SPOKEN LANGUAGE UNDERSTANDING USING LONG SHORT-TERM MEMORY NEURAL NETWORKS

Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi

Microsoft

ABSTRACT

Neural network based approaches have recently produced record-setting performances in natural language understanding tasks such as word labeling. In the word labeling task, a tagger is used to assign a label to each word in an input sequence. Specifically, simple recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have shown to significantly outperform the previous state-of-the-art – conditional random fields (CRFs). This paper investigates using long short-term memory (LSTM) neural networks, which contain input, output and forgetting gates and are more advanced than simple RNN, for the word labeling task. To explicitly model output-label dependence, we propose a regression model on top of the LSTM un-normalized scores. We also propose to apply deep LSTM to the task. We investigated the relative importance of each gate in the LSTM by setting other gates to a constant and only learning particular gates. Experiments on the ATIS dataset validated the effectiveness of the proposed models.

Index Terms— Recurrent neural networks, long short-term memory, language understanding

1. INTRODUCTION

In recent years, neural network based approaches have demonstrated outstanding performance in a variety of natural language processing tasks [1–8]. In particular, recurrent neural networks (RNNs) [9, 10] have attracted much attention because of their superior performance in language modeling [1] and understanding [5, 6] tasks. In common with feed-forward neural networks [11–14], an RNN maintains a representation for each word as a high-dimensional real-valued vector. Critically, similar words tend to be close with each other in this continuous vector space [15]. Thus, adjusting the model parameters to increase the objective function for a particular training example tends to improve performance for similar words in the similar contexts.

In this paper we focus on spoken language understanding (SLU), in particular, word labeling with semantic information [16–22]. For example, for the sentence "I want to fly from Seattle to Paris," the goal is to label the word "Seattle" and 'Paris" as the departure and arrival cities of a trip, respectively. The previous state-of-the-art model that is widely used for this task [20, 23, 24] is the conditional random field (CRF) [25], which produces a single, globally most likely label sequence for each sentence. Another popular discriminative model for this task is the support vector machine [26, 27].

Recently, RNNs [5, 6] and convoluational neural networks (CNNs) [7] have been applied to SLU. The Elman [9] architecture adopted in [5] uses past hidden activities, together with the observations, as the input to the same hidden layer, which in turn applies a nonlinear transformation to convert the inputs to activities. The Jordan [10] architecture exploited in [6] uses past predictions at the output layer instead of the past hidden activities as additional inputs to the hidden layer. In [7] CNNs are used similar to that in [28] to extract features through convolving and pooling operations. CNNs achieved comparable performances to RNNs on SLU tasks.

The RNNs in [5, 6] are trained to optimize the frame cross-entropy criterion. More recently, sequence discriminative training is used to train RNNs [29]. Similar work is conducted in [7] for CNNs. The main motivation of using sequence discriminative training is to overcome the label biasness [25] problem that is addressed by CRFs. It incorporates dependence between output tags and adds a knowledge source for performance improvements.

In this paper we apply long short-term memory (LSTM) neural networks to the SLU tasks. LSTM [30, 31] has some advanced properties compared to the simple RNN. It consists of a layer of inputs connected to a set of hidden memory cells, a connected set of recurrent connections amongst the hidden memory cells, and a set of output nodes. Importantly, input to and output of the memory cells are modulated in a context-sensitive way. To avoid the gradient diminishing and exploding problem, the memory cells are linearly activated and propagated between different time steps.

We further extend the basic LSTM architecture to include a regression model that explicitly models the dependencies between semantic labels. To avoid label biasness problem, this model uses un-normalized scores before softmax. In another extension, we apply deep LSTM, which consists of multiple layers of LSTMs, to the task. To assess which gates in the LSTM models are important for SLU tasks we simplify the LSTM models by keeping only particular gates and compare the performance of the simplified models with that of the

complete LSTM.

