2015 13th International Conference on Document Analysis and Recognition (ICDAR)

A Context-Sensitive-Chunk BPTT Approach to Training Deep LSTM/BLSTM Recurrent Neural Networks for Offline Handwriting Recognition

Kai Chen*†, Zhi-Jie Yan† and Qiang Huo†
*Department of Electronics Science and Technology, University of Science and Technology of China, Hefei, China
†Microsoft Research Asia, Beijing, China
Email: {v-kachen, zhijiey, qianghuo}@microsoft.com

***Abstract*—We propose a context-sensitive-chunk based back-propagation through time (BPTT) approach to training deep (bidirectional) long short-term memory ((B)LSTM) recurrent neural networks (RNN) that splits each training sequence into chunks with appended contextual observations for character modeling of offline handwriting recognition. Using short context-sensitive chunks in both training and recognition brings following benefits: (1) the learned (B)LSTM will model mainly local character image dependency and the effect of long-range language model information reflected in training data is reduced; (2) mini-batch based training on GPU can be made more efficient; (3) low-latency BLSTM-based handwriting recognition is made possible by incurring only a delay of a short chunk rather than a whole sentence. Our approach is evaluated on IAM offline handwriting recognition benchmark task and performs better than the previous state-of-the-art BPTT-based approaches.**

I. Introduction

Recurrent neural networks (RNNs) (e.g., [1], [2]) are a kind of artificial neural networks that contain cyclic connections, which can model contextual information of a sequence dynamically, and their bidirectional version [3] provides a framework to utilize future and past information simultaneously at each time step. To address the difficulty of training RNNs, a special type of RNNs, namely long short-term memory (LSTM) networks, are proposed, which contain special units called memory blocks in recurrent hidden layers to ensure constant error flow through time (e.g., [4]–[6]), and their bidirectional version, namely BLSTM, has been successfully applied to automatic speech recognition (ASR) (e.g., [7], [8]), handwriting recognition (HWR) (e.g., [9]), and optical character recognition (OCR) for printed text (e.g., [10]). Inspired by the recent success of deep neural networks (DNNs) in ASR (e.g., [11], [12] and the references therein), deep (B)LSTMs, which mean stacking multiple (B)LSTM layers on top of each other just as feed-forward layers are stacked in DNNs, have also been applied to and become the state-of-the-art solutions to both ASR (e.g., [13]–[17]) and offline HWR (e.g., [18]–[21]).

As a type of RNN, deep LSTM can be trained via various gradient-based algorithms designed for general RNN, for example, real-time recurrent learning (RTRL), different variants of back-propagation through time (BPTT) algorithms including BPTT(∞), BPTT(h), BPTT(h, h'), *epochwise BPTT*, and a hybrid algorithm called FP/BPTT(h) (see [2] for details). Given a fully connected RNN with n units, RTRL's space complexity is $O(n^3)$ and FP/BPTT(h)'s is $O(n^3 + nh)$, so when the network has large number of units, these algorithms become impractical. Among BPTT-type methods, BPTT(∞) injects one error signal each time step and back-propagates it to the beginning of the sequence. BPTT(h) also injects one error every time but back-propagates it for at most h time steps. Per time step updating strategy makes these two methods very time consuming so they are of more theoretical than practical interest. As an improved version of BPTT(h), BPTT(h, h') [1] updates the network parameters every h' time steps through injecting h' error signals and back-propagating for at most h time steps during each update. When $h = h' = \texttt{sequence length}$, this approach is equivalent to epochwise BPTT (e.g., [2]), which needs to store network states of all time steps, while when $h = h' < \texttt{sequence length}$, the approach is called *truncated BPTT*, which is used in [16]. Like in training feed-forward DNNs, mini-batch based training can also be used to accelerate training, where the mini-batch can be defined over sequences (e.g., [22]) or truncated chunks (e.g., [16], [18]).

