Universität des Saarlandes
Department Informatik
Computer Linguistics

End-to-End Dialogue System Implementation

Seminar Paper

A Network-based End-to-End Trainable Task-oriented Dialogue System

Yusser Al Ghussin*
Matr.Nr. 7020727
yual00001@uni-saarland.de

March 21, 2022

Abstract

in this paper we will describe our implementation of the paper "A Network-based End-to-End Trainable Task-oriented Dialogue System" using python3 and Pytorch Library. the system in consistent of five components: Intent Network, Belief Tracker, Database Operator, Policy Network and Generation network. the project is on GitHub `https://github.com/Yusser95/N2N-Dialogue-System`.

Contents

1 Introduction

In general, sequence to sequence learning [8] has inspired several efforts to build end-to-end trainable, non-task-oriented conversational systems [10, 4, 2]. This family of approaches treats dialogue as a source to target sequence transduction problem, applying an encoder network [3] to encode a user query into a distributed vector representing its semantics, which then conditions a decoder network to generate each system response. These models typically require a large amount of data to train. They allow the creation of effective chatbot type systems but they lack any capability for supporting domain specific tasks, for example, being able to interact with databases [7, 6] and aggregate useful information into their responses.

Building a task-oriented dialogue system such as a hotel booking or a technical support service is difficult because it is application-specific and there is usually limited availability of training data.

This work propose a neural network-based model for task-oriented dialogue systems: the model is end-to-end trainable1 but still modularly connected; it does not directly model the user goal, but nevertheless, it still learns to accomplish the required task by providing relevant and appropriate responses at each turn; it has an explicit representation of database (DB) attributes (slot-value pairs) which it uses to achieve a high task success rate, but has a distributed representation of user intent. to allow ambiguous inputs; and it uses delexicalisation (Delexicalisation: we replaced slots and values by generic tokens e.g. keywords like Chinese or Indian are replaced by v.food in Figure 1 to allow weight sharing.) and a weight tying strategy [5] to reduce the data required to train the model, but still maintains a high degree of freedom should larger amounts of data become available. We show that the proposed model performs a given task very competitively across several metrics when trained on only a few hundred dialogues.

We model task-oriented dialog as a multi-task sequence learning problem, with components for encoding user input, tracking belief state, processing Database results, and generating system responses. The model architecture is as shown in Figure 1. Sequence of turns in a dialog is encoded using LSTM recurrent neural networks. Conditioning on the dialog history, state of the conversation is maintained in the LSTM state. The LSTM state vector is used to generate: (1) a skeletal sentence structure by selecting from a list of delexicalised system response candidates, (2) a probability distribution of values for each slot in belief tracker, and (3) a pointer to an entity in the retrieved KB results that matches the user's query. The final system response is generated by replacing the delexicalised tokens with the predicted slot values and entity attribute values. Each model component is described in detail in below sections.

The remainder of this article is organized as follows: Section 2 will review background of the existing implementations and libraries. We first introduce the existing implementations of this paper, the libraries used and their limitations and problems, and then we briefly talk about libraries used for this implementations and pre-trained models. Section 3 will describe Experiment settings such as the

data-set and classes structure and evaluating the model. Section 4 will summarize the results. Finally, we conclude and discuss future directions in Section 5.

