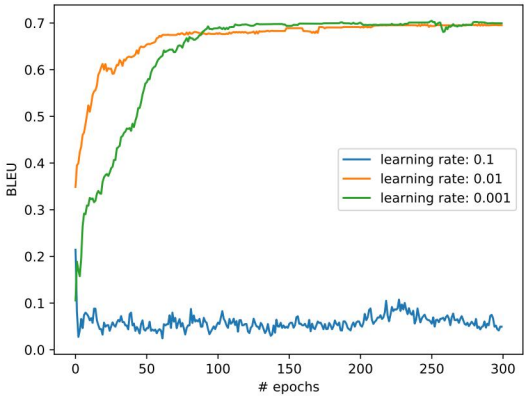

Figure 4: BLEU score with different learning rate

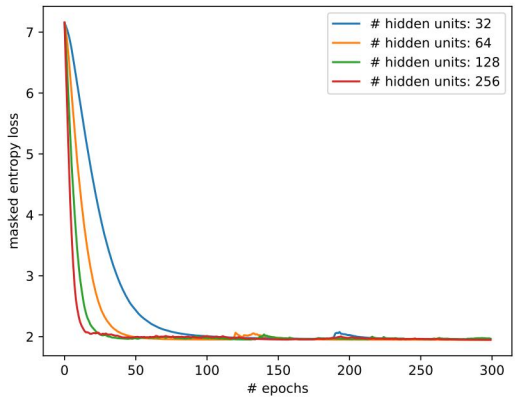

Figure 5: Masked entropy loss with different number of hidden units

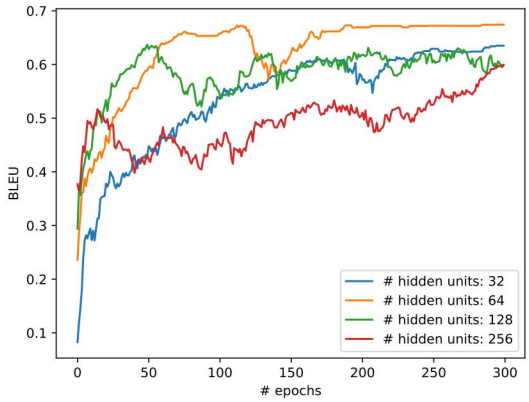

Figure 6: BLEU score with different number of hidden units

still improves a lot in lot of cases. And as expected, the model overfits when the size of the network goes bigger. We choose 64 as the default number of hidden units.

In the following, we show some of the example confirmation sentences generated by the model. We observe that the model succeeds to generate most part of the sentence, but it fails to learn to stop: it append random tokens after it generates the right sentences. A penalty based on repeating tokens and sentence length were added to the model, but it still does not solve the problem. I think there are still bugs in either the loss function or the padding process and this will be the next thing to solve.

ThingTalk:	`now` ⇒ @com.uber.time_estimate param:start:Location = location:current_location ⇒ `notify`
Ground Truth:	show me the distance of uber to here
Prediction:	show me the distance of uber to here here here

ThingTalk:	`now` ⇒ @tumblr-blog.post_text
Ground Truth:	i desire to make a tumblr blog post
Prediction:	i desire to make a tumblr blog post compost blog

Conclusion

In this project, I started from scratch and put up a basic seq2seq model to translate ThingTalk program to natural language in order to provide confirmation for Almond virtual assistant. The model uses a bidirectional GRU for encoder and the attention mechanism proposed in () in decoder. The implementation is still unfinished and buggy, and the model has not been trained on the full dataset. However, it already generates some confirmation close to the ground truth. It shows the proposed model is promising to solve the problem.

References

[Campagna et al. 2017] Campagna, G.; Ramesh, R.; Xu, S.; Fischer, M.; and Lam, M. S. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, 341–350. New York, New York, USA: ACM Press.

[Chen and Cherry 2014] Chen, B., and Cherry, C. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 362–367.

[Ganitkevitch, Van Durme, and Callison-Burch 2013] Ganitkevitch, J.; Van Durme, B.; and Callison-Burch, C. 2013. PPDB: The paraphrase database. In *Proceedings of NAACL-HLT*, 758–764. Atlanta, Georgia: Association for Computational Linguistics.

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