Creating a baseline

Adrian Tysnes

1 The quest for heuristics

After discussing the previous entry with my supervisor, it became clear to me that I should approach the task differently. Creating these labeling functions out of statistical analysis of the data is interesting, but can be done in much more advanced ways if it can be tested using the ground truth. The problem lies in that the only reason that I knew that it performed decently, and the only way that I would be able to potentially improve it, is through the access of the ground truth. I am therefore approaching the problem in a different way in this section.

1.1 Structural heuristics

The easiest thing to do is to just catch the structural content that is easy to identify. The majority of the slots should be identifiable with some regular expressions. These can be easily generated using the ontology that comes with the datasets. Some attempts were made to add a word or two to the expression - for example using *italian food* instead of just *italian*, but a quick eye - test seemingly shows that it is less effective. Some extra tinkering with the expressions is needed, for example the value *north* is used for detecting the *area* slot, but a lot of types of food start with the word *north*, and must not be captured.

1.2 Other heuristics

It is also useful for the sake of the implementation to see if some extra voting functions can play a part in the baseline, if for nothing else to simulate the effect that the voting aggregation functionality should have in the future. I ended up implementing two additional voters for the baseline:

- Turn-voter: The idea here is that some slots will occur more frequently at the beginning of the dialogue, and some will more often occur at the end.

For the baseline, the only functionality is that it gives a negative vote for specification of food types for the last utterance in a dialogue, unless the dialogue is 2 or fewer user utterances long, and it will vote negatively for contact information early in the dialogue, and vote positively for it later on.

- Questionnaire-voter: This has seemingmly very little effect as of now, but the idea was that some information could be found in the questionnaire the user potentially could answer to rate the dialogue with the system. Casts very few votes as of now.

1.3 The vote center

These voting functions are then collected in a *vote center*, that iterates through the dialogues and collects the votes for all the different voter types. The potential values that the functions of these voter types are

- Some slot value: Some vote function found a slot, and a corresponding slot value.
- 1: Lets the vote center know that one of their functions found a match for a slot, but no specific value for that slot.
- 0: A voter has a function for a slot, but did not detect it.
- -1: A voter has a function that found negative evidence for the existance of a slot.

There are many ways that these voting values could be aggregated, but for the baseline, the vote center sums up the votes, with *some slot value* having absolute presedence over the others.

2 The model

The architecture of the model is taken from *Liu et. al.*[2], as they have had success testing their model on the *DSTC 2* dataset. There are some things that could probably be improved with some slight adjustments, but it can be useful to reproduce the work of others. The specific dimensions used by *Liu et. al.* were somewhat vague at some points, so some of the information on that end might be a bit off. That will be discussed in future reports.

2.1 Overview

In this section, I will quickly give an overview of the architecture. The input to the model is a tuple of

- A tuple of the user utterance and its length: As of now, because my implementation is fully done in PyTorch using LSTMs, the utterance length is needed for packing the padded sequences.
- A tuple of the previous system utterance and its length: The previous system utterance is used together with the user utterance and the current dialogue state to generate a new dialogue state.
- The dialogue number: Because the model is learning on an utterance level, it needs to know when to reset the dialogue state for new dialogues.
- The slot labels from the voting process

The utterances are transformed in the embedding layer, using embeddings trained on a Wikipedia dump of february 2017 from NLPL [1]. The embeddings are then packed and seperately ran through a bidirectional, 3-layer LSTM, and are projected to two 400 dimension vectors. The two vectors are then concatenated, and projected onto a 200 dimensional vector, where a ReLU-function is applied. This vector and the current dialogue state (or a vector filled with 0s in the case that it is the first user utterance) are concatenated, and passed to a uni-directional, 2 layer LSTM, after a 0.5 dropout is applied. The new dialogue state is projected onto different dimensions to create a probability distribution for the different slots, and the ReLU-function is applied again.

3 Result

Testing the accuracy of the output is not trivial, and can be done in several ways. Since I do have the true labels of the dataset, I am going to test the results in two ways:

- With no labels: Without labels, the best way I can measure the performance is to vote on the development partition, and use those as the true labels. The strength of this method is that a different partition of the dataset could be used for training and development each time, leading to less impact by edge cases.
- With *some* labels: Another way that would be interesting to see is how these results stand to the actual labels. This would mean that a small set

of labeled data would be required, but the training process would still be performed without it. This would mean that the dataset partition would be static, which could cause some problems particularly for the *food* slot, as some values only occur a few times and could potentially not be seen at all in the training data.

Testing with both the results from the voting process and the actual labels also gives an idea of how good the labels are - or at least if they are good enough for the task at hand. For now, I have produced the proportional accuracy for all utterances - that is, how many total slots did it manage to get right, and how many did it get wrong, and the binary accuracy, how many utterances did it get all the slots right on. I have also produced the accuracy for the dialogue goal labels - but I have not taken all slots into consideration yet, so this number is a bit off for the labeled set.

Measure	Voting set	Labeled set
Proportional utterance	63%	54%
Proportional dialogue	69%	63%
Binary utterance	47%	45%
Binary Dialogue	33%	32%

The voting set is surprisingly close to the labeled set, in all instances but the proportional utterance accuracy. Next months report will include some more in-depth error analysis of the baseline.

References

[1] Murhaf Fares et al. "Word vectors, reuse, and replicability: Towards a community repository of large-text resources". In: *Proceedings of the 21st Nordic Conference on Computational Linguistics* (2017).

[2] Bing Liu et al. *Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems*. 2018. arXiv: `1804.06512 [cs.CL]`.