Although many algorithms have been proposed for RNN training, most of them cannot be applied to train bidirectional RNN due to the whole sequence dependence at each time step. Luckily epochwise BPTT can handle this case and leads to several successful applications (e.g., [7]–[10], [13]–[15], [20], [21]). Nowadays, GPUs are widely used in deep learning by leveraging massive parallel computations via mini-batch based training. When epochwise BPTT is used, GPU's limited memory restricts the number of sequences used in a mini-batch, especially for very long sequences, which leads to poor acceleration. In [18], a chunking and batching approach was proposed to solve this problem, but when the chunk size is not large enough, it performs worse than the epochwise BPTT due to the loss of contextual information of training chunks. We call this approach *chunk BPTT* thereinafter.

In the previous work of using (B)LSTM for ASR and OCR/HWR, the learned (B)LSTM will include both local acoustic/character model information and global language model information from the training data. If the training text

This work was done when Kai Chen was an intern in Speech Group, Microsoft Research Asia, Beijing, China.

is highly mismatched with recognition tasks, such a (B)LSTM may not work well. However, in a so-called hybrid approach [23] of using (B)LSTM and hidden Markov model (HMM) for ASR and OCR/HWR, it is possible to force the (B)LSTM to model mainly the local acoustic/character information which can be trained from an appropriate set of speech/image training data, while the language model can be learned from another large set of text data. In this paper, we propose a context-sensitive-chunk based BPTT (CSC-BPTT) approach to training deep (B)LSTM. We split each training sequence into short chunks with appended contextual observations. Using short context-sensitive chunks in both training and recognition brings following benefits: (1) the learned (B)LSTM will model mainly local observation dependency and the effect of long-range language model information reflected in training data is reduced; (2) mini-batch based training on GPU can be made more efficient; (3) short-delay BLSTM-based recognition is made possible by incurring only a delay of a short chunk rather than a whole sentence. As a first step, we evaluate our approach on IAM offline handwriting recognition benchmark task [24] and compare its performance with several state-of-the-art BPTT-based approaches [16], [18], [22].

The remainder of this paper is organized as follows. In section II, we give a brief introduction of prior arts and describe our proposed approach in detail. In section III, we present experimental results. In section IV, we analyze the observations and finally we conclude the paper in Section V.

II. Our Approach

In this section, we first review several previous deep (B)LSTM training methods. Then we introduce our proposed CSC-BPTT method and present some implementation details.

A. Prior Arts

1) Epochwise BPTT (e.g., [2], [7], [22]): Given a sequence, this method must accumulate the history of activations in the network over the entire sequence, along with the history of errors, then back-propagation is carried out to calculate gradients. After that, weights are updated accordingly. In order to accelerate training by GPU, mini-batch technique can be used as in DNN training, where several frames are processed per update, but the mini-batch here has to be defined over sequences due to the interdependence of frames in the sequence [22]. Therefore for very long sequences and large networks, memory size of GPU restricts the number of parallel sequences in a mini-batch so the acceleration is quite limited.

2) Truncated BPTT (e.g., [1], [16]): This method is well studied on traditional RNNs, and can also be directly applied to LSTMs. Activations are forward-propagated for a fixed length of time steps T_{trunc}, then errors are accumulated over this chunk and back-propagated. After weights are updated, if current sequence hasn't finished, preserve the network states and process next chunk; otherwise, reset the network states and process first chunk of a new sequence. Mini-batch technique can be applied and defined over truncated chunks. Note that the last chunk of each sequence can be shorter than T_{trunc} but can be padded to the full length with empty frames, though no gradient is generated for these padded frames. With this approach, we can break the barrier of GPU's memory limitation and process many chunks in parallel, but BLSTMs can't benefit from it for the bidirectional interdependence.

