Dialog State Tracking Challenge 6
End-to-End Goal-Oriented Dialog Track

Y-Lan Boureau[1] and Antoine Bordes[1] and Julien Perez[‡]

1 OVERVIEW

End-to-End dialog learning has emerged as a primary research subject in the domain of conversational systems. It consists in learning a dialog policy from transactional dialogs in a given domain. In this context, usable datasets are needed to evaluate learning approaches, yet remain scarce. For this task, a transaction dialog dataset was produced using a dialog simulation framework developed and released by Facebook AI Research. This document describes the dataset and specifies the evaluation metrics and the corpus format.

2 FAIR DIALOG DATASET - DESCRIPTION

Goal-oriented dialog requires skills that go beyond language modeling, e.g., asking questions to clearly define a user request, querying Knowledge Bases (KBs), interpreting results from queries to display options to users or completing a transaction. Such challenges make it hard to ascertain how well end-to-end dialog models would do, and whether they are in a position to replace traditional dialog methods in a goal-directed setting. However, because end-to-end dialog systems make no assumption on the domain or dialog state structure, they are holding the promise of easily scaling up to new domains. This challenge aims to make it easier to analyze the performance of end-to-end systems in a goal directed setting, using an expanded version of the Facebook AI Research open resource proposed in Bordes *et al.* (2017). It breaks down a goal-directed objective into several subtasks to test some crucial capabilities that dialog systems should have (and hence provide error analysis by design). All the tasks involve a restaurant reservation system, where the goal is to book a table at a restaurant. Solving our tasks requires manipulating both natural language and symbols from a KB. The tasks are generated by a simulation. Grounded with an underlying KB of restaurants and their properties (location, type of cuisine, etc.), these tasks cover several dialog stages and test if candidate models can learn various abilities such as performing dialog management, querying KBs, interpreting the output of such queries to continue the conversation or dealing with new entities not appearing in dialogs from the training set.

Task 1: Issuing API calls A user request implicitly defines a query that can contain from 0 to 4 of the required fields (sampled uniformly; in Figure 3, it contains 3). The bot must ask questions for filling the missing fields and eventually generate the correct corresponding API call. The bot asks for information in a deterministic order, making prediction possible.

Task 2: Updating API calls Starting by issuing an API call as in Task 1, users then ask to update their requests. The order in which fields are updated is random. The bot must ask users if they are done with their updates and issue the updated API call.

Task 3: Displaying options Given a user request, the KB is queried using the corresponding API call and the resulting facts are added to the dialog history (if too many facts satisfy the call, a random subset is returned to avoid overly lengthy data). The bot must propose options to users by listing the restaurant names sorted by their corresponding rating (from higher to lower) until users accept. For each option, users have a 25% chance of accepting. If they do, the bot must stop displaying options, otherwise propose the next one. Users always accept the option if this is the last remaining one. We only keep examples with API calls retrieving at least 3 options.

[*2]Xerox Research Centre Europe, Meylan, France `julien.perez at xrce.xerox.com`

[†1]Facebook AI Research, New York, USA `{abordes,ylan} at fb.com`

Task 4: Providing extra information Given a user request, we sample a restaurant and start the dialog as if users had agreed to book a table there. We add all KB facts corresponding to it to the dialog. Users then ask for the phone number of the restaurant, its address or both, with proportions 25%, 25% and 50% respectively. The bot must learn to use the KB facts correctly to answer.

Task 5: Conducting full dialogs For Task 5, we combine Tasks 1-4 to generate full dialogs just as in Figure 3. Unlike in Task 3, we keep examples if API calls return at least 1 option instead of 3.

3 Dataset Format and Evaluation

The dataset is organized in 5 JSON files corresponding to each of the tasks previously mentioned. All tasks are encoded with the same format. Basically, a dialog piece is followed by a list of next-utterance candidates. In the training set, the answer (the single correct candidate) is provided. The goal is to rank the candidates. Precisions @{1,2,5} will be used to evaluate the models. Rank of candidate utterances will be 1-indexed.

```
[
        {dialog_id : " ",
        utterances: [" ", ... ],
        candidates: [
                {candidate_id: " " , utterance: ""}, ...
                ]
        }
]
```

Figure 1: JSON Format of a dialog dataset file

```
[
        {dialog_id : " ",
        lst_candidate_id:
                [ {candidate_id: " ", rank: " "}, ... ]
        }
]
```

Figure 2: JSON Format of a result file

In addition to the corpus, a suite of helper python scripts is provided with the data:

dataset_walker.py Parse the JSON file and print out a plain version of the dataset comprising the dialog utterances, the candidate answers and the correct answers for each dialog in the corpus.

check_validity.py Load a result file and perform a series of verification regarding the format of the result file.

score.py Compute the score of a candidate result file compared to a reference. This is the script that will be used to evaluate the participants during the testing phase of the challenge.

baseline_random.py Produce a random baseline result file for a given task dataset file.

baseline_tfidf.py Produce a baseline result file for a given task dataset file using a TF-IDF similarity heuristic between the candidate answers and the corresponding dialog.