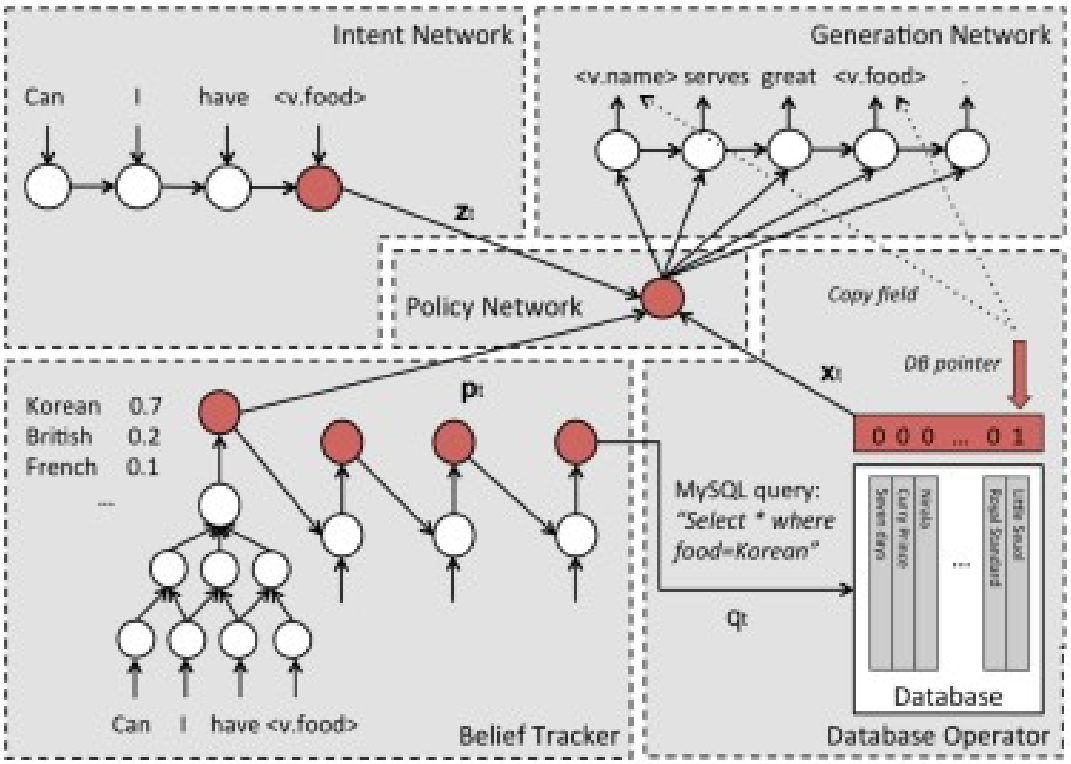

Figure 1: The proposed end-to-end trainable dialogue system framework

2 Background

in this section we will give a quick background on the related open-source implementation and the libraries we are using in our implementation.

2.1 Related Implementations

through research we found to open-source repositories on GitHub that contains implementation to the relevant paper "A Network-based End-to-End Trainable Task-oriented Dialogue System" [9]

the first repository is NNDial `https://github.com/shawnwun/NNDIAL` it is an implementation released by the author of the paper Tsung-Hsien (Shawn) Wen from Cambridge Dialogue Systems Group under Apache License 2.0. the implementation is written using python2 and used an old neural networks library called Theano. one of the advantages of this implementations that it had good documentation. we tried to rerun the code on windows machine but it gave an errors

related to the Theano Library version for windows when we tried to re run the experiment on Linux machine it gave errors related to the data.

The Second repository we found is dialog `https://github.com/edward-zhu/dialog` it is an implementation by Edward Zhu in 2018 it is written using python2 and pytorch library. this implementation is modified to have the advantages of our main paper and a second approach to end to end task oriented dialogue system in this paper "An End-to-End Trainable Neural Network Model with Belief Tracking for Task-Oriented Dialog" [1] this paper is very similar to our main paper bu it replaces the policy network and the generation network by a simple classifier to select a template from predefined templates defined for the task. this implementation suffers from bad documentation there were no instruction to rerun the code and the data thy were using and pre-trained model was messing and not mention how to get it.

we took advantage of the two repositories to rebuild the basic architectures using pytorch and python3 depending on the second repository and using the data, pre-trained embedding models and utils from the first repository.

2.2 Libraries

we depend in our implementation on two important Python libraries that are widely use and easy to install and work with Pytorch and NLTK.

PyTorch: is an open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab. It is free and open-source software

NLTK: The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

3 Experiment

The author

3.1 Data-set

the data set in our main paper is collected using Wizard-of-Oz (WOZ) paradigm the system was designed to assist users to find a restaurant in the Cambridge, UK area. There are three informable slots (food, pricerange, area) that users can use to constrain the search and six requestable slots (address, phone, postcode plus the three informable slots) that the user can ask a value for once a restaurant has been offered. There are 99 restaurants in the DB. 680 dialogue turns was collected. dataset is public available at: `https://www.repository.cam.ac.uk/handle/1810/260970`.

we used pre-trained GloVe 300d to get words embedding. the pre-trained model is publicly available at `https://nlp.stanford.edu/projects/glove/`

we used wor

3.2 Implementation

using Python3.9.9 and latest libraries version we were able to reconstruct the code from the related implementations and group the relevant classes into 9 python scripts (). which we will go into details what each of them contains.