3) Chunk BPTT ([18]): This is a simple but effective method. As Fig. 1-(a) shows, each sequence is firstly split into (possibly overlapping) chunks of a particular length N_c and recombined into a large batch as shown on the right. For chunks shorter than N_c, empty frames are appended. Then forward this batch in parallel and update the network parameters, just as epochwise BPTT does. Chunks of a batch only come from a small set of sequences, therefore the variations caused by various irrelevant factors are small within the batch. With this approach, [18] achieves comparable results and 3x speedup compared with epochwise BPTT, but a relatively large N_c is needed to alleviate performance degradation.

B. CSC-BPTT for LSTMs

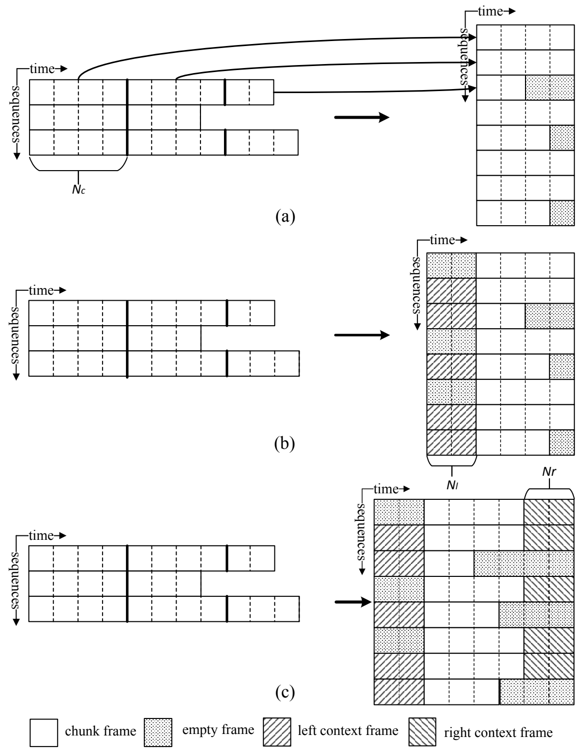

Figure 1. *Illustration of different chunk-based approaches. Each row on the left/right side corresponds to one sequence/chunk and each column corresponds to a batch of current time steps. **(a)** is proposed in [18], where the sequences are partitioned into chunks of a particular length, with empty frames appended to short chunks, and recombined into a large batch as shown on the right; **(b)** is our proposed method for LSTMs. Each sequence is firstly partitioned like (a). Then, left context from previous frames is appended. Finally, all chunks are padded to the same length with empty frames; **(c)** is our proposed method for BLSTMs. Given a particular length, the sequences are firstly split into chunks. Then, left and right contexts are appended. Finally, all chunks are padded to the same length with empty frames.*

Chunks in section II-A3 are treated as isolated sequences so that they can be processed in parallel. However, the interdependence between chunks is lost, which causes chunk

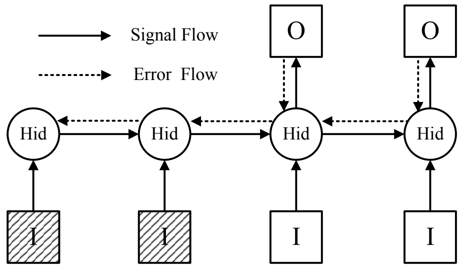

Figure 2. *Forward and backward procedure of CSC-BPTT to train LSTM.*

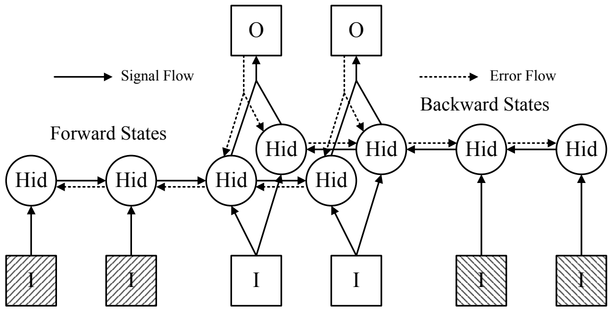

Figure 3. *Forward and backward procedure of CSC-BPTT to train BLSTM.*

BPTT with short chunks to perform worse than the epochwise BPTT. To compensate for the lost/incomplete past states, as is shown in Fig. 1-(b), we append N_l previous frames before each chunk, while for the first chunk of each sequence, N_l empty frames are appended. We call these chunks as context-sensitive chunks (CSCs). The appended N_l frames only act as context and give no output, as illustrated in Fig. 2.

