Investigation of Recurrent-Neural-Network Architectures and Learning Methods for Spoken Language Understanding

Grégoire Mesnil [1,3], *Xiaodong He*[2], *Li Deng,*[2] *and Yoshua Bengio* [1]

[1] University of Montréal, Québec, Canada
[2] Microsoft Research, Redmond, WA, USA
[3] University of Rouen, France

{gregoire.mesnil|yoshua.bengio}@umontreal.ca, {xiaohe|deng}@microsoft.com

Abstract

One of the key problems in spoken language understanding (SLU) is the task of slot filling. In light of the recent success of applying deep neural network technologies in domain detection and intent identification, we carried out an in-depth investigation on the use of recurrent neural networks for the more difficult task of slot filling involving sequence discrimination. In this work, we implemented and compared several important recurrent-neural-network architectures, including the Elman-type and Jordan-type recurrent networks and their variants. To make the results easy to reproduce and compare, we implemented these networks on the common Theano neural network toolkit, and evaluated them on the ATIS benchmark. We also compared our results to a conditional random fields (CRF) baseline. Our results show that on this task, both types of recurrent networks outperform the CRF baseline substantially, and a bi-directional Jordan-type network that takes into account both past and future dependencies among slots works best, outperforming a CRF-based baseline by 14% in relative error reduction.

Index Terms: spoken language understanding, word embeddings, recurrent neural network, slot filling

1. Introduction

A major task in speech understanding or spoken language understanding (SLU) is to automatically extract semantic concept, or to fill in a set of arguments or "slots" embedded in a semantic frame, in order to achieve a goal in a human-machine dialogue. Despite many years of research, the slot filling task in SLU is still a challenging problem, in parallel with the intent determination task [23][27].

Until fairly recently, the main technical approaches to solving the slot filling problem in SLU included generative modeling, such as HMM/CFG composite models [25], and discriminative or conditional modeling such as conditional random fields (CRF) [13][26]. A few years ago, a new approach emerged for advancing speech recognition, based on deep learning, which involves many layers of nonlinear information processing in deep neural networks [11][6]. Subsequently these techniques have been applied to intent determination or semantic utterance classification tasks of SLU [24][7]. Deep learning has also been successfully applied to a number of other human language technology areas including language modeling [16][21], especially with the use of the naturally deep architecture of recurrent neural networks [15]. Among these progresses, one important advance is the invention of word embeddings [2], successfully projecting very-high-dimensional, sparse vector for word representations into a low-dimensional, dense vector representation for a variety of natural language tasks [3][4][15].

In light of the recent success of these methods, especially the success of recurrent neural networks for language modeling [15], we carried out an in-depth investigation of recurrent neural networks for the slot filling task of SLU. In this work, we implemented and compared several important recurrent neural network architectures, e.g., the Elman-type networks [8] [15] and Jordan-type networks [12] and their variations. To make the results easy to reproduce and rigorously comparable, we implemented these models using the common Theano neural network toolkit [1], and evaluated them on the standard ATIS (Airline Travel Information Systems) benchmark. We also compared our results to a baseline using conditional random fields (CRF). Our results show that on the ATIS task, both Elman-type networks and Jordan-type networks outperform the CRF baseline substantially, and a bi-directional Jordan-type network that takes into account of both past and future dependencies among slots works best.

2. The Slot Filling Task

Semantic parsing of input utterances in SLU typically consists of three tasks: domain detection, intent determination, and slot filling. Originating from call routing systems, domain detection or intent determination tasks are typically treated as semantic utterance classification. Slot filling is typically treated as a sequence classification problem after semantic templates for concept classes or "slots" are defined.

An example sentence is provided in Table 1, with domain, intent, and slot/concept annotations illustrated, along with typical domain-independent named entities. This example follows the popular in/out/begin (IOB) representation, where *Boston* and *New York* are the departure and arrival cities specified as the slot values in the user's utterance, respectively.

