HIERARCHICAL FEATURE-BASED TRANSLATION FOR SCALABLE NATURAL LANGUAGE UNDERSTANDING

Ganesh N. Ramaswamy *Jan Kleindienst*

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

For complex natural language understanding systems with a large number of statistically confusable but semantically different formal commands, there are many difficulties in performing an accurate translation of a user input into a formal command in a single step. This paper addresses scalability issues in natural language understanding, and describes a method for performing the translation in a hierarchical manner. The hierarchical method improves the system accuracy, reduces the computational complexity of the translation, provides additional numerical robustness during training and decoding, and permits a more efficient packaging of the components of the natural language understanding system.

1. INTRODUCTION

State-of-the-art natural language understanding (NLU) systems typically perform a direct translation of the user input to a formal command in a single step. For single domain systems, such as those described in [4], [8], [9], single step natural language understanding may be sufficient. But with the growing interest in building voice portals that provide one interface to connect to a wide variety of services, challenges in building multi-domain systems have to be addressed. Some of the challenges, such as permitting the user to have several open transactions, permitting the user to switch between domains seamlessly, allowing the conversational context and dialog system services to be shared across multiple domains, and supporting dynamic vocabulary across multiple domains, have been addressed in [6]. However, the natural language understanding in [6] was still performed in a single step, with the formal commands from all of the domains treated as candidates for any given input. Other work in multi-domain systems include [2], where the focus was on designing a two-stage speech recognition system, and [7], which addresses discourse modeling for multi-domain systems.

One of the major challenges in building a multi-domain NLU system is that of scalability of the system, since the number of formal commands to be supported may be very large. Even with a single-domain system, if the number of formal commands is large, scalability issues have to be addressed. For example, if we were to use a feature-based translation approach for natural language understanding [1], [3], [5], [6], and attempt to support a large number of formal commands, a large number of features may be required to perform a direct one-step translation. But increasing the number of features introduces additional statistical noise that may result in degradation of accuracy of the system, in addition to other computational difficulties such as added computation time and memory, and possible numerical instability during training and decoding (due to the large number of features). Consider for example translating commands such as "forward this message" (action), "how do I forward a message" (help), or "did I forward a message" (query). All three of these sentences are related to forwarding a message, and as such the key word "forward" may appear to be important, but a feature set consisting of the word "forward" would only introduce noise and may lead to inaccuracies in the translation, for this particular example.

In order to handle the scalability problems brought forth by a complex natural language understanding task, this paper describes a hierarchical approach for natural language understanding, consisting of two or more levels of translation. For instance, in the above example, we could break the translation process into two levels. The first level may be responsible for deciding which category (action, help or query) the sentence belongs to, and we may use the presence or absence of features such as "how do I" and "did I" to make the decision. Once the category is determined, we can then proceed to level 2 of the translation which would determine the command itself, for which we may use features such as "forward", etc. When appropriate, we can use more than two levels in the hierarchical translation. The hierarchical approach would allow us to keep the total number of features at each level to be smaller than the number of features required in a direct one-step translation, and the features can be selected very efficiently to perform the limited task at each level. The small number of features at each level reduces the noise problem, reduces computational complexity, and also reduces numerical instability problems while training and decoding. In addition, the hierarchical models also allow for efficient packaging of the models. For example, suppose we built a system for handling just the action and query commands, but later we decide to add the help commands as well. In this case, we can keep the Level 2 models for action and query unchanged, incorporate a new Level 2 model for the help category and make minimal changes to the Level 1 model. If we did not use the hierarchical approach, it may be necessary to rebuild the entire system to incorporate any new additions.

