INTERSPEECH 2016
September 8–12, 2016, San Francisco, USA

Labeled Data Generation with Encoder-decoder LSTM for Semantic Slot Filling

Gakuto Kurata, Bing Xiang, Bowen Zhou

IBM Watson

gakuto@jp.ibm.com, {bingxia, zhou}@us.ibm.com

Abstract

To train a model for semantic slot filling, manually labeled data in which each word is annotated with a semantic slot label is necessary while manually preparing such data is costly. Starting from a small amount of manually labeled data, we propose a method to generate the labeled data with using the encoder-decoder LSTM. We first train the encoder-decoder LSTM that accepts and generates the same manually labeled data. Then, to generate a wide variety of labeled data, we add perturbations to the vector that encodes the manually labeled data and generate labeled data with the decoder LSTM based on the perturbated encoded vector. We also try to enhance the encoder-decoder LSTM to generate the word sequences and their label sequences separately to obtain new pairs of words and their labels. Through the experiments with the standard ATIS slot filling task, by using the generated data, we obtained improvement in slot filling accuracy over the strong baseline with the NN-based slot filling model.

Index Terms: spoken language understanding, semantic slot filling, labeled data generation, encoder-decoder LSTM, ATIS

1. Introduction

Semantic slot filling is an essential component of Spoken Language Understanding (SLU) [1]. Slot filling can be framed as a sequential labeling problem in which the most probable semantic slot labels are estimated for each word of the given word sequence. Slot filling is a traditional task and tremendous efforts have been done, especially since the 1980s when the Defense Advanced Research Program Agency (DARPA) Airline Travel Information System (ATIS) projects started [2]. In the 1990s, the dominant methods were rule-based approaches [3] and statistical approaches [4]. In the early 2000s, discriminative training was leveraged for slot filling and Conditional Random Fields (CRF) particularly performed well [5] since CRF can explicitly model label dependencies. Following the success of deep learning [6, 7], many researchers have been applying deep learning for slot filling. Recurrent Neural Network (RNN) [8, 9] and one of its specific architectures, Long Short-Term Memory (LSTM) [10], have been widely used since they can capture temporal dependencies through their recurrent hidden states [11, 12, 13].

To train recently proposed models for slot filling, large amount of labeled training data is necessary. Figure 1 is an example of labeled data in the ATIS corpus. However, since manually labeling data is costly and time-consuming, the amount of the manually labeled data is limited. After the SLU system is publicly launched, real user input can be collected. By automatically labeling these input with the model trained from the existing labeled data, we can augment the training data. Instead, in this paper, we focus on the very initial stage of the SLU system deployment when only a small amount of manually labeled data is available and unlabeled data is not available. We attempt to generate labeled data to augment the training data for the slot filling model with using the manually labeled data.

Sentence	show	flights	from	Boston	to	New	York	today
Slots	O	O	O	B-FromCity	O	B-ToCity	I-ToCity	B-Date

Figure 1: *Example of ATIS sentence and annotated slots labels.*

There has been previous research to generate word sequences by training RNN [14] or LSTM [15] from the existing text data. By enhancing this idea for labeled data generation, [16] proposed a novel method to train RNN by using the pair of word and its label as a unit of modeling, and generate sequences of words and their labels. One shortcoming of this method is that the generated sequences were not effective to improve the competitive NN-based slot filling model in the ATIS domain.

In this paper, we propose to use the encoder-decoder LSTM [17] to generate labeled data. The encoder-decoder LSTM was originally proposed in machine translation and then applied to grapheme-to-phoneme conversion [18], automatic speech recognition [19, 20] and so on. The encoder LSTM first encodes the input sequence into its last hidden state and then the decoder LSTM generates the output sequence conditioned on the encoded information. By using the manually labeled training data, we first build the autoencoder with the encoder-decoder LSTM [21] that uses the pair of word and its label as a modeling unit. Then by feeding the training data to the trained encoder-decoder LSTM, we generate the labeled sequences. Since the encoder-decoder LSTM is very powerful, the sequences that are same with the input sequences are tend to be re-constructed. However, our purpose is to generate *similar* sequences and not to re-construct the original sequences. To obtain a wide variety of sequences, we propose to add random perturbation to the encoded vector when generating the sequences as described in Figure 2. Another advantage of the encoder-decoder is that various types of sequences and arbitrary modeling units can be used as input and output. Starting from a simple encoder-decoder LSTM that accepts pairs of word and its label and also outputs the pairs, we tried various architecture such as accepting sequences of pairs and generating word and label sequences separately, as shown in Figure 3. By using separate output sequences for words and labels, new pairs of words and labels that are not included in the training data can be generated.

