Abstract

In spoken language understanding, getting manually labeled data such as domain, intent and slot labels is usually required for training classifiers. Starting with some manually labeled data, we propose a data generation approach to augment the training set with synthetic data sampled from a joint distribution between an input query and an output label. We propose using a recurrent neural network to model the joint distribution and sample synthetic data for classifier training. Evaluated on ATIS and live logs of Cortana, a Microsoft voice personal assistant, we showed consistent performance improvement on domain classification, intent classification, and slot tagging on multiple languages.

Index Terms: labeled data generation, recurrent neural network, spoken language understanding

1. Introduction

Recently, recurrent neural networks have demonstrated good performance in various natural language processing tasks such as language modeling (LM) \([1, 2]\), and spoken language understanding \([3, 4, 5, 6, 7]\). In language modeling, the main focus is to obtain a good model \(P(W)\) to describe how a word sequence \(W\) is generated. In spoken language understanding, the main focus is to obtain a good model \(P(Y|W)\) to classify \(W\) into different categories \(Y\), where \(Y\) can be a single discrete variable as in domain/intent classification, or a sequential variable as in slot tagging. Discriminative modeling does not care how the input data are generated. On the other hand, a language model only concerns about word prediction without involving any target output labels.

Manual annotation can be costly and usually limited. Usually, performance of a classifier improves with more manually labeled data. From this point of view, if we know how the labeled data are generated, i.e. the joint distribution \(P(W, Y; \Lambda)\) between \(W\) and \(Y\) where \(\Lambda\) denotes a model parameter, then we can sample labeled data from such distribution and train classifiers. In this paper, we attempt to learn the joint distribution from data. The main questions we want to explore are:

- Can we model the joint distribution \(P(W, Y; \Lambda)\) and sample reasonable synthetic data from such distribution?
- Can the synthetic data help?

We propose using a recurrent neural network (RNN) to generate labeled data for spoken language understanding. Typically, spoken language understanding involves three tasks: (1) domain classification, (2) intent classification, and (3) slot tagging. For domain and intent classification, the target label \(Y\) is a discrete variable. For slot tagging, the target label \(Y\) is a slot sequence. In the input layer for domain and intent classification, we use the target label as an auxiliary input to RNN. For slot tagging, an RNN is employed without auxiliary input. The output softmax layers of RNN for the three tasks are identical. In particular, each training word-label pair (word:label) is considered as an output vocabulary. This has two advantages. Firstly, a word-label pair will always be generated conforming to the manually annotated training set. In other words, the data sampling process will only associate labels to words that are observed in the training data. This is crucial as consistency of generated labels will affect discriminative classifier training. Secondly, by treating a word-label pair as an output vocabulary, we easily filter out ill-formed word sequences subject to the labeling constraints. For domain and intent classification, switching the domain and intent labels are not allowed within a query, assuming that single domain or intent is involved in the input query. The filtering process will further ensure that the final synthetic queries are reasonable so that they can be mixed with the original training set. The expanded training set can be applied to tasks directly without code changes.

Using RNN for data generation has been investigated. \([8]\) uses Long Short-term Memory (LSTM) RNN to generate text and handwriting. \([9]\) trains an RNN language model and samples text from it. The generated text is then used to train an N-gram language model to approximate an RNN language model. Our innovation is to generate labeled data and apply them to improve spoken language understanding by expanding the labeled training data.

The paper is organized as follows: In Section 2, we review the basic tasks in spoken language understanding, followed by the proposed approach in Section 3. We describe experiments and results in Section 4. We provide conclusions in Section 5.

2. Overview of spoken language understanding

Given a query \(W_1^N = w_1^N \ldots w_t^N \ldots w_M^N\), one way to approach spoken language understanding is to decompose it into three tasks: domain classification, intent classification, and slot tagging. For details, please refer to \([10, 11]\) for introduction. Domain classification is to classify a query into a category (e.g. movie, flight). Within a domain, there can be multiple intents. In a movie domain, buying a ticket and browsing a movie preview can be considered as different intents. Slot tagging is to extract useful entities from a query, such as movie names, place names etc. Mathematically, we want to predict domain \(d\), intent \(t\), and slot sequence \(y_1^N\) given a query \(W_1^N\):

\[
P(y_1^N, t, d|W_1^N) = P(y_1^N|W_1^N, t, d) \cdot P(t|W_1^N, d) \cdot P(d|W_1^N)\]

\[
\approx P(y_1^N|W_1^N, d) \cdot P(t|W_1^N, d) \cdot P(d|W_1^N)
\]