2. RECURRENT NEURAL NETWORKS

2.1. LSTM

RNNs incorporate discrete-time dynamics. The long short-term memory (LSTM) [30, 32] RNN has been shown to perform better at finding and exploiting long range dependencies in the data than the simple RNN [9, 10]. One difference from simple RNN is that the LSTM uses a *memory cell* with linear activation function to store information. Note that the gradient-based error propagation scales errors by the derivative of the unit's activation function times the weight that the forward signal weight through. Using linear activation functions allows the LSTM to preserve the value of errors because its derivative with regard to the error is one. This to some extent avoids the error exploding and diminishing problems as the linear memory cells maintains unscaled activation and error derivatives across arbitrary time lags.

We implemented the version of LSTM [33] described by the following composition function:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \quad (2)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \quad (3)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \quad (4)$$

$$h_t = o_t \odot tanh(c_t), \quad (5)$$

where σ is the logistic sigmoid function. i, f, o and c are respectively the *input gate, forget gate, output gate*, and *memory cell activation* vectors, all of which have the same size as the hidden vector h. $\odot$ denotes the element-wise product of the vectors. The weight matrices from the cell to gate vectors (e.g., W_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector. However, the weight matrices from input, hidden, and outputs are not diagonal.

2.2. Output regression

For language understanding tasks, it is beneficial to incorporate dependency of output labels [29, 34]. For example, the maximum entropy feature in the RNNLM [34] uses word hashing of the past observations as additional input to the output layer. However, for SLU, there is no such word hashing that can be applied because the output layers generate predictions of the semantic tags for the input and these semantic tags are not observed. Yet, it is still beneficial to incorporate the past predictions as additional input to the output layer. The following model exploits LSTM outputs and performs regression on the predictions. Specifically, we adopted the moving-average model plotted in Figure 1. For clarity, it is unrolled and includes two time instances $t-1$ and t.

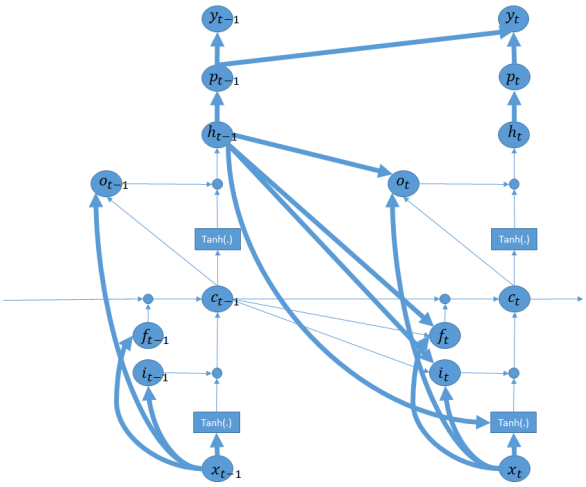

Fig. 1. Graphical structure of the LSTM and its moving average extension unrolled at time $t-1$ and t. x_ts represent inputs; p_ts are the excitations before softmax; y_ts are the activities after softmax; c_ts are the memory cell activities; f_t, i_t, and o_t are forget, input, and output gate respectively; and h_t is the output from LSTM. Small dot node means element wise product. Bold arrows connect nodes with full matrices. Thin arrows connect nodes with diagonal matrices.

Importantly, the dependence between output labels is modeled using the values before the softmax operation, which are not locally normalized, to avoid label biasness problem [25, 29]. This regression model can be described mathematically as

$$p_t = W_{hp}h_t, \quad (6)$$

$$q_t = \sum_{i=0}^{M} W_{pi}p_{t-i} + b_q, \quad (7)$$

$$y_t = softmax(q_t), \quad (8)$$

where the matrix W_{hp} transforms h_t into a vector that has the same dimension as the output y_t. Matrices W_{pi} are the regression matrices on the predictions p_{t-i} for $i = \{0, \cdots, M\}$ and M is the order of the moving average. W_{p0} is initialized to a diagonal matrix with diagonal components set to 1. For other W_{pi}s they are initialized to zero matrices.