We conducted experiments using the standard ATIS data set and confirmed that the generated labeled data improved the sufficiently competitive slot filling model.

The main contributions of this paper are three-folds:

- Proposed a method to generate labeled data using the encoder-decoder LSTM with adding perturbation to the encoded vector.
- Enhanced the encoder-decoder LSTM to generate word and label sequences separately to obtain new pairs of

Copyright © 2016 ISCA

http://dx.doi.org/10.21437/Interspeech.2016-727

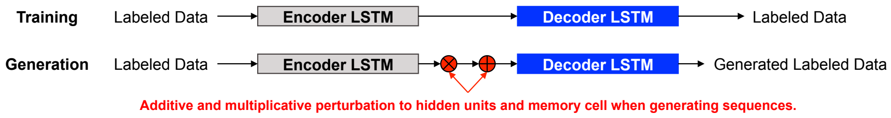


Figure 2: *Overview of proposed method. When training, the input sequence and the output sequence are identical and perturbation is not added. When generating, perturbation is added to the hidden units and memory cells that encodes the input sequence, and the output sequence different from the input sequence can be generated by the decoder LSTM.*

word and label that were not in the training data.

- Confirmed that the generated data was effective to improve slot filling model.

2. Encoder-decoder LSTM

We briefly revisit the LSTM architecture we use in this paper and the encoder-decoder LSTM.

2.1. LSTM

LSTM is a specific architecture of RNN and is easier to train thanks to its internal memory cells and gates. There are some variants in LSTM architectures and we used the architecture specified as below, which is similar to the architectures of [15, 22].

$$\begin{aligned}
i_t &= \tanh(W_{xi}x_t + W_{hi}h_{t-1} + b_i)\\
j_t &= \mathrm{sigm}(W_{xj}x_t + W_{hj}h_{t-1} + b_j)\\
f_t &= \mathrm{sigm}(W_{xf}x_t + W_{hf}h_{t-1} + b_f)\\
o_t &= \mathrm{sigm}(W_{xo}x_t + W_{ho}h_{t-1} + b_o)\\
c_t &= c_{t-1} \odot f_t + i_t \odot j_t\\
h_t &= \tanh(c_t) \odot o_t
\end{aligned}$$

x_t is the input to the LSTM at time step t, W_* are the weight matrices, and b_* are the bias vectors. $\odot$ denotes an element-wise product. c_t and h_t represent the memory cell vector and the hidden vector at the time step t. Note that this LSTM does not have peephole connections.

2.2. Encoder-decoder LSTM

The encoder-decoder LSTM is a general method to map the input and the output sequences with arbitrary length [17, 23, 24]. It first encodes the input sequence into a fixed-sized vector using one LSTM (encoder LSTM) and then generate the output sequence by decoding the encoded vector by another LSTM (decoder LSTM) whose hidden parameters are initialized with the encoded vector by the encoder LSTM. The LSTM parameters were optimized to maximize the log probability of the output sequences given the input sequences. When generating the sentences using the trained encoder-decoder LSTM, the most likely output sequence given the input sequence is generated. We used the simple left-to-right beam search decoder [17] when generating sequences.

3. Proposed Method

We explain our proposed method to generate the labeled data with using the encoder-decoder LSTM as depicted in Figure 2. When training the encoder-decoder model, we use the same labeled data in the training data for input and output. When generating labeled data, we first encode the labeled data in the training data by the encoder LSTM, add perturbation to the hidden vector and the memory cell, and then generate the labeled data with the decoder LSTM. We detail our idea to generate a wide variety of labeled data by adding perturbation and using various architectures of encoder-decoder models.

3.1. Perturbation to hidden vector and memory cell

While our purpose is to generate new labeled data, the encoder-decoder LSTM tends to re-generate the output sequence that is identical to the input sequence due to its novel modeling capability. We propose to add two types of perturbation, additive and multiplicative perturbation, to the encoded information before generating sequences. More concretely, assuming that the length of the input labeled data is N, the encoder LSTM encodes its information to the hidden vector h_N and the memory cell c_N.