The hierarchical NLU described here does not assume a specific input modality. The input to the NLU system is assumed to be a natural language text string, which may have originated from any input modality such as speech, handwriting or typed text, and passed through an appropriate recognition engine to create the text string, if necessary. However, in order to illustrate the performance of the proposed method, we describe how the hierarchical NLU may be integrated into a practical conversational systems, where the primary input modality is speech. In particular, we report results obtained by integrating the hierarchical NLU into the multi-domain conversational system consisting of email, calendar and address book applications previously described in [6]. As expected, the hierarchical NLU resulted in improvements in accuracy and speed of the natural language understanding, while also improving the numerical robustness of the system.

The remainder of this paper is divided into 4 sections. In Section 2, we review the feature-based approach for natural language

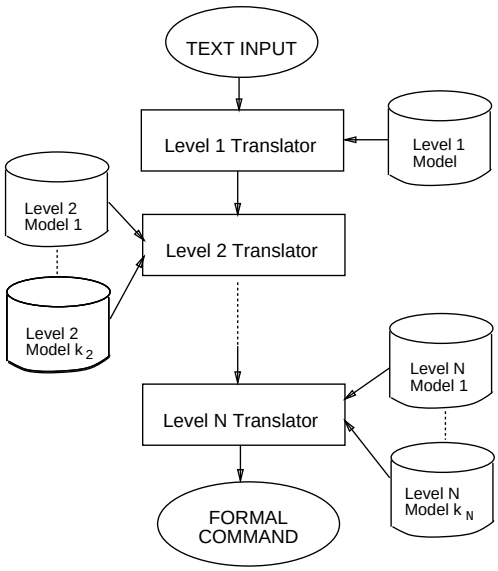

Figure 1: **Hierarchical Translation**

understanding and introduce the basic steps in designing a hierarchical NLU. Additional enhancements and variations are described in Section 3. Section 4 presents some experimental results, and Section 5 concludes the paper.

2. HIERARCHICAL TRANSLATION

In this section, we first review the single-step feature-based translation used in the NLU engine of the multi-domain conversational system described in [6]. We then motivate the need for a hierarchical approach for natural language understanding by examining the problems that arise when the number of formal commands and the number of features grow to be very large. The detailed description of a hierarchical NLU engine is presented through examples, and mathematical details are omitted for clarity and brevity.

The NLU engine of [6] consists of two sub-components: a tagger and a feature-based translator. The tagger assigns words and phrases in the recognized text to appropriate classes. The translator takes the tagged sentence and translates it to a formal language statement. For example, a sentence of the form "show me the first email from John Smith" may be tagged as

```
OPEN the COUNT MESSAGE from NAME
```

and translated into a formal language statement of the form

```
open_message(message=COUNT, sender=NAME)
```

where `COUNT` maps to 1, and `NAME` maps to John Smith. The tagger is based on parsing techniques similar to those described in [8], and the feature-based translator is based on the *maximum entropy* approach, [1], [3], [5]. The maximum entropy models are built from tagged versions of training sentences from the relevant domains. The first step is the selection of features, followed by the computation of weights for the features, done using the *improved iterative scaling* algorithm [3], [5]. For instance, in the example above, the features "OPEN" and "MESSAGE" may contribute towards the presence of "open_message" in the formal command produced by the translator. The weights for the features are chosen to maximize the likelihood of the training data. During translation, a formal command is chosen by examining the product of the weights for the features that are present in the input.

As we added more data to the training set, additional formal commands had to be defined to support the new concepts. As a result, the number of features in the translator's feature set had to be increased, and we discovered a critical scalability problem. There were several instances during training where the iterative scaling algorithm did not converge and produced infinite weights. While it was possible to make the training successful through manual intervention, there were other scalability problems during decoding. For example, the presence of large number of features resulted in the product of the weights exceeding the floating point limit of the hardware. Hence, it was necessary to consider a hierarchical approach to handle the complexity. In addition to resolving the computational difficulties, the hierarchical NLU also resulted in accuracy improvements and reductions in computation time and memory, as we shall see in Section 4.