There are various approaches to model the three terms above, such as support vector machines for domain and intent classification, recurrent neural network for contextual domain classification \([12]\); conditional random field \([13]\), recurrent neural
3. Generating labeled data using RNN

Given a set of labeled data \( \{W_i, Y_i\} \) where \( W_i \) denotes a word sequence and \( Y_i \) is a target label depending on a task. For slot tagging, \( Y_i \) is a sequence of tags. For domain and intent classification, \( Y_i \) is a discrete category. With success in using RNN for language modeling that models \( P(W; \lambda) \), we employ RNN to model a joint probability distribution \( P(W, Y; \lambda) \). With enough labeled data, we hope that RNN can capture the structure of a word sequence and the target label so that RNN can generate valid unseen labeled data. In summary, the proposed procedure is outlined as follows:

1. Estimate \( P(W, Y; \lambda) \) from manually labeled data.
2. Sample \( \{W_i, Y_i\} \) from \( P(W_i, Y_i; \lambda) \).
3. Augment Filter(\( \{W_i, Y_i\} \)) into the training set.
4. Train a classifier using the expanded training set.

3.1. Data generation for slot tagging

Since the output label is a slot sequence, we attach each word token with a position-specific slot label. We use \( (B, I, E) \) to encode the beginning, middle, and ending positions of a slot label. For a single word token like “Denver”, it becomes “Denver:CITY”. For a two-word tokens like “San Francisco”, it becomes “San:B_CITY Francisco:E_CITY”. For a token with three or more words like “San Francisco International Airport”, it becomes “San:B_AIRPORT Francisco:L_AIRPORT International:L_AIRPORT Airport:E_AIRPORT”. A joint word-slot token is treated as a vocabulary. Then we train an RNN language model per domain to model \( P(W, Y; \Lambda_d) = \prod_{i=1}^{\infty} P((w,y)_i|(w,y)_{i-1}, h_{i-1}; \Lambda_d) \) where \( \Lambda_d \) is the RNN model parameter of domain \( d \). Figure 1 shows the RNN architecture. During data sampling, a sampled query that contains inconsistent slot labels will be discarded. In particular, we first derive possible tag bigrams from the tagged training data. If a tag bigram in a sampled query does not exist in the training set, the sampled query will be removed.

3.2. Data generation for domain classification

Ideally when labeled data are sufficient, we train an RNN language model per domain, i.e. \( P(W|y) \) where \( y \) is the domain index. However, insufficient labeled training data is usually the case. To alleviate this issue, we pool training data from all domain for RNN training. Figure 2 shows the RNN architecture for query generation with auxiliary input. In query generation, we want to take the domain prior information into consideration. Using RNN without auxiliary domain input, it is hard to guarantee that the sampled queries would follow the same prior label distribution in the training data. To tackle this issue, auxiliary feature is used to provide the target domain label during training and query generation so that during query generation, we can control the label distribution in the sampled queries. To enable data filtering, we attach the domain label to each word. “Denver:FLIGHT” and “Denver:WEATHER” are individual vocabulary corresponding to the flight and weather domains. Mathematically, the RNN model is \( P((w,y)_i|(w,y)_{i-1}, h_{i-1}, y; \lambda) \). During data sampling, we generate domain-specific queries according to a prior distribution \( P(y) \) estimated from the training data. The prior distribution on the synthetic data should match the prior distribution estimated from the training data. If there is a switch of domain within a sampled query (e.g. fly:FLIGHT to:FLIGHT Denver:WEATHER), this query is discarded to ensure the labels are consistent.

3.3. Data generation for intent classification

A domain may consist of multiple intents. To generate intent labels, we follow a similar procedure as in Section 3.2. We employ intent label as auxiliary input and we augment an intent label to each word token in a training data. We train an RNN model per domain: \( P((w,y)_i|(w,y)_{i-1}, h_{i-1}, y; \lambda_d) \) where \( \lambda_d \) is the RNN model parameter of domain \( d \), and \( y \) denotes an intent label.

In data filtering, We only consider a query with one intent. Therefore, if there is a switch of intent within a sampled query (e.g. fly:buyTicket to:buyTicket Denver:checkSchedule), this query is discarded to ensure the labels are consistent. If slot labels are available, we can use the beginning, middle, and ending positions of a slot label as constraints to filter out ill-formed queries as described in 3.1.
interpretations of the sentences. The words in each sentence were \textit{labeled} with semantic categories. The training data consisted of 4978 sentences and 56590 words. Test data consisted of 893 sentences and 9198 words. The number of distinct slot labels were 128, including the common null label; there are a total of 25509 non-null slot occurrences in the training and testing data respectively. These were the standard settings used as a benchmark in the community.