Additive perturbation

$$\begin{aligned}
\hat{h_N} &= h_N + A^h\\
\hat{c_N} &= c_N + A^c\ ,
\end{aligned}$$

where A^h and A^c are vectors whose size are same with h_N and c_N and consisting of independent random variables sampled from an uniform distributions between $-p_a$ and p_a.

Multiplicative perturbation

$$\begin{aligned}
\hat{h_N} &= h_N \odot M^h\\
\hat{c_N} &= c_N \odot M^c\ ,
\end{aligned}$$

where M^h and M^c are vectors whose size are same with h_N and c_N and consisting of independent random variables sampled from an uniform distributions between $1-p_m$ and $1+p_m$. $\odot$ denotes an element-wise product.

When generating sequences, the hidden state and the memory cell of the decoder LSTM are initialized with $\hat{h_N}$ and $\hat{c_N}$ instead of h_N and c_N. By adding perturbation, we can expect that sequence that are not identical to the input sequence is generated. Since the sequence is generated with being constrained by the decoder LSTM, sequences similar with the labeled training data can be generated.

3.2. Various architectures

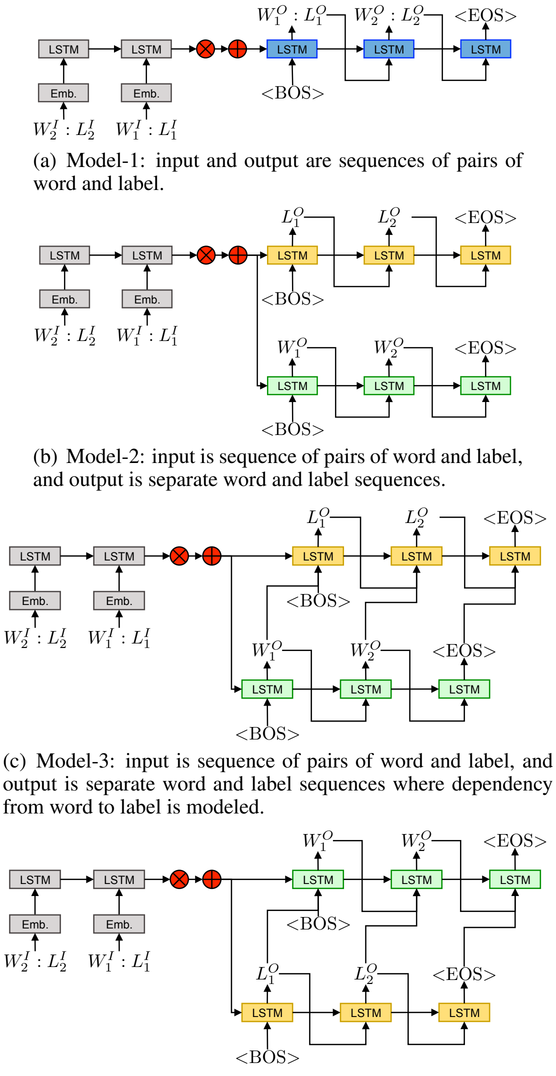
We explain the various architectures for labeled data generation as shown in Figure 3.

3.2.1. Model-1 (Figure 3(a))

Model-1 uses the standard encoder-decoder LSTM with using the pair of word and label as a modeling unit. When training, the loss function is set to maximize the log probability of the output sequence of pairs of word and label. While this Model-1 can only generates the pair appearing in the training data, a wide variety of sequences can be generated by adding perturbation.

3.2.2. Model-2 (Figure 3(b))

Model-2 has the same encoder LSTM with the Model-1, but has two independent output sequences for words and labels. This means that this model has two different decoder LSTMs for word sequences and label sequences. Two loss functions to maximize the log probabilities of the output word sequences



(a) Model-1: input and output are sequences of pairs of word and label.

(b) Model-2: input is sequence of pairs of word and label, and output is separate word and label sequences.

(c) Model-3: input is sequence of pairs of word and label, and output is separate word and label sequences where dependency from word to label is modeled.

(d) Model-4: input is sequence of pairs of word and label, and output is separate word and label sequences where dependency from label to word is modeled.

