Contextual Slot Carryover for Disparate Schemas

Chetan Naik, Arpit Gupta, Hancheng Ge, Lambert Mathias, Ruhi Sarikaya

Amazon Alexa Machine Learning
{chetnaik, arpgup, ghanche, mathiasl, rsarikay}@amazon.com

Abstract

In the slot-filling paradigm, where a user can refer back to slots in the context during a conversation, the goal of the contextual understanding system is to resolve the referring expressions to the appropriate slots in the context. In large-scale multi-domain systems, this presents two challenges - scaling to a very large and potentially unbounded set of slot values, and dealing with diverse schemas. We present a neural network architecture that addresses the slot value scalability challenge by reformulating the contextual interpretation as a decision to carryover a slot from a set of possible candidates. To deal with heterogeneous schemas, we introduce a simple data-driven method for transforming the candidate slots. Our experiments show that our approach can scale to multiple domains and provides competitive results over a strong baseline.

Index Terms: Spoken Language Understanding, Dialog State Tracking, Anaphora Resolution, Contextual Understanding.

1. Introduction

Slot-filling based spoken language understanding (SLU) system is often a central component [1] in conversational systems. A major challenge in the slot-filling paradigm is to handle conversational context, where a user utterance can refer back to a set of slots implicitly or explicitly.

Traditionally, the contextual interpretation of slots has been cast as a coreference resolution problem. There is a rich body of work on coreference resolution for written text, which rely on clustering[2, 3, 4] or ranking mention pairs [5, 6]. These have been extended to spoken dialog, by adding discourse specific features [7, 8, 9, 10, 11, 12]. However, most of these approaches follow a pipelined model of mention detection followed by coreference resolution; where linguistic features, syntax and discourse features are usually applied. In contrast, our proposed formulation does not rely on explicit linguistic features such as gender and type agreement, which are hard to acquire across languages. Furthermore, we can generalize the solution to sub-tasks such as zero pronouns naturally, as we don’t have to explicitly identify the anaphoric mentions.

Another challenge is dealing with multi-domain language understanding systems, where each domain has its own schema to represent slots and intents. Domains are developed mostly independently and hence do not share a common schema. Also, dialog assistants are now being extended by community developers using services such as Google DialogFlow1 or Alexa Skills Kit [13]. Domains developed by external developers are completely outside of a central repository of domain and slots, hence no assumption can be made about their schema. The lack of shared schema makes it hard to maintain contextual slots across domain boundaries. Table 1 shows an example conversation a user may have with such a dialog system. In this example user interacts with three domain: Weather, LocalSearch and Traffic. Each domain has its own schema to represents slots. The user starts off by asking weather in a city, follows it up with question about restaurant that serves Mexican cuisine, finally she asks about directions to the restaurant. In this example, we showcase the challenges of multi-domain system. Weather and LocalSearch use different schema to store information about the location, using slot keys WeatherLocation and City respectively. For carrying conversation across domains from U1 to U2, we need to be able to transform the slot [WeatherLocation: san francisco] to [City: san francisco], without having access to a common schema. Even within a domain there could be issues of diverse schemas - U2 and V2, the domain chooses to have different schemas to represent its user and system turns. To make the task more complex, some domains choose to represent all its slots as just a generic label Entity.

There has been work on improving semantic frame error rate for current turn by leveraging context turns by encoding dialog states. [14] compare various approaches encoding context. [15] describe a memory network architecture for knowledge carryover, [16] add semantic context from the frame. [17] use context features for domain classification. Our work differs from this body of literature; we keep the system for semantic frame prediction fixed and explore methods to explicitly add slots from previous turn. While the previous work assumes existence of a dialog manager which can be used to keep track of entities from previous turns, we make no such assumptions.

A closely related task is dialog state tracking [11, 18, 19], where the system has to predict a set of slot-value pairs which matches the contents of the current segment. Usually, state trackers produce a distribution over all possible slot-value pairs; this does not scale for open-ended slot values (such as Date or Time), as well as slots whose values are constantly being updated (such as Songs, Movies). Our approach avoids this by reformulating the tracking problem as a carryover decision for the current turn. More closely related is frame-tracking [20], which was introduced as an extension to state tracking, here the slots need to be tracked over multiple frames and maintain reference to original frame. A key difference here is that our formulation deals with the issue of disparate labels over a large-scale multi-domain system.

In this paper, we present a neural network architecture that addresses the challenges above. Main contributions of the paper.

1. We present the task of tracking slots in a conversation as a carryover decision. This allows us to scale to a potentially unbounded set of slot values, and allows us to generalize anaphora resolution to both explicit and implicit references.

2. We address diverse schema challenge by leveraging label embeddings (see sec 2.2.1) to generate potential candidates to be carried.