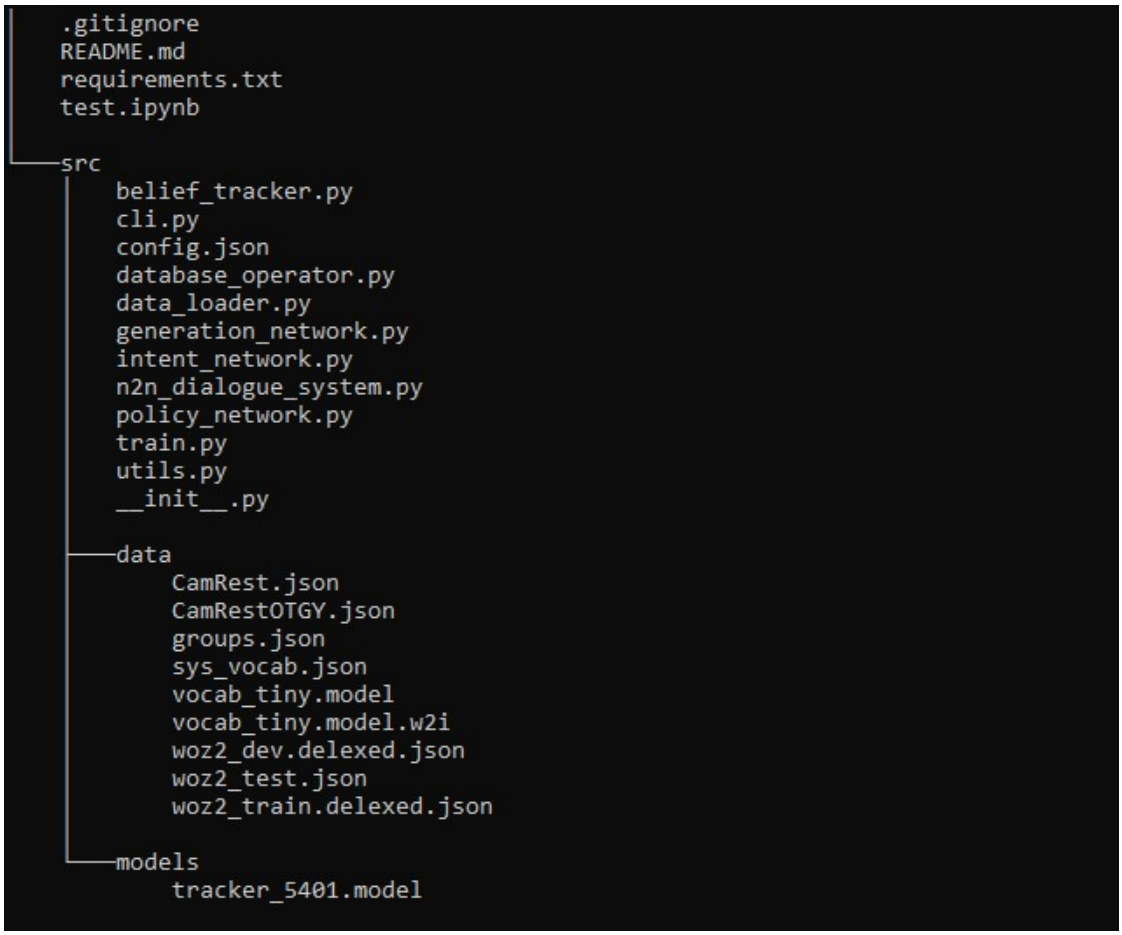

Figure 2: project files tree

- **intent_network.py** This script contains the following components:

  SentenceEncoder an one layer LSTM neural network that encode the pre-trained embeddings of the user query into a continuous dense vector that contains semantic information. and there is an option to train with attention but we didn't use in our experiment.

- **belief_tracker.py** This script contains the following components:

StateTracker an one layer LSTM neural network that maintains and adjusts the state of a conversation, such as user's goals, by accumulate evidence along the sequence of a dialog. the neural dialog model updates the probability distribution over candidate values for each slot type. For example, in our restaurant search domain, the model maintains a multinomial probability distribution over each of user's goals on restaurant area, food type, and price range. At turn k, the dialog-level LSTM updates its hidden state and use it to infer any updates on user's goals after taking in the user input encoding. Using each user input as new evidence, the task of a belief tracker is to maintain a multinomial distribution p over for each informable slots, and a binary distribution for each requestable slots. Each slot in the ontology has its own specialised tracker.

InformSlotTracker an one layer fully connected network that get slot value distribution from state at time t.

RequestSlotTracker an one layer fully connected network that get a request type activation (binary) state distribution from state at time t.

- **database_operator.py** This script contains the following components:

  KnowledgeBase this class provide api for knowledge retriving.