Sentence	*show*	*flights*	*from*	*Boston*	*to*	*New*	*York*	*today*
Slots/Concepts	O	O	O	B-dept	O	B-arr	I-arr	B-date
Named Entity	O	O	O	B-city	O	B-city	I-city	O
Intent	*Find_Flight*							
Domain	*Airline Travel*							

Table 1. ATIS utterance example IOB representation

For the slot filling task, the input is the sentence consisting of a sequence of words, and the output is a sequence of slot/concept IDs, one for each word. Traditionally, one of the most successful approaches for slot filling is the conditional random field (CRF) [13] and its variants. I.e., given the input word sequence $L_1^N = l_1, \dots, l_N$, the linear-chain CRF models the conditional probability of a concept/slot sequence $S_1^N = s_1, \dots, s_N$ as follows:

$$p(S_1^N | L_1^N) = \frac{1}{Z} \prod_{t=1}^{N} e^{H(s_{t-1}, s_t, l_{t-d}^{t+d})}$$

where

$$H(s_{t-1}, s_t, l_{t-d}^{t+d}) = \sum_{m=1}^{M} \lambda_m h_m(s_{t-1}, s_t, l_{t-d}^{t+d})$$

and $h_m(s_{t-1}, s_t, l_{t-d}^{t+d})$ are features extracted from the current and previous states s_t and s_{t-1}, plus a window of words around the current word l_t, with a window size of $2d + 1$.

3. Using RNNs for Slot Filling

3.1. Word embeddings

As an alternative to N-gram models, researchers came up with several different techniques based on learning Euclidean space structures for words. A real-valued embedding vector is associated with each word, and these embeddings are usually trained in an unsupervised way on a large corpus of natural language, e.g. Wikipedia. The architecture of these models can vary from shallow neural nets (NN) [19] or convolutional nets (SENNA) [4] to recurrent neural nets (RNN) [15]. The learned word embedding shows good generalization properties across many common natural language processing (NLP) tasks [4]. The neural network architectures evaluated in this paper are based on such word embeddings.

3.2. Short-term dependencies captured using a word context window

Without considering a temporal feedback, the neural network architecture corresponds to a simple feed-forward multi-layer perceptron (MLP), e.g., with a hidden layer and sigmoid activations. To capture short-term temporal dependencies in this setting, one can use a word-context window. With each word mapped to an embedding vector, the word-context window is the ordered concatenation of word embedding vectors. Here is an example of constructing the input vector with the word context window of size 3:

$$w(t) = [flights, \boldsymbol{from}, Boston]$$

$$'from' \rightarrow x_{from} \in \mathbb{R}^d$$

$$w(t) \rightarrow x(t) = [x_{flights}, x_{\boldsymbol{from}}, x_{Boston}] \in \mathbb{R}^{3d}$$

In the example, $w(t)$ is the 3-word context window around the *t*-th word 'from', x_{from} is the embedding vector of the word 'from', and d is the dimension of the embedding vector. Correspondingly, $x(t)$ is the ordered concatenated word embeddings vector for the words in $w(t)$.

In the feed-forward NN [11], the raw input vector *x* is first fed into the hidden layer *h*. After a non-linear transformation, the output of the hidden layer *h* is then fed into the output layer to generate the final output *y*.

3.3. Two types of RNN architectures

At the highest level of complexity for slot filling, one has to take into account the slot/concept dependencies (sequences of labels) beyond the words surrounding the word of interest (the context word window that captures short-term dependencies). Here we first describe two variants of RNNs for modeling slot sequences: the Elman-type RNN [8] and the Jordan-type RNN [12]. Using RNNs to model long-term dependencies will be presented in the next sub-section. In contrast to a feed-forward NN, in the Elman-type RNN, the output from the hidden layer at time *t*-1 is be kept and fed back to the hidden layer at time *t*, together with the raw input $x(t)$ for time *t*. This can be interpreted as having an additional set of virtual "context nodes", where there are connections from the hidden layer to these context nodes fixed with a weight of one. At each time step, the input is propagated in a standard feed-forward fashion, and then a parameter updating rule is applied, taking into account the influence of past states through the recurrent connections. In this way, the context nodes always maintain a copy of the previous values of the hidden nodes, since these propagate through the recurrent connections from time *t*-1, before the updating rule is applied at time *t*. Thus the network can maintain and learn a sort of state summarizing past inputs, allowing it to perform tasks such as sequence-prediction that are beyond the power of a standard feed-forward NN. Precisely, dynamics of the Elman-type RNN can be represented mathematically by