Evaluation uses per-response accuracies. Evaluation is conducted in a ranking, not a generation, setting: at each turn of the dialog, the participants have to test whether they can predict bot utterances and API calls by selecting a candidate, not by generating it.[1] Candidates are ranked from a set of candidate utterances and API calls.

[1] Lowe *et al.* (2016) termed this setting Next-Utterance Classification.

4 Data - Restaurant Reservation Simulation

The simulation is based on an underlying KB, whose facts contain the restaurants that can be booked and their properties queried. Each restaurant is defined by a type of cuisine (10 choices, e.g., French, Thai), a location (10 choices, e.g., London, Tokyo), a price range (cheap, moderate or expensive), a rating (from 1 to more than 200), and other characteristics like dietary restrictions and atmosphere. For simplicity, we assume that each restaurant only has availability for a single party size (2, 4, 6 or 8 people). Each restaurant also has an address and a phone number listed in the KB.

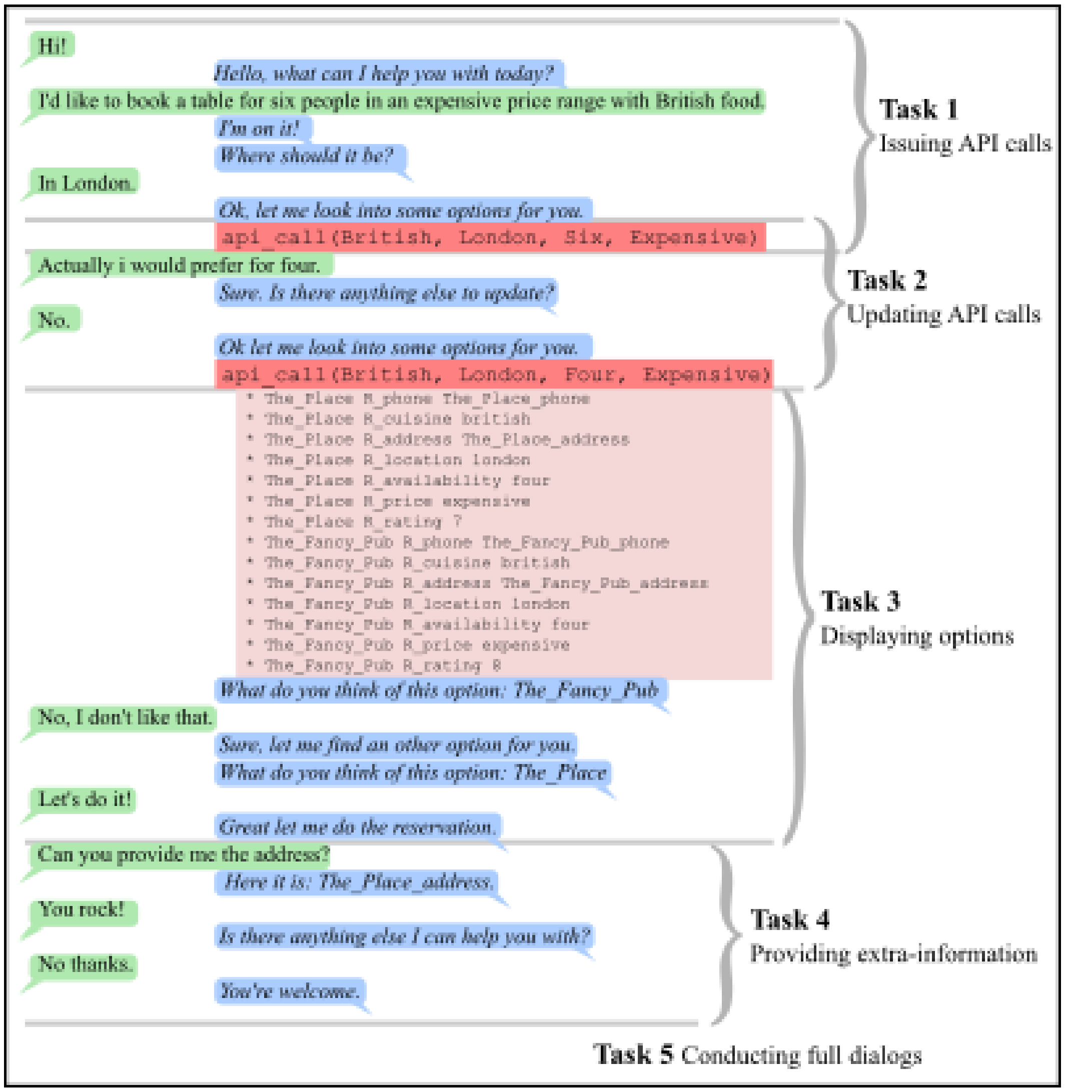
The KB can be queried using API calls, which return the list of facts related to the corresponding restaurants. Each query must contain a certain number of slots: a location, a type of cuisine, a price range, a party size, and possibly other required slots like dietary restriction, depending on the set used. Each data file has the same set of required slots for every dialog. A query can return facts concerning one, several or no restaurant (depending on the party size).