C. CSC-BPTT for BLSTMs

Due to bidirectional interdependence, BLSTM trained with chunk BPTT suffers from lost/incomplete past and/or future states. As is shown in Fig. 1-(c), we append N_l previous frames as left context and N_r future frames as right context to each chunk, while for the first/last chunk of the sequence, left/right context is filled with empty frames. Like the case of LSTM, appended frames only work as context and give no output. Fig. 3 illustrates the corresponding forward and backward procedure. Naturally in our cases, mini-batch is defined over CSCs.

Let's denote a CSC of length N_c with N_l left context and N_r right context as "N_l-N_c+N_r" for simplicity. In LSTM case, N_r always equals to 0, while for chunk BPTT, a traditional chunk can also be represented in this way as "0-N_c+0". We denote chunk configuration of epochwise BPTT as "0-Full+0" because a chunk here is the full sequence. To keep consistent with [1], truncated BPTT with chunk size T_{trunc} is denoted as BPTT(T_{trunc}, T_{trunc}) in this paper.

D. Implementation Issues

Our HWR system is based on (B)LSTM-HMM framework as in a neural-network/HMM hybrid system [23]. Frame-level state targets are provided by a forced alignment given by a Gaussian mixture model HMM (GMM-HMM) system. The activation function of (B)LSTM's output layer is softmax, whose unit number is the total number of HMM states. In training, a frame-level cross-entropy objective function is minimized. Our GPU-based training tool is developed by starting from Currennt open source toolkit [22]. We modified Currennt to implement all the methods in Section II and conducted all experiments on a Nvidia Telsa K20xm GPU. During training, a constant learning rate is used with a validation set monitoring the training progress. When improvement on validation set hasn't been observed for 10 continuous sweeps, training will be stopped. Sequences in training set are shuffled randomly before a new sweep starts. For our proposed approach, CSCs of a mini-batch can also be selected from different parts of different sequences randomly, but no much difference is observed in our experiments compared with the batching strategy shown in Fig. 1. In recognition, the same sequence split strategy as in training is adopted. The state-dependent scores derived from the (B)LSTM are combined with HMM state transition probabilities and language model (LM) scores to determine the recognition result by using a WFST-based decoder modified from the open source Kaldi toolkit [25].

Table I
OVERVIEW OF IAM OFFLINE HANDWRITING DATABASE

	Training	Validation	Test
# of text lines	6,159	900	1,861
# of words	53,823	7,897	17,615
# of characters	227,731	33,769	65,920
# of frames	1,978,170	298,960	563,078

III. EXPERIMENTAL RESULTS

A. Experimental Setup

We compare our method with the prior arts on IAM offline handwriting database [24]. There are two validation sets in IAM and we only used one. In preprocessing, each horizontal handwriting sentence in this database is normalized to have a height of 60 pixels. Each sentence is first split into frames by a sliding window of 30 pixels wide with a frame shift of 3 pixels. Then each frame is smoothed by applying a horizontal cosine window. Because the sliding window for feature extraction has a size of 30×60 and we use raw pixel values as original features, the dimension of original feature vector is 1800. Then, Principal Component Analysis (PCA) is used to reduce the dimension of each feature vector to 50. Finally, these 50-dimensional feature vectors are normalized such that each dimension of feature has a zero sample mean and unit sample variance on training set. Basic information of the data set is summarized in Table I.