We may also apply auto regression on the predictions. In this case, Eq. (7) is replaced with

$$p_t = \sum_{i=1}^{M} W_{pi}p_{t-i} + W_{hp}h_t, \quad (9)$$

$$y_t = softmax(p_t + b_p), \quad (10)$$

where b_p is a bias vector.

Our initial experiments show that this extension reduces training entropy but doesn't result in improved F1 scores. In addition, the auto-regression model is not as easy to train as the moving average model, we therefore only consider the moving average model in this paper.

2.3. Deep LSTM

The deep LSTM is created by stacking multiple LSTMs on top of each other. The output sequence of the lower LSTM forms the input sequence for the upper LSTM. Specifically, input x_t of the upper LSTM takes h_t from the lower LSTM. A matrix is applied on the h_t to transform it to x_t; the matrix can be constructed so that the lower and upper LSTMs have different hidden layer dimensions. This paper evaluates deep LSTMs with two layers.

2.4. LSTM simplification

The process in LSTM includes three gating functions. Each memory cell c_t has its net input modulated by the activity of an input gate, and has its output modulated by the activity of an output gate. These input and output gates provide a context-sensitive way to update the contents of a memory cell. The forget gate modulates amount of activation of memory cell kept from the previous time step, providing a method to quickly erase the contents of memory cells [35].

It is interesting to know which gating functions are important for SLU tasks. To answer this question, we simplify LSTM networks by activating only particular gates.

The simplest modification would ignore all gating functions. In this case, the memory cells accumulate a history of inputs without discarding past memories. Inputs to and outputs of the memory cells are not modulated. More advanced networks learn one of the gates and keep other gates fixed to one.

In the case of learning forget and input gates, the simplified model can be described as

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \quad (11)$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \quad (12)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \quad (13)$$

$$h_t = tanh(c_t). \quad (14)$$

The bias of the gates, e.g., b_f, are tuned to have a large value initially to memorize past activations. If a large negative bias value is used instead, the memory cell will forget its past activities.

2.5. Implementation details

We implemented the LSTM architectures using the computational network toolkit (CNTK) [36]. To support arbitrary recurrent neural networks, CNTK introduces a specific computation node that does delay operation. In this node, the forward computation does time-shift operation on input x_t as $y_t = x_{t-n}$, which is the past activity of its input at time $t - n$. The errors from its output are propagated backward as $\delta x_{t-n} += \delta y_t$. This delay node enables constructing dynamic networks with long context. CNTK includes other generic computation nodes such as times, element times, plus, tanh, and sigmoid. Each node implements its forward computation and error back-propagation.

To connect these nodes into a network, CNTK first runs an algorithm that detects strongly-connected-components [37] and represents these strongly connected components with their unique numbers. Since every directed graph is a directed acyclic graph (DAG) of its strongly connected components, we can use depth first search to arrange these components, together with other computation nodes, into a tree. Forward computation of the constructed tree followed the topological order of this DAG. If a node in a strongly-connected-component is reached and is not yet computed, all the nodes in this strongly-connected-component are evaluated time-synchronously.

We use truncated back-propagation-through-time (BPTT) to update the model parameters [38]. The depth of BPTT is equivalent to the minibatch size. Therefore, a sentence is broken into several minibatches. For recurrent neural networks including simple RNNs and LSTMs, the activities of delay node is set to a default value only at the beginning of a sentence; its activities are carried over to the following minibatches. We also compute multiple sequences with the same length in parallel. This allows efficient computation in recurrent neural networks because multiple sentences can be processed simultaneously using matrix-matrix operations. In practice, using same-length sentences in batches reduces training time without sacrificing performances. We implemented both momentum- and AdaGrad-based [39] gradient update techniques. CNTK can run on both GPU and CPU. For SLU tasks, which are small, we run experiments on CPUs.

