Neural Lexicons for Slot Tagging in Spoken Language Understanding

Kyle Williams
Microsoft
One Microsoft Way
Redmond, Washington, 98052
kyle.williams@microsoft.com

Abstract

We explore the use of lexicons in neural models for slot tagging in spoken language understanding. We develop models that encode lexicon information as features for use in a Long-short term memory neural network. Experiments are performed on data from 4 domains from an intelligent assistant under conditions that often occur in an industry setting, where there may be: 1) large amounts of training data, 2) limited amounts of training data for new domains, and 3) cross domain training. Results show that the use of neural lexicon information leads to a significant improvement in slot tagging, with improvements in the F-score of up to 12%.

1 Introduction

Spoken language understanding (SLU) is an important component of systems that interface with users, such as intelligent assistants. These systems are becoming increasingly popular as a means for people to accomplish tasks in their homes and on mobile devices. These tasks might include switching on the lights or booking a taxi. Typically, an SLU system detects the domain, intent, and semantic slots of an utterance (Li et al., 2017) and uses the information to perform actions.

It is common to use lexicons (also known as gazettes or dictionaries) to improve the performance of SLU systems (Ratinov and Roth, 2009). Lexicons are typically collections of phrases that are semantically related and thus provide knowledge that can aid the SLU system. For instance, a lexicon called *holidays* might contain the phrases *Thanksgiving, Christmas Eve, Labor Day.* Similarly, a lexicon called *days of the week* would contain *Monday, Tuesday, Wednesday*, etc. There are many ways lexicons can be built, such as by using domain experts or by harvesting information from knowledge graphs, such as DBPedia[1]. In an industry setting, it is also possible that lexicons already exist for other natural language applications.

Previous work has shown how lexicons can be used to improve slot tagging with Conditional Random Fields (Ratinov and Roth, 2009), where slot tagging refers to the process of identifying semantic entities of interest in an utterance. For instance, given the utterance *"book a taxi to the airport"*, a slot tagging model might identify *taxi* as a *transport_type* and *airport* as a *destination*. In this paper, we investigate the effect of integrating these types of lexicons into Long-Short Term Memory (LSTM) neural models in an industry setting. We focus on LSTM models since they have been shown to produce state-of-the-art results in many natural language tasks. We consider integrating lexicon features into a Long-short term memory neural network in two ways: 1) by considering lexicon membership as binary features and 2) by embedding the lexicons and allowing the model to learn the representation as part of the end-to-end training of the neural network.

To evaluate these approaches, we measure the performance of models on data from four domains belonging to an intelligent assistant under three data scenarios that commonly occur in production SLU systems. The first scenario is when there is a considerable amount of training data available to train a SLU system, as may occur if a sizeable investment has been made to collect data. The second scenario is when there is only a small amount of training data available, as may be the case when the SLU is expanded to cover new domains for which very little training data exists. The third scenario is cross domain slot prediction, where we use a model trained on utterances from one domain to identify entities in utterances belonging to another

[1] https://wiki.dbpedia.org/

domain. This setting commonly occurs when one attempts to leverage existing SLU models for use in a new domain.

2 Related Work

There have been many previous studies involving spoken language understanding for the slot tagging problem. Yao et al. (2014) investigate the use of LSTMs for slot tagging and compare the performance of the LSTM-based model to a standard RNN and a Conditional Random Field. Their results show the LSTM to outperform the two other models. Mesnil et al. (2015) evaluate several RNN-based models for slot tagging and show the RNN-based models to outperform the CRF-based model. Ma and Hovy (2016) propose a LSTM-CNN-CRF model, which induces character representations using a convolutional neural network. The character representations are then combined with word embeddings and fed into an LSTM, and lastly the output of the LSTM is fed into a CRF decoder. In Kurata et al. (2016), the authors use an encoder-labeler approach to first encode sentences into fixed size vectors and then use the encoded state as the initial state for a labeling LSTM.

The reason that many researchers have been using LSTMs for natural language understanding is due to their ability to model long-term dependencies. However, some researchers have proposed other architectures. For instance, Shi et al. (2016) propose the Recurrent Support Vector Machine (RSVM), which uses a recurrent neural network to induce a feature representation and a structured support vector machine to perform structured prediction on the output of the RNN.