Figure 1 shows an example of how a hierarchical NLU may be built. There are N translators, ranging from Level 1 Translator to Level N Translator. Each translator performs one portion of the translation process, and narrows down the search space of the formal commands for the subsequent level. Level 1 takes in the (tagged) text input and initiates the translation process, and Level N concludes the translation process by producing the formal command associated with the text input. The value of N may be chosen on the basis of the overall complexity of the translation task.

The first step in designing a hierarchical NLU is the categorization of the formal commands. Categories and nested subcategories of the formal commands should be defined and each formal command should be assigned to one (or more) of these categories. In general for Level i, there will be k_i categories. For Level 1, the obvious choice is $k_1 = 1$. For subsequent levels, the value of k_i for each level i may be chosen on the basis of the relative complexity of the translation task for the corresponding level.

For example, lets consider the Personal Information Management (PIM) applications involving email, calendar and address book, as in [6]. For this example, we may define the Level 2 categories to be:

- Email
- Calendar
- Address Book
- General
- DoNothing

In this case, we have chosen to use the individual applications (email, calendar, address book) as the basis for defining the categories. This is one of many possible ways to define categories. The Email category may contain sentences such as

```
do have any new mail // check_new_mail()
open the message from John
  //open_mail_message(sender=John)
```

Here we have shown the actual sentences on the left, and the corresponding formal command on the right (details pertaining to tagging the sentences are omitted for clarity). Similarly, the Calendar and Address Book categories will contain sentences relevant to the individual domains.

The General category is necessary because some of the sentences may be ambiguous, such as

```
open this // open_object (object=current)
okay // affirmation()
cancel that // undo()
```

and in these cases, the context history may be used to resolve the full meaning of the command (i.e. to figure out what are the objects or actions that are referenced).

The DoNothing category consists of sentences that cannot be reliably translated to a formal command within the domain. This category may contain out of domain sentences, incomplete sentences, or poorly phrased sentences such as

```
it is a nice day // do_nothing()
I think may be we // do_nothing()
```

The system will in general not do anything in response to these sentences, and may ask the user to repeat the command. The DoNothing category plays an important role in assuring the robustness of the system.

We can further divide each of the Level 2 categories into two or more Level 3 categories. For example, the Level 2 Email category may be divided into Level 3 categories such as "folder manipulation" and "message manipulation". As noted earlier, the number of levels and the number of categories in each level may be chosen on the basis of the overall complexity and the relative complexity at each level, respectively, of the translation tasks involved. For the purpose of illustration, we will restrict ourselves to using 2 levels ($N = 2$).

The first step in building the models for each of the levels is to collect sentences from all of the relevant domains and tag them appropriately, as described earlier. Then a training data file is created for each level in the translation process, to indicate the desired output at that level. For example, the training data for Level 1 will contain entries such as:

```
open the message from John  // Email
what is the date tomorrow // Calendar
what is Peter's phone number // Address Book
go to the next one // General
send this to ah I mean uh // DoNothing
```

On the left hand side, the entries are the actual sentences (again we have omitted the tagging details for clarity), and on the right hand side are the desired Level 1 outputs, as discussed earlier. Since we have five categories for Level 2, there are five possible outputs for Level 1. Building the Level 1 model may be done using the feature selection and iterative scaling algorithm referenced earlier, or using other similar techniques in statistical natural language processing.

The Level 2 Translator would take the (tagged) text input along with the output of the Level 1 translator (which would be one of the five categories described above) and produce the final formal command (in the most general case, where there may be more than two levels, Level 2 output would be one of the categories for Level 3, and so on; but since in our example there are only 2 levels, Level 2 would produce the final formal command). As noted earlier, there will be k_2 categories for Level 2. In our example $k_2 = 5$, and so there are five categories and hence five corresponding Level 2 models (i.e. Level 2 Model 1, ... , Level 2 Model 5). The training data will be partitioned into five different sets, corresponding to each category. Although overlapping (soft) partitioning may also be done, where certain portions of the training data appear in more than one category, for the purpose of illustration, we will restrict ourselves to hard partitioning of the training data. After partitioning, the training data for the Calendar category, for example, may contain entries such as:

```
what is the date tomorrow
  //query_date(day=tomorrow)
create a meeting with David at ten o'clock
  //create_calendar_entry(name=David,
    start_time=10)
```