3. We show our proposed model outperforms a strong rule-based baseline. We also demonstrate via experiments...
why our task is more complex than dialog state tracking by benchmarking our approach on DSTC-2 as well as on a dataset collected from a real virtual assistant device.

2. Approach

2.1. Task Definition

We define a dialog turn at time $t$ as the tuple $\{a_t, S_t, w_t\}$, where $w_t \in W$ is a sequence of words $\{w_{t-1}\}_{t-1}$; $a_t \in A$ is the dialog act; and $S_t$ is a set of slots, where each slot $s$ is a key value pair $s = (k, v)$, with $k \in K$ being the slot name (or slot key), and $v \in V$ being the slot value. $u_t = \{a_t^v, S_t^v, w_t^v\}$ represents a user-initiated turn and $v_t = \{a_t^v, S_t^v, w_t^v\}$ represents a system initiated turn.

Given a sequence of $D$ user turns $\{u_{t-D+1}, \ldots, u_{t-1}, u_t\}$; and their associated system turns $\{v_{t-D+1}, \ldots, v_{t-1}, v_t\}$; and the current user turn $u_t$, the task is to predict a carryover decision over each of the candidate slots in $C(S) = \cup_{i \in t, j \in t-D+1} S_i$ i.e we carryover slot $s \in C(S)$ to turn $u_t$ if $P(+1|s, u_t, u_{t-D+1}^{-1}, v_{t-D+1}^{-1}) > \tau$, where $\tau$ is a decision threshold to be optimized. This formulation allows us to scale to potentially unbounded slots and also handle diverse schemas as we will discuss later.

2.2. Model Definition

We use an encoder-decoder approach as shown in Figure 1 to classify each candidate slot as being relevant or not for the current turn.

2.2.1. Candidate Slot Generation

As shown in Table 1, the schemas associated with each turn can be in completely different label spaces. So, we use slot key embeddings to map the keys of the candidate slots in $C(S)$ into the schema associated with the current domain. We use pretrained word embeddings as the source for computing the slot key embeddings.

For each slot name $k$ we compute its label embedding by averaging over the $M_k$ associated slot value embeddings. For multi-word slot values, the embedding is constructed by averaging the associated word embeddings.

$$\Phi_W(v) = \frac{1}{|w \in v|} \sum_{w \in v} \Phi_W(w)$$ (1)

$$\Phi_K(k) = \frac{1}{M_k} \sum_{i=1}^{M_k} \Phi_W(v_i)$$ (2)

We now construct the transformed candidate set as $C'(S) = \{(k', v) | (k, v) \in C(S) \& \, \Phi_K(k) \oplus \Phi_K(k') > \beta\}$, where, $\oplus$ is the dot product and $\beta$ is a tunable threshold over the development set.

2.2.2. Dialog Encoding

We first embed the words in the utterance sequence $w_t$ using word embeddings $\Phi_W$ [21], to get the sequence $u_t$, which are then fed into an LSTM to recursively encode the current turn, the turns in context associated with the user and the system respectively:

$$h_t^0 = LSTM(a_t^v)$$ (3)

$$h_{t-1}^v = LSTM(a_{t-1}^v), \, t-D+1 \leq j < t$$ (4)

$$h_{t-1}^v = LSTM(a_{t-1}^v), \, t-D+1 \leq j < t$$ (5)

The LSTM is stateful i.e the LSTM output of the last token in the utterance is fed as an initial state to the LSTM for the next utterance.

2.2.3. Encoding Dialog Act

The dialog act for the current turn $a_t$ is encoded into a fixed length vector of dimension $D_A$, using the intent embedding dictionary $\phi_A$ as:

For each dialog act $k$ we compute its embedding by averaging over the $M_k$ associated utterance embeddings. For each utterance we calculate its embedding is constructed by averaging the associated word embeddings.

$$\Phi_U(u) = \frac{1}{|w \in u|} \sum_{w \in u} \Phi_W(w)$$ (6)

$$\Phi_A(k) = \frac{1}{M_k} \sum_{i=1}^{M_k} \Phi_U(u_i)$$ (7)

2.2.4. Encoding Candidate Slot

The candidate slot $s = (k, v)$ is encoded into a fixed length vector $D_s$, as a concatenation of the slot key embedding and the slot value embedding.

$$h_s = \phi_K(k) \oplus \phi_W(v)$$ (8)

Here, $\oplus$ implies concatenation; $\phi_W(v)$ and $\phi_K(k)$ is as defined in Equation 1 and Equation 2 respectively.

2.2.5. Recency Encoding

The slot distance $d_s$, defined as the integer offset of the candidate slot from the current turn, is encoded as one-hot $\{0, 1\}^{D_s}$.
The final distance encoding vector can then be constructed using an affine transform:

$$h_d = W_d \ast \text{OneHot}(d_i) + b_d \quad (9)$$

2.2.6. Attention Mechanism

We consider two levels of attention - the word level attention allows the model to focus on individual mentions in the utterance that influence the slot carryover decision, and the stream level attention which allows the model to focus on specific streams (user and system) in the dialog.