Dugas and Nichols (2016) use lexicon embedding features for named entity recognition in tweets. They produce lexicon embeddings that are concatenated with word embeddings; however, our work differs in the inclusion of the additional neural lexicon models, experiments comparing them under varying data conditions that commonly occur in an industry setting, and our evaluation is based on spoken utterances from an intelligent assistant rather than tweets. Furthermore, we analyze cases where lexicons are useful and cases where they are not.

3 Neural Lexicon Models

Our proposed neural lexicon models are based on an Long Short-Term Memory (LSTM) architecture. The architecture is shown visually in Figure 1 and described in detail below.

In each of our models we induce a feature representation based on the characters and words that appear in an input sequence of words. We closely follow the approach of previous studies (Kim et al., 2017; Lample et al., 2016) and induce both character and word embeddings using bidirectional LSTMs. As in Kim et al. (2017), for a given sequence of words $W = w_1, w_2, ..., w_n$ where word w_i has character $w_i(j)$ at position j. We define the following:

- Character embedding: e_c for each $c \in C$
- Character LSTM: ϕ_f^C, ϕ_b^C
- Word embedding: e_w for each $w \in W$
- Word LSTM: ϕ_f^W, ϕ_b^W,

where $\phi_f^C, \phi_b^C, \phi_f^W, \phi_b^W$ refer to the forward and backward character and word LSTMs. A character sensitive word representation v_i is computed as:

$$f_j^C = \phi_f^C(e_{w_i(j)}, f_{j-1}^C), \forall j = 1...|w_i| \quad (1)$$
$$b_j^C = \phi_b^C(e_{w_i(j)}, b_{j+1}^C), \forall j = |w_i|...1 \quad (2)$$
$$v_i = f_{|w_i|}^C \oplus b_1^C \oplus e_{w_i}, \quad (3)$$

where $\oplus$ represents the vector concatenation operation whereby the final states of the forward and backward LSTMs are concatenated with the word embedding. Next the model computes:

$$f_i^W = \phi_f^W(v_i, f_{i-1}^W), \forall i = 1...n \quad (4)$$
$$b_i^W = \phi_b^W(v_i, b_{i+1}^W), \forall i = n...1 \quad (5)$$

In other words, the forward and backward word LSTMs are used to induce character and context sensitive word representations. Finally, the states of the forward and backward LSTMs are concatenated to induce the final word representation r_i:

$$r_i = f_i^W \oplus b_i^W, \quad (6)$$

for each word $w_i, i = 1...n$. These r_i are the word representations that we use for slot prediction.

In this study we focus on slot tagging and predict the tag of each word $w_i, i = 1...n$ using $r_i, i = 1...n$. To do this we add a feed forward layer g, which takes as input the r_i at each timestep. We take the softmax of the the output of g to produce probabilities of semantic tags for each word w_i. We then minimize the cross entropy loss:

$$Loss^{\text{tag}} = -\sum_i p_i \log q_i, \quad (7)$$

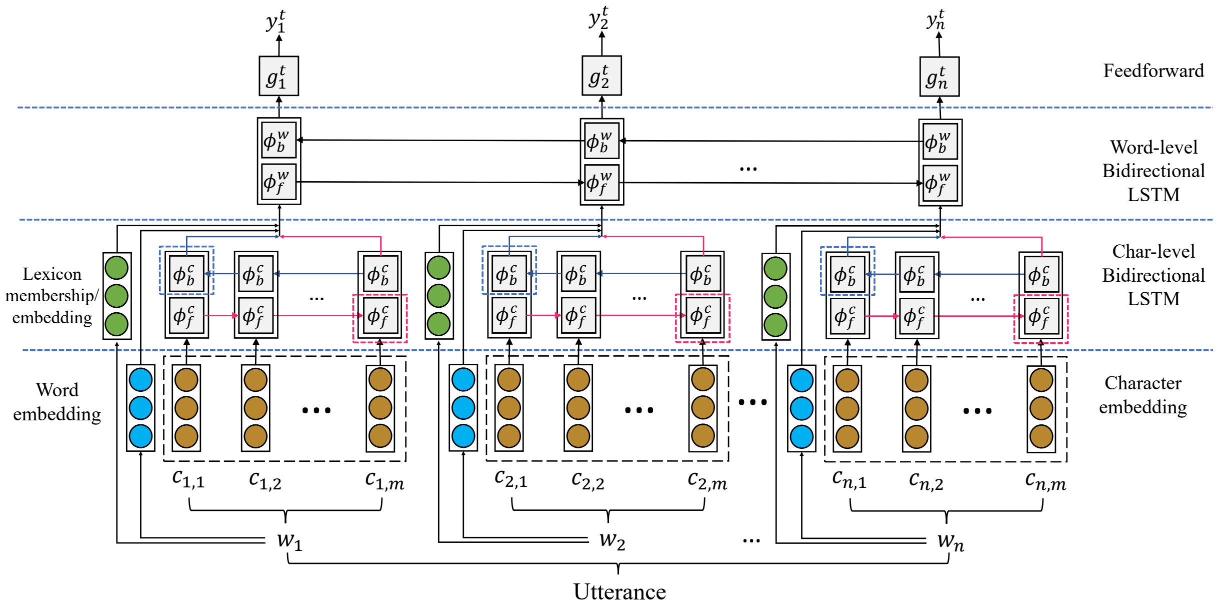

Figure 1: Overall network architecture

where p_i is the distribution of the true labels and q_i is the distribution of the predicted labels.

3.1 Lexicon Membership Model

Having described the basic form of our slot tagging model, we now describe how we extend the model to include lexicon features.

Assume that we have a collection of L lexicons, with each lexicon containing a collection of words and phrases belonging to that lexicon. For instance, a lexicon called *holidays* might contain: *Thanksgiving, Christmas, Labor Day*. Similarly, a lexicon called *days of the week* would contain *Monday, Tuesday, Wednesday*, etc. For each word w_i in an utterance we generate the unigram, bi-gram, and tri-gram beginning at word w_i, and we refer to this triple as t_i. For instance, for the utterance *"book a taxi to the airport"*, t_1 = ["book", "book a", "book a taxi"], t_2 = [a", "a taxi", "a taxi to"], etc. We use $t_i^j, j = 1, 2, 3$, to refer to the j-th element of t_i, i.e., the uni-gram, bi-gram, or tri-gram.

For each word w_i in the input utterance, we define a membership lexicon feature vector lex_i of length $|L|$, where each element $\text{lex}_i(l), l = 1, 2, ..., |L|$, is defined as:

$$\text{lex}_i(l) = \begin{cases} 1 & \text{if } t_i^j \text{ in lexicon } l, j = 1,2,3 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

In other words, every word w_i has an associated feature vector lex_i of length $|L|$. The k-th element of lex_i is 1 if a uni-gram, bi-gram, or tri-gram rooted at w_i exists in the k-th lexicon and is zero otherwise.

The word representation (Eq. 3) for each word is modified to include lexicon information:

$$v_i^{\text{lex}} = f_{|w_i|}^C \oplus b_1^C \oplus e_{w_i} \oplus \text{lex}_i. \quad (9)$$

The effect of this is to append a binary feature lexicon membership feature vector to the word representation. We then input the v_i^{lex} into the word LSTM layer to induce the character, lexicon, and context sensitive word representation. This can be seen visually in Figure 1 where the lexicon membership features are appended to the word embeddings and output of the character-level LSTM before being fed into the word-level LSTM. We refer to this model as the Lexicon Membership Model (M^{Member}).

3.2 Lexicon Embeddings Model

In the next model we propose to embed the lexicon information. Similar to the case of the word embeddings, we define lexicon embeddings as follows:

- Lexicon embedding: e_l for each $l \in L$,
- No Lexicon embedding: e_o,

that is, each of the $|L|$ lexicons is represented by an embedding and e_o represent an additional embedding, which is used in cases where a uni-gram, bi-gram, or tri-gram rooted at a word does not belong to any lexicon. We then define the embedded lexicon feature vector as:

$$\text{ELex}_i = \begin{cases} e_l & \text{if } t_i^j \text{ in lexicon } l, j = 1, 2, 3, l \in L \\ e_o & \text{otherwise.} \end{cases} \tag{10}$$

In other words, if a uni-gram, bi-gram, or tri-gram rooted at w_i appears in a lexicon l, we assign the embedding of lexicon l to ELex_i. If two or more of the n-grams rooted at a word w_i match a lexicon, we use the longest match.