  $\boldsymbol{load}_k bthisfunctionloadstheknowledgebasefromjsonfiletoKnowledgeBaseclass.$

- ***policy_network.py*** *This script contains the following components:*

  PolicyNetwork *a two fully connected layers that concatenate the state tracker and sentence encoding and knowledge base one hot vector to produce action vector that represent the final sentence.*

- ***generation_network.py*** *This script contains the following components:*

  ConditionNet *a one fully connected layer that work as an attention mechanism instead of decoding responses directly from a static action vector. it goes into ConditionNet and then to the final Generator the paper called it an Attentive Generation Network.*

  Generator *is network consist of one embedding layer and an LSTM layer and final fully connected layer it works as decoder to generate the final sequence word by word.*

  load_generator_model *is a function to load the ConditionNet and Generator networks with appropriate dimensions.*

- ***n2n_dialogue_system.py*** *This script connect all the network components to get on model.*

- ***utils.py*** *contain common functions that are used in different parts of the system like padding and tekonizing.*

- ***data_loader.py*** *this script contains functions to load the training data, knowledge base and embeddings.*
- ***train.py*** *this script load and train, evaluate and test the model it runs according to the config.json file in the src/ directory as we can see in Figure 2*
- ***cli.py*** *command line interface script load the model specfied in the config.json file in our example models/tracker_5401.model and wait for user input to start chatting with the system in a continuous loop.*

there is a test.ipynb notebook at the ROOT directory of the project that one can use to start training and test the cli on predefined list of sentences.

3.3 Training

we trained the model on desktop with single rayzen9 - CPU for 200000 iteration it took around two days to finish training. we used one layer LSTM in the models with hidden size of 300. we used RMSprop or Root Mean Squared Propagation optimizer on the state and slote tracking results with learning rate 0.0001.

4 Results

we got state and slot tracking accuracy of 99.7 and we checked the results manually on the test results. we can see in Figure 3 an example of the system cli results.

5 Conclusion

this is very basic implementation of the paper to model the basic functionality. there is a lot of improvements that can be implemented like changing the LSTM layers to GRU and make the model train faster or make the model trainable on cuda GPU. test the system on different data sets.

```
********************
usr:  Hello, I am looking for an expensive restaurant that serves Australian food.
********************
160
160
[['food', 'australian'], ['pricerange', 'expensive']]
[]
sys:  <sos> there are no expensive australian restaurants. <eos>
********************
usr:  What about British food?
********************
198
198
[['food', 'british']]
['saint johns chop house', 'midsummer house restaurant', 'the cambridge chop house', 'the copper
kettle', 'travellers rest', 'cotto', 'restaurant one seven', 'fitzbillies restaurant', 'the oak
bistro', 'graffiti', 'grafton hotel restaurant']
{'address': 'Midsummer Common', 'area': 'centre', 'food': 'british', 'location': '52.21251,0.12774',
'phone': '01223 369299', 'pricerange': 'expensive', 'postcode': 'C.B 4, 1 H.A', 'type': 'restaurant',
'id': '508', 'name': 'midsummer house restaurant'}
sys:  <sos> midsummer house restaurant is in the centre area , its phone is 01223 369299 . their
address is Midsummer Common <eos>
```

Figure 3: system results on test data

References

[1] Ian Lane Bing Liu. An end-to-end trainable neural network model with belief tracking for task-oriented dialog. *arXiv*, preprint:1708.05956, 2017.

[2] Yoshua Bengio Aaron C. Courville Iulian Vlad Serban, Alessandro Sordoni and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *arXiv*, preprint:1507.04808, 2015b.

[3] Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *EMNLP*, page 1724–1734, 2014.

[4] Zhengdong Lu Lifeng Shang and Hang Li. Neural responding machine for short-text conversation. *ACL*, Beijing, China, July:1577–1586, 2015.

[5] Blaise Thomson Matthew Henderson and Steve Young. Word-based dialog state tracking with recurrent neural networks. *SIGDIAL*, page 292–299, 2014.

[6] Hang Li Pengcheng Yin, Zhengdong Lu and Ben Kao. Neural enquirer: Learning to query tables. *arXiv*, preprint:1512.00965:2440– 2448, 2015.

[7] Jason Weston Sainbayar Sukhbaatar, arthur szlam and Rob Fergus. Endtoend memory networks. *NIPS*, page 2440– 2448, 2015.

[8] Oriol Vinyals Sutskever et al.2014] Ilya Sutskever and Quoc V. Le. Sequence to sequence learning with neural networks. *NIPS*, p, 2014.

[9] Nikola Mrkšic´ Milica Gašic´ Lina M. Rojas-Barahona Pei-Hao Su Stefan Ultes Tsung-Hsien Wen, David Vandyke1 and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv*, preprint:1604.04562, 2017.

[10] Oriol Vinyals and Quoc V. Le. A neural conversational model. *ICML*, Deep Learning Workshop, Lille, France, 2015.