Figure 3: *Various types of encoder-decoder models for labeled data generation. Input is manually labeled data of word sequence* $\boldsymbol{W^I} = \{W_1^I, W_2^I\}$ *whose labels are* $\boldsymbol{L^I} = \{L_1^I, L_2^I\}$. *The order of the input sequences are reversed when encoded by the encoder LSTM. Output is* $\boldsymbol{W^O} = \{W_1^O, W_2^O\}$ *whose labels are* $\boldsymbol{L^O} = \{L_1^O, L_2^O\}$. *When training the model, input and output are identical as* $\boldsymbol{W}^I = \boldsymbol{W}^O$ *and* $\boldsymbol{L}^I = \boldsymbol{L}^O$ *and perturbations are not added to the hidden vector and the memory cell. Please note that the length of output sequences is not always the same with that of input sequences when generating sequences.* $W_*^* : L_*^*$ *stands for pair of word and label. "<BOS>" and "<EOS>" are beginning and ending symbols for output sequences. "Emb." and "LSTM" boxes represent embeddings and LSTM layers. Softmax layer on top of LSTM layer in decoder LSTM is not depicted for clarity.*

and the output label sequences are summed up for training. Since output word sequences and label sequences are independently generated, only the combinations of generated word and label sequences that have the same length are used as the generated labeled data.

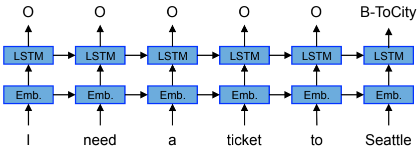


Figure 4: *LSTM-based slot filling model. Softmax layer on top of LSTM layer is not depicted for clarity.*

3.2.3. Model-3 (Figure 3(c))

Model-3 is an enhanced version of Model-2. We feed the generated word from the lower output sequence to the decoder LSTM for label sequence generation as shown in Figure 3(c).

3.2.4. Model-4 (Figure 3(d))

Model-4 is another enhanced version of Model-2. The output sequences in this model are reversed from Model-3, where the generated label from the lower output sequence is fed to the decoder LSTM for word sequence generation as shown in Figure 3(d).

4. Experiments

We first explain the experimental setup and procedure. Then we report the performance of the slot filling model trained by using the generated data and add analysis on the generated data.

4.1. Experimental setup and procedure

We used the ATIS corpus, which has been widely used as the benchmark for SLU [2, 25, 26, 27]. Figure 1 shows an example sentence and its semantic slot labels in In-Out-Begin (IOB) representation. The slot filling task was to predict the slot label sequences from input word sequences.

The ATIS corpus contains the official split of the training data of 4,978 sentences and evaluation data of 893 sentences. The unique number of slot labels is 127 and the vocabulary size is 572. We randomly selected 80% of the original training data, calling it as *train-set*, and named the remaining 20% as *heldout-set*. The original evaluation data is called as *evaluation-set*.

The basic experimental flow is as follows:

1. Train the encoder-decoder model for labeled data generation using the train-set. The same data in the train-set is used for the input and output when training the encoder-decoder model.
2. Generate the sentences using the trained encoder-decoder model and the labeled data in the train-set. 4 labeled sequence are generated from each manually labeled sequence in the train-set.
3. Train the LSTM-based slot filling model with using the train-set and the labeled data generated in the previous step. The total size of the training data is 5 times larger than the original train-set.
4. Evaluate the performance of the LSTM-based slot filling model on the evaluation-set.

The performance of slot filling was measured by the F_1-score: $F_1 = \frac{2\times Precision\times Recall}{Precision+Recall}$, where precision is the ratio of the correct labels in the system's output and recall is the ratio of the correct labels in the ground truth of the evaluation data [28].

As for the model for slot filling, we used the LSTM-based slot filling model, that has been reported to achieve sufficiently competitive F_1-score [12, 29]. The architecture of the LSTM-based slot filling model is depicted in Figure 4.