We then adapt Eq. 9 to use the lexicon embeddings instead of the lexicon membership feature:

$$v_i^{\text{ELex}} = f_{|w_i|}^C \oplus b_1^C \oplus e_{w_i} \oplus \text{ELex}_i. \tag{11}$$

The effect of this is to append a lexicon embedding to the word representation. As was the case with before, we input the v_i^{ELex} into the word LSTM layer to induce the character, lexicon, and context sensitive word representation as in Eq. 4-6. This can be seen visually in Figure 1 where the lexicon embedding features are appended to the word embeddings and output of the character-level LSTM before being fed into the word-level LSTM. We refer to this model as the Lexicon Embeddings Model (M^{Embed}).

4 Experiments

We conduct experiments under three settings in order to evaluate how the lexicon models affect slot tagging performance.

- **All Training Data:** In this setting we use all of the available training data for each domain when training the slot tagging models.
- **Limited Training Data:** In this setting we simulate the constrained data scenario that often occurs when expanding an SLU system to support new domains and limit the amount of training data available.
- **Cross Domain:** In this setting we consider cross domain prediction and train on one domain and then predict common slots in other domains.

4.1 Data

We conduct all of our experiments on data from four domains belonging to an intelligent assistant. The source of the data is user utterances, which were transcribed by a speech-to-text system and then had their semantic slots labeled by trained annotators. The domains that we use are *Recipes*, *Services*, *Location*, and *Time*. The *Recipes* domain focuses on assisting users with recipes. The *Services* domain is used to help users find services, such as car repairs. The *Location* and *Time* domains identify location and time information in user utterances. These four domains differ in terms of the types of queries they contain, as well as their data sizes and the size of their lexicons. Table 1 shows details on the slots and amount of data available for each domain. As can be seen from the table, the results differ with the *Location* and *Time* domains having large amounts of training data available, while the *Services* domain has very limited data.

Domain	Slots	Train	Validation	Test
Recipes	10	20,000	6,809	1,186
Services	10	1800	194	500
Location	4	136,783	34,954	51,976
Time	4	129,340	34,644	46,803

Table 1: Description of datasets.

4.2 Lexicon Descriptions

We now describe the lexicons associated with each domain. The lexicons were created as part of the data pipeline for an intelligent assistant. Table 2 lists the domains and the number of lexicons associated with each of them. For the *Recipes* domain there are 3 lexicons: ingredients, recipe names, and cocktail names, with members such as: *banana pudding* and *mojito*. For the *Services* domain there is only 1 lexicon, which contains types of services, such as: *pet sitting*. The *Location* domain has 13 lexicons for countries, cities, schools, etc. Finally, the *Time* domain contains 20 lexicons for days of the week, holidays, etc.

The size of the lexicons also varies. For instance, the lexicons in the *Time* domain usually have tens of members. By contrast, in the *Location* domain, there are over 300,000 city names and only about 400 airport names. There are almost 30,000 recipes compared to about 300 cocktails and ingredients in the *Recipes* domain. Lastly, for the *Services* domain there are about 2,300 services types. As this analysis has shown, the properties of the lexicons vary largely among the datasets.

We also analyze the lexicon prevalence in the

Domain	Number of Lexicons	Prevalence
Recipes	3	28.95%
Services	1	29.78%
Location	13	99.13%
Time	20	81.63%

Table 2: Prevalence of lexicon features in training data.

training data. For the training data we compute how many utterances have a sequence of words that belongs to at least one lexicon. These results are shown in the last column of Table 2, where it can be seen that the prevalence of lexicons differs vastly across the domains. For instance, 99.13% of the training utterances in the *Location* domain contain a word or phrase that belongs to a lexicon. By comparison, for the *Recipes* domain, only 28.95% of utterances contain a word or phrase that belongs to a lexicon. As will be seen later, we generally see larger improvements in domains with larger lexicon coverage.

4.3 Methodology

Having described our models and data, we now describe our experimental methodology. For all experiments we randomly initialize all model parameters and shuffle the training dataset for each epoch. To account for randomization, we repeat each experiment 10 times and report the mean of the F-1 metric. To test for significance we make use of the Wilcoxon signed-rank test.