For data generation for slot tagging, we employed the RNNLM toolkit. The number of hidden units was set to 50 with direct connections. For domain and intent data generation, we modified the toolkit to support auxiliary inputs. The number of hidden units was set to 100 with direct connections. All RNN models were trained with backpropagation through time using the flag “-bpt 4 -bpt-block 10”. We randomly held out 10\% of the training data for validation.

4.2. Slot tagging result

We employed conditional random field \cite{Lafferty.2001} for initial experiments using CRF++ \cite{D_tran.2007}. All models were built using the L2 regularization with $C$ set to 1 as default. All models were trained until convergence using the default stopping criterion. We only employed lexical features, including word unigram, word bigram and slot transitions. No pruning of features were performed. The training and test data were tagged using the BIO format. Table 1 shows the F1 scores of models trained on various training data. With 100k RNN-generated queries added for training, we achieved 1.45 absolute improvement on F1 score compared to the baseline trained only on the manual training set. RNN can generate queries that appeared in the training set. The mix between the newly generated queries and the training queries may not be optimal. Therefore, we removed the training queries from the synthetic data set and repeated the training set twenty times to ensure that the contribution of the manual training data was significant. In this case, the resulting synthetic queries covered more variations that were unseen in the training data. In the original training set, there were only 6467 word bigrams, while the synthetic training set contained 30784 word bigrams. Retraining using the filtered data set yielded additional 0.33 on F1 score. In summary, the RNN-based synthetic data may provide complementary information for training the CRF tagger. Sequential RNN tagger performed the best compared with CRF tagger trained with synthetic data. Sequential RNN tagger was not trained with the synthetic data because they hurt performance slightly.

For comparison, we used the same RNN training set to train a 4-gram language model with word tokens attached with slot labels. Then we generated the same amount of data and followed the same process to filter out ill-formed queries. In this case, 4-gram LM was less effective in generating labeled queries and thus yielded worse results compared to RNN. This may be due to the inability to capture the structure of a query using the limited training queries. Looking at the RNN synthetic data, words of the same slot type were shuffled around different queries, resulting in better generalization effect. We were curious to know what the popular labeled word sequences were generated in the synthetic data. We replaced word tokens with slot labels and counted the top word sequences sorted with frequencies as shown in Table 2. In fact, the top-three patterns were actually the prefix of some training patterns. We also discovered new carrier phrase patterns as shown in Table 3, showing that RNN captured meaningful carrier phrase variations. For instance, the phrase “show me all flights” was very common in the training data. However, the phrase never followed with “leaving” in the training data. Instead, “from” was usually the following word. “flights and fares” and “information on a flight from” were unseen in the training data.

### 4.3. Microsoft Cortana data set

We evaluated our approach on Microsoft internal Cortana live log data set on Chinese and Italian. Table 4 shows the total data statistics in Cortana live log for domain and intent classification on Chinese and Italian. There were eight domains including communication, reminder and weather. The average number of intent per domain was sixteen. Table 5 and 6 show the data statistics for slot tagging on Chinese and Italian respectively. For slot tagging experiments on Italian, we applied domain-adapted N-best machine translation \cite{Bougares.2012} to translate English queries into Italian as additional labeled training data to relieve the data sparsity issue in a new locale. We found that the MT data were useful for slot tagging although translation quality may be poor. The vocabulary coverage from the MT data was also richer compared to the live training log. Therefore, we employed MT and live training data for RNN training. To emphasize that RNN would work well on live log, we first processed the MT data followed by live data during stochastic gradient updates. We also repeated the live data multiple times to make the contribution of the live data more significant. On the other hand, we did not apply the English-to-Chinese MT data since they hurt the slot tagging performance. We had more live training data on Chinese than Italian. For training semi-Markov CRF slot taggers \cite{Yang.2010}, we followed the same heuristics as in Section 4.2 to replicate the training queries multiple times and mixed them with the RNN synthetic labeled data.