**Word Attention:** For each stream defined in Section 2.2.2, we attend over the words in that stream and compute a per-stream context vector. For stream vector sequence $h^n_s$, and slot embedding $h_s$, we compute the word level attentional context vector as:

$$e_{js} = g(h^n_{t_j}, h_s) \quad (10)$$

$$\alpha_{js} = \text{softmax}(e_{js}) \quad (11)$$

$$c^n_j = \sum_{j=1}^{Nt} \alpha_{js} h^n_{t_j} \quad (12)$$

Here $j$ an index into $h^n_s$, represents the hidden encoding of the associated input word at that position. We then compute the importance of the word to the slot as the similarity defined in $e_{js}$, obtain the normalized weights $\alpha_{js}$ that is then used to compute the weighted context vector $c^n_j$. Similarly, we can compute $c^n_{t-1}$ and $c^n_{t-1}$.

**Stream Attention:** As before, we attend over each individual stream $c_k \in \{c^n_{t-1}, c^n_{t-1}, c^n_t\}$ to obtain the final context vector $h_c$:

$$e_{ks} = g(c_k, h_s) \quad (13)$$

$$\alpha_{ks} = \text{softmax}(e_{ks}) \quad (14)$$

$$h_c = \sum_{k=1}^{D} \alpha_{ks} c_k \quad (15)$$

2.2.7. Decoder

The vectors from the encoders are concatenated and sent to the final softmax layer to get the class probabilities as follows:

$$z = h_c \odot h_a \odot h_s \odot h_d \quad (16)$$

$$\hat{o} = \text{softmax}(W_{\text{decoder}} \ast z + b_{\text{decoder}}) \quad (17)$$

3. Experiments

3.1. Data Setup

For the experiments, we use subset of data collected on a commercial voice assistant. Table 2 summarizes the statistics in the training, development and test sets across different domains. We first list the number of dialogs in each domain, and then give some overall statistics. Around 20% of the dialogs have utterances from more than one schema. For dialogs belonging to multiple schema we classify them to schema for the current turn. Also, as expected for a voice assistant, we have a significant imbalance where the number of positive candidate slots are much smaller than number of possible candidates for each turn. This is due to cross domain interactions which follow each other but are not part of the same goal, which is common in a digital assistant. Furthermore, some domains chose not to associate any label with an entity mention which we represent as **Entity** slot this results in very large number of potential candidates as we consider all possible target slots in the current domain for such entities.

To demonstrate complexity of our data we also report results on dataset released in Dialog State Tracking Challenge [22]. We modified the dataset to fit our carryover task accordingly. We consider only the 1-best ASR and 1-best SLU hypothesis. Unlike our commercial dataset, in DSTC the slots tracked as part of the goal only occur from the user turn. So, we remove candidates from system turns as a pre-processing step. Also, in DSTC task dialogs can be system initiated but for our task is always user initiated, hence we remove the first system turn.
3.2. Training Setup and Evaluation Metrics

We introduce two baselines. The 'Naive Baseline' system carries over all the slots from the most recent turn in the dialogue session. This is because, most recent entities are more likely to be referred to by the users in a spoken dialogue system. We also use a stronger elaborate ‘Rule Baseline’, where we detect the referring expression, and for each referring expression, linguistic and semantic features are used to retain only those antecedent candidate slots that agree in gender, number and type. Algorithm 1 shows an example rule that executes for the use case U2 in Table 1.

Table 2: Contextual Carryover Data Setup

<table>
<thead>
<tr>
<th>Domain</th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Music</td>
<td>4587</td>
<td>558</td>
<td>580</td>
</tr>
<tr>
<td>Weather</td>
<td>6067</td>
<td>738</td>
<td>729</td>
</tr>
<tr>
<td>Local Businesses</td>
<td>1439</td>
<td>162</td>
<td>158</td>
</tr>
<tr>
<td>Video</td>
<td>1000</td>
<td>99</td>
<td>141</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>386</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>Home Automation</td>
<td>1945</td>
<td>230</td>
<td>291</td>
</tr>
<tr>
<td>Others</td>
<td>1481</td>
<td>163</td>
<td>178</td>
</tr>
<tr>
<td>Total</td>
<td>16905</td>
<td>1999</td>
<td>2152</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>avg. turns per session</th>
<th>2.2</th>
<th>2.14</th>
<th>2.18</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>% of discrete schema sessions</th>
<th>19.86</th>
<th>18.75</th>
<th>20.53</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Avg. +ve carryover candidates per turn</th>
<th>0.37</th>
<th>0.39</th>
<th>0.35</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Avg. -ve carryover candidates per turn</th>
<th>4.01</th>
<th>4.00</th>
<th>4.00</th>
</tr>
</thead>
</table>