For each experiment, we allow for up to 30 epochs of training and employ early stopping when there is no improvement in the lowest loss on the validation set for 5 epochs. We use a batch size of 10 and set the learning rate to 5×10^{-4}. We set the dropout probability to 0.5. To train the network, we make use of stochastic gradient descent and the Adam optimization algorithm (Kingma and Ba, 2014). We train the network end-to-end to predict the slot tags for each utterance, thus allowing the network to learn the character, word and lexicon representations automatically.

Following previous studies, we set the size of the character and lexicon embeddings to 25 and the size of the word embeddings to 100. The character and lexicon LSTMs have 25 units and the word LSTMs have 100 units.To evaluate our model we report the F-1 score as adapted for entity recognition (Tjong Kim Sang and De Meulder, 2003).

Method	**Recipes**	**Services**	**Location**	**Time**
LSTM BL	91.20	78.11	84.79	94.05
CRF	90.21	74.51	86.07	93.43
CRF+Lex	90.69	75.47	**87.72**	93.55
M^{Embed}	**91.27**	77.37	86.01†	**94.28**†
M^{Member}	91.16	**78.13**	85.42	94.15

Table 3: F1 score for different models using full training set.

4.3.1 Baselines

We consider three baselines:

LSTM Baseline (LSTM BL): We consider a baseline LSTM model that does not include any lexicon information. This model is described by Equations 1-7.

Conditional Random Field (CRF): Linear chain CRF where we make use of n-gram features and brown cluster-based features.

Conditional Random Field + Lexicon Features (CRF+Lex): The same CRF as above, except we include binary features indicating lexicon membership.

4.4 Results

4.4.1 Full Dataset

This experiment uses all of the available training data. The results are shown in Table 3. For the *Recipes* domain, the highest F1 score is achieved by the M^{Embed} model; however, the difference is not statistically significant compared to the baseline LSTM. For this domain, the LSTM models all outperform the CRF models. For the *Services* domain, we observe that the M^{Member} model achieves the highest F1 score; however, it is also not statistically significant compared to the baseline LSTM. Once again, the LSTM models outperform the CRF models. For the *Location* domain we observe a statistically significant improvement in performance for the M^{Embed} model compared to the baseline LSTM. This improvement exceeds 1%. However, the CRF model outperforms the LSTM models. Lastly, for the the *Time* domain we observe a significant improvement in the F1 score compared to the baseline LSTM for the M^{Embed} model. Furthermore, all LSTM models outperform the CRF baselines.

The results in this experiment show that the M^{Embed} model achieves a statistically significant improvement over the baseline in two of the four

Method	Recipes	Services	Location	Time
Baseline	52.74	46.83	40.01	68.75
M^{Loss}	48.11	41.08	31.75	67.69
M^{Embed}	57.86	**52.54**†	51.40	**74.09**†
M^{Member}	**58.97**†	48.98	**52.78**†	70.97

Table 4: F1 score for different models using 1,000 samples during each training iteration.

Method	Location-Services	Time-Services
Baseline	20.09	84.91
M^{Loss}	20.24	85.42
M^{Embed}	19.98	**86.84**†
M^{Member}	**20.88**	86.65

Table 5: F1 score for models trained on LOCATION and TIME and tested on SERVICES.

datasets. As was previously discussed, these are datasets where the lexicons cover a majority proportion of the training data. Thus, the results indicate that the inclusion of lexicon information is useful if there is large lexicon coverage in the training data. As a general observation, the LSTM-based models tend outperform the CRF models in 3 of the 4 domains, which is similar to the findings of previous studies (Yao et al., 2014)

4.4.2 Limited Training Data

In this experiment, we investigate how the proposed models perform in the case of limited training data. We follow a similar methodology as before with the following changes: 1) during each training epoch we randomly sample 1,000 training samples; 2) we use a batch size of 1; 3) we lower the learning rate to 5×10^{-5}. The results of this experiment are shown in Table 4.