Similarly, the other four training data sets will contain entries relevant to the respective categories. Level 2 models may also be built with the same feature-based translation approach described thus far, or another similar method from statistical natural language understanding. With the Level 2 models built as such, the Level 2 Translator will examine the Level 1 output and load the appropriate Level 2 model corresponding to the category chosen by Level 1. Then, the Level 2 Translator will translate the text input into one of the formal commands for that category and this formal command will be the output of the NLU engine. The formal command may then be submitted to the appropriate application, or another component in the conversational system (such as a dialog engine) for further processing.

Also, as noted in the introduction, the hierarchical method lends itself to a more efficient packaging of the models. Categories may be added or removed from the system with changes made only to small subset of the models.

3. PRACTICAL CONSIDERATIONS

We have so far described how to build a basic hierarchical translator that uses two or more levels of decision making to arrive at a formal command. Now we describe some variations to the scheme than can further improve the performance of the hierarchical translator.

The first variation improves the accuracy of the system by considering more than the top ranking output from any intermediate level for subsequent levels. For example, instead of just considering the first choice of Level 1 output as the category selected for Level 2 translation, we can consider the top two (or more) of the choices of the Level 1 translation. Suppose that for a given input sentence, the top two choices of Level 1 translation are Email and Calendar, then the Level 2 translation made be done for both of these categories, and the formal command with the highest combined score of Level 1 and Level 2 translation (which may be computed as a product of the individual Level 1 and Level 2 scores) could be selected as the final formal command. We may select more than two categories, but as the number of categories increases, so does the computational complexity. The number of categories to be selected may also be dynamically determined by looking the distribution of the scores. For instance, if the top ranking category has a high enough score (above a predetermined threshold), then it may be sufficient to output just the top ranking category.

Another variation is to modify the training data sets for Level 2 (and all subsequent levels, if any) to include not just the entries relevant to the category but all the entries. For example, the training data for Level 2 Model 1 (for the Email category) may contain entries such as

```
do have any new mail // check_new_mail()
what is the date tomorrow // Error
it is a nice day // Error
```

The entries that are relevant to the given category, such as the first entry above, are mapped to the correct formal command. Other entries are included, and are marked as an "Error" to indicate that these sentences do not belong to the current category. The purpose

Table 1: **Experiments with Hierarchical Translation.** The table below shows the error rates for the single-step translation and the hierarchical translation. For the experiments involving just the translation, CPU times are shown in parenthesis.

Data	Single-Step Translation	Hierarchical Translation
Training Set		
Translation only	5.9 % (235 s)	2.2% (80 s)
Tagging & Translation	9.2 %	5.5%
Test Set		
Translation only	7.0% (16 s)	6.3 % (5 s)
Tagging & Translation	10.8%	10.4%
Speech, Tagging & Translation	22.5%	21.8%

of doing this is to correct errors that may have been made at an earlier level. For example, if the sentence "what is the date tomorrow" is incorrectly translated at Level 1 as belonging to the Email category (it should really belong to the Calendar category), then while making the Level 2 translation, now it is possible to recognize that the Level 1 Translator may have made an error. Whenever Level 2 translation results in an error, we then attempt to consider the second choice of the Level 1 output. If the second choice of the Level 1 translator happens to be the correct category, then it is likely that the subsequent attempt at Level 2 translation for the same command may result in the correct formal command. If even the second choice of the Level 1 output happens to lead to a Level 2 error, then we try the third choice and so on until all choice are exhausted. If all choices of the Level 1 output result in a Level 2 error, then we automatically map the sentence to the DoNothing category, and this case the user will be asked to rephrase the input. This variation ensures additional robustness of the NLU system by reducing the consequences of intermediate errors in the hierarchical translation.