$$h(t) = f\big(Ux(t) + Vh(t-1)\big)$$

where we use the sigmoid function at the hidden layer:

$$f(x) = \frac{1}{1 + e^{-x}}$$

and a softmax function at the output layer:

$$y(t) = g\big(Wh(t)\big), \qquad where \;\; g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

where *U* and *V* are weight matrices between the raw input and the hidden nodes, and between the context nodes and the hidden nodes, respectively, while *W* is the output weight matrix. The input vector $x(t)$ has as dimensionality that of the word embeddings d multiplied by the number of words in the context window. $h(t)$ corresponds to the hidden layer, and $y(t)$ has as dimensionality the number of classes; i.e., 127 for the ATIS slot filling task.

Jordan-type RNNs are similar to Elman-type networks, except that the context nodes are fed from the output layer instead of from the hidden layer. The context nodes in a Jordan-type network are also referred to as the state layer. The difference between Elman and Jordan-type networks appears only in the hidden layer input:

$$h(t) = f(Ux(t) + Vy(t-1))$$

3.4. Long-term dependencies captured using a RNN

To capture dependencies beyond the input window, we need to exploit the time-connection feedback, giving rise to the RNN architectures. Learning long-term dependencies with RNNs raises an optimization problem known as the vanishing gradient problem. Indeed, capturing longer time dependencies correspond to training a deeper model when the RNN is unfolded in time (i.e., each time instance of a layer being a separate layer of a deep net). Rather than training classical RNNs in this way, we can directly provide some of the past information from different time steps. Instead of relying only on learning through one-step recurrences to capture context, one can combine recurrence with the idea of input window. This is achieved by feeding the network with concatenation of the *T* previous time steps vectors (from the output layer as in the Jordan-type network or the hidden layer as in the Elman-type network) in addition to the use of word context windows. This also provides a way to obtain the most accurate number of time steps to consider for a particular task. In the case of Elman-type networks, the feedback from the output layer leads to the following backward model (predicting from the future to the past) and forward model (predicting from the past to the future):

$$h_{backward}(t) = f(Ux(t) + \sum_{k=1}^{T} V_k h(t+k))$$

and

$$h_{forward}(t) = f(Ux(t) + \sum_{k=1}^{T} G_k h(t-k))$$

Further, since having only backward or forward time dependencies uses only partial information available, it would be helpful to consider both past and future dependencies together. Bi-directional Elman-type RNNs have been studied in the past [18] and in this paper, we consider variants of bi-directional Jordan-type RNNs:

$$h_{bidir}(t) = f(Ux(t) + \sum_{k=1}^{T} V_k y_b(t+k) + \sum_{k=1}^{T} G_k y_f(t-k))$$

where y_b and y_f denote the output of a backward model and a forward model in the Jordan-type RNN, respectively.

3.5. Learning methods

3.5.1. Fine-tuning word embedding

Once the word embeddings have been learned in an unsupervised fashion [3][15], it is possible to fine-tune them during supervised training on the task of interest. Actually, this is double-edged: the model could fit the task better but the risk of overfitting may arise. We compare both cases experimentally in Section 4.

3.5.2. Sentence-level and word-level gradient descents

The average length of a sentence in the ATIS data set is about 15 words. With such relatively long inputs, training RNN models could be tricky if updates are done online (i.e., after each word). This is because the predictions at the current word have been made with model parameters that are no longer current, and the sequence of predictions does not correspond to the one that could be performed with a fixed parameter set. For instance, if we want to predict or perform an update at the 17th slot of a sentence with a forward RNN model, we would have to re-compute all the values from the beginning of the sentence in order to get "correct" predictions consistent with the current model parameters.

For training the Elman-type RNN, one option to prevent the above problem is to perform mini-batch gradient descent with exactly one sentence per mini-batch. For a given sentence, we perform one pass that computes the mean loss for this sentence and then perform a gradient update for the whole sentence. This approach performs well even if the mini-batch size varies for the sentences with different lengths.