As can be seen from the table, there are large improvements in all domains when the lexicon-based models are used. In the *Recipes* domain the F1 score is 52.74% for the baseline and is 58.97% for the best performing M^{Member} model. For the *Services* model, the M^{Embed} model achieves an F1 score of 52.54% compared to 46.83% for the baseline. In the *Location* domain the F1 score is 12.77% higher than the baseline using the M^{Member} model, and for the *Time* domain the improvement is 5.34% better using the highest performing M^{Embed} model. The M^{Member} and M^{Embed} models thus achieve the highest F1 score on two domains each. However, it is interesting to note that when the M^{Embed} model outperforms the M^{Member} model is it usually by about 3-4%. By contrast, when the M^{Member} model performs best it usually only performs better than the M^{Embed} model by around 1%.

The results on these smaller datasets suggests that lexicons can have a significant effect on slot tagging performance when training data is limited. In these cases, the additional knowledge provided by the lexicons leads to large improvement in performance. This is an encouraging result for SLU systems that are being extended to new domains as it is sometimes possible to acquire lexicons at a low cost via sources such as DBPedia.

4.4.3 Cross Domain

In this experiment, we use the models trained for the *Location* and *Time* domains to predict common slots in the *Services* domain. Since some of the labels in the *Services* domain do not exist in the *Location* and *Time* models, we assign those labels a tag of Other. The results of this experiment are shown in Table 5. When training on *Location* and predicting *Services* the highest precision and F1 scores are achieved by the M^{Member} model; however, the improvement is not significant. When training on *Time* and predicting *Services* the highest performance is achieved by the M^{Embed} model and is statistically significant.

5 Discussion and Conclusion

Our results show that the M^{Embed} model, which represents lexicon information with embeddings, performs well across domains and experiments. For instance, it achieves a significantly better performance than the baselines in the *Location* and *Time* domains when all available training data is used. In that experiment, the *Location* and *Time* domains had relatively large lexicon coverage. The results suggest that lexicons can help improve performance when large amounts of training data are available and when lexicon coverage is high. For the *Services* and *Recipes* domains, where lexicon coverage was low, the lexicon-based models led to no significant improvement in performance.

When the training data was limited, the M^{Embed} model achieved significantly better performance than the baseline on 2 of the 4 domains. In the other 2 domains, the M^{Member} model performed best. The experiments showed that, when training data is small, the use of lexicons can lead to large improvement in slot tagging performance. For instance, Table 4 shows improvements in the F1 score of about 12% for the *Location* domain and of around 6% for the other domains.

These findings have strong implications for an industry setting. The experiments clearly show that lexicons can be very useful to improve slot tagging when training data is limited, as is the case when expanding to new domains. In these cases, practitioners can benefit greatly by acquiring lexicons from online sources, such as knowledge bases, or using existing lexicons that may have been previously collected. Lexicons can also be beneficial in cases where there are large amounts of training data, but only if the lexicon coverage is high. Our experiments show that using lexicons as embedding features generally leads to good improvements in a variety of situations.

References

Fabrice Dugas and Eric Nichols. 2016. DeepNNNER: Applying BLSTM-CNNs and Extended Lexicons to Named Entity Recognition in Tweets. In *Proceedings of the 2nd Workshop on User Generated Text*, pages 138–144.

Young-Bum Kim, Karl Stratos, and Dongchan Kim. 2017. Domain Attention with an Ensemble of Experts. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 643–653.

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980.*

Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. 2016. Leveraging sentence-level information with encoder lstm for semantic slot filling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2077–2083.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. *arXiv preprint arXiv:1603.01360.*

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. 2017. Investigation of Language Understanding Impact for Reinforcement Learning Based Dialogue Systems. *arXiv preprint arXiv:1703.07055.*

Xuezhe Ma and Eduard Hovy. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNS-CRF. *arXiv preprint arXiv:1603.01354.*

Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. 2015. Using Recurrent Neural Networks for Slot Filling in Spoken Language Understanding. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(3):530–539.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155.

Yangyang Shi, Kaisheng Yao, Hu Chen, Dong Yu, Yi-Cheng Pan, and Mei-Yuh Hwang. 2016. Recurrent Support Vector Machines For Slot Tagging In Spoken Language Understanding. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 393–399.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147.

Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi. 2014. Spoken Language Understanding Using Long Short-term Memory Neural Networks. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 189–194. IEEE.