3. EXPERIMENTS

3.1. Dataset

We evaluated the standard LSTM and our extensions on the ATIS database [16, 23, 40]. We also include results of simple RNN and CNN for comparison. This dataset focuses on the air travel domain, and consists of audio recordings of people making travel reservations, and semantic interpretations of the sentences. In this database, the words in each sentence are labeled with their value with respect to certain seman-

Table 1. F1 scores (in %) on ATIS with different modeling techniques. LSTM-ma(3) denotes using moving average of order 3. Deep denotes deep LSTM.

CRF	RNN	CNN	LSTM	LSTM-ma(3)	Deep
92.94	94.11	94.35	94.85	94.92	95.08

tic frames. The training data consists of 4978 sentences and 56590 words, selected from the ATIS-2 and ATIS-3 corpora. Test data consists of 893 sentences and 9198 words, selected from the ATIS-3 Nov93 and Nov94 datasets. The number of distinct slot labels is 127, including the common null label; there are a total of 25509 non-null slot occurrences in the training and testing data respectively. Based on the number of words in the dataset and assuming independent errors, changes of approximately 0.6% in F1 measure are significant at the 95% level.

ATIS dataset also has the named-entity feature, which contains strong information of semantic tags. For example, a sentence of *from Boston at* would have named entity tag of *from B-city_name at* and semantic tag of *O B-fromloc.city_name O*. Clearly, named entity provides strong cue for semantic tags. We believe results with lexicon features only would make model comparison more meaningful. Therefore, in the following, we only report experiments with lexicon feature[1].

3.2. Results with LSTM, output regression, and deep LSTM

The input x_t in the LSTM consists of the current input word and the next two words in a context window of size 3. Its hidden-layer dimension is 300 and minibatch size is 30. The LSTM extension described in Sec 2.2 in addition uses moving average regression on three predictions p_t, p_{t-1} and p_{t-2}. The result is denoted as LSTM-ma(3) in Table 1. Regression matrices e.g. W_{pi}s are full.

We described in Sec. 2.3 the deep LSTM architecture. In this experiment, we use 200 hidden layer dimension for the lower LSTM and 300 hidden layer dimension for the upper LSTM. Learning rate was 0.1 per sample. Minibatch size was 10.

Table 1 also lists performances in F1 score achieved by RNN [5] and CNN [7]. Performance by CRF [25] is 92.94% on this task. Their results are optimal in the respective systems. RNN in [5] is able to improve F1 score from 92.94% by CRF to 94.11%. CNN [7] also significantly improves F1 score over CRF. LSTMs further improve F1 scores to 94.92% and 94.85% with and without using the moving average, respectively. Deep LSTM achieves the highest F1 score of 95.08%. These improvements are significant compared against simple RNNs and CNNs.

We observed that the moving average extension has small but consistent improvements over LSTMs. For example, if the hidden layer dimension is reduced to 100, LSTM achieves a 94.62% F1 score while LSTM-ma(3) obtains a 94.86% F1 score. The similar observations were made on internal production datasets.

[1]If using lexicon and named entity feature, LSTM obtained 96.60% F1 score and simple RNN obtained 96.57% F1 score [5]. As mentioned in the dataset description in Sec. 3.1, we believe comparing models trained only on lexicon feature would be more meaningful. Therefore, we didn't do further experiments using named entity features.

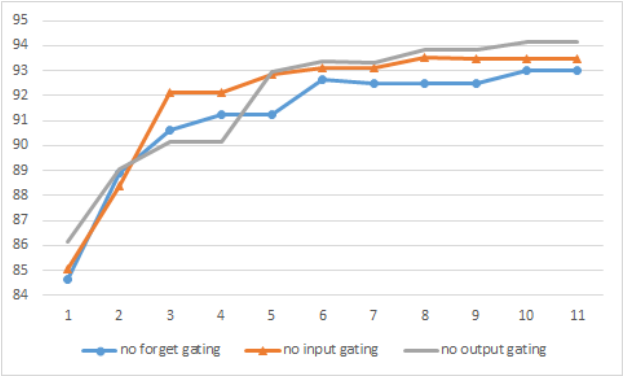

Fig. 2. F1 score versus training iterations by simplified LSTM with forget gate fixed to 1.0 in round-marked curve, input gate fixed to 1.0 in triangle-marked curve, and output gate fixed to 1.0 in real curve.