Using the KB, conversations are generated in the format shown in Figure 3. Each example is a dialog comprising utterances from a user and a bot, as well as API calls and the resulting facts. Dialogs are generated after creating a user request by sampling an entry for each of the required slots: e.g. the request in Figure 3 is [cuisine: British, location: London, party size: six, price range: expensive]. We use natural language patterns to create user and bot utterances. There are more patterns for the user than for the bot (the user can use several ways to say something, while the bot always uses the same way to make it deterministic). Those patterns are combined with the KB entities to form thousands of different utterances. We split types of cuisine and locations in half, and create two KBs, one with all facts about restaurants within the first halves and one with the rest. In Bordes *et al.* (2017), the two KBs had 4,200 facts and 600 restaurants each (5 types of cuisine $\times$ 5 locations $\times$ 3 price ranges $\times$ 8 ratings). The data provided here has been expanded to comprise more slots and thus yield many more restaurants, but the two KB still have disjoint sets of restaurants, locations, types of cuisine, phones and addresses, while sharing all other sets of values. We use one of the KBs to generate train and test dialogs, using only one of the extra slots in the queries. There are 4 sets of test dialogs: (1) one that uses the same KB as for the train dialogs, and the same set of slots in the queries; (2) one that uses the second KB (with disjoint sets of restaurants, locations, cuisines, phones and addresses), termed Out-Of-Vocabulary (OOV), but the same set of slots in the queries; (3) one that uses the same KB as for the train dialogs, but one additional slot for the queries; and (4) one that uses the second KB (OOV) and an additional required slot.

For training, systems have access to the training examples and both KBs. Evaluation is conducted on all four test sets. Beyond the intrinsic difficulty of each task, the challenge on the OOV test sets is for models to generalize to new entities (restaurants, locations and cuisine types) unseen in any training dialog – something natively impossible for embedding methods. Ideally, models could, for instance, leverage information coming from the entities of the same type seen during training.

We generate five datasets, one per task. Training sets are relatively small (10,000 examples) to create realistic learning conditions. The dialogs from the training and test sets are different, never being based on the same user requests. Thus, we test if models can generalize to new combinations of fields.

5 ILLUSTRATION OF TASKS



Hi!

Hello, what can I help you with today?

I'd like to book a table for six people in an expensive price range with British food.

I'm on it!

Where should it be?

In London.

Ok, let me look into some options for you.

```
api_call(British, London, Six, Expensive)
```

Actually i would prefer for four.

Sure. Is there anything else to update?

No.

Ok let me look into some options for you.

```
api_call(British, London, Four, Expensive)
```

```
* The_Place R_phone The_Place_phone
* The_Place R_cuisine british
* The_Place R_address The_Place_address
* The_Place R_location london
* The_Place R_availability four
* The_Place R_price expensive
* The_Place R_rating 7
* The_Fancy_Pub R_phone The_Fancy_Pub_phone
* The_Fancy_Pub R_cuisine british
* The_Fancy_Pub R_address The_Fancy_Pub_address
* The_Fancy_Pub R_location london
* The_Fancy_Pub R_availability four
* The_Fancy_Pub R_price expensive
* The_Fancy_Pub R_rating 8
```

What do you think of this option: The_Fancy_Pub

No, I don't like that.

Sure, let me find an other option for you.

What do you think of this option: The_Place

Let's do it!

Great let me do the reservation.

Can you provide me the address?

Here it is: The_Place_address.

You rock!

Is there anything else I can help you with?

No thanks.

You're welcome.

Figure 3: **Goal-oriented dialog tasks.** A user (in green) chats with a bot (in blue) to book a table at a restaurant. Models must predict bot utterances and API calls (in dark red). Task 1 tests the capacity of interpreting a request and asking the right questions to issue an API call. Task 2 checks the ability to modify an API call. Task 3 and 4 test the capacity of using outputs from an API call (in light red) to propose options (sorted by rating) and to provide extra-information. Task 5 combines everything.

REFERENCES

Bordes, A., Boureau, Y.-L., and Weston, J. (2017). earning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Lowe, R., Serban, I. V., Noseworthy, M., Charlin, L., and Pineau, J. (2016). On the evaluation of dialogue systems with next utterance classification. *arXiv preprint arXiv:1605.05414*.