In all of the examples considered so far, we have assigned each formal command to only one category, but in general, some formal commands may have to be mapped to two or more categories. In particular, if we do not use the General and DoNothing categories at Level 1, then it is desirable to include the entries that would otherwise belong to these categories to all other relevant categories.

4. PERFORMANCE EVALUATION

In this section, we describe some of the experiments that were done to compare the performance of the single-step translation and the hierarchical translation. We used the multi-domain conversational system described in [6] as the test-bed for the comparison. The conversational system provides a single interface to email, calendar and address book applications and supports seamless switching between domains and sharing of contexts across domains.

We first ran a number of different experiments on the system using a single-step translator. We then repeated the same experiments on the same data, by replacing the single-step translator with a hierarchical translator, while retaining the same speech recognition engine and the same tagger. The error rates obtained on the training data and an independent test set are shown in Table 1. For the experiments involving only translation, CPU times used for the translation process are also shown (CPU times for speech recognition and tagging are omitted since they are not relevant here). The "translation only" error rates assume correct speech recognition and correct tagging. The "translation and tagging" error rates, which are the total NLU error rates, assume correct speech recognition. The "speech, tagging and translation" error rates are the end-to-end cumulative error rates that the user would perceive while using the system.

The hierarchical translator used for these experiments was built as described in Section 2, without any of the enhancements presented in Section 3. Even with this basic version, the hierarchical translator consistently outperformed the single-step translator on both error rates and CPU times. As noted earlier, the hierarchical approach also improves numerical robustness and reduces manual intervention during training. Additional error rate reductions may be obtained with the variations and enhancements described in Section 3.

5. CONCLUSIONS

We have described a hierarchical approach for natural language understanding that addresses scalability issues and provides improvements in accuracy, computational resources and numerical stability. Although we focused on feature-based translation, the hierarchical approach can easily be adapted to other similar statistical natural language understanding techniques. We believe that the hierarchical approach will permit us to build very large natural language understandings systems in the future, including voice portals and other interfaces connecting to a large number of services and domains.

REFERENCES

[1] Berger, A., Della Pietra, S., and Della Pietra, V., " A Maximum Entropy Approach to Natural Language Processing," *Computational Linguistics*, Vol. 22, No. 1, March 1996.

[2] Chung, G., Seneff, S., and Hetherington, L., "Towards Multi-Domain Speech Understanding Using a Two-Stage Recognizer," *Eurospeech*, Budapest, Hungary, 1999.

[3] Della Pietra, S., Della Pietra, V., and Lafferty, J., "Inducing Features of Random Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 4, April 1997.

[4] Lamel, L., Rosset, S., Gauvain, J. L., and Bennacef, S., "The LIMSI ARISE System for Train Travel Information," *International Conference on Acoustics, Speech and Signal Processing*, Phoenix, Arizona, March 1999.

[5] Papineni, K., Roukos, S., and Ward, T., "Feature-Based Language Understanding," *EUROSPEECH*, Rhodes, Greece, 1997.

[6] Ramaswamy, G., Kleindienst, J., Coffman, D., Gopalakrishnan, P., and Neti, C., "A Pervasive Conversational Interface for Information Interaction," *Eurospeech*, Budapest, Hungary, September 1999.

[7] Seneff, S., Goddeau, D., Pao, C., and Polifroni, J., "Multi-Modal Discourse Modeling in a Multi-User Multi-Domain Environment," *ICSLP*, 1996.

[8] Ward, T., Roukos, S., Neti, C., Gros, J., Epstein, M., and Dharanipragada, S., "Towards Speech Understanding Across Multiple Languages," *International Conference on Spoken Language Processing*, Sydney, Australia, December 1998.

[9] Zue, V., et. al., "JUPITER: A Telephone-Based Conversational Interface for Weather Information," *IEEE Transaction on Speech and Audio Processing*, Vol. 8, no. 1, January 2000.