Table 2: Contextual Carryover Data Setup

Algorithm 1 Example rule carrying over City slot as it is compatible in type to anaphor "there"

```plaintext
if ReferringPhrase.Type == CandidateSlot.Type
    if CandidateSlot.Type == City
        if ReferringPhrase.Value == "there"
            CarriedSlots += CandidateSlot
    end if
end if
```

For the model, we initialize the word embeddings using 300 dimensional pre-trained GloVe [23] vectors. The model is trained using mini-batch SGD with Adam optimizer [24] with standard parameters to minimize the class weighted cross-entropy loss. In our experiments, we use 128 dimensions for the LSTM hidden states and 256 dimensions for the hidden state in the decoder. Similar to [25], we pre-train a LSTM for the named entity recognition task and use this model to initialize the parameters of the LSTM based encoders. All model setups are trained for 20 epochs with early stopping criterion optimised on a dev set. We only select those hypothesis, whose \( \tau \) > 0.5, which was optimized over the dev set. For each utterance, independent carryover decisions are made for each candidate slot. In case of conflict, we select the candidate with highest score. We evaluate the models by comparing the hypothesis and reference slots to measure precision, recall and F1 scores.

3.3. Results and Discussion

Table 3 shows the cumulative impact of various training strategies of our proposed model. We see that compared to a strong rule-based system, our proposed approach gives significant gains in accuracy. Our proposed encoder-decoder improves upon the strong rule based baseline. Adding word attention helps improve the precision and F1 but at the cost of recall; the results are significant compared to the system without attention. Intuitively, the slot value matching referring tokens in the dialog turn indicates that it is relevant to the conversation. The word attention model captures this intuition as part of the model learning process. This alleviates the need to explicitly define semantic type similarity features, and detecting anaphoric mentions like we do in the rule based system. Adding stream attention improves recall, but the overall F1 degrades. Stream attention, which helps isolate user and system turns did not help; we speculate that the distance feature already captures this, and there is insufficient data to train this appropriately.

For completeness we also include performance on the public DSTC2 dataset in Table 4. We do not claim to be solving DSTC2 but only use this dataset as a comparison of task complexity - the DSTC2 task is relatively simple as evidenced by the naive baseline having a high F1 score on this task, but very low on our commercial assistant task.

Table 3: Modeling Architecture Impact on Accuracy for Multi-Domain Dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Baseline</td>
<td>17.01</td>
<td>92.50</td>
<td>28.74</td>
</tr>
<tr>
<td>Rule Baseline</td>
<td>91.79</td>
<td>67.11</td>
<td>77.53</td>
</tr>
<tr>
<td>Encoder-Decoder</td>
<td>73.31</td>
<td>96.17</td>
<td>83.20</td>
</tr>
<tr>
<td>+ word attention</td>
<td>75.76</td>
<td>94.65</td>
<td>84.16</td>
</tr>
<tr>
<td>+ stream attention</td>
<td>73.48</td>
<td>96.18</td>
<td>83.31</td>
</tr>
</tbody>
</table>

Table 4: Results on DSTC2 dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Baseline</td>
<td>80.58</td>
<td>75.44</td>
<td>77.93</td>
</tr>
<tr>
<td>Encoder-Decoder</td>
<td>97.22</td>
<td>95.30</td>
<td>96.24</td>
</tr>
<tr>
<td>+ word attention</td>
<td>97.20</td>
<td>97.65</td>
<td>97.42</td>
</tr>
<tr>
<td>+ stream attention</td>
<td>97.23</td>
<td>95.61</td>
<td>96.42</td>
</tr>
</tbody>
</table>

Table 5: Cross-domain accuracy of encoder-decode w/ word attention for the model in Table 3

<table>
<thead>
<tr>
<th>Category</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Within-Domain</td>
<td>78.19</td>
<td>95.7</td>
<td>86.09</td>
</tr>
<tr>
<td>Cross-Domain</td>
<td>61.66</td>
<td>87.05</td>
<td>72.19</td>
</tr>
</tbody>
</table>

4. Conclusion

In this work, we presented the task of contextual carryover of slots in a multi-domain large-scale dialog system. To address the scalability of the solution over a large set of slot values we re-formulated this as a slot carryover decision to identify the most relevant set of slots at the current turn. Furthermore, we proposed an efficient way to leverage label embeddings to deal with heterogeneous schemas. We presented empirical results demonstrating the efficacy of our neural network formulation over a strong rule-based baseline. We also quantified the gains from various components of the proposed approach.

5. References

spoken language understanding,” in INTERSPEECH, 2013.