3.3. Results with simplifications

As described in Sec. 2.4, LSTM models can be simplified by fixing one of the gates to 1.0, and learning the other two gates. We have shown in Eqs. (11-14) the case in which the output gate is set to 1.0 while the input and forget gates are learned. Figure 2 plots F1 scores with this simplified LSTM as a function of training iterations. The hidden layer dimension of this network is 100. The minibatch size is set to 8, and the learning rate is 0.01. The result is shown in real curve. For comparison, we also plot the simplified network with forget gate fixed to 1.0 while learning other two gates. This result is represented in round-marked curve. Results with the input gate fixed to 1.0 is plotted in triangle-marked curve.

All of the simplified networks are able to converge in ten iterations. It is clear that the F1 score of 93.02% without learning forget gate is lower than that obtained with the other two configurations. The best F1 score of 94.14% is achieved with both forget and input gates learned, represented in Eqs. (11-14). This result is close to that of using all gates in the standard LSTM, which achieves 94.25% with the same hidden layer dimension and minibatch size. Therefore, if two gates are to be included in the simplified LSTM, one of the

gates should be the forget gate.

We further applied moving average regression extension in Sec. 2.2 on the outputs from the simplified LSTMs. With moving average order of 3, F1 score is improved to 94.28% from 94.14%. This score is on par with 94.25% by the standard LSTM.

The importance of gates is different when only one gate is to be learned, i.e., the other two gates are fixed to 1.0. Under this condition, the best F1 score is 90.16% with output gate learned. Learning other gates has lower F1 scores. For instance, learning only forget gate obtains a F1 score of 73.64%. We plan to conduct further analysis to understand dynamics and importance of gating in the LSTM networks.

4. CONCLUSIONS AND DISCUSSIONS

We have presented an application of LSTMs to spoken language understanding. The LSTMs achieved state-of-the-art results on the ATIS database. We further made extensions of LSTM by performing regressions on the output of LSTMs and building stacks of LSTMs. We observed that these extensions slightly yet consistently improve performances on this dataset. We investigated the importance of gates in LSTMs and observed that the forget gate is essential in the LSTM network if two or more gates are learned.

There are many possible extensions of the work. For instance, we may extend the LSTM gating to work directly on the weights instead of activation similar to [41]. We may also investigate using other architectures of neural networks [8, 42–44] and employ sequence discriminative training to the LSTM for SLU [29].

We plan to conduct error analysis on ATIS and other datasets to understand and validate this modeling technique and its extensions.

5. REFERENCES

[1] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH*, 2010, pp. 1045–1048.

[2] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network based language model," in *ICASSP*, 2011, pp. 5528–5531.

[3] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Cernocky, "Strategies for training large scale neural network language models," in *ASRU*, 2011.

[4] E. Arisoy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Deep neural network language models," in *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, 2012, pp. 20–28.

[5] K. Yao, G. Zweig, M. Hwang, Y. Shi, and Dong Yu, "Recurrent neural networks for language understanding," in *INTERSPEECH*, 2013.

[6] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for language understanding," in *INTERSPEECH*, 2013.

[7] P. Xu and R. Sarikaya, "Convolutional neural network based triangular CRF for joint detection and slot filling," in *ASRU*, 2013.

[8] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul, "Fast and robust neural network joint models for statistical machine translation," in *ACL*, 2014.

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

[10] M. Jordan, "Serial order: a parallel distributed processing approach," *Advances in Psychology*, vol. 121, pp. 471–495, 1997.

[11] Y. Bengio, R. Ducharme, Vincent, P., and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Reseach*, vol. 3, no. 6, 2003.

[12] H. Schwenk, "Continuous space language models," *Computer Speech and Language*, vol. 21, no. 3, pp. 492 – 518, 2007.