A similar learning technique has also been applied for training the Jordan-type RNN, which corresponds to performing parameter updates after each word. Like in the mini-batch case, we compute all slot values for the sentence in one pass. Then, we keep this history of values as an approximation to the exact values and perform one gradient step for each word. In the experiments, we have observed fast convergence between the exact and approximate slot values.

3.5.3. Dropout regularization

We found that training bi-directional RNNs on the slot/concept predictions of previously trained RNNs gave us poor generalization results due to overfitting. In order to address this issue, we implemented a recently introduced regularization technique called dropouts [10] that omits a given proportion of the hidden nodes for each training sample as well as parts of the input vector. Experimental results show that it allows us to improve the performance of the bi-directional RNN over regular RNNs.

4. Experimental Evaluation

We use the ATIS corpus as used extensively by the SLU community, e.g. [9][17][22]. The training set contains 4978 utterances selected from Class A (context independent) training data in the ATIS-2 and ATIS-3 corpora, while the test set contains 893 utterances from the ATIS-3 Nov93 and Dec94 datasets. In the evaluation, we only use lexical features in the experiments.

4.1. Corpus for learning word embeddings

The methods and data used for learning word embeddings might impact performance on the slot filling task we are interested in. In order to evaluate that impact, different procedures for learning word embeddings have been considered, including SENNA [4] and RNNs [14]. For SENNA, we directly download the embeddings pre-trained on the Wikipedia corpus. RNN word embeddings were obtained by running the RNNLM toolkit available online [14]. We also took into account the dimension *d* of the embedding as a factor of variation. For data, we consider three semantically different corpora for embedding training: Wikipedia, Gutenberg, and Broadcast news. A Wikipedia snapshot is downloaded from [20]. Gutenberg corresponds to digitized books with expired copyrights which we downloaded and built ourselves. For the Broadcast news corpus, we directly downloaded the word embeddings provided by [14] on the RNNLM website.

4.2. Results on word embeddings

To evaluate the impact on SLU of learning methods and data for word embedding training, we test different types of word embeddings trained on different corpora and by different methods. We first consider a frame-level MLP setting (e.g., a feed-forward MLP with inputs of only word embedding features within a word context window), and we compare results of using the embeddings as is versus fine-tuning them during training. Results are also compared with randomly initialized word embeddings of various dimensions $\{50, 80, 100\}$ which are fine-tuned during training. In the experiment, an MLP with 1000 hidden nodes is trained. 1000 sentences of the original ATIS training set are held out as a validation set for choosing the best word context window size and hyper-parameters. The model is then re-trained with the best set of hyper-parameters on the whole ATIS training set.

method	Corpus	embedding's dimension	w/o fine-tuning	w/ fine-tuning
SENNA	Wikipedia	50	92.01	**92.38**
RNNLM	Wikipedia	100	90.51	90.61
	Gutenberg	100	90.20	90.31
	Broadcast	80	90.14	90.58
Random	N/A	50	N/A	90.26
		80		90.94
		100		90.81

Table 2: F1 score on the ATIS task for different methods and training corpora for the embeddings, with the corresponding dimension d of word embeddings.

We first observe from Table 2 that fine-tuning word embeddings is helpful, improving results by a small but consistent margin across the board. We also find that SENNA embeddings gives the best performance. We hypothesize that this may essentially be due to a conditioning issue, since the norm of SENNA word vectors is kept normalized to 1 during training while it is not the case for RNNLM. As indicated by the RNNLM results, word embeddings trained on a semantically different corpus (Wikipedia, Gutenberg, and Broadcast) lead to similar performance, and the differences become even smaller after fine-tuning. In later experiments, we use the embeddings from SENNA.

4.3. Results on Jordan-RNN

There are several choices for feeding the Jordan-type RNN with outputs from previous or future time steps. The first option takes the output probabilities of the NN. Intuitively, probabilities would allow the model to perform a kind of disambiguation since no hard decision is made. The second option considers the hard decision of the model in both the training and testing phases. The third option differs from the last choice during training: the model is trained with ground truth labels, while at test time, since no ground truth labels are available, the hard decisions are used.