To conduct forced alignment, a context-independent GMM-HMM based HWR system is built first by maximum likelihood training [26]. For simplicity, each character is modeled by a 3-state left-to-right GMM-HMM with self-loop transitions. There are 78 character classes in our character set, including 52 case-sensitive English letters, 10 digits, 15 punctu-

Table II
EVALUATION RESULTS OF DEEP LSTM UNDER DIFFERENT CHUNK CONFIGURATIONS (%)

Config.	Validation Set			Test Set		
	FER	CER	WER	FER	CER	WER
0-Full+0	34.70	12.81	24.29	37.36	15.71	29.59
BPTT(32,32)	34.80	12.78	24.17	37.46	15.85	29.55
0-128+0	36.01	13.34	25.18	38.36	16.01	30.44
0-64+0	37.11	14.25	25.87	39.38	16.89	31.11
0-32+0	38.34	13.48	25.35	40.82	16.98	31.49
32-32+0	34.71	12.69	24.03	37.20	15.42	29.48
16-32+0	35.07	12.71	23.73	**37.09**	**15.16**	**29.03**

ation marks (! # " ′ & ( ) * − , / . ; : ?) and a "space" symbol. Consequently, the (B)LSTM output layer has $3 \times 78 = 234$ units. The memory block of (B)LSTMs has the same topology as that in [14]. Random noise subject to Gaussian distribution $\mathcal{N}(0, 0.6)$ is added during training to improve generalization. In recognition, a standard word trigram LM is used, which is built by using 500 out of 4,000 documents from a Linguistic Data Consortium (LDC) corpus (catalog number LDC2008T15) and has a vocabulary of 200k most frequently occured words in the training corpus. To construct context-sensitive chunks, different numbers of contextual frames are tried, including 4, 8, 16 and 32 frames. For the above feature extraction procedure, 1 character has about 8 frames of feature vectors on average.

In training (B)LSTMs, learning rate is carefully tuned for each experimental configuration, and the one leading to the best validation performance is selected to perform recognition on test set. Frame error rate (FER), character error rate (CER) and word error rate (WER) are used to evaluate different models. It is noted that our 200k-word lexicon leads to an out-of-vocabulary (OOV) word rate of about 8% and 10% on validation and testing sets, respectively.

B. Experimental Results on Deep LSTM

The deep LSTMs used here have 5 hidden layers with 256 memory cells in each layer, resulting in approximately 2.5 million parameters. These networks are randomly initialized and optimized with epochwise, truncated, chunk and CSC BPTT methods, respectively. For epochwise and truncated BPTT, 32 sequences are processed in parallel in each mini-batch, and $T_{trunc} = 32$, while for chunk and CSC BPTT, the networks are updated every 4k frames (including contextual frames). We treat the result of epochwise BPTT as the baseline. Table II summarizes the experimental results. Among all the four algorithms evaluated, CSC-BPTT performs best. Both "16-32+0" and "32-32+0" achieve better results than the baseline. Other methods work as expected: truncated BPTT is slightly better than epochwise BPTT but can't beat ours while the performance of chunk BPTT approaches to that of epochwise BPTT as the chunk size increases. The result also shows that only small chunk sizes are required for CSC-BPTT to achieve such promising results.

Table III
EVALUATION RESULTS OF DEEP BLSTM UNDER DIFFERENT CHUNK CONFIGURATIONS (%)

Config.	Validation Set			Test Set		
	FER	CER	WER	FER	CER	WER
0-Full+0	23.68	10.28	20.61	26.20	12.46	25.38
0-16+0	30.18	11.92	23.24	32.75	14.54	28.60
4-16+4	25.08	10.37	21.07	27.57	12.45	25.48
8-16+8	23.86	10.01	20.28	26.31	12.17	25.05
0-32+0	27.22	10.94	21.12	29.56	12.78	26.26
8-32+8	23.79	10.25	20.32	26.26	12.29	25.02
16-32+16	23.10	9.85	19.97	**25.34**	**11.71**	24.21
0-64+0	25.24	10.73	21.20	27.81	12.76	25.99
16-64+16	23.04	9.70	19.33	25.45	11.79	**24.11**
32-64+32	23.04	9.77	19.66	25.43	11.79	24.49
0-128+0	24.45	11.08	21.64	26.66	13.03	26.37