### Table 1: Slot tagging F1 scores on the ATIS test set.

<table>
<thead>
<tr>
<th>Data setup</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>90.94</td>
</tr>
<tr>
<td>Train + 100k(4-gram LM)</td>
<td>91.27</td>
</tr>
<tr>
<td>Train + 100k</td>
<td>92.39</td>
</tr>
<tr>
<td>Train x 20 + 100k(exclude train)</td>
<td>92.72</td>
</tr>
<tr>
<td>Sequential RNN</td>
<td>94.81</td>
</tr>
</tbody>
</table>

### Table 2: Unseen labeled queries generated from RNN. The highlighted words are novel N-grams generated by RNN.

- show me all flights leaving ...
- show me the flights and fares from ...
- i would like information on a flight from ...

### Table 3: Unseen carrier phrases generated from RNN. The highlighted words are novel N-grams generated by RNN.

- flight from ...
- i would like information on a flight from ...
- show me all flights leaving ...
- show me the flights and fares from ...

### Table 4: Total number of tokens of Cortana Chinese and Italian data set for domain and intent classification.

<table>
<thead>
<tr>
<th>Task</th>
<th>Chinese</th>
<th>Italian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>105k</td>
<td>237k</td>
</tr>
<tr>
<td>Test</td>
<td>21k</td>
<td>11k</td>
</tr>
</tbody>
</table>
Table 5: Number of word tokens of Cortana Chinese data set for slot tagging on reminder domain.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reminder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live train</td>
<td>248k</td>
</tr>
<tr>
<td>Test</td>
<td>43k</td>
</tr>
<tr>
<td># slots</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 6: Number of tokens of Cortana Italian data set for slot tagging on communication, reminder, and weather domain.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Comm</th>
<th>Reminder</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live train</td>
<td>43k</td>
<td>38k</td>
<td>26k</td>
</tr>
<tr>
<td>MT train</td>
<td>683k</td>
<td>1.6M</td>
<td>817k</td>
</tr>
<tr>
<td>test</td>
<td>2k</td>
<td>2k</td>
<td>1k</td>
</tr>
<tr>
<td># slots</td>
<td>18</td>
<td>16</td>
<td>9</td>
</tr>
</tbody>
</table>

4.4. Slot tagging result

Table 7 shows the Chinese slot tagging results. Our approach improved the F1 score by 0.39 compared to the baseline trained on live log data only. Increasing the synthetic data from 1M to 5M query samples only yielded slight improvement. Therefore, we used 1M sampled query on Italian to avoid processing a lot of queries. Table 8 shows the Italian slot tagging results. We observed consistent gain on the F1 score on multiple domains. In particular, we achieved 1.84, 2.87 and 0.72 gain on the F1 score on communication, reminder, and weather domains respectively. Sequential RNN tagger [4] yielded good performance compared to the semi-Markov CRF baseline as shown in Table 7 and Table 8. When synthetic data were added, they helped slightly on training the Chinese RNN tagger. This is reasonable because they were actually generated from RNNLM. In this sense, the gain from adding the synthetic data may come from the “combination” effect between CRF and RNN at the data level.

4.5. Intent classification result

The vocabulary size of each domain was not very large for language model training. Therefore, we used smaller class size (1 to 10) for RNN training. The intent classification error rates of using generated data on Chinese reminder domain and Italian communication, reminder and weather domain were reported in Table 9 and Table 10, respectively. In Chinese reminder domain, by adding 2M tokens to original training data, we reduced classification error rate from 2.7% to 1.2%. However, adding more data did not translate into better classification performance. When the generated tokens were increased to 10M, the improvement over the baseline model got smaller. In Italian Cortana data set, we also observed the same trend. Therefore, adding the right amount of generated tokens was crucial.

4.6. Domain classification result

Different from RNN for intent data generation, we set the output class size to 100 in RNN training. The reason was that the vocabulary size from all domains was larger. Similar to intent classification, we observed improvement in classification performance. By adding a suitable size of generated data, we obtained significant improvement. However, increasing the amount of synthetic data further to 20M tokens did not help as shown in Table 12. The trend was in line with intent classification. With more synthetic data, the chance of introducing noise into training may increase. In the best case, we reduced the classification error rate by 40% relative compared to the baseline on both languages.

5. Conclusions

In this paper, we propose using recurrent neural network to generate synthetic labeled data to improve spoken language understanding. Our approach employs data augmentation so that it can be applied easily to existing systems without code changes. We summarize the following heuristics when applying our approach: (1) A good data filtering method to remove ill-formed queries; (2) A good mixing proportion between the training and the synthetic data; (3) The right amount of synthetic data to be generated. In the future, we will investigate applying the RNN-based labeled data generation for speech recognition.

6. Acknowledgments

We would like to thank Thomas Pan for discussion.
7. References