Perturbation		F_1-score
Baseline	-	94.77
No Perturbation	-	94.85
Additive	$p_a = 0.2$	**95.08**
	$p_a = 0.4$	94.53
	$p_a = 0.6$	94.87
	$p_a = 0.8$	94.76
Multiplicative	$p_m = 0.2$	94.90
	$p_m = 0.4$	**95.31**
	$p_m = 0.6$	94.66
	$p_m = 0.8$	94.26

Table 1: *F_1-score with data generation by Model-1. "Baseline" did not use any additional training data. "No Perturbation" used Model-1, but did not add perturbation. "Additive" perturbations were sampled from $[-p_a, +p_a]$. "Multiplicative" perturbations were sampled from $[1 - p_m, 1 + p_m]$.*

For training the encoder-decoder model and the LSTM-based slot filling model, we randomly initialized parameters in accordance with the normalized initialization [30]. We used *ADAM* for learning rate control [31] and *dropout* for generalization with a dropout rate of 0.5 [32, 33].

From a pilot experiment, we found that a small encoder-decoder model yields collapsed sequences by adding perturbation, which is reasonable since the encoded information is packed into a small size of vector and it is sensitive to the perturbation. Thus, we set the number of hidden units to sufficiently large size, 1,024, for the encoder-decoder model. The dimension of the embeddings for the pair of word and label was set to 64.

The hyper-parameters of the LSTM-based slot filling model were chosen according to the heldout-set, which means that we reported the F_1-score on the evaluation-set with the hyper-parameters that achieved the best F_1-score on the heldout-set.

4.2. Experimental result

Table 1 shows the results with trying various perturbation values for Model-1. Since Model-1 with no perturbation basically generates the same labeled data with the train-set, F_1-score of 94.85 which is similar with 94.77 of the baseline that only used the original train-set was obtained. By looking at the rows of additive perturbation and multiplicative perturbation, we obtained improvement with smaller perturbation values. By increasing p_m to 0.8 for multiplicative perturbation, we saw a significant drop in the F_1-score by the trained LSTM-based slot filling model. While the decoder LSTM has strong constraints to generate the reasonable labeled data, large perturbation to the initial hidden state and the memory cell resulted in generating the useless labeled data.

Table 2 lists the F_1-scores by the LSTM-based slot filling model trained using the generated labeled data by Model-1, Model-2, Model-3, and Model-4. By looking at the rows of Model-2, we found improvement by using the perturbation comparing with the row without perturbation. For Model-3, the best F_1-score was worse than Model-2 while improvement by using perturbations was confirmed. Model-4 performed worse than the baseline. Dependencies between words and labels introduced for Model-3 and Model-4 were not effective in this task.

Model	Perturbation	F_1-score
Baseline	-	94.77
Model-1	-	94.85
	Additive	95.08
	Multiplicative	**95.31**
Model-2	-	94.75
	Additive	**95.32**
	Multiplicative	94.94
Model-3	-	94.81
	Additive	**95.16**
	Multiplicative	95.03
Model-4	-	94.68
	Additive	**94.75**
	Multiplicative	94.68

Table 2: *F_1-score with data generation by Model-1, Model-2, Mode-3, and Model-4. "Baseline" did not use any additional training data. For each model, 3 rows correspond to no perturbation, additive perturbation, and multiplicative perturbation. We tried various perturbation values p_a and p_m and reported results of model that achieved best for heldout-set.*

4.3. Analysis on generated labeled data

The best F_1-score of 95.32 was obtained with Model-2 by setting $p_a = 0.2$ and $p_m = 0.0$. We investigated the generated labeled data by this configuration. We found 11 new pairs of word and label that are not included in the train-set. 3 pairs were appropriate and the others were not correct. While the number of the appropriate pairs was limited, we can select the appropriate pairs with other methods such as using generated frequency or examining the order of labels in the generated data [16][1]. Combining with these method is our future work.

We also investigated the number of new pairs obtained by Model-3 and Model-4 with the same configuration of $p_a = 0.2$ and $p_m = 0.0$. We found 4 new pairs from Model-3, one of which was correct, and no new pairs from Model-4. Training the decoder LSTMs with dependencies between words and labels becomes similar to training the decoder LSTM with using a pair of word and label as a modeling unit like Model-1, and thus limits the capability of generating new pairs.

5. Conclusions

We proposed a method to generate labeled data from a small amount of manually labeled data with using the encoder-decoder models. We confirmed that by adding perturbations to the encoded vector, variety of labeled data can be generated, which resulted in the improvement of F_1-score of the slot filling model over the competitive LSTM-based slot filling model.

We also confirmed that new pairs of word and label can be obtained by the encoder-decoder model that has separate output sequences for word and label, while the ratio of the correct pair was not so high. We would like to pursue a method to purify the generated pairs.