C. Experimental Results on Deep BLSTM

We use 5-hidden-layer deep BLSTMs to compare the proposed approach with other methods. Each hidden layer has 256 memory cells (128 for forward and 128 for backward states), which results in approximately 1.8 million parameters. According to section II, these randomly initialized networks are optimized by epochwise, chunk and CSC BPTT methods, respectively. 32 sequences are processed in parallel by epochwise BPTT while chunk and CSC BPTT methods update the networks every 32 chunks. Again epochwise BPTT is used as the baseline.

Table III summarizes the evaluation results under different chunk configurations. Similar to deep LSTMs, chunk BPTT approaches the performance of epochwise method as chunk size increases, but the gap always exists. All CSC-BPTT experiments except "4-16+4" achieve better performance than the baseline, while even the exception case achieves comparable performance with the baseline and performs better than the best chunk BPTT result under the configuration of "0-128+0". "16-32+16" gives the best FER and CER on test set, and the relative error reductions against the baseline are 3.28% and 6.02% respectively, while the best WER is brought by "16-64+16", and the relative error reduction is 5.0%. In both N_l-16+N_r and N_l-32+N_r experiments, longer context window leads to better performance, while for N_l-64+N_r experiments, this is not the case: "16-64+16" performs better than "32-64+32". In Table II, "16-32+0" also outperforms "32-32+0". To conclude, 16 frames seem a good choice to provide contextual information, which is equivalent to about 2 characters. It should be noted that "8-16+8" and "8-32+8" perform worse than "0-Full+0" in terms of FER while better in terms of CER and WER. Furthermore, large gap of FER between "4-16+4" and "0-Full+0" only leads to small gap in CER and WER. These observations show that our method makes BLSTMs model mainly the local dependency of character images, and the influence of long-range language model information learned from the training set is reduced successfully.

IV. Discussion

In our (B)LSTM-HMM based HWR systems, the (B)LSTM is responsible for character modeling. However, the interdependence of time steps in (B)LSTM makes epochwise and truncated BPTT training take advantage of LM information prematurely, which affects the learning of character model negatively. Chunk technique can weaken the influence of LM by breaking the interdependence between chunks, and make (B)LSTM focus on character modeling. In reality, the variability of a character image may be affected by the images of several characters before and after it, which has been empirically confirmed by our experimental results. However, epochwise and truncated BPTT just ignore this prior knowledge and try to learn the dependency automatically. Even with the capacity to bridge variant time intervals through input, output and forget gates, it is still a difficult mission because these gates do not open or close synchronously. Splitting a sequence into chunks with particular length is equivalent to reset those gates and cell states manually and periodically. For offline HWR, we found that "16-32+0" in LSTM, "16-32+16" and "16-64+16" in BLSTM perform better than epochwise training, which indicates that there is no need to model long-term image dependencies. Meanwhile, chunks without contextual frames give poorer performance even with larger size. These observations show that contextual information is important for chunk technique, and 16 frames seem enough, which correspond roughly to 2 characters according to our feature extraction method. CSC-BPTT reduces the number of frames needed for each chunk, which makes it easier to model and train deep (B)LSTM in parallel on GPU. The short-chunk requirement also makes deep BLSTM applicable to low-latency decoding with the delay of a short chunk. To the best of our knowledge, this is the first successful attempt to perform such a low-latency decoding by using BLSTM for sequence pattern recognition.