For all these options, we measure the precision, recall and F1-score using the conlleval.pl script [22] and compare it to a CRF baseline. The CRF hyper-parameters, i.e., window sizes and regularizers, have been chosen using 5-fold cross validation on the ATIS training set. We use the CRFpp toolkit [5] to run these experiments. Results are reported in Table 3. All the Jordan-type RNNs outperform the CRF baseline. As expected, the probability-based J-RNN outperforms the hard-decision-based version. More interestingly, the forward Jordan-type RNN trained with ground-truth labels obtains the best performance although there is a condition mismatch (e.g., no ground truth label is available in testing). This may be because training with model-predicted labels, either in the hard-decision form or the probability form, could introduce unnecessary noise or convergence difficulty in training.

Model	Precision	Recall	F1-score
CRF baseline	94.08	91.82	92.94
Prob. – (past)	92.93	93.66	93.39
Prob. – (future)	92.93	93.58	93.26
Hard – (past)	92.52	93.76	93.14
Hard – (future)	92.55	93.76	93.15
Ground – (past)	93.42	94.11	93.77
Ground – (future)	92.76	93.87	93.31

Table 3: results on several choices of sequential inputs in the Jordan-type RNN predicting from the past/future i.e., forward/backward: probabilities, hard decisions or ground-truth during training and hard decisions for testing.

4.4. Results on slot filling accuracy

We compare the performance of the introduced RNNs and CRF at the sequential level, along with a frame-level MLP and a Logistic Regression models. Since the NN-based models use word embeddings that leverage unsupervised information from Wikipedia, we clustered all the words in Wikipedia into 200 clusters and add a cluster ID for each word as a discrete feature to the CRF and the Logistic Regression models to make the results comparable. As before, baselines have been trained with CRFpp with 5-fold cross validation for the regularization parameter and the optimal window size.

Experimental results in Table 4 show that models that consider sequential dependency outperform models that don't, and the RNN models consistently outperform the CRF model. We also observe that the Elman-type RNN's forward version (e.g., use past information) performs very well while its backward version (e.g., use future information) gives worse results, though mathematically these two versions are symmetric to each other. Further analysis of the ATIS dataset shows that most of the concept slots to be predicted in ATIS are located in the second half of sentences, which makes the backward model perform predictions with very little historical information. This is also shown by the best hyper-parameters found for the Elman-type RNN which included a window of size 3 for the forward model and 13 for the backward model. The backward model was trying to get the historical information inside the word context window while it was available in the hidden layer for the forward model.

The Jordan-type RNNs, although giving similar results to the forward Elman-type RNN, has shown to be more robust to this problem. Further, the bi-directional version of the Jordan-type RNN improves upon both the backward and forward Jordan models. It is trained with dropout regularization [9] and rectifiers nodes i.e., $f(x) = \max(0, x)$, for 700 epochs and with a batch size of 100. Input nodes are dropped out with a probability $p = 0.2$ while for hidden nodes we used $p = 0.5$. Compared to the CRF+Wiki baseline, it yields an absolute improvement of the F1 score of 0.98%, corresponding to a relative error reduction of 14%.

Models	Prec.	Rec.	F1
Logistic Regress.	91.54	90.73	91.13
Logistic Regress.+Wiki	91.82	91.82	91.82
Frame-NN	92.17	92.59	92.38
CRF	94.08	91.82	92.94
CRF+Wiki	93.77	92.25	93.00
Elman-RNN (past)	93.25	94.04	93.65
Elman-RNN (future)	91.75	92.49	92.12
Jordan-RNN (past)	93.42	94.11	93.77
Jordan-RNN (future)	92.76	93.87	93.31
Bi-dir. Jordan-RNN	93.82	94.15	**93.98**

Table 4: Detailed performance measures (precision, recall, and F1 score) for a set of models evaluated on ATIS.