As for perturbations, we simply used additive and multiplicative perturbation sampled from normal distributions. We would like to explore more sophisticated perturbation.

We focused on labeled data generation in this paper and using the generated data with other sophisticated modeling of slot filling [34, 35] is our future work.

[1] If "B-ToCity" label comes after "I-ToCity" label, this order of labels is not appropriate.

6. References

[1] R. De Mori, F. Bechet, D. Hakkani-Tur, M. McTear, G. Riccardi, and G. Tur, "Spoken language understanding," *IEEE Signal Processing Magazine*, vol. 3, no. 25, pp. 50–58, 2008.

[2] P. Price, "Evaluation of spoken language systems: The ATIS domain," in *Proc. DARPA Speech and Natural Language Workshop*, 1990, pp. 91–95.

[3] S. Seneff, "TINA: A natural language system for spoken language applications," *Computational linguistics*, vol. 18, no. 1, pp. 61–86, 1992.

[4] S. Miller, R. Bobrow, R. Ingria, and R. Schwartz, "Hidden understanding models of natural language," in *Proc. ACL*, 1994, pp. 25–32.

[5] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. ICML*, 2001, pp. 282–289.

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[7] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[8] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[9] M. I. Jordan, "Serial order: A parallel distributed processing approach," *Advances in psychology*, vol. 121, pp. 471–495, 1997.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu, "Recurrent neural networks for language understanding," in *Proc. INTERSPEECH*, 2013, pp. 2524–2528.

[12] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig, and Y. Shi, "Spoken language understanding using long short-term memory neural networks," in *Proc. SLT*, 2014, pp. 189–194.

[13] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 3, pp. 530–539, 2015.

[14] A. Deoras, T. Mikolov, S. Kombrink, M. Karafiát, and S. Khudanpur, "Variational approximation of long-span language models for LVCSR," in *Proc. ICASSP*, 2011, pp. 5532–5535.

[15] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[16] Y.-C. Tam, Y. Shi, H. Chen, and M.-Y. Hwang, "Rnn-based labeled data generation for spoken language understanding," in *Proc. INTERSPEECH*, 2015, pp. 125–129.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. NIPS*, 2014, pp. 3104–3112.

[18] K. Yao and G. Zweig, "Sequence-to-sequence neural net models for grapheme-to-phoneme conversion," *Proc. INTERSPEECH*, pp. 3330–3334, 2015.

[19] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proc. NIPS*, 2015, pp. 577–585.

[20] L. Lu, X. Zhang, K. Cho, and S. Renals, "A study of the recurrent neural network encoder-decoder for large vocabulary speech recognition," in *Proc. INTERSPEECH*, 2015, pp. 3249–3253.

[21] J. Li, M.-T. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *arXiv preprint arXiv:1506.01057*, 2015.

[22] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. ICML*, 2015, pp. 2342–2350.

[23] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models." in *Proc. EMNLP*, 2013, pp. 1700–1709.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[25] D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, C. Pao, A. Rudnicky, and E. Shriberg, "Expanding the scope of the ATIS task: The ATIS-3 corpus," in *Proc. HLT*, 1994, pp. 43–48.

[26] Y.-Y. Wang, A. Acero, M. Mahajan, and J. Lee, "Combining statistical and knowledge-based spoken language understanding in conditional models," in *Proc. COLING-ACL*, 2006, pp. 882–889.

[27] G. Tur, D. Hakkani-Tur, and L. Heck, "What is left to be understood in ATIS?" in *Proc. SLT*, 2010, pp. 19–24.

[28] C. J. van Rijsbergen, *Information Retrieval*. Butterworth, 1979.

[29] G. Kurata, B. Xiang, B. Zhou, and M. Yu, "Leveraging sentence-level information with encoder LSTM for natural language understanding," *arXiv preprint arXiv:1601.01530*, 2016.

[30] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, 2010, pp. 249–256.

[31] D. Kingma and J. Ba, "ADAM: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[33] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[34] B. Peng and K. Yao, "Recurrent neural networks with external memory for language understanding," *arXiv preprint arXiv:1506.00195*, 2015.

[35] N. T. Vu, P. Gupta, H. Adel, and H. Schütze, "Bi-directional recurrent neural network with ranking loss for spoken language understanding," in *Proc. ICASSP*, 2016, pp. 6060–6064.