V. Conclusion

In this paper, we have proposed a CSC-BPTT approach to training deep (B)LSTM for sequence pattern recognition. Its effectiveness has been confirmed on an offline handwriting recognition task. This modeling and training approach can also be applied to other tasks such as ASR and OCR, and we will report those results elsewhere.

Acknowledgment

The authors would like to thank Lijuan Wang and Xiong Zhang for sharing the baseline systems trained by epochwise BPTT on IAM database.

References

[1] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation*, vol. 2, no. 4, pp.490-501, 1990.

[2] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent neural networks and their computational complexity," in Y. Chauvin and D. E. Rumelhart (Eds.), *Back-propagation: Theory, Architectures and Applications*, pp.433-486, 1995.

[3] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 9, no. 11, pp.2673-2681, 1997.

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

[5] F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to forget: continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp.2451-2471, 2000.

[6] F. A. Gers, N. N. Schraudolph and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, pp.115-143, 2003.

[7] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks* vol. 18, no. 5, pp.602-610, 2005.

[8] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM networks for improved phoneme classification and recognition," *Proc. ICANN-2005*, Springer LNCS 3697, pp.799-804.

[9] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on PAMI*, vol. 31, no. 5, pp.855-868, 2009.

[10] T. M. Breuel, A. Ul-Hasan, M. A. Al-Azawi and F. Shafait, "High-performance OCR for printed English and Fraktur using LSTM networks," *Proc. ICDAR-2013*, pp.683-687.

[11] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 16, pp.82-97, 2012.

[12] L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, nos. 3-4, pp.197-387, 2014.

[13] A. Graves, A. R. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," *Proc. ICASSP-2013*, pp.6645-6649.

[14] A. Graves, N. Jaitly and A. R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," *Proc. ASRU-2013*, pp.273-278.

[15] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," *Proc. ICML-2014*, pp.1764-1772.

[16] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," *Proc. INTERSPEECH-2014*, pp.338-342.

[17] H. Sak, O. Vinyals, G. Heigold, A. Senior, E. McDermott, R. Monga and M. Mao, "Sequence discriminative distributed training of long short-term memory recurrent neural networks," *Proc. INTERSPEECH-2014*, pp.1209-1213.

[18] P. Doetsch, M. Kozielski and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," *Proc. ICFHR-2014*, pp.279-284.

[19] P. Voigtlaender, P. Doetsch, S. Wiesler, R. Schluter, and H. Ney, "Sequence-discriminative training of recurrent neural networks," *Proc. ICASSP-2015*, pp.2100-2104.

[20] T. Bluche, H. Ney and C. Kermorvant, "A comparison of sequence-trained deep neural networks and recurrent neural networks optical modeling for handwriting recognition," *Proc. SLSP-2014*, pp.199-210.

[21] B. Moysset, T. Bluche, K. Maxime, M. F. Benzeghiba, R. Messina, J. Louradour and C. Kermorvant, "The A2iA multi-lingual text recognition system at the second Maurdor evaluation," *Proc. ICFHR-2014*, pp.297-302.

[22] F. Eyben, J. Bergmann and F. Weninger, *CURRENNT: CUDA-enabled Machine Learning Library For Recurrent Neural Networks*, http://sourceforge.net/projects/currennt/.

[23] H. Bourlard and N. Morgan, *Connectionist Speech Recognition: a Hybrid Approach*, Kluwer, Norwell, MA, 1993.

[24] U.-V. Marti and H. Bunke, "The IAM-database: an English sentence database for offline handwriting recognition," *IJDAR*, vol. 5, no. 1, pp.39-46, 2002.

[25] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, K. Vesely, "The Kaldi Speech Recognition Toolkit," *Proc. ASRU-2011*, code available at http://kaldi.sourceforge.net/.

[26] X. Zhang, M. Wang, L.-J. Wang, Q. Huo, and H. Li, "Building handwriting recognizers by leveraging skeletons of both offline and online samples," *Proc. ICDAR-2015*.