5. Conclusion and Discussion

We carried out comprehensive investigations of RNNs for the task of slot filling in SLU. We implemented and compared several RNN architectures, including the Elman-type and Jordan-type networks with their variants. We also studied the effectiveness of word embeddings for slot filling. To make the results easy to reproduce and to compare, we implemented all networks on the common Theano neural network toolkit, and evaluated them on the ATIS benchmark. Our results show that both Elman and Jordan-type networks outperform the CRF baseline substantially, both giving similar performance. A bi-directional version of the Jordan-RNN gave the best performance, outperforming the CRF-based baseline by 14% in relative error reduction. Future work will explore more efficient training of RNNs and the choice of more comprehensive features [28] and using a different RNN training toolkit [14] incorporating more advanced features.

6. References

[1] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio, "Theano: A CPU and GPU Math Expression Compiler," *Proc. Python for Scientific Computing Conference (SciPy) 2010*.

[2] Y. Bengio, R. Ducharme and P. Vincent, "A Neural Probabilistic Language Model", in NIPS 2000

[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in ICML 2008.

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," in Journal of Machine Learning Research, vol. 12, 2011.

[5] CRPpp: http://crfpp.googlecode.com/svn/trunk/doc/index

[6] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition, in *IEEE Transactions on Audio, Speech, and Language Processing*," vol. 20, no. 1, pp. 30-42, January 2012.

[7] L. Deng, G. Tur, X. He, and D. Hakkani-Tur, "Use of Kernel Deep Convex Networks and End-To-End Learning for Spoken Language Understanding," IEEE Workshop on Spoken Language Technologies, December 2012.

[8] J. Elman, "Finding structure in time," in Cognitive Science, 14 (2), 1990.

[9] Y. He and S. Young, "A data-driven spoken language understanding system," in IEEE ASRU 2003.

[10] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv: 1207.0580v1, 2012.

[11] G. Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath,, and Brian Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," in *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82-97, Nov. 2012.

[12] M. Jordan, "Serial order: A parallel distributed processing approach," in Tech. Rep. No. 8604. San Diego: University of California, Institute for Cognitive Science.

[13] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in ICML 2001.

[14] T. Mikolov, http://www.fit.vutbr.cz/~imikolov/rnnlm/

[15] T. Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur, "Extensions of recurrent neural network based language model," in ICASSP 2011.

[16] A. Mnih and G. Hinton, "A scalable hierarchical distributed language model" in NIPS, 2008, pp. 1081-1088.

[17] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding," in Interspeech 2007.

[18] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," in IEEE Transactions on Signal Processing, November 1997.

[19] H. Schwenk and J-L. Gauvain, "Training neural network language models on very large corpora," in HLT/EMNLP 2005.

[20] C. Shaoul and C. Westbury. 2010. The Westbury lab wikipedia corpus.

[21] Socher, R., Lin, C., Ng, A., and Manning, C. "Learning continuous phrase representations and syntactic parsing with recursive neural networks," Proc. ICML, 2011.

[22] G. Tur, D. Hakkani-Tur, and L. Heck, "What is left to be understood in ATIS?" in IEEE SLT, 2010.

[23] G. Tur and L. Deng, "Intent Determination and Spoken Utterance Classification," in Chapter 4, Tur and De Mori (eds). Spoken Language Understanding: Systems for Extracting Semantic Information from Speech, pp. 81-104, Wiley, 2011

[24] G. Tur, L. Deng, D. Hakkani-Tur, and X. He, "Towards Deeper Understanding Deep Convex Networks for Semantic Utterance Classification," in ICASSP, 2012.

[25] Y. Wang, L. Deng, and A. Acero, "Spoken Language Understanding — An Introduction to the Statistical Framework," *IEEE Signal Processing Magazine*, vol. 22, no. 5, pp. 16-31, 2005.

[26] Y. Wang, L. Deng, and A. Acero, "Semantic Frame Based Spoken Language Understanding," in Chapter 3, Tur and De Mori (eds) Spoken Language Understanding: Systems for Extracting Semantic Information from Speech, , pp. 35-80, Wiley, 2011.

[27] S. Yaman, L. Deng, D. Yu, Y. Wang, and A. Acero, "An integrative and discriminative technique for spoken utterance classification," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 16, no. 6, pp. 1207–1214, 2008.

[28] K. Yao, G. Zweig, M-Y. Hwang, Y. Shi, D. Yu, "Recurrent neural networks for language understanding", submitted to Interspeech 2013.