[13] H.-S. Le, I. Oparin, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Structured output layer neural network language model," in *ICASSP*, 2011, pp. 5524–5527.

[14] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proceedings of the international workshop on artificial intelligence and statistics*, 2005, pp. 246–252.

[15] T. Mikolov, W.T. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *NAACL-HLT*, 2013.

[16] C. Hemphill, J. Godfrey, and G. Doddington, "The ATIS spoken language systems pilot corpus," in *Proceedings of the DARPA speech and natural language workshop*, 1990, pp. 96–101.

[17] P. Price, "Evaluation of spoken language systems: The ATIS domain," in *Proceedings of the Third DARPA Speech and Natural Language Workshop*. Morgan Kaufmann, 1990, pp. 91–95.

[18] W. Ward et al., "The CMU air travel information service: Understanding spontaneous speech," in *Proceedings of the DARPA Speech and Natural Language Workshop*, 1990, pp. 127–129.

[19] Y. He and S. Young, "A data-driven spoken language understanding system," in *ASRU*, 2003, pp. 583–588.

[20] C. Raymond and G. Riccardi, "Generative and discriminative algorithms for spoken language understanding," *INTERSPEECH*, pp. 1605–1608, 2007.

[21] R. De Mori, "Spoken language understanding: A survey," in *ASRU*, 2007, pp. 365–376.

[22] F. Béchet, "Processing spontaneous speech in deployed spoken language understanding systems: a survey," *SLT*, vol. 1, 2008.

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

[24] A. Moschitti, G. Riccardi, and C. Raymond, "Spoken language understanding with kernels for syntactic/semantic structures," in *ASRU*, 2007, pp. 183–188.

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

[26] M. Henderson, M. Gasic, B. Thomson, P. Tsiakoulis, K. Yu, and S. Young, "Discriminative spoken language understanding using word confusion networks," in *IEEE SLT Workshop*, 2012.

[27] R. Kuhn and R. De Mori, "The application of semantic classification trees to natural language understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 449–460, 1995.

[28] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 24932537, Aug 2011.

[29] K. Yao, B. Peng, G. Zweig, D. Yu, X. Li, and F. Gao, "Recurrent conditional random fields for language understanding," in *ICASSP*, 2014.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1438, 1997.

[31] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*, 2013.

[32] F. Gers and J. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Trans. on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.

[33] A. Graves, "Generating sequences with recurrent neural networks," Tech. Rep. arxiv.org/pdf/1308.0850v2.pdf, 2013.

[34] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," in *SLT*, 2013.

[35] F. Gers and F. Cummins, "Learning to forget: continual prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, 2000.

[36] D. Yu, A. Eversole, M. Seltzer, K. Yao, B. Guenter, O. Kuchaiev, F. Seide, H. Wang, J. Droppo, Z. Huang, Y. Zhang, G. Zweig, C. Rossbach, and J. Currey, "An introduction to computational networks and the computational network toolkit," Tech. Rep. MSR, Microsoft Research, 2014, http://codebox/cntk.

[37] S. Dasgupta, C. Papadimitriou, and U. Vazirani, *Algorithms*, McGraw Hill, 2007.

[38] R. Williams and J. Peng, "An efficient gradient-based algorithm for online training of recurrent network trajectories," *Neural Computation*, vol. 2, pp. 490–501, 1990.

[39] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

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

[41] D. D. Monner and J. A. Reggia, "A generalized LSTM-like training algorithm for second-order recurrent neural networks," *Neural Networks*, vol. 25, pp. 70–83, 2012.

[42] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *NIPS*, 2014.

[43] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork RNN," in *International Conf. on Machine Learning*, 2014.

[44] T. Breuel, A. Ul-Hasan, M. A. Azawi, and F. Shafait, "High-performance OCR for printed Englist and Fraktur using LSTM networks," in *Intertional conference on document analysis and recognition*, 2013, pp. 683–687.